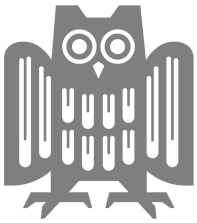
Smarter Screen Space Shading



Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften der
Fakultät für Mathematik und Informatik der
Universität des Saarlandes

Vorgelegt durch
Oliver Nalbach
im August 2017 in Saarbrücken

Betreuender Hochschullehrer – Advisor:
Prof. Dr. Hans-Peter Seidel

Gutachter – Reviewers:
Prof. Dr. Hans-Peter Seidel
Dr. Tobias Ritschel
Prof. Dr. Diego Gutiérrez

Kolloquium – Examination

Datum – Date:
2017-11-10

Dekan – Dean:
Prof. Dr. Frank-Olaf Schreyer

Vorsitzender – Chair:
Prof. Dr. Philipp Slusallek

Protokoll – Reporter:
Dr. Shida Beigpour

Abstract

This dissertation introduces a range of new methods to produce images of virtual scenes in a matter of milliseconds. Imposing as few constraints as possible on the set of scenes that can be handled, e. g., regarding geometric changes over time or lighting conditions, precludes precomputations and makes this a particularly difficult problem. We first present a general approach, called deep screen space, using which a variety of light transport aspects can be simulated within the aforementioned setting. This approach is then further extended to additionally handle scenes containing participating media like clouds. We also show how to improve the correctness of deep screen space and related algorithms by accounting for mutual visibility of points in a scene. After that, we take a completely different point of view on image generation using a learning-based approach to approximate a rendering function. We show that neural networks can hallucinate shading effects which otherwise have to be computed using costly analytic computations. Finally, we contribute a holistic framework to deal with phosphorescent materials in computer graphics, covering all aspects from acquisition of real materials, to easy editing, to image synthesis.

Kurzzusammenfassung

Diese Dissertation stellt eine Reihe neuer Methoden vor, um Bilder virtueller Szenen innerhalb von Millisekunden zu erzeugen. Die Beschränkung auf so wenige Annahmen wie möglich hinsichtlich der zu verarbeitenden Szenen, z.B. was Änderungen der Geometrie oder der Beleuchtung angeht, macht Vorberechnungen unmöglich und erschwert das Gesamtproblem. Wir beschreiben zunächst einen allgemeinen Ansatz, von uns Deep-Screen-Space genannt, mit dem viele Lichttransportaspekte innerhalb des oben genannten Rahmens simuliert werden können. Dieser Ansatz wird dann auf Szenen, die trübe Medien enthalten, erweitert. Wir zeigen außerdem, wie die Genaugikeit der Deep-Screen-Space-Methode und verwandter Algorithmen durch Berücksichtigung der gegenseitigen Sichtbarkeit von Punkten in einer Szene verbessert werden kann. Danach betrachten wir das Problem der Bildsynthese von einem komplett anderen Blickwinkel und nähern eine Renderingfunktion mithilfe von Maschinenlernen an. Wir zeigen, dass neuronale Netzwerke Beleuchtungseffekte, die sonst auf kostspielige Weise analytisch berechnet werden müssen, unmittelbar halluzinieren können. Zuletzt stellen wir eine ganzheitliche Methodik zum Umgang mit phosphoreszenten Materialien in der Computergrafik vor, die alle Aspekte, von der Ausmessung echter Materialien, über eine einfache Bearbeitung dieser, bis hin zur Bildsynthese, abdeckt.

Summary

In this dissertation, we introduce new methods to generate images of virtual, three-dimensional scenes within only a few milliseconds. One of our main goals is to impose as few constraints as possible on which kind of scenes can be handled. For example, it should be possible that scenes are arbitrarily animated or that they contain not only opaque surfaces but also transparent volumes. We first present a general approach, called deep screen space, using which a variety of light transport aspects can be simulated within the aforementioned setting. This approach is then further extended to additionally handle scenes containing participating media like clouds. We also show how to improve the correctness of deep screen space and related algorithms by accounting for mutual visibility of points in a scene. After that, we take a completely different point of view on image generation using a learning-based approach to approximate a rendering function. We show that neural networks can hallucinate shading effects which otherwise have to be computed analytically in a costly way. Finally, we contribute a holistic framework to deal with phosphorescent materials in computer graphics, covering all aspects from acquisition of real materials, to easy editing, to image synthesis. We will now summarize the aforementioned methods in detail.

Deep Screen Space

Computing shading such as ambient occlusion, sub-surface scattering or indirect light in screen space has recently received a lot of attention. While being efficient to compute, screen space methods suffer from the incompleteness of the available information about the scene. In Chapter 4 we propose a deep screen space to overcome this problem while retaining computational efficiency. In screen space methods, primitives are projected to the virtual camera sensor, culled, shaded and finally rasterized, using a z-buffer to resolve occlusions. Deep screen space works similarly using adaptive tessellation of the primitives into surfels instead of rasterization into pixels. The surfels are crafted such that they share the same projected size in screen space, then optionally shaded and stored on-GPU as an unstructured surfel cloud. Due to the tessellation scheme, geometry closer to the camera is represented by finer grained surfels, i. e., in a more detailed way, than distant primitives — as in a classic framebuffer — but it is not affected by occlusion or under-sampling. The surfel cloud can then be used to compute shading using splatting to a hierarchical framebuffer layout.

Volume Shell Splatting

In Chapter 5, we generalize deep screen space to volumes. For this, the current view of the volume is converted into a transmittance interval map, containing depth intervals in which the transmittance to the camera is reduced by the same amount. These intervals then receive indirect illumination in a modified splatting procedure which walks over the depth intervals corresponding to receiver pixels and sums up the contributions from each surfel.

The Bounced Z-Buffer

Synthesizing images of scenes with indirect illumination at interactive frame rates, e. g., using deep screen space (Chapter 4), commonly ignores indirect shadows. In Chapter 6 we extend indirect lighting algorithms that splat shading to a framebuffer to include indirect visibility. To this end we propose the bounced z-buffer: While a common z-buffered framebuffer, at each pixel, maintains the distance from the closest surface and its radiance along a direction from the camera to that pixel, our representation contains the distance from the closest surface and its radiance after one indirect bounce into a certain other direction. Consequently, with bounced z-buffering, only the splat from the nearest emitter in one direction contributes to each pixel.

Deep Shading

In computer vision, convolutional neural networks (CNNs) have recently achieved unprecedented performance for inverse problems where RGB pixel appearance is mapped to attributes such as positions, normals or reflectance. In computer graphics, screen space shading has boosted the quality of real-time rendering, converting the same kind of attributes of a virtual scene back to appearance, enabling effects like ambient occlusion, indirect light, scattering and many more. We consider the diagonal problem: synthesizing appearance from given per-pixel attributes using a CNN. The resulting Deep Shading renders screen space effects at competitive quality and speed while not being programmed by human experts but learned from example images.

Capture and Reproduction of Phosphorescence

Finally, in Chapter 8, we propose a pipeline to accurately acquire, efficiently reproduce and intuitively manipulate phosphorescent appearance. In contrast to common appearance models, a model of phosphorescence needs to account for temporal change (decay) and previous illumination (saturation). For reproduction, we propose a rate equation that can be efficiently solved in combination with other illumination in a mixed integro-differential equation system. We describe an acquisition system to measure spectral coefficients of this rate equation for actual materials. Our model is evaluated by comparison to photographs of actual phosphorescent objects. The framework is completed by an artist-friendly interface to control the behavior of phosphorescent materials by specifying spatio-temporal appearance constraints.

Zusammenfassung

In dieser Dissertation stellen wir neue Methoden vor, um Bilder virtueller, dreidimensionaler Szenen innerhalb von wenigen Millisekunden zu erzeugen. Besonderes Augenmerk liegt dabei darauf, so wenige Anforderungen wie möglich an die Art der zu verarbeitenden Szenen zu stellen. So soll es beispielsweise möglich sein, dass Szenen beliebig animiert sind oder das sie nicht nur lichtundurchlässige Oberflächen enthalten, sondern auch durchsichtige Volumen. Wir beschreiben zunächst einen allgemeinen Ansatz, von uns Deep-Screen-Space genannt, mit dem viele Lichttransportaspekte innerhalb des oben genannten Rahmens simuliert werden können. Dieser Ansatz wird dann auf Szenen, die trübe Medien enthalten, erweitert. Wir zeigen außerdem, wie die Genaugikeit der Deep-Screen-Space-Methode und verwandter Algorithmen durch Berücksichtigung der gegenseitigen Sichtbarkeit von Punkten in einer Szene verbessert werden kann. Danach betrachten wir das Problem der Bildsynthese von einem komplett anderen Blickwinkel und nähern eine Renderingfunktion mithilfe von Maschinenlernen an. Wir zeigen, dass neuronale Netzwerke Beleuchtungseffekte, die sonst auf kostspielige Weise analytisch berechnet werden müssen, unmittelbar halluzinieren können. Zuletzt stellen wir eine ganzheitliche Methodik zum Umgang mit phosphoreszenten Materialien in der Computergrafik vor, die alle Aspekte, von der Ausmessung echter Materialien, über eine einfache Bearbeitung dieser, bis hin zur Bildsynthese, abdeckt. Wir werden die soeben genannten Methoden nun im Detail zusammenfassen.

Deep-Screen-Space

In jüngster Zeit hat die Berechnung von Beleuchtungseffekten im Bildraum — wie Umgebungsverdeckung, Lichtstreuung in Gegenständen oder indirekter Beleuchtung — viel Aufmerksamkeit erregt. Während sie effizient berechenbar sind, leiden Methoden, die im Bildraum arbeiten, an der Lückenhaftigkeit der über die Szene verfügbaren Informationen. In Kapitel 4 schlagen wir einen tiefen Bildraum (Deep-Screen-Space) vor, der dieses Problem löst und dabei die Effizienz der Berechnungen erhält. Bildraummethoden projizieren grafische Primitive auf den virtuellen Kamerasensor, verwerfen nicht sichtbare Primitive und beleuchten und rasterisieren die verbleibenden, wobei ein Tiefenpuffer zur Klärung von gegenseitigen Verdeckungen verwendet wird. Die Deep-Screen-Space-Methode funktioniert ähnlich, nutzt aber adaptive Tessellierung der Primitive in kleine Scheiben (Surfels) statt der Rasterisierung zu Pixeln. Die Surfel werden dabei derart erzeugt, dass sie eine einheitliche Größe im Bildraum haben, dann beleuchtet und schließlich als ungeordnete Surfel-Wolke im Grafikkartenspeicher abgelegt. Dank dieses Tessellierungsschemas wird kameranahe Geometrie durch feinkörnige Surfel repräsentiert, das heißt detaillierter, als weit entfernte Primitive — genau wie in einem herkömmlichen Framebuffer — wird aber nicht von Verdeckungen oder Unterabtastung negativ beeinflusst. Die Surfel-Wolke kann dann mittels Splatting in einen hierarchisch gegliederten Framebuffer für Beleuchtungsberechnungen verwendet werden.

Volume-Shell-Splatting

In Kapitel 5 verallgemeinern wir den Deep-Screen-Space-Ansatz auf Volumen. Dafür wird, für jeden Pixel der aktuellen Ansicht auf das Volumen, eine Reihe von Tiefenintervallen bestimmt, die jeweils der gleichen Verminderung der optischen Durchlässigkeit entsprechen. Die einzelnen Intervalle werden dann mittels eines modifizierten Splatting-Ansatzes indirekt beleuchtet, der über die Tiefenintervalle iteriert und die Beiträge der einzelnen Surfels aufsummiert.

Der Bounced-Z-Buffer

Bei der Synthese von Bildern animierter Szenen mit indirekter Beleuchtung in Echtzeit, zum Beispiel mit Deep-Screen-Space (Kapitel 4), werden indirekte Schatten in der Regel vernachlässigt. In Kapitel 6 zeigen wir, wie diese Schatten im Deep-Screen-Space und auch bei weiteren Algorithmen, die auf Splatting in einen Framebuffer basieren, korrekt gehandhabt werden können. Zu diesem Zweck schlagen wir den Bounced-Z-Buffer vor: Während ein herkömmlicher Z-Puffer für jeden Pixel die Entfernung zum nächsten Objekt und die von dort in Richtung der Kamera ausgehende Strahldichte speichert, verwaltet unsere Datenstruktur die Entfernung zur nächsten Oberfläche und die von dort eintreffende Strahldichte nach Reflexion in eine bestimmte andere Richtung. Daraus resultierend, trägt bei Verwendung des Bounced-Z-Buffers nur der Splat des nächsten Emitters in einer bestimmten Richtung zum Wert eines Pixels bei.

Deep Shading

Im Bereich des maschinellen Sehens haben sogenannte Convolutional Neural Networks (CNNs) in jüngster Vergangenheit noch nie da gewesene Erfolge bei inversen Problemen, bei denen Farbwerte von Pixeln auf Attribute wie Positionen, Normalen oder Reflexionseigenschaften abgebildet werden müssen, erzielt. In der Computergrafik wiederum haben Bildraumverfahren die visuelle Qualität in der Echtzeit-Bildsynthese vorangebracht, indem sie, in umgekehrter Weise, die soeben genannten Attribute auf Farbwerte abbilden und so Effekte wie Umgebungsverdeckung, indirekte Beleuchtung oder Lichtstreuung ermöglichen. Wir betrachten das dazu diagonale Problem, die Synthese von Beleuchtungseffekten aus im Bildraum definierten Attributen mithilfe eines CNNs. Mit diesem, von uns Deep-Shading getauften Verfahren, können Beleuchtungseffekte im Bildraum in konkurrenzfähiger Qualität und Berechnungszeit errechnet werden und das ohne manuelle Programmierung durch menschliche Experten, sondern allein durch Maschinenlernen anhand von Beispielbildern.

Messung und Reproduktion von Phosphoreszenz

Zu guter Letzt stellen wir in Kapitel 8 Methoden vor, um phosphoreszente Materialien akkurat auszumessen, effizient Bilder dieser zu berechnen und ihre Bearbeitung ungeübten Benutzern zugänglich zu machen. Im Gegensatz zu gängigen Materialmodellen muss ein Modell, das Phosphoreszenz unterstützt, zeitliche Veränderungen (das Abklingen des Nachleuchtens) und die zurückliegende Beleuchtung (Sättigung des Materials) berücksichtigen. Unser Modell basiert auf Ratengleichungen, die effizient gelöst werden können und sich mit anderen Lichttransportaspekten zu einem Integro-Differentialgleichungssystem kombinieren lassen. Wir beschreiben ein Messsystem, mit dem die spektralen Koeffizienten dieser Ratengleichungen für echte Materialien bestimmt werden können. Das verwendete Modell wird durch einen Vergleich mit Fotografien echter phosphoreszenter Objekte evaluiert. Unsere Methodik zum Umgang mit phosphoreszenten Materialien in der Bildsynthese wird durch eine benutzerfreundliche Bedienoberfläche komplettiert, mit der sich neue Materialien, durch einfaches Festlegen des Erscheinungsbildes zu verschiedenen Zeitpunkten und unter verschiedener Beleuchtung, definieren lassen.

Contents

1 Introduction

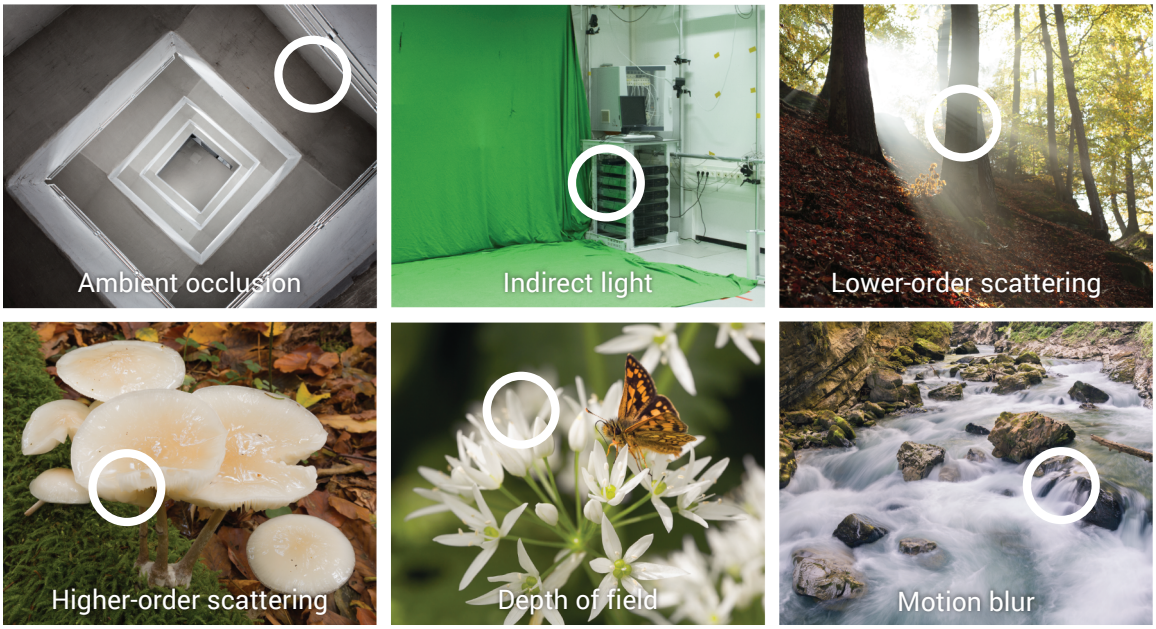


Figure 1.1: Real-world examples for the different shading effects we seek to reproduce computationally. The indirect light example is by Avishek Chatterjee.

This dissertation presents a range of new methods to synthesize images of virtual scenes. We focus on generalizable methods supporting completely dynamic scenes while still allowing for compute times of only a few milliseconds. In this first chapter we motivate our specific goals, give an overview over our contributions and outline the thesis.

1.1 Motivation

The benefits of computer graphics are ubiquitous today: Rendered images, i. e., images synthesized by an algorithm as opposed to, e. g., taken by an analog camera, can be found in applications ranging from entertainment to educational purposes. Increasing visual quality is being expected not only from feature films and still images but also from computer games or

augmented and virtual reality applications. But while the former are pre-computed the latter have to generate renderings on-demand, which implies the need for rendering methods that are both *interactive* and fully *dynamic*. By interactivity, we mean the ability of a method to compute a new rendered image without disruptive lags, e. g., when a user of a virtual reality application wearing a head-mounted display changes her view by moving her head. By dynamism, we refer to the capability of a method to do so, no matter how the scene being rendered is changing over time, for example due to interactions of the user with it in a computer game. These changes include varying lighting conditions as well as objects being moved, deformed or fractured. Both aforementioned requirements are inherently irreconcilable: The key to interactivity is pre-computation but to allow pre-computation assumptions about what will be rendered have to be made which in turn limits dynamism.

Figure 1.2: Five conflicting goals that we face in real-time rendering.

Interactivity and dynamism are not the only desirable features of a rendering approach: The images produced should at the same time be of high *quality*, i. e., be unbiased while exhibiting small variance both spatially and temporally. A more aesthetic but nevertheless desirable trait is *generality*. Instead of crafting highly-specialized methods for single aspects of light transport (Figure 1.1) — we commonly refer to them as "shading effects" in the following — which requires expertise and makes the resulting algorithms cumbersome to use and combine, the ultimate goal is to tackle many aspects of shading computation at once using a versatile framework. Finally, we strive for *completeness*. The interaction of light with solid surfaces might be the most common case, yet the range of visual phenomena does not stop there but also covers participating media and extends to specialized but still interesting effects such as phosphorescence.

It is needless to say that there is no method so far which accomplishes all of the goals listed above. However, in this dissertation we present new algorithms for image synthesis that try to achieve an even balance of as many desired features as possible.

1.2 Contributions

In the following, we will underline the precise contributions of the five publication this thesis is based on [Nalbach et al. 2014a; Nalbach et al. 2014b; Nalbach et al. 2015; Nalbach et al. 2017b; Nalbach et al. 2017a] to the respective sub-areas of computer graphics.

Interactive Global Illumination

So called *screen space approaches* for computing different shading effects such as ambient occlusion (Section 2.5.3), sub-surface scattering (Section 2.6.3) or indirect light (Section 2.5.1) have recently received a lot of attention. Their distinguishing characteristic is that they base their computations on information about the surface points directly visible in every image pixel like their position in the virtual world or the corresponding surface normal. Due to their high efficiency, these approaches have become widely adopted, for example in the game industry [Aalund 2013] but they suffer from the key limitation of insufficient information about parts of the rendered scene which are not or only barely visible. In this thesis, we propose a *deep screen*

space (Chapter 4) to overcome this limitation while retaining computational efficiency [Nalbach et al. 2014a; Nalbach et al. 2014b; Nalbach et al. 2015], in particular we contribute:

- An approximate scene representation — the "deep screen space" — in form of a cloud of small disks, which is designed so that all disks would have the same size if projected to screen space, and which is computed on-the-fly and stored on the GPU by means of fast hardware tessellation.
- An approach to compute various shading effects from this representation (termed *shell splatting* by us) using splatting to a multi-resolution interleaved framebuffer, that adapts the precision of the shading computation to the actual range of the light transport.
- An extension of shell splatting from light transport between surface to light transport between surfaces and heterogeneous participating media (Chapter 5).
- A method to correctly account for indirect visibility in shell splatting and other splatting-based approaches using a *bounced z-buffer* (Chapter 6).

Screen Space Shading by Example

Traditionally, screen space algorithms are hand-crafted and tweaked to address one shading effect at a time, requiring not only expertise in light transport theory but also experience and profound knowledge about the hardware used to perform the calculations. This clashes with the generality and completeness of the resulting algorithms. In this work, we show how deep convolutional neural networks (CNNs) that have recently achieved new levels of performance for the inverse problem of mapping per-pixel appearance to attributes, can be leveraged for rendering itself by learning from examples. We call the resulting approach *deep shading* (Chapter 7) [Nalbach et al. 2017a]. More precisely, we present:

- A general CNN architecture that can be instantiated and trained to approximate arbitrary shading effects in screen space using training data which can be produced by approximate or unbiased reference methods.
- An instance of our architecture which can simultaneously reproduce multiple shading effects, e. g., ambient occlusion and shallow depth-of-field, in a single network.
- A proof-of-concept that screen space shading methods can not only be learned from rendered images but also from real world image appearance.

Rendering of Advanced Physical Effects

Most rendering applications confine themselves to the simulation of instantaneous effects. However, especially in the dynamic settings we are interested in, temporally varying appearance can become a useful device to enrich virtual worlds, making them more immersive. One example for such a visual quality entailing a change over time is phosphorescence, the re-emission of light with a substantial time delay after preceding excitation of a material by photons. Typical applications of phosphorescence include emergency signs, door handles, light switches, user panels or automotive parts but also toys, artwork and fashion, with the practical importance of the effect varying from entertainment to saving lives. In this thesis, we introduce the first comprehensive framework addressing the phenomenon (Chapter 8) [Nalbach et al. 2017b]. It consists of:

- A phenomenological model of phosphorescence based on rate equations covering all necessary aspects while ignoring those not relevant for practically simulating it.
- An acquisition system to measure spectral coefficients to this model for actual materials.
- An efficient way to reproduce phosphorescence enabled by the design of our model.
- An artist-friendly interface to control the behavior of phosphorescent materials by specifying spatio-temporal appearance constraints.

1.3 Structure of the Thesis

After this introduction, in Chapter 2, we unfold the theoretical background regarding light transport that is relevant to all of our contributions. By a subsequent review of previous work (Chapter 3) we discuss existing solutions to the various theoretical problems. In the next three chapters (Chapter 4–6), we present the basic deep screen space pipeline, its extension to participating media and our method to account for indirect visibility in splatting-based approaches, respectively. We move on to deep shading in Chapter 7 and eventually to our phosphorescence framework in Chapter 8. Finally, we conclude the thesis and discuss future directions of research in Chapter 9.

2

Theoretical Background

In this chapter, we formally define the problem we are trying to solve in the remainder of the thesis, namely light transport. We start with the radiometric quantities we seek to compute in rendering (Section 2.1). Next, we describe elements of local light behavior including light sources (Section 2.2) and models for scattering at surfaces (Section 2.3) and in participating media (Section 2.4). Finally, we arrive at the modeling of global light transport. In many cases, a simplified model of light transport restricted to scenes containing only solid objects inside a vacuum is sufficient to achieve convincing results when using it in a simulation. Furthermore, we will only consider either completely opaque or optically dense solids in this thesis, which will allow for additional simplifications. This case, mathematically backed by the rendering equation [Kajiya 1986], is introduced in Section 2.5. When liquids or phase mixtures come into play, a more extensive formulation based on the radiative transfer equation [Chandrasekhar 1950] is necessary. This situation is described in Section 2.6. For both settings, we will also discuss simplifications commonly performed to make them amenable to real-time rendering.

2.1 Important Quantities

A pixel on a (pinhole) camera's sensor receives light from a specific incident direction and in rendering our goal is to compute that quantity. Unfortunately, defining the amount of light arriving at a point $\mathbf{x}$ from a given direction $\vec{\boldsymbol{\omega}}$ is problematic as $\mathbf{x}$ has no area and its probability to be hit by photons is consequently zero. But considering a small surface patch $\mathrm{d}A^{\perp}$ at $\mathbf{x}$ which is perpendicular to and centered around $\vec{\boldsymbol{\omega}}$ and a small cone $\mathrm{d}\omega$ of incident directions around $\vec{\boldsymbol{\omega}}$ we can look at the limit when these become infinitesimally small:

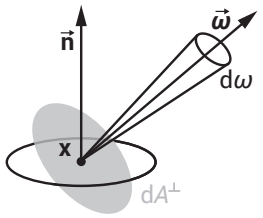


Figure 2.1: Radiance as differential flux per solid angle and area.

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega\ \mathrm{d}A^{\perp}}\ [\mathrm{W\,sr^{-1}\,m^{-2}}].$$

Where Φ [W] is radiant flux, i. e., energy per time and L is called the *radiance* (Figure 2.1). In this thesis, we define $L(\mathbf{x}, \vec{\boldsymbol{\omega}})$ as the radiance arriving at point $\mathbf{x}$ for a direction $\vec{\boldsymbol{\omega}}$.

Notation example	Typical use
a, b, c	Scalar
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Vector describing a point
$\vec{\boldsymbol{\omega}}$	Normalized vector describing a direction
λ	Wavelength
Symbol	**Meaning**
$\mathbf{x}, \mathbf{x}'$	Points on surfaces
$\vec{\boldsymbol{\omega}}_{\text{in/out}}$	Direction from which light is arriving at / in which light is leaving a point
$\vec{\mathbf{n}}$	Surface normal at a particular point
Ω^+	Upper unit hemisphere centered around the surface normal at a particular point
Ω	Full unit sphere
$\mathbb{R}^n$	Vector of real numbers
$\widehat{\mathbb{R}}^n$	Normalized vector of real numbers
$\mathbb{R}_{\geq 0}$	Set of non-negative real numbers
L	Function measuring radiance
E	Function measuring irradiance
I	Function measuring intensity
Φ	Function measuring radiant flux
$\mathrm{d}\Phi$	Infinitesimal flux
$\mathrm{d}\omega$	Infinitesimally small cone of directions
$\mathrm{d}A^{\perp}$	Infinitesimally small surface patch that is defined perpendicular to a given vector
L_{e}	Radiance emitted at a point
L_{dir}	Reflected radiance from light sources
L_{ind}	Reflected radiance from non-emissive surfaces
$f_{\text{r}}(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}})$	BRDF at $\mathbf{x}$
σ_{a}	Absorption coefficient
σ_{s}	Scattering coefficient
σ_{t}	Attenuation or extinction coefficient, sum of absorption and scattering coefficients
$\rho(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}})$	Phase function at $\mathbf{x}$
$\tau(\mathbf{x}, \mathbf{x}')$	Transmittance between $\mathbf{x}$ and $\mathbf{x}'$

Table 2.1: Notation conventions and symbols as used by us throughout this thesis.

Another important quantity is the *irradiance* $E(\mathbf{x})$ $[\mathrm{W\,m^{-2}}]$, which is the integral of incoming radiance over all directions from which a point $\mathbf{x}$ can be reached. For a point $\mathbf{x}$ on a surface, those are the directions in the upper hemisphere centered around the surface normal $\vec{\mathbf{n}}$ at $\mathbf{x}$:

$$E(\mathbf{x}) = \int_{\Omega_+} L(\mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{in}}) \langle \vec{\mathbf{n}}, \vec{\boldsymbol{\omega}}_{\text{in}} \rangle \, \mathrm{d}\vec{\boldsymbol{\omega}}_{\text{in}}.$$

The term $\langle \vec{\mathbf{n}}, \vec{\boldsymbol{\omega}}_{\text{in}} \rangle$ accounts for the fact that radiance is defined with respect to a surface patch perpendicular to the direction of light as seen in Figure 2.1. The perpendicularity only holds for one of all possible incoming directions. For all other directions their radiance contribution has to be attenuated according to the increasing area over which light is spread when the incident angle becomes flatter.

Complementary to irradiance is the notion of *radiosity* R (also called *radiant exitance*) whose definition is the same as for irradiance with the difference that it integrates the radiance leaving rather than arriving at a point.

Intensity I is mainly useful when talking about light sources (Section 2.2) and defined as the

infinitesimal measure of power per steradian:

$$I = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega}\ [\mathrm{W\,sr^{-1}}].$$

All of the aforementioned radiometric quantities — radiance, irradiance, radiosity and intensity — have spectral counterparts that consider only light of a specific wavelength λ, e. g., spectral radiance $L(\mathbf{x}, \vec{\boldsymbol{\omega}}, \lambda)$. They correspond to partial derivatives of the original quantities with respect to wavelength and only become relevant when wavelength changes of photons are considered. This is only necessary for special effects such as phosphorescence (Chapter 8). Thus we will mostly ignore them for now.

2.2 Light Sources

In order for light to reach a virtual camera sensor there need to be primary sources of photons. While there are accurate ways to model real-world lamps [Goesele et al. 2003], simple models are often sufficient to achieve plausible results and allow for analytic computation of relevant quantities (Section 2.1) as well as for efficient methods to solve important sub-problems like visibility computation (Section 3.6.1). As the contributions of this thesis are largely agnostic with respect to the way light sources are modeled, we will only present the types of light source used throughout this work.

Given a point $\mathbf{x}$, we model the emissive behavior at $\mathbf{x}$ by a function

$$L_\mathrm{e} : \mathbb{R}^3 \times \widehat{\mathbb{R}}^3 \to \mathbb{R}_{\geq 0} \quad [\mathrm{W\,sr^{-1}\,m^{-2}}],$$

where $L_\mathrm{e}(\mathbf{x}, \vec{\boldsymbol{\omega}})$ is the radiance emitted at point $\mathbf{x}$ into direction $\vec{\boldsymbol{\omega}}$. If $L_\mathrm{e}(\mathbf{x}, \vec{\boldsymbol{\omega}}) > 0$ for any direction $\vec{\boldsymbol{\omega}}$, $\mathbf{x}$ is (located on) a *light source*.

Point Lights

A point light has no spatial extent and isotropic emission so that it can be defined by specifying its position $\mathbf{x}$ and either intensity I or power Φ (which are related by $\Phi = 4\pi\, I$). Given a second point $\mathbf{y}$, $\mathbf{y}$ can only receive light from direction

$$\vec{\boldsymbol{\omega}}_\mathrm{in} = \frac{\mathbf{x} - \mathbf{y}}{\|\mathbf{x} - \mathbf{y}\|_2}$$

due to the source at $\mathbf{x}$. Furthermore, if $\mathbf{x}$ and $\mathbf{y}$ are mutually visible then

$$L(\mathbf{y}, \vec{\boldsymbol{\omega}}_\mathrm{in}) = L_\mathrm{e}(\mathbf{x}, -\vec{\boldsymbol{\omega}}_\mathrm{in}) = \frac{I}{\|\mathbf{x} - \mathbf{y}\|_2^2}.$$

Directional Lights

Directional light sources are a non-physical approximation for lights which are so distant that their rays of light arriving in a scene can be considered to be parallel. For a particular direction $\vec{\boldsymbol{\omega}}_\mathrm{in}$, the direction in which the virtual light source is thought to be located, the radiance from the source at any point $\mathbf{x}$ is defined to be equal to a constant, $L(\mathbf{x}, \vec{\boldsymbol{\omega}}_\mathrm{in}) = c$, unless there is any occluding object in said direction. In principle, the amount of energy emitted by a directional light source is unbounded but as the virtual scenes we want to render are not, light cannot accumulate arbitrarily.

2.3 Material Modeling

2.3.1 The Bi-directional Reflectance Distribution Function

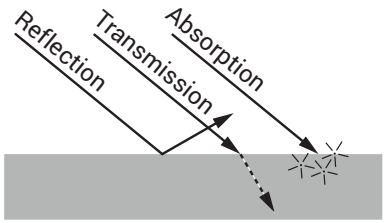


Figure 2.2: Light hitting a surface may be reflected, transmitted or absorbed.

Light interacts with different materials in different ways. Photons hitting a surface may be reflected, enter the respective object (be transmitted) or be absorbed and converted into other forms of energy (Figure 2.2). In case of opaque surfaces, it is sufficient to model this local behavior by a *bi-directional reflectance distribution function* (BRDF) [Nicodemus et al. 1977] describing which ratio of the incoming radiance from which incoming direction is reflected into which outgoing direction, ignoring transmission. The BRDF is a function

$$f_r : \widehat{\mathbb{R}}^3 \times \mathbb{R}^3 \times \widehat{\mathbb{R}}^3 \to \mathbb{R}_{\geq 0}$$

where $f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out})$ is the ratio of radiance reflected into direction $\vec{\omega}_{out}$ to the incident differential irradiance from direction $\vec{\omega}_{in}$ at the spatial location $\mathbf{x}$.

Law of Conservation of Energy

Following the law of conservation of energy, the reflection by a surface should not increase the total amount of energy, which can be stated as

$$\int_{\Omega^+} f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) \langle \vec{\mathbf{n}}, \vec{\omega}_{out} \rangle \, \mathrm{d}\vec{\omega}_{out} \leq 1 \quad \text{for all } \vec{\omega}_{in}. \tag{2.1}$$

Helmholtz Reciprocity Principle

A second important principle is that of reciprocity [von Helmholtz 1866] which, applied to BRDFs, requires that swapping in- and outgoing directions does not affect the value of the BRDF:

$$f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) = f_r(\vec{\omega}_{out}, \mathbf{x}, \vec{\omega}_{in}) \quad \text{for all } \vec{\omega}_{in}, \vec{\omega}_{out}, \mathbf{x}. \tag{2.2}$$

An important corollary of Equation 2.2 is that light transport simulation does not have to use light sources as starting points but is equally valid considering the reverse direction, starting from the virtual camera sensor (Section 3.3).

2.3.2 Material Characterization

The range of materials that can be characterized by a BRDF can be roughly split into three cases (Figure 2.3). First, perfectly *diffuse* (or *Lambertian*) materials reflect light equally into all directions, i. e.,

$$f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) = k_d \overset{(2.1)}{\leq} \frac{1}{\pi} \quad \text{for all } \vec{\omega}_{in}, \vec{\omega}_{out}. \tag{2.3}$$

Second, a *perfect mirror* reflects all light into the reflected incoming direction:

$$f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) = \frac{\delta(\vec{\omega}_{out} + \vec{\omega}_{in} - 2\langle \vec{\omega}_{in}, \vec{\mathbf{n}} \rangle \vec{\mathbf{n}})}{\langle \vec{\omega}_{in}, \vec{\mathbf{n}} \rangle}, \tag{2.4}$$

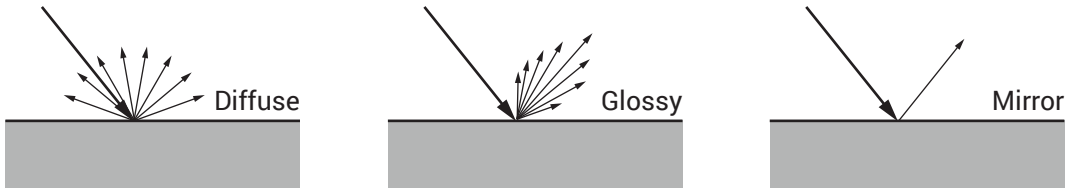


Figure 2.3: We distinguish between perfectly diffuse, moderately glossy and mirroring surfaces.

where δ is the Dirac delta. Third, all other materials are called *glossy* or *specular* where the level of glossiness can however vary drastically.

2.4 Volume Modeling

2.4.1 The Phase Function

While the local scattering behavior of surfaces is described by the BRDF, the local scattering behavior in participating media is described by a *phase function*

$$\rho : \widehat{\mathbb{R}}^3 \times \mathbb{R}^3 \times \widehat{\mathbb{R}}^3 \to \mathbb{R}_{\geq 0}$$

which can be seen as a probability density function (PDF) where $\rho(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}})$ relates to the probability of radiance arriving at $\mathbf{x}$ from direction $\vec{\omega}_{\text{in}}$ being scattered towards $\vec{\omega}_{\text{out}}$.

Law of Conservation of Energy

Like BRDFs, phase functions need to be energy conserving. There are however three important differences affecting the formulation of this requirement: First, light may be scattered into all spherical directions, so the integral is over the full sphere instead of just an upper hemisphere. Second, while the BRDF implicitly models absorption, the phase function only models scattering and has to integrate exactly to one. Third, the cosine term is dropped: Light is thought to be interacting with small orientationless particles. The constraint then is:

$$\int_{\Omega} \rho(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}}) \, \mathrm{d}\vec{\omega}_{\text{out}} = 1 \quad \text{for all } \vec{\omega}_{\text{in}}. \tag{2.5}$$

Helmholtz Reciprocity Principle

The Helmholtz reciprocity principle also has to hold for phase functions:

$$\rho(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}}) = \rho(\vec{\omega}_{\text{out}}, \mathbf{x}, \vec{\omega}_{\text{in}}) \quad \text{for all } \vec{\omega}_{\text{in}}, \vec{\omega}_{\text{out}}, \mathbf{x}. \tag{2.6}$$

2.4.2 Important Phase Functions

We will now define two important phase functions that we will use in this thesis (Chapter 5).

The Isotropic Phase Function

The simplest possible case is the *isotropic phase function* which is the volumetric equivalent of the diffuse BRDF and constant independent of the pair of directions:

$$\rho_{\text{diff}}(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}}) \overset{(2.5)}{=} \frac{1}{4\pi} \quad \text{for all } \vec{\boldsymbol{\omega}}_{\text{in}}, \vec{\boldsymbol{\omega}}_{\text{out}}. \tag{2.7}$$

The Henyey-Greenstein Phase Function

A popular non-isotropic phase function is the one due to Henyey and Greenstein [1941] which only depends on the angle between $-\vec{\boldsymbol{\omega}}_{\text{in}}$ and $\vec{\boldsymbol{\omega}}_{\text{out}}$:

$$\rho_{\text{HG}}(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}}) = \frac{1-g^2}{4\pi\,(1+g^2-2g\,\langle -\vec{\boldsymbol{\omega}}_{\text{in}}, \vec{\boldsymbol{\omega}}_{\text{out}}\rangle)^{1.5}} \quad \text{for all } \vec{\boldsymbol{\omega}}_{\text{in}}, \vec{\boldsymbol{\omega}}_{\text{out}}. \tag{2.8}$$

The parameter $g \in [-1, 1]$ is called the anisotropy coefficient and controls whether photons are scattered in directions similar to their previous direction or rather into backwards directions.

2.4.3 Absorption and Scattering Coefficients

As mentioned in Section 2.4.1, the phase function only models the likelihood of different scattering directions in case a scattering event occurs. The rate at which these events themselves occur while light is traveling through a participating medium is controlled by its scattering coefficient σ_{s} [m^{-1}]. Light may also be absorbed i. e., converted into a different form of energy. The rate at which absorption occurs is steered by the medium's absorption coefficient σ_{a} [m^{-1}]. Both absorption and scattering coefficients can vary throughout the medium. If this is the case, the medium is called *heterogeneous*, otherwise it is called *homogeneous*. The sum of absorption and scattering coefficient is called *attenuation* or *extinction coefficient* and denoted σ_{t}.

2.5 Light Transport in a Vacuum

Having defined the local behavior of light, how it emerges and how it interacts with a surface, we can model its transport through a virtual scene, which is governed by a long sequence of such local events.

2.5.1 The General Case

The first case that we consider is that of light transport between surfaces inside a vacuum which can be modeled by the rendering equation due to Kajiya [1986]:

$$L(\mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}}) = \underbrace{L_{\text{e}}(\mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}})}_{\text{Light emitted by the surface}} + \underbrace{\int_{\Omega_+} f_{\text{r}}(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}, \vec{\boldsymbol{\omega}}_{\text{out}}) L(\mathbf{x}, -\vec{\boldsymbol{\omega}}_{\text{in}}) \langle \vec{\mathbf{n}}, \vec{\boldsymbol{\omega}}_{\text{in}} \rangle \, \mathrm{d}\vec{\boldsymbol{\omega}}_{\text{in}}}_{\text{Light reflected by the surface}} \tag{2.9}$$

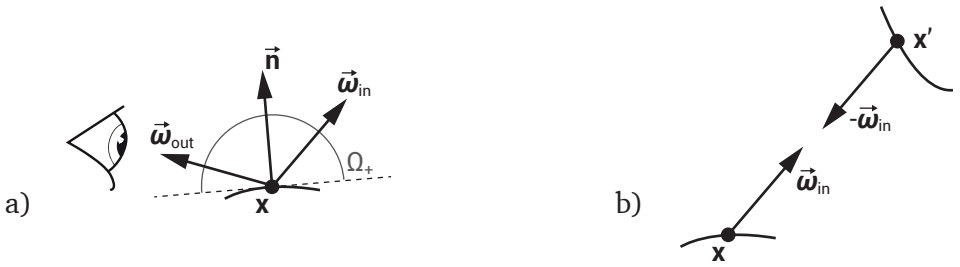


Figure 2.4: An example for the geometric configuration addressed by Equation 2.9.

Equation 2.9 computes the radiance $L(\mathbf{x}, \vec{\omega}_{out})$ as the sum of two components: first, $L_e(\mathbf{x}, \vec{\omega}_{out})$, the light directly emitted by the surface along the considered ray (Section 2.2), and second the light reflected at $\mathbf{x}$ into direction $\vec{\omega}_{out}$. The latter quantity is obtained by integrating the reflected radiance over all possible incident directions $\vec{\omega}_{in}$ at $\mathbf{x}$ which, in this simplified case of opaque materials, corresponds to the upper unit hemisphere centered around the surface normal $\vec{\mathbf{n}}$ at $\mathbf{x}$ as light from all other directions is blocked by the surface itself (cf. Figure 2.4 a). To determine the radiance reflected for one incident direction $\vec{\omega}_{in}$, we need to know how much radiance is arriving from this particular direction and the ratio of it which is reflected into direction $\vec{\omega}_{out}$. Incident radiance is recursively defined and has to be weighted by a cosine factor due to its definition with respect to a perpendicular surface patch (Section 2.1). The reflected ratio, finally, is modeled by the BRDF (Section 2.3).

To allow a precise definition of some important terms used in the remainder of this thesis, we will perform one recursive substitution in Equation 2.9. First, assume that $\mathbf{x}'$ is the first point visible in direction $\vec{\omega}_{in}$ seen from point $\mathbf{x}$ as illustrated in Figure 2.4 b. Then, under the assumption of empty space between $\mathbf{x}$ and $\mathbf{x}'$ nothing can interfere with light traveling between these two points and

$$L(\mathbf{x}, -\vec{\omega}_{in}) = L(\mathbf{x}', -\vec{\omega}_{in}). \tag{2.10}$$

Using Equation 2.10, we can expand Equation 2.9 to

$$\begin{aligned} L(\mathbf{x}, \vec{\omega}_{out}) =& L_e(\mathbf{x}, \vec{\omega}_{out}) \\ &+ \int_{\Omega_+} f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) L_e(\mathbf{x}', -\vec{\omega}_{in}) \langle \vec{\mathbf{n}}, \vec{\omega}_{in} \rangle \, \mathrm{d}\vec{\omega}_{in} \\ &+ \int_{\Omega_+} f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) \left(\int_{\Omega_+} f_r(\vec{\omega}_{in'}, \mathbf{x}', -\vec{\omega}_{in}) L(\mathbf{x}', -\vec{\omega}_{in'}) \langle \vec{\mathbf{n}}', \vec{\omega}_{in'} \rangle \, \mathrm{d}\vec{\omega}_{in'} \right) \langle \vec{\mathbf{n}}, \vec{\omega}_{in} \rangle \, \mathrm{d}\vec{\omega}_{in} \\ =:& L_e(\mathbf{x}, \vec{\omega}_{out}) + L_{dir}(\mathbf{x}, \vec{\omega}_{out}) + L_{ind}(\mathbf{x}, \vec{\omega}_{out}) \end{aligned} \tag{2.11}$$

where we also applied linearity of integration to split the outer integral.

In Equation 2.11, we call the term on the second line the *direct light* (L_{dir}) and the term on the third line the *indirect light* (L_{ind}) at point $\mathbf{x}$. In this thesis and the computer graphics literature the indirect light is also sometimes loosely referred to as *global illumination* (GI). If we substitute L in the definition of the indirect light by the non-recursive L_e we obtain *single bounce* indirect light. Otherwise, we say that we are dealing with the *multi-bounce* case. Figure 2.5 shows how a scene evolves visually with an increased number of simulated bounces.

Figure 2.5: From left to right: the two important cases of direct light only and single bounce indirect light and different degrees of multi-bounce indirect lighting. For many scenes the most striking change occurs when switching from direct to single bounce and more than three bounces barely have an additional effect on visual appearance.

Depending on the nature of f_r in the outer integral of L_{ind} we further distinguish between indirect light with *diffuse or glossy receivers*. Finally, we also differentiate between *diffuse and glossy senders*, this notion refers to the nature of the BRDF term in the inner integral of L_{ind}. A lot of methods we will present in Chapter 3 do only consider diffuse senders or receivers for the computation of indirect light, even if the actual BRDF of the respective surfaces is glossy.

The integral in Equation 2.9 is over normalized directions. Alternatively, it is also possible to integrate over all surface points $\mathbf{x}'$ in the scene. This formulation is closer to some of the actual light transport simulation algorithms we will present in Chapter 3 and the following chapters and also allows to introduce a few more important terms:

$$L(\mathbf{x}, \vec{\omega}_{out}) = L_e(\mathbf{x}, \vec{\omega}_{out}) + \int_{\mathbf{x}' \in S} f_r(\vec{\omega}_{in}, \mathbf{x}, \vec{\omega}_{out}) V(\mathbf{x}, \mathbf{x}') L(\mathbf{x}', -\vec{\omega}_{in}) \frac{\langle \vec{\mathbf{n}}, \vec{\omega}_{in} \rangle \langle \vec{\mathbf{n}}', -\vec{\omega}_{in} \rangle}{\|\mathbf{x} - \mathbf{x}'\|_2^2} \, \mathrm{d}A_{\mathbf{x}'}, \tag{2.12}$$

where S is the set of all points on surfaces in the scene to be considered and the function $V(\mathbf{x}, \mathbf{x}')$ computes the so-called *visibility term*:

$$V(\mathbf{x}, \mathbf{x}') = \begin{cases} 1, & \text{if } \mathbf{x} \text{ and } \mathbf{x}' \text{ are mutually visible} \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

If a rendering algorithm does not try to approximate the visibility function in the inner integral of the indirect light term (Equation 2.11), we say that it ignores *indirect visibility*. The conversion between Equation 2.12 and Equation 2.9 can be found in the original paper [Kajiya 1986].

Many real-time approaches for solving Equation 2.12, in particular screen space methods (Section 3.6.3), limit the points $\mathbf{x}'$ in the scene that are considered to only those which are within a certain *effect range*. Mathematically, the visibility $V(\mathbf{x}, \mathbf{x}')$ is then treated as 0 whenever the distance between $\mathbf{x}$ and $\mathbf{x}'$ is larger than the chosen range.

In the remainder of this section, we will list some common approximations to Equation 2.12 that are frequently used in real-time rendering instead of the full global illumination.

Ambient occlusion | Environment lighting | AO * Env. Light. | Directional occlusion

Figure 2.6: From left to right: Ambient occlusion only approximates visibility (first). Combining it with unshadowed environment lighting (second) already yields convincing results (third). But only directional occlusion (last) correctly treats incoming light and visibility in a joint manner.

2.5.2 Scene Lighting by Environment Maps

An easy way to create the impression of varied illumination but in the bounds of a very reduced direct lighting model is to use a lighting environment, or *environment map*, L_{env} as the source of emission:

$$L_{\text{dir}}(\mathbf{x}, \vec{\omega}_{\text{out}}) \approx \int_{\Omega_+} f_{\text{r}}(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}}) L_{\text{env}}(\vec{\omega}_{\text{in}}) \langle \vec{\mathbf{n}}, \vec{\omega}_{\text{in}} \rangle \, \mathrm{d}\vec{\omega}_{\text{in}}. \tag{2.14}$$

The environment map is a function defining an incident radiance for each direction $\vec{\omega}_{\text{in}}$ of the unit sphere. In particular, the function is not parameterized by a position in space. Formally, the term above may also be seen as an approximation to the indirect light $L_{\text{ind}}(\mathbf{x}, \vec{\omega}_{\text{out}})$ and then be combined with other direct light approximations.

Despite its crudeness, environment lighting — also called image-based lighting (IBL) — typically works well, especially for scenes consisting only of a single mostly convex object as can be seen from Figure 2.6. Equation 2.14 is easy to evaluate, for example using ray tracing (Section 3.3). In Chapter 7, we will show how environment map based lighting can be learned and approximated by a deep CNN.

2.5.3 Ambient Occlusion

The main shortcoming of environment lighting is that it completely ignores the visibility function. To alleviate this, *ambient occlusion* (AO) [Zhukov et al. 1998], a variant of accessibility shading [Miller 1994] may be introduced (Figure 2.6). Formally, two modifications are performed: First, an (approximate) visibility function $\widetilde{V}$ is re-introduced into indirect environment lighting. $\widetilde{V}(\mathbf{x}, \vec{\omega}_{\text{in}})$ is defined as 1 if there is some point visible in direction $\vec{\omega}_{\text{in}}$ starting from $\mathbf{x}$ within a certain distance range and 0 otherwise. Second, the integral over all incident directions is split into a product of two integrals, one over the visibility and geometric terms (called the ambient occlusion) and one over the unoccluded incident lighting:

$$L_{\text{ind}}(\mathbf{x}, \vec{\omega}_{\text{out}}) \approx \underbrace{\int_{\Omega_+} \widetilde{V}(\mathbf{x}, \vec{\omega}_{\text{in}}) \langle \vec{\mathbf{n}}, \vec{\omega}_{\text{in}} \rangle \, \mathrm{d}\vec{\omega}_{\text{in}}}_{\text{Ambient occlusion}} \int_{\Omega_+} f_{\text{r}}(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}}) L_{\text{env}}(\vec{\omega}_{\text{in}}) \, \mathrm{d}\vec{\omega}_{\text{in}}. \tag{2.15}$$

Splitting the integral is not backed mathematically and, as it decouples near-range visibility from direction, only yields an acceptable approximation if the second integral varies slowly with $\vec{\omega}_{\text{out}}$.

The latter typically implies that dependence of f_r on directions has to be dropped, i. e., a diffuse BRDF has to be used. In the most extreme case, the environment map is set to be a constant function, turning the second integral into a single constant *ambient lighting* term.

AO is one of the shading effects most commonly addressed in real-time rendering. The long list of possible algorithms include ray tracing (Section 3.3) or screen space methods (Section 3.6.3). It is also one of the effects we tackle with our deep screen space pipeline (Chapter 4). Furthermore, in Chapter 7, we train a deep neural network to compute AO.

2.5.4 Directional Occlusion

Another indirect light approximation is *directional occlusion* (DO) which, like ambient occlusion, uses approximate visibility but does not separate the latter from the incident light:

$$L_{\text{ind}}(\mathbf{x}, \vec{\omega}_{\text{out}}) \approx \underbrace{\int_{\Omega_+} \widetilde{V}(\mathbf{x}, \vec{\omega}_{\text{in}}) L_{\text{env}}(\vec{\omega}_{\text{in}}) \langle \vec{\mathbf{n}}, \vec{\omega}_{\text{in}} \rangle \, \mathrm{d}\vec{\omega}_{\text{in}}}_{\text{Directional occlusion}} \int_{\Omega_+} f_r(\vec{\omega}_{\text{in}}, \mathbf{x}, \vec{\omega}_{\text{out}}) \, \mathrm{d}\vec{\omega}_{\text{in}}. \tag{2.16}$$

Again f_r is assumed to be diffuse in practice. Directional occlusion has gained increased interest since computational methods of similar efficiency to those for ambient occlusion have been proposed [Ritschel et al. 2009b] (Section 3.6.3). Similar to ambient occlusion, we will return to directional occlusion in chapters 4 & 7.

2.6 Light Transport in the Presence of Participating Media

In the last section, we only considered light bouncing between solid objects located in a vacuum. In a real scene, however, interactions of light and matter are not restricted to the boundaries between solid objects and free space. Scattering events may also occur when photons hit small water droplets in mid-air or light enters the inside of a translucent material such as wax.

2.6.1 The General Case

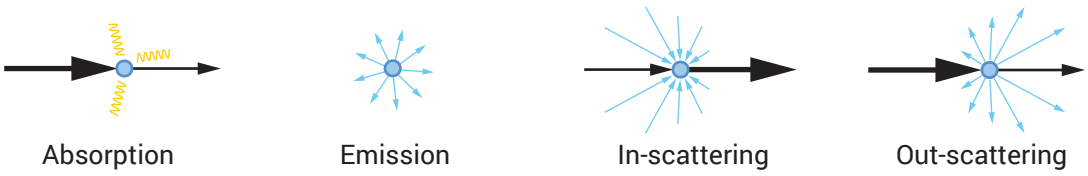


Figure 2.7: Light traveling through a participating medium may be absorbed, newly emitted, in-scattered or out-scattered. The black arrows visualize how the amount of photons traveling along a particular ray in space changes due to these different interactions.

We distinguish between four types of scattering events that may happen at any point in space (Figure 2.7). First, light can be *absorbed*, i. e., converted into a different form of energy. Second, light can also be *emitted* by the participating medium. What appears unusual at first glance actually allows to model phenomena like explosions or phosphorescent volumes (Chapter 8). Finally, light can *scatter*, changing its direction. Considering radiance traveling along a ray in

space, we call it *out-scattering* when radiance is lost due to it changing to a different direction and *in-scattering* when radiance is gained due to additional photons changing their path to coincide with the ray.

The more general behavior of light in the presence of participating media can be modeled by the equation of radiative transfer [Chandrasekhar 1950] which, in its integral form [Pharr and Humphreys 2010], reads:

$$L(\mathbf{x}, \vec{\omega}_{\text{in}}) = \underbrace{\tau(\mathbf{x} \leftarrow \mathbf{x}')\, L(\mathbf{x}', -\vec{\omega}_{\text{in}})}_{\text{Attenuated light from first surface}} + \underbrace{\int_0^t \tau(\mathbf{x} \leftarrow \mathbf{x}'')\, S(\mathbf{x}'', -\vec{\omega}_{\text{in}})\, \mathrm{d}t'}_{\text{Attenuated in-scattered or emitted light}} \tag{2.17}$$

Here, $\mathbf{x}'$ is the location of the first surface point in direction $\vec{\omega}_{\text{in}}$ from point $\mathbf{x}$, $\tau(\mathbf{x} \leftarrow \mathbf{x}')$ is a *transmittance* function, describing the amount of light surviving absorption and out-scattering between points $\mathbf{x}'$ and $\mathbf{x}$, $\mathbf{x}'' = \mathbf{x} + t'\vec{\omega}_{\text{in}}$ is a position along the ray at distance t' (as before, we assume direction vectors to be normalized), with $\mathbf{x}' = \mathbf{x} + t\vec{\omega}_{\text{in}}$ and S a *source term* accounting for in-scattered or emitted light.

Transmittance can be computed using the Beer-Lambert law:

$$\tau(\mathbf{x} \leftarrow \mathbf{x}') = \exp\left(-\int_0^t \sigma_{\text{t}}(\mathbf{x} + t'\vec{\omega}_{\mathbf{x},\mathbf{x}'})\, \mathrm{d}t'\right), \tag{2.18}$$

where $\vec{\omega}_{\mathbf{x},\mathbf{x}'}$ is the unit vector pointing from $\mathbf{x}$ to $\mathbf{x}'$ and t the distance between the two points.

Depending on the properties of the involved participating media, the light that is eventually observed by a camera or human has undergone a varying order of scattering events on its path from the light source. There are two extreme cases that are commonly dealt with in rendering. The first is that of a single scattering event as it might be common, for example, in a dust cloud. The second is that of a countless number of scattering events, such as it occurs, e. g., in human skin. For these cases, there are specialized approximations which can be solved comparatively efficiently (Chapter 3). We will formalize them in the following.

2.6.2 Single Scattering

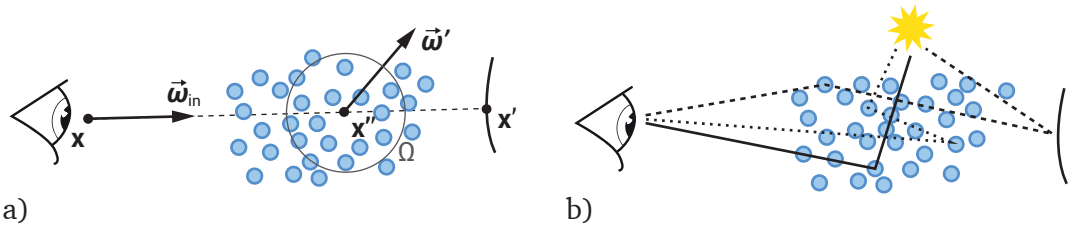


Figure 2.8: Left: The source term is computed by integrating over all spherical directions at points $\mathbf{x}''$ inside the medium. Right: Different possible paths through a medium. Direct single scattering (solid), indirect single scattering (dashed) and a multiple scattering path (dotted).

The first special case is based on the assumption of *single scattering*, i. e., only one in-scattering event along the path from a light source to the camera is allowed. Analogous to Section 2.5, depending on the number of light-surface interactions on the path from the location of the

in-scattering event to a light source we distinguish between *direct single scattering* and *indirect single scattering*. Figure 2.8 b visualizes these two cases (solid and dashed lines) and gives an example for paths ignored in single scattering (dotted line). As, in this thesis, we will not deal explicitly with rendering volumes emitting light, we ignore this case for now. The simplified source term appearing in Equation 2.17 is then:

$$S(\mathbf{x}'', -\vec{\boldsymbol{\omega}}_{\text{in}}) \approx \sigma_{\text{s}}(\mathbf{x}'') \int_{\Omega} \rho(\vec{\boldsymbol{\omega}}', \mathbf{x}'', -\vec{\boldsymbol{\omega}}_{\text{in}})\, L(\mathbf{x}'', \vec{\boldsymbol{\omega}}')\, \mathrm{d}\vec{\boldsymbol{\omega}}', \tag{2.19}$$

where we are integrating over all directions $\vec{\boldsymbol{\omega}}'$ of the unit sphere Ω. Here, σ_{s} is the scattering coefficient (Section 2.4.3), determining the probability of scattering and ρ the phase function (Section 2.4.1).

For a more extensive theoretical background, we refer to survey papers like the one by Max [Max 1995] or the book chapter by Pharr [Pharr and Humphreys 2010].

2.6.3 Multiple Scattering

The second extreme that the problem of light scattering is often reduced to is *multiple scattering*, i. e., the number of scattering events is so high that their precise count becomes unimportant for the visual appearance. In such cases, the BRDF can be extended to a bi-directional surface scattering distribution function (BSSRDF)

$$f_{\text{r}} : \widehat{\mathbb{R}}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \widehat{\mathbb{R}}^3 \to \mathbb{R}_{\geq 0}$$

where $f_{\text{r}}(\vec{\boldsymbol{\omega}}_{\text{in}}, \mathbf{x}_{\text{in}}, \mathbf{x}_{\text{out}}, \vec{\boldsymbol{\omega}}_{\text{out}})$ relates outgoing radiance leaving the surface at $\mathbf{x}_{\text{out}}$ into direction $\vec{\boldsymbol{\omega}}_{\text{out}}$ to the differential irradiance incident from direction $\vec{\boldsymbol{\omega}}_{\text{in}}$ at the spatial location $\mathbf{x}_{\text{in}}$. In fact, the BRDF is a simplified BSSRDF where $\mathbf{x}_{\text{out}} = \mathbf{x}_{\text{in}}$. An example for a BSSRDF model is the one by Jensen et al. [2001] (Section 3.3).

2.7 Extensions of the Integration Domain

In this chapter, we have considered the computation of the radiance $L(\mathbf{x}, \vec{\boldsymbol{\omega}})$ arriving at a point $\mathbf{x}$ from a direction $\vec{\boldsymbol{\omega}}$. If we want synthesized images to resemble photographs taken with a real camera as closely as possible computing a single radiance value $L(\mathbf{x}, \vec{\boldsymbol{\omega}})$ for each pixel in the final image is however not enough. A real-world imaging system collects radiance from multiple directions and over a certain time span to determine what each point in the image looks like.

Depth of Field

In an actual camera light can, in general, arrive at a certain point on the image sensor not from one but from a cone of directions, the size of which is determined by the width of the aperture. This leads to objects appearing as more or less blurry in the final image depending on their distance (Figure 2.9, left). To reproduce this effect of a limited *depth of field* (DoF) in a rendering, we consequently have to perform an integration over said cone of incident directions taking the properties of the virtual lens system into account.

Depth of field

Motion blur

Figure 2.9: Left: Due to a limited depth of field, bright unfocused points in a scene appear as large disks in the image. Right: Objects in an image appear blurred along their motion trajectory with respect to the image plane. Image credit: left by Michal Kulesza, right by Carl Johnson, both CC0.

Motion Blur

Real cameras also do not capture the light distribution within a scene at a certain point in time but accumulate photons hitting the camera sensor within a finite time span. This fact makes a difference for scenes changing over time, e. g., due to moving objects, which appear blurred along their movement trajectory (Figure 2.9, right). Rendering this effect of *motion blur* (MB) requires defining radiance with respect to a specific point in time and integration over this new temporal domain.

3

Practical Background

In this chapter, we will introduce existing methods to evaluate the equations of Chapter 2, in particular the rendering equation (Equation 2.9) and the equation of radiative transfer (Equation 2.17). A few of them are unbiased and converge to the true solutions of these equations in the limit, most however are of approximate nature. We will mainly focus on approaches running at interactive framerates as they constitute our direct competitors. Even within the scope of real-time rendering there exists a plethora of applicable algorithms. Presenting them in a meaningful order is hard since a single method as presented in a paper may draw from different computational ideas or solve several rendering problems at once (e. g., surface and volume shading). We chose to roughly group them into sections depending on the main underlying strategy. Within each section, we first present methods targeted towards light transport in the absence of participating media before moving on to volume rendering approaches. A more extensive overview of the state-of-the-art in interactive rendering can be found for example in the survey by Ritschel et al. [2012].

Before we start to review precise approaches in Section 3.3, we introduce the important notions of gathering and scattering (Section 3.1) as well as the general technique of Monte Carlo Integration which many rendering methods are based on (Section 3.2).

3.1 Gathering and Scattering

Most approaches we will review in the following can be categorized as being based either on *gathering* or on *scattering* (Figure 3.1). In a rendering context, in gathering, points in space at which shading has to be computed search for relevant other points in space, geometric primitives or proxy geometry which potentially has an influence on them. The *receivers* of an effect iterate over possible *senders*. The opposite is *scattering*: Here, points, primitives or proxies that have an effect on their surroundings handle distribution of this information — senders iterate over potential receivers. Some rendering methods also exhibit both computational schemes at different computation stages.

Which of the two options is preferable depends, not surprisingly, on the precise situation at hand. Enumerating receivers may be easier to achieve than enumerating senders and vice versa. Also hardware considerations play an important role. Current GPUs outperform the fastest CPUs

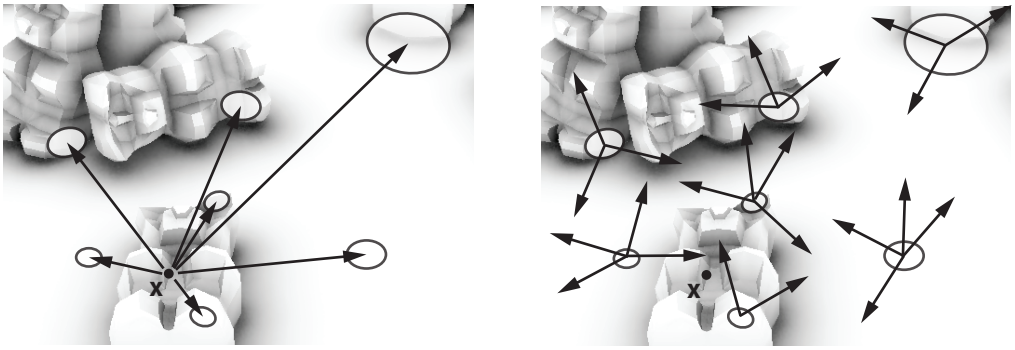


Figure 3.1: A point $\mathbf{x}$ is receiving shading from senders represented as disks. Left: In gathering, the receiving point $\mathbf{x}$ collects information from interesting senders (disks). Right: In scattering, senders distribute information to receivers potentially interested in it.

available by a factor of 50 in terms of possible floating point operations per second. But to make use of the parallel power of GPUs, rendering algorithms should consist of fine-grained parallel computations. To achieve this, scattering can often be the better choice: As an example, consider a gathering computation over several senders which is performed for each pixel in the image. The number of parallel threads is then bounded by the number of pixels. If we however perform scattering, this typically results in several threads being executed for each pixel (e. g., one per sender affecting the pixel) all across the image, we obtain a higher total number of shorter threads. Even taking into account the increased need for synchronization between different threads, the change from gathering to scattering may pay off.

3.2 Monte Carlo Integration

Monte Carlo (MC) integration is a general technique to solve integral equations such as Equation 2.9 and Equation 2.17. Given an integral of a function f over a domain D

$$\int_D f(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

we can approximate its value by generating random samples

$$\mathbf{x}_i \in D \quad \text{for } i \text{ from } 0 \text{ to } N-1 \text{ for } N \in \mathbb{N}$$

according to a probability distribution with probability density function

$$p(\mathbf{x}) > 0 \quad \text{for all } \mathbf{x} \text{ with } f(\mathbf{x}) \neq 0$$

by evaluating

$$\frac{1}{N}\sum_{i=0}^{N-1}\frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}.$$

In particular, p need not to be uniform. This technique is known as *importance sampling*.

3.3 Ray Tracing-based Approaches

Trying to apply Monte Carlo integration to the rendering equation (Equation 2.9) directly leads to the problem of determining the point $\mathbf{x}'$ which is the nearest point on a surface in direction $\vec{\boldsymbol{\omega}}_{\text{in}}$ seen from $\mathbf{x}$. *Ray casting* or *tracing* [Appel 1968] is the most general method to solve this problem. Representing the surfaces in a scene by geometric primitives like triangles or quads or implicitly using mathematical constraints such as in constructive solid geometry, ray-surface intersections can be computed analytically.

The recursive nature of the rendering equation demands for recursive Monte Carlo evaluation and hence recursive execution of the ray tracing is necessary leading to the notion of *path tracing* [Whitted 1980]. The underlying computational scheme is a prime example of gathering. Some effects like shallow depth of field or motion blur require extending the integration to additional domains such as the camera lens (as for depth of field) or time (as for motion blur) which can be done using distributed ray tracing [Cook et al. 1984]. Typically, methods based on ray tracing compute precise intersections and therefore inherently support indirect visibility.

A wide range of path tracing variants, such as bi-directional path tracing [Lafortune and Willems 1993], Metropolis light transport [Veach and Guibas 1997; Pauly et al. 2000] or gradient domain path tracing [Kettunen et al. 2015], can be found in the literature but their details are of lesser importance to this thesis as our contributions are not directly based on path tracing and, despite efforts to speed up path tracing by leveraging the computational power of GPUs [Aila and Laine 2009] they only achieve close-to-interactive framerates when using very few MC samples and hiding the resulting noise by applying sophisticated filtering [Bauszat et al. 2015], trading in the original unbiasedness of path tracing for a low variance. However, path tracing still plays an important role as a reference method in terms of rendering quality, e. g., for our deep screen space framework (Chapter 4), and as a method to compute unbiased training images in deep shading (Chapter 7).

A notable exception of a ray casting-based approach with particular relevance to our contributions is the CHC++ algorithm [Mattausch et al. 2015] which exploits coherence in ray origin and direction to efficiently cull groups of rays and primitives in a hierarchy. To this end, the distance to the nearest primitive is stored along each ray to be traced which results in a data structure that is the same as our bounced z-buffer (Chapter 6). We however use this data structure not for ray tracing but to add indirect visibility to splatting-based approaches, avoiding any sort of primitive hierarchy, and not even requiring the geometry to fit into the GPU memory at once.

Precomputed radiance transfer [Sloan et al. 2002] uses ray tracing as a pre-process to determine a transfer function at points on (or nearby) the surfaces of objects. Such a function captures to which extent light from a given direction influences shading at the sample position taking into account effects like self-occlusion. Spherical harmonics (SH) are used to cache the transfer function. At run-time, using SH to represent scene lighting then allows for fast computation of the final shading.

Methods based on path tracing can be naturally extended to participating media, e. g., as for bi-directional path tracing [Lafortune and Willems 1996]. For multiple scattering, using a BSSRDF model (Section 2.6.3) such as the one by Jensen et al. [2001], allows to render visually plausible translucent objects without simulating each scattering event explicitly.

3.4 Photon Mapping

Classic path tracing starts at the location of the virtual camera sensor and finds paths connecting it to a light source. While this connection is often easy to establish, e. g., for a point light source there is a unique ray towards the light source, there are some cases where it is not. The most notable problem is sampling paths that start at the light source which are then immediately refracted by a surface, like glass, before hitting a non-refractive surface (caustics). Finding such a path the other way around is virtually impossible as it is unclear how a point on a refractive surface will change the ray direction and whether this direction will point towards the light source before actually determining said point.

One way to resolve such situations is to use *photon mapping* [Jensen 1996]. Contrarily to common path tracing, trajectories of photons are sampled starting at the light source and information about surface hits is stored in a photon map. The photon map then, on one hand, directly yields the caustics and, on the other hand, can be used for importance sampling in a standard path tracing pass as it bears information about which parts of a scene are actually illuminated and hence interesting ray targets. While the original photon mapping algorithm was designed for light-surface interactions it was quickly adapted to volume rendering [Jensen and Christensen 1998].

The role of photon mapping in the context of this thesis is limited to being a possible high-quality reference method as the approaches presented in this thesis are not directly related to photon mapping.

3.5 Discretization-based Approaches

The methods of Section 3.3 and Section 3.4 use MC integration over an infinite continuous domain. A way to reduce the resulting computational complexity is to replace the continuous surfaces by discrete approximations. Here we have to distinguish two cases according to which the algorithms taking such an approach can be typically categorized: In the first class of methods, the discretized scene is what is immediately used for rendering the scene. In the second class of methods, the coarse representation might be used only for computing indirect effects like ambient occlusion or indirect light but not become visible in the final image. Instead, the points visible in each pixel are still determined using the original scene representation and the discretized version of the scene is merely used to shade them.

3.5.1 Methods Rendering a Discretized Scene

Examples for the first class are the methods by [Wand et al. 2001] and [Stamminger and Drettakis 2001] which turn the scenes into a point cloud before rendering it, starting from common primitives or procedurally generated geometry, respectively. Apart from points, small disks called surface elements or *surfels* [Pfister et al. 2000] or micro polygons consisting of quads [Cook et al. 1987] are alternative options.

A classic finite elements approach which works directly on a polygonal representation of a scene is the Radiosity algorithm [Goral et al. 1984]. Assuming that all surface patches are perfectly diffuse, form factors between all pairs of patches are computed and the rendering equation

(Equation 2.9) is turned into a linear system of equations where the unknowns correspond to the radiosities of the surface patches. Unlike most other rendering approaches, the solution can then be computed analytically. Due to the restriction to diffusely reflecting surfaces and comparatively high computational requirements, the classic Radiosity algorithms are rarely used anymore.

Analogous finite elements methods for rendering participating media include the extension by Rushmeier and Torrance [1987] to isotropically scattering volumes and the diffusion-based approach by Stam [1995] which solves anisotropic multiple scattering analytically on a grid.

3.5.2 Methods Using an Augmented Discretized Representation

The second class of methods where the discretization is only used in the background arguably comprises many approaches more. Again, different types of discretization are possible.

A seminal idea for the interactive computation of global illumination was the introduction of *virtual point lights* (VPLs) [Keller 1997]. Instead of gathering indirect light from randomly sampled points using path tracing, a small number (typically at most a few hundred or significantly less) of virtual point lights is distributed in the scene. This can be done, for example, using photon mapping as in the instant radiosity method by Keller [1997]. It is assumed that the radiosity of the VPLs is distributed evenly over all directions of the upper hemisphere. The radiosity itself is determined by assuming a diffuse reflection of the incident radiance. Finally, the VPLs are used in the same way that "real" point lights are used for the computation of direct light. This usually includes correct handling of indirect visibility (with respect to the point light approximation).

Light cuts [Walter et al. 2005] are a way to speed up illumination computations based on a high number of virtual light sources. Whereas the original method by Keller [1997] simply sums up contributions of all VPLs, Walter et al. [2005] first cluster all lights into a tree structure and then find relevant sub-trees, the light cuts, to base the illumination computation for each respective pixel on. This allows to use higher resolution indirect light at faster speed. As clusters are used in the computations, also shading and visibility become more approximate.

Instead of reconstructing each pixel of the image separately as in standard path tracing approaches, Lehtinen et al. [2012] re-use rays sampled on paths through different pixels across the whole image. For this, sampled rays are stored indexed by their hit points, which is again a point-based representation, and can be seen as coarse sampling of the complete indirect *light field*. This representation is then used in a sophisticated, offline reconstruction pass. The method is visibility-aware but not in a precise way.

Point-based discretizations have also found their way into the rendering of participating media. The MC-based approach to evaluate a BSSRDF to approximate multiple scattering [Jensen et al. 2001] (Section 3.3) can be sped up significantly by pre-computing the irradiance for a selection of points distributed evenly across the surface of an object [Jensen and Buhler 2002]. The BSSRDF is then evaluated with respect to nearby irradiance samples. Effectively, the irradiance samples are used analogously to the VPLs in methods derived from instant radiosity.

For rendering of heterogeneous participating media with arbitrary scattering orders, particularly including single scattering, Raab et al. [2008] show how to combine instant radiosity with a MC volume integration approach to form a more efficient offline method than path tracing alone. One shortcoming of methods based on VPLs is that the discretization to points leads to intensity singularities close to the VPLs. This problem has been overcome by means of intelligent bias

compensation [Engelhardt et al. 2012] or exchanging the point lights by ray lights [Novák et al. 2012]. For animated volumes, Weber et al. [2013] show how to compensate for changes in the rendered volume in an efficient way by progressively readjusting the VPLs.

Also surfels are popular for the computation of indirect effects. Bunnell [2005] suggests to replace a polygonal mesh by creating a surfel at each vertex the size of which is averaged from the size of the surrounding primitives. Light transport is then computed between the individual surfels using disk-to-disk form factors allowing to approximate ambient occlusion and one bounce of diffuse indirect lighting. Visibility between the surfels is ignored so that the resulting algorithm does not support indirect visibility. For meshes of moderate complexity, interactive framerates are possible.

Another approach using surfels in a similar way is due to Christensen [2008]. The main differences are the use of much finer surfels, organization of the surfels in an octree data structure, hierarchical gathering of the indirect light for each pixel similar to light cuts [Walter et al. 2005], handling of indirect visibility and support for multiple bounces and glossy receiver surfaces. The method achieves a rendering quality sufficient for use in feature films but, given rendering times of several minutes per image, has to be considered an offline method.

Micro rendering [Ritschel et al. 2009a] is a method similar to that of Christensen [2008] but running at interactive speeds. It differs by using a bounding volume hierarchy instead of an octree, importance sampling of the indirect radiance allowing better reconstruction of glossy reflections at lower sampling rates and an implementation using the GPU instead of the CPU. The drawback is a more implicit and approximate handling of indirect visibility. The main differences to our deep screen space approach are the same as for the method of Christensen [2008].

All of the methods above are limited in their support for dynamic scenes. Changes in the scene geometry clearly also induce changes in its discretized representation. Depending on how complex this representation is, these can lead to significant computational cost. For example, if a spatial acceleration structure is used, it either has to be updated or else it will not serve its purpose anymore and rendering will slow down. This is why, in deep screen space (Chapter 4), we will avoid any kind of maintenance of the discretized scene representation and instead re-generate it for each frame but in a particularly efficient way.

3.6 Rasterization-based Approaches

In a narrow sense, *rasterization* is the process of producing a raster image consisting of a rigid grid of pixels from, initially mathematically defined, geometric primitives. In a rendering context however, the term refers to a complete pipeline (Figure 3.2) in which the aforementioned primitives are first transformed to a local coordinate system, the origin of which is given by the location of the virtual camera sensor, before being projected to the image plane where they are finally rasterized. Using a special data structure, the z-buffer [Catmull 1974; Straßer 1974], which tracks the distance to the closest primitive to the camera rasterized so far intersecting each pixel, visibility can be resolved. The output of this pipeline, stored in a data structure called *framebuffer*, is then equivalent to that of tracing the primary ray through the center of each pixel in ray tracing (Section 3.3). Rasterization offers great potential for parallel execution and, in fact, the GPU is a type of hardware specifically designed to execute it. Rasterization can be leveraged in many ways as an efficient building block for various shading purposes as we will discuss in the following.

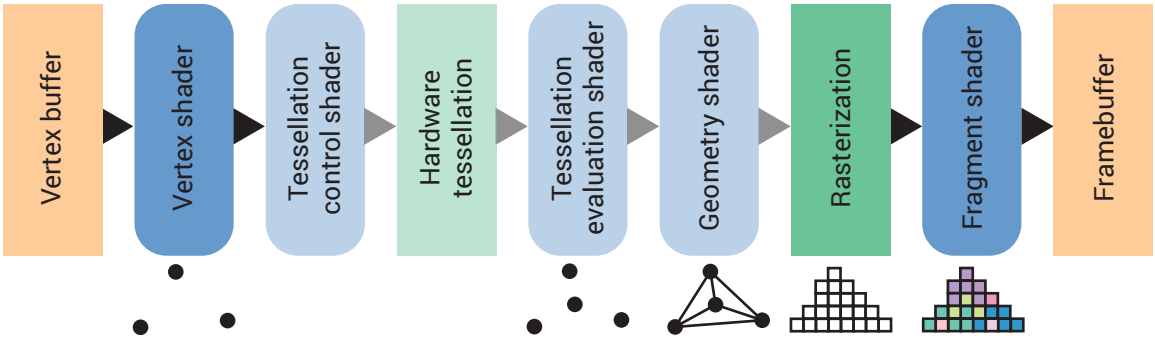


Figure 3.2: The rasterization pipeline in OpenGL 4.0. Geometry is typically stored in a vertex buffer (leftmost). First, it is processed on a per-vertex basis by a vertex stage. Optional tessellation and geometry stages may tessellate primitives into smaller ones or process whole primitives, respectively. Finally, primitives are rasterized and the resulting fragments shaded and stored in a framebuffer. The rounded boxes correspond to stages which can be programmed using special programs called shaders and lightly colored steps are optional. All methods presented in this paper have been realized by leveraging such shaders.

3.6.1 Shadow Mapping and its Applications

Maybe the first approach to cleverly exploit rasterization in an, by then, unusual way is *shadow mapping* [Williams 1978]: To handle visibility between a light source and points in a scene, the scene is rendered using rasterization as seen by the light source, storing the distance to the first primitive in each pixel. For directional lights, one view covering all relevant parts of the scene is sufficient, for point lights, six views with 90° opening angle each can cover the whole surrounding scene. Then, to compute visibility with respect to the light source at arbitrary points in the scene, they are projected to the shadow map's image plane and the depth value of the respective pixel is looked up. If the distance stored in the shadow map is significantly smaller than that of the point being shaded it is assumed that another primitive is blocking its visibility to the light source, otherwise the point is assumed to receive the light of the source. Visibility can only be assumed, not known for certain, as the shadow map only samples visibility at a discrete set of points in the scene and the comparison is with respect to the nearest neighbor for which there is a sample. The precision of this approximation directly depends on the resolution of the shadow map. While shadow mapping was designed to approximate visibility between points in a scene and point or infinitely far away light sources, many clever uses of the idea of rendering from the light source's point of view exist to achieve an abundance of other shading effects.

Dachsbacher and Stamminger [2005] render *reflective shadow maps* (RSMs) which store not only a depth value but also position and normal of the respective point as well as the value of the (diffusely) reflected flux from the light source. The RSM can then be used to approximate one bounce of indirect light by projecting points to be shaded to the RSM and sampling spatially nearby pixels which are then treated like VPLs. The approach can be seen as a variant of instant radiosity using a very efficient way of distributing photons in the scene.

To enable approximately correct handling of indirect visibility for approaches such as RSMs [Dachsbacher and Stamminger 2005], Ritschel et al. [2008] introduce *imperfect shadow maps* (ISMs). The original scene geometry is first approximated by a cloud of point primitives. Then, low-resolution shadow maps, one per VPL, are rendered from this representation where each

ISM only receives a fraction of the complete point cloud, i. e., the geometry is sub-sampled. The ISMs are then used in the same way as common shadow maps to determine visibility from each sampled VPL. By turning the ISMs into imperfect reflective shadow maps themselves, in principle, multi-bounce indirect light is also possible. To turn the scenes' triangles into points, Barák et al. [2013] propose to use the fast hardware tessellation supported by modern GPUs. The low resolution approximate nature of ISMs has the obvious drawback of being insufficient for resolving darkening due to fine geometric details. ISMs, in particular when used with efficient tessellation [Barák et al. 2013], support the same level of dynamism as RSMs.

Basic shadow mapping only handles shadows from opaque objects. *Deep shadow maps* [Lokovic and Veach 2000] are, like shadow maps, rendered from the point of view of the light source but do not store a single depth value but a coarse representation of the transmittance function (Section 2.6) which allows to look up the transmittance value at a certain distance from the light source in a certain direction. The transmittance function has to be pre-computed by costly techniques such as ray tracing to find all intersections with transparent surfaces or ray marching to determine volume transmittance. Deep shadow maps allow to handle light attenuation by both partly transparent surfaces (without refraction) and heterogeneous participating media, so that shadowing by primitives like hair, fur or smoke can be handled.

Attempting to improve the computational efficiency of deep shadow maps, *opacity shadow maps* [Kim and Neumann 2001] are of similar nature but generated more efficiently: Transmittance is sampled at the same set of depth values for all shadow map pixels, resulting in planar slices perpendicular to the light's direction for which the integrated transmittance is stored. This trades in some flexibility regarding the types of volumes which may be rendered as such a slicing is rather suited for mostly homogeneous media where transmittance changes in a similar way at all shadow map pixels.

Similar to the two preceding approaches, Mertens et al. [2004] also construct a compact representation of 1D visibility functions along rays from the light source. The difference is that the function is produced by using k-means clustering to cluster the intersections with semi-transparent surfaces found along each ray to find the depth-discretization of the transmittance function. This restricts the method to rendering individual transparent primitives and cannot be directly transferred to continuous volumes.

Deep opacity maps [Yuksel and Keyser 2008] iterate the idea by using same-depth slices similar to opacity shadow maps but with the depth not being measured from the position of the light source but from the point where the medium is entered in each pixel. Again, the use of regular intervals implicitly assumes a largely homogeneous medium such as found when rendering hair.

For the case of single scattering from direct light in homogeneous media, recently highly efficient methods emerged, e. g., by making use of prefiltering and rectification [Klehm et al. 2014].

Techniques based on shadow mapping are usually efficient enough to allow for recomputation of the shadow map if necessary and for fast look-ups. Thus, they allow for high levels of dynamism while still being situated in the real-time realm.

3.6.2 Splatting

The methods reviewed so far have been mainly based on gathering computations. *Splatting* is a general computational method exploiting the rasterization pipeline to achieve a particularly efficient implementation of scattering. For each sender, artificial geometric primitives, e. g., screen

space disks or squares, are rendered using rasterization in such a way that their projected images cover pixels corresponding to potentially relevant receiver points. The artificial primitive does not become visible itself but is only a tool to invoke shading computations at the aforementioned pixels which then compute the senders contribution.

Dachsbacher and Stamminger [2006] show that RSMs can be applied more efficiently when using scattering from each VPL implemented by splatting instead of performing gathering at each pixel. Quads are used as the splatting geometry. As the influence of each VPL falls off with increasing distance from its location, it is possible to fit the quad such that it covers all pixels in which the VPLs intensity is still above a certain threshold. By exchanging quads by ellipsoids, the case of glossy senders of indirect light can be handled, too, allowing to render caustics. Receiver surfaces are however assumed to be diffuse as glossy receivers would reveal the small number of VPLs being the source of indirect illumination. As for the original RSMs using gathering, indirect visibility is not handled.

While Dachsbacher and Stamminger [2006] use a fixed resolution two-dimensional buffer during splatting, Nichols and Wyman [2009] accelerate the approach using multiple buffers of different resolutions. Splats are by default drawn to the lowest resolution buffer but split into several parts and then drawn in one of the higher resolution buffers if necessary. This allows to cover larger effect distances than with a fixed-resolution buffer at higher speed. The same strengths and limitations of the original approach apply. Different to the paper by Dachsbacher and Stamminger [2006], glossy senders are not demonstrated.

In the approach by Sloan et al. [2007], scene geometry is approximated by spherical proxies whose influence on receiver pixels is again computed by splatting of the (increased) proxies. The approach handles AO and GI for dynamic geometry but suffers from over-occlusion and unblocked indirect lighting due to missing handling of visibility. Also, the computation has to be performed at a lower resolution than used for the direct lighting and remaining shading to achieve high frame rates.

Ambient occlusion volumes (AOVs) [McGuire 2010] use splatting from actual polygons to receiving pixels. The region of space possibly darkened due to AO from a polygon is first determined by a 3D volume. Then, the projection of this volume to the image plane is approximated by triangles and splatted. Using an analytic expression to determine the occlusion due to a polygon at a point, the method achieves high quality. Yet, it suffers from over-occlusion in areas where small geometry is stacked as occlusion from the same direction is summed up arbitrarily often. In Chapter 6, we demonstrate how this problem can be solved for AOV (and other splatting-based approaches). Another drawback of the method is that its performance depends a lot on geometric complexity and the view of the scene which makes it difficult to choose appropriate quality settings and renders the method too unpredictable for hard real-time applications.

Splatting is also useful for rendering participating media. For example, for the case of single scattering from specularly reflecting or refracting objects, pixels receiving in-scattering light can be bounded using thin line primitives which allows for real-time frame rates [Hu et al. 2010].

3.6.3 Screen Space Approaches

Technically, the term *screen space* refers to the coordinate system to which primitives are transformed in the rasterization pipeline before the actual rasterization occurs, namely one in which the x and y-coordinates are aligned with horizontal and vertical image axes respectively and the

z component relates to depth, i. e., the distance to the virtual camera. When talking about *screen space methods*, we are actually referring to algorithms which use per-pixel information about the visible point in the 3D scene associated with each pixel. This information may consist of different attributes like the position of the point in world space, its surface normal or information related to shading computations such as the surface reflectance. The framebuffer data structure storing this information over a two-dimensional grid is also called a *geometry buffer* or *g-buffer*.

The first shading effect to be solved using a screen space method was ambient occlusion. Mittring [2007] describes how information about image depth at each pixel alone was used in the video game Crysis to render corners and creases in a scene darker than their surroundings. Random points in a sphere around the point of interest are sampled, projected to the image plane and their depth is compared to the value stored in the depth buffer. If and only if the depth of the sampled point is larger than the stored value, an occlusion is assumed. Ambient occlusion is approximated by the ratio of unoccluded samples.

A more analytic approach introduced around the same time as the one by Mittring [2007] is due to Shanmugam and Arikan [2007]. Given the position $\mathbf{p}$ of a g-buffer pixel, the original surface is approximated by a sphere centered at $\mathbf{p}$ with the radius chosen such that the sphere roughly covers one pixel when projected to the image. The contribution of the pixel to a receiver is approximated by the analytic AO at the receiver due to the spherical proxy. The normal at the receiver point is taken into account to also approximate the cosine term in Equation 2.15. For the distant range, coarse spherical proxies derived from the original geometry are used as in the approach of Sloan et al. [2007].

There is a wealth of other algorithms to approximate AO using screen space information. A recent survey of such methods is due to Aalund [2013]. Notable ones are *horizon-based ambient occlusion* (HBAO) [Bavoil et al. 2008] whose optimized version HBAO+ is currently widely used in real-time applications, as well as *line-sweep ambient obscurance*, a smart evaluation scheme for HBAO.

Screen space methods are not limited to AO. As Ritschel et al. [2009b] demonstrate also DO and one bounce of indirect light can be approximated, by associating each sample with the respective direction in an environment map or using a buffer containing diffuse shading from direct light, respectively. McGuire and Mara [2014] show how using a general "screen space ray tracing" operation different shading effects can be tackled in a joint framework.

Shallow depth of field can be faked as a post process, too. Potmesil and Chakravarty [1981] describe a two-pass method in which first a sharp image is rendered using a pinhole camera model which is then blurred by distributing the sharp pixels' intensities according to the size of their depth-dependent circles of confusion. It can be considered as an early screen space approach. Subsequent methods improve the speed of the filtering technique while accepting a degeneration of the circle of confusion towards a Gaussian, e. g., using iterative filtering [Rokita 1993] or using anisotropically filtered MIP maps [Lee et al. 2009b] but follow the same general idea of applying a spatially varying blur. Contrarily, by using layered rendering some shortcomings related to occlusions can be resolved and the quality of the approximation can be improved [Lee et al. 2009a]. The general technique by Potmesil and Chakravarty [1981] can also be applied to the approximation of motion blur and is again the foundation for subsequent methods [Navarro et al. 2011].

Regarding volume rendering, computing scattering in homogeneous environments is possible using the approach of Elek et al. [2013] which works by applying a distance-dependent

point spread function crafted to create the same blurry appearance objects enclosed in fog or underwater environments would get.

Also the characteristic appearance of multiple scattering inside solid objects can be faked in screen space. Sampling the dipole model for the BSSRDF [Jensen et al. 2001] using samples taken from a geometry buffer can yield convincing results when the scattering distance is small enough [Mertens et al. 2005]. The dipole model can be replaced by a faster approximation using a sum of gaussians of similar quality [Jimenez et al. 2009] or more recent scattering profiles of comparable cost but achieving a closer match with ground truth references [Christensen 2015]. Effectively, a screen space implementation of sub-surface scattering boils down to applying an anisotropic blur which varies for each color channel or wavelength band.

In general, screen space methods have several properties which make them very popular choices for real-time applications. First, they are output-sensitive: As the computation is only based on information which is visible in the respective image, samples can only be taken from information that is relevant and little time is wasted, e. g., on sampling very distant parts of the scene which do not have a visible effect on the current view. Second, they are inherently dynamic: Whatever is changing in the scene, the screen space information directly reflects the change. Geometric changes due to techniques like bump or displacement mapping which become only visible in the rendered image and are not part of the actual primitive representation of the scene are directly supported. Third, they are level-of-detail approaches, by which we mean that count and precision of samples is proportional to their relevance in the image. For example, when computing SSAO, a detailed object close to the camera covers many pixels in screen space which allows for precise sampling while an object in the distance only covers few pixels which are however sufficient as the occlusion approximation has to be less precise. Finally, they are typically orders of magnitude faster to compute than competing unbiased or higher quality approaches as they are effectively merely sophisticated image filters. Often the computational complexity is constant per pixel which has the additional benefit of predictable timing which is also crucial in real-time applications where algorithms have to be tweaked to deliver the best possible quality given a hard time limit per frame.

While screen space methods ace the areas of interactivity, dynamism and completeness as outlined in Chapter 1, they come with fundamental drawbacks that have to be taken into account before opting for a screen space method.

The first problem is that the information about the scene to be rendered contained in a geometry buffer is incomplete: Objects which are out-of-screen or occluded from the camera's point-of-view cannot be sampled and geometry seen under grazing angles generally covers little area in screen space which often does not relate to its true area in world space. This leads to underestimation of the effects it has on its surroundings. The resulting undesired impression that shading is detached from the actual scene surfaces is sometimes called the "shower door" effect [Meier 1996]. Suggested approaches to overcome these problems include using multiple views [Vardis et al. 2013], using multiple scene layers [Shade et al. 1998; Mara et al. 2016], e. g., produced using depth peeling [Everitt 2001], but they may only tackle one of the two shortcomings at a time and are restricted to a fixed number of views or layers which will never be sufficient for all possible cases. Biasedness remains a threat. In Chapter 4–6, we will introduce a framework which tries to lift the aforementioned limitations while keeping the trademark advantages of screen space methods.

The second problem is that even for a simple effect like AO, a dozen of different highly optimized implementations exists [Aalund 2013]. This heterogenity makes it difficult to obtain a "synergistic

effect" when integrating several screen space effects in a single application. In Chapter 7, we will show how different individual screen space shaders as well as combinations of them can be learnt from examples to obtain a homogeneous set of screen space methods without programming or optimization effort.

3.7 Voxel-based Approaches

Voxel-based methods for global illumination sample the light distribution in a scene on discrete points of a lattice. For example, in the state-of-the-art method by Kaplanyan and Dachsbacher [2010], two grids are stored containing local light intensities and scene geometry, respectively which are re-generated each frame using reflective shadow maps and depth peeling from the camera. Light transport is then simulated by propagating light repeatedly to neighboring cells. This allows to compute diffuse indirect lighting with approximate visibility from the blocker grid and approximate single scattering.

[Crassin et al. 2011] initialize a lattice in the same way but then build an octree structure on top of the initial lattice. For visible pixels on the screen a final gathering step is performed using cone tracing in the octree structure. The precision of this is sufficient to achieve not only diffuse but also glossy indirect lighting.

Principal ordinate propagation [Elek et al. 2014] allows to render heterogeneous participating media at interactive frame rates by decomposing the illumination environment into a number of directional or point light sources. Light propagation is then computed over several grids (called light propagation maps [Fattal 2009]) aligned with these main directions of light propagation. Using VPLs as sources, the method also allows to render indirect illumination on volumes to some extent.

Usually, voxel-based methods can simulate light propagation over larger distance ranges than competing approaches but achieve a lower-frequency approximation as the voxel grid has to be coarse to be efficient.

4

Deep Screen Space

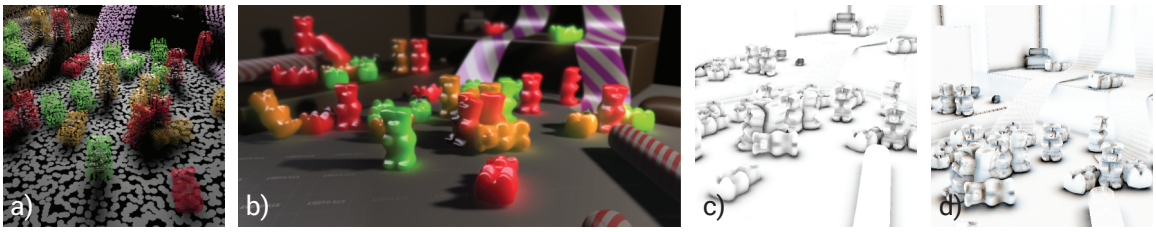


Figure 4.1: Our approach replaces the framebuffer by a collection of view-dependent surfels produced on-the-fly (a) to compute shading such as sub-surface scattering (b, 1024 × 512 px, 27 ms), ambient occlusion (c, 512 × 512 px, 22 ms) or directional occlusion (d, 512 × 512 px, 24 ms).

4.1 Introduction

In this chapter, we introduce a shading pipeline which draws inspiration from screen space approaches (Section 3.6.3) and can be used to compute a wide range of different shading effects. The method retains computational efficiency while overcoming many of the inherent limitations of screen space.

In screen space methods, primitives are projected to the virtual camera sensor, culled, shaded and finally rasterized, using a z-buffer to resolve occlusions. Deep screen space works analogously but uses adaptive tessellation of the primitives into surfels instead of rasterization into pixels. The surfels are crafted such that they share the same projected size in screen space, then optionally shaded and stored on-GPU as an unstructured surfel cloud. Due to the tessellation scheme, geometry closer to the camera is represented by finer grained surfels, i. e., in a more detailed way, than distant primitives — as in a classic framebuffer — but it is not affected by occlusion or under-sampling. The surfel cloud can then be used to compute shading.

Instead of performing the shading computation using gathering as in most screen space methods, we propose to use splatting to a multi-resolution, interleaved framebuffer. This allows surfels to scatter shading information in a detailed way to pixels corresponding to nearby points while shifting to more approximate shading for pixels that are located further away. All of this comes without the need of building a hierarchical representation over the surfel cloud which helps to

preserve fine-grained parallel computations. This makes deep screen space suitable for efficient execution on GPUs, unlike other hierarchical approaches which require maintaining a stack or per-thread state, such as when tracing rays in a bounding volume hierarchy [Aila and Laine 2009] or enumerating a light cut [Walter et al. 2005].

Our approach does not require expensive pre-computations, allowing for the same full dynamism that is also possible using screen space methods. A final important quality our approach shares with the latter is its output sensitivity and the fact that it strictly adapts the computation to the current view. All these parallels motivate us to term our approach *deep screen space*.

We start this chapter by introducing the deep screen space pipeline in Section 4.2. In Section 4.3 we show how to instantiate the general method to compute various different effects like AO, SSS or GI. We go on to analyze the performance of our method and the influence of different parameters in Section 4.4. Finally, we identify the differences of deep screen space to previous approaches (Section 4.5) before concluding the chapter with an outlook to future work (Section 4.6).

4.2 The Deep Screen Space Pipeline

We will now explain our general approach which is independent of a specific shading effect such as ambient occlusion, sub-surface scattering or computation of indirect light. In our pipeline, given the scene primitives (i. e., triangles) as input, we first compute a view-dependent surfel-based representation on-the-fly (Section 4.2.1, Figure 4.2 c). The contribution of the surfels to the current view is then splatted onto a multi-resolution deferred shading buffer (Section 4.2.2, Figure 4.2 d–f), which is finally merged to form a single image (Section 4.2.3).

4.2.1 Tessellating the Scene Into a Surfel Cloud

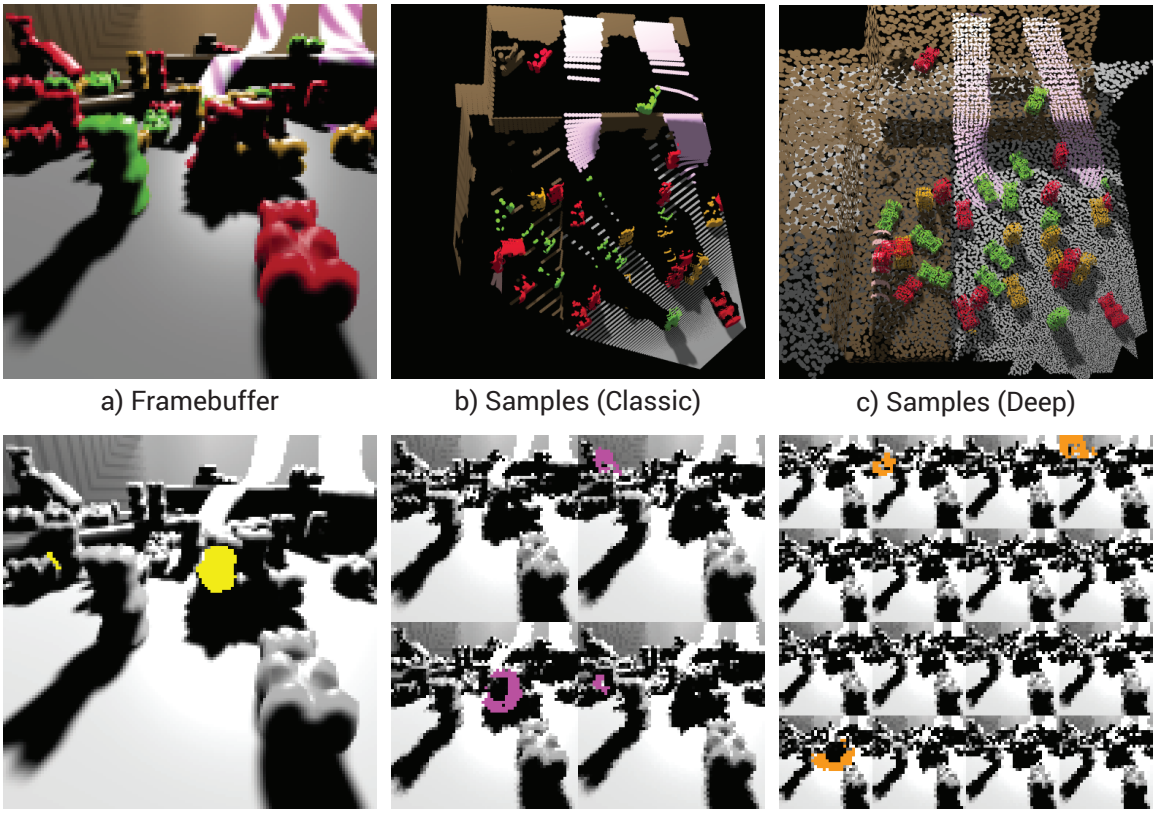
The first step comprises turning the input triangle mesh of the scene into a surfel cloud [Pfister et al. 2000]. We will refer to this process as *surfelization* in the following. In this sub-section, we will first explain the general idea and then discuss how to overcome its initial shortcomings.

Each surfel can be seen as an oriented disk that is defined by its position, normal and radius and may also be equipped with additional attributes like material information depending on the shading effect which is desired to be computed in the end. The surfels in the cloud are supposed to roughly approximate the original scene geometry but using a uniform primitive type which significantly simplifies the shading computations. This explicitly includes producing surfels also from primitives that are seen under grazing angles or which are even back-facing for the current view and which would thus not appear as pixels in a common framebuffer.

To implement this step in an efficient way, we leverage hardware tessellation [Barák et al. 2013], a feature supported by all modern GPUs: After specifying a *tessellation level*, the GPU takes care of producing subdivided primitives. We seek to achieve a tesselation into approximately equal size in screen space for all surfels, independent of their distance in world space and their orientation. For this, we first have to determine the ideal target (world space) radius r^* for the surfels stemming from one triangle, which can be computed as

$$r^* = s \cdot \tan\left(\frac{\alpha}{2}\right) \cdot (d + d_{\text{near}}),$$

where s linearly scales the surfel size relative to the size of the screen. The radius depends on



a) Framebuffer b) Samples (Classic) c) Samples (Deep)

d) Multi-res. level 0 e) Multi-res. level 1 f) Multi-res. level 2

Figure 4.2: The geometric information contained in a classic framebuffer (a) is incomplete (b). Our deep framebuffer (c), a cloud of surfels, is complete. To compute its contribution to the original framebuffer, the surfels are splatted in multiple resolutions (d–f). For pixels close to a surfel the full resolution is used (d). For more distant pixels, a hierarchy of random sub-samples of the framebuffer (e, f) is employed. With increasing sub-sampling, same-sized splats can cover increasingly large image regions without much overdraw.

the vertical opening angle α of the virtual camera and the distance d of the triangle's center to the camera's near clipping plane which has a distance of d_{near}. The formula ensures that surfels do not become infinitely small and that surfel size has its minimum at the near clipping plane while increasing in both directions from there when moving along the camera axis (Figure 4.3).

We use OpenGL for our implementation. Tessellation in OpenGL consists of a control stage, the tessellation itself and a subsequent evaluation stage (Figure 4.3). First, the *tessellation control shader* (TCS) computes the tessellation level to use which determines into how many output primitives an input primitive is then tessellated by the fixed-functionality tessellation stage. After that, the *tessellation evaluation shader* (TES) computes the attributes of the (tesellated) output primitives. We use tessellation in *point mode* to output individual point primitives instead of smaller triangles, which would be the common behavior.

Tessellation in OpenGL produces

$$v(i) = \begin{cases} 3/4\, i^2 + 3/2\, i + 1 & \text{if } i \text{ is even} \\ 3\lceil i/2 \rceil^2 & \text{if } i \text{ is odd} \end{cases} \tag{4.1}$$

point primitives for a triangle if all tessellation levels are set to i in the TCS. Using the formula

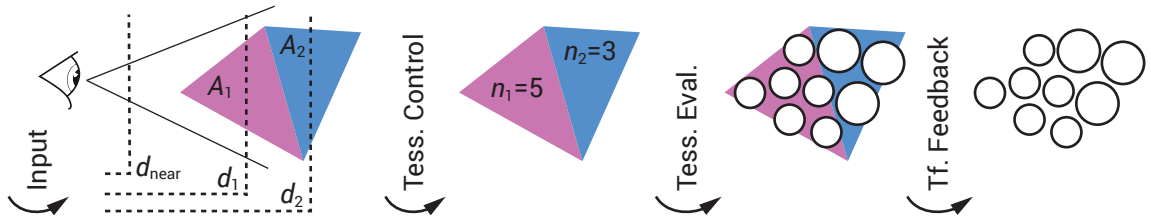


Figure 4.3: Our surfel tessellation pipeline for two triangles. The distance of the triangles is measured according to their centers of mass (first step). A TCS computes the tessellation levels to achieve the desired surfel sizes (second). The TES computes the attributes of the final surfels (third) which can be stored using the transform feedback mechanism (last).

for the even case in Equation 4.1, we approximate the necessary tessellation level for a triangle with area A by solving for i in

$$v(i) \cdot \pi r^{*2} = A$$

only considering possible cases (i. e., where $A \geq 3\pi r^{*2}$ and $i > 0$) and taking the ceil which yields

$$i^* = \left\lceil \frac{\sqrt{12\pi A - 3\pi^2 r^{*2}}}{3\pi r^*} - 1 \right\rceil .$$

In the TES, we compute the attributes of the resulting surfels by averaging positions, normals etc. using the barycentric coordinate which is made available by the tessellator unit. Also the surfels' shared radius has to be re-computed taking $v(i^*)$ into account, as the target radius r^* cannot be matched perfectly in general. If necessary for the targeted shading effect, additional attributes, e. g., irradiance or reflectance, are computed as well. The final surfels can be processed further by the shading pipeline or be cached using the so-called *transform feedback* mechanism in OpenGL.

Limitations and Corresponding Remedies Using hardware tessellation to generate surfel clouds has a few shortcomings. First, surfels at edges will protrude from the shape of the original triangle by r^*. In particular we get overlapping surfels from adjacent triangles. To avoid this, we simply move the three vertices of the original triangle based on which the new vertex positions are computed towards the triangle's center of mass by r^* in the TCS.

Second, the tessellation creates regular patterns of surfels which might become visible in one way or the other when computing the surfels' shading contribution. We therefore jitter each surfel on the triangle plane by a maximum distance of r^*.

Third, the TES will create at least three surfels for each original triangle which limits the minimal surfel size possible. Without countermeasures, the two negative consequences of this are reduced effect distance due to the too-small surfels (Figure 4.4) and possibly higher computational cost due to the increased number of surfels. The other extreme are triangles which are too large to be tessellated into fine-enough surfels because of the limited maximal tessellation level.

This third problem can be circumvented by performing the surfelization in two passes. We first compute an initial surfel cloud with fine surfels, smaller than the target size (Figure 4.5, first two steps). This surfel cloud is cached using transform feedback. Instead of simply placing the surfels at the barycentric coordinates determined by the hardware tessellation pattern, we can achieve a pseudo-random placement as follows: We compute the world space position corresponding to the coordinates as before but then hash this position into an offset in a lookup texture containing

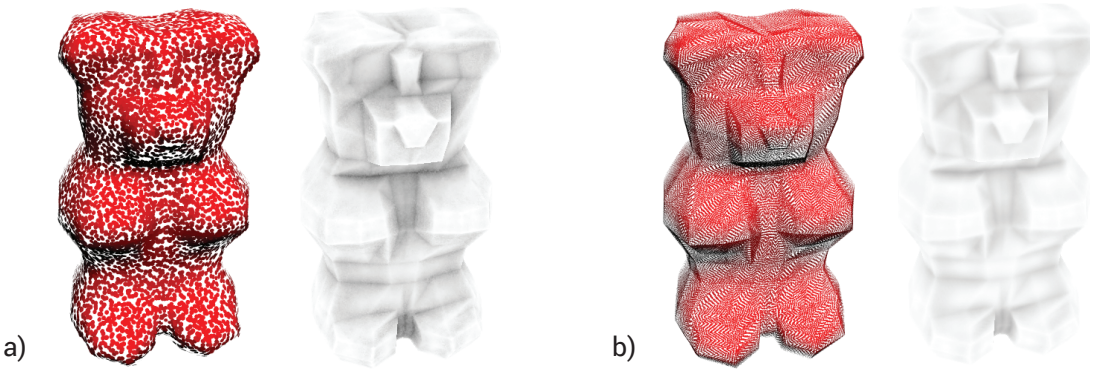


Figure 4.4: The problem with too many triangles per pixel. The desired surfel size (a, left) is not achieved by our tessellation method if the initial mesh already is too finely tessellated (b, left). As a consequence, the effect distance will be less despite the exact same settings being used (a/b,right). While this poses an inconsistency, the result still looks similar, just lighter in this case.

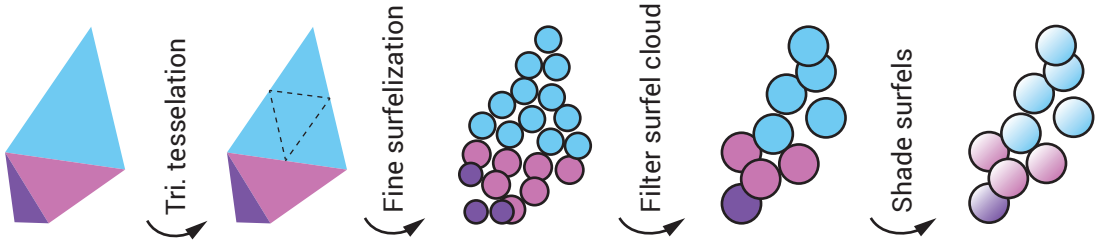


Figure 4.5: We handle extremely large or small triangles by pre-tessellation (first step) and post-filtering (third step) passes, respectively.

pre-computed pseudo-random numbers. Using the respective random number, the surfel is placed uniformly at random on the initial triangle. This results in more even placement of surfels, also for thin triangles.

In a second pass, we compare for each surfel P its area A_P resulting from the first tessellation pass with the actually desired ("ideal") area $A_P^* = \pi r^{*2}$. We discard the surfel with probability

$$\max\left(1 - \frac{A_P}{A_P^*}, 0\right)$$

by either emitting it from the geometry shader or not (Figure 4.5, third step). The surfels which are not discarded are then assigned their ideal radius r^*. Shading of the surfels, if necessary, can also be integrated into this step (Figure 4.5, last).

This second filtering pass is only conceptually separate from the splatting. In fact, it can be integrated in the splatting pipeline to save a superfluous pass over the surfels. While our two passes add computational cost during tessellation, they pay off in speed by a faster splatting stage due to the lower number of surfels and in quality of the constructed surfel cloud (Figure 4.6).

4.2.2 Splatting the Contribution of the Surfel Cloud

We will now describe how to compute the particular shading contribution from the deep screen space to a common geometry buffer. Unlike most screen space methods which gather from

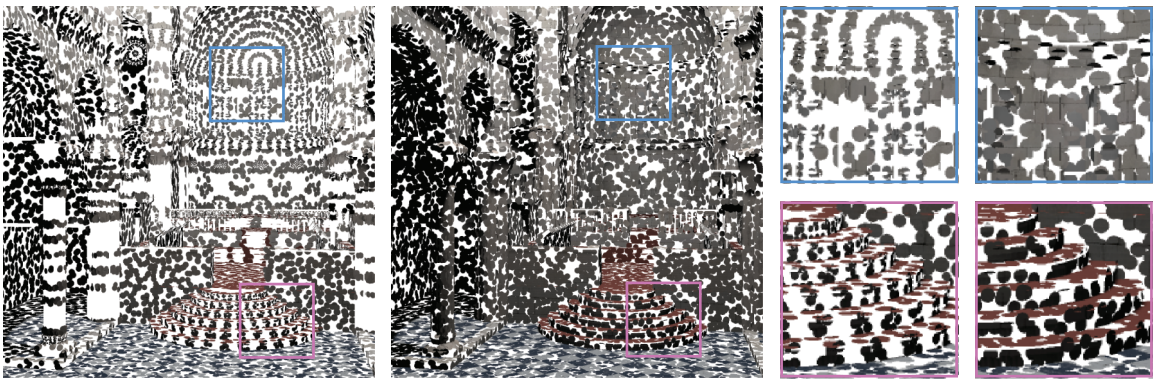


Figure 4.6: Due to limitations of hardware tessellation, surfelization in one pass has problems with small or thin triangles leading to uneven surfel distribution (left, 120k. surfels, 4.1 ms). Two passes are more expensive but also more robust (middle, 109k. surfels, 7.4 ms).

nearby pixels we use splatting to scatter the shading contribution from each surfel to multiple pixels. While this might appear to be a “step back” as gathering, in particular in regular stencils, is the preferred computational pattern in shaders, no obvious way exists to enumerate surfels nearby a particular pixel’s position in an unordered cloud without a costly hierarchy.

Hierarchical Interleaved Framebuffer Layout

To compute the shading, we employ a special framebuffer layout based on interleaved sampling [Segovia et al. 2006] but using multiple image levels with multiple interleaved patterns at the same time. While the system of Segovia et al. [2006] is not competitive or required on current GPUs anymore when gathering, it proves to be very useful for splatting, in particular in combination with our extension to multiple image levels.

Instead of using a single 2D texture as target as most algorithms based on splatting do, we splat into an array of several textures. For this, we use an OpenGL array texture with $l_{\max}$ layers where each layer corresponds to a different level of shading precision. On level $l \geq 0$ we partition the pixels of the original framebuffer into $2^l \times 2^l$ smaller “sub-buffers". For this, we take patches of $2^l \times 2^l$ pixels and randomly assign one pixel to each sub-buffer, taking the same relative position in the sub-buffer that the neighborhood had in the original image. Figure 4.2 d–f show a possible layout for the first three levels.

The hierarchy might appear similar to MIP maps, but there is a striking difference: The total number of pixels on each image level is identical and does not shrink by a factor of four. Pixels are merely distributed among more and more sub-images as the level number increases.

We use small quad primitives for splatting. If we draw a quad of size $n \times n$ on image level l the OpenGL fragment shader is invoked, i. e., computation threads are started, for a random subset of (at least) n^2 pixels in a $2^l n \times 2^l n$ neighborhood of the original framebuffer. This allows us to sub-sample shading effects, balancing precision and effect distance in different ways at the same computation costs, depending on the level we choose for splatting. By retaining the whole set of pixels on all levels, every detail of the scene still has the chance to receive shading from each surfel, only the probability decreases with increasing level.

Shell Splatting

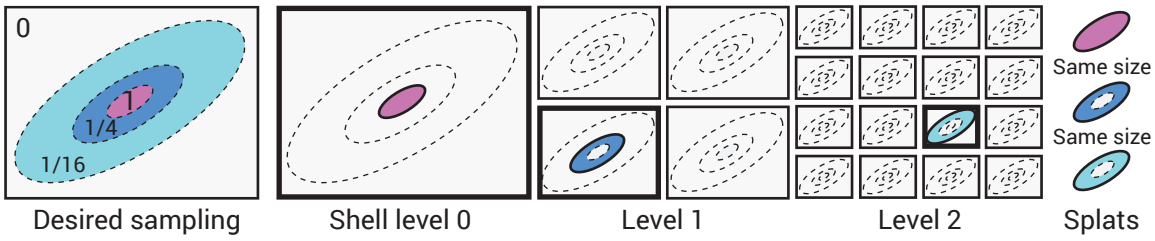


Figure 4.7: To splat with radially decreasing density, we splat multiple shells — each to a random sub-buffer of a set of buffer which have been sub-sampled with decreasing density. The innermost shell (purple) affects all pixels within a certain range while the shell on level 1 (dark blue), which has the same size in screen space, may cover pixels twice as far away but can only have an effect on a subset of them, namely the ones which have been assigned to the chosen sub-buffer. While each splat only covers a small subset of all pixels, using a sufficient number of levels, the whole screen can be covered by the sum of all splats.

One main goal of our approach is to sample the shading contributions of the surfels to the framebuffer finely for nearby and coarsely for far-away geometry. To this end, we compute the shading contribution to disjunct, increasingly large shells around a surfel's center (in world space) on increasing levels of our hierarchical framebuffer. For each surfel, a single splat is drawn per framebuffer level into a randomly selected sub-buffer to compute the contribution to the respective shell. We call this approach *shell splatting*. A schematic of this is shown in Figure 4.7 while Figure 4.2 d–f shows an actual rendering of the first three framebuffer levels including several shells which have been splatted.

On the implementation side the splatting is realized by invoking a geometry shader $l_{\max}$ times for each surfel. This geometry shader emits a point primitive at the position of the surfel's center relative to the chosen sub-buffer. Note that although we consider shells in world space we can bound them using small quad-shaped point primitives. The sub-buffer to use is chosen in a round-robin manner based on the index of the surfel. As there is no automatic clipping between individual sub-buffers, we later need to check for every fragment whether it really belongs to its given sub-buffer.

To compensate for the fact that, on level l, each pixel is only affected by $1/(2^l \cdot 2^l)$ of all surfels on average, the radii of the surfels have to be scaled by a factor of 2^l (so that their area is multiplied by $2^l \cdot 2^l$). In other words, each enlarged surfel is used as approximation for $2^l \cdot 2^l$ original surfels.

The size of the splats is determined by a function `getMaxDist` based on the surfel's properties and a precision threshold ϵ. Semantically, this function returns the world-space distance from the surfel's center at which the effect of the surfel becomes smaller than ϵ. This is similar to the bounds used in light cuts [Walter et al. 2005], classic radiosity oracles or the proxy sphere scaling in [Sloan et al. 2007]. The screen space size for the point primitive is then computed based on this distance. The definition of `getMaxDist` is effect-dependent and may, for example, take the irradiance at the surfel into account. We will discuss the effect-specific implementations of `getMaxDist` in Section 4.3.

For our approach to work it is necessary that `getMaxDist` is linear in the surfel's radius. This is however naturally true for all shading effects that we demonstrate. A side effect of this is that the size of the individual splats in screen space is the same across all precision levels l because

with each level, the surfel's radius, and therefore also the radius where its effect is larger than ϵ, increases by a factor of 2^l while the extent of the sub-buffers shrinks inversely (Figure 4.7, right column). In particular, this means that the maximal distance of the effect increases by a factor of two with each additional precision level.

Finally, the actual evaluation of the surfels' contributions happens in the fragment shader threads invoked by the splatted primitives. The fragment shader is passed information about the respective surfel responsible for the splat as well as the inner and outer radius of the associated shell. First, the world space position and normal associated with the fragment's position for the respective image level have to be looked up. To accelerate this lookup, we pre-compute shuffled (hiearchical) versions of the geometry buffer. This avoids indirection of texture reads and makes them spatially coherent [Segovia et al. 2006]. If the world space position is not inside the shell given by the surfel center and the inner and outer radius, we discard the fragment. Otherwise, we can compute the effect of the surfel by a function `computeEffect` that takes the surfel's attributes and the receiving pixel's position and normal as input. We will detail the implementation of this function for different effects in Section 4.3. To sum up the contributions of all surfels scattering to a certain pixel, we use the blending functionality of OpenGL which ensures synchronization of the write accesses to the GPU memory.

Optimizations To improve the performance of shell splatting, culling can be used: We can use view frustum culling against the surfels' shells of influence before actually drawing a splat. An even higher level of output-sensitivity is achieved by culling against a min / max MIP map over the deferred position texture. This data structure can be imagined as a screen space bounding volume hierarchy. During splatting, the smallest MIP level l_{MIP} where all pixels covered by the splat in the original image have been reduced to the same pixel p_{MIP} is computed. The values at p_{MIP} of level l_{MIP} in the min and max MIP maps define a local axis-aligned bounding box of all potential receiver points in the framebuffer. If an outer shell does not intersect this box, the splat would be without effect and can be culled in advance. Construction of the min / max MIP map takes additional time but usually pays off.

4.2.3 Reconstructing the Final Image

After splatting, we have an array texture where different image levels are still partitioned into sub-buffer grids. Unshuffling them (Figure 4.8, top row) will typically leave us with noisy individual levels because of the randomized sub-sampling of the surfel cloud. We blur each image level with a separated blur in x- and y-direction [Bavoil et al. 2008] using geometry-based weighting [Sloan et al. 2007] (Figure 4.8, bottom row). Summing up contributions from the different levels produces the final shading effect.

4.3 Applications

Next, we will discuss some of the shading effects enabled by our approach, i. e., implementations of the `getMaxDist` and `computeEffect` functions, and compare them to reference solutions (Figure 4.9).

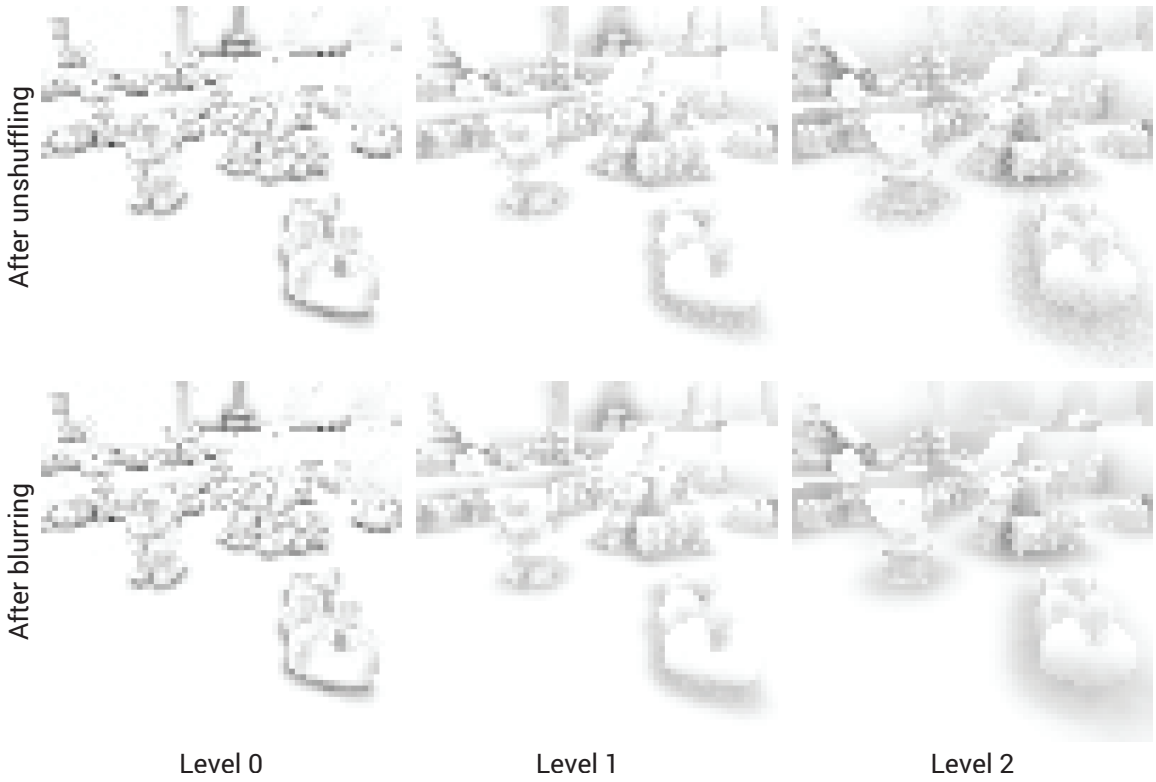


Figure 4.8: The first three levels of a hierarchical framebuffer after unshuffling them for the case of AO (top row). Each column contains the effect of the surfels on their surroundings within a particular distance interval. MC noise is reduced by applying a geometry aware blur (bottom row).

Ambient Occlusion The ambient occlusion of a surfel to a point (pixel) can be computed analytically in the `computeEffect` function using the point-to-disk form factor [Wallace et al. 1989]. The same goes for the surfel's radius of influence.

Ambient occlusion (Figure 4.9, first row) in screen space [Bavoil et al. 2008] looks plausible but misses indirect shadows from triangles not visible in the image. Our approach runs at similar speed and is closer to the ray-traced reference. Remaining artifacts are due to overestimation of occlusions as contributions from the same direction are summed up, discretization into disks and clamping of the maximal effect distance. The top row of Figure 4.10 shows the effect for an animated scene.

Directional Occlusion Directional occlusion [Ritschel et al. 2009b] as defined in Section 2.5.4 combines occlusion computation with image-based lighting. Again, we use the point-to-disk form factor but instead of just accumulating occlusions, a lookup into the environment map is made and the occluded value is subtracted. This results in colored shadows and directional occlusion effects (Figure 4.9, second row).

For this shading effect, our approach can provide better quality at higher speed. As before for AO, SSDO [Ritschel et al. 2009b] lacks occlusions from objects not present in screen space while our approach reproduces them. Due to the large amount of pixels required to gather in SSDO deep screen space can even produce better quality at higher speed.

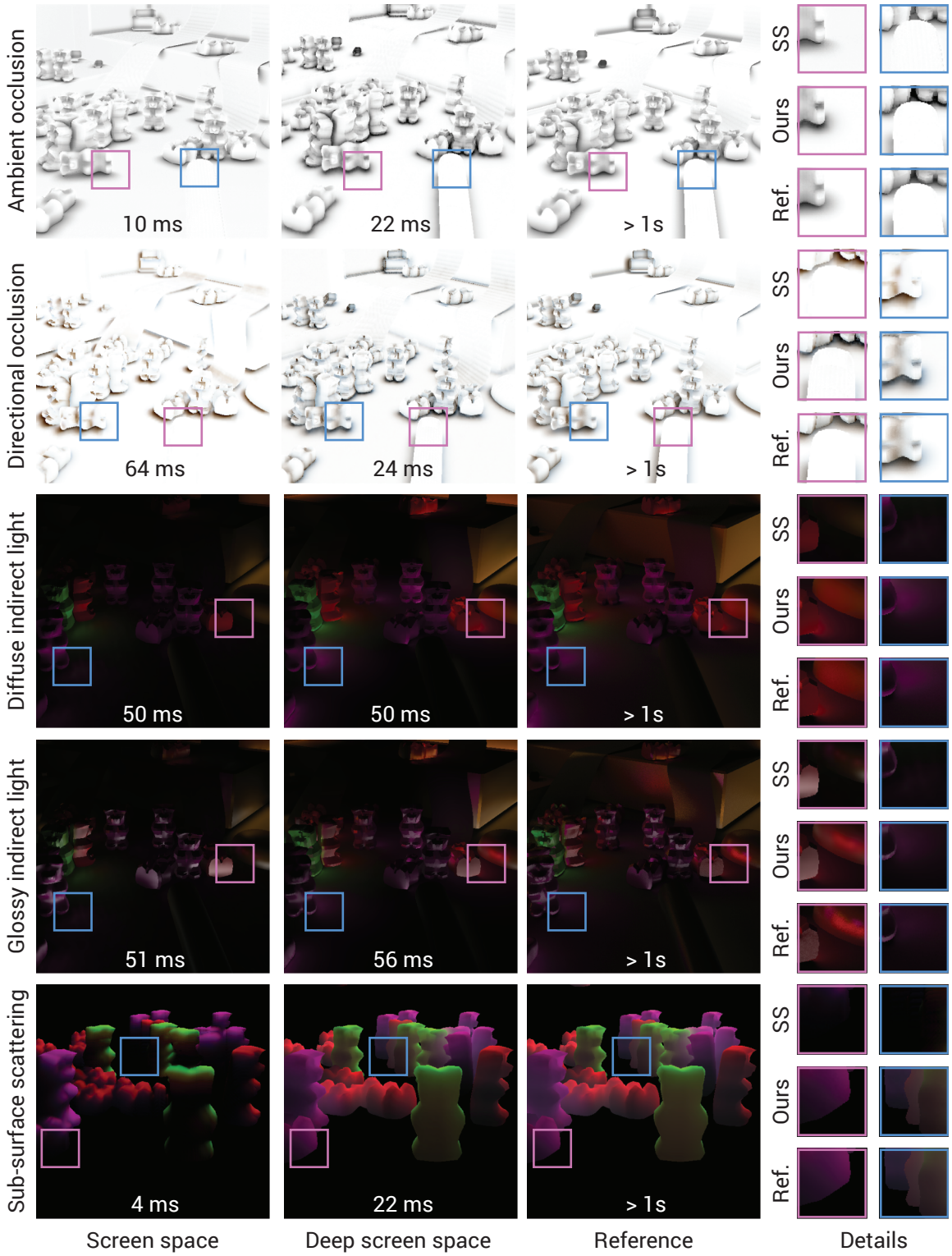


Figure 4.9: Ambient occlusion, directional occlusion, indirect lighting and scattering for our test scene (512 × 512 px, 50 k tris). The first column is a screen space reference (comparable in speed to ours), the second our result, the third is a reference (comparable in quality to ours). The two rightmost columns show details.

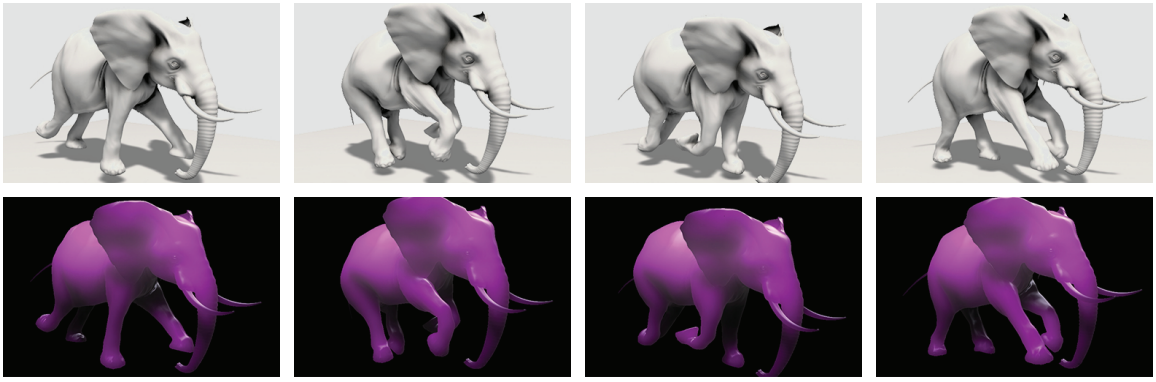


Figure 4.10: Frames of an animation (800 × 600 px) showing temporally coherent ambient occlusion (top, 27 ms) and sub-surface scattering (bottom, 22 ms).

One-bounce Indirect Illumination To approximate one bounce of indirect light (with diffuse senders), surfels are additionally shaded in the tessellation evaluation stage (Section 4.2.1), i. e., irradiance at the position of the surfel and reflectance are used to determine the surfel's radiosity (Section 2.1). The contribution to receiver pixels is again based on a point-to-disk form factor. When computing the splat size using the `getMaxDist` function we can now take advantage of the additional shading information: If a surfel has radius r and radiosity $B \in \mathbb{R}^3$, we compute the effect distance by

$$\frac{r}{\epsilon} \max(B_\mathrm{r}, B_\mathrm{g}, B_\mathrm{b}).$$

This also discards surfels which are in shadow and therefore cannot bounce off any light.

We can achieve results that improve over SSGI [Ritschel et al. 2009b] in terms of quality, in particular regarding the possible effect distance, at comparable speed (Figure 4.9, third and fourth row). The reasons for these advantages are the same as for directional occlusion.

Multiple Scattering For sub-surface scattering, irradiance again has to be determined and stored when producing each surfel. We use the diffusion approximation of Jensen and Buhler [2002] which is evaluated using the same formulas as in the original paper but replacing the gathering using a hierarchy by shell splatting. For simplicity, we chose `getMaxDist` to be the same as for diffuse bounces as after fixing the scattering parameters the maximal distance only varies depending on the amount of irradiance and the surfel's radius.

Regarding performance, we compare our results to screen space scattering [Jimenez et al. 2009]. For a comparison in terms of quality, we distributed 2 million sampling points on the translucent objects and evaluated their contribution to each pixel [Jensen and Buhler 2002]. Using path tracing in this case would be prohibitive. We observe quality similar to Jensen and Buhler [2002] which takes several seconds to compute. Our speed is in the order of milliseconds as for [Jimenez et al. 2009] which is a slightly more efficient but cannot capture scattering at a global scale, e. g., from light not visible in the framebuffer (Figure 4.9, last row). Multiple scattering also works well for animated scenes with deforming meshes (Figure 4.10, bottom row).

4.4 Discussion

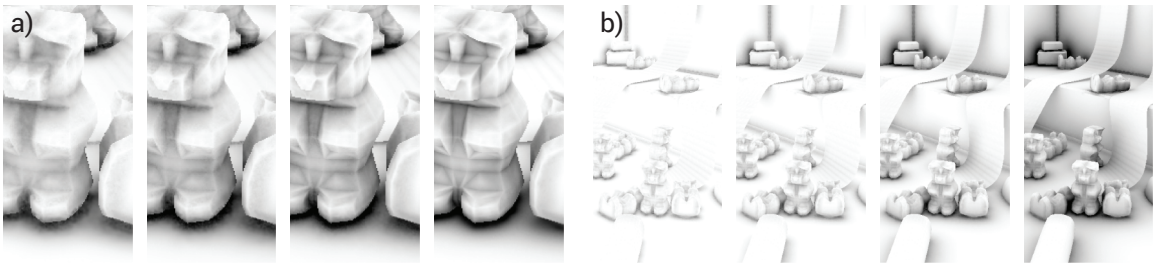


Figure 4.11: Decreasing ϵ reduces blur and noise while computation time increases (a). Increasing the number of levels and keeping ϵ constant increases computation time and effect distance (b).

Parameters Running time and quality of shell splatting depend on the desired image resolution, the number of surfels, the splat size and the depth complexity of the framebuffer. The number of splats depends on the chosen surfel size and scene complexity: scenes with high depth complexity are more costly but the cost for splatting one depth layer is limited by the constant screen space size of the surfels. To capture small-scale effects, e. g., occlusion of a floor on the leg of a chair, we need to choose a sufficiently small surfel scale s. The size of the splats depends on the effect threshold ϵ (as well as on surfel properties for some effects). In combination with a suitable number of precision levels $l_{\max}$ for the hierarchical framebuffer, we need to choose ϵ small enough so that the shading effect covers a sufficient distance in screen space. A larger value for ϵ in conjunction with a higher number of levels can yield the same effective distance range, but the effect will become less detailed because of coarser sampling. We analyze the effect of ϵ and $l_{\max}$ in Figure 4.11. In conclusion, we have three parameters (ϵ, $l_{\max}$ and s) that, on one hand, need to be adjusted but, on the other hand, offer fine control over the trade-off between quality and speed.

Stage	AO	DO	GI	SSS
Surfelization	1.6 ms	1.6 ms	1.6 ms	1.6 ms
Shuffling of the g-buffer for fast lookups	2 ms	2 ms	3 ms	2 ms
Shell splatting	14 ms	16 ms	44.7 ms	14.6 ms
Unshuffling of the hierarchical framebuffer	1 ms	1 ms	1.2 ms	1 ms
Blurring of the hierarchical framebuffer	3 ms	3 ms	5 ms	2.5 ms
Summing of the framebuffer levels	0.4 ms	0.4 ms	0.5 ms	0.3 ms
Total	22 ms	24 ms	56 ms	22 ms

Table 4.1: Computation time for different stages and shading effects.

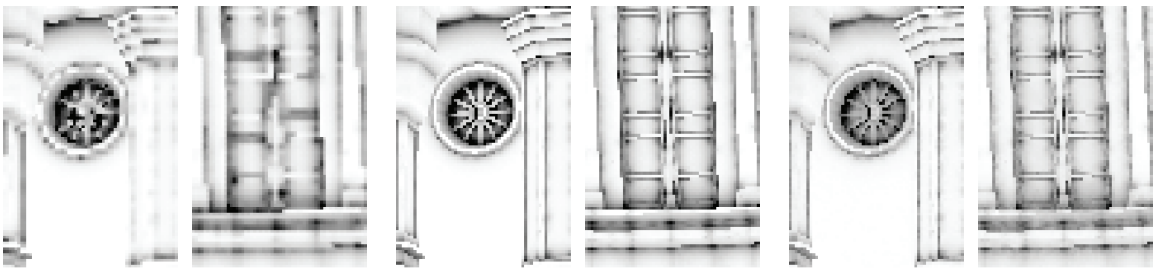
Performance Table 4.1 gives a performance breakdown of different stages of the deep screen space pipeline for the scene used in Figure 4.9 (512×512 px, 50 k triangles).

Scenes with many small or thin triangles require the use of the more sophisticated tessellation approach (Figure 4.5). Table 4.2 shows how time consuming different sub-steps of the tessellation are and how the geometric complexity changes. During the initial tessellation step which is necessary to handle very large triangles, we have to process all of the scenes' triangles. However,

Stage	Time	Data size after
Initial full scene		122 k tris
Tri tessellation	1.2 ms	19 k tris
Fine surfelization	4.1 ms	797 k surfels
Filtering surfels	3.0 ms	375 k surfels

Table 4.2: Amount of data after different tessellation stages (Figure 4.5) at the example of the Graveyard test scene (shown in Figure 6.5).

we can cull the triangles against the view frustum (plus an additional safety margin depending on the effect quality settings). The most costly step in the surfelization is usually the second in which we produce the fine surfel cloud from which the final surfels are selected afterwards because it writes many small surfels. The filtering step again has to consider all of the small surfels, but in practice, it can be integrated into the splatting pass and run concurrently.



Up-sampled, 47 ms Full resolution, 165 ms Ours, 42 ms

Figure 4.12: When computing the effect in half the resolution and up-sampling the result small details are lost (left). Splatting in full resolution produces crisp shading but is slow (middle). Our approach balances the two extremes and is fastest thanks to the hierarchical scheme (right).

Comparison to Algorithmic Alternatives Up-sampling and blurring are common operations in interactive global illumination. Either the image is up-sampled from a low resolution where splatting is efficient but details might be lost or splatting in full resolution is used resulting in high quality but suffering from reduced performance due to overdraw. We compare our approach to those alternatives in Figure 4.12.

Limitations Our approach, like all screen space approaches we are aware of, does not support indirect visibility. This leads to problems like over-occlusion in the case of AO and overestimation of indirect light for GI as in similar approaches [Bunnell 2005; Sloan et al. 2007]. While Bunnell [2005] can account for the problem to some extent by computing the interactions of surfels with each other, we can not as we use a much finer surfel cloud which is larger by a few orders of magnitude and which is not organized spatially. For the same reasons it is unclear how to extend computation from one to multiple indirect bounces, which is the second limitation.

4.5 Comparison to Previous Work

In this section, we will compare our method to the related previous work already introduced in Chapter 3.

Deep screen space is not the first attempt of overcoming the limitations of screen space shading for effects like AO [Mittring 2007; Shanmugam and Arikan 2007; Bavoil et al. 2008], sub-surface scattering [Jimenez et al. 2009] or diffuse bounces [Ritschel et al. 2009b].

First, multi-resolution computation can improve screen space shading, demonstrated for the case of diffuse bounces by Nichols and Wyman [2009]. Here, a hierarchy is built in screen space as a regular quad tree. The contribution of VPLs [Keller 1997] to the screen is computed by splatting at different increasingly coarse resolutions. Instead of reducing the resolution of each layer and then splatting one VPL to one layer, we splat to multiple randomly sub-sampled images. In combination with feature-aware blurring, spatially small details also receive a shading contribution, just with fewer samples. An example is a thin blade of grass: When reducing the resolution, the blade at some point completely disappears. In our approach, only the number of samples that the blade will receive decreases but fine details are never lost completely (cf. Figure 4.12).

Second, occlusion is an issue for screen space shading. The restriction to the first visible surface was addressed using multiple views [Ritschel et al. 2009b] and shadow maps [Vardis et al. 2013] but could also be solved using layered depth images (LDIs) [Shade et al. 1998]. However, occlusion is not the only problem of screen space shading: Under-sampling of geometry under grazing view angles is never resolved even when using LDIs. However, there is no reason why such occluders or emitters should be underestimated depending on the viewing angle.

Next, computation in sweeps along a discrete number of directions [Timonen 2013] has been proposed leading to significantly improved performance but also to banding artifacts. In our approach, we use blurring to reduce the uniform high-frequency noise arising from sub-sampling of the surfel cloud whereas banding is structured and much harder to remove in post-processing.

The gathering of common screen space image filtering has been previously replaced by splatting in the work of Sloan et al. [2007] and McGuire [2010]. The first uses pre-computed sphere proxy geometry with limited geometric detail, the latter generates splatting primitives from triangles. In contrast, we generate our splatting primitives on-the-fly and can capture important details like shadow edges or textures that may vary across a triangle primitive or proxy sphere and hence cannot be represented well using the latter.

Other competing approaches also rely on surfels [Bunnell 2005; Christensen 2008] or points [Jensen and Buhler 2002] but organize them in a hierarchical data structure. However, this hierarchy needs to be built and updated in the case of geometric changes which is costly if the scene is large or undergoing major deformations. Also, the traversal of said hierarchies is algorithmically complex and not well suited for execution in threads running on a GPU. Furthermore, the discretization into a surfel cloud is usually done once in these approaches and limits the geometric detail. We avoid a pre-defined discretization and traversing or building hierarchies altogether. Our approach produces new surfels even when just changing the camera view and is only limited by the geometric detail the scene contains.

Screen space shading is similar to instant radiosity [Keller 1997] where discrete points (VPLs) replaced the finite element polygons of the classic Radiosity algorithm (Section 3.5.1). Reflective shadow maps [Dachsbacher and Stamminger 2005] are a particularly efficient way to produce such points for indirect illumination from a single light. The idea is to rasterize the scene from the view of the light and use the visible parts as emitters to splat illumination. Such approaches work well for a single primary light but no obvious way exists to extend it to general emitters or occluders e. g., in the presence of environment maps or to ambient occlusion occluders.

Finally, our approach relates to the classic idea of micro-polygons in REYES [Cook et al. 1987] that subdivides primitives to become pixel-sized triangles and shades their vertices. Replacing polygons by points has also been used to render large [Wand et al. 2001] or procedural geometry [Stamminger and Drettakis 2001] efficiently. In those methods, the micro geometry creates the final image while we compute shading contributions from the surfels to framebuffer pixels without the surfels ever becoming visible in the final image themselves.

4.6 Conclusion and Future Work

This chapter introduced a general technique for indirect shading of dynamic, deforming geometry that overcomes most limitations of screen space methods while providing similar efficiency. Its main limitations are the approximateness of the surfel discretization and the limitation to a single bounce.

The original deep screen space method as presented in this chapter also lacks a visibility operator between surfels and receivers. In Chapter 6 we will show how to lift this limitation and account for indirect visibility. Furthermore, we have so far only considered light transport between surfels and surfaces. The next chapter, Chapter 5, will see the extension of shell splatting to volumetric receivers, allowing to compute indirect lighting of heterogeneous participating media.

In the future, our level-of-detail scene representation using surfels might become useful for the computation of even more shading effects like depth of field or motion blur. These two applications are different in that the surfel cloud would need to be rendered directly instead of serving e. g., as invisible sender of indirect light. This would require using a much finer grained tessellation with actually pixel-sized surfels. Also the splatting stage would have to be adjusted: a depth of field effect requires splatting of disk-shaped primitives, motion blur can be approximated by splatting lines or curves. It is uncertain, whether a sub-sampling scheme analogous to shell splatting could be devised.

5

Volume Shell Splatting

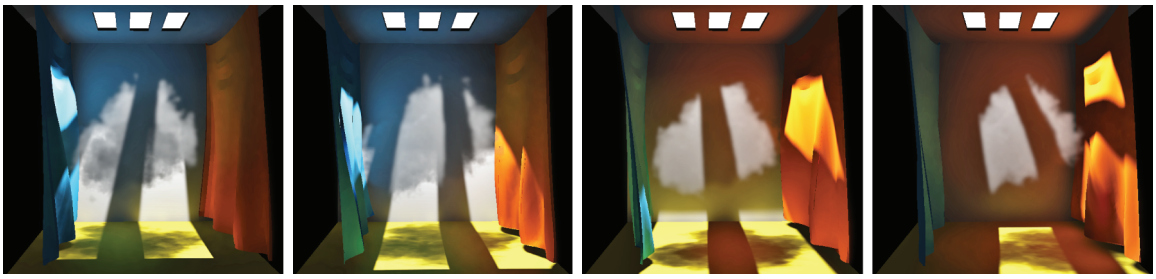


Figure 5.1: Cropped frames from an animated scene with dynamic light, participating medium and surfaces, rendered with our surface-to-volume indirect lighting at 25 fps, 800 × 600 px.

5.1 Introduction

In the last chapter, we introduced deep screen space, a general method to compute various types of surface-to-surface light transport. Those included plausible simulation of indirect light that constitutes a major step towards the realistic appearance of virtual scenes, which is however not exclusive to light transport between surfaces but also applies to participating media. This is why we will now extend deep screen space to also account for surface-to-volume transport.

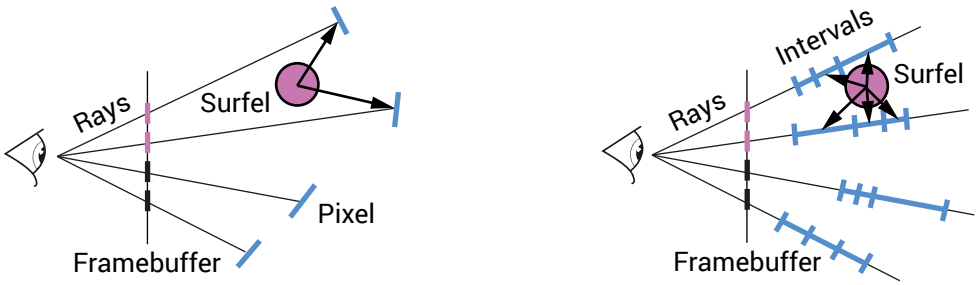
While offline rendering of participating media (Chapter 3) can produce stunning imagery such methods usually do not deliver real-time results, in particular not for interactive scenes. Our algorithm will retain the same advantages as deep screen space: support of scenes which are fully dynamic with respect to geometry and light sources and level-of-detail computation at interactive speed. Additionally, it will handle participating media which may be heterogeneous and which may be animated themselves.

Achieving our goal of adapting deep screen space to participating media mainly requires adjustment of the method's splatting stage (Section 4.2.2), the surfelization (Section 4.2.1) works the same as before. First, a representation of the receiving volume allowing for fast scattering computations is necessary. For this, the current view of the volume is converted into a transmittance interval map containing depth intervals in which the transmittance to the camera is reduced by the same fraction of the total extinction. These intervals then receive the indirect illumination

by splatting, not anymore from surfels to pixels but now from surfels to ray intervals. Again, we use the same hierarchical framebuffer layout as in the original deep screen space method which delivers high precision for surfel-interval pairs that exchange much light and which is coarser for pairs exchanging little, without constructing any explicit hierarchical data structure.

We will now describe the adapted volume shell splatting in Section 5.2, followed by a discussion of results and properties of the method (Section 5.3). The chapter is concluded by a comparison to relevant previous work in Section 5.4.

5.2 Volume Shell Splatting



a) Splatting to surfaces b) Splatting to volume

Figure 5.2: In surface-to-surface shading, surfels influence nearby pixels (a). For the surface-to-volume case (b) surfels scatter information to sub-segments of ray intervals at nearby pixels.

A naive approach to scatter indirect light from a surfel cloud to a volume would be to iterate for each surfel over all points on all camera rays and compute its contribution. However, in practice the number of necessary sampling points is in the order of hundreds and a pixel receives splats from many surfels, making this simple approach prohibitively slow. We will follow the general idea but only traversing nearby rays expected to receive a significant amount of indirect light and using a compact representation of the medium along each camera ray consisting of only a few ray segments. This will allow for a parallel hierarchical evaluation with guaranteed error bounds at interactive rates.

In Section 5.2.1, we will first introduce our volume representation that is suitable for splatting onto, a *transmittance interval map*. Such a map decomposes each camera ray into a low number of intervals which are expected to have similar contribution to the virtual camera sensor. During the shell splatting stage (Section 4.2.2) we then march over the transmittance interval map as detailed in Section 5.2.2.

5.2.1 Transmittance Interval Map

Our strategy is to split the ray under each pixel into n intervals of equal transmittance (Section 2.6.1). This pre-computation is independent for each pixel so we will describe it for a sample pixel p in the following. We define the n ray intervals along the camera ray through pixel p by a sequence $\{d_1, \ldots, d_n \in \mathbb{R}\}$ of distance intervals. The 3D positions $\{\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^3\}$ corresponding to the points where each interval starts or ends can be computed from those distances, the view parameters and the known position of pixel p within the framebuffer.

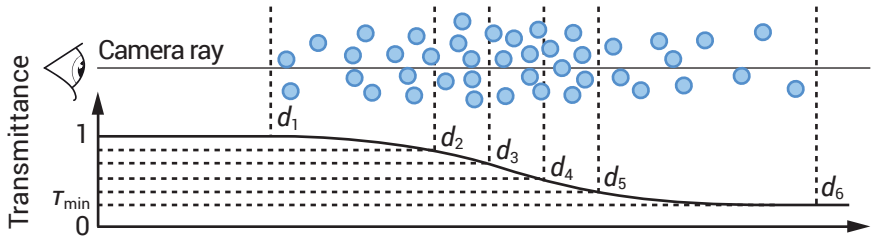


Figure 5.3: Transmittance interval map at one pixel. Intervals between the distances d_1 to $d_n = d_6$ equally divide the transmittance range between the maximum 1 and the minimum τ_{min}.

Let $\tau(d)$ be the transmittance with respect to the camera position at distance d along the ray as defined in Section 2.6.1, which can be computed iteratively by front-to-back ray-marching [Max 1995]. We first determine the starting and end points of the medium along each ray. The starting point formally corresponds to the largest distance d_1 such that $\tau(d_1) = 1$. The end point is located at the distance d_n where τ first reaches its minimum τ_{min}. Both points are computed by ray-marching through the volume once from front to back using a maximum of n_{max} steps and potentially stopping when an opaque surface is encountered. Whether and where the latter occurs can be answered using a deferred shading position buffer.

Once d_1 and d_n have been determined, we start a second ray-marching pass, this time marching from d_1 to d_n and caching the smallest sampling distances d_2 to d_{n-1} such that

$$\tau(d_i) < 1 - i \cdot \frac{1 - \tau_{\text{min}}}{n - 1} \tag{5.1}$$

as shown in Figure 5.3. As τ is approximated by accumulation with finite step sizes, its inversion that we compute is also approximate and the fraction of transmittance o different intervals may still be slightly different. Put differently, the vertical distance intervals between the dotted lines in Figure 5.3 may not be precisely the same. However, these small differences do not become noticeable in practice.

In addition to d_1 to d_n we store the minimal transmittance values τ_{min} for all pixels. They are later used during splatting (Section 5.2.2) to efficiently determine the extinction of the in-scattering on the way to the camera. In the following, we will use the name *transmittance interval map* for the map holding all d_i and the minimal transmittance value τ_{min} for each pixel.

Jittering To avoid banding artifacts, we jitter the length of the first ray-marching step. This is an approach commonly taken by other ray-marching-based algorithms. Similarly, we add this same random offset to all the transmittance thresholds determining where to place the samples per pixel. Equation 5.1 is effectively changed to

$$\tau(d_i) < 1 - (i - \xi_p) \cdot \frac{1 - \tau_{\text{min}}}{n - 1}, \tag{5.2}$$

where ξ_p is value between 0 and 1 chosen uniformly at random for each pixel p. Figure 5.4 demonstrates the effectiveness of this approach.

Encoding During splatting (Section 5.2.2) values from the transmittance interval map have to be fetched hundreds of times, once for each relevant surfel. A memory-efficient encoding is mandatory to reduce bandwith consumption.

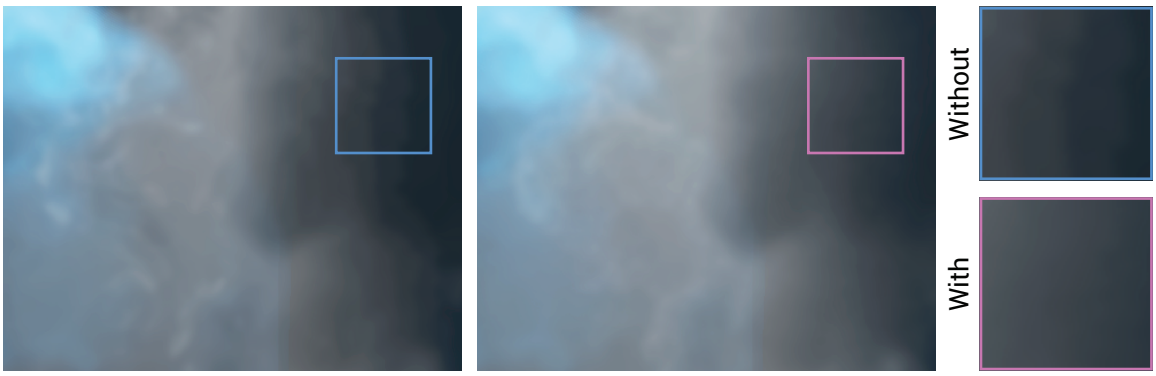


Figure 5.4: To avoid banding artifacts (left) we use jittering of the ray intervals (right).

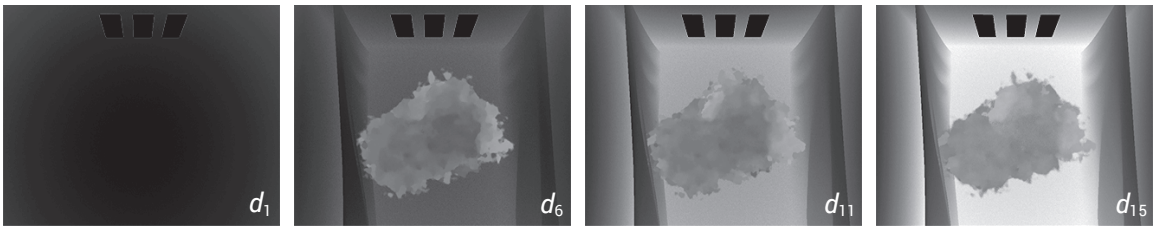


Figure 5.5: Four sampling depths from a transmittance interval map. For each pixel, the respective value of d_i is shown; brighter pixels correspond to larger depth values. Note the smaller range of grey values, i. e., smaller depth range, for the cloud compared to the surroundings. This is due to the adaptive sampling which concentrates samples in denser areas.

We use $n = 16$ and store the values of d_i in an unsigned integer texture of $4 \times 32 = 128$ bits per pixel. Instead of storing absolute depths, we encode the number of ray marching steps between the different sample positions (and from the camera to the first sample) in 8 bit each. Assuming that the samples are not further than 255 steps apart, which is valid if the medium's density does not vary at extremely high frequencies, this encoding allows a precise reconstruction of sampling positions given the camera's position and the ray marching step length (Figure 5.5).

Finally, to compute the in-scattering light at each sampling position the scattering coefficients $\sigma_{s,i}$ of the medium at those points are needed. Since the sampling positions are known we can prefetch the coefficients. For our case, we store the coefficients $\sigma_{s,1}$ to $\sigma_{s,n}$ at the respective distances d_1 to d_n by encoding them relative to the maximal coefficient $\sigma_{s,\max} = \max\{\sigma_{s,1}, \ldots, \sigma_{s,n}\}$ in 8 bit each. Again, we use a 128 bit texture. The remaining 8 bits are used to encode $\sigma_{s,\max}$ relative to the maximal scattering coefficient across the whole medium which we assume to be known.

5.2.2 Splatting to the Ray Segments

To light a volume from a surface we need to enumerate pairs of deep screen space surfels, representing the surface, and pixels of the transmittance interval map, representing the volume. Instead of enumerate all such pairs, which would be computationally expensive, we enumerate pairs with a probability that is proportional to an approximate bound on the amount of exchanged light. Doing so, strongly-coupled pairs are more likely to be evaluated resulting in higher precision

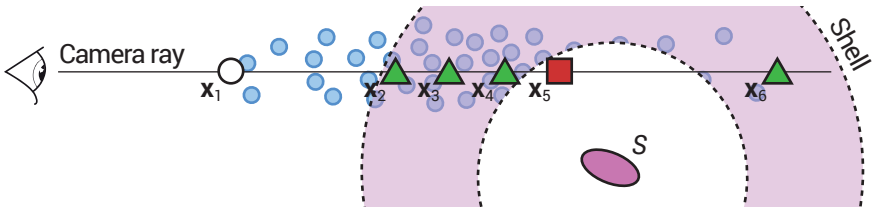


Figure 5.6: The contribution of the surfel S is only evaluated at those sampling points $\mathbf{x}_i$ that are actually located inside the shell (triangles) associated with the surfel.

while pairs with weak exchange are computed rarely and with less precision. The scattering mechanism using a hierarchical interleaved framebuffer with different precision levels is the same as in Section 4.2.2, just the computations eventually happening for each receiver pixel are different.

Surfel Information To compute the light transport from the surfels to the volume, surfels have to be shaded during surfelization as already described in Section 4.2.1 to compute their radiosity. In particular, the surrounding participating medium has to be taken into consideration. While any approach to compute light scattering works, we used ray marching for rendering the results presented in this chapter.

Computing the Splat Size As outlined in Chapter 4Splatting, the size of the splat for a surfel is determined by a function `getMaxDist` depending on the surfel's attributes and a threshold ϵ. Given the radiosity B_S and radius r_S of the surfel, we compute the world-space radius of the splat as

$$r_S \max\left(B_{S,\mathrm{r}}, B_{S,g}, B_{S,\mathrm{b}}\right)/\epsilon,$$

which is based on the assumption of a quadratic fall-off of the radiance received by other points with increasing distance. Note that surfels which are in shadow are discarded automatically as their radiosity amounts to 0. Unfortunately, we cannot easily take the density of the medium around the surfel into account here.

The bound for volumes is naturally less tight, in particular for small splats: When a small splat is drawn to nearby pixels the potential for overdraw is small. When a small splat is drawn to nearby rays little overdraw in the image occurs, however, due to the additional depth dimension, a small splat is likely to have an effect on only a small fraction of the ray intervals.

Splatting At each pixel p covered by a splat, we are given a surfel S with its position $\mathbf{x}_S$, normal $\vec{\mathbf{n}}_S$, radius r_S and radiosity B_S as well as the inner and outer radius of the shell associated with the splat. Using three texture lookups we can fetch the transmittance interval map containing d_1 to d_n, the transmittance $\tau_{\min}$, the cached scattering coefficients $\sigma_{\mathrm{s},i}$ and the maximum coefficient $\sigma_{\mathrm{s,max}}$ with respect to which they have been encoded.

We march over all sampling points $\mathbf{x}_2$ to $\mathbf{x}_n$, which are recovered from the depths d_i, adding up the radiance contributed by S to the pixel p. At each point, we first have to check whether $\mathbf{x}_i$ actually is inside the shell belonging to the splat (Figure 5.6). If this is not the case we proceed

with the next point, otherwise we compute the contribution of the surfel S by

$$\underbrace{S_{\text{surfel}}(\mathbf{x}_i, -\vec{\boldsymbol{\omega}}_{\text{in}})}_{\text{In-scattering}} \cdot \underbrace{\left(1 - \frac{i\,(1-\tau_{\min})}{n-1}\right)}_{\text{Transmittance}} \cdot \underbrace{(d_i - d_{i-1})}_{\text{Segment length}}, \tag{5.3}$$

where $\vec{\boldsymbol{\omega}}_{\text{in}}$ is the direction from the camera towards $\mathbf{x}_i$. Accumulating these contributions corresponds to evaluating a Riemann sum to approximate the integral in Equation 2.17 from Chapter 2 by a discrete sum. S_{surfel} is a source term (Section 2.6.2) corresponding to in-scattering from a surfel S at $\mathbf{x}_i$ towards the camera. The second factor in Equation 5.3 corresponds to the transmittance between d_i and the camera the value of which can be derived efficiently from the definition of d_i (cf. Section 5.2.1), and the third factor is the length of the ray segment associated with the sample.

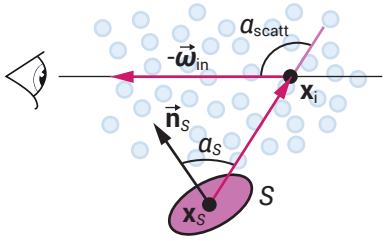


Figure 5.7: In-scattering at a point $\mathbf{x}_i$ from a surfel S at position $\mathbf{x}_S$ with normal $\vec{\mathbf{n}}_S$ towards the camera.

The in-scattering is computed as

$$S_{\text{surfel}}(\mathbf{x}_i, -\vec{\boldsymbol{\omega}}_{\text{in}}) = \sigma_{\text{s},i} \cdot \rho(\alpha_{\text{scatt}}) \cdot B_S \cdot \frac{\pi {r_S}^2 \max(\cos\alpha_S, 0)}{\max\left(\|\mathbf{x}_i - \mathbf{x}_S\|_2^2, \delta\right)},$$

where α_S is the angle between the surfel's normal and the vector pointing from the surfel to $\mathbf{x}_i$ and ρ is the phase function depending on the scattering angle α_{scatt} between the vector from $\mathbf{x}_S$ to $\mathbf{x}_i$ and $-\vec{\boldsymbol{\omega}}_{\text{in}}$. To avoid intensity singularities near the surfels, we clamp the distance $\|\mathbf{x}_i - \mathbf{x}_S\|_2^2$ in the computation of the in-scattering to a minimum of $\delta > 0$. The somewhat complicated geometrical configuration is depicted in Figure 5.7.

Blurring After splatting and un-shuffling the array of textures (Section 4.2.2) containing the solution for the different shells, we perform a blurring operation to hide Monte Carlo noise. While deep screen space for surface-to-surface transport can take advantage of a spatially-varying blurring kernel depending on the scene geometry, this is not possible for volumes which do not have a single surface. We use an isotropic Gaussian blur instead. To perform this efficiently, we build a MIP map over the un-shuffled hierarchical framebuffer and replace level l with an up-sampled version of level $l+1$ of its MIP map. This effectively leads to a blur with a kernel whose extent increases by a factor of 2 for each framebuffer level, similar to the original method.

5.2.3 Implementation Details

Surfel Lighting We used point lights as the light sources in our experiments. Visibility was determined using shadow mapping.

Direct Single Scattering Our approach accounts for indirect single scattering from surfaces. A typical rendered image should however also show the effects if direct single scattering from the main light sources in the virtual scene. For this, we perform analogous computations to the surfel splatting stage merely replacing the surfel by the light source and dropping the restriction to a shell. This way, we can re-use the cached scattering coefficients and known transmittance values.

Number of Ray Marching Steps As a hint for the necessary number of ray-marching steps when preparing the transmittance interval map, the depth of the wall in our test scene (Figure 5.8) corresponded to roughly 200 steps. The maximal number of steps $n_{\max}$ should cover the whole range to the far plane.

Phase Function For our results, we used the popular phase function by Henyey and Greenstein [1941] introduced in Section 2.4.2 with varying anisotropy settings.

5.3 Results and Discussion

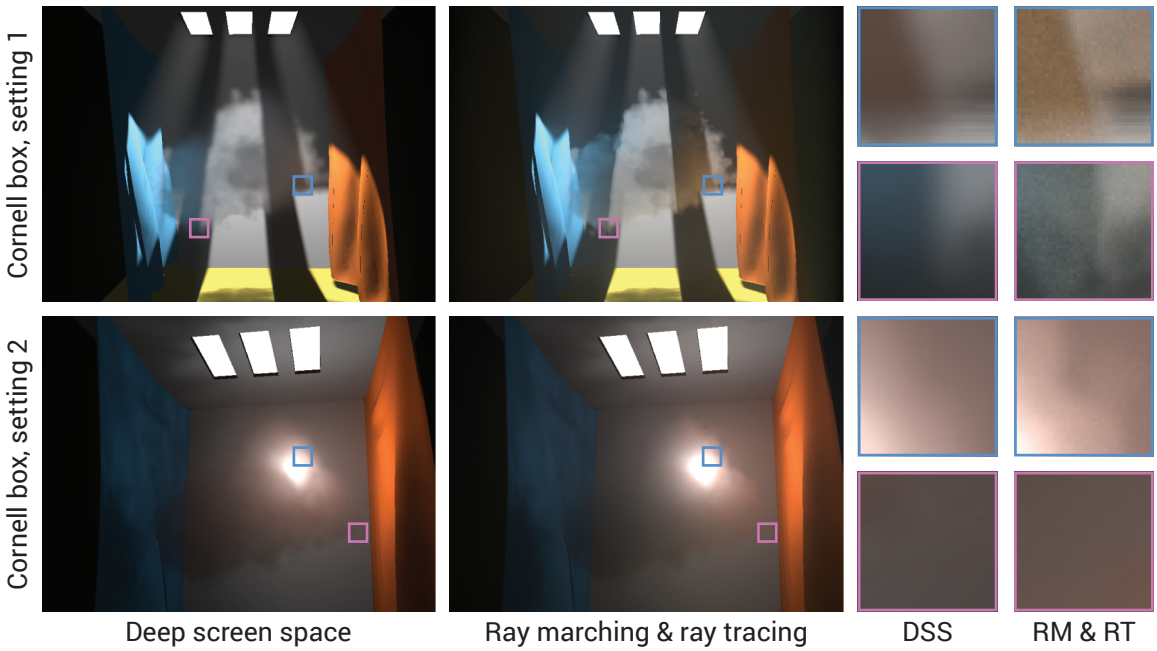


Figure 5.8: The first column contains our results, the second shows the reference, the last two columns contain details of our result and the reference, respectively. The single scattering has been linearly scaled to accentuate it.

In Figure 5.8, we compare results obtained using our method to a reference based on ray-marching and ray-tracing for the gathering of indirect light. Where not specified otherwise, we used a resolution of 800×600 px and an Nvidia GTX 770 GPU for rendering. The number of levels for the hierarchical framebuffer was constantly set to six with an effect threshold $\epsilon = 0.06$ and about 14 k surfels were used to represent the surfaces of the scene.

Visual Comparison A notable difference is in brightness. On one hand, our method exhibits increased brightness towards the inside of the medium due to the lack of attenuation between the surfels and the receiving sampling points. On the other hand, the brightness is underestimated over the long range since we clamp contributions less than ϵ. The sub-sampling in our method leads to noise which makes blurring necessary. This is in particular visible around the boundaries of the medium which are less defined. Still, our result looks plausible while being orders of magnitude faster than the ray-marching approach.

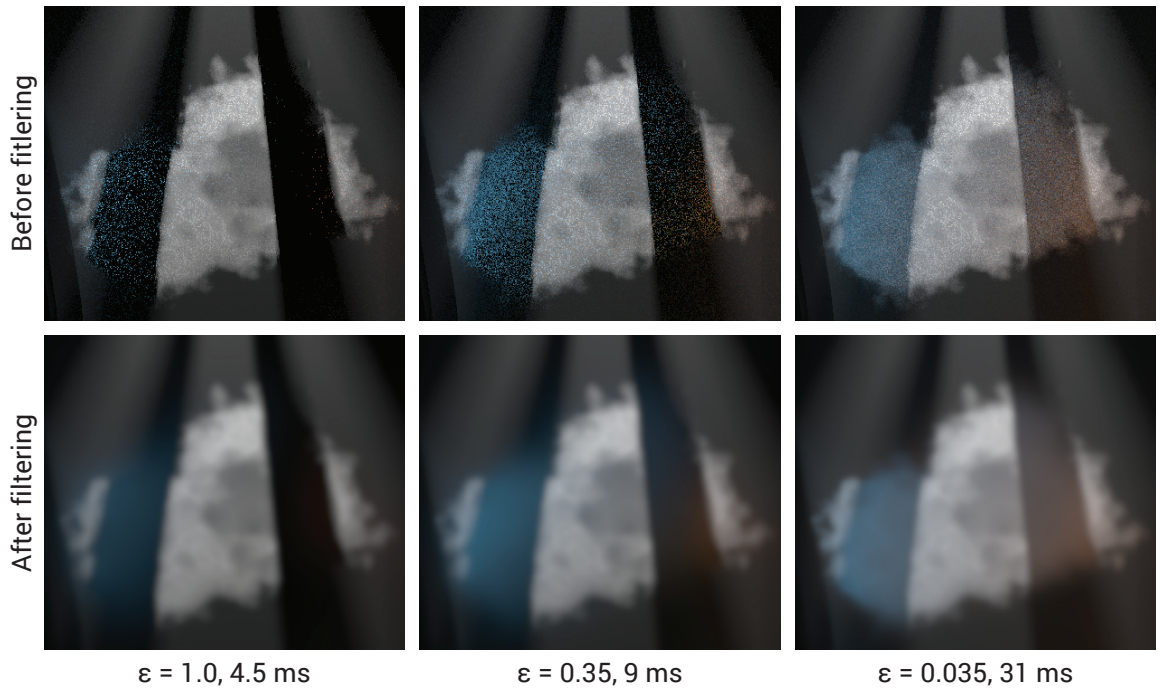


Figure 5.9: Crops from the Cornell box scene using different values for ϵ, without (top) and with (bottom) blurring. Only direct and indirect single scattering are shown. The timings correspond to the splatting for indirect scattering alone as other parts are unaffected by the changed setting.

The main parameter influencing quality is the effect threshold ϵ. Figure 5.9 shows how it influences the result and computation times. For lower quality settings (larger values of ϵ), the cloud becomes darker and its boundary is less defined due to stronger blurring.

Stage		Time
Shadow Mapping		0.6 ms
Transmittance Interval Map	+	8 ms
Scattering Coefficient Caching	+	1.4 ms
Surfel Cloud Generation	+	3 ms
Splatting	+	20 ms
Unshuffling	+	1 ms
Blurring	+	3 ms
Summing	+	0.4 ms
Total	=	37.4 ms

Table 5.1: Computation time for different stages.

Performance Table 5.1 gives an exemplary performance breakdown for the top row of Figure 5.8. As is to be expected, the running time is dominated by the time for splatting. Another costly part is generating the transmittance interval map. We only list the timings for computations related to the indirect lighting of the volume. Besides classic direct lighting of the surfaces, our result images also show direct single scattering and absorption of direct light by the medium which typically took about 1.5 ms and 3 ms to compute, respectively.

Limitations The main shortcoming of deep screen space is its lack of indirect visibility, i. e., we have no information about the visibility between the surfels and the points receiving shading from them. In particular, the indirect light is not occluded by surfaces or attenuated by the medium.

One theoretical shortcoming of our method is the fact that it does not allow to bound the effect of a surfel onto rays but only onto ray intervals. If a surfel only affects a small number n' of intervals in a ray consisting of a much larger number $n \gg n'$ of intervals, the splatting will still traverse all of the n ray intervals. In future work, it could be useful to also cull groups of ray intervals.

Our chosen encoding for the volume map using sixteen 8 bit values to fit everything into just one texture is not sufficient if either the density of the medium or the lighting exhibit high frequencies, e. g., due to rapidly changing visibility. However, nothing prevents the use of more samples when accepting slower computations.

We have only shown renderings of monochromatic media, i. e., media whose properties do not vary with the wavelength. The extension to the more general case only amounts to performing analogous computations for each wavelength, though, obviously at larger computational cost.

5.4 Comparison to Previous Work

For indirect lighting of volumes, many offline methods based on the idea of instant radiosity [Keller 1997] exist. Virtual point lights (VPLs) are distributed in the scene from which light is gathered at sampling points inside the medium [Raab et al. 2008]. One shortcoming of those methods is that the discretization to points leads to intensity singularities close to the VPLs. This problem has been overcome by means of intelligent bias compensation [Engelhardt et al. 2012] or exchanging the point lights by ray lights [Novák et al. 2011]. Weber et al. [2013] show how to compensate for changes in the rendered volume in an efficient way by progressively readjusting the VPLs.

Volume rendering based on diffusion [Fattal 2009; Kaplanyan and Dachsbacher 2010] is a general solution to render different types of illumination, also supporting multiple scattering. This sometimes includes indirect light from surfaces [Elek et al. 2014] by using VPLs. The rendered volumes however typically have a low spatial resolution and the interaction between surfaces and volumes is limited to a low number of VPLs. Our technique resolves fine details between surface pieces and their nearby rays on a resolution close to the pixel resolution, at similar speed, and uses thousands of virtual lights.

For the case of single scattering from direct light in homogeneous media, recently highly efficient methods emerged, e. g., by making use of prefiltering and rectification [Klehm et al. 2014]. In contrast, our method deals with indirect single scattering in heterogeneous media.

Regarding indirect single scattering from specular surfaces (volume caustics) in heterogeneous media, the method by Hu et al. [2010] produces plausible results at interactive rates. Leveraging that specular surfaces reflect light in only few directions, pixels affected by caustics are bounded by lines which are cheap to splat. However, our method deals with reflections from diffuse surfaces where the indirect light is spread over a much larger region of space.

Our transmittance interval map is similar to Deep Shadow Maps [Lokovic and Veach 2000] but created from the sensor's perspective. In contrast to deep shadow maps which store a mapping

from depths to transmittance values, the transmittance interval map stores an inverse mapping from transmittance values to depths. Different from the original, our map is built in linear time while avoiding sorting or iteration over the transmittance values which do not fit GPUs.

Opacity shadow maps [Kim and Neumann 2001] seek to improve the efficiency of deep shadow maps by representing the volume using geometric primitives and rasterizing them to planar maps which perpendicular to the light's direction and integrating the density along rays from the light. In contrast to that, we do not use planar slices as we cannot expect them to divide the medium into intervals of similar importance and yielding good sampling intervals. Deep opacity maps [Yuksel and Keyser 2008] iterate the idea by using slices of same depth with respect to the depth at which the medium is entered in each pixel. However, the approach targets hair rendering and, using regular intervals, implicitly assumes a largely homogeneous medium. Again for rendering hair, Mertens et al. [2004] construct a compact representation of 1D visibility functions along rays, too, however it works by rasterizing individual hairs and cannot be directly transferred to continuous volumes.

5.5 Conclusion and Future Work

In this chapter, we presented a method to compute light transport from surfaces to volumes in fully dynamic scenes in real-time. This was achieved by using two types of discretizations that allow for efficient level-of-detail radiance transfer. The first is the deep screen space surfel cloud modeling the senders of indirect light relevant for the current view. The second is the transmittance interval map that re-parameterizes the volume as a collection of ray intervals, and that is also view-adaptive. Transport from surfaces into the volume can then be performed by modified shell splatting. All steps can be performed at interactive rates for dynamic scenes and work without maintaining any pre-computed data structures that have to be re-used across multiple frames.

Even though we have extended the set of possible light paths which we can simulate using deep screen space, there are more general forms of transport still lacking. So far, our senders of indirect light have always been surfaces but transport from volumes to surfaces might also be of interest, e. g., for self-emitting volumes such as they are used to render explosions. This would require an appropriate level-of-detail proxy representation of the volume itself, maybe not using surfels anymore but using a cloud of spheres. Another limitation so far is the assumption of diffuse senders. An extension to glossy receivers could be enabled by differently shaped splatting primitives like ellipses instead of disks.

6

The Bounced Z-Buffer

Figure 6.1: Screen space methods for indirect illumination such as SSGI (left) not only underestimate effects of invisible geometry (left, blue inset) but also routinely ignore visibility (right, purple inset). Our approach accounts for indirect visibility (middle) to produce results similar to a ray tracing reference (right). Our approach renders this scene containing 202 k triangles in 65 ms at 768 × 512 px.

6.1 Introduction

Many real-time approaches omit correct handling of visibility between senders responsible for a shading effect and the corresponding receiving points. This leads to different kinds of bias. In the case of global illumination, the senders reflect indirect light into the direction of the receivers and lack of indirect visibility becomes visible as light bleeding through surfaces (Figure 6.1, left). In the case of ambient occlusion, senders contribute to darkening at nearby receiving points as they occlude parts of the hemisphere over the receivers. Incorrect treatment of visibility here leads to summing up multiple occlusion events from the same direction and the resulting indirect shadows consequently appear too dark compared to a reference (Figure 6.2, left).

In this chapter, we will extend a class of indirect lighting algorithms that splat (Section 3.6.2) shading to a framebuffer to include indirect visibility. To this end we propose the bounced z-buffer: While a common z-buffered framebuffer, at each pixel, maintains the distance from the closest surface and its radiance along a direction from the camera to that pixel, our representation

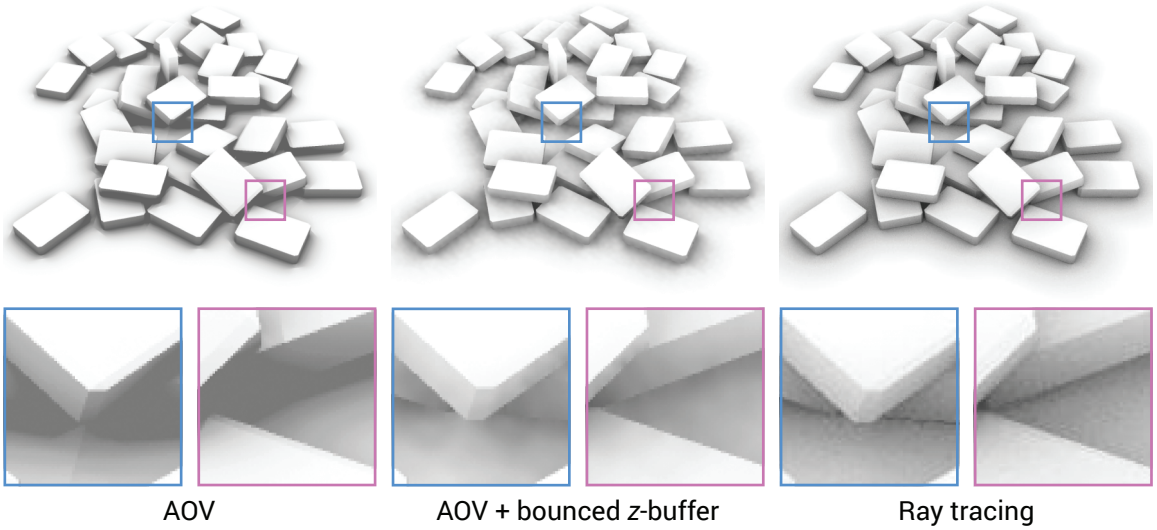


Figure 6.2: Many splatting-based methods like AOV over-occlude dense geometry which is then compensated by tone-mapping (left). Augmenting a bounced z-buffer (middle) retrieves more detailed shading in problematic areas and appears closer to the reference (right).

contains the distance from the closest surface and its radiance after one indirect bounce into a certain other direction. Consequently, with bounced z-buffering only the splat from the nearest emitter in the previously chosen direction contributes to each pixel. Importance-sampling the bounced directions according to the product of cosine term and BRDF and sharing information between neighboring pixels allows to approximate full shading.

The idea is simple to implement and can be used for different splatting-based approaches that lack visibility. More precisely, we demonstrate its use for ambient occlusion volumes (AOV, Figure 6.2) [McGuire 2010] and deep screen space (Figure 6.1, also cf. Chapter 4).

In the next section, we will introduce the general idea of the bounced z-buffer. Then, we will show how to extend deep screen space to use a (hierarchical) bounced z-buffer (Section 6.3). Finally, we will evaluate its effectiveness in Section 6.4 and how it relates to existing methods (Section 6.5).

6.2 General Method

We propose to add visibility to approaches that use splatting to transfer information from geometry, represented as triangles or an approximate cloud of disks, to pixels on which that geometry potentially has an effect [Dachsbacher and Stamminger 2006; Sloan et al. 2007; Nichols and Wyman 2009; McGuire 2010; Nalbach et al. 2014a]. Splatting nicely fits the parallel processing model of GPUs and is output-sensitive because we can cull primitives too far from view frustum bounds to have a noticeable effect on visible primitives.

Rasterization can be seen as splatting of the scenes' primitives onto rays from the camera while tracking for each pixel the one closest to the camera using a z-buffer. We transfer this idea to splatting-based GI and AO approaches. The first difference is that we are interested in not only one direction per pixel following a simple pinhole projection but in multiple ones depending on the surface BRDF. The second difference is that the splats do not simply correspond to projections

of the primitives anymore but to projected regions of influence with respect to the effect being computed. Adding indirect visibility requires three steps: modifications to the framebuffer setup and the splatting (Section 6.2.1 and Section 6.2.2) as well as a final reconstruction step (Section 6.2.3).

6.2.1 Direction Setup

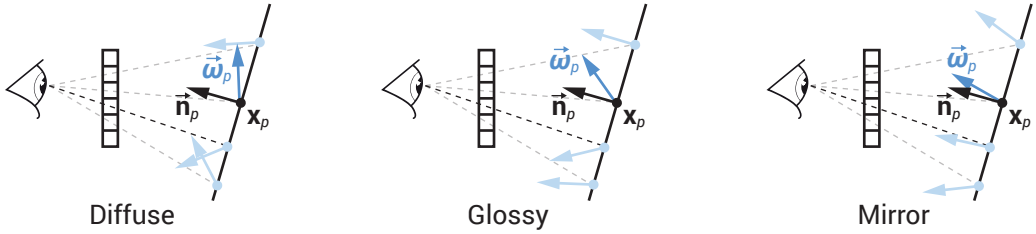


Figure 6.3: Sample directions (blue) in a bounced z-buffer are chosen according to view, surface normal and BRDF at each pixel. Here, the camera is looking at a flat surface for simplicity.

Our input is a deferred shading buffer, rasterized from the view of the camera, which provides for each pixel p the position $\mathbf{x}_p$, normal $\vec{\mathbf{n}}_p 3$ and material information ρ_p, the latter in the form of parameters to an analytical BRDF model, of the primitive visible at p. Based on this information each pixel is associated with a direction $\vec{\boldsymbol{\omega}}_p$ as depicted in Figure 6.3. For AO, uniform sampling of the hemisphere centered around the surface normal is used (Figure 6.3, left), for GI importance sampling according to the product of BRDF and cosine term, to also deal with glossy receivers (Figure 6.3, middle and right). The sampled directions are stored in a two-dimensional buffer.

6.2.2 Splatting

While splatting-based approaches typically use blending [Dachsbacher and Stamminger 2006; Nichols and Wyman 2009; McGuire 2010; Nalbach et al. 2014a] to sum up contributions from multiple primitives, we only want to keep the information from the closest one. Therefore any blending is disabled and replaced by depth-buffering. When a splat, for example in the form of a point [Nalbach et al. 2014a] or a polygon [McGuire 2010], is drawn to the framebuffer for a primitive P, at each covered pixel p, first an intersection test of the ray from $\mathbf{x}_p$ into direction $\vec{\boldsymbol{\omega}}_p$ and P is performed. Note, that this is just a single-primitive intersection that does not require an acceleration structure. If there is no intersection, the fragment is discarded. However, if the primitive is hit at position $\mathbf{y}$, the contribution of P to $\mathbf{x}_p$ is computed — this is either radiance from a Lambertian or glossy sender or occlusion for AO shading — and the contribution is written to the respective framebuffer pixel. The squared distance between $\mathbf{x}_p$ and $\mathbf{y}$, scaled by a maximal distance $d_{\max}$ to keep it in the range $[0,1]$, is used as the fragment's depth value z_p. Due to the depth buffering, at each pixel p only the result from the primitive closest to $\mathbf{x}_p$ in direction $\vec{\boldsymbol{\omega}}_p$ among all those which were splatted onto p is kept. Figure 6.4 gives an example.

An important observation is that the fact that the ray associated with some pixel p hits a primitive P does not imply that a splat caused by P will also cover p. This is because there usually is a fall-off of the shading effect limiting the splat size. For example, for AO only primitives within a certain radius are considered as occluders.

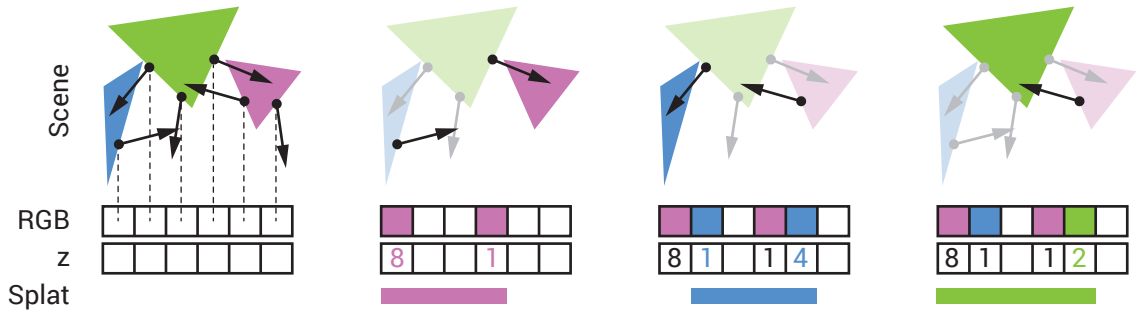


Figure 6.4: First: An empty framebuffer. For each pixel, a ray (arrows) starting at the first visible surface point has been sampled (Section 6.2.1). Second: A splat caused by the purple primitive covers the four leftmost pixels in the framebuffer but only two of the associated rays (opaque) actually intersect the primitive and lead to buffer updates. Last: Due to depth buffering, finally, each pixel p only contains the result from the closest primitive in direction $\vec{\omega}_p$. Note, that the location and size of the splats depend on the respective algorithm and are in general different from a simple projection of the primitive to the screen.

6.2.3 Reconstruction

After all primitives have been splatted, the framebuffer represents an unstructured sampling of the field of incident indirect illumination (or occlusion). From this, exitant indirect illumination is computed in two steps: Importance weighting and filtering.

Weighting As the sample directions are importance-sampled, we first need to divide the incident indirect illumination by the probability density α_p for picking direction $\vec{\omega}_p$ at position $\mathbf{x}_p$ with normal $\vec{\mathbf{n}}_p$. This can either be implemented by storing α_p when $\vec{\omega}_p$ is generated as described in Section 6.2.1 in an additional buffer or by re-computing it on-the-fly.

Filtering The splatting stage computes $L_{\text{dir}}(\mathbf{x}_p, -\vec{\omega}_p)$, the light that arrives at $\mathbf{x}_p$ after a single surface reflection from direction $\vec{\omega}_p$. To reconstruct the exitant indirect illumination $L_{\text{ind}}(\mathbf{x}_p, \vec{\omega}_{\text{out}})$ at pixel p into the direction $\vec{\omega}_{\text{out}}$ of the camera, a weighted sum over the incoming radiance values $L_{\text{dir}}(\mathbf{x}_q, -\vec{\omega}_q)$ of all pixels q in a neighborhood $\mathcal{N}_p$ around p is used:

$$L_{\text{ind}}(\mathbf{x}_p, \vec{\omega}_{\text{out}}) = \frac{\sum_{q \in \mathcal{N}_p} w(p,q)\, f_{\text{r}}(\vec{\omega}_{\text{out}}, \mathbf{x}_p, \vec{\omega}_q)\, L_{\text{dir}}(\mathbf{x}_q, -\vec{\omega}_q)\, \langle \vec{\mathbf{n}}_p, \vec{\omega}_q \rangle}{\sum_{q \in \mathcal{N}_p} w(p,q)}.$$

The weight $w(p,q)$ depends on the differences in position, normal and material for pixels p and q, for example using a Gaussian fall off. In other words, in one step, the incident indirect illumination at q is used to reconstruct the incident indirect illumination at p and at the same time convolved with the BRDF f_{r} and the geometric term $\langle \vec{\mathbf{n}}_p, \vec{\omega}_q \rangle$.

Thanks to the simplicity of the approach, scenes can be rendered in high resolution, resulting in very similar values of position, normal and material. A filter with large support in the spatial, angular and material domain results in less variance but has a stronger bias, compared to a less-biased sharper filter yielding a noisier result. Our reconstruction defers the BRDF and geometry term to be computed exactly once per directional sample per pixel so that BRDF and geometry details in the framebuffer (e. g., from bump maps) are preserved.

6.3 Z-Buffered Deep Screen Space

In this section, we will show how to add a bounced z-buffer to the deep screen space pipeline. The surfelization process is identical to the original method described in Chapter 4. Thus, we will focus on the necessary changes to the hierarchical framebuffer and on how shell splatting has to be adapted.

Construction of the Hierarchical Framebuffer

To add visibility, for every pixel p, one unique direction ω_p is sampled as described in Section 6.2.1. The direction buffer is shuffled in the same way as this is done with other deferred buffers (Section 4.2.2). In particular, this means that for each pixel, the same direction is used on all levels of the framebuffer hierarchy.

Splatting

The buffer is splat with shells as explained in Section 4.2.2 but now visibility is tracked for each pixel on each level as explained in Section 6.2.2. At this stage, each level of the hierarchy maintains its own z-buffer.

After splatting and before reconstruction the framebuffer levels have to be combined. As only the contribution of the closest surfel in direction $\vec{\boldsymbol{\omega}}_p$ is relevant, this is done by selecting the result from the finest level that was hit by a surfel. Since the levels correspond to disjoint distance ranges and are ordered with respect to increasing distance, this yields the most "precise" result among all levels.

As an optimization, we can drop the check whether a fragment is really belonging to the splatted shell (Section 4.2.2). While it is necessary in the original method to ensure a disjoint distribution of the effect among different precision levels, this is not the case for z-buffered deep screen space. So, instead of discarding a fragment at a pixel p corresponding to a world space position $\mathbf{x}_p$, we compute which shell around the surfel which $\mathbf{x}_p$ belongs to and then perform the same computations as if the fragment was actually placed on the respective level. Note, that in the end only the highest-precision hit for each pixel / direction is kept, so we never override hits found on finer levels. We may override "non-hits" on finer levels by actual surfel-hits but we never replace information computed with one precision (i. e., sub-sampling rate) by information computed with a lower precision. In practice this approach is not only faster but also produces cleaner results.

Reconstruction

The final image is then reconstructed as explained in Section 6.2.3. Note that when splatting, different from the original approach, it is not advised to reduce the size of splats from darker surfels as they now contribute valuable information as blockers of indirect light.

6.4 Results and Discussion

In the following, we demonstrate the z-buffered deep screen space approach presented in the previous section applied to AO and one-bounce indirect illumination with diffuse as well as glossy receivers. Results for combining our approach with ambient occlusion volumes [McGuire 2010] are shown in Figure 6.2.

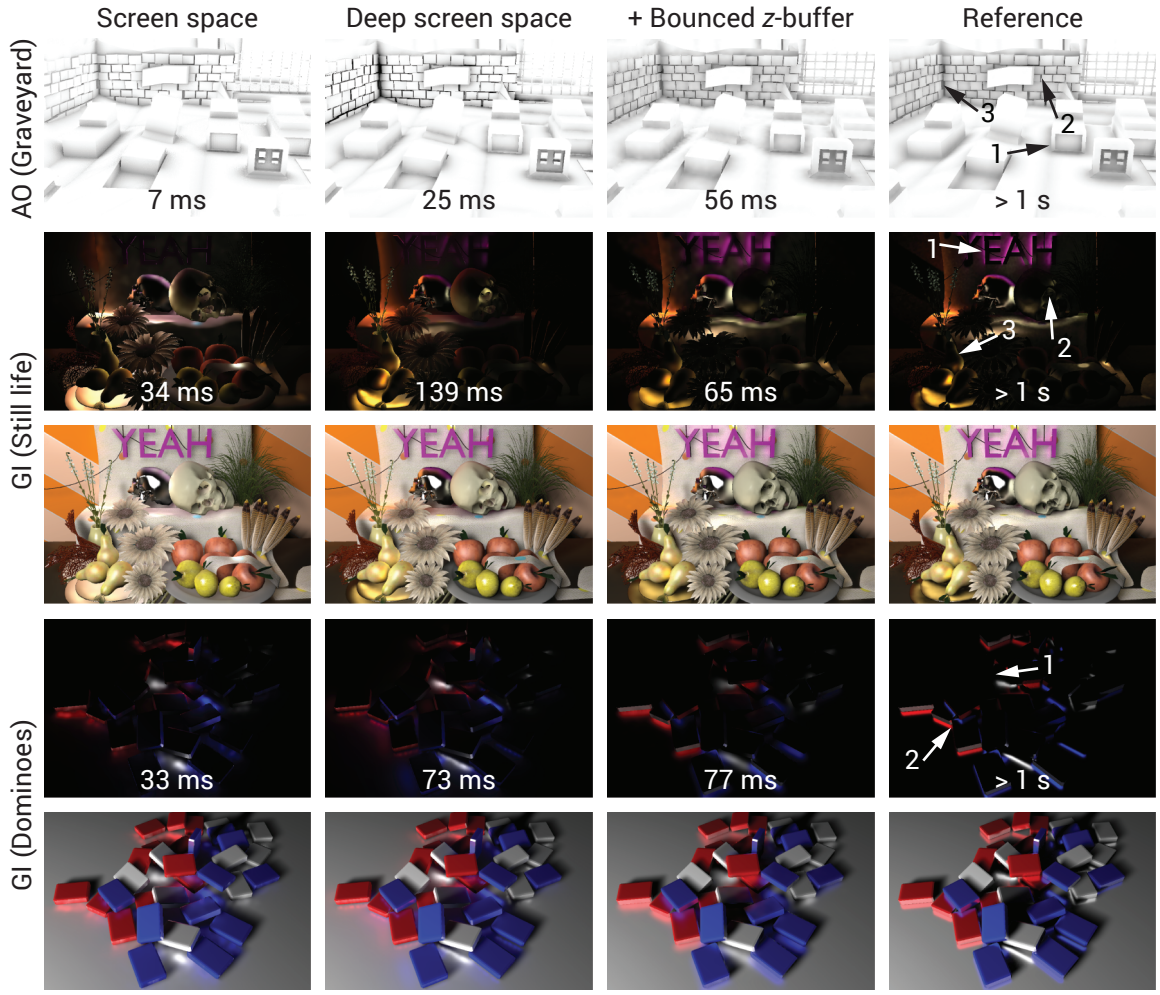


Figure 6.5: Comparison of our method (3rd col.) to screen space competitors (1st col.), the original shell splatting (2nd col.) and a reference (4th col.). The competing AO method used is horizon-based AO (HBAO) [Bavoil et al. 2008], for the indirect light it is screen space GI [Ritschel et al. 2009b].

Comparison to Previous Methods Figure 6.5 provides a qualitative evaluation of our adapted method (third column) versus the original deep screen space without indirect visibility (second column), a corresponding screen space method (first column) as well as a reference based on ray-tracing (last column).

For AO, we chose horizon-based AO (HBAO) [Bavoil et al. 2008] as the screen space method, which is state of the art in quality. For GI, we used screen space GI [Ritschel et al. 2009b]. We applied the same geometry-aware blur to both, our method and the screen space reference. The original DSS result is blurred per level as in Section 4.2.3. All results were obtained for a resolution of 768 × 512 px on an Nvidia GTX 770 GPU. We always used a 4 samples per pixel for our new method where each sample corresponds to a unique direction.

The first row in Figure 6.5 shows AO for a graveyard scene (122 k triangles, 375 k surfels). Albeit being the fastest of the approaches we compare, HBAO fails to reproduce darkening due to triangles which are not visible or seen under grazing angles. This leads to a quite different 3D perception of the scene (e. g., arrow 1). Also the subtle shadowing of the brick wall due to differently protruding bricks (2), which is visible in the reference, is missing. Deep screen space

Stage	Graveyard	Still Life	Dominoes
Tessellation	8.3 ms	8.8 ms	6.5 ms
Shuffling	4.0 ms	4.5 ms	4.5 ms
Splatting	38.5 ms	46.8 ms	61.2 ms
Un-shuffling	1.3 ms	1.5 ms	1.5 ms
Reduction	0.3 ms	0.3 ms	0.3 ms
Blurring	3.5 ms	3.2 ms	3.0 ms
Total	55.8 ms	65.1 ms	77 ms

Table 6.1: Computation time for different stages and effects.

reproduces darkening from invisible geometry but over-occlusion such as in the corner of the brick wall (3) is a problem. This makes scaling of the effect necessary which in turn reduces more subtle shadowing too much. Our method resembles the reference most and suffers the least from under- and over-occlusion. The most noticeable difference to the reference is noise due to the low number of sampled directions per pixel. Yet, unlike the reference, ours is interactive.

The second and third row in Figure 6.5 show indirect light and the combination with direct light, respectively, for a still life scene with glossy as well as diffuse objects (202 k triangles, 167 k surfels). Screen space GI misses both, indirect light, e. g., from the letters in the background (1), and indirect occluders, e. g., inside of the skull (2) which is mistakenly lit by the cloth below. Again, the screen space method is fastest in comparison. Deep screen space deals with the light missing for the screen space method but the lack of indirect visibility is even worse as can be seen inside the skull (2) or on the pears (3) to the left. Our adapted deep screen space again is closest to the reference with the same artifacts as for AO.

In the last two rows of Figure 6.5 we show a scene containing many colorful dominoes on a highly specular surface (1.4 k triangles, 237 k surfels). Since many faces of the dominoes are not visible or cover only little screen area, the amount of available indirect light samples in screen space varies strongly, resulting in perceivable noise. Due to the lack of indirect visibility, the deep screen space result overestimates indirect light (e. g., 1) or mixes light from differently colored emitters in some places, leading to purple areas not found in the reference (2). The z-buffered deep screen space is again closest to the reference, even though we cannot achieve the same level of specularity due to the blurring which is necessary to reduce noise from sub-sampling.

Time Complexity Timings of the individual stages are given in Table 6.1. As the surfelization is unchanged in comparison to the unmodified deep screen space, the same analysis applies (Section 4.4). The most expensive step is again the splatting stage. Since all splats have the same size on screen, the runtime depends on the number of visible splats in the first place, i. e., the number of surfels that cannot be culled times the number of levels in the framebuffer. Again, since the surfels are constructed to have the same projected size, the number of surfels basically only depends on the depth complexity of the visible part of the scene.

Memory Requirements Table 6.2 summarizes the memory consumption of our method. Most of the memory is necessary to store the framebuffer layout, i. e., the lookup textures used for shuffling and unshuffling. Next is the hierarchical deferred shading information itself. The only difference between AO and GI is that for AO the effect can be computed using float textures while GI needs 3D vectors.

Data	Memory requirement
Min / max MIP map	6 MB
Framebuffer layout	24 MB
Shuffled positions	18 MB
Shuffled bounce directions	18 MB
Layered framebuffer	6 MB (AO) / 18 MB (GI)
Total	72 MB (AO) / 84 MB (GI)

Table 6.2: Memory consumption of our method, for different shading effects and an output resolution of 768 × 512 px.

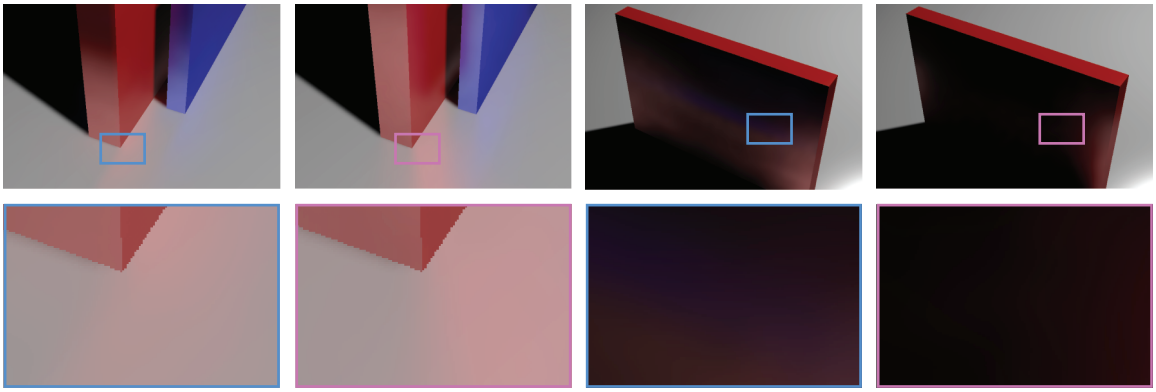


Figure 6.6: Effect of ϵ on precision for far-range shading.

Problems and Limitations In shell splatting, many small surfels are approximated by fewer enlarged ones for the far range, which implicitly assumes that the enlarged version of the picked surfel is representative for surrounding surfels. This assumption breaks when the surface normal changes rapidly in the neighborhood of the selected surfel, for example at convex or concave corners and becomes observable for large values of ϵ which regulates the trade-off between the amount of sub-sampling and runtime (Chapter 4) as in Figure 6.6 (top left). Similarly, we may observe artifacts resulting from enlarged surfels protruding through other geometry and becoming wrongly visible to a pixel (Figure 6.6, top right).

Another (typical) problem is temporal coherency. Not only are the sampling directions randomly selected for each frame but also does the subset of the surfel cloud which is sampled at each pixel change, leading to flickering in animated scenes. We currently apply simple re-projection [Scherzer et al. 2012] for animated scenes (but not in any figure in this thesis). Further research could address spatio-temporal filtering of the bounced z-buffer itself.

6.5 Comparison to Previous Work

Regarding the z-buffered deep screen space introduced in Section 6.3, the general discussion of the original method in Section 4.5 also applies to the z-buffered variant, with the exception that the latter adds support for indirect visibility. This is different from most screen space methods, with the exception of the image-space blocker accumulation by Sloan et al. [2007]: Their approach tracks a spherical harmonics approximation of visibility in screen space that is updated

with spherical harmonics (SH) exponentiated values. It is limited to diffuse or moderately glossy scenes and requires to store a number of SH coefficients depending on the lighting and shadow frequencies. Our approach adapts to all ranges of materials from diffuse to mirror-like.

Instant Radosity [Keller 1997] accounts for visibility but does not scale well to large scenes as VPLs are also placed where they do not contribute to the framebuffer. Similarly, shadow maps for each VPL, even when accelerated [Ritschel et al. 2008], process geometry that does not contribute. At the same time, the level of detail resolved is limited by the shadow maps' resolution and low compared to screen space shading.

Some point-based global illumination methods splat approximations of scene geometry into buffers to resolve indirect visibility [Christensen 2008; Ritschel et al. 2009a], too. Differently, we do not resolve visibility of all directions from a scattered set of sparse points, but from sparse scattered positions into sparse scattered directions. We also avoid finding a surfel or light cut and use approximate level-of-detail in the 2D framebuffer instead. Our approach can be seen as an irregular decoupling of a hemicube [Cohen and Greenberg 1985] or micro-buffers [Ritschel et al. 2009a] into a map of depth-resolved samples of indirect radiance. This light field is the same as the one used by Lehtinen et al. [2012] where sophisticated reconstruction is applied to a low number of indirect light samples from an initial path-tracing pass. In contrast, our method produces such samples by means of splatting. We then use simple cross-bilateral blur without re-projection for reconstructions in milliseconds. Executing their improved reconstruction on our result would produce even better results, but at much higher reconstruction times (seconds to minutes).

Mattausch et al. [2015] exploit coherence in ray position and direction to efficiently cull groups of rays and primitives in a hierarchy. To this end, the distance from the the nearest primitive is stored along each ray, which is the same data structure we use. Our work avoids creating any hierarchy, both for occluders and occluding primitives, and does not even require the geometry to fit into GPU memory at once.

6.6 Conclusion and Future Work

We have devised a conceptually simple and computationally efficient method to add visibility to splatting-based approaches for global illumination rendering.

Our method has so far only considered the case of opaque surfaces. A natural extension would be to also allow for transparent senders, e. g., in z-buffered deep screen space. Similar to order-independent transparency [McGuire and Bavoil 2013], multiple surfels could be stored per gathering direction. To some extent, we already do this by storing one candidate per surrounding shell. Compositing then could use information about all stored surfels and blend them in a meaningful way, taking the individual translucency into account.

So far, we use information about the receiver surfaces for importance sampling of individual directions. For glossy receivers, focusing on a particular direction makes sense as the surface will mainly reflect objects from a small cone of possible directions. For diffuse receivers however, a wide range of incident directions are relevant while it becomes less important which particular direction is sampled. This observation indicates that sampling cones according to the BRDF at each receiver pixel might be beneficial and allow more precise but sparser sampling for glossy receivers and less precise but denser sampling for diffuse ones. Associating each pixel not with a

direction but a certain opening angle and taking the distance information to the found sample into account could also allow for improved filtering.

Finally, to improve temporal coherence, approaches for spatio-temporal filtering the bounced z-buffer itself — and not just of the final result — could be investigated.

7

Deep Shading

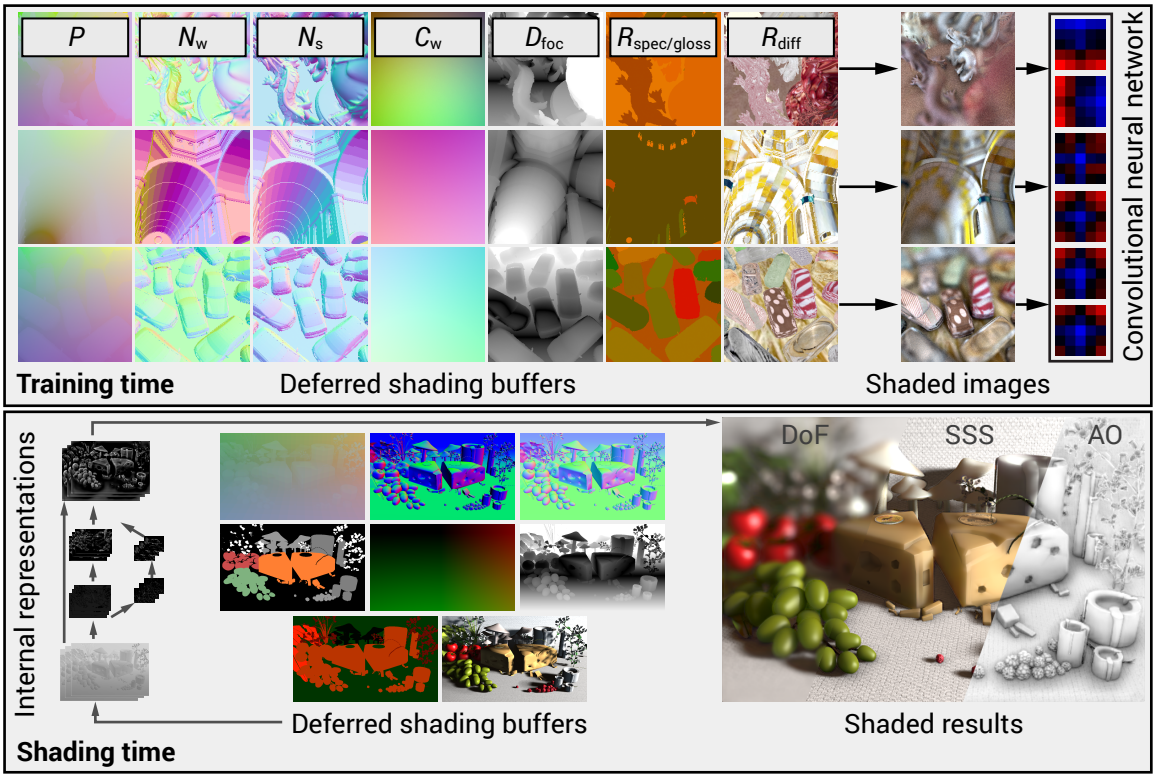


Figure 7.1: In training (top), we learn a mapping from deferred shading buffers containing positions, normals, etc. to appearance using a convolutional neural network (CNN). At run-time (bottom), the CNN is used to reproduce effects such as depth-of-field, sub-surface scattering or ambient occlusion at interactive rates (768 × 512 px, 1 ms rasterizing attributes, 27 / 27 / 9 ms network execution).

7.1 Introduction

In computer vision, convolutional neural networks (CNNs) have recently achieved new levels of performance for several inverse problems where RGB pixel appearance has to be mapped to

attributes such as positions, normals or reflectance. Deep network architectures have opened up avenues for several novel applications. In computer graphics, in turn, mapping per-pixel attributes such as positions, normals or reflectance of a virtual 3D scene to RGB pixel appearance is known as screen-space shading. A wide range of screen-space methods (Section 3.6.3) has recently enabled the computation of effects like ambient occlusion, indirect light, scattering, depth-of-field, motion blur, or anti-aliasing. In this chapter, we investigate solving this second problem of appearance synthesis — which has previously only been tackled by analytic approximations — by employing machine learning, too, making use of deep CNN architectures. We call this approach *Deep Shading*.

The benefit of Deep Shading is that quality and performance similar to human-written shaders can be achieved, only by learning from exemplary data which may be rendered or even come from actual photographs. This avoids human effort in programming and ultimately also allows for deep "multi-shaders" that combine previously separate screen-space effects in one single CNN.

After introducing the necessary machine learning terminology in Section 7.2, we will turn to the problem of how to generate a meaningful corpus of training data and introduce our CNN architecture in Section 7.3. We analyze how well the approach works for different shading effects (Section 7.4) and how its success depends on different aspects related to the training data and network architecture used (Section 7.5). Finally, we close with a comparison to relevant previous work in both computer graphics and computer vision in Section 7.6.

7.2 Background

Here we briefly summarize some aspects of machine learning, neural networks, deep learning and training of convolutional neural networks, to the extent necessary for immediate application to the computer graphics problem of shading.

7.2.1 Problem Formulation

For our purposes, it suffices to view (supervised) learning as fitting a sufficiently complex and high-dimensional function $\tilde{f}$ to data samples generated by an underlying function f in a robust way. The function f may be extremely abstract, very expensive to evaluate, or its implementation unknown, so it cannot be computed directly. For example, f may label objects seen in an image, return a flawless image for one corrupted by noise, or synthesize speech from written text.

We are given n function values $f(\mathbf{x}_i)$ for n exemplary inputs $\mathbf{x}_i$ from a large, potentially high-dimensional and infinite set of possible inputs X. From these, the behavior of f is to be deduced and emulated by $\tilde{f}$. In particular, f should be approximated well also for new inputs from X for which the function value has not been observed previously, achieving a generalization.

In our case, the function f is computing a particular shading effect, such as an ambient occlusion term or the indirect light in a scene. The domain of f consists of deferred shading buffers of a given (spatial) resolution which contain per-pixel attributes such as position, normal and material parameters, while the range of f covers RGB images of the same resolution. The exemplary values $f(\mathbf{x}_i)$ can be produced in arbitrary quantity, for example by path tracing or other image synthesis algorithms as well as be recorded using RGBD cameras. In this setting, f is

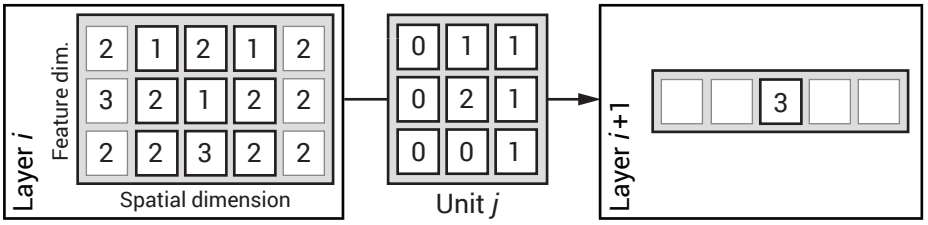


Figure 7.2: CNN convolution in the simplified case of one spatial dimension. At layer i (left), the data has a spatial extent of five (horizontal) and three features (vertical). Unit j (middle) has kernel width three in the spatial dimension and takes all features present at layer i as input. The fan-in is highlighted by a black outline. Right: unit j produces one feature at one spatial location in layer $i+1$.

technically not even a function as the deferred shading buffers only contain partial information about the actual underlying three-dimensional scene. In general, this means that there are arbitrary many valid RGB image outputs corresponding to the same input. Nevertheless, the approximation $\tilde{f}$ can try to produce the most common output for each case.

7.2.2 (Convolutional) Neural Networks

Neural networks (NNs) are a way of defining powerful non-linear approximations $\tilde{f}$. A neural network is typically comprised of computational *units* or *neurons* mapping multi-dimensional inputs to scalar outputs. Computationally, the neuron k applies a non-linear function a called *activation function* to some affine combination of their input $\mathbf{x}$ which is governed by a vector of *weights* $\mathbf{w}_k$:

$$k(\mathbf{x}) = a\left(\begin{bmatrix}\mathbf{x}^\top & 1\end{bmatrix}\mathbf{w}_k\right).$$

The weights used by the neurons constitute the parameters of the final function $\tilde{f}$. Defining $\mathbf{w}$ as the set of weights for the entire network we can express the resulting function given by the combination of network architecture and weights $\mathbf{w}$ as $\tilde{f}_{\mathbf{w}}$.

A common type of activation function are *Rectified Linear Units* (ReLUs) which are defined by

$$a(x) = \max(0, x).$$

A variant of ReLUs are leaky ReLUs:

$$a(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \gamma \cdot x, & \text{otherwise,} \end{cases}$$

where γ is a small positive constant, e. g., $\gamma = 0.01$.

Multiple units are typically arranged in a hierarchical fashion and grouped into *layers*, with the outputs of one layer serving as inputs to following layers. There are usually no connections between units of the same layer. The *fan-in*, i. e., the vector of inputs of each unit, can either cover all outputs of the previous layer (*fully-connected*) or only a few. Units may also connect to several preceding layers in the hierarchy.

Convolution layers are a particular type of layer defining a regular spatial arrangement of the units: units are arranged into multiple regular and same-sized grid slices. Each unit in the convolution layer $i+1$ connects to the units of all slices of layer i within a certain local spatial extent centered at the respective unit as depicted in Figure 7.2. All units of a slice share their

weights so that the computation happening for each slice can be seen as a 3D convolution with a *kernel* which is as "high and wide" as the spatial fan-in of the units and as "deep" as the number of slices in the previous layer. We will refer to the *spatial kernel size* simply as *kernel size*.

Pooling layers are another important type of layer. Their distinctive feature is that they perform a reduction in spatial resolution by reducing (spatial) neighborhoods to single scalar values, for example by selecting the maximum value of the neighborhood (max pooling) or by computing the average (average pooling). Pooling may also proceed by weighting each element of the neighborhood with a certain learned factor.

Multiple such convolution and pooling layers can be stacked to form deep convolutional neural networks. Due to the hierarchical structure, values computed by units of the final layer of the network depend on a large number of elements of the original input $\mathbf{x}$. This set of elements, which usually corresponds to a spatial neighborhood of a certain size, is also called the *receptive field* of the network.

Pooling layers are complemented by *de-convolutional* or *up-sampling layers*. There, a single scalar value is blown up to a complete neighborhood by multiplying it with a pre-defined or learned stencil. In this way, the resolution of intermediate representations inside a network can be increased again [Long et al. 2015] which is critical for our task where we want to produce per-pixel appearance in the original resolution.

7.2.3 Network Training

The process of optimizing the network weights $\mathbf{w}$ such that $\tilde{f}_{\mathbf{w}}$ actually approximates f is called *training*. For this, first, an appropriate *loss* function has to be defined which measures the deviation of the network outputs $\tilde{f}_{\mathbf{w}}(\mathbf{x}_i)$ from $f(\mathbf{x}_i)$. A common choice of loss function is the squared $\mathscr{L}_2$-norm: $\left\|\tilde{f}_{\mathbf{w}}(\mathbf{x}_i) - f(\mathbf{x}_i)\right\|_2^2$. Whenever the domain of f corresponds to image appearance we may also use a more perceptual loss function based on the structural similarity (SSIM) index [Zhao et al. 2017] in addition or as an alternative.

Training tries to minimize the loss for the given set of training data. As the function defined by a deep neural network is highly complex and non-linear, only approximate optimization is feasible. In practice, a variant of mini-batch *stochastic gradient descent* (SGD) is employed. For each exemplar of a small batch of the sample data, the gradient with respect to the parameters $\mathbf{w}$ is determined and then averaged across the batch. An update to $\mathbf{w}$ is computed by multiplying the averaged gradient with a small factor α and subtracting it from $\mathbf{w}$. Different learning methods mostly differ in their strategy for choosing the value for α over time so that optimization is neither over-shooting the local optimum, nor approaching it too slowly.

A small loss for the inputs of the training data set does not guarantee a good generalization on unseen data. If either $\tilde{f}$ depends on too many parameters or the size of the training set is not significant, *over-fitting* can occur. In this case, optimization erroneously trades in generality for a slightly smaller loss on the training data.

To detect and prevent over-fitting, it is recommendable to keep a small portion of the available training data, e. g., 10%, as *validation set*. The validation set is not used in the optimization but the loss $\tilde{f}$ achieves on it is monitored. As training and validation set come from the same distribution of sample data, $\tilde{f}$ is expected to achieve a comparable loss on both if it achieves generalization. Contrarily, diverging losses on the training and validation sets indicate over-fitting and that a reduction of the network complexity might be necessary.

7.3 Deep Shading

Here, we detail the training data we produced for our task, the network architecture proposed and the process of training it.

7.3.1 Data Generation

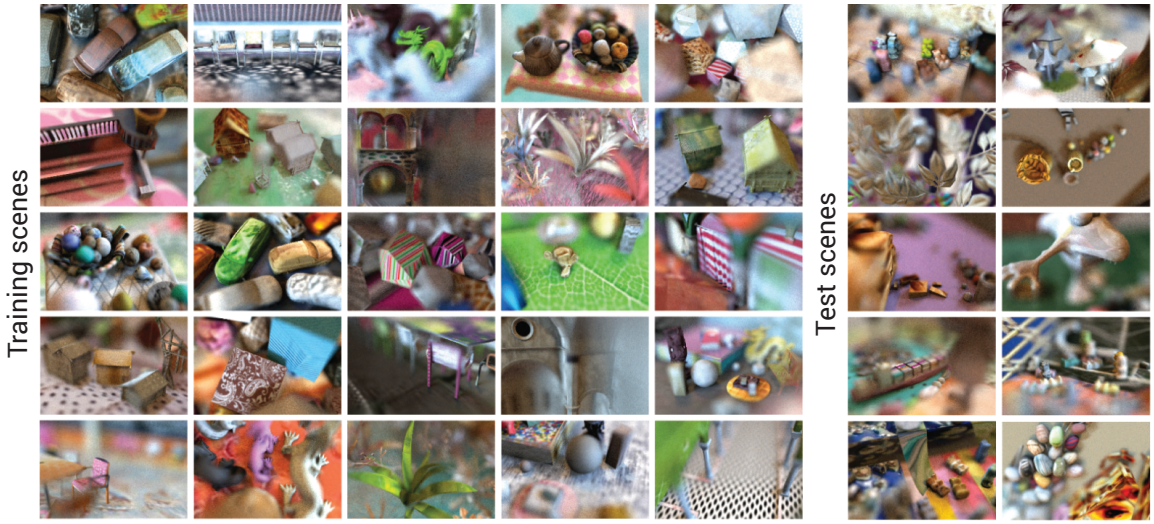


Figure 7.3: Random textures and lighting create a large variety of training and test images.

Structure of the Data Our data sets consist of 61,000 pairs of deferred shading buffers and corresponding shaded reference images which come in a resolution of 512 × 512 px for AO and 256 × 256 px for all other effects. Of these 61,000 pairs, we use 54,000 images to train the network, 6,000 for validation and the remaining 1,000 for testing (Section 7.5.2). Training and validation were sampled from the same set of ten scenes of different nature while the test images come from four other scenes not used in training or validation.

To generate the 60,000 training and validation images we first rendered 1,000 pairs of deferred shading inputs and appearance outputs for each of the set of ten scenes. These *base images* are then rotated (in steps of 90°) as well as flipped horizontally and vertically and the resulting new images are added to the data set to increase its robustness and size in an easy way. Special care has to be taken when transforming attributes stored in view space, here the respective positions and vectors have to be transformed themselves by applying appropriate rotations or mirroring. For the test set, we proceed analogously but using the distinct set of four scenes and appropriately less base images per scene. Generating one set, i. e., rendering and subsequent data augmentation, takes up to about 170 hours of computation on a single high-end GPU.

The base images all show unique and randomly sampled views of the respective scene seen through a perspective camera with a fixed field-of-view of 50°. View positions are sampled from a box fitted to the scenes' spatial extents. Section 7.4 contains additional information on the computation of training sets for each of the shading application we demonstrate. Figure 7.3 shows samples of typical ground truth images for a combination of image-based lighting and depth-of-field.

About half of our scenes are common scenes from the computer graphics community such

as Crytek Sponza or Sibenik Cathedral or are carefully modeled scenes from sources such as BlendSwap. The remaining scenes have been composed by ourselves using objects from publicly available sources to cover as many object categories as possible, e. g., vehicles, vegetation or food. Procedurally generated scenes would be another more sophisticated option to explore in future work.

Computation and Encoding of the Network Inputs (Attributes) We use OpenGL's rasterization pipeline to compute the deferred shading buffers. They contain per-pixel geometry, material and lighting information and we encode them as individual 16 bit float images to store them. Positions are stored in camera space (P_s) as for our purposes absolute world positions do not contain more information than the former and would encourage a network to memorize geometry. Normals are represented as unit vectors in Cartesian coordinates in both camera and world space (N_s and N_w). Additionally, depth ($D_s = P_{s,3}$), distance to the focal plane (D_{focal}) and a high-level parameter B corresponding to the radius of the circle of confusion of the lens system are provided to capture camera sensor-related parameters. To be able to compute view-dependent effects, the normalized direction to the camera (C_w) is an additional input.

Material parameters (R) combine surface and scattering properties. For surfaces, we use the set of parameters to the Phong [1975] reflection model, i. e., RGB diffuse and specular colors (denoted as R_{diff} and R_{spec}) as well as scalar glossiness (R_{gloss}). For scattering we use the model by Christensen [2015] which is parameterized by the length of the mean free path for each color channel (R_{scatt}). Direct light (denoted by L or L_{diff} for diffuse-only) is not computed by the network but provided as an input to it as is the case with all corresponding screen-space shaders we are aware of. Fortunately, for the types of light sources common in real-time rendering it can be quickly computed and fed into the network. Specifically, we again use the Phong reflection model and shadow maps for visibility.

Finally, to support motion blur, per-pixel object motion F is encoded as a 2D polar coordinate in each pixel assuming that the motion during exposure time is small enough to be approximated well by a translation. The first component holds the direction between 0 and π (motion blur is time-symmetric for time-symmetric shutter functions), the second component holds the distance in that direction.

In summary, each pixel of the network input is associated with a high-dimensional feature vector. Some of these input dimensions are redundant and correlated, e. g., normals are derivatives of positions and camera space differs from world space only by a linear transformation. Nonetheless, those attributes are the output of a typical deferred shading pass in a common interactive graphics application and can be produced within milliseconds from complex geometric models. Redundant attributes come at almost no additional cost but help to improve the performance of networks for certain effects. At the same time, for some effects that do not need certain labels they can be excluded from the respective network input increase speed.

Computation and Encoding of the Ground Truth Outputs (Appearance) The reference images store per-pixel RGB appearance and are produced from virtual scenes using rendering. More specifically, we use path tracing (Section 3.3) for AO, DO and IBL and sample multiple lens positions or points in time for depth-of-field and motion blur, respectively. For anti-aliasing, reference images are computed with super-sampling of 8×8 px relative to the label images. We use 64 samples per pixel (spp) to compute the AO training and validation data and 256

samples for the remaining effects. While this means that some Monte Carlo noise remains in the ground truth outputs, we found out that compute time is better invested into producing more individual images (Figure 7.16, left). The test sets however are rendered at higher sample counts to guarantee for noiseless images.

All per-object attributes which are later allowed to vary at run-time such as material parameters are sampled randomly for each training sample. For effects including depth-of-field and sub-surface scattering we found it beneficial to texture objects by randomly assigned textures from a large representative texture pool [Cimpoi et al. 2014] to increase the information content with respect to the underlying blurring operations. Automatic per-object box mapping is used to assign UV coordinates.

We do not apply any gamma curve or tone mapping to our reference images used in training. It therefore has to be applied as a post-process after executing the network.

In practice, some effects like AO and DO do not compute final appearance in terms of RGB radiance but rather a quantity which is later multiplied with albedo. We found networks that do not try to emulate this obvious multiplication to be substantially more efficient while also requiring less input data and therefore opt for a manual multiplication where possible. However, the networks for effects that go beyond this simple case need to include the albedo in their input and calculations. In Section 7.4 we will get back to where albedo is used in detail: Table 7.1 provides an overview in the column “albedo”.

In a similar vein, we have found that some effects are best trained for a single color channel while others need to be trained for all channels at the same time. In the first case, the same network is executed for all three input channels simultaneously using vector arithmetic after training it on scalar images showing only one of the color channels. In the second case, one network with different weights for the three channels is run. We refer to the first case as “mono” networks and to the latter as “RGB” networks to distinguish them (Table 7.1).

7.3.2 Network Structure

Our network is U-shaped, with a left and a right *branch* as depicted in the top part of Figure 7.4. The first (left) branch is reducing spatial resolution (*down* branch) and the second (right) branch is increasing it again (*up* branch). We refer to the layers producing outputs of one resolution as a *level*. The bottom of Figure 7.4 shows one such level in detail. Overall, we used up to six of such levels. The corresponding per-level resolutions ranged from 512×512 px to 16×16 px.

We refer to the layers of a particular level and branch (i. e., left or right) as a *step*. Each step is comprised of a convolution and a subsequent activation layer. The convolutions (blue in Figure 7.4) have a fixed extent in the spatial domain which is the same for all convolutions throughout one instance of the network architecture, but which may vary between the network instances for different screen-space effects.

We gain a significant speedup by using convolution in groups with 2^n groups on level n. This means that both input and output channels of a convolution layer are grouped into 2^n same-sized blocks where outputs from the m-th block of output channels may only use values from the m-th block of input channels. The consecutive activation layers (orange in Figure 7.4) consist of *leaky ReLUs* as described by Maas et al. [2013] which multiply negative values by a small constant instead of zero as done for regular ReLUs (Section 7.2.2) which helps to accelerate training of the resulting network.

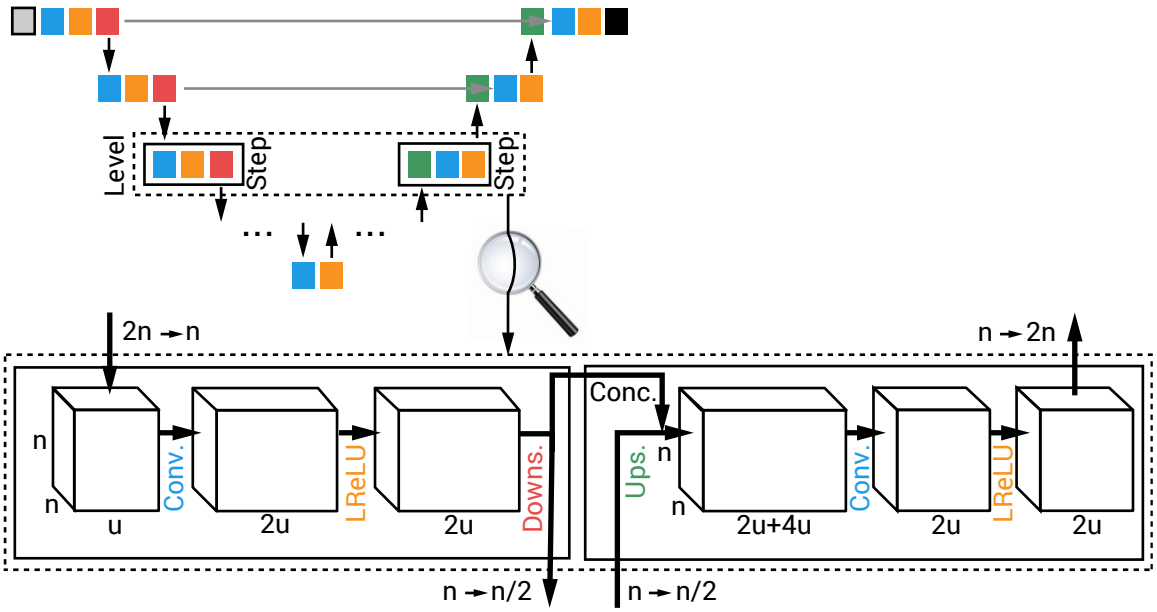


Figure 7.4: Top: The big picture with one branch going down and another going up again in a U-shape. Bottom: One level of our network. Boxes represent in- and outputs of the layers, the arrows correspond to the operations performed by the respective layers. The spatial resolution is denoted by multiples of n, the number of channels by multiples of u. The convolution groups are not emphasized for simplicity.

The change in resolution between two steps on different levels is performed by re-sampling layers. These are realized by 2×2 px mean-pooling on the down (red in Figure 7.4) and by bilinear up-sampling (green in Figure 7.4) on the up branch.

The layout of this network is the same for all our effects but the precise number of kernels on each level, their size and the number of levels vary. All designs have in common that the number of kernels increases by a factor of two on the down part to decrease by the same factor again on the up part. Denoting the number of kernels used on the first level (level 0) by u_0 this results in $u_0 \cdot 2^l$ output features being produced on level l. A typical start value is $u_0 = 16$ resulting in a 256-dimensional feature vector for every pixel in the coarsest resolution for the frequent case of 5 levels. The coarsest level consists of only one step, i. e., one convolution and one activation layer, as depicted in Figure 7.4 (top). Additionally, the convolution steps in the up-branch access the outputs of the corresponding step of the same output resolution in the down part (gray arrows in Figure 7.4). This allows to retain fine spatial details.

A typical network has about 130,000 learnable parameters which are comprised of the weights and bias terms (Table 7.1). We call the CNN resulting from training on a specific input and specific labels a *Deep Shader*.

7.3.3 Implementation Notes

Training

To implement and train our networks, we used the popular open-source neural network framework Caffe [Jia et al. 2014]. The network inputs are formed by loading the respective attributes

from sets of image files and concatenating the individual channels. This way, we form input vectors with 3 to 18 components per pixel. A critical aspect of training a deep neural network is choosing the right learning rate to achieve fast convergence without the risk of diverging computations. To facilitate learning of networks of varying complexity without the need of hyper-parameter optimization we use an adaptive learning rate method (ADADELTA [Zeiler 2012]) with a *momentum* of 0.9 which selects the learning rate autonomously.

We use a loss function based on the structural similarity (SSIM) index [Zhao et al. 2017] which compares two image patches in a perceptually motivated way and which we found to work best for our task after evaluating various options (Section 7.5.3). The loss between the output of the network and the ground truth is determined by tiling the two images into 8×8 px patches and combining the SSIM values computed between corresponding patches for each channel. SSIM ranges from -1 to 1 where higher values indicate higher similarity. Structural dissimilarity (DSSIM) is defined as $(1 - SSIM)/2$ and used as the final loss.

Testing

A trained CNN should ideally generalize to new inputs not seen in training. To determine whether this is the case, we compute a test error as the average loss over our test sets (Section 7.3.1) which, as mentioned previously, only feature 3D scenes not used during training. The resulting SSIM values for different Deep Shaders are listed in Table 7.1.

Run-time Network Evaluation

While Caffe is useful for training the network, it is inconvenient for use inside an interactive application. For example, the g-buffers produced by OpenGL have a different memory layout and would first need to be transformed before being able to execute the network. Thus, instead of integrating Caffe into a rendering framework we re-implemented the forward pass of the CNN using OpenGL shaders operating on array textures. OpenGL also enables hardware-supported up- and down-sampling as well as to drop actual concatenation layers by simply accessing two layered inputs instead of one when performing convolutions.

7.4 Results

We will now analyze trained Deep Shaders for different shading effects. Table 7.1 provides an overview of their input attributes, structural properties and resulting SSIM achieved on the test sets. Additionally, the time needed to execute the network using our OpenGL implementation on an Nvidia GeForce GTX 1070 GPU is given. For visual comparison, we show examples of Deep Shaders applied to new (non-training) scenes compared to the reference implementations used to produce the training sets in Figure 7.5–7.8.

Some shading effects like IBL or DO depend on an input environment map which is accessed globally. As our CNN structure, however, only performs local operations we cannot expect it to handle arbitrary environment maps e. g., given as an additional input slice. Yet, it is possible to train networks for such shading effects with training images computed using a specific environment map. In this case, the network will implicitly "bake" the lighting information into the weights it learns. We still consider this useful in the same way as static environment maps are used in practice. Generalizing over different environment maps by using the latter as additional network input in a differently structured CNN remains future work.

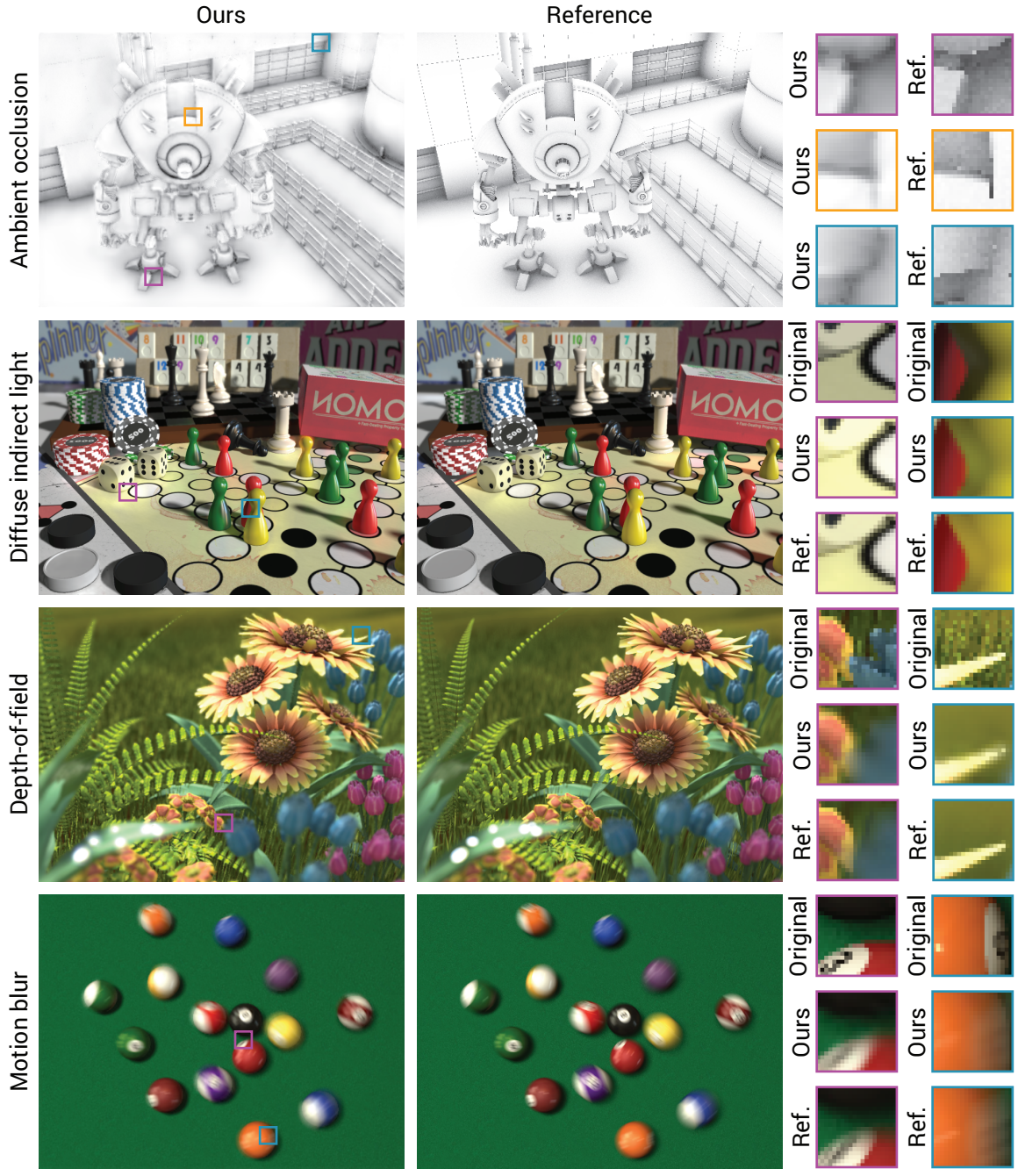


Figure 7.5: Results for the ambient occlusion, indirect lighting, depth-of-field and motion blur Deep Shaders. 'Original" shows the scenes before applying the respective effects.

Effect	Attributes	Albedo	Mono	u_0	Levels	Kernel	Size	SSIM	Time
AO	N_s, P_s	✗	✓	8	6	3 × 3 px	71 k	.805	9 ms
GI	N_s, P_s, L_{diff}	✓	✓	16	5	3 × 3 px	134 k	.798	28 ms
DoF	$D_{\text{focal}} \cdot B$, D_s, L	✓	✓	16	5	3 × 3 px	133 k	.959	27 ms
MB	F, L, D_s	✓	✓	16	5	3 × 3 px	133 k	.937	26 ms
SSS	P_s, R_{scatt}, L	✓	✓	16	5	3 × 3 px	133 k	.905	27 ms
AA	D_s, L	✓	✓	8	1	5 × 5 px	1.2 k	.982	1.8 ms
Multi	(see AO, DoF)	✓	✗	16	5	3 × 3 px	135 k	.933	26 ms
IBL	N_w, C_w, R	✓	✗	300	1	1 × 1 px	3.9 k	.973	21 ms
DO	N_w, N_s, P_s	✗	✗	16	5	3 × 3 px	135 k	.589	26 ms
RS	N_s	✓	✗	16	5	5 × 5 px	370 k	.622	80 ms

Table 7.1: Structural properties of the networks for different effects, resulting degrees of freedom, SSIM on the test set and time for executing the network using our OpenGL implementation on 768 × 512 px inputs. In case of mono networks, the time refers to the simultaneous execution of the network for all three color channels. The SSIM is always with respect to the raw output of the network, e. g., indirect irradiance for GI. The final image might show even higher SSIM.

Ambient Occlusion Ambient occlusion, a prototypical screen-space effect, simulates darkening in corners and creases due to a high number of blocked light paths and is typically defined as the percentage of directions in the hemisphere around the surface normal at a point which are not blocked within a certain distance. Our ground truth images are computed using ray-tracing with a constant effect range defined in world space units. In an actual application, the AO term is multiplied with the ambient lighting term before adding it to the image.

The CNN faithfully reproduces darkening in areas with nearby geometry (Figure 7.5, first row), the most noticeable difference to the reference being blurrier fine details. To evaluate how well our learned shader performs in comparison to optimized screen-space AO techniques, in Figure 7.6, we show a same-time comparison to Horizon-based Ambient Occlusion (HBAO) [Bavoil et al. 2008] which is an efficient technique used in games. As our method does not have any parameters — apart from the network structure itself — that can be tweaked, we adjust HBAO to same computation time by choosing the same effect radius, and using 24 sampling steps into 16 sampling directions. On the test set, we achieve a higher SSIM than HBAO which we consider remarkable given that our method has been learned by a machine. Furthermore, our method does not exhibit the high frequency banding artifacts which are typical for HBAO (Figure 7.6, insets on top row) and creates less “cut off” indirect shadows where the screen space information is insufficient (Figure 7.6, bottom insets) as the CNN was trained on unbiased data. We made AO the subject of further in-depth analysis of alternative network designs described in Section 7.5 and seen in Figure 7.14 a+b.

Diffuse Indirect Light A common challenge in rasterization-based real-time rendering is indirect lighting. To simplify the problem, the set of relevant light paths is often reduced to a single indirect bounce, diffuse reflection (Section 2.5.1) and restricted to a certain radius of influence. The ground truth in our case consists of the “indirect radiance”, i. e., the light arriving at each pixel after one interaction with a surface in the scene. From this, the final indirect component can be computed by multiplying with the diffuse color. We compute our ground truth images in screen-space using SSGI [Ritschel et al. 2009b]. The position of the light source is sampled uniformly at random per image. As we are assuming diffuse reflections, the direct light input to the network is computed using only the diffuse reflectance of the material. In the

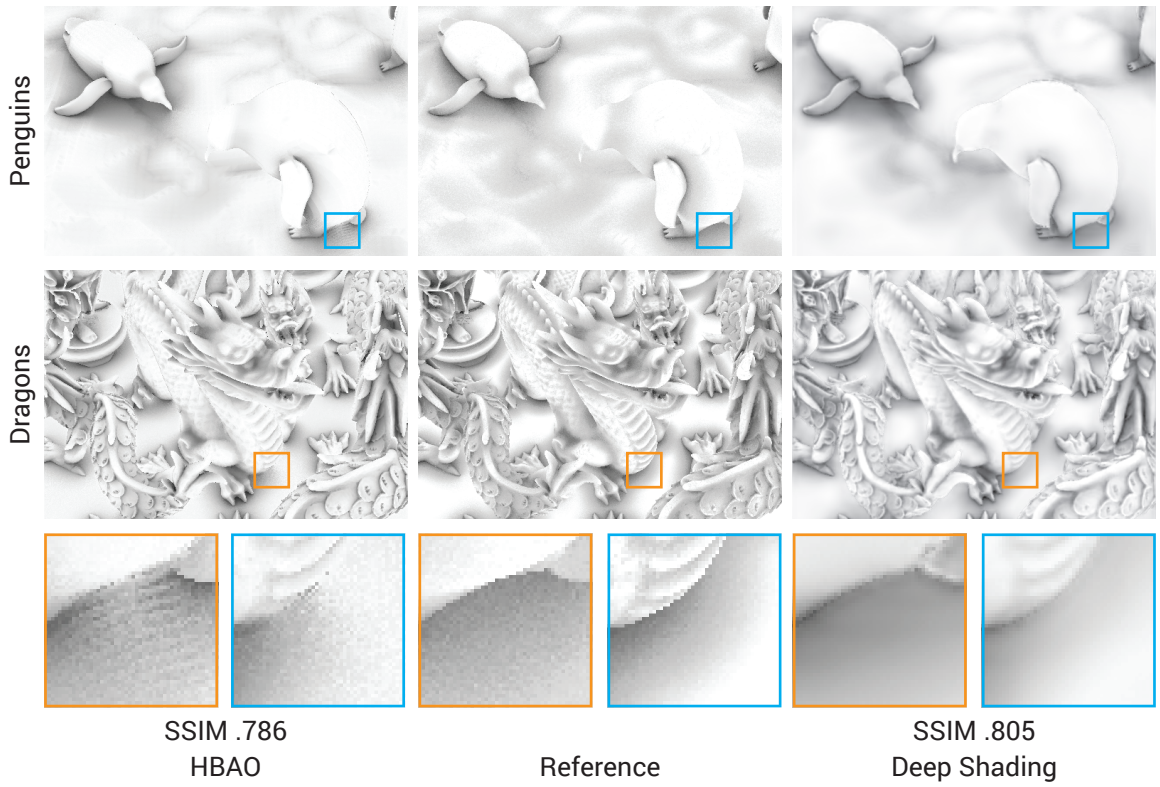


Figure 7.6: In a same-time comparison, Deep Shading for AO is on-par with state-of-the-art methods like HBAO, both numerically and visually. The given SSIM values are w.r.t. the full AO test set.

absence of advanced effects like fluorescence or dispersion, the light transport in different color channels is independent from each other. We therefore apply a monochromatic network. The network successfully learns to brighten areas in shadow applying the color of nearby lit objects (Figure 7.5, second row).

Depth-of-field As a simple rasterization pass can only simulate a pinhole camera, the appearance of a shallow depth of field (DoF) has to be faked by post-processing when multiple rendering passes are too costly. In interactive applications this is typically done by adaptive blurring of the sharp pinhole-camera image. We learn our own depth of field blur from sample data which we generate in an unbiased way by averaging renderings from multiple positions on the virtual camera lens. The amount of blurriness depends on the distance of each point to the focal plane as well as on the circle of confusion. Both parameters are sampled randomly during data generation and then multiplied ($D_s \cdot B$) to form a single blurriness attribute which is fed to the network and permits direct manipulation of the DoF shallowness at run-time. To allow for depth order-dependent behavior of the shader we also provide plain image depth. The Deep Shader again is trained independently for each channel assuming a non-dispersive lens. The trained network blurs things in increasing distance from the focal plane by increasing extents. In the third row of Figure 7.5, the sunflowers appear sharper than the grass in the background or the leaves in front.

Motion Blur Motion blur is the analog to depth of field in the temporal domain. Images of objects moving with respect to the camera appear to be blurred along the motion trajectories of the objects for non-infinitesimal exposure times. The direction and strength of the blur depends on the speed of the object in the image plane [McGuire et al. 2012]. For training, we randomly move objects inside the scene for random distances. Motions are restricted to those which are parallel to the image plane, so that the motion can be encoded by an angle and magnitude alone. We also provide the Deep Shader with a depth image to allow it to account for occlusion relations between different objects correctly, if possible. Our Deep Shader performs motion blur in a convincing way that manages to convey a sense of movement and comes close to the reference image (Figure 7.5, bottom row).

Sub-surface Scattering Simulating the scattering of light inside an object is crucial for achieving realistic appearance for translucent materials like wax and skin. A popular approximation to this is screen-space sub-surface scattering (SSSS) [Jimenez et al. 2009] which essentially applies a spatially-varying blurring kernel to the different color channels of the image. We produce training data at every pixel by iterating over all other pixels and applying Pixar's scattering profile [Christensen 2015] depending on the distance between the 3D position at the two pixels. After training the Deep Shader independently for all RGB channels on randomly textured training images with random parameters to the blurring profile we achieve images which transport the same sense of translucency as the reference method (Figure 7.7, top row).

Anti-aliasing While aliasing on textures can be reduced by applying proper pre-filtering, this is not possible for sharp features produced by the geometry of a scene itself. Classic approaches compute several samples of radiance per pixel which typically comes with a linear increase in computation time. This is why state-of-the-art applications like computer games offer simple post-processing filters like fast approximate anti-aliasing (FXAA) [Lottes 2011] as an alternative, which operate on the original image and auxiliary information such as depth values. We let our network learn such a filter on its own, independently for each channel.

Applying our network to an aliased image (Figure 7.7, second row) replaces jagged edges by smooth ones. While it cannot be expected to reach the same performance as the 8×8 px multi-sampled anti-aliasing (MSAA) we use for our reference which can draw from orders of magnitude of additional information, the post-processed image shows fewer disturbing artifacts. At the same time the network learns to not over-blur interior texture areas that are properly sampled but only blurs along depth discontinuities.

Image-based Lighting In image-based lighting a scene is shaded by sampling directions in an environment map to determine incoming radiance, assuming the latter is unblocked. The network is trained to render IBL based on diffuse and specular colors as well as gloss strengths using the Phong shading model which also depends on the surface normal and camera direction. As a special case, for IBL we also apply grouped convolution using three groups on the final convolution layer, effectively using one group per color channel. In an application the IBL is typically added to shading from a small number of main light sources.

As can be seen from the vehicles in the third row of Figure 7.7, the network handles different material colors and levels of glossiness. The two main limitations are a slight color shift compared to the reference as seen in the shadowed areas of the bikes' tires and an upper bound on the

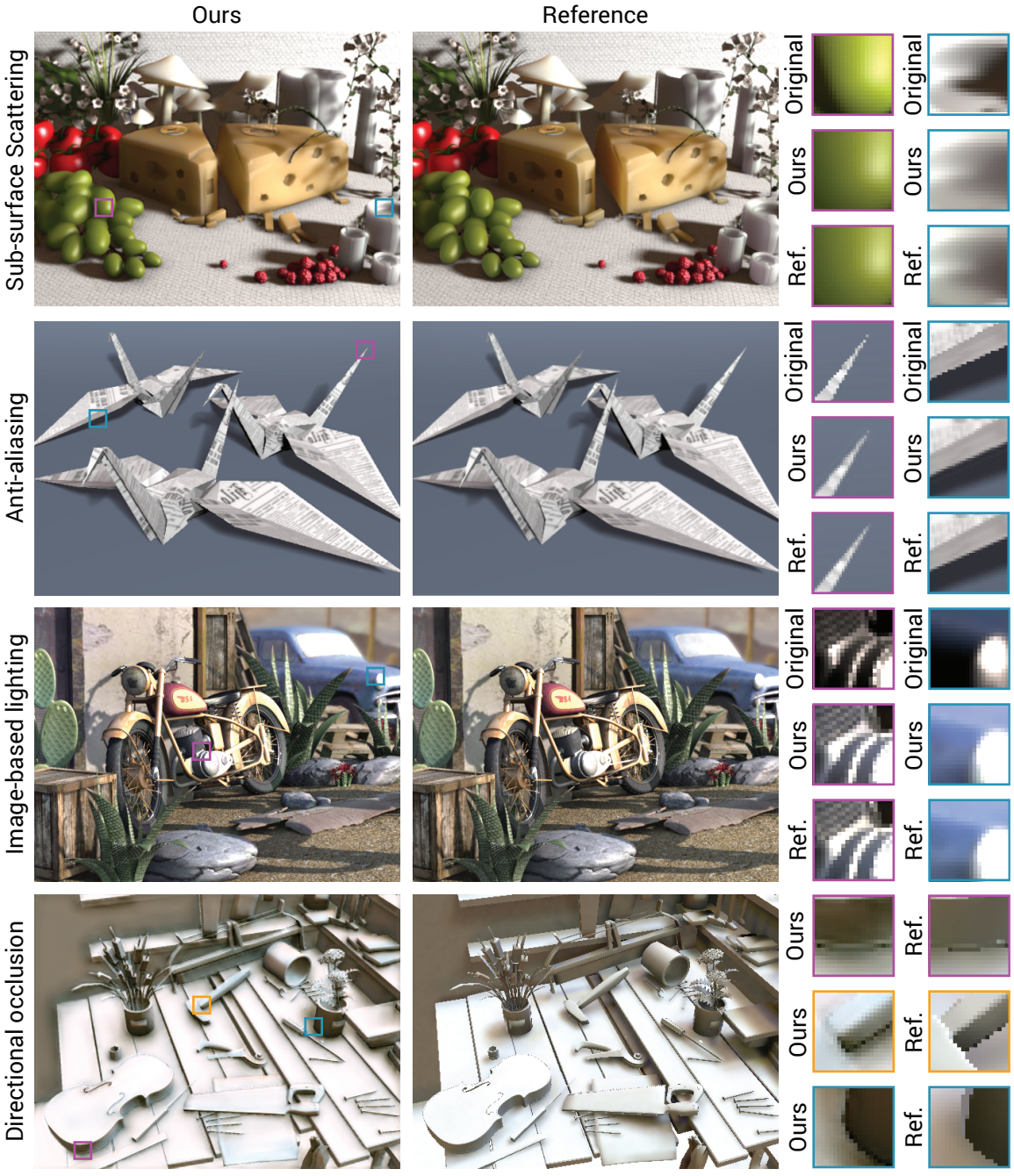


Figure 7.7: Deep Shader outputs for sub-surface scattering, anti-aliasing, image-based lighting and directional occlusion. ‘Original” shows the scenes without the respective effects.

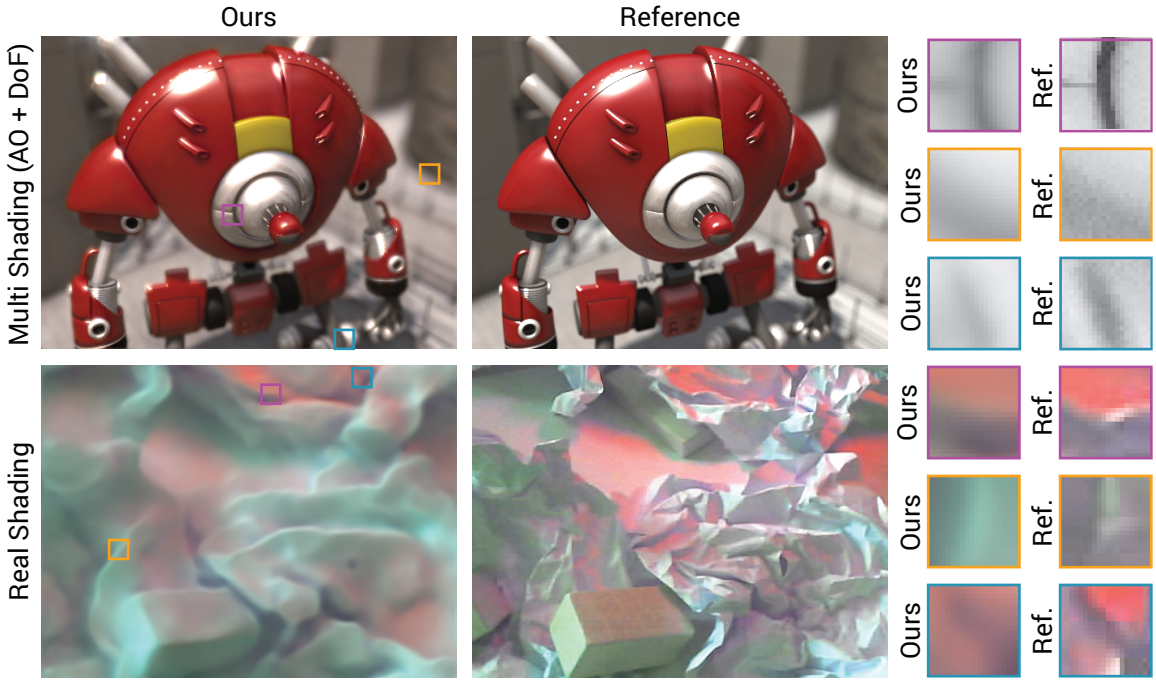


Figure 7.8: Deep Shading goes beyond classic screen-space shading. A single Deep Shader can compute multiple effects at once (top row) or reproduce real world shading captured using an RGBD camera (bottom row). For multi shading, the insets show the effect applied to the same scene but with diffuse white surfaces to highlight the effect of the network.

level of glossiness. The latter is not surprising as the extreme here is a perfect mirror which would need a complete encoding of the illumination used in training, which has a resolution of several megapixels, into a few hundred network kernels.

Directional Occlusion Directional occlusion [Ritschel et al. 2009b] is a generalization of AO where sample directions are associated with radiance samples from an environment map and light from unblocked directions is accumulated. DO is applied by using it directly as ambient lighting term. As for AO, ray-tracing is used to resolve occluded directions within a fixed world-space radius. While the related AO works well, DO is more challenging for Deep Shading. The increased difficulty comes from indirect shadows now having different colors and appearing only for certain occlusion directions. As can be seen in the bottom row of Figure 7.7, the color of the light from the environment map and the color of shadows match the reference but occlusion is weakened in several places. This is due to the fact that the indirect shadows resulting from DO induce much higher frequencies than unshadowed illumination or the indirect shadows in AO, which assume a constant white illumination from all directions, and are harder to encode in a network.

Multi Shading Finally, we learn a Deep Shader that combines several shading effects at once and computes a scene shaded with ambient occlusion to produce soft shadows and additional shallow depth-of-field. The network uses the union of attributes of the AO and DoF networks simultaneously. Note, that this single Deep Shader realizes both effects together in a single network. Unlike the AO network, the network output is not multiplied with the remaining

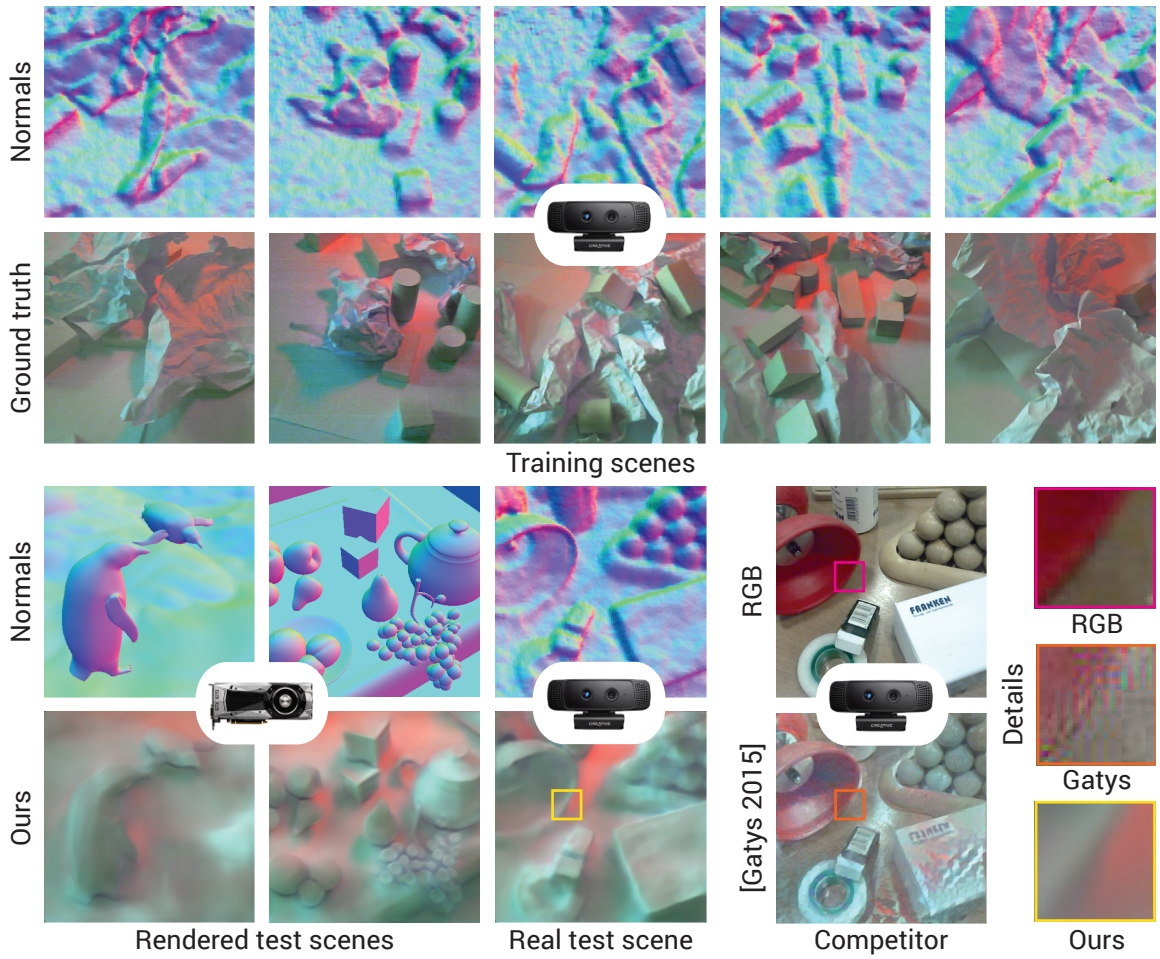


Figure 7.9: Deep Shading learnt from RGBD video that captures screen-space normals (first row) and appearance (second row). The CNN can learn the correlation of the two, including directional light, occlusion and bounces and transfer it to novel synthetic or captured normal images (3rd and 4th row, left). This performs better than established deep-learning-based style transfer [Gatys et al. 2015] from RGB to RGB images (bottom rows, right).

shading but already applied by the CNN itself as the DoF component cannot be decoupled. An image generated using the network (Figure 7.8, top row) exhibits both effects present in the training data. Shallow depth-of-field is added and corners are darkened, as can be seen particularly well when applying the network to a fully white radiance input (cf. insets).

Real Shading As an addition, we demonstrate that Deep Shaders can not only be trained from renderings but also from actual photographs. In a prototypical experiment, we captured 12,000 samples of RGB and depth images of a scene of neutrally colored objects in a characteristically lit environment (Figure 7.9 a) using a standard RGBD camera (Creative Senz3D) by fixing the position of the camera while moving and rearranging different objects in front of it. The main light sources are distant enough to create an almost directional direct lighting environment where incident lighting only depends on the normal in camera space which is at the same time interacting with visibility changes due to occlusions by nearby geometry and global illumination effects in general. Using the known camera intrinsics, we derive camera positions from the captured depth

values which are in turn used to determine camera space normals. After registering normal and RGB data, we train a network to map the former to the latter.

Images generated by the network for real and rendered depth data (Figure 7.8, bottom row and Figure 7.9 b+c) reproduce the lighting environment with red or bluish tints depending on the objects' normals and including AO-like darkening in corners and creases, even for geometry very different to the training data. The main limitation is due to the precision of our depth camera which cannot capture fine details in the geometry. Consequently, geometry details below a certain scale are ignored by the network which is apparent when comparing its output on training depth images to ground truth (Figure 7.9 b vs. Figure 7.9 d). Finally, learning the correlation of normals and appearance outperforms deep-learning based style transfer using only RGB information [Gatys et al. 2015], which changes hues and structures but fails to capture the light transport aspects, e. g., changing the red shiny plastic into diffuse white paper or adding shadows such as Deep Shading does.

We believe that more detailed shading could be learned using improved depth sensors and that an extension to proper world normals, e. g., by mounting the camera on a gonioreflectometer, or even to other varying attributes like albedo, is only a matter of acquisition.

Animations For applying the AO and GI Deep Shaders to dynamic scenes, we found it beneficial to increase temporal coherence by warping CNN outputs from a small number of previous frames to the current view and blending them with the current result [Scherzer et al. 2012]. Figure 7.10 shows several sample frames for fully dynamic scenes.

7.5 Analysis

In the first part of this section, we address some shortcomings in the form of typical artifacts produced by our method and also discuss how the network reacts when applying it to new resolutions and attributes rendered with a different field of view. The remainder of the section explores some of the countless alternative ways to apply CNNs, and machine learning in general, to the problem of screen-space shading. We cover different choices of actual network structure (Section 7.5.2), loss function (Section 7.5.3) and training data anatomy (Section 7.5.4) as well two techniques competing with deep CNNs, namely multi-layer perceptrons and random forests (Section 7.5.5).

7.5.1 Visual Analysis

Typical Artifacts Light transport can become highly complex and the mapping from screen-space attributes to shading is inherently ambiguous due to partial information, hence we cannot expect a CNN to act correctly for every given input. Even what looks plausible in a static image may start to look painterly or surrealistic when seen in motion: patterns resembling correct shading emerge but being inconsistent with the laws of optics and with each other. We show exemplary artifacts in Figure 7.11.

Capturing high frequencies is a key challenge for Deep Shaders (Figure 7.11, first image). If the network does not have enough capacity or was not trained enough the results might over-blur with respect to the reference. We consider this a graceful degradation compared to typical

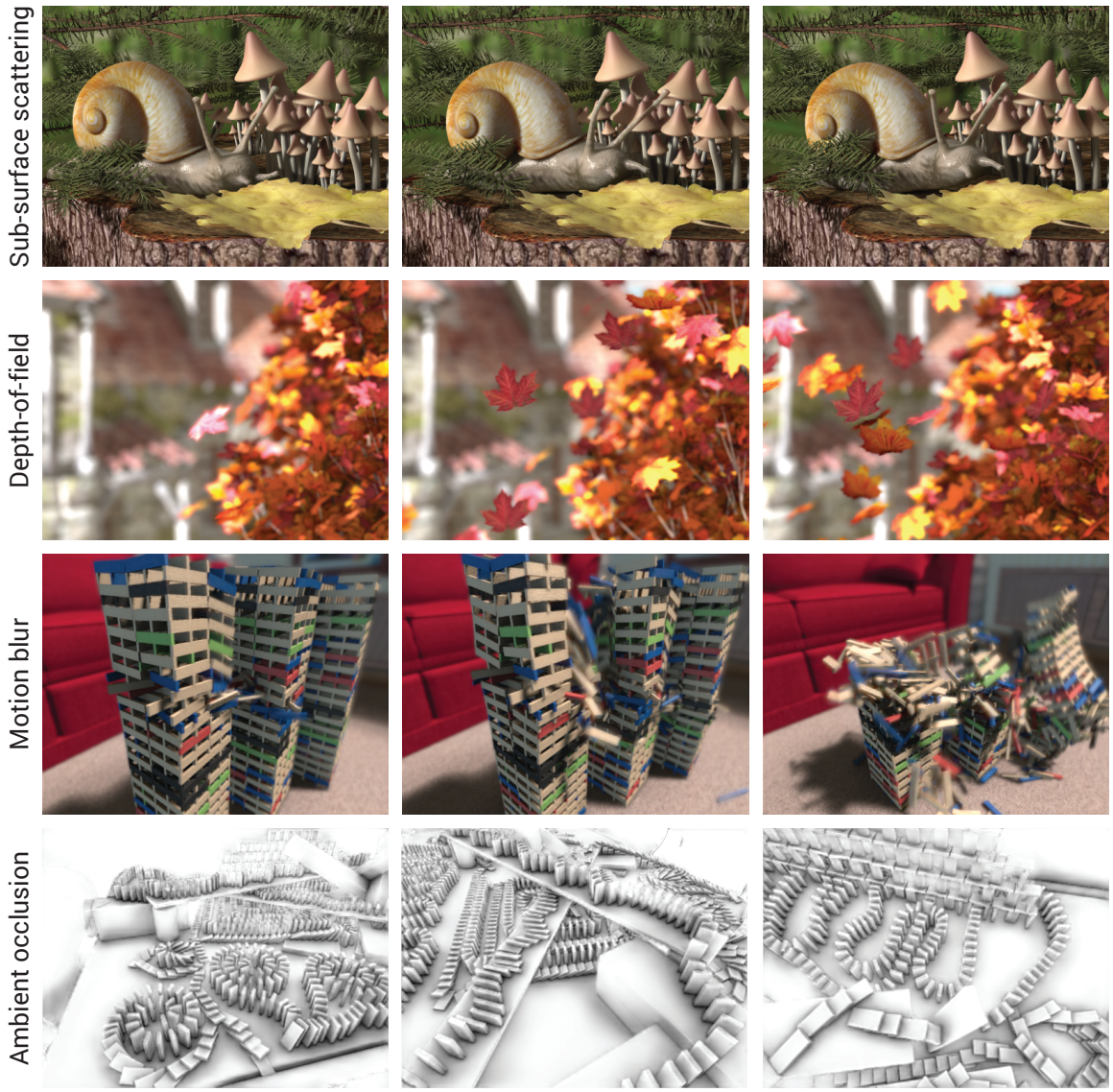


Figure 7.10: Snapshots from animated sequences rendered using various Deep Shaders.

artifacts of man-made shaders such as ringing or Monte Carlo noise (Figure 7.6) which are highly unstable over time and unnatural with respect to natural image statistics. Sometimes, networks trained on RGB tend to produce color shifts (Figure 7.11, second) which can typically be weakened by increasing the number of features. CNN-learned filters may also introduce high frequencies resembling ringing due to false-positive neuron activations resulting from over-fitting (Figure 7.11, third). Sometimes effects propagate into the wrong direction in world space, e. g., geometry may cast occlusions on things behind it, ignoring the correct depth ordering (Figure 7.11, fourth). At attribute discontinuities, the SSIM loss lacking an inter-channel prior sometimes fails to prevent color ringing (Figure 7.11, last).

Range of Values While many attributes used as an input to Deep Shaders are limited to a certain range e. g., normals, others are theoretically unbounded and a CNN might react unexpectedly when confronted with values far outside the training range. We tackle this by, first,

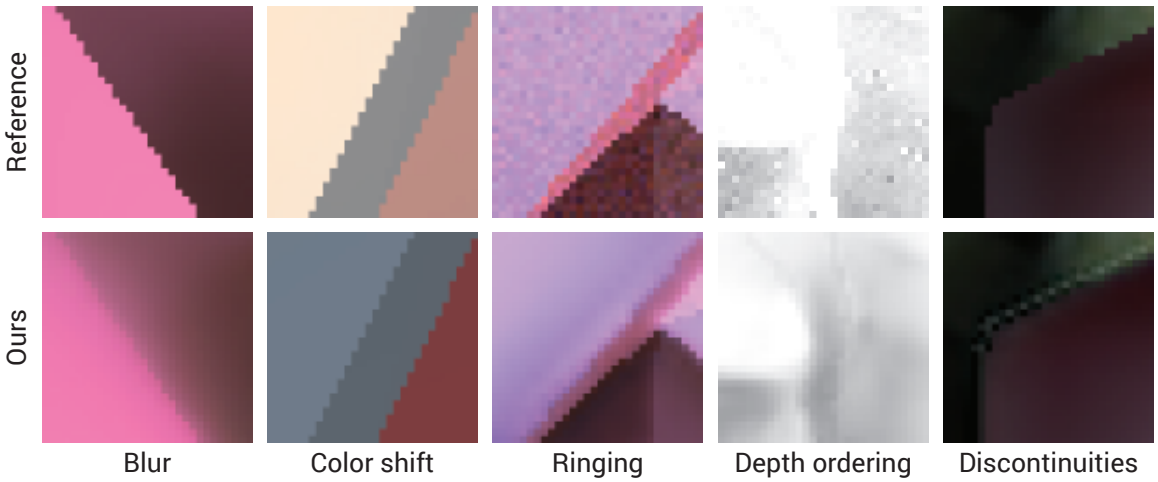


Figure 7.11: Typical artifacts of our approach. From left to right: blur, color shift, ringing, missing depth ordering and problems at attribute discontinuities.

choosing particularly large ranges of possible values during training and second, if necessary, scaling inputs on which light transport depends linearly (e. g., the unblurred input to the depth of field network) in a linear way before feeding them to the network and undoing this transformation afterwards.

Effect Radius Typically, screen-space shading is faded out based on a distance term and only accounts for a limited spatial neighborhood. As we train in one resolution but later apply the same trained network also to different resolutions, the effective size of the neighborhood changes. As a solution, when applying the network at a resolution which is larger by factor of N compared to the training resolution, we also scale the effect radius accordingly, dividing it by N. While the effect radius is not an input to the network but fixed in the training data, it can still be adjusted at test time by scaling the attributes determining the spatial scale of the effect, e. g., of the camera space positions in the case of AO, DO or GI, or of the distance to the focal plane in the case of DoF. To conclude, effect radius and resolution can be changed at virtually no additional cost (per pixel) without re-training the network.

Internal Camera Parameters As we compute our training data using a fixed FOV (50°), it is not clear how the trained networks perform on framebuffers rendered using a different FOV. Figure 7.13 investigates the influence of a FOV mismatch on image quality. To keep the image content as similar as possible while changing FOV, we performed a dolly-zoom. Judging from the minimal fluctuation of the error, the network is very robust to FOV "mismatches".

7.5.2 Network Structure

To better understand how the structural parameters of our CNN architecture control its expressiveness and computational demand we investigate two modes of variation: varying spatial extent of the kernels as well as the number of kernels on the first level u_0 which also determines the number of kernels for the remaining levels (Section 7.3.2). We seek the smallest network with adequate learning capacity, that generalizes well on previously unseen data. The results

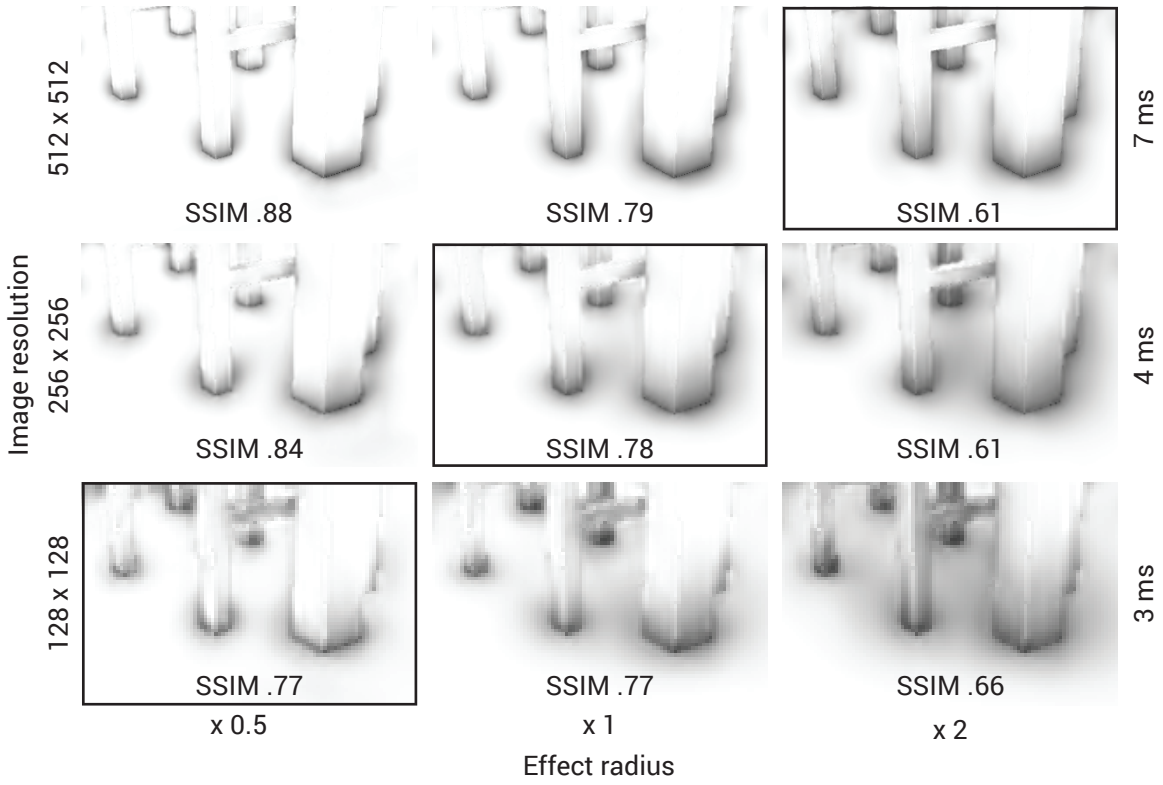


Figure 7.12: Increasing (decreasing) the resolution shrinks (enlarges) effect size relative to the resolution. The radius can be increased (decreased) again with no effects on timings by scaling the input attribute determining the radius accordingly, e. g., the positions for AO. Consequently, images on the diagonal are similar. All images show outputs produced by the same network. The timings for each row are identical. The SSIM is higher for smaller effect radii that are easier to reproduce.

are summarized in Figure 7.14 a+b for the example of AO.

Spatial Kernel Size The green and yellow lines in Figure 7.14 a show the evolution of training, validation and test error with an increasing number of training iterations for a medium number of kernels ($u_0 = 8$) and varying the spatial extent of the kernels. We see that with a kernel size of 5×5 px, the training profile slightly lags behind that for kernel size of 3×3 px, but both approach a similar test loss at 100k iterations, i. e., the networks have sufficient capacity to approximate the mapping with neither beginning to over-fit. We observe a similar relative timing relationship between the pairs of networks with $u_0 = 4$ and $u_0 = 16$. As, regarding running time, the one with a kernel size of 3×3 px is about twice as fast as the one with 5×5 px we opt for the former. Regarding memory consumption, different spatial kernel sizes have only a small impact as the memory needed to store convolution parameters is insignificant compared to the memory usage of the intermediate representations.

Initial Number of Kernels The orthogonal mode of variation is u_0, the number of kernels on the first level, also influencing the kernel counts of subsequent layers which are expressed as multiples of u_0. Again, we plot the training, validation and test errors, this time for different u_0 (Figure 7.14 a, green vs. blue line, yellow vs. purple line). Reducing the number to $u_0 = 4$ clearly leads to a loss of expressiveness as can be seen from both the training and test loss. Further,

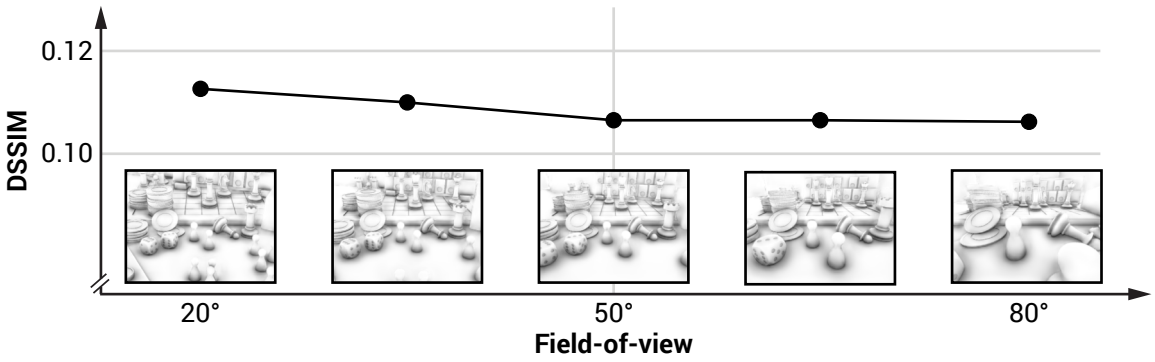


Figure 7.13: Effect of FOV on image quality. The horizontal axis is FOV in degrees. The central line is the reference of 50°. The vertical axis is DSSIM error w.r.t. the reference. Note that the vertical axis spans only a small difference (.106 to .112), indicating FOV has no large impact on visual quality.

nets with $u_0 = 16$ perform only slightly better than those with $u_0 = 8$ (Figure 7.14 b) while losing out in compute time by more than a factor of 6. This is in part due to increased memory consumption, in particular for the intermediate representations.

Structural Choices for Other Effects We perform an analogous analysis for other effects. We start off with spatial extents of 3×3 px and 5×5 px, with $u_0 = 8$, and proceed to increase or decrease u_0 in accordance with over-fitting / under-fitting characteristics exhibited by the the train-test error curves. Table 7.1 indicates the final choices of network structure. Additionally, the corresponding train-test error curves are shown in Figure 7.14 c with their test loss-vs.-speed characteristics captured in Figure 7.14 d.

The number of iterations shown in Figure 7.14 a+c, though sufficient to make decisions about the structural parameters, still leave the network with scope to learn more (indicated by negative slopes of the train-test curves). We therefore resumed training for the respective optimal choices of $u_0 = 8$ for about 400 k additional iterations until the losses settled completely.

It is worth noting that, while the training errors we measured are smaller than the respective validation errors as expected, in some cases we measured smaller test than training errors (e. g., Figure 7.14 a). This is possible because we use disjoint sets of scenes for training and validation on one hand and testing on the other hand, i. e., the scenes used for testing might be "easier" to tackle for the network. For example, they might contain less cases where the deferred shading inputs to the network are insufficient to derive ground truth results because of lacking information about occluded geometry.

7.5.3 Choice of Loss Function

The choice of loss function in the optimization has a significant impact on how Deep Shading will be perceived by a human observer. We trained the same network structure using the common $\mathscr{L}_1$ and $\mathscr{L}_2$ losses as well as the perceptual SSIM metric and also using combinations of the three. Figure 7.15 shows a visual comparison of results produced by the respective nets. We found $\mathscr{L}_1$ and $\mathscr{L}_2$ to be prone to producing halos instead of fading effects out smoothly as can be seen in the first two columns. The combination of $\mathscr{L}_2$ with SSIM also exhibits these kind of artifacts to a lesser extent. SSIM and SSIM + $\mathscr{L}_1$ both produce visually pleasing results with pure SSIM being

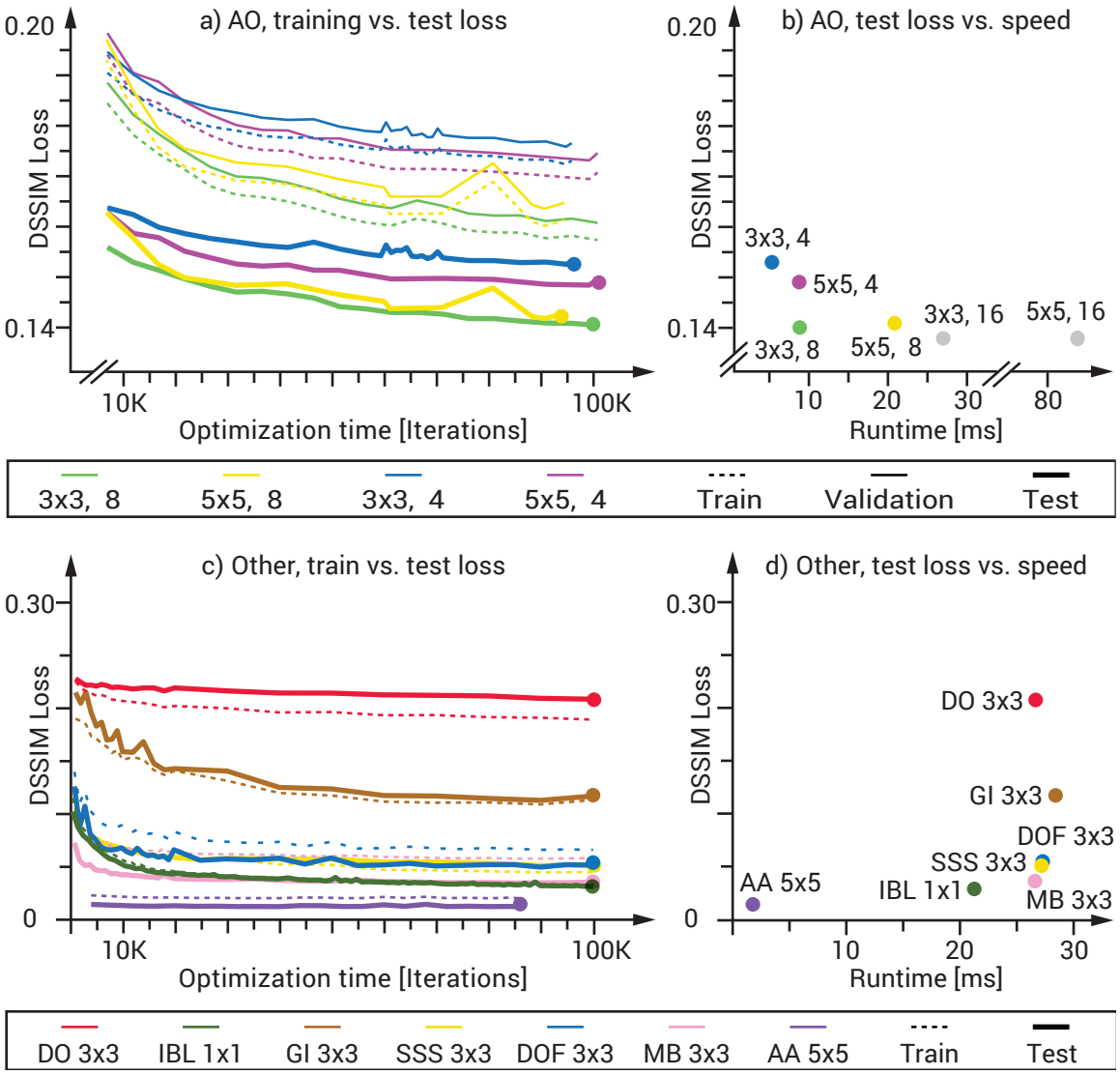


Figure 7.14: Analysis of different design choices for different effects in terms of compute time and DSSIM loss. The vertical axes on all plots corresponds to DSSIM loss (less is better). The horizontal axes of the line plots range over the number of training iterations. The scatter plots have computation time of the Deep Shader as the horizontal axis. a): Train, test and validation loss as a function of iterations for different designs of AO (curves). b): Relation of final loss and compute time for different designs for AO. c): Loss as a function of iterations for the chosen designs for other effects (curves). d): Comparison of compute time and final loss for the other effects, as a means of placing their relative complexity.

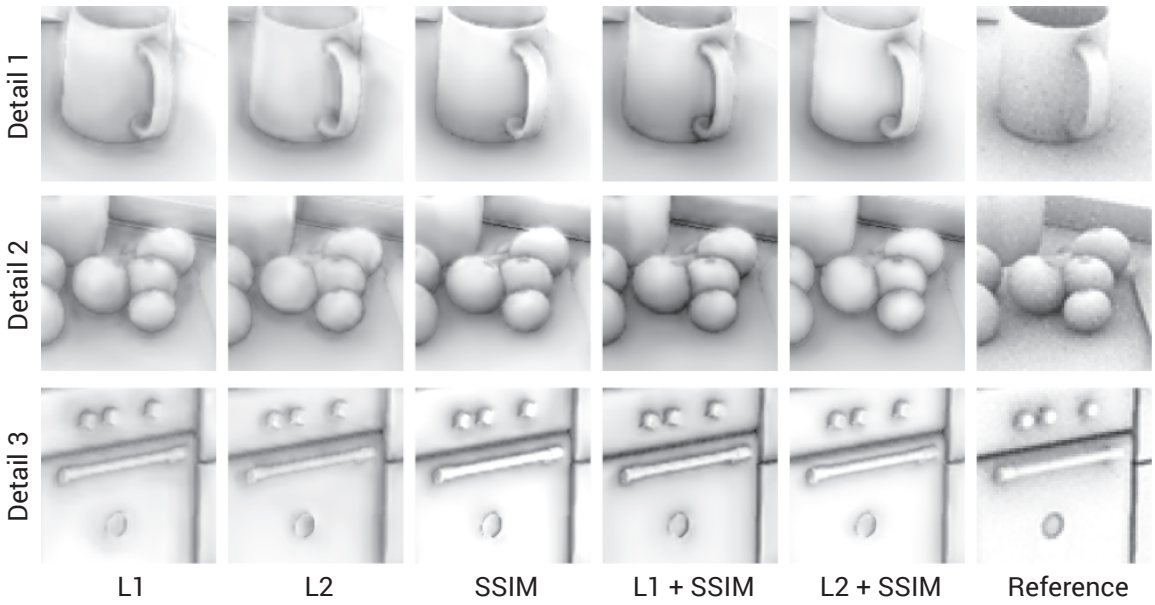


Figure 7.15: Outputs produced by the same network trained with different loss functions for the case of AO.

more faithful to the amount of contrast found in the reference images.

7.5.4 Training Data Trade-offs

Ideally, a training set consists of a vast collection of images with no imperfections from Monte Carlo noise and showing a large number of different scenes. Yet, in practice, the time budget to produce training data is typically limited and the question how to spend this time best arises. In this section we investigate different possible trade-offs.

Amount of Noise vs. Image Set Size The time spent to generate a training set is roughly linear in both, the number of MC samples taken per pixel and the number of individual images rendered (before data augmentation), e. g., we can render twice as many images if we only use half as many samples per pixel. To investigate to which extent noise in the training data affects the quality of the trained network and to find out which trade-off between the number of samples and individual views should be taken, we performed a same-time comparison using AO as an example. As can be seen from the left part of Figure 7.16, a larger number of views per scene is typically more desirable than noiseless images. Only for very low sample counts leading to excessive noise (around 16 tspp in the case of AO), the variance in the individual images begins to hinder proper training of a network.

Scene Diversity Another factor that has influence on the quality of the trained Deep Shaders is the diversity of scenes in the training set, e. g., a CNN that has only seen round objects during training will fail to correctly re-produce its effect for square objects. In our training sets, we use 1000 views from each of 10 different scenes as our starting points (cf. Section 7.5.4). To see how well CNNs perform for less diverse data we produced DO training sets of the same total size but for a lower number of different scenes. DO was chosen as we observed it to be particularly

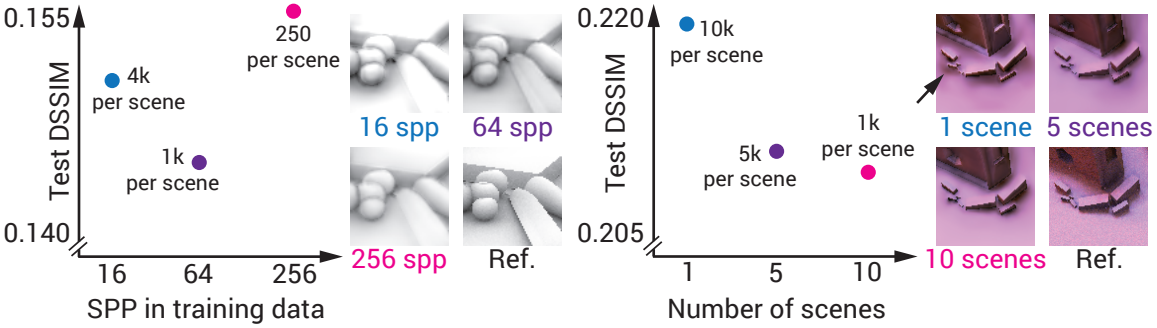


Figure 7.16: Left: Data points correspond to the same time budget to produce training data but using different numbers of samples per pixel. The resulting number of individual views per scene (before data augmentation) is given for each point. Next to this, AO produced by the corresponding trained networks is shown. Right: Data points correspond to the same time budget to produce training data but with different trade-offs regarding the scene count. Next to this, patches from a test scene are shown.

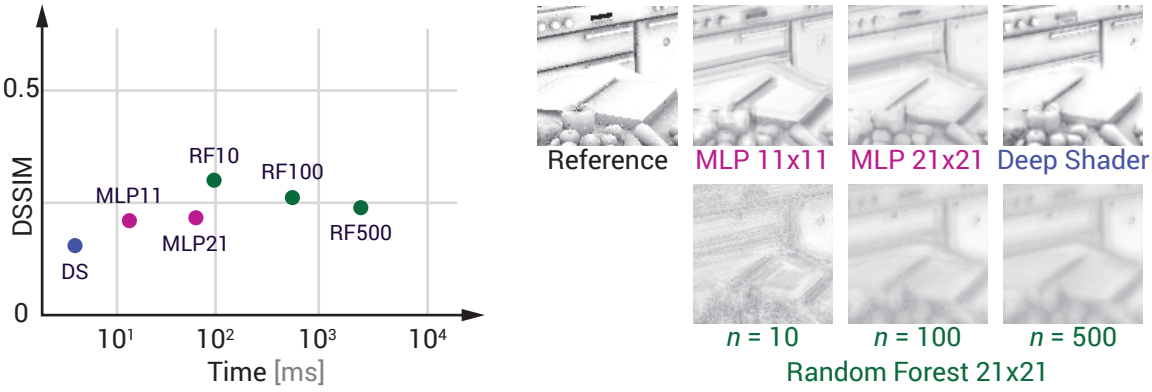


Figure 7.17: AO computed using random forests, shallow MLPs and Deep Shading. The vertical axis is image error (DSSIM) on a linear scale. The horizontal axis is compute time for 256 × 256 px images on a logarithmic scale. n indicates the number of trees.

sensitive to the scene diversity. The resulting DSSIM values for (the same) test set are plotted in the right part of Figure 7.16. While the error for five scenes compared to a single one is 5% smaller, increasing the number further to 10 scenes leads to only a smaller advantage of about another 1% which indicates that our scene set is of acceptable diversity. In the case of DO, the difference in the loss visually translates to a more correct placement of darkening. A network trained with only one scene tends to create “phantom occlusions” in free spaces (Figure 7.16, far right).

7.5.5 Comparison With Other Regression Techniques

Besides deep CNNs, competing approaches such as shallow multilayer perceptrons (MLPs) and random forests (RFs) [Criminisi and Shotton 2013] could supposedly be used for our objective, having found use in other image synthesis tasks such as estimation of filter parameters [Kalantari et al. 2015] or relighting [Ren et al. 2013; Ren et al. 2015]. To investigate these alternatives we perform a comparison, training our Deep Shader, MLPs, and RFs to produce AO for 256 × 256 px deferred shading buffers. As for MLPs and RFs direct regression on the full buffers would be

prohibitively expensive, we train the former on patches of 11×11 px and 21×21 px and the latter on patches of 21×21 px to predict AO for the central pixel. To compute AO in high resolution later on they are swept across the whole input buffers. The MLPs consist of 2 hidden layers with 50 nodes each while the RFs are split across four cores and have a minimum of five samples per leaf.

We train the MLPs and RFs using $\mathscr{L}_2$ loss as in [Ren et al. 2015] and evaluate all approaches using SSIM. For MLPs, we measure their execution time using the same OpenGL implementation as used for our CNNs while we employ scikit-learn [Pedregosa et al. 2011] on a regular workstation for the RFs.

Figure 7.17, shows the relative speed and quality of MLPs and RFs compared to Deep Shading. Pixel-wise predictions with RFs clearly lose out on both visual quality and running time. RF run-times increase linearly with the number of trees, more of which are necessary to construct a better ensemble. Even with more readily parallelizable variants of RFs [Bosch et al. 2007] there would have to be a run-time improvement of more than two orders of magnitude to be comparable to a Deep Shader. For the MLPs, we actually observe a degradation of image quality with increasing patch size: as the number of parameters increases quadratically with the diameter of the filter, the resulting MLP becomes prone to over-fitting. Only far more training data and training iterations could mitigate this problem.

In conclusion, deep CNNs have an edge over competing approaches for our problem setting as they allow for large receptive field sizes through stacked (smaller) convolutions and down-sampling, without an exponential increase in the number of parameters, while leveraging the expressive power of deeper representations [Hastad 1986].

7.6 Comparison to Previous Work

Related previous work can be found in both computer graphics where attributes have to be converted into appearance and computer vision where appearance has to be converted into attributes.

Attributes-to-appearance The rendering equation (Section 2.5.1) is a reliable forward model for computing appearance in the form of radiance incident at a virtual camera sensor when a three-dimensional description of the scene using attributes like positions, normals and reflectance is given. As described in Chapter 3, a large number of methods for solving it exist, such as finite elements, path tracing or photon mapping. The high-quality results these achieve come at the cost of significant computational effort. Interactive performance is only possible through advanced parallel implementations in specific GPU programming languages [Owens et al. 2007], demanding not only substantial programming effort, but also proficiency. By leveraging deep learning architectures, we seek to overcome those computational costs by focusing computation on converting attributes into appearance according to example data rather than using physical principles.

Our approach is based on screen-space shading that has been demonstrated to approximate many visual effects at high performance (Section 3.6.3) such as ambient occlusion (AO) [Mittring 2007], indirect light (GI) [Ritschel et al. 2009b], sub-surface scattering (SSS) [Jimenez et al. 2009], participating media [Elek et al. 2013], depth-of-field (DOF) [Rokita 1993] and motion blur

(MB) [McGuire et al. 2012]. Anti-aliasing too can be understood as a special form of screen-space shading, where additional depth information allows to post-blur along the "correct" edge to reduce aliasing in FXAA [Lottes 2011]. All of these approaches proceed by transforming a deferred shading buffer [Saito and Takahashi 1990], i. e., a dense map of pixel-attributes, into RGB appearance. Seen from this point-of-view, our trained CNNs also constitute screen-space approaches.

Applications of machine learning to image synthesis have been scarce so far with a few notable exceptions. A general overview of how computer graphics could benefit from machine learning, combined with a tutorial from a CG perspective, is given byHertzmann [2003]. The CG2Real system [Johnson et al. 2011] starts from simulated images that are then augmented by patches of natural images. It achieves images that are locally very close to real world example data, but it is founded in a simulation system, sharing all its limitations and design effort. Recently, CNNs ahve been used to transfer artistic style from a corpus of example images to any new exemplar [Gatys et al. 2015]. Our work is different as shading needs to be produced in real-time and in response to a great number of guide signals encoding the scene features instead of just locally changing RGB structures when given other RGB structures. Dachsbacher [2011] has used neural networks to reason about occluder configurations. Neural networks have also been used as a basis of pre-computed radiance transfer [Ren et al. 2013] (PRT) by running them on existing features to fit a function valid for a single scene. In a similar spirit, Ren et al. [2015] have applied machine learning to re-lighting. There a multi-layer perceptron (MLP) learns how image pixels change color in response to modified lighting. Both works [Ren et al. 2013; Ren et al. 2015] demonstrate high-quality results when generalizing over light conditions but share the limitation to static 3D scenes or 2D images, respectively, without showing generalization to new geometry or animations such as we do. Such a generalization is critical for real applications where geometry is dynamic, resulting in a much more demanding problem that is worth addressing using deep learning. We would argue that Deep Shading achieves this generalization required to make learning a competitive image-synthesis solution in practice, in the same way that screen-space shading is highly adopted by the gaming industry for its inherent support for dynamic scenes.

Earlier, neural networks were used to learn a mapping from character poses to visibility for PRT [Nowrouzezahrai et al. 2009]. Without the end-to-end learning made possible by deeper architectures, the aforementioned approaches do not achieve generalization between scenes, but remain limited to a specific room, character and so on. Kalantari et al. [2015] have used sample data to learn optimal parameters for filtering Monte Carlo Noise. Our function domain, i. e., screen-space attributes, might appear very similar to theirs but does not include any noisy estimate of the shading to be computed. Also the way the method is applied is very different. While Kalantari et al. [2015] learn parameter values for pre-defined analytic denoising filters, we learn the entire light transport in screens-space. Not much is known about the complexity of the mapping from attributes to filter settings and what is the effect of sub-optimal learning. In our case, the mapping from attributes to the value is as complex as shading itself. At the same time, the stakes are high: learning a mapping from attributes to shading results in an entirely different form of interactive image synthesis, not building on anything such as Monte-Carlo ray-tracing that can be slow to compute.

For image processing, convolution pyramids [Farbman et al. 2011] have pursued an approach that optimizes over the space of filters to the end of fast and large convolutions. Our approach optimizes over pyramidal filters as well, but allows for a much larger number of internal states and much more complex filters defined on much richer input. Similar to convolution pyramids our network is based on a "pyramidal" CNN allowing for fast but large filters to produce long-range

effects such as distant shadows or strong depth of field.

Appearance-to-attributes The inverse problem of turning image appearance into semantic and non-semantic attributes lies at the heart of computer vision. Of late, deep neural networks, particularly CNNs, have shown unprecedented advances in typical inverse problems such as detection [Krizhevsky et al. 2012], segmentation and detection [Girshick et al. 2014], or depth [Eigen et al. 2014], normal [Wang et al. 2015] or reflectance estimation [Narihira et al. 2015]. These advances are underpinned by three developments: availability of large training datasets, deep but trainable (convolutional) learning architectures, and GPU accelerated computation. Another key contributor to these advances has been the ability to train end-to-end, i. e., going from input to desired output without having to devise intermediate representations and special processing steps.

One recent advance is of particular importance in the application of CNNs to high-quality shading: The ability to produce dense per-pixel output, even for high resolutions, by CNNs that do not only decrease, but also increase resolutions as proposed by [Long et al. 2015; Hariharan et al. 2015] resulting in fine per-pixel solutions. For the problem of segmentation, Ronneberger et al. [2015] even apply a fully symmetric U-shaped net where each down-sampling step is matched by a corresponding up-sampling step that may also re-use earlier intermediate results of the same resolution level.

CNNs have also been employed to replace certain graphics pipeline operations such as changing the viewpoint [Dosovitskiy et al. 2015; Kulkarni et al. 2015]. Here, appearance is already known, but is manipulated to achieve a novel view. In our work, we do not seek to change a rendered image but to create full high-quality shading from the basic output of a GPU pipeline such as geometry transformation, visible surface determination, culling, direct light, and shadows.

We seek to circumvent manual programming of efficient screen-space shaders and even elude the need to come up with analytic approaches to recreate shading at hand by instead learning from examples and optimizing over deep convolutional networks to achieve a single general screen-space shader that is optimal in the sense of certain training data.

7.7 Conclusion and Future Work

We have proposed Deep Shading, a system leveraging deep learning to turn attributes of virtual 3D scenes into appearance. It is the first example of performing complex shading by pure learning from data without any physical consideration. We have shown that CNNs can model any screen-space effect as well as combinations of them at competitive quality and speed, giving a proof-of-concept that image synthesis can be learnt from data without human intervention and programming effort.

We share limitations with common screen-space shading techniques, namely missing shading from objects not contained in the image due to occlusion, clipping or culling but also inherit its benefits such as handling of large, dynamic scenes in an output-sensitive manner. In future refinements, the Deep Shader could even learn to fill-in more missing information which it already does in some cases, such as for AO (Figure 7.6) where the network seems to complete known geometry configurations in a natural way. Using different scene representations, such as surfels or patches could help, too.

Some shading effects like DO and GI are due to very complex relations between screen space attributes. Not all configurations are resolved correctly by a network with limited capacity, such as ours which runs at interactive rates. We have however observed that typical artifacts are less salient than from human-designed shaders which suffer e. g., from ringing or MC noise (Figure 7.6). Instead, CNNs reproduce patterns encountered during training which are inherently natural as they have been computed using traditional light transport methods and appear visually plausible. A perceptual study could verify this observation. Temporal coherence can be addressed in the same way as for classic screen-space shading by temporal integration, resulting in mostly coherent behavior as seen in the supplemental video.

We have demonstrated that Deep Shading can achieve a better approximation than common methods in the case of AO, given the same time budget (Figure 7.6). We would hope that even more diverse training data, advances in learning methods, and new types of deep representations or losses will allow surpassing the performance of human-programmed shaders for more effects in a not-so-distant future.

8

Capture and Reproduction of Phosphorescence

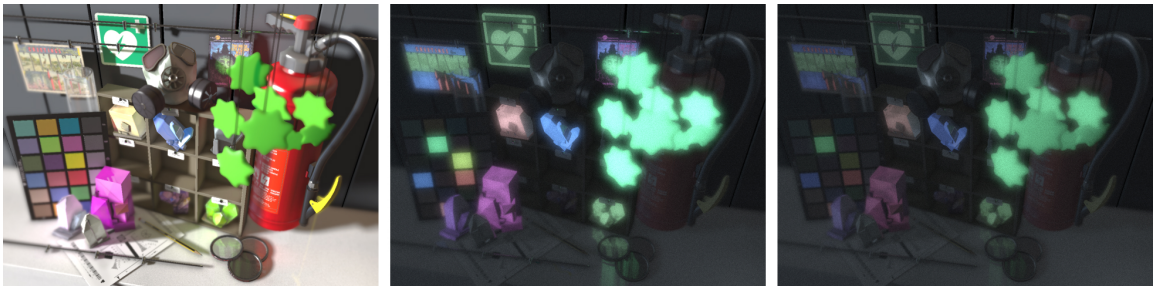


Figure 8.1: Designed phosphorescent materials before (left) and after having switched off the light for 1 s (middle) and for 80 s (right). Phosphorescent appearance changes differently for different materials over time (Render time 454 ms with, 448 ms without phosphorescence, 640 × 480 px).

8.1 Introduction

Phosphorescent materials re-emit light with a substantial temporal delay after they were exposed to illumination. They are widely used in dark conditions where visibility is important but where no electric power is available. Typical applications of such materials include emergency signs, door handles, light switches, user panels or automotive parts but also toys, artwork and fashion. The practical importance of phosphorescence ranges from entertainment to saving lives. Despite all these use cases, no comprehensive pipeline for digital acquisition, reproduction and manipulation of phosphorescent appearance has been available to date. In this chapter, we will propose such a framework.

We first devise a phenomenological model of phosphorescence based on rate equations, which allows for efficient simulation under arbitrary changes of illumination and geometry over time (Section 8.3). Our second contribution is a practical setup to acquire parameters for our model from physical samples (Section 8.4). We propose a reproduction method (Section 8.5) that allows to simulate phosphorescent appearance using the acquired data with only a small computation and storage overhead, despite phosphorescence extending the integration domain

by two dimensions, making straightforward Monte Carlo integration prohibitive when aiming for real-time performance. The reproduction fidelity is evaluated by comparing rendered images to actual photographs. Our final contribution is a method to design phosphorescent materials. Instead of tweaking abstract rate equation parameters, an artist specifies appearance at key frames and suitable parameter values for a given time-varying geometry and lighting are computed (Section 8.6).

8.2 Physical Background

A substance is called *luminescent* if it itself emits light (and does not do so just because of heat). In particular, *photoluminescence* is emission in response to light itself. It is necessary to distinguish between photoluminescence and other forms of luminescence due to chemical reactions, electrical energy or mechanical pressure which may be similar in appearance but different in their origins. If the re-emission happens without significant time-delay (<10 ns) the material is typically called *fluorescent* while *phosphorescence* is characterized by a significant delay, up to hours or even days. Detailed introductions to various aspects of the photoluminescence of solids are available [Leverenz 1968] but the precise physical reasons for phosphorescence at the quantum mechanical level are beyond the exposition of this paper. Instead, we focus on three clearly observable phenomena of phosphorescent appearance — decay, saturation and wavelength shift — described in the following.

First, phosphorescent emission is subject to *decay*, where the material once illuminated re-emits a decreasing amount of light over time. The fall-off is often described by differential equations that, under simplified conditions, have exponential functions as closed form solutions [Chen et al. 2011]. Different from reflection, phosphorescent emission depends not only on the illumination at the current but also at previous points in time.

Second, phosphorescence is subject to *saturation* and depends on the light previously absorbed in a nonlinear way: A material does not necessarily double its emission when we expose it to twice the illumination. At some point, no additional phosphorescent re-emission can be induced by additional illumination. Another consequence of this behavior is that, given a configuration of material and incident flux density, after a certain time an equilibrium is reached where under constant lighting the re-emission stagnates, too.

Third, photoluminescence entails a *wavelength shift*. Absorbed light of one wavelength is re-radiated at a possibly different wavelength. Donaldson [1954] introduced re-radiation matrices to describe the wavelength shift occurring during re-radiation in fluorescent pigments. For the materials we acquired, no evidence for hue shifts was observed when varying the incident wavelengths and therefore we model re-emission using vectors instead of matrices.

8.3 A Model of Phosphorescence

We will first give a motivation and then continue to introduce our phenomenological model which is formulated using rate equations, before finally discussing its limitations.

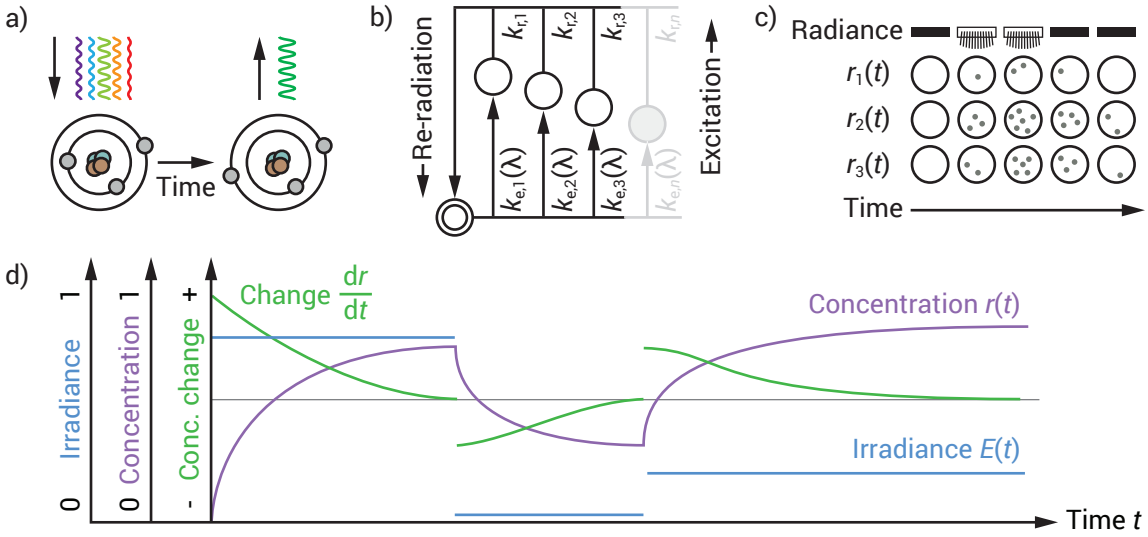


Figure 8.2: a): A substance is excited by light of one and re-radiates light of a different spectrum. b): From an unexcited state (lower left corner) molecules enter one out of n_c excited states. The excitation rate k_e depends on the exciting wavelength λ, the re-radiation rate k_r is independent of it. c): We model the concentration of molecules in various states. Here, for three states, the (real-valued) concentrations are visualized as varying dot populations. d): Plots of irradiance (blue), continuous concentration (purple) and change of concentration (green) over time for one state.

8.3.1 Motivation

In physics literature, various models exist to describe phosphorescent materials, in particular for the decay of phosphorescent emission. Some are closed-form formulas (e. g., double exponentials [McKeever and Chen 1997]), with or without a physical interpretation, while others are based on differential equations directly modeled according to the assumed underlying quantum-mechanical principles [Jonscher and de Polignac 1984; Lee et al. 2001]. However, to our knowledge, there is no straightforward model describing the concurrent process of simultaneous excitation and re-radiation as found in everyday situations.

In simple terms, phosphorescence is a photoelectric effect which is caused by *excitation* of a substance by photons and later de-excitation resulting in *re-radiation* of new photons (Figure 8.2 a) [Randall and Wilkins 1945; Leverenz 1968]. Upon excitation, molecules enter excited states, from which they eventually return to the ground state, possibly by a transition coinciding with the re-emission of new photons. The re-emitted photons typically have a longer wavelength compared to the photons causing the excitation (Stokes shift). (An exception is Raman scattering whose effect is however too weak to be of practical importance in rendering.) We opt for a state-based phenomenological model that omits any considerations not relevant for plausible reproduction of the appearance of common phosphorescent materials like effects occurring only at extreme temperatures or under special types of irradiation.

8.3.2 Overview of the State-based Model

We use a state-based model as depicted in Figure 8.2 b. Upon excitation (Figure 8.2 b, motion up), molecules enter one out of several states from which they return to the ground state (Figure 8.2 b, motion down). This second transition results in visible light and our computations are aimed at

determining this quantity. We did not observe the need to model other transitions to be closer to models used in physics.

Figure 8.2 c shows molecules in a discrete excited state as dot populations which vary over time and depend on the illumination. Instead of using discrete molecules it is however more convenient to model the *concentrations* of molecules in different states per unit volume of a substance. In the continuous case, each concentration takes values between 0 and 1, shown for a single state in Figure 8.2 d as a purple line. Its change over time (Figure 8.2 d, green) depends on illumination (Figure 8.2 d, blue) as well as on its current value to model saturation effects.

We formulate the relation between the current illumination and concentration and the resulting change of the latter as a *rate equation*. The stronger the illumination, the higher is the positive change of concentration. Negative changes of concentration, which simultaneously result in the re-emission of light, are due to the re-radiation. As the concentration approaches its maximum it cannot grow further (saturation). In the same way, when the concentration approaches zero, the re-radiation does, too. For a specific illumination, there is a specific concentration where the increase and decrease cancel each other out such that the appearance reaches an equilibrium. These effects are responsible for the nonlinearities depicted in Figure 8.2 d. Computing the concentrations means to solve differential equations, e. g., using numerical integration such as the forward Euler method.

In practice, the concentration is also spatially-variant and illumination itself is coupled with phosphorescence at other spatial locations described by integral equations which can be solved using light transport simulation, e. g., by Monte Carlo rendering. In summary, rendering phosphorescence amounts to solving a space-, time- and wavelength-varying system of integro-differential equations.

8.3.3 Rate Equations

Let the i-th of n_c concentrations at position $\mathbf{x}$ and time t be denoted as $r_i(\mathbf{x}, t)$ and the spectral irradiance as $E(\mathbf{x}, \lambda, t)$. We define the change of concentration as

$$\frac{\mathrm{d}r_i}{\mathrm{d}t}(\mathbf{x}, t) = \underbrace{(1 - r_i(\mathbf{x}, t)) \int k_{\mathrm{e},i}(\lambda)\, E(\mathbf{x}, \lambda, t)\, \mathrm{d}\lambda}_{\text{Excitation}} - \underbrace{k_{\mathrm{r},i} r_i(\mathbf{x}, t)}_{\text{Re—radiation}}, \tag{8.1}$$

where $k_{\mathrm{e},i}$ and $k_{\mathrm{r},i}$ are the *excitation* and *re-radiation* rates and $1 - r_i(\mathbf{x}, t)$ is the capacity remaining to full saturation. For Equation 8.1, which is a first-order ODE, to have a unique solution, initial conditions are necessary. This amounts to specifying all initial concentrations at the start of the simulation ($t = 0$) for all spatial locations $\mathbf{x}$, e. g., by assuming no previous excitation ($r_i(\mathbf{x}, 0) = 0$) or by fixing initial lighting conditions and running a pre-simulation to determine the resulting equilibrium state which can the be used for initialization.

While Equation 8.1 models the internal state of phosphorescent materials and its change over time, the visible light caused by the re-radiation is introduced into image synthesis as an additional isotropic emissive component in the rendering equation (Equation 2.9):

$$L(\mathbf{x}, \vec{\boldsymbol{\omega}}_\mathrm{out}, \lambda, t) = \underbrace{L_\mathrm{e}(\mathbf{x}, \vec{\boldsymbol{\omega}}_\mathrm{out}, \lambda, t)}_{\text{Emission}} + \underbrace{L_\mathrm{p}(\mathbf{x}, \lambda, t)}_{\text{Phosphorescence}} + \underbrace{\mathbf{R}L(\mathbf{x}, \vec{\boldsymbol{\omega}}_\mathrm{in}, \lambda, t)}_{\text{Reflection}}, \tag{8.2}$$

where L_e is the classic emissive radiance, L_p the new phosphorescent re-emission and $\mathbf{R}$ the shorthand reflection operator notation [Arvo et al. 1994] to convert incoming to outgoing radiance. The phosphorescent emission occurs at multiple wavelengths which we model as

$$L_p(\mathbf{x}, \lambda, t) = \sum_{i=1}^{n_c} \Lambda_i(\lambda) k_{r,i} r_i(\mathbf{x}, t), \tag{8.3}$$

where $\Lambda_i(\lambda)$ is the per-state *re-radiation* function that models how much state i re-radiates at wavelength λ. We discuss solving Equation 8.2 in Section 8.5.

8.3.4 Limitations

In our model, we make a few simplifying assumptions. First, we do not consider potential effects of the thickness of the phosphorescent substance and therefore can only capture the behavior for the specific material thickness used in the respective acquisition process (Section 8.4).

Second, we do not model directional dependency. This is in conjunction with the fact that all material samples examined by us behaved mostly isotropic in both excitation and emission. Measuring the emission from five different angles (0 to 80° in 20° steps) we found the fluctuations in intensity to be less than 5%. Layered materials [Hanrahan and Krueger 1993; Jakob et al. 2014] where a phosphorescent substance is covered by a glossy finish or the like might behave differently, which is however orthogonal and independent of the nature of phosphorescence.

Finally, we do not explicitly enforce Stokes shift as the state of the material, given by the state concentrations r_i, does not track the precise wavelength of the photons having caused excitation. Thus, technically, excitation from one wavelength may result in re-radiation at a shorter wavelength. Yet, the overlap of the non-zero regions of the excitation spectra, given by the $k_{e,i}(\lambda)$, and the re-radiation spectra, given by the $\Lambda_i(\lambda)$, is typically very small which largely excludes this possibility.

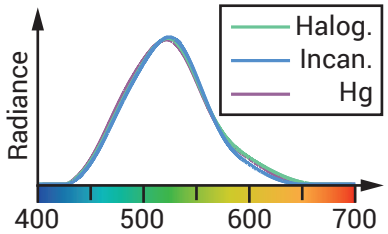


Figure 8.3: Re-radiation spectrum of phosphorescent stickers after excitation by a halogen lamp, an incandescent light bulb and a mercury vapor lamp. The curves have been scaled by their respective average.

We measured the re-radiation of our materials after excitation by several everyday light source types with different emission spectra and found that they were virtually the same: Whether excited by a lamp with smooth spectrum and strongest output in the red range (e. g., halogen light bulb) or by a gas discharge lamp with very thin blue and green spectral lines (e. g., mercury vapor), the resulting re-radiation is nearly indistinguishable regarding its spectral distribution as shown in Figure 8.3 and even more so when observed by a human with only three types of cone cells.

8.4 Acquisition

To reproduce a specific material using our model, we need to acquire the constants $k_{r,i}$ and $k_{e,i}$ (Equation 8.1) as well as Λ_i (Equation 8.3) by performing adequate measurements. We draw inspiration from existing acquistion processes for similar parameters for models that are suitable for describing but not for effectively simulating phosphorescence [Randall and Wilkins 1945].

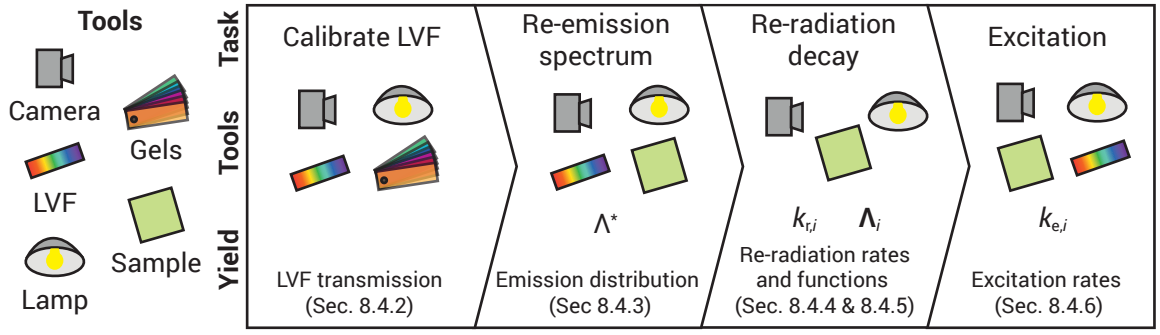


Figure 8.4: The acquisition workflow. After calibrating the LVF (Section 8.4.2) it is used to measure the re-emission's spectrum (Section 8.4.3). Next, the decay process is sampled which allows to determine re-radiation rates (Section 8.4.4) and functions (Section 8.4.5). Finally, excitation by different wavelengths is determined using the LVF to produce the different wavelength bands simultaneously, and excitation rates are fit (Section 8.4.6).

Typically, a sample is illuminated in a specific way, the response over time is captured and this observation is used to fit the parameters of the model.

After describing our general setup (Section 8.4.1) and spectral acquisition method (Section 8.4.2), we discuss how to measure the re-radiation's spectral distribution (Section 8.4.3), re-radiation (Section 8.4.4) and excitation rates (Section 8.4.6) as well as the final re-radiation functions (Section 8.4.5). Figure 8.4 gives an overview over the different acquisition steps and the respective parameters that are estimated. Our obtained parameters for various substances and a comparison to photographs are shown in Section 8.4.7. For brevity, we drop the spatial variation in this section and always consider a specific location $\mathbf{x}$.

8.4.1 General Setup

Our acquisition setup is seen in Figure 8.5. In general, it consists of a strong light source with known, adjustable spectrum such as a lamp with a smooth spectrum in combination with bandpass filters or a monochromator as well as an instrument to capture the emission spectra of the samples like a spectrometer or a simple digital RGB or gray camera combined with additional bandpass filter(s). We take the latter approach using a Canon 5D Mark II as detailed in Section 8.4.2. The camera's high resolution is beneficial as it allows for averaging measurements across multiple pixels reducing the effect of noise.

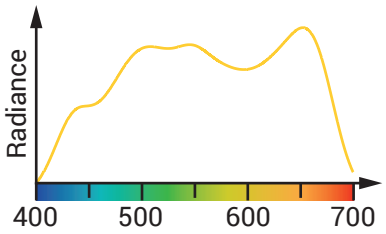


Figure 8.6: Spectral distribution of our halogen lamp. Note that values close to the ends of the camera's sensitivity range (400 to 700 nm) are less reliable.

We used a 400 W halogen lamp as our light source (Figure 8.6). For some measurements, we shaped the light's spectrum using a *linear variable bandpass filter* (LVF), as seen in Figure 8.5 b+d. The transmission spectrum of this type of filter corresponds to a narrow spectral band of about 25 nm where the center of the band varies linearly across the filter. The room was strictly dark otherwise. Xenon lamps constitute an alternative type of light source to combine with an LVF with the advantage of having stronger output in the near-UV range but also coming with drawbacks such as being more expensive, more tenuous to operate and having spikes in the emission spectrum.

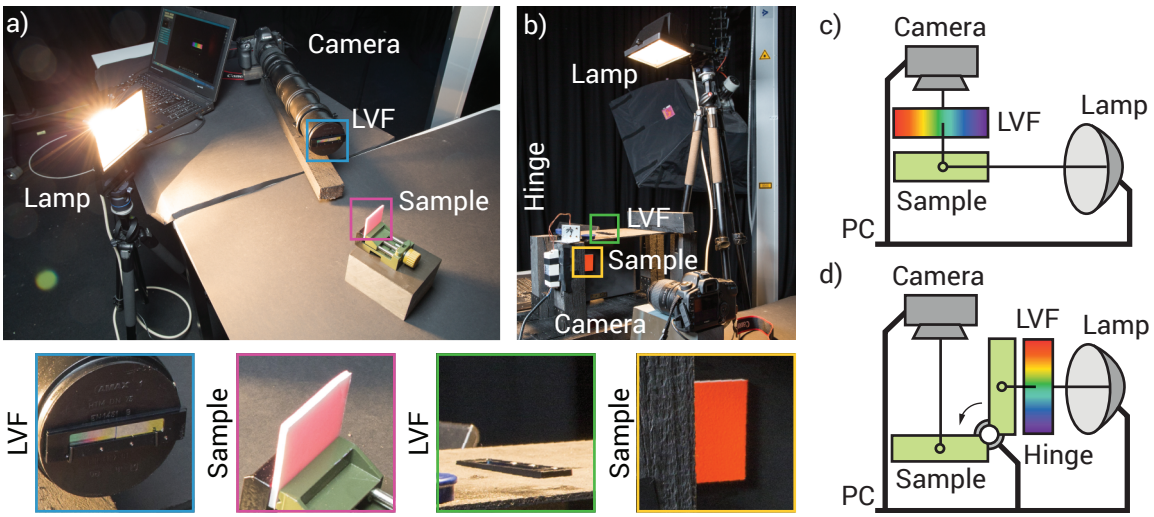


Figure 8.5: Our two measurement setups as described in Section 8.4.1.

During the actual measurements, we adjusted the camera settings (ISO and aperture) depending on the emission intensity of the different materials and afterwards performed a relative calibration of the resulting RGB values of the images taken to account for this. We deemed the camera's dynamic range sufficient to capture the samples' emission and felt no need for turning to high-dynamic-range imaging.

Before obtaining a measurement series, we allowed the lamp and camera to reach a stable temperature to eliminate temperature-dependent fluctuations in the sensor's noise level or the lamp's emission. For computing the fits in Table 8.1 we discretized all spectra into $n_\lambda = 8$ spectral bands, i. e., uniform intervals over the range from 400 to 700 nm.

8.4.2 Spectral Acquisition Using an LVF

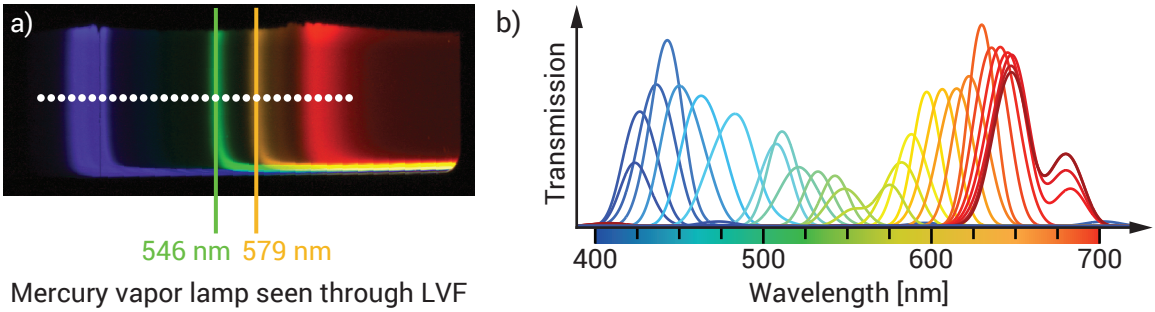


Figure 8.7: a): Spectrum of a mercury vapor lamp seen through an LVF. The spectral lines' wavelengths are known and hence used to select sampling points (white dots). b): Each line corresponds to the LVF's transmission at one of the sampling points highlighted in a).

Professional spectrometers are expensive. An alternative is taking RGB images of the respective sample using a camera with known spectral sensitivity through a number of cheap color filters with different, known transmission curves and solving the resulting system of linear equations to find the spectrum explaining the measured values best in a least-squares sense [Kirk and O'Brien 2011]. Unfortunately, phosphorescence is a time-varying effect which makes it necessary

to take these images at the same instant which is hard to achieve with common filters. An LVF however can be seen as a set of many filters with well-localized transmission curves in compact form and allows to take many differently filtered images at the same time (Figure 8.5 a+c), while still being less pricey than a spectrometer.

Before being able to take spectral images, the LVF has to be calibrated by computing its own spectral transmission curves at each spatial location. For this, we first identified the locations of the 400 and 700 nm bands on the filter by localizing the spectral lines of a mercury vapor lamp and extrapolating them linearly. Then, we picked a set of 31 equidistant sample locations from 400 to 700 nm (Figure 8.7 a). To determine the spectral transmission at these locations we first combined the LVF with each of a set of 108 Roscolux color filters and imaged a gray card with known reflectance which was itself lit by a white LED light with known spectral distribution. We then optimized for the most likely transmission curve at each of the 31 locations (Figure 8.7 b), analogously to [Kirk and O'Brien 2011]. During this and other LVF-based measurements we always averaged the observations (spatially) over all pixels corresponding to the same wavelength.

After calibrating the LVF, placing it between the camera and a sample (Figure 8.5 a+c) provides us with information about the RGB response for 31 different bandpass filters at the same time. This information can be used to optimize for the spectrum of the measured signal with appropriate precision, in the same way as during calibration of the LVF, just that now the signal is unknown while the filters spectral distribution is known.

Validation of the Spectral Acquisition

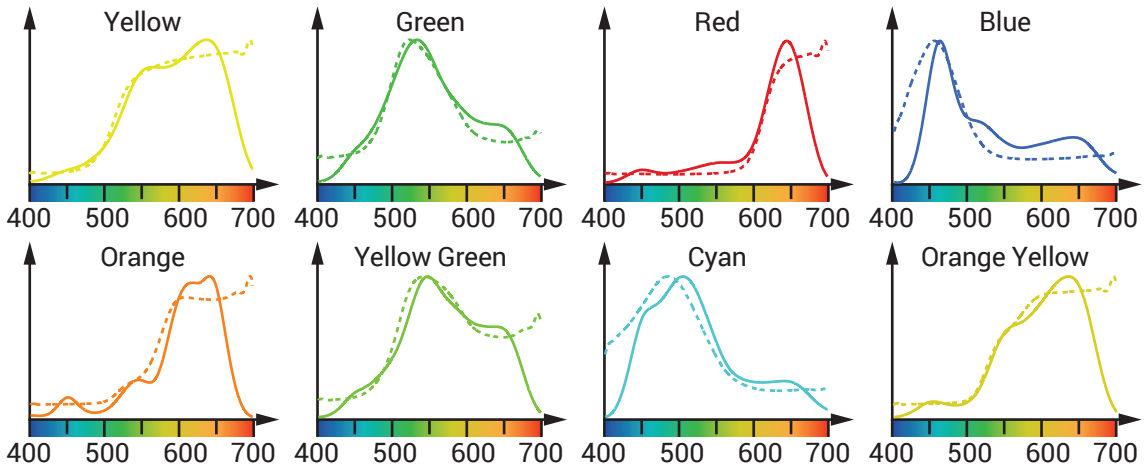


Figure 8.8: Measured (solid lines) and ground truth (dashed) spectral reflectance for a selection of color checker patches. The horizontal axis corresponds to wavelength [nm].

We validated our spectral acquisition method by imaging patches of a GretagMacbeth color checker with known reflectances. In Figure 8.8 we compare measured spectra to the ground truth distributions. The spectra are generally reproduced well, problems only arise at the ends of the spectrum were the sensitivity of the used digital camera vanishes. This limitation is shared with other approaches also using consumer cameras.

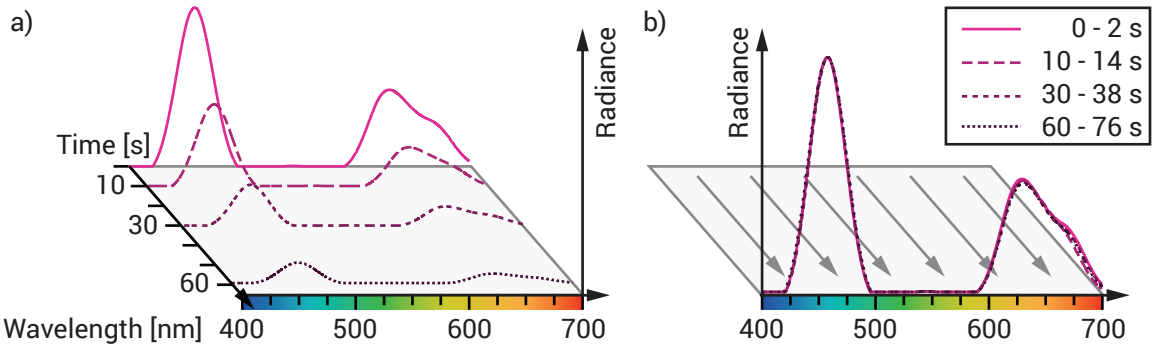


Figure 8.9: The spectral distribution of the emission is constant over time and only scaled globally (a). This can be seen more clearly when dividing the curves by their maximum (b). Also note the increasing integration times (box) necessary to capture images with a similar signal to noise ratio.

8.4.3 Re-radiation Spectrum

To instantiate our model, the spectral emission of a material has to be captured over time. A problem with this is that capturing the time-dependent behavior demands for dense temporal sampling, while the spectral acquisition asks for low noise images which in turn require long exposure times given the low intensity nature of phosphorescent emission. These two requirements contradict each other. We however found that the relative spectral intensity of our material samples was constant over time as can be seen for one sample in Figure 8.9. This allows to decouple the measurement of the relative distribution from the measurement of the change of absolute intensity over time, yielding a more robust fit. For these reasons, we first determine the spectral distribution $\Lambda^*(\lambda)$ of the emission separately and normalize it to have a maximum value of 1 (Figure 8.12).

8.4.4 Re-radiation Rate

By measuring how emission decays after turning the illuminant off, we can estimate the re-radiation rates. In the absence of illumination, i. e., $E(\lambda, t) = 0$, Equation 8.1 has the closed form solution

$$r_i(t) = r_i(0) \cdot e^{-k_{\mathrm{r},i}\, t}. \tag{8.4}$$

Thus, sampling $L_\mathrm{p}(\lambda, t)$ (Equation 8.3) we can fit the underlying re-radiation rate directly constant.

Procedure Initially, we illuminate the sample by a strong all-frequency light for at least 90 minutes (Figure 8.5 a+c). This time is typically sufficient to achieve saturation so that we can assume that $r_i(0) = 1$ for all i which is useful when fitting the re-radiation functions later (Section 8.4.5). Next, we turn off the light and sample $L_\mathrm{p}(\lambda, t)$ by taking RGB photographs at pre-defined times, typically between $t = \epsilon$ and $t = \epsilon + 180\,\mathrm{s}$, where ϵ is the delay between turning off the light and starting the measurement which we keep to a constant value of a few milliseconds by operating lamp and camera using a PC. Special care has to be taken if the lamp exhibits an afterglow. In this case the lamp has to be automatically shielded to avoid a reflection

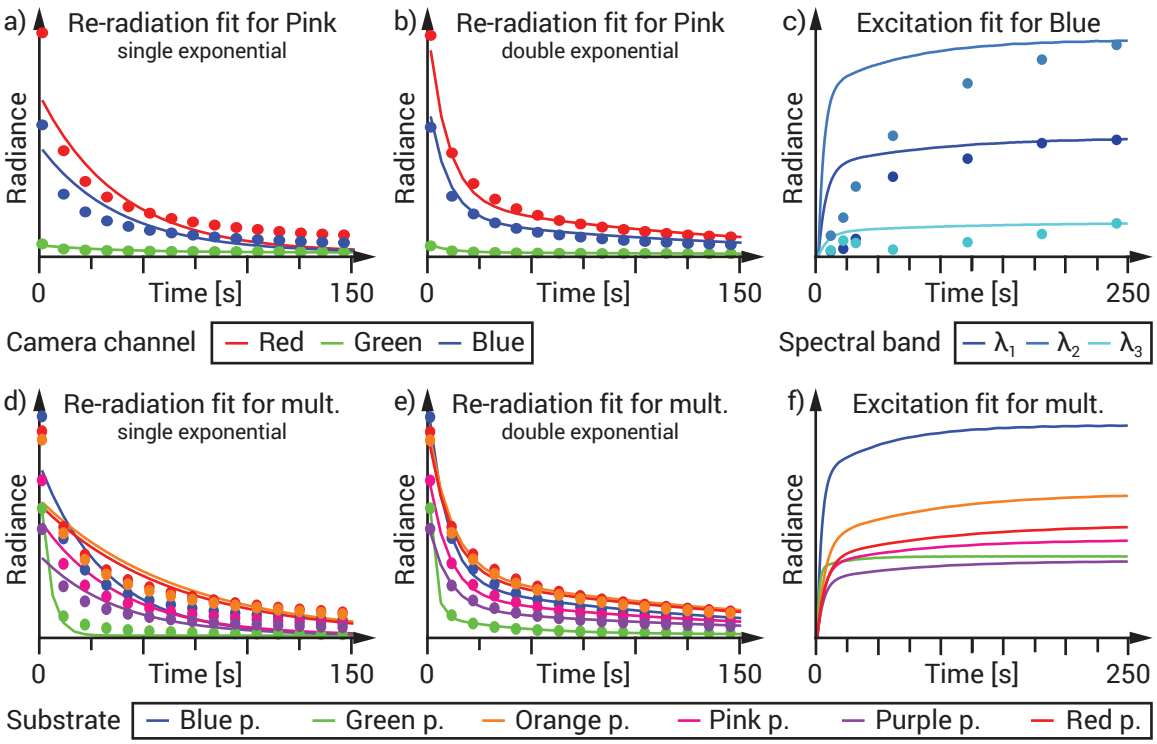


Figure 8.10: Measurements and fits. *a):* Re-radiation measurements *(points)* fitted by one exponential *(lines)* for "Pink paint". *b):* A double exponential fitted to the same data. *d), e):* Decay fits for various materials using one or a sum of two exponential functions, respectively. *c):* Excitation measurements and ODE simulation *(Section 8.4.6)* for "Blue paint" and for others *(f)*.

of the afterglow by the sample. We found taking one image every 5 seconds to be sufficient. Figure 8.10 a+b show the resulting curves for a sample material as points.

Fitting As the spectral distribution is constant over time (Section 8.4.3), the measured RGB values are multiples of the camera's RGB response to the spectrum given by $\Lambda^*(\lambda)$, which we denote by $\mathbf{\Lambda}^*_{\mathbf{3}} \in \mathbb{R}^3$. We (simultaneously) fit a sum of n_c exponential decay functions of the form

$$\sum_{i=1}^{n_\mathrm{c}} a_i \cdot \mathbf{\Lambda}^*_{\mathbf{3}j} \cdot e^{-k_{\mathrm{r},i} t}$$

to the observed RGB values of $L_\mathrm{p}(\lambda_j, t)$ for $\lambda_j \in \{1,2,3\}$ using Matlab's constrained nonlinear optimization functionality, restricting the parameters to the positive range. Constancy of the spectral distribution implies the single scaling factor a_i and a single decay rate $k_{\mathrm{r},i}$ per exponential term which are shared among all wavelengths. Note that while fitting a single exponential function corresponds to fitting a line in log-space, this cannot be generalized to sums of exponential functions.

Discussion An exemplary fit for one material is shown in Figure 8.10 a+b. Additional individual plots are included in Appendix A. Fits for multiple other materials are shown in Figure 8.10 d+e and the corresponding parameter values are listed in Table 8.1.

Decay	Material	k_r	k_e λ_1	λ_2	λ_3	Λ λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8
	Blue	0.032	0.0170	0.0500	0.0040	0.34	4.37	3.65	0.55	0.0	0.01	0.0	0.0
	Green	0.227	0.1000	0.0350	0.0060	0.0	0.0	0.07	0.64	0.05	0.0	0.0	0.0
	Orange	0.015	0.0150	0.0060	0.0003	0.04	1.85	0.47	0.0	0.45	7.70	4.88	2.49
	Pink	0.027	0.0060	0.0150	0.0033	0.59	4.07	0.49	0.0	0.0	1.22	1.77	0.09
	Purple	0.028	0.0120	0.0180	0.0008	0.46	3.08	0.36	0.0	0.0	0.19	0.08	0.0
	Red	0.016	0.0120	0.0038	0.0000	0.13	3.03	0.44	0.0	0.02	5.49	6.21	0.63
	White	0.047	0.0180	0.0400	0.0058	0.26	2.55	2.49	0.61	0.0	0.01	0.0	0.10
	Yellow	0.251	0.1450	0.0380	0.0070	0.0	0.0	0.08	0.80	0.15	0.12	0.02	0.0
	Bottle	0.185	-	-	-	0.0	0.04	0.41	0.58	0.25	0.06	0.01	0.0
	Gloves	0.067	-	-	-	0.10	0.88	5.29	5.28	2.01	0.66	0.12	0.01
	Stickers	0.039	-	-	-	0.01	6.02	40.75	40.66	14.86	5.08	1.11	0.14

Table 8.1: Re-radiation and excitation rates in units s^{-1}. The first column visualizes the decay gradients (scale of 60 s). The response for some wavelengths was too low to be measured with our setup, thus excitation rates are only given for the first three wavelength bands. For the last three materials, only their decay behavior was measured.

We found that $n_c = 2$ produces a good fit for all measured materials (cf. Figure 8.10 b). As outlined in Section 8.3, such a "double exponential" fit for decay curves is common in physics [Chen et al. 2011] with explanations including multiple trap states [McKeever and Chen 1997]. It is worth noting that stretched exponentials [Lee et al. 2001] or power law decays [Jonscher and de Polignac 1984] provide different interpretations leading to different ODEs which however contain the time since illumination as a term, defying efficient simulation in general conditions. A fit using $n_c = 1$ provides an approximation with larger numerical error but still resembles typical phosphorescent behavior (Figure 8.13). The variety of shapes in Figure 8.10 d+e shows that different substances have different decay behavior that can be substantially different from a single exponential.

Numerically, using a sum of two (three) exponential functions compared to only one, the median approximation error of the fit over all materials is reduced to 8% (4%) of its original magnitude. The main improvement of the fit concerns the long-term error: The error for the last time sample is reduced to 18% (3%). We however found $n_c = 2$ to be a good compromise as adding one exponential more would also roughly increase evaluation time by 50% (Section 8.5).

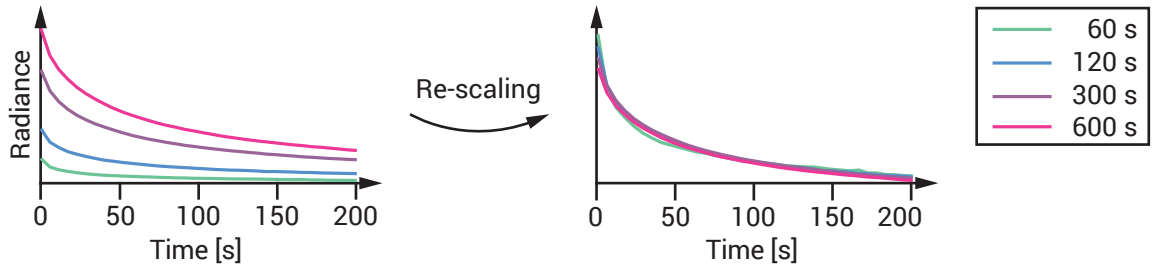


Figure 8.11: The decay curves from different excitation levels (left) are the same except for scaling. Undoing the scaling (right) brings them to alignment. Remaining differences can be attributed to slightly different delays between switching the light off and starting the measurement.

By measuring the decay from different excitation levels, we verified that the shape of the curve is independent of the excitation state i. e., that the $k_{r,i}$ are the same for all curves (Figure 8.11).

Also, constancy of the spectrum (Figure 8.9) indicates that a single $k_{\mathrm{r},i}$ shared by all wavelengths as used in our model is appropriate.

8.4.5 Re-radiation Function

The re-radiation function $\Lambda_i(\lambda)$ models to which extent excitation in state i results in re-radiation at wavelength λ. For n_λ discrete wavelength bands, the operator becomes a vector $\mathbf{\Lambda}_i \in \mathbb{R}^{n_\lambda}$. As outlined in the following, we can re-use the data acquired during the decay acquisition (Section 8.4.4) without conducting further measurements.

Fitting First consider $L_\mathrm{p}(\lambda_j, t)$ during the experiment conducted in Section 8.4.4 and denote the re-radiation vectors in this case as $\mathbf{\Lambda}_{\mathbf{3}i} \in \mathbb{R}^3$. With $n_\lambda = 3$, for a given position, Equation 8.3 becomes

$$L_\mathrm{p}(\lambda_j, t) = \sum_{i=1}^{n_\mathrm{c}} \mathbf{\Lambda}_{\mathbf{3}i,j} k_{\mathrm{r},i} r_i(t) \tag{8.5}$$

$$= \sum_{i=1}^{n_\mathrm{c}} \mathbf{\Lambda}_{\mathbf{3}i,j} k_{\mathrm{r},i} e^{-k_{\mathrm{r},i} t} = \sum_{i=1}^{n_\mathrm{c}} a_i \cdot \mathbf{\Lambda}^*_{\mathbf{3}j} \cdot e^{-k_{\mathrm{r},i} t}. \tag{8.6}$$

As in Equation 8.6, we have an equality between two sums of exponential functions, we can compare their coefficients which have to be identical for the equality to hold i. e.,

$$\mathbf{\Lambda}_{\mathbf{3}i,j} k_{\mathrm{r},i} \stackrel{!}{=} a_i \cdot \mathbf{\Lambda}^*_{\mathbf{3}j}.$$

This yields

$$\mathbf{\Lambda}_{\mathbf{3}i,j} = a_i \cdot \mathbf{\Lambda}^*_{\mathbf{3}j} / k_{\mathrm{r},i}.$$

We obtain the spectral $\mathbf{\Lambda}_{i,j}$ by applying the scaling found using the RGB case to the discretized version $\mathbf{\Lambda}^*$ of $\Lambda^*(\lambda)$, i. e.,

$$\mathbf{\Lambda}_{i,j} = a_i \cdot \mathbf{\Lambda}^*_j / k_{\mathrm{r},i}.$$

Discussion The second equality holds since $r_i(0) = 1$ and the illumination is turned off (i. e., under the conditions of the experiment in Section 8.4.4). As in general $a_i \neq a_k$ for $i \neq k$, one $\mathbf{\Lambda}_i$ per state is necessary to reproduce the measured emission behavior, where all $\mathbf{\Lambda}_i$ are however scaled versions of the per-material $\mathbf{\Lambda}^*$.

8.4.6 Excitation Rate

After the previous experiments, re-radiation rate and function are known. They describe how fast a material loses excitation and the spectrum of the resulting emission. To complete the parameter set, what remains to be determined is the excitation rate, i. e., how fast excitation is gained.

Conceptually, we need to observe the change of the phosphorescent intensity of an initially unexcited sample being exposed to a constant flux of a certain wavelength band until equilibrium is reached. Using the other, already known, parameters, we could then fit the excitation rate in Equation 8.1 so that the shape of the curve and above all the equilibrium level are matched

best. However, in practice the phosphorescent emission can only be measured when there is no significant irradiance as otherwise emission is dwarfed by simple reflection of the sample. As an approximation, we measure the phosphorescent emission directly after the illuminant is switched off.

Procedure Before we can start the measurement, we have to keep the sample in darkness for several hours until no emission can be detected anymore to bring the sample into a known internal state ($r_i(t) = 0$). Next, we expose it to an illuminant of a specific wavelength band for t seconds (Figure 8.5 b+d), then switch off the latter and measure the emission immediately to obtain an estimate of $L_\mathrm{p}(t)$. We execute this process for all n_λ wavelength bands and different excitation times. As the time for the excitation to reach an equilibrium is typically unknown, we recommend to initially double the excitation time with each sample until emission does not change anymore before selecting further intermediate sampling times as desired.

This procedure is more challenging to perform than the previous ones. While re-radiation can be easily measured in a single run, with linear effort in the number of samples, to obtain emission after time t, an uncharged sample has to be previously charged for precisely this amount of time (quadratic effort). To accelerate the process, we perform the experiment for all wavelengths in parallel by illuminating the sample through an LVF placed directly on top of the sample so that different spatial locations correspond to different excitation wavelengths. To quickly remove the LVF from the sample in order to take the actual measurement we built an automated trap door-like mechanism in which the sample is swung away from the LVF and directed towards the camera, before releasing the shutter, immediately when the light is turned off (cf. Figure 8.5 b+d). In our setup, the light takes about 320 ms to fade, then the door begins to open and stops 280 ms later.

Fitting As the excitation rate is wavelength-dependent we perform the fit for the n_λ different wavelength bands independently using the respective excitation-curve measured for light of that band. We choose the excitation rate $k_{\mathrm{e},i}(\lambda_j)$ for band j using Matlab's nonlinear least-squares regression such that when simulated using the measured spectrum E of the light source and the already determined $k_{\mathrm{r},i}$ and $\Lambda_i(\lambda_j)$ it explains the sampled excitation curve best.

Note, that in our case the sample curve consists of (linear) RGB values measured by the camera and in order to be able to compare them to the spectral simulation results the latter have to be mapped to RGB, too, using the camera's spectral sensitivity curves. We used [Kawakami et al. 2013] as a source for the curves for our camera model. For numerical reasons, we normalized the length of E to 1 in the simulations instead of scaling it correctly relative to the samples of L_p (i. e., performing an absolute calibration) to avoid tiny values of $k_{\mathrm{e},i}$ which would only complicate reproduction. The resulting samples and a fit using $n_\mathrm{c} = 1$ is seen in Figure 8.10 c. The spectrum of the light source is shown in Figure 8.6.

Discussion Our excitation measurement process bears two potential sources of imprecision: the short delay when moving the sample and camera noise. Furthermore, slight fluctuations of the aforementioned delay and vibrations of the sample when our trap door stops may lead to inconsistencies. To further improve the precision it is possible to account for the decay during the delay before the actual photograph is taken.

Comparing Figure 8.10 a and Figure 8.10 c it can be seen that the re-radiation fit is precise while

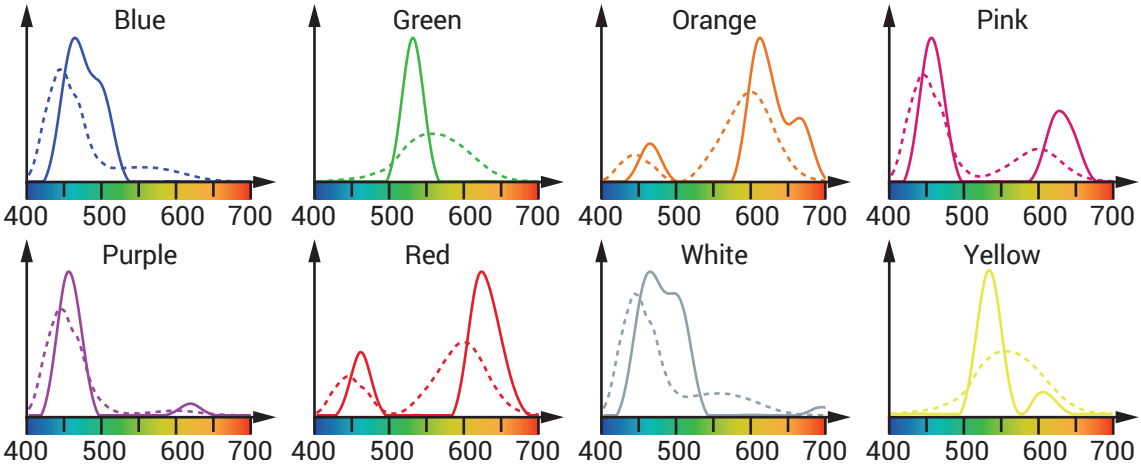


Figure 8.12: Spectral intensity distributions (solid lines) of the samples shown in Figure 8.13, measured according to Section 8.4.2. Relative spectral radiance is plotted for varying wavelength [nm]. We also show the corresponding best approximations as a weighted sum of the CIE2006 color matching functions (dashed) as an indication for how well the spectra can be represented in a three channel color space. The spectral acquisition captures the localized peaks better.

the excitation fit reaches the correct equilibrium but at a faster rate than measured. Despite the imprecision in the measurement we attribute this mostly to the model itself. One reason for this is that we fit the re-radiation first to match it as well as possible as differences in the re-radiation are very noticeable when observed in darkness. The speed at which the (correct) equilibrium is reached during excitation is of lesser importance as, while the illuminant is switched on, the phosphorescent emission is dwarfed by reflection and the process of excitation cannot be observed by the naked eye.

8.4.7 Results

We acquired parameters for 11 phosphorescent materials from special paints to everyday objects. As previously described, after initially measuring the emission spectra at a 10 nm resolution using the calibrated LVF (Section 8.4.2) we used $n_\lambda = 8$ wavelength bands of 37.5 nm width, from 400 to 700 nm for the remaining fit. The outcome is shown in Table 8.1 and Figure 8.12. For the parameters to a double exponential model see Appendix A.

Figure 8.13 compares synthetic images rendered using the acquired data according to Section 8.5 to actual photographs. The most prominent differences appear for the yellow and green paints which fade particularly fast before continuing to glow at low brightness. Especially the fit with only one exponential fails to capture this dualistic behavior while the double exponential fit can cover both short and long-term behavior. For some materials their albedo is close to their re-radiation spectrum which is however no general rule as can be seen e. g., for the “white” and “stickers” materials (Table 8.1).

The acquisition process might appear to be very time consuming but this is only true if only one physical material sample is available. Determining the emission spectrum (Section 8.4.2) merely requires a single photo at a sufficient excitation level. For the re-radiation acquisition, exciting the sample to saturation usually takes half an hour and the following measurement only takes about 3 minutes in which the decay curve is sampled. For the excitation rates, if a sufficient

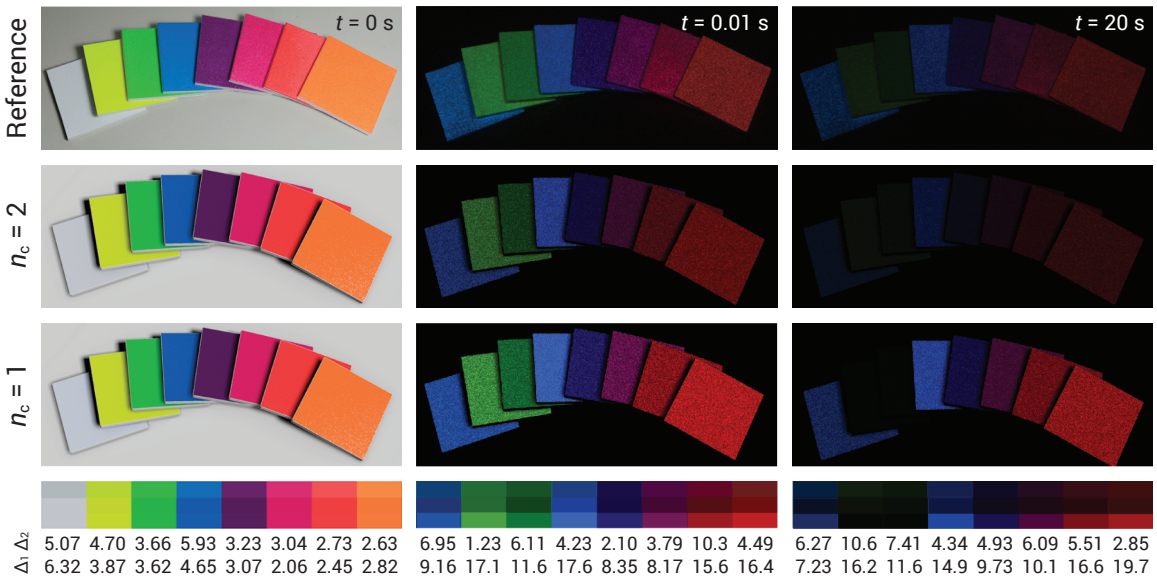


Figure 8.13: Materials acquired with $n_\lambda = 8$ and using single ($n_c = 1$) and double exponential ($n_c = 2$) fits compared to photographs. For display and to enable comparisons, spectral re-emission is mapped back to RGB using the camera sensitivity curves. The actual emission spectra are displayed in Figure 8.12. The CIEDE2000 differences to the reference for $n_c = i$ are labeled with Δ_i. The avg. / min. / max. error over all samples and points in time amounts to 5.46 / 1.23 / 10.64 for the double- and 13.16 / 7.24 / 19.75 for the single-exponential fit.

number of samples is at hand (about 10 per material), it is not necessary to wait for de-excitation of the samples and the total time is reduced to not more than two hours per material.

8.5 Reproduction

To reproduce phosphorescence Equation 8.1, an ordinary differential equation, has to be solved. As the irradiance E appears in Equation 8.1 this also requires solving the time-stationary rendering equation which itself depends on the emission from previous points in time.

Representation To solve Equation 8.1 we need a spatial discretization of the concentration r_i. For this, the scene surfaces are sampled into a point cloud. This sampling pattern is held constant but deformed in every frame assuming isometric deformations of objects. A particularly efficient sampling of this form avoiding explicit sample locations is produced by parameterizing the phosphorescent surfaces and associating one sample with each texel of a texture. For our results, we generated UV maps for all phosphorescent objects using the Blender 3D modeling software. The model parameters — re-radiation and excitation rates as well as emission spectra — can either be stored per material or, if spatial variation is desired, in textures.

Solver We perform incremental Euler integration, starting at frame 1. At each sample position, we store n_c floats in the range $[0, 1]$ to maintain the r_i which are initialized to 0. Let $\mathbf{r}_i^{(j)}$ denote the value of r_i at frame j. Frame $j > 1$ is produced as follows. First the irradiance $\mathbf{E}^{(j)}$ is approximated using any classic (spectral) rendering method and stored as an n_λ-component

vector. The approximation may include the current phosphorescent emission (as in Equation 8.2) or not. Next, the change of ratio $\dot{\mathbf{r}}_i^{(j)}$ is computed according to Equation 8.1 using $\mathbf{E}^{(j)}$ and the per-material or per-texel parameters. Finally, $\mathbf{r}^{(j)}$ is updated by $\mathbf{r}_i^{(j+1)} = \mathbf{r}_i^{(j)} + \Delta_t \dot{\mathbf{r}}_i^{(j)}$, where Δ_t is the time step size.

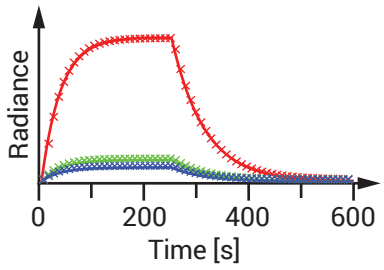


Figure 8.19: Runge-Kutta (lines) vs. Euler (crosses). The different lines correspond to RGB of the final result.

Discussion We use Euler integration to solve Equation 8.1 as we found no need to use a more advanced solver after comparing to other solvers such as Runge-Kutta: Figure 8.19 shows the change of phosphorescent emission in a typical situation where the light is first switched on and then turned off. As can be seen from the plot, simple Euler integration delivers nearly identical results compared to the more sophisticated Runge-Kutta (2,3) method employed by Matlab's ode23 solver. Using a different solver would be straightforward, though.

Our discretization in time is synced to the rendering speed. We observed this to be sufficient for the materials and lighting conditions we used. Even the highly space-time discontinuous behavior for animated scenes such as Figure 8.16 is captured well. The spatial discretization of $\mathbf{r}_i$ has to be fine enough to capture the spatial variation of illumination and phosphorescence parameters.

Results We implemented the solver described above using OpenGL shaders. As an example, for Figure 8.1 4.7 M sampling points realized by textures were used. One step of the phosphorescence simulation required 3.4 ms on an Nvidia Geforce GTX 770, including the time for computing the irradiance from direct light using shadow maps. The results in Figure 8.1, Figure 8.14, Figure 8.16 and Figure 8.20 have been post-processed to emulate scotopic vision by the method of Thompson et al. [2002], i. e., applying slight amounts of blur and noise to enhance their nocturnal appearance.

A comparison of real and reproduced phosphorescence for complex objects is shown in Figure 8.14. Figure 8.16 demonstrates the approach for real-time animations. Figure 8.15 shows global phosphorescent light transport. Finally, we demonstrate two possible extensions to volumes: Figure 8.17 shows frames of a fluid simulation based on smoothed-particle hydrodynamics where each particle represents one sample location and tracks its own current level of excitation. Figure 8.18 depicts an animated cloud of phosphorescent dust which is hit by laser sheets. This simulation is grid-based using one sample per grid cell.

Comparison to Monte Carlo Rendering The only previous model for rendering phosphorescence is due to Glassner Glassner [1995]. Since it is given as a closed-form formula, sampling irradiance over previous points in time is necessary to determine the current phosphorescent emission. This either requires an irradiance cache that grows linearly with simulation length or re-sampling of the irradiance in each time step leading to a sampling effort for the phosphorescence component which grows quadratically with simulation length. Furthermore, the evaluation of the decay cannot be cached and also leads to quadratic effort in time. Either of these options defies use in interactive applications (Figure 8.16) while for our method memory consumption and effort per frame are constant.

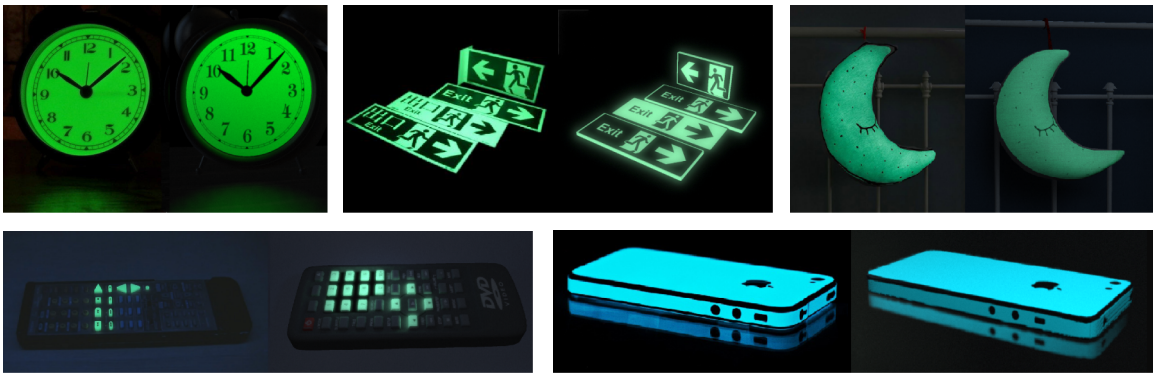


Figure 8.14: Photographs of phosphorescent objects compared to results rendered using our approach with designed materials.

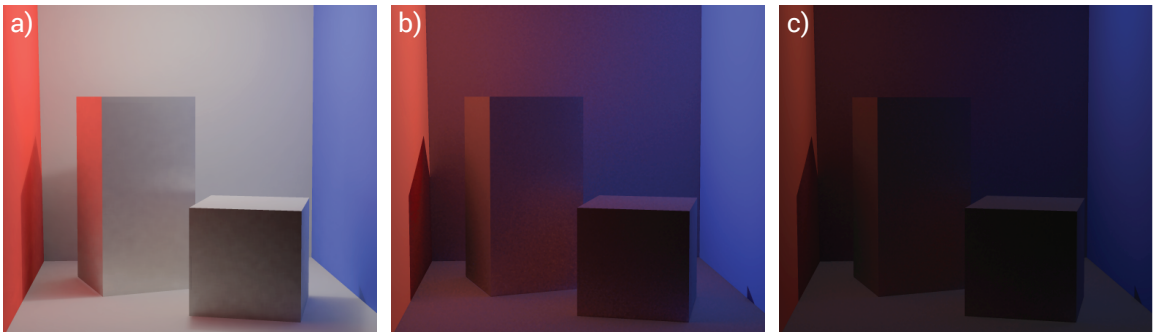


Figure 8.15: Path-traced Cornell box with phosphorescent red and blue walls and non-phosphorescent boxes: Initial illumination (a) and after the light has been turned off for 0.1 (b) and 60 s (c), respectively. Non-phosphorescent areas are lit by bounced phosphorescent emission.

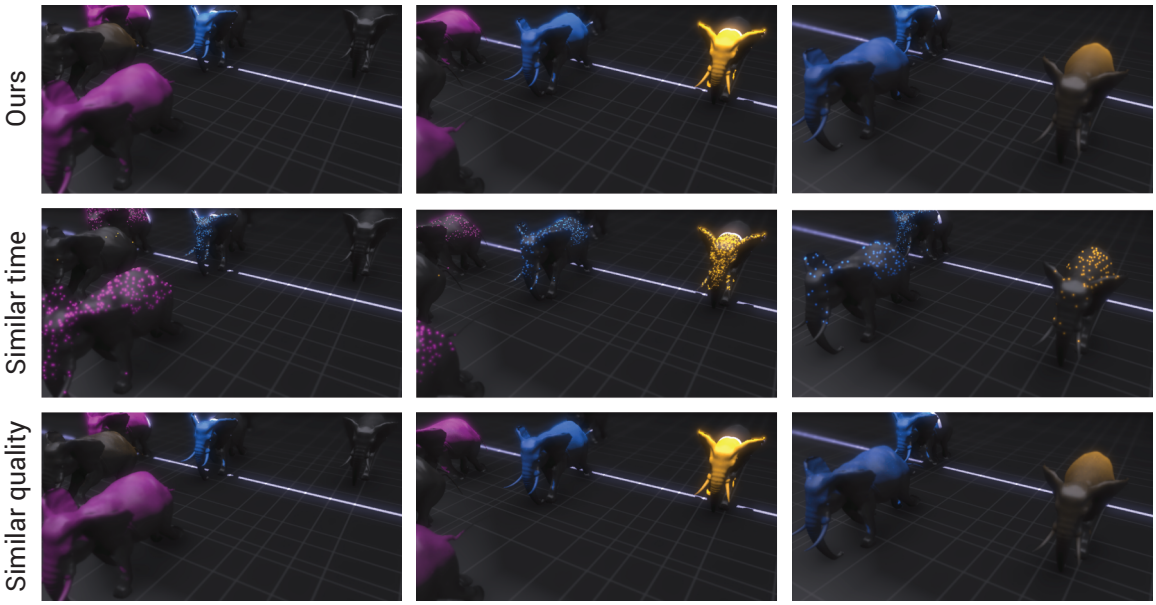


Figure 8.16: Animation (1024 × 512 px) computed using our approach (top, 142 fps) and a MC method based on [Glassner 1995], using temporal importance sampling according to the decay function, running in the same time (middle, 1 time sample per pixel) and the same quality (bottom, 1024 tspp).

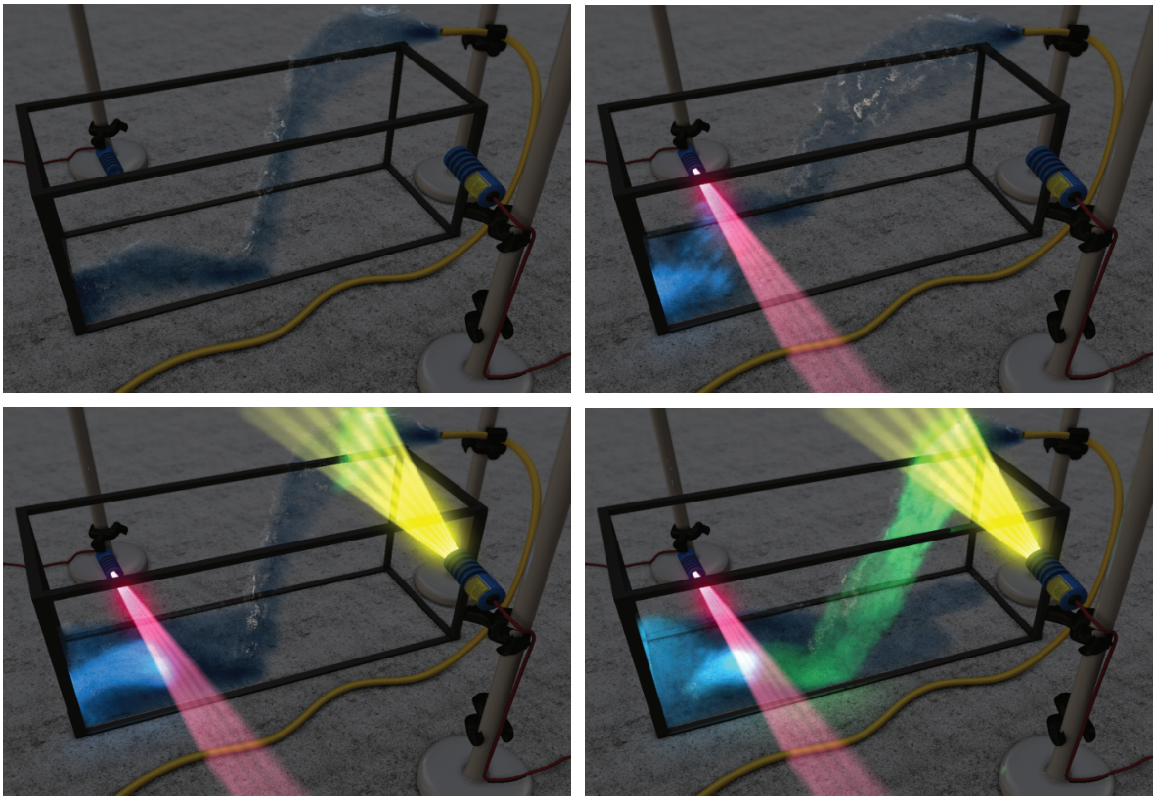


Figure 8.17: Our model also covers hypothetical liquids with re-radiation depending on the exciting wavelengths. This sequence was rendered using a particle-based simulation, tracking per-particle concentrations r_i.

8.6 Manipulation

Tweaking the parameters of our model to achieve a desired appearance is tedious for novice users. While we provide presets for a selection of typical phosphorescent materials (Table 8.1) users may also wish to create novel or non-physical materials. To this end, we propose to automatically adjust [Schoeneman et al. 1993] the parameters of our model to match user-provided constraints. Different from classic inverse lighting applications constraints need to be provided in both space and time.

We will describe the manipulation for a single spatial location $\hat{\mathbf{x}}$. For multiple locations, the procedure is performed independently. The user first specifies appearance constraints, i. e., outgoing radiance values $\hat{L}_i$ for a small number n_u of key frames corresponding to times $\hat{t}_1, \ldots, \hat{t}_{n_\mathrm{u}}$. In our manipulation tool, we use a discretization into three channels ($n_\lambda = 3$) and a single concentration ($n_\mathrm{c} = 1$) to reduce the number of constraints a user needs to provide.

We now would like to find an ordered set of parameters $\theta = \{\hat{\mathbf{\Lambda}}, \hat{k}_\mathrm{e}, \hat{k}_\mathrm{r}\}$:

$$\theta = \operatorname*{arg\,min}_{\theta' \in \mathbb{R}^7} f(\theta'), \quad f(\theta') = \sum_{i=1}^{n_\mathrm{u}} \left\| L^{\theta'}(\hat{\mathbf{x}}, \hat{t}_i) - \hat{L}_i \right\|_2^2, \tag{8.7}$$

where $L^{\theta'}$ is the outgoing radiance for phosphorescence parameters chosen according to θ'. The parameter set has seven degrees of freedom: two RGB values $\hat{\mathbf{\Lambda}}$ and $\hat{k}_\mathrm{e}$ and a scalar $\hat{k}_\mathrm{r}$. Note,

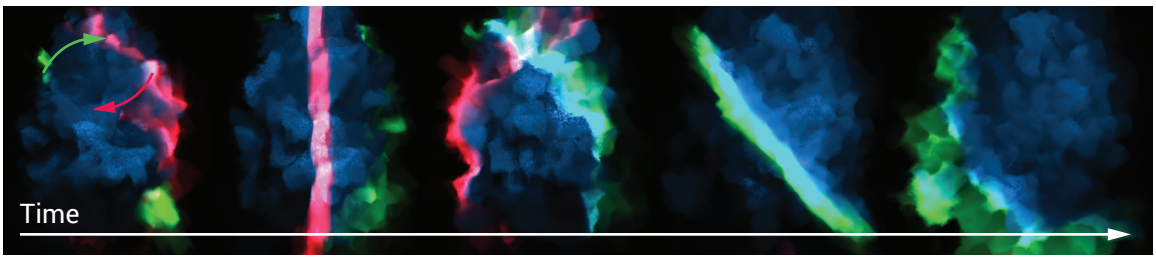


Figure 8.18: Laser sheets rotating around an animated phosphorescent dust cloud. The green sheet leads to strong excitation while the red one has little effect.

that the data we fit to are provided for an arbitrarily varying illumination defined by the user. This is more general than during the acquisition process (Section 8.4) where we chose specific illumination conditions to allow for controlled fitting.

Equation 8.7 is minimized in three steps. First, the irradiance $\hat{E} = E(\hat{\mathbf{x}})$ is computed and stored for all frames up to the time $\hat{t}_{n_u}$ of the last constraint. Effects of global light transport of phosphorescent emission are ignored for simplicity. Then, we sample a large number of typical parameter sets and for each parameter set θ' the outgoing radiance $L^{\theta'}$ consisting of both reflected light and phosphorescent re-emission is computed using the pre-computed $\hat{E}$ by numerical integration of Equation 8.1 up to the last key frame. The cost $f(\theta')$ is evaluated by summing over all key frames. As these computations are independent for all θ' they are well-suited for a parallel implementation on a GPU. Finally, the parameter set with the lowest total cost is selected using parallel minima reduction. The time to pre-compute E depends on how light transport is simulated and on the length of the sequence, the optimization itself is performed interactively.

After finding the best parameters θ for the spatially-sparse constraints we can propagate them to the rest of the scene using edit propagation [Pellacini and Lawrence 2007]. Examples of manipulation of phosphorescent appearance are shown in Figure 8.20.

Sampling the Parameter Sets

Parameter	a	$b_{\min}$	$b_{\max}$
k_r	0.001	0	5.0
k_e	0.001	0	12.0
$\mathbf{\Lambda}$	0.1	0	9.0

Table 8.2: Parameters used in the computation of sample parameter sets. For k_e and Λ, which are vectors, the given values are with respect to one component.

We sample our parameter sets on a high-dimensional grid. In each dimension, the grid typically covers 10 different possible values which are chosen on a logarithmic scale as $a \cdot 2^b$, where a is a parameter-dependent scale and the exponent b is sampled equidistantly from a parameter-dependent range $[b_{\min}, b_{\max}]$. We chose these ranges to cover the materials we measured. Table 8.2 contains the values of a, $b_{\min}$ and $b_{\max}$ which we used for our manipulation example.

Figure 8.20: A user places spatio-temporal appearance constraints (dots, top). We find the parameter set that reproduces them best while conforming to the model. Rendering the animation again yields an appearance consistent with the constraints (bottom). The manipulation is performed interactively and modifying one constraint with 4 key frames out of a 200 frame sequence requires 80 ms.

8.7 Comparison to Previous Work

The area of computer graphics has seen little previous work on the specific phenomenon of phosphorescence. Nevertheless, different aspects of our pipeline like acquisition or manipulation have been treated in similar ways within different contexts.

Rendering Rendering of fluorescence and phosphorescence effects was introduced by Glassner [1995]. However, his model does not account for saturation effects and it is unclear how to handle new excitation: materials just have an initial amount of emission which decays over time. No specific image generation approach was proposed, limiting the reproduction method to Monte Carlo rendering. We would argue that a MC approach is not a particularly good fit for phosphorescence. It requires to evaluate an integral over the additional, non-separable temporal domain, roughly resulting in a rendering cost per image similar to the cost of rendering an entire image sequence. In our same-time comparisons (Figure 8.16), MC has consequently shown to produce a substantial amount of noise. Our approach handles varying excitation in dynamic scenes, includes saturation and allows for efficient simulation of re-radiation with an only moderate computational overhead (milliseconds in our GPU implementation), independent of the animation length.

The inelastic scattering framework by Gutierrez et al. [2005] extends rendering of effects where light is re-radiated at different wavelengths to participating media. Analogously to Glassner's full radiance equation, a full radiative transfer equation is derived. However, no

time-dependent effects such as phosphorescence are supported. Using our method, we can synthesize phenomenologically-based but visually convincing re-radiating volumes (Figure 8.17 & Figure 8.18).

Acquisition Fluorescence of materials can be measured using special tools like Goniofluorometers [Bendig et al. 2008; Holopainen et al. 2008]. Wilkie et al. [2006] and Hullin et al. [2010] acquired bi-spectral, bi-directional reflection and re-radiation distribution functions for several materials. A high-resolution hyper-spectral imaging system including support for measuring fluorescence was proposed by Kim et al. [2012]. Phosphorescence was not addressed in either of the works. Our additional measurements go beyond the spectral shift alone to cover temporal aspects.

In concurrent work, Alvarez-Cortes et al. [2016] propose to use diffraction filters to image spectral distributions of light sources at low cost while we are using linear variable bandpass filters (LVFs) for this purpose. While pictures taken using these two types of filters both show "images of the spectrum" the spatial resolution is typically much higher for an LVF which is beneficial as we are operating at very low intensity levels and averaging pixels corresponding to the same wavelength helps increasing robustness.

Temporal aspects of appearance due to physical effects have been previously acquired and simulated using stateless, time-dependent BRDFs [Gu et al. 2006]. In our approach we model changes over time as differential equations that express the dependency of the current appearance on the previous illumination state. A related topic in which incident light needs to be integrated over time, too, is fading [Kimmel et al. 2013]. There, the spectral absorption properties of materials change due to chemical reactions. In contrast, in phosphorescence incident light is re-emitted while the material returns to its ground state.

Finally, material appearance is subject to other advanced physical effects such as diffraction [Stam 1999], polarization [Wilkie et al. 2001] or layering [Hanrahan and Krueger 1993; Jakob et al. 2014] which are orthogonal to our approach.

Manipulation A broad overview on existing methods for artistic material manipulation is provided by Schmidt et al. [2014]. While manipulation of time-dependent appearance, to our knowledge, has not been addressed in previous work, the exponential spatial attenuation present in volumes and tackled by Nowrouzezahrai et al. [2011] bears similarities to the exponential temporal attenuation which we face in phosphorescence.

8.8 Conclusion and Future Work

In this chapter, we have presented a holistic pipeline which, based on a simple model, allows for acquisition of actual materials and an efficient implementation suitable for interactive manipulation. We believe that phosphorescence, enabling appearance changes in reaction to illumination, could find uses as a special effect or element of storytelling in movies or interactive applications such as games, e. g., to convey the nocturnal appearance of a scene.

Our model is phenomenological and not strictly motivated by physics. While it is physically plausible, the exact nature of phosphorescence is beyond what might be required in computer graphics. We opt to choose the most simple model that describes the perceivable effects — decay,

saturation and specific re-radiation spectrum — and model phosphorescence as the result of changes in multiple concentrations. This allows a good fit to data which we measured and is in agreement with the material science literature [McKeever and Chen 1997].

Orthogonal extensions include spatially variant acquisition or layered materials [Hanrahan and Krueger 1993] with phosphorescent layers and anisotropic emission behavior. To achieve this, existing frameworks like the one by Jakob et al. [2014] could be easily adopted. Investigating effects of the thickness of the phosphorescent substances is another direction of future work.

We produced images in a physically-based real-time renderer and a path tracer. Phosphorescence is only perceived at scotopic conditions and reproduction of these has its own challenges [Thompson et al. 2002]. Images need to be processed to best match the scotopic visual experience. We have complemented the appearance differences by numerical results in Figure 8.13 for inspection. A more precise comparison would consider the additional effects of scotopic vision. The formulation as a differential equation also requires to render all previous frames before we can actually obtain a certain frame at a specific time t. In the kind of interactive settings our approach is targeted at, this is however happening anyway.

Our manipulation supports time-varying constraints to manipulate discrete materials but is limited to a single exponential that keeps the number of constraints required for a unique solution low. More advanced manipulation should account for the joint effects of global illumination and phosphorescence and use advanced propagation.

9

Conclusion and Outlook

In this final chapter, we will first relate our contributions to the various goals in interactive image synthesis that we introduced in Chapter 1. We then will discuss which aspects are already working well and which need to be improved further or offer potential for extensions. In particular, we will also talk about possible combinations of the different approaches we have presented. Finally, we take a more general outlook into the future of real-time rendering.

9.1 Evaluation of Our Objectives

In this thesis, screen space shading served as a starting point from which we explored different directions of evolving its core ideas and properties. This exploration was driven by the goal to make screen space methods less biased, achieve higher quality and improve their generality. At the same time, we wanted to preserve their desirable traits as much as possible: the resulting methods were supposed to stay suitable for interactive applications, to support arbitrarily dynamic scenes well and to be applicable to a wide range of shading effects. We will now discuss how we managed to achieve these goals.

Interactivity

All our methods have in common that they are suitable for interactive applications. This is essentially due to an exploitation of the limits of human perception as we will detail in the following.

In the deep screen space pipeline (Chapter 4–6), we make extensive use of level-of-detail computations. We approximate distant geometry by surfels which may be large in world space but which are detailed enough when projected to the screen and therefore perceptually sufficiently resolved. And when actually computing shading by means of splatting, we gradually trade precision for speed with increasing distance of the light transport as the effects we compute are diffuse in nature and their intensity falls off so that decreasing precision becomes less noticeable.

For training our CNNs in deep shading (Chapter 7) we employ SSIM, a perceptually-based metric. The neural network thus can focus computations on what will actually be visually important in the final result. We can define a network structure that is sufficiently fast to evaluate and network training automatically yields results which are optimal in a perceptual way given the

computational budget.

Our model of phosphorescence (Chapter 8) also takes human perception into account. We do not try to model what happens at the quantum scale, as the precise processes cannot even be directly observed. Instead we model what is really being noticed about the process by a human observer. The resulting phenomenological model may not predict the behavior of phosphorescent objects in extreme experimental conditions, like at freezing temperatures or under strong irradiation, but captures the aspects of phosphorescence which are actually relevant in everyday situations. It is based on simple rate equations which are efficient to evaluate using Euler integration.

Dynamism

The key to reconciling interactivity with strong dynamism was to avoid expensive pre-computations. If these are only valid with respect to a particular camera view, a static scene geometry or one lighting condition, either dynamism is limited or recomputation is slow. For example, we leverage hardware tessellation to compute the approximate deep screen space scene representation and instead of building a costly hierarchy over initially unordered surfels for gathering, we move this hierarchy to the splatting target in a scattering-type computation where it is much faster to generate. Deep shading, which is directly computing on screen space information, naturally also carries over its flexibility. For rendering phosphorescence efficiently, we rely on texturing and support dynamic objects by quickly updating an atlas that tracks the location and orientation of phosphorescent surface points in the scene.

Quality

One of our main goals was to outperform screen space approaches in terms of rendering quality. The more complete scene information in deep screen space allowed us to recover, for example, otherwise entirely missing indirect shadows or to enable light scattering through objects from their back-facing side (Figure 4.9). When moving in a scene, portions of the shading do not appear and disappear anymore with the currently available subset of the geometry — as in screen space — but are actually temporally consistent. We further reduced remaining bias by adding support for indirect visibility (Section 6.3) in Chapter 6.

More complete scene information was also what allowed deep shading to surpass common screen space methods in a same-time comparison (Figure 7.6). Even though both are provided the same incomplete input at run-time, deep shading gains basic knowledge about the structure of natural scenes during the preceding training stage which uses unbiased ground truth data and can later draw advantages from this.

Regarding phosphorescence, the only previous model due to Glassner [1995] is built around a closed-form formula which makes it necessary to sample irradiance over previous points in time to determine the current phosphorescent emission for each new frame. No noisefree results can be obtained in real time. Our method merely requires a fast update to a texture and produces smooth results (Figure 8.16).

Generality

With deep shading, we presented a method that covers many aspects of light transport in a unified way. Not only can we train the same neural network structure on all common shading effects individually, but it is also possible to learn combinations of them (Section 7.4) which are then reproduced in a joint set of computations.

Completeness

Finally, all our methods are highly universal. We applied deep screen space and deep shading to both simple surface shading but also volume scattering effects. With our phosphorescence framework we extended the capabilities of real time rendering by a new aspect of temporally varying appearance and also demonstrated its use for phosphorescent participating media, not just surfaces.

9.2 Current State of our Contributions and Remaining Future Work

Interactive Global Illumination

Our contributions to the area of interactive computation of global illumination in a scene revolved around the idea of a deep screen space, a novel level-of-detail disk-based scene representation (Chapter 4). We showed that, using fast hardware tessellation, it is possible to construct the deep screen space representation within milliseconds. A corollary of this is that it is feasible to recompute the representation for each frame to be rendered, which means arbitrary geometric changes or camera movements are automatically supported. This is different from many competing methods which require a costly pre-computation or update to a supporting data structure to happen in such cases. The deep screen space was complemented by shell splatting (Section 4.2.2), using which we were able to approximate light transport with both solid and volumetric receivers (Chapter 4–5). The initial problem of lacking indirect visibility was fixed using the bounced z-buffer (Chapter 6).

Rendering using deep screen space currently achieves interactive frame rates for moderate resolutions and is not fast enough for applications with an extremely tight computational budget such as games, yet. It seems however a promising choice for preview computation, e. g., in a 3D modeling software. Given the continuing improvement of hardware capabilities, the current state of the art in high-performance rendering — screen space methods in particular — will saturate once its remaining limitations in terms of quality will be merely a result of inherent bias, not of restricted sample counts. At that point, a shift to higher quality but more demanding approaches, like deep screen space, will become inevitable.

We used deep screen space to render ambient or directional occlusion, indirect light and multiple scattering in objects. The list of very common screen space effects also includes shallow depth of field and motion blur. These suffer from limitations due to under-sampled geometry in the same way that the ones we already covered do and would also benefit from deep screen space. They are however different in that the surfel cloud itself would have to be rendered. A particularly fine grained tessellation would become necessary. Also the splatting stage would need adjustment as depth-of-field requires splatting of disk primitives while motion blur could likely be approximated splatting curves or just lines. As these effects have no distinct “fall-off” of intensity, a sub-sampling as done by shell splatting is not possible, though.

Additional future work concerns the types of light paths we can simulate using the approach. With deep screen space, we can so far handle light transport between surfaces and from surfaces to volumes. Extending this to volumetric senders, e. g., for self-emitting volumes such as they are used to render explosions or as they occur for hypothetical phosphorescent mixtures, would require an appropriate level-of-detail proxy representation of the volume. Also, we have so far only considered diffuse senders of indirect light. An extension to glossy receivers could be

enabled by differently shaped splatting primitives like ellipses instead of disks.

Our bounced z-buffer extends splatting-based methods by support for indirect visibility. So far, we have only considered opaque blockers of indirect light but semi-transparent ones seem also possible. Similar to order-independent transparency [McGuire and Bavoil 2013], multiple surfels could be stored for each pixels gathering direction. To some extent we already do this by storing one candidate per surrounding shell. The final compositing then could use information about all stored surfels and blend them in a meaningful way, taking the individual translucency into account.

The bounced z-buffer approach can also benefit from a general optimization: Instead of considering only directions from which indirect light is arriving, it might make more sense to consider complete cones, with a width depending on the glossiness of the receiving surface. This would allow more precise but sparser sampling if the latter is glossy and less precise but denser sampling if it is diffuse.

Screen Space Shading by Example

In Chapter 7, we investigated the use of machine learning techniques for screen space image synthesis. Using a general deep neural network architecture, we are able to reproduce a plethora of different effects just by learning from exemplary data. The ability to learn from unbiased ground truth data allowed us to come up with deep shaders that outperform conventional screen space approaches in terms of quality, given the same computational budget. Furthermore, said computational budget which is defined by the network structure can be allocated in an optimal way by means of network training to reproduce multiple effects at once, something that is not possible to accomplish in an easy way using manual coding.

Similar to deep screen space, deep shading can go beyond what is traditionally possible in screen space, but by gaining knowledge about the general structure of natural scenes instead of using a more complete input. The current main drawbacks of the idea are the lack of temporal coherence and the lower bound on the performance of the method which is way higher than for sampling-based approaches which allow a free choice of the sample count. The first one could be addressed by training the network on short sequences and using a spatio-temporal loss, possibly accompanied by adjustments to the network structure indicated by the new loss. Also extending the network input by the result produced for the previous frame or using a recurrent network could allow for some kind of reprojection [Scherzer et al. 2012]. The second might be resolved by the advent of specialized deep learning hardware designed for fast execution of neural networks [Lacey et al. 2016].

Beyond improvements to the current state of deep shading, the question arises how far the general idea of applying deep learning, or machine learning in general, to rendering can take us. While previous applications in this direction already produced high quality results, e. g., for re-lighting and pre-computed radiance transfer [Ren et al. 2013; Ren et al. 2015], they were limited to and had to be trained for a single and de facto static scene. Deep shading is the first approach in this direction which actually achieves a generalization for arbitrary dynamic geometry while really replacing the rendering process itself by a learned operation, but its current downside is the limited quality due to the surrounding screen space setting.

Replacing classic rendering by what is essentially curve fitting might seem odd as deep learning is usually used for problems where a mapping is too abstract or not well understood so that it cannot be devised manually, e. g., for determining the object category recognized in an image by a

human. Contrarily, rendering techniques are based on well-known analytic computations derived from optical principles. But these computations are actually very complex. Naive solutions fail in many cases, for example when trying to use simple distributed ray tracing from the camera for rendering caustics. A lot of thought has to be put into coming up with specialized algorithms and clever approximations, taking into account external preconditions like human perception. Learning offers the potential to exploit the statistics of natural three dimensional scenes in new ways. The observation that objects usually not only have a front but also a backside as made by deep shading (Figure 7.6) is only a first example for this.

To pursue this direction further we need to find ways to make free-form data such as meshes accessible to something operating on a very rigid structure such as a CNN. A next step there might be to augment the input to deep shading by a coarse voxelized scene representation such as used by light propagation volumes [Kaplanyan and Dachsbacher 2010] which can still be obtained reasonably efficiently but has more complete information about the full scene.

Rendering of Advanced Physical Effects

We were able to expand the set of visual effects viable in real-time rendering by introducing our phosphorescence framework in Chapter 8. Our holistic framework deals with all relevant aspects of the phenomenon: modeling, acquisition, reproduction and editing. Using our texture-based rendering method the approach is suitable for real-time applications as is, which is different from previous methods which are too time- and memory-consuming.

The main weakness currently is our manipulation framework which is based on a very simple parallel exhaustive search. This approach is fast when using only RGB rendering and a single-exponential model as in this case the valid parameter space is low dimensional enough to be just sampled thoroughly. Spectral and multi-exponential fits are however not well supported. Adding support for these requires a more efficient optimization for the best set of parameters. A simple improvement would be to first determine a normalized re-radiation spectrum before finding the remaining parameters. Just averaging and then normalizing the user's spectral constraints for this might already work well.

The acquisition system we have proposed still requires a lot of human interaction and careful handling. This is why for acquisition of a larger number of different phosphorescent materials, a more robustly constructed and possibly automatized measurement device would be useful in practice, e. g., using 3D printed moving parts driven by an actuator instead of relying on gravity to achieve smaller temporal delays during the measurements and have a more precise starting point for fitting.

Other methods working in texture space, for example for sub-surface scattering in skin [d'Eon et al. 2007], have benefited performance-wise from a transfer to screen space [Jimenez et al. 2009], so the question whether this might also be the case for rendering phosphorescence arises. Clearly, new excitation of a phosphorescent material can be computed in screen space as this is equivalent to determining the incident light for the computation of simple surface shading. However, the state of points on objects would have to be tracked across new frames. While re-projection [Scherzer et al. 2012] is a suitable tool for this task, the occlusion and culling inherent to screen space would lead to an immediate loss of this information for points going out of view. In summary, this idea seems appropriate for objects moving slowly in screen space, at best.

9.3 Possible Algorithmic Combinations

The different methods presented in this thesis can not only be advanced individually but they also naturally lend themselves to various combinations.

We already used the deep screen space pipeline to transport light bounced off surfaces but we could also illuminate surfaces or volumes in a scene by the re-radiation from phosphorescent surfaces. This would be equivalent to indirect light computation with the only difference that surfels would not have to be shaded by the primary light sources anymore but based on the current re-emission. Only phosphorescent objects would have to be considered as senders and be surfelized which should allow for fast rendering.

A combination of deep screen space and deep shading seems difficult to achieve at first as the deep screen space is an unordered cloud of disks while common CNNs operate on regular input defined on a grid. But projecting the cloud on a permutohedral lattice [Adams et al. 2010] could help to make deep screen space accessible to filtering by a neural network. If successful, this approach would yield higher quality, less biased images as the method can draw from a larger amount of information about the scene.

To model phosphorescence, we used an intuitive model based on rate equations which achieves a sufficiently precise but still imperfect fit to real world materials. Replacing the phenomenological model by one based on quantum mechanical principles would however be way too complex. For such a setting, a recurrent neural network (RNN) might be a solution. Such a network could learn to return the right amount of re-radiation depending on the illumination that a material has been exposed to at previous simulation steps.

9.4 Outlook

Given the long history of rendering within computer graphics it might seem as if all problems in (real-time) rendering had been solved already. This is not quite the case. The complexity of possible light paths increases exponentially with the path length allowed which still manifests in a lack of real-time methods for rendering global illumination with multiple bounces and a lack of efficient methods for rendering volume scattering with arbitrary scattering orders. Even direct lighting without bounces becomes difficult to perform efficiently when visibility and shading are supposed to be treated jointly for complex light sources with spatially varying intensity.

While it seems likely that the computational power of hardware will continue to increase in the future, the same holds for the demand for better rendering quality. Virtual and augmented reality applications require particularly high resolutions, frame rates and visual quality to become more convincing. In areas such as mobile or automotive computing reaching these goals is even more complicated due to additional constraints regarding size and energy consumption of the computing devices. Considering all these challenges, we believe that specialized real-time methods will continue to be the most relevant for the aforementioned kinds of applications for a long time, with a bit more demanding but higher quality methods of the spectrum, such as deep screen space, gradually replacing faster but less precise ones.

An example for a method whose use in practice has gradually increased over a long period of more than 70 years, due to different technological advances, is the idea of artificial neural networks [McCulloch and Pitts 1943]. Nowadays, deep learning is state-of-the-art in most areas

of computer vision but applications to image synthesis have so far only scratched the surface. Deep shading is a first proof that learning can not only be applied to small sub-problems in rendering but that it might replace big chunks of the synthesis pipeline. Learning could also advance high quality (offline) rendering as it, in an abstract sense, promises the ability to focus computations on the specifics of natural scenes, which are what we want to render. However, reasoning in path space, for example, still needs to be explored.

Appendix A

Phosphorescence Appendix

A.1 Double-exponential Fit

The parameters for a double-exponential fit that we acquired can be found in Table A.1.

A.2 Fitted Decay Curves

Figure A.1 contains single-exponential and double-exponential fits for the decay behavior we measured (points).

A.3 Saturation Curves

Figure A.2 contains the data we acquired in our saturation experiments (points) as well as the result of simulating the single- and double-exponential models with the parameters we found. Each line and set of points of one color corresponds to the process of excitation by light of only that wavelength band.

Material	$k_{r,1}$	$k_{r,2}$	$k_{e,1}$			$k_{e,2}$			Λ_1								Λ_2							
			λ_1	λ_2	λ_3	λ_1	λ_2	λ_3	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8
Blue	0.008	0.116	0.0030	0.0075	0.0008	0.0420	0.1100	0.0090	0.43	5.59	4.67	0.71	0.0	0.01	0.0	0.0	0.09	1.22	1.02	0.15	0.0	0.0	0.0	0.0
Green	0.027	0.398	0.0110	0.0044	0.0007	0.1500	0.0600	0.0090	0.0	0.0	0.09	0.79	0.07	0.0	0.0	0.0	0.0	0.0	0.04	0.32	0.03	0.0	0.0	0.0
Orange	0.006	0.091	0.0036	0.0016	0.0000	0.0480	0.0190	0.0000	0.05	2.07	0.53	0.0	0.51	8.65	5.48	2.80	0.01	0.33	0.08	0.0	0.08	1.36	0.86	0.44
Pink	0.008	0.111	0.0014	0.0030	0.0005	0.0200	0.0400	0.0073	0.73	5.00	0.61	0.0	0.0	1.50	2.17	0.11	0.15	1.01	0.12	0.0	0.0	0.3	0.44	0.02
Purple	0.108	0.007	0.0360	0.0440	0.0030	0.0027	0.0030	0.0003	0.12	0.80	0.09	0.0	0.0	0.05	0.02	0.0	0.57	3.84	0.45	0.0	0.0	0.24	0.11	0.0
Red	0.007	0.093	0.0026	0.0009	0.0000	0.0380	0.0130	0.0000	0.14	3.42	0.50	0.0	0.03	6.21	7.03	0.72	0.02	0.55	0.08	0.0	0.0	1.00	1.13	0.12
White	0.147	0.009	0.0390	0.1000	0.0130	0.0025	0.0060	0.0009	0.08	0.80	0.78	0.19	0.0	0.0	0.0	0.03	0.36	3.55	3.47	0.84	0.0	0.01	0.0	0.14
Yellow	0.025	0.414	0.0160	0.0040	0.0010	0.2500	0.0600	0.0110	0.0	0.0	0.10	1.01	0.18	0.15	0.02	0.0	0.0	0.0	0.04	0.44	0.08	0.06	0.01	0.0
Bottle	0.020	0.324	-	-	-	-	-	-	0.0	0.06	0.55	0.79	0.34	0.08	0.01	0.0	0.0	0.02	0.21	0.30	0.13	0.03	0.0	0.0
Gloves	0.178	0.011	-	-	-	-	-	-	0.03	0.31	1.87	1.86	0.71	0.24	0.04	0.0	0.14	1.30	7.78	7.76	2.95	0.98	0.18	0.01
Stickers	0.132	0.009	-	-	-	-	-	-	0.0	1.70	11.5	11.5	4.19	1.43	0.31	0.04	0.0	7.62	51.6	51.5	18.8	6.43	1.41	0.18

Table A.1: Re-radiation and excitation rates in units s^{-1} for the double-exponential fit.

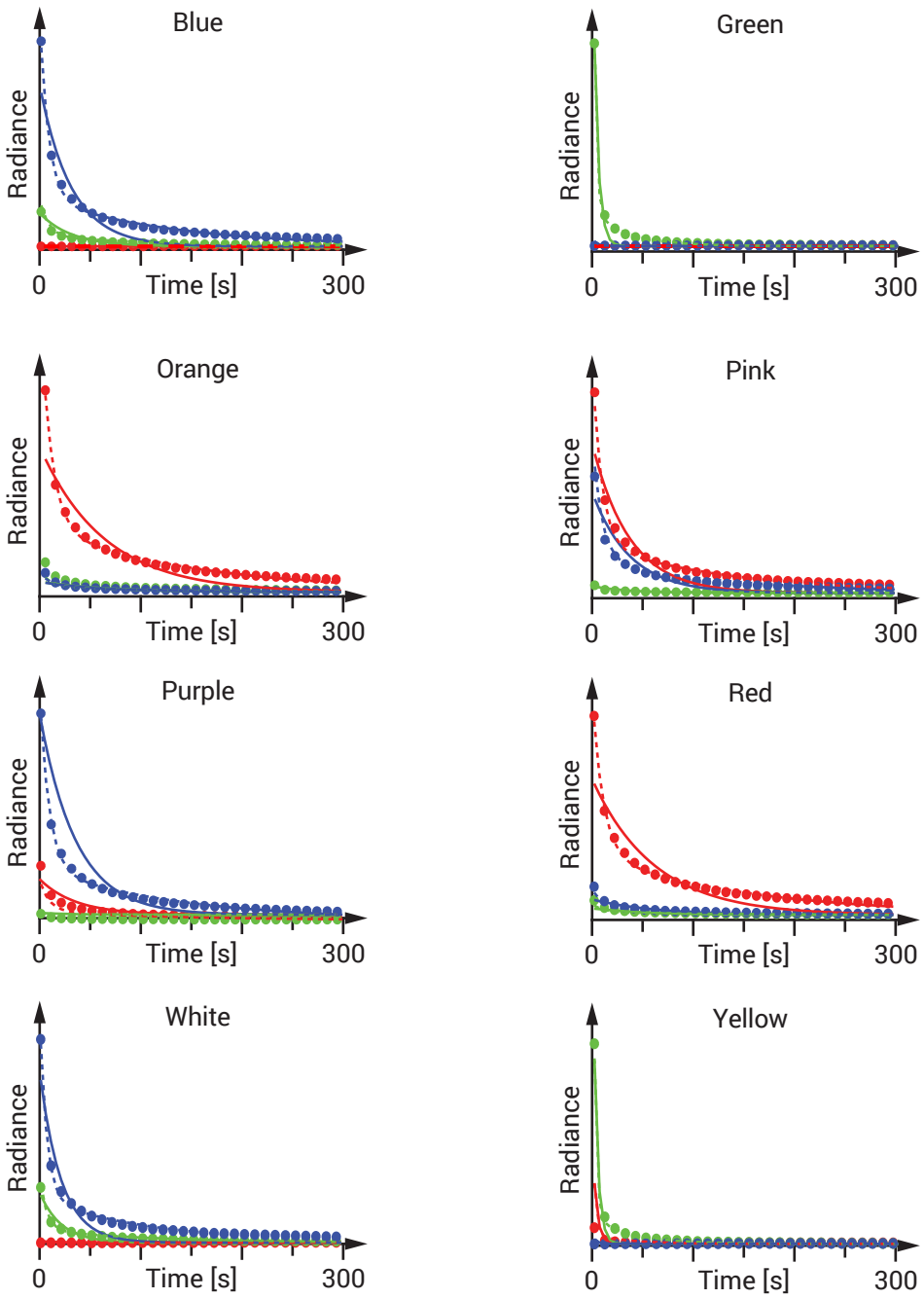


Figure A.1: Measured decay of the phosphorescence intensity and fitted single- (solid lines) and double-exponential (dashed) decay functions.

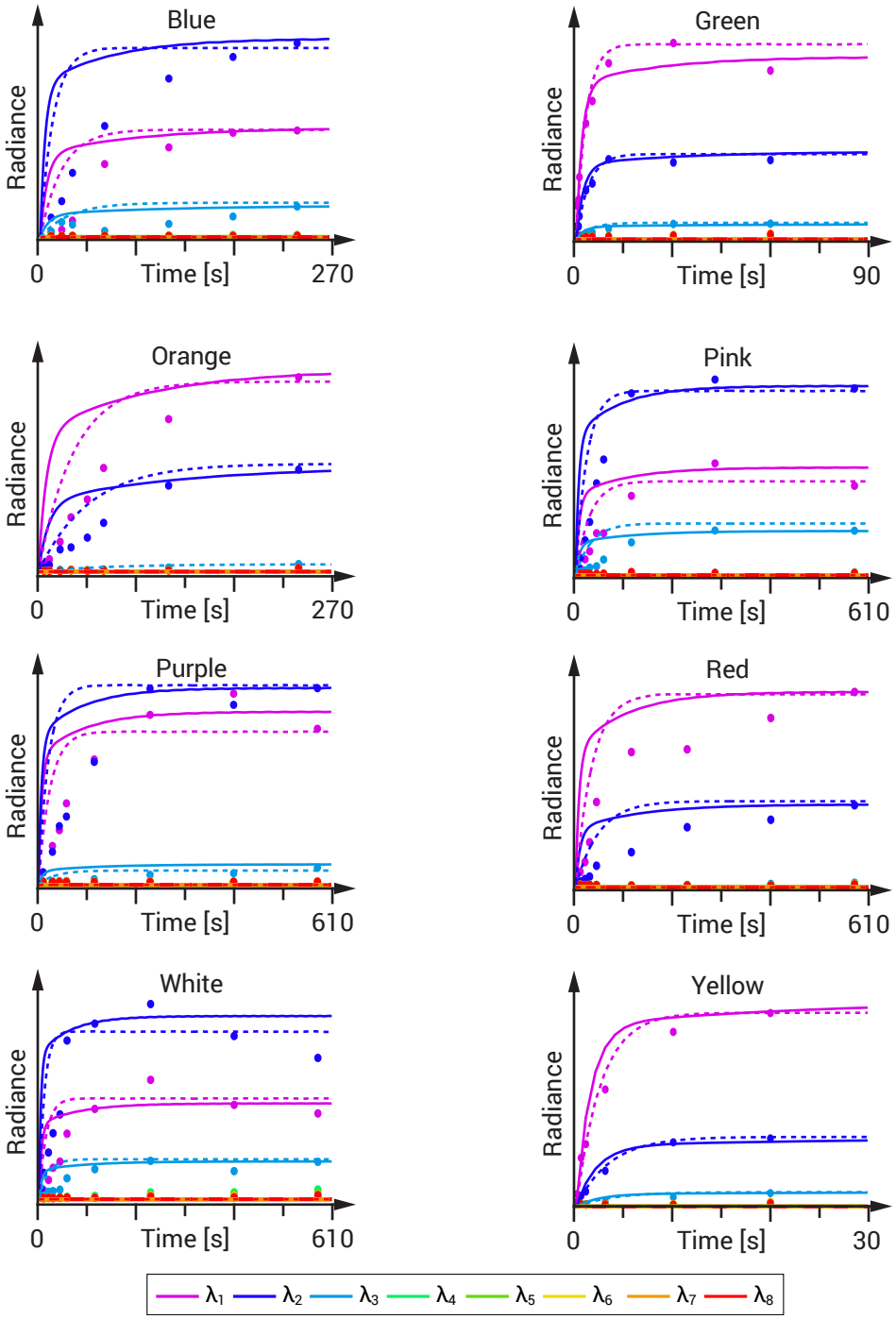


Figure A.2: Measured saturation of the phosphorescence intensity and curves simulated using fitted parameters for the single- (solid) and double-exponential (dashed) models.

Bibliography (Own Work)

Nalbach, O., Arabadzhiyska, E., Mehta, D., Seidel, H. and Ritschel, T. (2017a): Deep Shading: Convolutional Neural Networks for Screen-Space Shading. In *Eurographics Symposium on Rendering (EGSR)* Volume 36, 2, 3

Nalbach, O., Ritschel, T. and Seidel, H.-P. (2014a): Deep Screen Space. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)* 2, 3, 58, 59

Nalbach, O., Ritschel, T. and Seidel, H.-P. (2014b): Shell Splatting for Indirect Lighting of Volumes. In *International Symposium on Vision, Modeling and Visualization (VMV)* 2, 3

Nalbach, O., Ritschel, T. and Seidel, H.-P. (2015): The Bounced Z-Buffer for Indirect Visibility. In *International Symposium on Vision, Modeling and Visualization (VMV)* 2, 3

Nalbach, O., Seidel, H.-P. and Ritschel, T. (2017b): Practical Capture and Reproduction of Phosphorescent Appearance. In *Eurographics* Volume 36, 2, 3

Bibliography

Aalund, F. P. (2013): A Comparative Study of Screen-Space Ambient Occlusion Methods. Bachelor Thesis 2, 28, 29

Adams, A., Baek, J. and Davis, M. A. (2010): Fast High-Dimensional Filtering Using the Permutohedral Lattice. In *Eurographics* Volume 29,, 753–762 122

Aila, T. and Laine, S. (2009): Understanding the efficiency of ray traversal on GPUs. In *High Performance Graphics (HPG)*, 145–149 21, 32

Alvarez-Cortes, S., Kunkel, T. and Masia, B. (2016): Practical Low-Cost Recovery of Spectral Power Distributions. *Computer Graphics Forum*, 35 (1), 166–178 115

Appel, A. (1968): Some Techniques for Shading Machine Renderings of Solids. In *AFIPS Joint Computer Conference*, 37–45 21

Arvo, J., Torrance, K. and Smits, B. (1994): A framework for the analysis of error in global illumination algorithms. In *ACM SIGGRAPH*, 75–84 99

Barák, T., Bittner, J. and Havran, V. (2013): Temporally Coherent Adaptive Sampling for Imperfect Shadow Maps. *Computer Graphics Forum*, 32 (4), 87–96 26, 32

Bauszat, P., Eisemann, M., Eisemann, E. and Magnor, M. (2015): General and robust error estimation and reconstruction for monte carlo rendering. *Eurographics* 34 (2) 21

Bavoil, L., Sainz, M. and Dimitrov, R. (2008): Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks* 28, 38, 39, 44, 62, 77

Bendig, M., Hanika, J., Dammertz, H., Goldschmidt, J. C., Peters, M. and Weber, M. (2008): Simulation of Fluorescent Concentrators. In *IEEE Symposium on Interactive Ray Tracing*, 93–98 115

Bosch, A., Zisserman, A. and Munoz, X. (2007): Image classification using random forests and ferns. In *IEEE International Conference on Computer Vision (ICCV)*, 1–8 91

Bunnell, M. (2005): Dynamic Ambient Occlusion and Indirect Lighting. In Pharr, M., editor: *GPU Gems 2*, 223–233 24, 43, 44

Catmull, E. E. (1974): A Subdivision Algorithm for Computer Display of Curved Surfaces. Ph. D thesis, University of Utah 24

Chandrasekhar, S. (1950): Radiative Transfer. 5, 15

Chen, B., Hao, H.-C., Zhu, J. and Lu, M. (2011): A Phenomenological Model for Decay Process of Long-Persistent Phosphorescence. *Chinese Physics Letters* 28 (5) 96, 105

Christensen, P. H. (2008): Point-based approximate color bleeding. Pixar – Technical report 24, 44, 65

Christensen, P. H. (2015): An Approximate Reflectance Profile for Efficient Subsurface Scattering. In *ACM SIGGRAPH 2015 Talks* 29, 72, 79

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., and Vedaldi, A. (2014): Describing Textures in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3606–3613 73

Cohen, M. F. and Greenberg, D. P. (1985): The hemi-cube: A radiosity solution for complex environments. In *ACM SIGGRAPH* Volume 19, ACM, 31–40 65

Cook, R. L., Carpenter, L. and Catmull, E. (1987): The Reyes image rendering architecture. In *ACM SIGGRAPH* Volume 21,, 95–102 22, 45

Cook, R. L., Porter, T. and Carpenter, L. (1984): Distributed Ray Tracing. In *ACM SIGGRAPH* Volume 18,, 137–145 21

Crassin, C., Neyret, F., Sainz, M., Green, S. and Eisemann, E. (2011): Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum*, 30 (7), 1921–1930 30

Criminisi, A. and Shotton, J. (2013): Decision forests for computer vision and medical image analysis. 90

Dachsbacher, C. (2011): Analyzing visibility configurations. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 17 (4), 475–486 92

Dachsbacher, C. and Stamminger, M. (2005): Reflective shadow maps. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 203–231 25, 44

Dachsbacher, C. and Stamminger, M. (2006): Splatting Indirect Illumination. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)* , ISBN 1–59593–295–X, 93–100 27, 58, 59

d'Eon, E., Luebke, D. and Enderton, E. (2007): Efficient Rendering of Human Skin. In *Eurographics Symposium on Rendering (EGSR)*, 147–157 121

Donaldson, R. (1954): Spectrophotometry of fluorescent pigments. *British Journal of Applied Physics*, 5 (6), 210–214 96

Dosovitskiy, A., Tobias Springenberg, J. and Brox, T. (2015): Learning to Generate Chairs With Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1538–1546 93

Eigen, D., Puhrsch, C. and Fergus, R. (2014): Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems (NIPS)*, 2366–2374 93

Elek, O., Ritschel, T., Dachsbacher, C. and Seidel, H.-P. (2014): Interactive Light Scattering with Principal-Ordinate Propagation. In *Graphics Interface*, 87–94 30, 55

Elek, O., Ritschel, T. and Seidel, H.-P. (2013): Real-Time Screen-Space Scattering in Homogeneous Environments. *IEEE Computer Graphics and Applications*, (3), 53–65 28, 91

Engelhardt, T., Novák, J., Schmidt, T.-W. and Dachsbacher, C. (2012): Approximate Bias Compensation for Rendering Scenes with Heterogeneous Participating Media. In *Pacific Graphics*, 2145–2154 24, 55

Everitt, C. (2001): Interactive Order-Independent Transparency. Nvidia – Technical report 29

Farbman, Z., Fattal, R. and Lischinski, D. (2011): Convolution Pyramids. In *ACM SIGGRAPH Asia* Volume 30,, 175:1–175:8 92

Fattal, R. (2009): Participating media illumination using light propagation maps. *ACM Transactions on Graphics*, 28 (1), 7 30, 55

Gatys, L. A., Ecker, A. S. and Bethge, M. (2015): A neural algorithm of artistic style. *arXiv 1508.06576* 82, 83, 92

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014): Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 580–587 93

Glassner, A. S. (1995): A model for fluorescence and phosphorescence. In *Photorealistic Rendering Techniques*, 60–70 110, 111, 114, 118

Goesele, M., Granier, X., Heidrich, W. and Seidel, H.-P. (2003): Accurate Light Source Acquisition and Rendering. In *ACM SIGGRAPH* Volume 22,, 621–630 7

Goral, C. M., Torrance, K. E., Greenberg, D. P. and Battaile, B. (1984): Modeling the Interaction of Light Between Diffuse Surfaces. In *ACM SIGGRAPH* Volume 18,, 213–222 22

Gu, J., Tu, C.-I., Ramamoorthi, R., Belhumeur, P., Matusik, W. and Nayar, S. (2006): Time-varying surface appearance: Acquisition, modeling and rendering. In *ACM SIGGRAPH* Volume 25,, 762–771 115

Gutierrez, D., Munoz, A., Anson, O. and Seron, F. J. (2005): Non-linear Volume Photon Mapping. In *Eurographics Symposium on Rendering (EGSR)*, 291–300 114

Hanrahan, P. and Krueger, W. (1993): Reflection from layered surfaces due to subsurface scattering. In *ACM SIGGRAPH*, 165–174 99, 115, 116

Hariharan, B., Arbeláez, P., Girshick, R. and Malik, J. (2015): Hypercolumns for Object Segmentation and Fine-grained Localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 447–456 93

Hastad, J. (1986): Almost optimal lower bounds for small depth circuits. In *ACM Symposium on Theory of Computing*, 6–20 91

Helmholtz, H. L. F. von (1866): Handbuch der physiologischen Optik., Allgemeine Encyklopädie der Physik 8

Henyey, L. G. and Greenstein, J. L. (1941): Diffuse radiation in the Galaxy. *Astrophysical Journal*, 93, 70–83 10, 53

Hertzmann, A. (2003): Machine learning for computer graphics: A manifesto and tutorial. In *Pacific Graphics*, 22 92

Holopainen, S., Manoocheri, F. and Ikonen, E. (2008): Goniofluorometer for characterization of fluorescent materials. *Applied Optics*, 47 (6), 835–842 115

Hu, W., Dong, Z., Ihrke, I., Grosch, T., Yuan, G. and Seidel, H.-P. (2010): Interactive Volume Caustics in Single-scattering Media. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 109–117 27, 55

Hullin, M. B., Hanika, J., Ajdin, B., Seidel, H.-P., Kautz, J. and Lensch, H. P. A. (2010): Acquisition and Analysis of Bispectral Bidirectional Reflectance and Reradiation Distribution Functions. In *ACM SIGGRAPH* Volume 29,, 97:1–97:7 115

Jakob, W., d'Eon, E., Jakob, O. and Marschner, S. (2014): A Comprehensive Framework for Rendering Layered Materials. In *ACM SIGGRAPH* Volume 33,, ISSN 0730–0301, 118:1–118:14 99, 115, 116

Jensen, H. W. and Buhler, J. (2002): A rapid hierarchical rendering technique for translucent materials. In *ACM SIGGRAPH* Volume 21,, 576–581 23, 41, 44

Jensen, H. W. (1996): Global Illumination using Photon Maps. In *Eurographics Workshop on Rendering (EGWR)*, 21–30 22

Jensen, H. W. and Christensen, P. H. (1998): Efficient simulation of light transport in scenes with participating media using photon maps. In *ACM SIGGRAPH*, 311–320 22

Jensen, H. W., Marschner, S. R., Levoy, M. and Hanrahan, P. (2001): A practical model for subsurface light transport. In *ACM SIGGRAPH*, 511–518 16, 21, 23, 29

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. (2014): Caffe: Convolutional architecture for fast feature embedding. In *ACM International conference on Multimedia*, 675–8 74

Jimenez, J., Sundstedt, V. and Gutierrez, D. (2009): Screen-space perceptual rendering of human skin. *ACM Transactions on Applied Perception*, 6 (4), 23:1–23:15 29, 41, 44, 79, 91, 121

Johnson, M. K., Dale, K., Avidan, S., Pfister, H., Freeman, W. T. and Matusik, W. (2011): CG2Real: Improving the realism of computer generated images using a large collection of photographs. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 17 (9), 1273–1285 92

Jonscher, A. K. and Polignac, A. de (1984): The time dependence of luminescence in solids. *Journal of Physics C: Solid State Physics*, 17 (35), 6493–6519 97, 105

Kajiya, J. T. (1986): The rendering equation. In *ACM SIGGRAPH* Volume 20,, 143–150 5, 10, 12

Kalantari, N. K., Bako, S. and Sen, P. (2015): A Machine Learning Approach for Filtering Monte Carlo Noise. In *ACM SIGGRAPH* Volume 34,, 122:1–122:12 90, 92

Kaplanyan, A. and Dachsbacher, C. (2010): Cascaded light propagation volumes for real-time indirect illumination. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 99–107 30, 55, 121

Kawakami, R., Hongxun, Z., Tan, R. T. and Ikeuchi, K. (2013): Camera Spectral Sensitivity and White Balance Estimation from Sky Images. *International Journal of Computer Vision*, 105 (3), 187–204 107

Keller, A. (1997): Instant radiosity. In *ACM SIGGRAPH*, 49–56 23, 44, 55, 65

Kettunen, M., Manzi, M., Aittala, M., Lehtinen, J., Durand, F. and Zwicker, M. (2015): Gradient-Domain Path Tracing. In *ACM SIGGRAPH* Volume 34,, 123:1–123:13 21

Kim, M. H., Harvey, T. A., Kittle, D. S., Rushmeier, H., Dorsey, J., Prum, R. O. and Brady, D. J. (2012): 3D Imaging Spectroscopy for Measuring Hyperspectral Patterns on Solid Objects. In *ACM SIGGRAPH* Volume 31,, 38:1–38:11 115

Kim, T.-Y. and Neumann, U. (2001): Opacity shadow maps. In *Eurographics Workshop on Rendering (EGWR)*, 177–182 26, 56

Kimmel, B. W., Baranoski, G. V. G., Chen, T. F., Yim, D. and Miranda, E. (2013): Spectral Appearance Changes Induced by Light Exposure. *ACM Transactions on Graphics*, 32 (1), 10:1–10:13 115

Kirk, A. G. and O'Brien, J. F. (2011): Perceptually Based Tone Mapping for Low-light Conditions. In *ACM SIGGRAPH* Volume 30,, 42:1–42:10 101, 102

Klehm, O., Seidel, H.-P. and Eisemann, E. (2014): Prefiltered Single Scattering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 71–8 26, 55

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012): Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 1097–1105 93

Kulkarni, T. D., Whitney, W., Kohli, P. and Tenenbaum, J. B. (2015): Deep Convolutional Inverse Graphics Network. In *Advances in Neural Information Processing Systems (NIPS)*, 2539–2547 93

Lacey, G., Taylor, G. W. and Areibi, S. (2016): Deep Learning on FPGAs: Past, Present, and Future. *arXiv 1602.04283* 120

Lafortune, E. P. and Willems, Y. D. (1993): Bi-Directional Path Tracing. In *International Conference on Computational Graphics and Visualization Techniques (COMPUGRAPHICS)*, 145–153 21

Lafortune, E. P. and Willems, Y. D. (1996): Rendering Participating Media with Bidirectional Path Tracing. In *Eurographics Workshop on Rendering (EGWR)*, 91–100 21

Lee, K., Siegel, J., Webb, S., Leveque-Fort, S., Cole, M., Jones, R., Dowling, K., Lever, M. and French, P. (2001): Application of the stretched exponential function to fluorescence lifetime imaging. *Biophysical Journal*, 81 (3), 1265–1274 97, 105

Lee, S., Eisemann, E. and Seidel, H.-P. (2009a): Depth-of-field Rendering with Multiview Synthesis. In *ACM SIGGRAPH Asia* Volume 28,, 134:1–134:6 28

Lee, S., Kim, G. J. and Choi, S. (2009b): Real-Time Depth-of-Field Rendering Using Anisotropically Filtered Mipmap Interpolation. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 15 (3), 453–464 28

Lehtinen, J., Aila, T., Laine, S. and Durand, F. (2012): Reconstructing the Indirect Light Field for Global Illumination. In *ACM SIGGRAPH* Volume 31,, 51:1–51:10 23, 65

Leverenz, H. W. (1968): An introduction to luminescence of solids. 96, 97

Lokovic, T. and Veach, E. (2000): Deep Shadow Maps. In *ACM SIGGRAPH*, 385–392 26, 55

Long, J., Shelhamer, E. and Darrell, T. (2015): Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440 70, 93

Lottes, T. (2011): FXAA. Nvidia – Technical report 79, 92

Maas, A. L., Hannun, A. Y. and Ng, A. Y. (2013): Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning (ICML)* 30 73

Mara, M., McGuire, M., Nowrouzezahrai, D. and Luebke, D. (2016): Deep G-Buffers for Stable Global Illumination Approximation. In *High Performance Graphics (HPG)* 29

Mattausch, O., Bittner, J., Villanueva, A. J., Gobbetti, E., Wimmer, M. and Pajarola, R. (2015): CHC+RT: Coherent Hierarchical Culling for Ray Tracing. *Eurographics* (2) 21, 65

Max, N. (1995): Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 1 (2), 99–108 16, 49

McCulloch, W. S. and Pitts, W. (1943): A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5 (4), 115–133 122

McGuire, M. (2010): Ambient Occlusion Volumes. In *High Performance Graphics (HPG)* 27, 44, 58, 59, 61

McGuire, M. and Bavoil, L. (2013): Weighted Blended Order-Independent Transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2 (2), 122–141 65, 120

McGuire, M., Hennessy, P., Bukowski, M. and Osman, B. (2012): A reconstruction filter for plausible motion blur. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 135–142 79, 92

McGuire, M. and Mara, M. (2014): Efficient GPU Screen-Space Ray Tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3 (4), 73–85 28

McKeever, S. and Chen, R. (1997): Luminescence models. *Radiation Measurements*, 27 (5), 625–61 97, 105, 116

Meier, B. J. (1996): Painterly rendering for animation. In *Luminescence models*, 477–484 29

Mertens, T., Kautz, J., Bekaert, P. and Van Reeth, F. (2004): A self-shadow algorithm for dynamic hair using density clustering. In *ACM SIGGRAPH 2004 Sketches*, 44 26, 56

Mertens, T., Kautz, J., Bekaert, P., Van Reeth, F. and Seidel, H.-P. (2005): Efficient Rendering of Local Subsurface Scattering. *Computer Graphics Forum*, 24 (1), 41–49 29

Miller, G. (1994): Efficient Algorithms for Local and Global Accessibility Shading. In *ACM SIGGRAPH* , ISBN 0–89791–667–0, 319–326 13

Mittring, M. (2007): Finding next gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses*, 97–121 28, 44, 91

Narihira, T., Maire, M. and Yu, S. X. (2015): Direct Intrinsics: Learning Albedo-Shading Decomposition by Convolutional Regression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2992–2993 93

Navarro, F., Serón, F. J. and Gutierrez, D. (2011): Motion blur rendering: State of the art. *Computer Graphics Forum*, 30 (1), 3–26 28

Nichols, G. and Wyman, C. (2009): Multiresolution splatting for indirect illumination. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 83–90 27, 44, 58, 59

Nicodemus, F., Richmond, J., Hsia, J., Ginsberg, I. and Limperis, T. (1977): Geometrical Considerations and Nomenclature for Reflectance., NBS monograph 8

Novák, J., Engelhardt, T. and Dachsbacher, C. (2011): Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 119–124 55

Novák, J., Nowrouzezahrai, D., Dachsbacher, C. and Jarosz, W. (2012): Virtual ray lights for rendering scenes with participating media. In *ACM SIGGRAPH* Volume 31,, 60:1–60:11 24

Nowrouzezahrai, D., Johnson, J., Selle, A., Lacewell, D., Kaschalk, M. and Jarosz, W. (2011): A Programmable System for Artistic Volumetric Lighting. In *ACM SIGGRAPH* Volume 30,, 29:1–29:8 115

Nowrouzezahrai, D., Kalogerakis, E. and Fiume, E. (2009): Shadowing dynamic scenes with arbitrary BRDFs. *Computer Graphics Forum*, 28 (2), 249–258 92

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E. and Purcell, T. J. (2007): A Survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26 (1), 80–113 91

Pauly, M., Kollig, T. and Keller, A. (2000): Metropolis light transport for participating media. In *Eurographics Workshop on Rendering (EGWR)*, 11–22 21

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011): Scikit-learn: Machine Learning in Python. *arXiv 1201.0490* 91

Pellacini, F. and Lawrence, J. (2007): AppWand: Editing measured materials using appearance-driven optimization. In *ACM SIGGRAPH* Volume 26,, 54 113

Pfister, H., Zwicker, M., Baar, J. van and Gross, M. (2000): Surfels: Surface Elements As Rendering Primitives. In *ACM SIGGRAPH*, 335–342 22, 32

Pharr, M. and Humphreys, G. (2010): Physically Based Rendering, Second Edition: From Theory To Implementation. 2nd edition. 15, 16

Phong, B. T. (1975): Illumination for computer generated pictures. *Communications of the ACM*, 18 (6), 311–317 72

Potmesil, M. and Chakravarty, I. (1981): A Lens and Aperture Camera Model for Synthetic Image Generation. In *ACM SIGGRAPH* Volume 15,, 297–305 28

Raab, M., Seibert, D. and Keller, A.; Keller, A., Heinrich, S. and Niederreiter, H., editors (2008): Unbiased Global Illumination with Participating Media., 591–605 23, 55

Randall, J. and Wilkins, M. (1945): The phosphorescence of various solids. *Royal Society of London*, 184 (999), 347–64 97, 99

Ren, P., Dong, Y., Lin, S., Tong, X. and Guo, B. (2015): Image based relighting using neural networks. In *ACM SIGGRAPH* Volume 34,, 111:1–111:12 90, 91, 92, 120

Ren, P., Wang, J., Gong, M., Lin, S., Tong, X. and Guo, B. (2013): Global illumination with radiance regression functions. In *ACM SIGGRAPH* Volume 32,, 130:1–130:12 90, 92, 120

Ritschel, T., Dachsbacher, C., Grosch, T. and Kautz, J. (2012): The State of the Art in Interactive Global Illumination. *Computer Graphics Forum*, 31 (1), 160–188 19

Ritschel, T., Engelhardt, T., Grosch, T., Seidel, H.-P., Kautz, J. and Dachsbacher, C. (2009a): Micro-Rendering for Scalable, Parallel Final Gathering. In *ACM SIGGRAPH Asia* Volume 28,, 132:1–132:8 24, 65

Ritschel, T., Grosch, T., Kim, M. H., Seidel, H.-P., Dachsbacher, C. and Kautz, J. (2008): Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. In *ACM SIGGRAPH Asia* Volume 27,, 129:1–129:8 25, 65

Ritschel, T., Grosch, T. and Seidel, H.-P. (2009b): Approximating dynamic global illumination in image space. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 75–82 14, 28, 39, 41, 44, 62, 77, 81, 91

Rokita, P. (1993): Fast generation of depth of field effects in computer graphics. *Computers & Graphics*, 17 (5), 593–595 28, 91

Ronneberger, O., Fischer, P. and Brox, T. (2015): U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, 234–241 93

Rushmeier, H. E. and Torrance, K. E. (1987): The zonal method for calculating light intensities in the presence of a participating medium. In *ACM SIGGRAPH* Volume 21,, 293–302 23

Saito, T. and Takahashi, T. (1990): Comprehensible rendering of 3-D shapes. In *ACM SIGGRAPH* Volume 24,, 197–206 92

Scherzer, D., Yang, L., Mattausch, O., Nehab, D., Sander, P. V., Wimmer, M. and Eisemann, E. (2012): Temporal Coherence Methods in Real-Time Rendering. *Computer Graphics Forum*, 31 (8), 2378–2408 64, 83, 120, 121

Schmidt, T.-W., Pellacini, F., Nowrouzezahrai, D., Jarosz, W. and Dachsbacher, C. (2014): State of the Art in Artistic Editing of Appearance, Lighting, and Material. In *Eurographics* 115

Schoeneman, C., Dorsey, J., Smits, B., Arvo, J. and Greenberg, D. (1993): Painting with light. In *ACM SIGGRAPH*, 143–146 112

Segovia, B., Iehl, J. C., Mitanchey, R. and Péroche, B. (2006): Non-interleaved deferred shading of interleaved sample patterns. In *ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, 53–60 36, 38

Shade, J., Gortler, S., He, L.-w. and Szeliski, R. (1998): Layered depth images. In *ACM SIGGRAPH*, 231–242 29, 44

Shanmugam, P. and Arikan, O. (2007): Hardware accelerated ambient occlusion techniques on GPUs. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 73–80 28, 44

Sloan, P.-P., Govindaraju, N. K., Nowrouzezahrai, D. and Snyder, J. (2007): Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. In *Pacific Graphics*, 97–105 27, 28, 37, 38, 43, 44, 58, 64

Sloan, P.-P., Kautz, J. and Snyder, J. (2002): Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments. In *ACM SIGGRAPH* Volume 21,, 527–536 21

Stam, J. (1995): Multiple scattering as a diffusion process. In *Eurographics*, 41–50 23

Stam, J. (1999): Diffraction shaders. In *ACM SIGGRAPH*, 101–110 115

Stamminger, M. and Drettakis, G. (2001): Interactive sampling and rendering for complex and procedural geometry. In *Eurographics Workshop on Rendering (EGWR)*, 151–162 22, 45

Straßer, W. (1974): Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten. Ph. D thesis, Technische Universität Berlin 24

Thompson, W. B., Shirley, P. and Ferwerda, J. A. (2002): A spatial post-processing algorithm for images of night scenes. *Journal of Graphics Tools*, 7 (1), 1–12 110, 116

Timonen, V. (2013): Line-Sweep Ambient Obscurance. In *Eurographics Symposium on Rendering (EGSR)*, 97–105 44

Vardis, K., Papaioannou, G. and Gaitatzes, A. (2013): Multi-view ambient occlusion with importance sampling. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D)*, 111–118 29, 44

Veach, E. and Guibas, L. J. (1997): Metropolis Light Transport. In *ACM SIGGRAPH*, 65–76 21

Wallace, J. R., Elmquist, K. A. and Haines, E. A. (1989): A Ray tracing algorithm for progressive radiosity. *ACM SIGGRAPH*, 23 (3), 315–324 39

Walter, B., Fernandez, S., Arbree, A., Bala, K., Donikian, M. and Greenberg, D. P. (2005): Lightcuts: A scalable approach to illumination. In *ACM SIGGRAPH* Volume 24,, 1098–1107 23, 24, 32, 37

Wand, M., Fischer, M., Peter, I., Heide, F. Meyer auf der and Straßer, W. (2001): The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *ACM SIGGRAPH*, 361–370 22, 45

Wang, X., Fouhey, D. F. and Gupta, A. (2015): Designing deep networks for surface normal estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 539–547 93

Weber, C., Kaplanyan, A., Stamminger, M. and Dachsbacher, C. (2013): Interactive Direct Volume Rendering with Many-light Methods and Transmittance Caching. In *International Symposium on Vision, Modeling and Visualization (VMV)*, 195–202 24, 55

Whitted, T. (1980): An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23 (6), 343–349 21

Wilkie, A., Tobler, R. F. and Purgathofer, W. (2001): Combined Rendering of Polarization and Fluorescence Effects. In *Eurographics Workshop on Rendering (EGWR)*, 197–204 115

Wilkie, A., Weidlich, A., Larboulette, C. and Purgathofer, W. (2006): A Reflectance Model for Diffuse Fluorescent Surfaces. In *International Conference on Computer graphics and InteractiveTechniques in Australasia and Southeast Asia (GRAPHITE)*, 321–331 115

Williams, L. (1978): Casting curved shadows on curved surfaces. In *ACM SIGGRAPH* Volume 12,, 270–274 25

Yuksel, C. and Keyser, J. (2008): Deep opacity maps. *Computer Graphics Forum*, 27 (2), 675–680 26, 56

Zeiler, M. D. (2012): ADADELTA: An Adaptive Learning Rate Method. *arXiv 1212.5701* 75

Zhao, H., Gallo, O., Frosio, I. and Kautz, J. (2017): Loss Functions for Image Restoration With Neural Networks. *IEEE Transactions on Computational Imaging*, 3 (1), 47–57 70, 75

Zhukov, S., Iones, A. and Kronin, G. (1998): An ambient light illumination model. In *Eurographics Workshop on Rendering (EGWR)*, 45–55 13