FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Adaptively Clustered Reflective Shadow Maps

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Christoph Weinzierl-Heigl
Matrikelnummer 0625044

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass. Dipl.-Ing. Reinhold Preiner

Wien, 24.08.2017

(Unterschrift Verfasser) (Unterschrift Betreuer)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Adaptively Clustered Reflective Shadow Maps

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Christoph Weinzierl-Heigl
Registration Number 0625044

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass. Dipl.-Ing. Reinhold Preiner

Vienna, 24.08.2017

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Christoph Weinzierl-Heigl
Bernoullistraße 4/16/10, 1220 Wien

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

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

"Big haul like that. Surprised you made it this far."

I want to thank my girlfriend *Sarah* for her patience and for pushing me to continue to work on my thesis. Additional thanks go to my friends and colleagues, *Thomas*, *Lukas*, *Florian* and *Markus* who helped me by providing ideas, proofreading and pushing me to continue my work.

A big salute to my supervisor *Reinhold* – for his support, ideas and general awesomeness – without whom this thesis would not have been possible.

"Thank you all"

Kurzfassung

In dieser Diplomarbeit präsentieren wir eine Methode zur glaubhaften Approximierung von indirekter Beleuchtung – einem Teilgebiet der globalen Beleuchtung (engl.: global illumination, abgk: GI) in der Computergraphik – welche in Echtzeit auf moderner Computerhardware berechnet werden kann. Die längste Zeit war dieses Themenfeld den sogenannten „Offline Rendering“ Methoden vorbehalten, die meistens Strahlenverfolgung (engl.: ray-tracing) zur Grundlage haben und bei denen die Generierung eines einzelnen Bildes bis zu mehrere Stunden in Anspruch nehmen kann. Diese Offline-Methoden sind großteils immer noch in der Filmbranche zur Erstellung computeranimierter Filme in Verwendung und haben erst in jüngster Vergangenheit ihren Vorstoß in die Welt der Echtzeitgraphik geschafft.

Im Vergleich zu den Offline-Methoden, wurde und wird in der Echtzeitgraphik jedoch hauptsächlich die sogenannte Rasterisierungsmethode (engl.: rasterization) angewandt, die auf standardisierter Hardware (Graphikkarten) und entsprechenden Software-Interfaces basiert. Erst durch das Aufkommen von echter, programmierbarer Graphikhardware in den ersten Jahren des 21. Jahrhunderts, sowie der seither stetig zunehmenden Rechenkapazität wurde es vor etwa einem Jahrzehnt möglich, das Thema der globalen Beleuchtung in der Echtzeitgraphik zu behandeln. Seit dieser Zeit hat eine sehr aktive Forschungsgemeinschaft eine Vielzahl an Techniken zur Approximierung von GI mittels rasterisierungsfreundlicher Methoden entwickelt.

Wir stellen eine weitere Variante vor, die als Evolutionsstufe einer Untermenge von Methoden die unter dem Begriff *Instant Radiosity* zusammengefasst werden können, betrachtet werden kann. Bei diesen Methoden wird indirekte Beleuchtung durch weitere virtuelle Lichtquellen simuliert, die an den Schnittpunkten der Lichtstrahlen mit den reflektierenden Objekten platziert werden. Unsere Methode namens *Adaptively Clustered Reflective Shadow Maps* verwendet eine neuartige, adaptive Gruppierung basierend auf dem k-Means Algorithmus um die Anzahl an notwendigen virtuellen Lichtquellen zu reduzieren, ohne dabei die Bildqualität negativ zu beeinträchtigen. Dazu wird unter anderem eine komplexere Technik zur Berechnung der indirekten Schattenwürfe angewandt. Die zuvor erwähnte Gruppierungsmethode unterstützt dabei die zeitliche Stabilität des Algorithmus auf natürliche Weise durch das wiederverwenden der Gruppierungsinformation aus dem zuvor berechneten Bild. Diese Methode wird erweitert um eine gleichzeitige Evaluierung der eigenen Ergebnisse um eventuell auftretende ungleichmäßige Verteilungen in der Gruppierung zu beheben. Unsere Resultate zeigen, dass diese neue Variante hinsichtlich Leistung und Bildqualität Vorzüge gegenüber den bisherigen Implementierungen aufweist.

Abstract

In this thesis, we present an approach to compute approximate but plausible indirect lighting, a subtopic of global illumination (GI) in computer graphics, in real-time by utilizing the capabilities of modern computer hardware. For years, this topic could only be handled in so-called offline rendering processes that mostly utilize ray-tracing techniques where single images can take up to multiple hours to generate. These offline methods are still largely used in the computer animated movies industry, and only in recent years have ray-tracing methods begun their foray into interactive and real-time computer graphics.

However, in contrast to offline methods, real-time computer graphics have traditionally been, and still are (mostly), computed using so-called rasterization methods, by using standardized hardware (graphics cards) and software interfaces. Through the advent of programmable graphics hardware in the early 2000s and seemingly ever-increasing compute capabilities, the topic of global illumination has in recent years finally stepped into the realm of real-time computer graphics. Over the past years, a very active research community produced multitudes of techniques to approximate GI using rasterization-friendly methods.

We introduce another variant that is an evolution to a subset of these methods that can be summarized under the term *instant radiosity*, where indirect lighting is simulated by placing many virtual light sources at the intersection points of light rays with reflecting geometry. Our method, called *Adaptively Clustered Reflective Shadow Maps*, uses a novel, adaptive clustering approach inspired by *k*-Means clustering, to reduce the number of required virtual lights without negatively impacting image quality through application of a sophisticated shadowing technique. The aforementioned clustering naturally facilitates temporal coherency by re-using clustering information from the previous frame while simultaneously evaluating its results to counter potential uneven cluster distributions. The results show that our new method exhibits advantages, both performance-wise and image-quality-wise, over previously employed methods.

Contents

CHAPTER 1

Introduction

As human beings, we perceive our environment through our senses, the most important one of which is the ability to *see*. All the things that we observe with our eyes are nothing more than electromagnetic energy hitting our retina, stimulating our visual nervous system, which in turn sends out signals to our brain that are then interpreted as colors. From the almost infinite range of electromagnetic frequencies like radio signals, microwaves, infrared- and ultraviolet rays, x-rays and so on we can however only see a very narrow range of the spectrum around 380-780 nanometers in length. Radiation within that wavelength will be interpreted as colors ranging from red to green to blue.

In visual computing the topic of *image synthesis* (or rendering) concerns itself with various techniques and methods to create computer-generated images of a scene. These approaches cover a broad range of subtopics, from the accurate representation of material reflectance, refraction and absorption to the accurate modeling of light sources themselves. In such a global representation of a rendering system, it is only natural to assume that objects are not only lit from direct light sources, like lamps or the sun, but also from indirect light sources via object interreflection.

Imagine any natural setting, for instance sitting at your desk with the desk lamp on and reading this thesis, in an otherwise dark room. The lamp acts as a direct light emitter. Light rays emitted from its bulb hit the wooden material of your desk as well as the paper in front of you. The room is however not completely dark otherwise, as some of the light is reflected off your desk, the paper and so on. This reflected light in turn hits other objects, walls, the ceiling, where part of it is absorbed, another part is again reflected, again hitting other objects until all energy has been absorbed. This is what is commonly known as *indirect lighting*. In computer graphics it is called *global illumination* (GI). And as it turns out, it has – for a not so brief amount of time – been a topic in visual computing that was only achievable with offline rendering solutions, where the process of image synthesis takes upwards of multiple seconds to compute a final picture. Only in recent years has our computer hardware become powerful

enough, to have the possibility to implement simple forms of global illumination that run in real-time (i.e., at or above interactive framerates) on consumer-grade machines.

1.1 Overview & Motivation

In order to realize global illumination using visual-computing methods, we employ a so-called *many-lights approach* (MLA), which uses – in addition to a direct light emitter (i.e., a light bulb, flashlight, etc.) – additional *virtual* light sources placed at intersection points of where the light rays hit surrounding surfaces to simulate the reflection of light and thus create the illusion of indirect illumination. For a convincing effect, we have to faithfully replicate common properties of indirect light such as smooth gradients, soft shadows and "light bleeding". The latter describes the effect of having the reflector's surface influence the color of the reflected (indirect) light, where for instance a bright red curtain would radiate slightly reddish indirect light when hit by light rays. Furthermore, another important requirement especially in real-time applications is that our method also supports fully dynamic scenes (i.e., moving light sources, moving/animated objects, destruction, etc.) in order to be applicable in a broad variety of situations like visualization in architecture, e-learning or games. This of course adds an additional layer of complexity to this work since we cannot rely as much on pre-computations, thus requiring a highly efficient technique that is able to generate plausible images in real-time.

However, in order to achieve this goal a few concessions have to be made especially with respect to the indirect lighting: Remember from the desk-lamp example on the first page how we described the process of indirect lighting as light rays "bouncing" around the scene from one object to another until all energy is spent. It is here, where the most computational effort would be required, and thus it is a logical step to try and reduce the number of times we "follow" these light bounces around the scene. This is especially true since we know that about 95% of the energy is spent within the first two indirect light bounces, and further research [44] revealed that even only tracing a single bounce of indirect illumination is able to produce convincing results. Thus, many real-time implementations of indirect lighting rely on tracing only a single bounce of indirect illumination. This is also one concession we make for our work.

In the research field of global illumination, the many-lights approach and the group of algorithms derived from it can be summarized under the term *Instant Radiosity* after the original method [23]. Keller was the first to propose and implement the idea of using virtual light sources to simulate indirect illumination, where the physical term *radiosity* can be described as "energy leaving a surface" and *instant* refers to the ease of implementation of this approach. Since then, a myriad of additional research on how to improve upon this method has been published and we later introduce the relevant methods and the theoretical background in more detail. Many of these publications, especially the ones aiming for real-time implementations, also rely on simplifications in order to achieve their performance goal and thus exhibit certain weaknesses, either in image quality (visual artifacts) or in their realization.

With our work, we aim to address some of these shortcomings by combining existing methods in a new and creative way while also contributing our own novel ideas as well. This thesis

expands on our previous work [49], in which we already experimented with *instant radiosity*-based approaches. We will provide a brief introduction into the theoretical background in Chapter 2 and after outlining the related work and their advantages and disadvantages we present our own contributions for this work.

1.2 Scope Of The Work

In this thesis, we present a novel approach called *Adaptively Clustered Reflective Shadow Maps* for applying "single-bounce"[1] indirect illumination for scenes consisting of arbitrary polygonal meshes.

We provide an introduction to the idea behind our approach, the algorithms and methods in use as well as implementation specifics for the integration into an existing rendering pipeline. Furthermore we provide detailed results, highlight advantages and disadvantages, analyze the difficulties of our method and how they are addressed by our implementation.

1.3 Structure Of This Thesis

This thesis is structured into the following chapters:

CHAPTER 2 provides information about the background of this work by means of introducing the reader to the concept of *Bidirectional Reflectance Distribution Functions* (BRDFs) in order to understand how surfaces reflect incoming light rays and the *Rendering Equation* (RE) as described by Kajiya [21], which mathematically explains the problem of light transport within a scene.

CHAPTER 3 evaluates related work in real-time global illumination, analyzing the strengths and weaknesses of those methods. In the end we outline how to improve upon these methods and state our own contributions.

CHAPTER 4 introduces our approach, describing the basic idea, stating prerequisites, then further moving on to detailed descriptions of the various parts of our technique.

CHAPTER 5 provides a technical description of how to integrate our method into an existing G-Buffer based rendering pipeline by exploiting already given data as well as extending the buffers as required by our technique.

CHAPTER 6 shows the results of our implementation. It gives a detailed evaluation of our method and compares the results of our work with the related work. Furthermore we outline differences, advantages, disadvantages and the limits of this thesis' work.

CHAPTER 7 summarizes the contributions and contents of this thesis. Finally, we discuss possible future work and enhancements to the technique.

[1] Only the first reflection of light from an intersected object is considered for indirect illumination.

CHAPTER 2

Background

We have already briefly touched the topic of global illumination in the introduction using familiar terms and concepts. In this chapter, we introduce the scientific background of GI that is required to understand the remainder of this thesis.

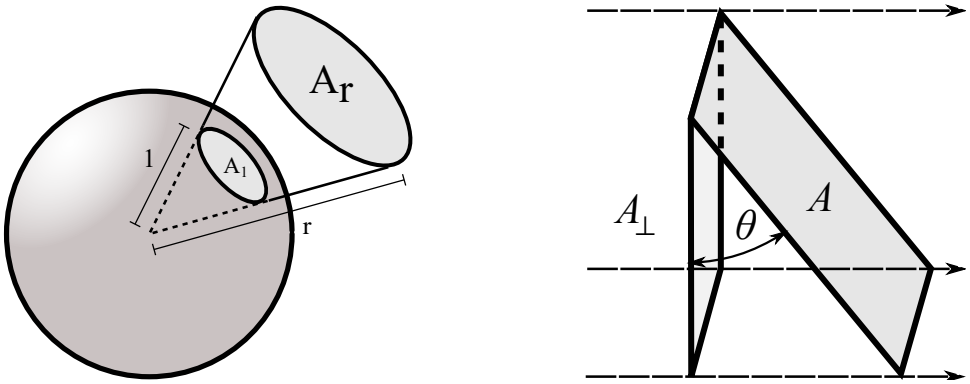
2.1 Physical Units

Since lighting is a natural phenomenon, it adheres to the laws of physics. Photons emitting from a light source, is basically energy traveling through a transmissive medium that is at some point converted into power. In computer graphics, we want to model reality and create perceptually plausible images. Hence, our models are often based on physical criteria. In this section, we will give a short introduction into the physical terms used throughout this thesis.

RADIANT FLUX (or short *flux*) is a term used for radiant power described by Φ. In physics, power is energy per unit time, measured in *Watts*, which is in turn described through Joule per second. For lighting calculations, we are talking about the energy of photons. As photons travel through a medium per unit time, their energy is converted into power.

As working with actual wavelenghts is too complicated and computationally intensive for use in real-time computer graphics, flux is often given as a discretized, tri-chromatic color value – a mixture of the red, green and blue color channels. The factor of time is inherent to the process, as our images are synthesized for a specific point in time of the scene.

SOLID ANGLE is expressed in a dimensionless unit called *steradians* (*sr*). An object's solid angle (ω) is equal to the area of the segment of a unit sphere, centered at the angle's vertex that the object covers (refer to Figure 2.1(a)). It is similar to how a planar angle in radians equals the length of an arc of a unit circle. The solid angle plays a crucial role when normalizing the transferred flux from a given light source.



(a) *The solid angle ω of a cone with radius r and base A_r is equal to the area of its projected spherical cap A_1 onto the unit sphere.*

(b) *Projecting an oblique area A onto the plane perpendicular to the ray by using angle θ yields $A_\perp$. The ray direction is given by the dashed lines.*

Figure 2.1: *Solid angle geometry in (a) and perpendicular projection in (b).*

RADIANCE is flux (Φ) per solid angle (ω) per unit area perpendicular to the ray ($A_\perp$). It is denoted by L calculated as

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega\,\mathrm{d}A_\perp}.$$

We can rewrite $A_\perp$, by projecting the surface area (A) onto the plane perpendicular to the ray via simple trigonometry (refer to Figure 2.1(b)) as $A_\perp = A\cos\theta$, which yields

$$L = \frac{\mathrm{d}\Phi}{\mathrm{d}\omega\,\mathrm{d}A\cos\theta}.$$

IRRADIANCE is the flux incident on a surface per unit surface area. In a simple form it is described by $E = \frac{\Phi}{A}$ or for infinitesimal amounts of flux and area it can be $E = \frac{\mathrm{d}\Phi}{\mathrm{d}A}$. We can easily relate this to the radiance given above as,

$$L\cos\theta\,\mathrm{d}\omega = \frac{\mathrm{d}\Phi}{\mathrm{d}A} = E. \tag{2.1}$$

RADIOSITY also known as *Radiant Exitance*, is the flux leaving a surface (i.e., going in the opposite direction) per unit surface area. It is given by $B = \frac{\Phi}{A} = \frac{\mathrm{d}\Phi}{\mathrm{d}A}$ (in some publications B can be denoted as M). Again, we can easily spot the similarity between irradiance and radiosity. The difference between those two is in the nomenclature, where irradiance means flux *received* by a surface, while radiosity means the flux *leaving* a surface.

2.2 Bidirectional Reflectance Distribution Function

In order to describe and understand light transport in a scene, we first have to understand how surface properties influence the paths of incoming light. Fred Nicodemus first introduced the

Bidirectional Reflectance Distribution Function (BRDF) [32] in 1965. It was the first comprehensive model to describe surface properties in a physical way and is used to describe effects for light reflected into the upper hemisphere of a sphere surrounding a point $\vec{x}$ that is aligned along its surface normal $\hat{\mathbf{n}}$.

In its original form, the BRDF is a parametrized function defined by,

$$f_r(\vec{x}, \omega_i, \omega_o) = \frac{L_o(\vec{x}, \omega_o)}{E_i(\vec{x}, \omega_i)}$$

where $\vec{x}$ is a positional vector and ω_i, ω_o are directional vectors denoting the *incident* (or incoming) direction and the *reflected* (or outgoing) direction. L and E, like discussed before, denote the radiance and irradiance respectively. The BRDF gives the proportion of light reflected from the incoming direction ω_i into outgoing direction ω_o at position $\vec{x}$.

Similar to Equation (2.1), we can easily rewrite it as,

$$f_r(\vec{x}, \omega_i, \omega_o) = \frac{L_o(\vec{x}, \omega_o)}{L_i(\vec{x}, \omega_i) \cos\theta_i \, \mathrm{d}\omega_i} \tag{2.2}$$

where θ_i gives the angle between the surface normal $\hat{\mathbf{n}}$ and the incoming direction ω_i. Since the incoming radiance for any given volumeless direction is zero, we actually have to consider infitesimal solid angles $\mathrm{d}\omega_i$ around the incoming direction ω_i (see Figure 2.2).

More complex (i.e., higher dimensional) variations of a BRDF with $f_r(\vec{x}, \omega_i, \omega_o, \ldots)$ can account for additional effects like polarization, phosphorescence and fluorescence. We do not deal with such effects in this work, hence we also do not go into detail regarding those specializations.

For the sake of completeness, it is worth noting that there also exist functions that take care of effects regarding the lower hemisphere (i.e., when using transparent objects). In this case, light is not only *reflected* off the surface, but also *transmitted* through it, in which case we also have to concern ourselves with *Bidirectional Transmittance Distribution Functions* (BTDFs). A combination of both these variants is termed *Bidirectional Scattering Distribution Functions* (BSDFs).

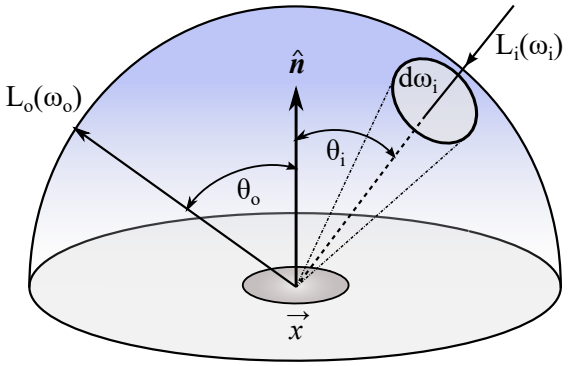


Figure 2.2: *The BRDF hemisphere is centered around $\vec{x}$ and aligned according to its corresponding normal $\hat{\mathbf{n}}$.*

2.2.1 Characteristics

Physically-based BRDFs have a few key characteristics that we will briefly touch below.

BIDIRECTIONALALITY As the name suggests, BRDFs are *bidirectional*. This means that switching the incoming and outgoing directions will not change the result of the BRDF: $f_r(\vec{x}, \omega_i, \omega_o) = f_r(\vec{x}, \omega_o, \omega_i)$. This characteristic is called *Helmholtz Reciprocity*.

POSITIVITY The amount of energy (reflected and incident) cannot be negative: $f_r \geq 0$.

ENERGY CONSERVING The amount of *reflected* energy cannot exceed the amount of incident energy: $f_r(\vec{x}, \omega_i, \omega_o) \leq 1$.

2.2.2 Categorization

Material reflectance properties can be classified into four main groups. Since we are dealing with analytical BRDFs later in this chapter, it is important to understand these major reflector categories (see Figure 2.3):

DIFFUSE reflectors have the characteristic of reflecting incoming light equally into all directions. Hence, they are independent of the outgoing direction ω_o. In reality, no material is an ideal diffuse reflector, but for instance materials with rough surfaces like natural, unpolished wood exhibit the main characteristics of a diffuse reflector.

ROUGH SPECULAR (also known as DIRECTIONAL DIFFUSE) materials concentrate the distribution of reflected light around the general direction of ω_o. The *glossiness* of a material describes the variance in the reflection direction. That is, a higher glossiness results in a more mirror-like behavior. Glossy specular surfaces show a highlighted region around the general reflection direction. Many polished materials like coated wood or clean metal display such a behavior.

PERFECTLY SPECULAR surfaces reflect light *only* into the reflection direction ω_o. In practice, such materials do not exist, as even mirrors that appear to be perfect reflectors have small imperfections in their surface structure, so called *microfacets*.

RETRO-REFLECTIVE materials scatter most of the incoming light back into the general direction of where they originated. Such materials are quite common in day to day life and are used on road surfaces and signs, safety clothing, etc.

Most natural materials cannot be strictly classified into one of those categories but instead are a mixture of at least two of them. Another categorization is based on whether a material exhibits *isotropic* or *anisotropic* reflective properties: Isotropic reflections are rotational invariant which is the case for a large number of materials. Anisotropic reflections on the other hand change with respect to the rotation of the surface around its normal vector. Examples are radially brushed metals or optical discs.

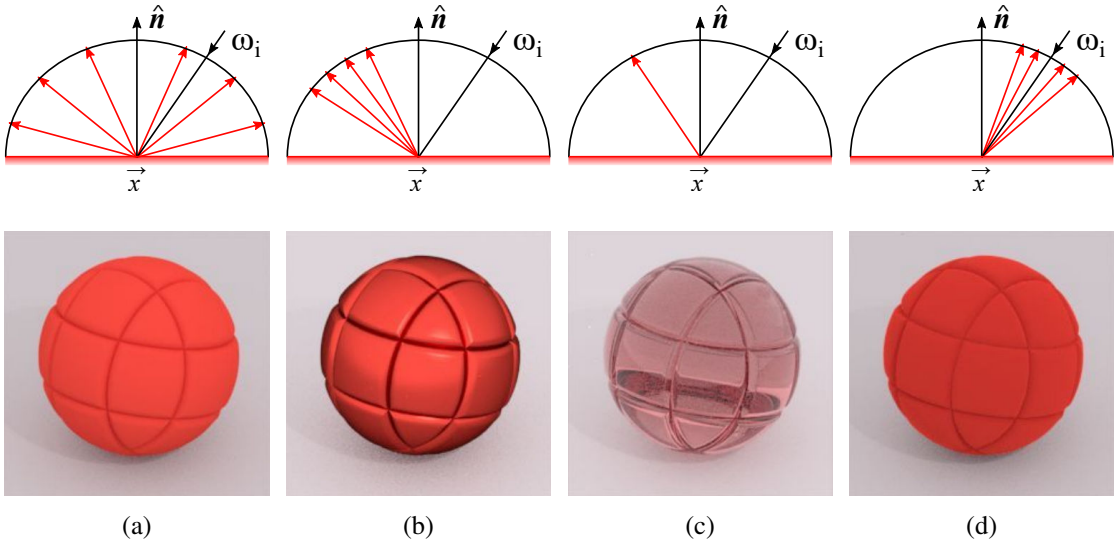


Figure 2.3: *Each column represents the reflective properties (top) and an example render (bottom): (a) ideal diffuse, (b) rough specular, (c) perfect specular and (d) retro-reflective. Renderings courtesy of Andrea Weidlich.*

2.2.3 Models

To represent the material characteristics given above we either need mathematical models or raw data captured from real materials. As such, BRDF data can be captured from physical objects by using *Gonioreflectometers*. However, capturing this data is a very time-consuming and memory-intensive task that is most often not performed for real-time applications. Instead, we often rely on simplified analytical models that are easier to evaluate.

Analytical models are often expressed as a combination of three components: An 1) *ambient* component I_a, 2) a *diffuse* component I_d and 3) a *specular* component I_s. These components are usually calculated separately and various models for the various parts exist. Since most materials are represented by a combination of these classifications, the analytical models are mostly calculated independently but are superimposed for final composition as $I(I_a, I_d, I_s)$. These ambient, diffuse and specular terms are thereby often weighted by coefficients k_a, k_d and k_s.

It is worth mentioning that the ambient component historically was given by $I_a = k_a$. This was born out of necessity for use in computer graphics, where k_a is a static reflectance term that was meant to be used as a drop-in solution to account for ambient lighting before global illumination methods (either pre-computed or generated in real-time) took its place and allowed the evaluation of more complex I_a functions.

The following list provides an overview of some popular analytical variants which can be further categorized into *empirical* and *physically based* models.

Empirical

Empirical models are created from observation and testing. They simulate physical behavior but are not meant to be accurate, just plausible. Superpositions of multiple models are often used to gain a higher degree of realism. Many well-known models are of this type:

LAMBERT: This model is named *Lambert's Cosine Law* after Johann Heinrich Lambert [26] and applies for diffuse surfaces only. For application in computer graphics, it is usually given as $I_d = k_d \cdot \cos\theta$, which states that the radiant intensity I_d observed on an ideal diffuse reflector k_d is directly proportional to the cosine of the angle θ between the surface normal $\hat{\mathbf{n}}$ and the direction of the incident light $\hat{\boldsymbol{l}}$. Since we concern ourselves only with reflections on the positive hemisphere around the surface normal we can clamp any value below 0° or above 180° to zero. Thus, it can be rewritten as $I_d = k_d \cdot \max(0, \left\langle \hat{\mathbf{n}} | \hat{\boldsymbol{l}} \right\rangle)$.

PHONG: this model for specular illumination by Phong [34] approximates the highlights of glossy materials (as described in Section 2.2.2) through $I_s = k_s \cdot \max(0, \langle \hat{\mathbf{v}} | \hat{\mathbf{r}} \rangle^{\alpha})$ where $\hat{\mathbf{v}}$ is the viewing direction, $\hat{\mathbf{r}}$ is the reflection of light direction $\hat{\boldsymbol{l}}$ around the surface normal, α is a glossiness factor and k_s the specular reflection coefficient. Since Phong's model only calculates specular highlights, it is often superimposed with a diffuse lighting model like Lambert's to achieve a convincing illumination.

BLINN: Recalculating $\langle \hat{\mathbf{v}} | \hat{\mathbf{r}} \rangle$ for varying light directions as in the Phong model was - especially at the time of its invention - a computationally heavy task. For efficiencies sake James Blinn [5] came up with a modification of the Phong model, thus sometimes called *Blinn-Phong* model, which is calculated by $I_s = k_s \cdot \max(0, \left\langle \hat{\mathbf{n}} | \hat{\mathbf{h}} \right\rangle)^{\alpha'}$. Here, the $\langle \hat{\mathbf{v}} | \hat{\mathbf{r}} \rangle$ dot product is substituted by $\left\langle \hat{\mathbf{n}} | \hat{\mathbf{h}} \right\rangle$, where

$$\hat{\mathbf{h}} = \frac{\hat{\boldsymbol{l}} + \hat{\mathbf{v}}}{\left\| \hat{\boldsymbol{l}} + \hat{\mathbf{v}} \right\|}. \tag{2.3}$$

is the halfway vector between the light direction and the viewing direction. This has a significant benefit for so called *directional lights*, which model light sources that are very far away from the surface such as the sun, where the light direction is assumed to be constant. In this case, the halfway vector becomes a quasi-constant that can be calculated once per light, instead of having to be recalculated at each pixel, therefore drastically reducing the computational effort. Whereas Phong highlights always appear circular, Blinn highlights become elliptical when viewed from a steep angle which is more realistic. This behavior is due to its usage of the halfway vector which was inspired by physically based microfacet BRDFs (see below). Similar to the Phong model, it must be combined with a diffuse illumination model to yield a plausible shading.

Physically based

Phyiscally based models are built to be physically accurate, hence are usually more complex in terms of the maths involved, harder to implement and do also often require knowledge of physical material constants.

COOK-TORRANCE: Similar to the Phong and Blinn models discussed above, this model also models specular reflectance, hence it must be combined with a diffuse model. Cook and Torrance's model is based on simulating surface microfacets, where the material's roughness (m) decides the orientation of the facets. On smooth surfaces, neighboring microfacets are oriented similarly, whereas on rough materials their orientation varies greatly. It is calculated as

$$I_s(\hat{\mathbf{n}},\hat{\boldsymbol{l}},\hat{\mathbf{v}},m,f_\lambda) = \frac{F(f_\lambda)\cdot D(\hat{\mathbf{n}},\hat{\mathbf{h}},m)\cdot G(\hat{\mathbf{n}},\hat{\boldsymbol{l}},\hat{\mathbf{v}})}{\pi\cdot\left\langle\hat{\mathbf{n}}|\hat{\boldsymbol{l}}\right\rangle\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{v}}\rangle}$$

where $\hat{\mathbf{n}},\hat{\boldsymbol{l}},\hat{\mathbf{v}}$ are the known directional vectors for surface normal, light direction and reflection direction. The *Fresnel factor F* is computed via *Schlick's approxmiation* [42]

$$F(f_\lambda) = f_\lambda + (1-f_\lambda)\cdot(1-\langle\hat{\mathbf{h}}|\hat{\mathbf{v}}\rangle)^5$$

where the reflection coefficient at normal incidence is given by f_λ. The *microfacet distribution* (roughness) is calculated as

$$D(\hat{\mathbf{n}},\hat{\mathbf{h}},m) = \frac{1}{\pi\cdot m^2\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{h}}\rangle^4}\cdot e^{\left(\frac{\langle\hat{\mathbf{n}}|\hat{\mathbf{h}}\rangle^2-1}{m^2\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{h}}\rangle^2}\right)}.$$

where $\hat{\mathbf{h}}$ is the halfway vector as defined in Equation (2.3). Finally, the *geometrical attenuation* is given by

$$G(\hat{\mathbf{n}},\hat{\boldsymbol{l}},\hat{\mathbf{v}}) = \min\left(1, \frac{2\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{h}}\rangle\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{v}}\rangle}{\langle\hat{\mathbf{v}}|\hat{\mathbf{h}}\rangle}, \frac{2\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{h}}\rangle\cdot\left\langle\hat{\mathbf{n}}|\hat{\boldsymbol{l}}\right\rangle}{\left\langle\hat{\boldsymbol{l}}|\hat{\mathbf{h}}\right\rangle}\right).$$

This factor determines how incoming light is influenced by the microfacet structure of the surface and therefore how much light is reflected or absorbed by self-shadowing or masking effects.

The interested reader is referred to Cook and Torrance's publication [8] for a more detailed discussion of the model and its physical background. We can however easily grasp that the computational effort is much higher than for the simple empirical models discussed above. Nonetheless, methods of this complexity have already found their way into real-time applications [22, 43], resulting in much more realistic shading.

WARD: As an example of an anisotropic model, Ward's BRDF [47] is based on elliptical Gaussian distributions. His work is based on real measurements from a gonioreflectometer. The important specular term is represented by

$$I_s(\hat{\mathbf{n}},\hat{\boldsymbol{l}},\hat{\mathbf{v}},\hat{x},\hat{y},\alpha_x,\alpha_y) = k_s\cdot\frac{1}{4\pi\alpha_x\alpha_y}\cdot\frac{1}{\sqrt{\left\langle\hat{\mathbf{n}}|\hat{\boldsymbol{l}}\right\rangle\cdot\langle\hat{\mathbf{n}}|\hat{\mathbf{v}}\rangle}}\cdot e^{\left[\frac{(\langle\hat{\mathbf{h}}|\hat{x}\rangle/\alpha_x)^2+(\langle\hat{\mathbf{h}}|\hat{y}\rangle/\alpha_y)^2}{\langle\hat{\mathbf{h}}|\hat{\mathbf{n}}\rangle^2}\right]}$$

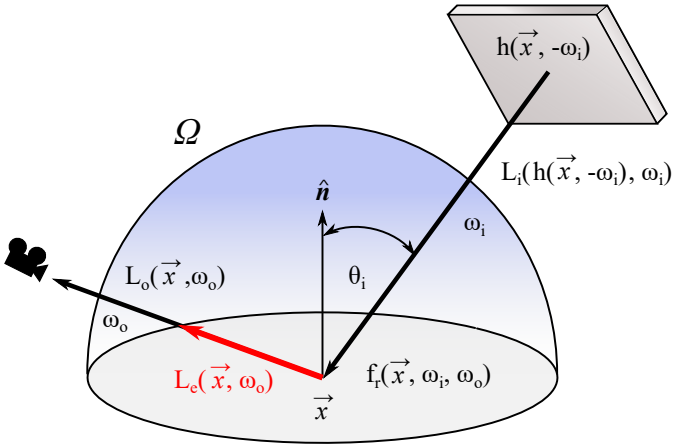


Figure 2.4: *Showcases the various different parts of the Rendering Equation in graphical form.*

where α_x and α_y are material constants that describe the size of the highlights, while $\hat{x}$ and $\hat{y}$ are perpendicular, directional vectors on the surface plane that describe the orientation of the elliptical highlight.

Many other analytical models exist today. The ones presented above belong to the more prominent BRDFs used in computer graphics. The work in this thesis is however only based on Lambert's diffuse and Blinn's specular model. In particular our radiosity-based global illumination method only calculates diffuse interreflection.

2.3 The Rendering Equation

After an understanding of material reflectance properties and how to mathematically describe them has been established, we can now formally describe how light emitted by a light source propagates through space. A mathematical way to formulate this was introduced in 1986 by James Kajiya [21]: the *Rendering Equation* (RE). There are multiple possible formulations and variants of his equation. In this work, we focus is on the *directional formalism* (see Figure 2.4 for a graphical interpretation):

$$L_o(\vec{x},\omega_o) = L_e(\vec{x},\omega_o) + \int_\Omega L_i(h(\vec{x},-\omega_i),\omega_i)\cdot f_r(\vec{x},\omega_i,\omega_o)\cdot\cos\theta_i\,\mathrm{d}\omega_i \qquad \textbf{(2.4)}$$

Parts of the Rendering Equation

$L_e(\vec{x},\omega_o)$	Denotes radiance emitted from position $\vec{x}$ into direction ω_o.
$\int_\Omega \ldots \mathrm{d}\omega_i$	The integral that computes the irradiance over the hemisphere Ω that contains all directions ω_i and is centered around the surface normal $\hat{\mathbf{n}}$ at $\vec{x}$.

$L_i(h(\vec{x}, -\omega_i), \omega_i)$	Specifies incoming radiance at point $\vec{x}$ from direction ω_i. L_i is parameterized by the visibility function h, which determines whether $\vec{x}$ is visible from direction ω_i.
$f_r(\vec{x}, \omega_i, \omega_o)$	The BRDF (discussed in Section 2.2) giving the proportion of light reflected from ω_i to ω_o at position $\vec{x}$.
$\cos\theta_i$	The weakening factor of incoming radiance in order to transform it to irradiance (see Section 2.1). In computer graphics this is often rewritten as a dot product given by $\langle \hat{\mathbf{n}} \vert \omega_i \rangle$.
$L_o(\vec{x}, \omega_o)$	Finally, based on the emittance term and the integral over Ω, the RE gives the overall radiosity leaving the surface at position $\vec{x}$ into direction ω_o.

It becomes clear that when interpreting the rendering equation, the outgoing radiance L_o of one surface point will have to be considered when determining the incident radiance L_i at another point. By introducing an integral operator T for $\int_\Omega \ldots \mathrm{d}\omega_i$ the equation can be rewritten in a short form as

$$L_o = L_e + TL_{o'}$$

which can be further expanded into the following, recursive form

$$L_o = L_e + T(L_{e'} + T(L_{e''} + T(\ldots))).$$

The Radiosity Equation

The method described in this thesis primarily uses diffuse interreflection, which is characterized by the fact that it distributes light equally in all directions ω_o. This allows a reduction of the BRDF term $f_r(\vec{x}, \omega_i, \omega_o)$ to $f_d(\vec{x})$. It then only represents the diffuse reflectivity at position $\vec{x}$, which allows further simplification of the rendering equation into the following form:

$$L_o(\vec{x}, \omega_o) = L_e(\vec{x}, \omega_o) + f_d(\vec{x}) \int_\Omega L_i(h(\vec{x}, -\omega_i), \omega_i) \cdot \cos\theta_i \, \mathrm{d}\omega_i. \tag{2.5}$$

In this variant, $f_d \equiv f_r$ moves out of the integral, which means that the BRDF term does not have to be evaluated with respect to ω_o for each incoming direction ω_i over the hemisphere Ω. Instead, f_d remains static per position $\vec{x}$, which reduces the complexity significantly. This yields the so called *radiosity equation* [15], which includes a number of variable substitutions and describes the light transport rather in terms of relationships between two points in space (x, x'):

$$\begin{aligned} B(x) &= E(x) + f_d(x) \int_S B(x') \cdot G(x, x') \cdot V(x, x') \, \mathrm{d}A' \\ &= E(x) + f_d(x) \int_S B(x') \cdot F(x, x') \, \mathrm{d}A' \end{aligned} \tag{2.6}$$

where, L_i, L_o have been substituted with B, the emittance term L_e is now given by E, the integral over the hemisphere is given by $\int_S$ where S is the entirety of all discretized surfaces in the scene.

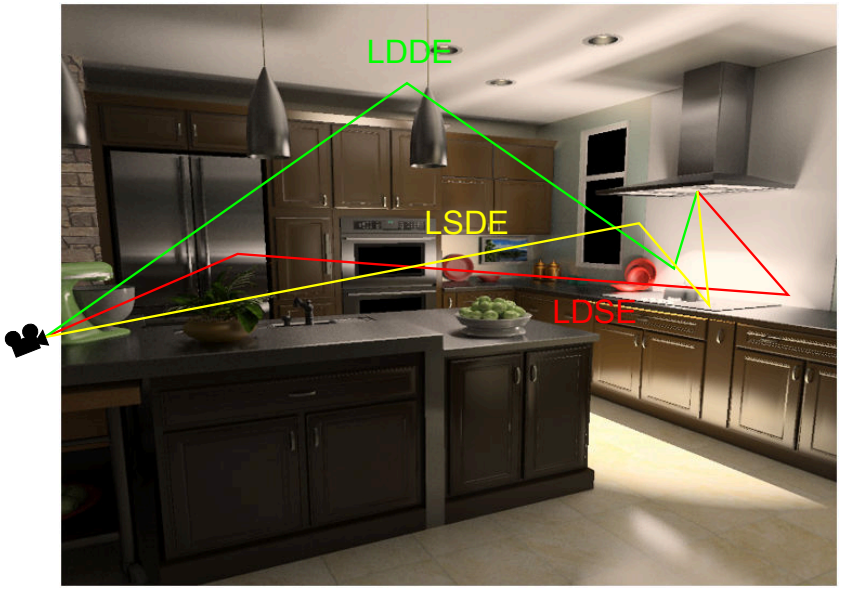


Figure 2.5: *Rendering of a complex scene with a global illumination renderer. Depicted are some possible light paths using Heckbert's notation. Image courtesy of Hašan et al. [18].*

$\cos\theta_i \, \mathrm{d}\omega_i$ has been replaced with the geometry term $G(x,x')\,\mathrm{d}A'$. Furthermore, the visibility function h is rewritten as $V(x,x')$. G and V are often further combined into a *form factor* $F(x,x')$, which gives the proportion of energy transferred from x' to x. This notation will be used in later sections of this work.

The Light Transport Notation

Before introducing the most common methods of solving the rendering and the radiosity equation, a notation by Heckbert [19] to classify those solvers is presented first: he *Light Transport Notation*. It is used to describe the type of rays that occur when light is transported through the scene, of which his notation identifies four:

L is used for rays that originate from a light emitting source,

D describes a diffuse light bounce[1],

S describes a specular light bounce and

E is a ray that travels to the eye/view point.

By expanding the notation further with operators inspired from Regular Expressions

? zero or one occurence,

[1] A *bounce* describes a ray that originates from reflecting off of an intersecting surface.

***** zero or more occurrences,

+ one or more occurrences, as well as

| that allows to decide between multiple selections

it becomes more flexible and easier to depict complex ray variations. Figure 2.5 shows an example of possible combinations of paths in a scene. The single-bounce radiosity solver presented in this thesis, can thus be categorized using Heckbert's notation as an **LDE** solver.

Solving the Rendering Equation

The inherent coupling between the terms L_o and L_i makes analytical solutions of the RE mostly impossible. However, due to that same feature we can immediately spot a possibility to employ an iterative (or recursive) algorithm to numerically integrate and approximate a solution.

In visual computing, two main groups of algorithms to numerically approximate the rendering equation emerged:

RADIOSITY: The original Radiosity-approach (although related, not to be confused with the physical term described in Section 2.1) by Goral et al. [15] is a finite-element method that accounts only for diffuse interreflection of type **LD*E** and is thus meant for solving the RE given in Equation (2.5). It either iteratively or recursively divides the surfaces of a scene into smaller subsurfaces, called *patches*. For each of those patches, it calculates a *form factor* (sometimes also called view factor), which is a coefficient that quantifies the visiblity between two patches: patches that are farther away from each other or that are oriented at oblique angles will have smaller form factors. If the visibility is (partially) blocked, the form-factor will be reduced or equal to zero. These form-factors act as inputs for a system of linear equations that yields the brightness of each patch.

Nowadays, this method is not as widely used any more, especially when talking about real-time solutions. It has been altered and adapted, and as with many other methods that have evolved over time, the original term *Radiosity* has however stuck and is still commonly used when talking about global illumination and diffuse interreflection.

MONTE CARLO: These kind of methods are based on pseudo-random sampling patterns to achieve numerical solutions for non-analytic problems with bad convergence behavior. They are not only used for global illumination but are applicable to a broad field of computational algorithms. Since the RE's integral is almost impossible to solve analytically because an infinite amount of incoming directions (ω_i) would have to be considered, an approximation using only a finite subset of directions that are pseudo-randomly distributed over the hemisphere is sufficient to create a converging solution. Thus, *Monte Carlo*-based methods lend themselves well to this kind of problem and allow the creation of solvers that are able to approximate the "full" variant of the Rendering Equation (refer to (2.4)) consisting of **L(S|D)*E** paths.

Various different global illumination solvers exist that are based on Monte Carlo methods, such as *Ray Tracing* [50], *Path Tracing* [21], *Photon Mapping* [20] or *Metropolis Light Transport* [45]. The general idea behind most of them is, to shoot rays in an informed random direction into the scene and follow their paths (influenced by reflection, absorption, etc.) through the scene. By using a good sampling pattern and a repeated application of this method, the numerical integration converges towards a correct solution of the rendering equation.

Historically, both, Radiosity- and Monte Carlo-based approaches have their roots in offline rendering solutions. As computers and graphics hardware became more powerful over the years, radiosity-inspired methods however became more prevalent in real-time applications as its ideas were more easily integrable into the standard rasterization pipeline, as will be discussed in the next section. With the advent of compute languages such as `OpenCL` or `CUDA`, it is nowadays also possible to create Monte Carlo-based real-time ray-tracing solutions that run entirely on programmable graphics hardware.

CHAPTER 3

Related Work

Now that the most important aspects of the theory behind global illumination have been covered, we give an introduction to the related work that is in large parts based on the previously presented methodologies. However, before going into detail on the radiosity-based approaches (starting in Section 3.2.1), we also require the reader to understand the complexities of shadows and how they relate to global illumination. The following paragraphs should act as an introduction to this topic as far as our related work is concerned. A more complete discussion of the various existing shadow generation methods is given in state-of-the-art reports [13, 48].

3.1 Shadow Mapping

Shadow generation is an important topic in real-time computer graphics, as shadows provide important visual cues about the scene that would otherwise be hard or even impossible to grasp:

- Shadows aid in understanding relative object positions and dimensions in a scene. Without a cast shadow, it is not easily possible to visually determine the location of an object.
- They facilitate understanding the geometric complexity of the shadow receiver and occluder.

In real-time applications, shadows are generated for a low number of light sources, as each shadow casting light source requires its own shadow computation. With each additional light source, the number of shadow computations required during illumination increases. This problem persists for global illumination methods, which is why a lot of the subsequently discussed methods try to either reduce the amount of shadow maps that are needed (or have to be updated) [25, 11], or to speed up the rendering of large amounts of shadow maps [39, 41]. A characteristic feature of shadows generated from indirect light is that they are usually very subtle. Hence, certain other methods [36] do not generate shadows from indirect lighting at all, since even without *indirect* shadows convincing images can be produced [44].

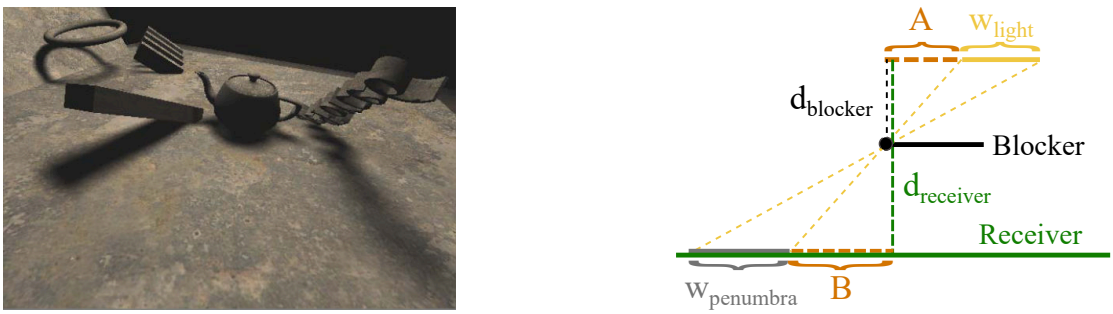
The most common method for shadow generation in real-time applications is *Shadow Mapping* [51]. It is an image-based approach that produces a two-dimensional depth map – called *shadow map* – that is constructed from the point of view of the light source. Each pixel p of the shadow map stores a depth value $d(\vec{x})$ for a world-space position $\vec{x} \in \mathbb{R}^3$. The translation from world-space position to shadow map pixel is constructed via a surjective mapping $T : \mathbb{R}^3 \to \mathbb{R}^2$, such that $p = T(\vec{x})$. The shadow map itself encodes a function $z(p)$ that represents the depth of the blocker that is closest to the light source for each p. To define whether a pixel with world-space position $\vec{x}$ is considered to be in shadow the corresponding binary shadow test $f(d,z)$ is evaluated for each single pixel as

$$f(d(\vec{x}), z(p)) = \begin{cases} 0 & \text{if } d(x) > z(p) \\ 1 & \text{otherwise,} \end{cases} \tag{3.1}$$

where $d(\vec{x})$ gives the depth value of the world-space position. Another way to describe f is via a unit step function, $H(t)$, where $f(d,z) = H(d-z)$. The step, from "in-shadow" to "lit" occurs where $d - z = 0$. This further emphasizes the fact that this simple evaluation, produces hard shadow boundaries. However, as stated above, we expect very soft shadows with smooth gradients from indirect illumination, which simply cannot be achieved this way without applying additional filtering.

3.1.1 Percentage-Closer Soft Shadows

The so called *Percentage-Closer Soft Shadows* (PCSS) by Fernando [14] is a shadow filtering method that is based on a variable multi-sampling approach. It aims to generate more realistic soft shadows that exhibit characteristics such as contact hardening.



(a) (b)

Figure 3.1: *(a) Various shapes with PCSS shadows. We can observe effects such as hardening on contact (beam on the left) and softer shadows when the distance between blocker and receiver increases (object on the right). (b) Basic idea of the PCSS algorithm based on similar triangles. Image courtesy of Fernando [14].*

Outline

The idea behind PCSS is to perform multiple binary shadow tests for a single pixel that also incorporates neighboring texels in the shadow map. The results of the depth comparisons are averaged, giving a value *between* zero and one (rather than only 0 or 1), which softens the resulting shadow. This technique is known as *Percentage-Closer Filtering* (PCF) [37]. By adaptively varying the sizes of the filter kernel, different degrees of soft shadows can be created, i.e., small kernels give harder shadows, large kernels produce softer shadows. Figure 3.1(a) gives examples of the results that can be achieved using this technique.

Soft Shadows Through Varying Filter Sizes

Fernando employs a simple heuristic to determine the penumbra[1] size

$$w_{penumbra} = \frac{(d_{receiver} - d_{blocker}) \cdot w_{light}}{d_{blocker}} \tag{3.2}$$

that is based on

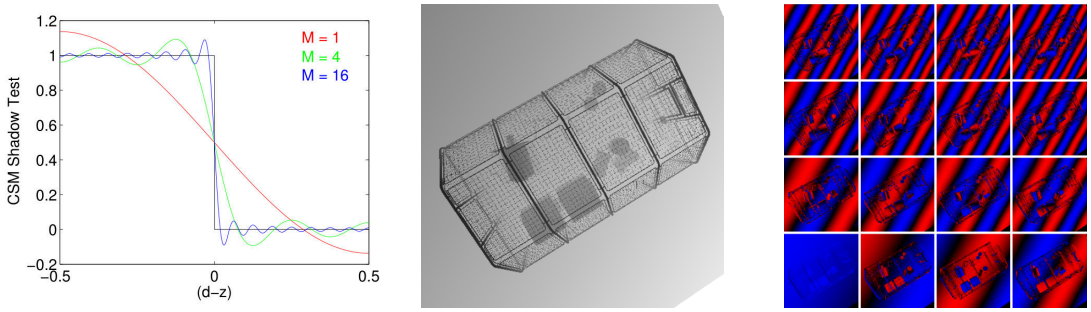
- occluder depth ($d_{blocker}$),
- receiver depth ($d_{receiver}$) and
- light source dimension (w_{light}).

The receiver depth is trivially acquired for each shaded pixel by performing a shadow map lookup at its pixel location. Also the light source's dimension can be easily supplied to the shader from an external source such as a texture or as a uniform variable. For $d_{blocker}$, a simple search algorithm – the *blocker search* – is applied to find the average occluder depth. Its search region depends on the size of the light source as well as the distance to the light. Finally, the resulting value $w_{penumbra}$ can be used to vary the parameters of the PCF kernel. See Figure 3.1(b) for a visualization of the equation.

Conclusions

PCF & PCSS: Employing PCF during shadow computation is rather trivial since it seamlessly replaces a traditional shadow map lookup. It is able to produce far more natural looking shadows and the computational effort scales with the size of the filter kernel. The variable filter size that is used for PCSS, is able to simulate even more perceptually plausible shadows that exhibit certain features like hardening on contact that would be expected from real soft shadows.

[1] Shadow regions can be classified into two regions: The *umbra* defines the core shadow, where the occluder completely hides the light source. The *penumbra* is the region where part of the light source is visible. This degree of visibility varies throughout the penumbra region: near the umbra, only a small area of the light source is visible, on the other side of the penumbra, almost the entire light source becomes visible until the shadow completely vanishes.



(a) (b) (c)

Figure 3.2: *(a) Approximation of the shadow test step function using Fourier expansion for different orders M. (c) Basis images for $M = 16$ generated from a linear depth map shown in (b). Images courtesy of Annen et al. [1].*

3.1.2 Convolution Shadow Maps

While PCF-based shadow filtering achieves softer shadow boundaries, it is not entirely optimized for graphics hardware, since it cannot apply cheap pre-filtering techniques such as bi-linear or tri-linear filtering. This is due to the pre-filtering occurring *before* the shadow test, which would first average the depth values from the shadow map and then perform the $f(d,z)$ test. PCF on the other hand filters *after* the shadow test. *Convolution Shadow Maps* (CSMs) [1] introduce a way to apply standard pre-filtering techniques as well as other filters such as Gaussian blur directly to the shadow map.

Outline

To this end, Annen et al. show a way to perform a Fourier series expansion on the shadow test function as given in (3.1). They demonstrate that $f(d,z)$ can be made separable with respect to d and z in order to perform a pre-convolution of the shadow function z. In other words, this separability allows for pre-filtering of the shadow map.

Convolution Of The Shadow Test

Separability is achieved by transforming the z-values such that the shadow test $s(\vec{x}) = f(d,z)$ can be written as a sum. Its finite expansion up to truncation order N is given by

$$s(\vec{x}) = \sum_{i=1}^{N} a_i(d) B_i(z) \tag{3.3}$$

where B_i are basis funcions in term of z and a_i are the corresponding coefficients depending on d. In practice, this means that the shadow map is converted to "basis images" by applying each basis function to the shadow map as $B_i(z(p))$. The authors demonstrate that this expansion can

be applied to a convolution of the shadow function $s_f(\vec{x})$ such that

$$s_f(\vec{x}) = \sum_{i=1}^{N} a_i(d(\vec{x}))\,[w * B_i]\,(p) \tag{3.4}$$

where w is the convolution kernel. This forms their key observation, which states that "any convolution operation on the shadow function is equivalent to convolving the individual basis images $B_i(z(p))$". This decoupling of $d(\vec{x})$ from $z(p)$ is important, because it allows convolving the basis images $B_i(z(p))$ *before* the shadow test.

In order to perform the Fourier series expansion on the shadow function $f(d,z)$, a representation based on the unit step function $H(t)$ where $f(d,z) = H(d-z)$ is used. An example of this approximation is given in Figure 3.2(a). The final separable shadow test function is given by

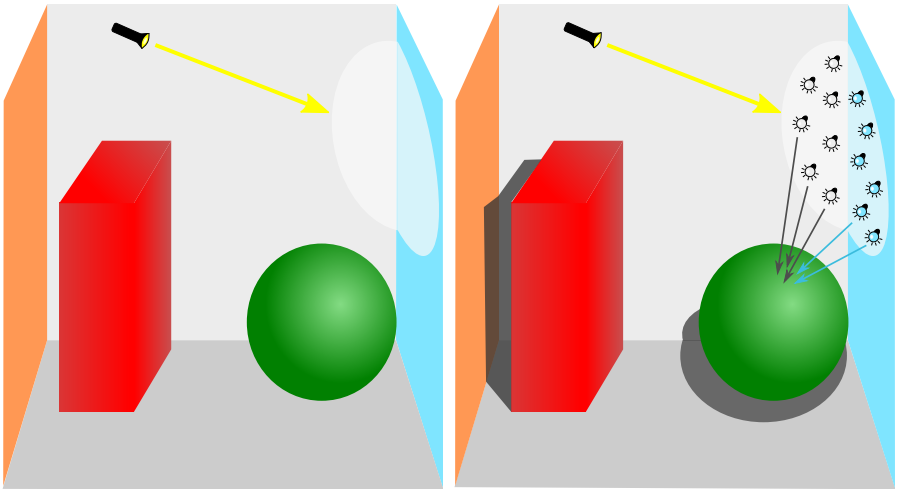
$$\begin{aligned} f(d,z) \approx \frac{1}{2} &+ 2\sum_{k=1}^{M} \frac{1}{c_k}\cos(c_k d)\sin(c_k z) \\ &- 2\sum_{k=1}^{M} \frac{1}{c_k}\sin(c_k d)\cos(c_k z) \end{aligned} \tag{3.5}$$

where $c_k = \pi(2k-1)$ and $N = 2M$. The terms $\sin(c_k z)$ and $\cos(c_k z)$ are computed based on the shadow map and stored in additional texture maps (see Figure 3.2 (b) and (c)) depending on the order M. On these texture maps, arbitrary filtering functions can be applied to yield smoother shadow terms. For the complete derivation of Equations (3.4) and (3.5) as well as the Fourier expansion of $H(d-z)$ please refer to the work by Annen et al. [1].

Conclusions

CONVOLUTION: The approximation of the shadow test function using a Fourier expansion allows for using cheap filtering techniques such as bi-linear and tri-linear filtering. This reduces shadow aliasing and is therefore a simple yet effective method to achieve soft shadows.

BASIS IMAGES: A possible downside are the increased memory requirements for storing the basis images compared to a straight-forward approach such as PCF.



(a) (b)

Figure 3.3: *The Instant Radiosity approach: In (a), the scene is lit only directly and light rays are traced from the main light source. At the intersection points with scene geometry, VPLs are placed and used to simulate indirect illumination in (b).*

3.2 Radiosity Methods

As the work presented in this thesis is mostly based on the radiosity approach described above, the following sections give an introduction to the origins of the many-lights approach and its evolution over time. At the end of this chapter, the reader should be familiar with the techniques and terms that are used in the remainder of this thesis and understand their strengths and weaknesses that also form the basis for our own contributions.

3.2.1 Instant Radiosity

In 1997, Keller [23] proposed the idea of *Instant Radiosity* (IR). Most of the methods discussed throughout Chapter 3 can be classified as evolutions or variations of this original instant radiosity approach. It is an efficient technique to generate global illumination effects for diffuse and not-too-glossy materials. In contrast to the original Radiosity approach, which models the energy transport via form factors of surface patches, the basic idea of IR is to instead place a number of *Virtual Point Lights* (VPLs) throughout the scene which are then used to compute the overall illumination effects for every pixel on the screen. Figure 3.3 shows the basic principle of all Instant Radiosity-based methods.

Outline

Starting from the light sources, rays are cast into the scene. VPLs are placed at the intersection points of those rays with scene geometry. The VPLs are aligned along the intersecting surface's normal and emit a colored light according to the BRDF of the underlying material. The scene

is via a separate pass for each of the VPLs, where just like for each real light source, a shadow map is created for each virtual point light source.

Indirect Lighting

For the calculation of the indirect light contribution from a surface point two points x' (emitter) to another point x (receiver), usually a simple form factor is used:

$$F(x,x') = \frac{\cos\theta_x \cdot \cos\theta_{x'}}{\pi \cdot \|x-x'\|^2}. \tag{3.6}$$

The inverse squared distance term $1/\|x-x'\|^2$ causes a singularity problem that manifests as "light sparks" during rendering when the distance between points x and x' gets very small. This is however a very common problem for the approach because VPLs are commonly placed near scene geometry. A typical way to circumvent this problem is to either clip or re-modulate the form factor to the maximum representable value in the framebuffer.

VPL distribution

The raycasting process uses a pseudo-random, low-discrepancy sampling pattern known as *Halton sequences* [16] to determine the direction of each ray. Due to this pseudo-random sampling pattern, a quasi-Monte Carlo integration for the evaluation of the Rendering Equation (refer to Section 2.3) is achieved when the results of the separate shading processes of each single VPL are accumulated into a buffer and weighted by their contribution factor $1/N$.

Conclusions

SIMPLENESS: This simple and straight-forward approach allows for global illumination without the need of costly precomputation. It works directly in image space without requiring special data-structures or subdivision surfaces, which makes it perfectly suitable for drop-in usage in common, hardware-accelerated rendering pipelines.

RECURSIVENESS: Recursive application of this method (i.e., casting rays from the created VPLs) allows to easily account for multiple indirect light bounces within the scene.

SCALABILITY: Keller notes that his technique is mostly decoupled from scene complexity and depends mainly on the hardware rendering speed.

OVERMODULATION: The overmodulation of the geometry term poses a serious and common problem for this approach. It is often overcome by clamping the energy so that the result cannot grow without bounds. This however leads to the undesirable effect of energy loss during indirect illumination.

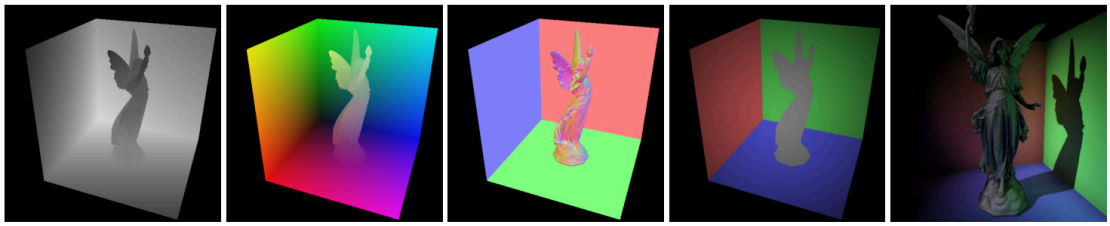


Figure 3.4: *The Reflective Shadow Map (RSM) is a texture-array consisting of depth, world-space position, normals and flux (f.l.t.r) buffers rendered from the point-of-view of the light source. On the far right, the final rendered image of the scene from the camera perspective is depicted. Image courtesy of Dachsbacher and Stamminger [9].*

3.2.2 Reflective Shadow Maps

This method by Dachsbacher and Stamminger [9] is another variant of a many-lights approach for the computation of one-bounce indirect illumination. In contrast to Instant Radiosity (and most other described methods in this section), where VPLs are placed via a quasi-random sampling pattern, the authors propose a different approach: The scene is rendered from the point of view (PoV) of the light source to generate an extended shadow map buffer that also stores normals and radiant flux. This Reflective Shadow Map (RSM) is then used to compute the indirect illumination from it by treating every pixel as a VPL. An example of such an RSM is given in Figure 3.4.

Outline

In a first step, the scene is rendered from the PoV of the light source, generating the RSM consisting of a depth map, and three additional buffers that store world-space position, normals and radiant flux. Afterwards, in a separate pass, the scene is rendered normally from the camera's viewpoint into a low resolution buffer. For each pixel x and corresponding normal n of this low resolution image, the indirect illumination is evaluated by looking up pixel lights x_p in the RSM according to a precomputed sampling pattern with about 400 individual samples. Between those illumination pairs, the irradiance E_p is calculated according to

$$E_p(x,n) = \Phi_p \cdot \frac{\max(0, \langle n_p | x - x_p \rangle) \cdot \max(0, \langle n | x_p - x \rangle)}{\|x - x_p\|^4}$$

and summed up for each individual normal-point pair to form the final indirect illumination at x.

Finally the full resolution image is rendered with per-pixel direct lighting and additively blended with the upsampled indirect illumination. The upsampling of the low resolution image is performed via adaptive bi-linear interpolation. For pixels where this interpolation is not possible, the full evaluation as done on the low resolution variant must be performed.

Interpolation

The adaptive screen-space interpolation scheme outlined above, is based on bi-linear interpolation. When upsampling the low-resolution buffer storing the indirect illumination to full resolution, the algorithm checks whether four surrounding low-res samples are similar to the corresponding high res pixel. Similarity is decided based on world-space position and normal difference. Interpolation in a weighted, bi-linear manner is performed only if at least three out of the four samples are similar.

Conclusions

G-BUFFER: The deferred rendering setup employed in this approach is used throughout many modern graphics engines. Many of the buffers used during calculation are either already easily accessible by the programmer or can be added without much effort, which allows an easy integration into existing rendering pipelines.
Although the authors described a completely novel indirect illumination process in their work, the most notable contribution is their proposed G-Buffer setup. Nowadays, most authors refer to this G-Buffer setup when they talk about RSMs.

INTERPOLATION: The interpolation method described by the authors depends heavily on scene and depth complexity. It works well for smooth surfaces but fails on complex objects, at which point the costly evaluation has to be performed in full for large parts of the scene. This has the undesirable side effect that the computation time can fluctuate heavily.

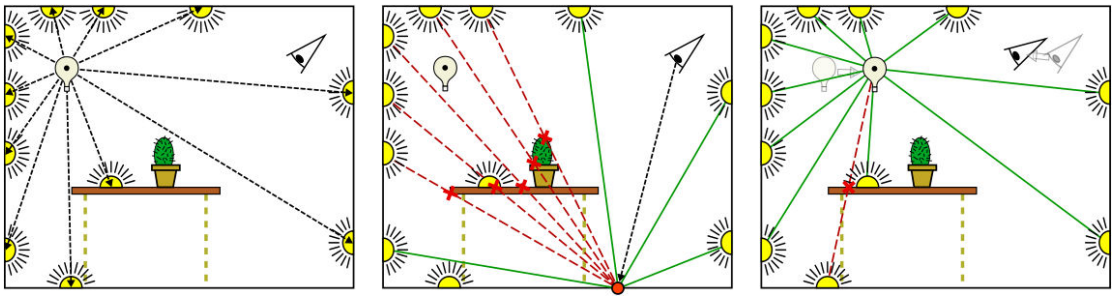
3.2.3 Incremental Instant Radiosity

While Keller's original Instant Radiosity approach is extremely simple and scales with available compute power, it is still a rather brute-forcing method. Hence, especially for use in real-time applications, optimizations would have to be applied to reduce the computational effort in order to deliver high enough frame rates for interactivity. To that end, Laine et al. [25] developed a method to cache and reuse VPLs over successive frames, called *Incremental Instant Radiosity* (see Figure 3.5).

Outline

Since the movement of light sources through a scene is mostly continuous and temporally coherent, the authors came to the conclusion that many of the calculations done in the previous frame (i.e., the placements of the VPLs) might still be valid for the current frame as well. In principle, only if the position of an existing VPL cannot be seen from the light source's new position is a reinitialization of that VPL necessary. Due to this behavior, most of the VPLs can be reused between frames.

However, in order to also maintain a near ideal distribution of the VPLs, they would sometimes also have to be removed and reseeded at new positions. This is a side effect that occurs due to the movement of the light source, which skews the VPL distribution away from the ideal and thus has to be handled accordingly. To guarantee real-time frame rates, an application-defined



(a) (b) (c)

Figure 3.5: *Similar to Instant Radiosity, VPLs are placed on intersection points between scene geometry and light rays (a). In (b), indirect illumination for a shaded point (red) is computed. Whenever the main light source or the camera moves, the existing VPLs that are still valid are evaluated. Invalid VPLs are discarded and new VPLs are created instead. Image courtesy of Laine et al. [25].*

upper boundary for the number of new VPLs in each frame is defined. On the other hand, the minimum number of new VPLs must be a nonzero value to ensure that even if all VPLs stay valid, the quality of the VPL distribution is maintained.

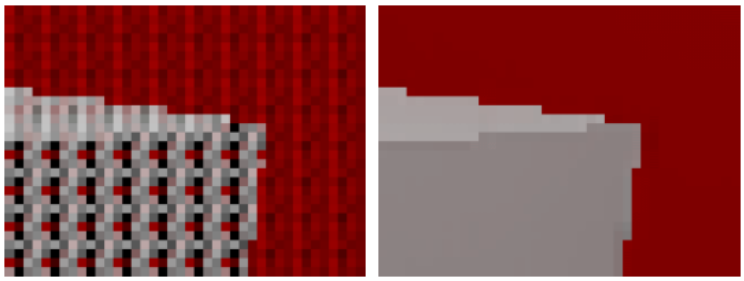
One downside of their approach is that dynamic objects are omitted from the VPL validity check due to the usage of a CPU ray-tracing method. Thus, indirect illumination can only bounce off static scene geometry. On the other hand, dynamic objects are still allowed to receive indirect illumination alleviating the aforementioned limitation somewhat.

To account for shadows from indirect illumination, paraboloid shadow maps [6] are created for VPL. Calculation of the indirect illumination is then performed on a tiled G-Buffer, where each tile is lit by a different subset of all VPLs. This allows to the cost of shading and additive blending significantly. In the end, the tiles are recombined into a full-sized picture, exhibiting a mosaic pattern (see Figure 3.5(a)) that stems from the fact that each neighboring pixel has been lit using a different set of VPLs. To eliminate this pattern, a geometry-aware box filter of the same size as the number of tiles in the G-Buffer is applied.

Tiled G-Buffer Illumination

As has already been shown in the work by Dachsbacher and Stamminger (described in Section 3.2.2), it is helpful to reduce the amount of pixels that have to be shaded in order to minimize the cost of performing the indirect illumination. While the approach by Dachsbacher and Stamminger performs the indirect illumination calculations on a down-sampled picture and then applies interpolation on the original picture, the Incremental Instant Radiosity offers a slightly different solution:

1. The G-Buffer is split into $n \times m$ tiles, where each tile represents an interleaved set of pixels of the initial G-Buffer. The pixel (x, y) in the tile (i, j) is thereby computed from the initial



(a) (b)

Figure 3.6: *(a) After shading on the tiled G-Buffer is completed and the resulting sub-images are merged back together, a structured noise pattern emerges. (b) By employing a geometry-aware box blur, the noise pattern can be largely eliminated. Image courtesy of Laine et al. [25].*

G-Buffer as $(xn+i, ym+j)$, where $0 \leq i < n$ and $0 \leq j < m$. This is a variant of interleaved sampling as described by Keller and Heidrich [24].

2. In the indirect illumination step, each tile is lit independently by a different subset of VPLs taken from the entire set. This facilitates the low-resolution accumulation of indirect illumination and reduces the workload on the graphics hardware, where accumulation/blending operations are still a huge bottleneck.

3. After illumination on the tiles is completed, the splitting process is reversed and the various tiles are merged back into a singular, full-size image.

Geometry-aware Box Blur

Since the merged indirect illumination buffer exhibits a structured noise pattern due to interleaved illumination, a box filter of size $n \times m$, which equals the number of tiles in the split G-Buffer, must be applied. This has the desirable effect of completely removing the structured noise from the surfaces. For each pixel, the geometric distance and the normal difference between the processed pixel and a kernel sample is computed. Two tresholds, α for geometric differences and β for normal differences are used to decide whether the pixels are similar enough to be considered in the filtering. This way, it can be guaranteed that the filter will not blur over borders that result from depth discontinuities in the scene or on sharp edges (see Figure 3.6).

Conclusions

The idea of performing interleaved sampling & filtering, is not new and has been used before [24]. In order to speed up the processing to meet real-time requirements the combination of methods is also used in our work.

VPL CACHE: The VPL management is an effective way to reduce computation costs by reusing data from the previous frame. However, since those computations are done entirely on the

CPU, a costly roundtrip between system- and graphics-memory has to be done in each frame in order to keep both data pools up to date.

INDIRECT ILLUMINATION: Only a single bounce of indirect lighting is performed by this implementation, but as Tabellion and Lamorlette [44] have demonstrated, this is sufficient even for high-quality offline rendering. Furthermore, although dynamic objects are still lit by indirect illumination, they do not contribute to it themselves, which is a drawback of this method.

G-BUFFER TILING: Generating a tiled G-Buffer and performing the indirect illumination using VPL subsets on each tile is another important optimization to achieve real-time frame rates. This way, the costly accumulation process can be split up into as many subitems as needed, relieving the graphics hardware's memory subsystem and in turn providing more computational resources for shading..

3.2.4 Imperfect Shadow Maps

The most influential contribution for this thesis is given by the work of Ritschel et al. [39, 41]. They introduce a method that uses a point-based representation of the scene that allows them to re-render low-quality shadow maps for hundreds of VPLs in each frame. Their point-based nature, however leads to leaks and holes in the resulting depth maps, which is why they are called *Imperfect Shadow Maps* (ISMs).

Outline

Whereas all previously described methods rely on standard, multi-pass (i.e., one pass per VPL) shadow mapping solutions [51] for the inclusion of shadows from virtual indirect light sources, this method uses point samples of the scene geometry. In contrast to polygons, each point sample can be randomly distributed to one out of hundreds of point-based shadow maps in a single pass. Thus, using a sufficient amount of point samples an approximate scene representation can be generated for a large amount of imperfect shadow maps. Furthermore, all ISMs are included in a single large texture atlas which also speeds up the rendering significantly.

Therefore, this method is especially advantageous if a suitable point sampling of the scene is already available, as is the case in point-based rendering [35]. In mesh-based rendering however an intermediary step is required to create the point representation from the mesh polygons. The original approach of the authors is to approximate the 3D scene by a set of points with roughly uniform density, which they calculate in a preprocessing step. By assigning each point on the surface to its corresponding triangle, this method also supports dynamic scenes without the need to recompute the point representation. Only fully dynamic objects of varying mesh topology, such as implicit surfaces or blobby objects where mesh-transformations cannot be simply mapped to the point samples, pose a limitation to the approach. However, modern implementations [4] make use of the tessellation hardware in current GPUs, which allows to generate the point representation on-the-fly.

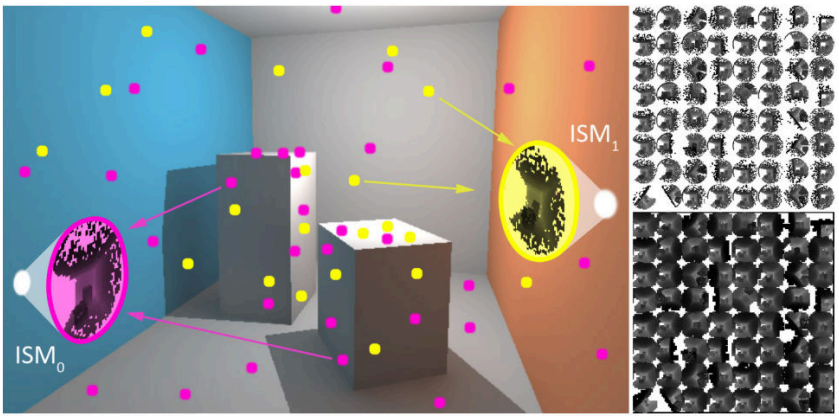


Figure 3.7: *Generation of Imperfect Shadow Maps: For two exemplarily selected VPLs (magenta and yellow), their generated ISMs are shown. The small, yellow and magenta dots represent some of the possible sample positions of the scene that are used to splat into the respective ISMs. On the right hand side, the resulting texture-atlas consisting of numerable ISMs is shown (top: before pull/push, bottom: after pull/push small holes in the depth map filled). Image courtesy of Ritschel et al. [39].*

For the gathering of indirect illumination, the same method as described above (refer to Section 3.2.3) is used, where only a subset of the VPLs are evaluated at each pixel. The combination of both these methods, 1) low-quality point-based ISMs and 2) an efficient shading scheme allows the real-time computation of indirect illumination from hundreds of VPLs that even respect visibility.

Point Generation

Points are approximated from a polygonal mesh representation with roughly uniform density. For this purpose, a random triangle is selected, where the selection probability is proportional to the triangle's area. Then, a random point on the chosen triangle is picked and its barycentric coordinates are stored alongside the triangle's index. This way, translations, rotations and scaling operations can be applied just like on the original geometry. Thus, common forms of animation such as hierarchical or skeletal animations, and even cloth deformation can be applied, while more dynamic effects such as dynamic destruction and implicit object surfaces cannot be presampled.

To allow such effects, the point representation of the affected geometry would have to be computed on-the-fly. At the time of the original work by Ritschel et al. in 2008, graphics hardware was not capable of performing this at runtime. More recently however, tessellation shaders were introduced to the programmable graphics pipeline, which were tailormade for this task. In 2013 Barák et al. [4] therefore introduced a method that performs the point generation entirely on the graphics hardware.

ISM Processing

All ISMs are stored within a single large texture atlas where each ISM receives a random subset of the sample points. ISMs are rendered from the point-of-view of the corresponding VPL where the view vector equates to the surface's normal. Since the goal is to illuminate an entire hemisphere of objects, a paraboloid mapping [6] is employed.

A single ISM for each VPL is created by splatting[2] the preprocessed sample points into the depth buffer. The size of a point splat is determined by its distance from the corresponding VPL. Since the splatting process will most likely leave holes and gaps within the shadow map, a pull-push approach [30] is applied in order to enhance the quality of the depth maps. To that end, an image pyramid is created first. In the *pull-phase*, each level of the pyramid downsamples the image by a factor of two and valid pixels are averaged. In the *push-phase*, the holes are filled iteratively using interpolation, beginning from the coarsest level down to the finest level. The resulting ISM atlas can be seen in Figure 3.7.

During the indirect illumination pass, the depth of a ray direction is queried from the ISMs and used to determine the visibility term of the RE. While a low amount of VPLs/ISMs results in artifacts that stem from the low-quality nature of the ISMs, a much smoother result can be achieved through blending the visibility information from a large amount of ISMs, which is why a texture-atlas should usually consist of several hundreds of VPLs.

Conclusions

VPL SHADOW MAPS: Although techniques, like *instanced rendering* (also called *instancing*) offer a solution to the problem of multi-pass shadow mapping, these methods are still troublesome when used with large amounts of instances and complex geometry. For shadow-mapping a scene, the entire scene geometry would have to be instanced possibly hundreds of times, and various CPU-based culling optimizations that are usually dependent on the point of view of the camera thus cannot be utilized, since the computed culling is likely to vary for each VPL. The usage of ISMs alleviates these drawbacks since the point splats for all VPLs can be rendered in a single pass. On modern graphics hardware, even the precomputation of the point representation is not necessary and can instead be computed on-the-fly using tessellation shaders.

SHADOW QUALITY: A drawback of the ISM method is that due to the low-quality nature of the depth map, additional steps have to be taken to yield a sufficiently smooth shadow term. A single ISM would produce noticeable stair stepping artifacts and would require a disproportionate amount of point samples to achieve a quality that is comparable to a classic shadow mapping for polygonal meshes. However, since indirect lighting effects are usually smooth and comprised of low-frequency changes, these artifacts can typically be hidden by a combination of the following methods: a) Blending a large amount of ISMs, b) increasing the number of point samples per ISM, c) performing a pull-push operation

[2] "Splatting" can be intuitively imagined by throwing a paintball onto a wall (in our case, the *image plane*) where it will burst upon impact, thus leaving a colored marking.

on the ISMs to fill holes, d) using a more sophisticated shadow map sampling technique (like PCSS [14], described in Section 3.1.1).

OTHER APPLICATIONS: The authors show that given enough VPLs and corresponding ISMs even specular illumination effects, such as caustics can be recreated to a certain degree. Also, this method can be easily expanded to include multi-bounce indirect illumination, by generalizing the idea of RSMs and ISMs – in the same way that classic shadow maps can be generalized to reflective shadow maps – to IRSMs (Imperfect Reflective Shadow Maps).

3.2.5 Clustered Visibility

One of the first relevant clustering techniques for global illumination, was introduced by Dong et al. [11] in their 2009 paper *Real-Time Indirect Illumination with Clustered Visibility*. Similar to the ISM method described previously, this method again tries to speed up the costly shadow mapping process for the VPLs. However, while ISMs are generated in one single pass for all VPLs, the method by Dong et al. relies on classic shadow maps that are created for clusters of VPLs and then sampled using a special soft shadowing technique. Instancing techniques are applied to speed up the multi-pass rendering process required for classic shadow mapping.

Outline

In this method, VPLs are again generated and distributed using quasi-random number generators. VPLs with similar orientation and position are then grouped together to form clusters using an algorithm inspired by *k*-means clustering. These groups of VPLs are then used to form *virtual area lights* (VALs), for which a sophisticated shadowing method – called CSSM (*Convolution Soft Shadow Maps*) [2] – is used. This significantly reduces the amount of required shadow maps while maintaining the smooth shadow penumbras that are common for indirect illumination. Lighting is still performed on a per-VPL basis, only the shadow casting process utilizes the VAL/cluster information. A graphical overview of this method can be seen in Figure 3.8.

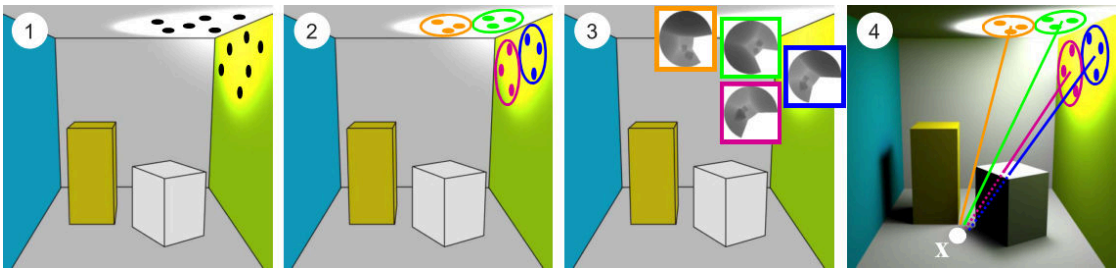


Figure 3.8: *Overview of the algorithm: (1) VPLs (black dots) are initialized in the first step. (2) Similar VPLs are clustered into virtual area lights. (3) Soft shadow maps are created for each VAL cluster. (4) The indirect illumination process uses the VPLs for illumination and the VAL shadow maps for soft shadow generation. Image courtesy of Dong et al. [11].*

k-Means Clustering

Clustering of the VPLs is based on the similarity between the VPL properties (alignment and position) and the cluster centers. The clustering consists of two steps:

1. Starting from arbitrary cluster centers, each point is assigned to its nearest cluster using the distance metric

$$\mu(c,\vec{x}) = \mathring{w}_{\vec{x}}\Delta_{\vec{x}} + \mathring{w}_{\alpha}\Delta_{\alpha} \tag{3.7}$$

   between the point $\vec{x}$ and the cluster center c. $\Delta_{\vec{x}}$ and Δ_{α} denote the euclidean distance and the angle between the point's normal and the cluster normal respectively. The influence of both parameters on the result can be weighed independently by $\mathring{w}_{\vec{x}}$ and $\mathring{w}_{\alpha}$.

2. The cluster centers are then updated by averaging all point positions that were previously assigned to the clusters.

This process is repeated until convergence is achieved. In the end, a mapping of VPLs to VALs is performed.

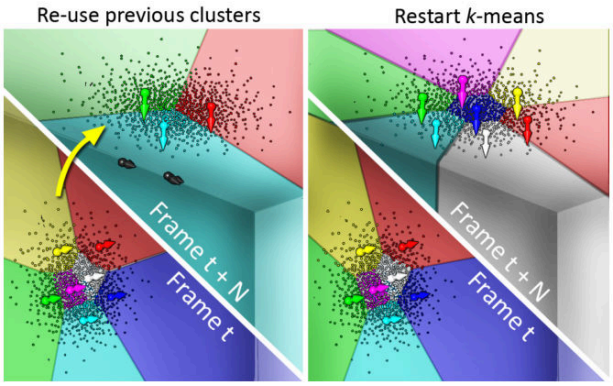


Figure 3.9: *In this scene, a spotlight is moving from the wall (lower half) to the ceiling (upper half) over the course of a few frames. On the left side, clustering information from the previous frame is reused, causing the number of clusters to decrease when the spot moves to the ceiling. On the right side, each frame the k-Means is restarted from the same initial cluster assignment. Thus, the number of clusters stays the same. Image courtesy of Dong et al. [11].*

Furthermore, the authors use a simple, but effective way to achieve a temporally coherent illumination: Instead of reusing information from the previous frame, the clustering process starts from an identical, initial cluster assignment in each frame. This way they do not have to deal with vanishing clusters and varying cluster sizes in dynamic scenes. See Figure 3.9 for a comparison between reusing information from the previous frame and restarting the k-Means algorithm in each frame. This problem also plays a crucial role in our own work, which will be discussed in greater detail in Chapter 4.

Convolution Soft Shadow Maps

Dong et al. [11] use *Convolution Soft Shadow Maps* (CSSMs) [2] to generate soft shadows with variable penumbra sizes from high quality shadow maps constructed per VAL cluster. CSSM is an extension to CSM [1] (Section 3.1.2) that also incorporates ideas from Percentage-Closer Soft Shadows [14] (Section 3.1.1). It works by generalizing the concept of CSMs on the process of estimating an average depth value $d_{blocker}$, as required in Equation (3.2). In PCSS, $d_{blocker}$ is estimated using an expensive "blocker search" heuristic that samples the shadow map region. CSSM instead applies a similar convolution method as for the normal shadow test to estimate $d_{blocker}$. Since only blockers with a depth value smaller than $d(\vec{x})$ should be averaged, the standard depth test is simply inverted. To that end, new basis images $[\bar{B}_i(z(p))z(p)]$ are computed alongside the regular CSM basis images for use in the inverted shadow test $\bar{f}(d,z) = 1 - f(d,z)$. An average blocker depth can then be efficiently estimated using pre-filtered basis images as

$$d_{blocker} \equiv z_{avg}(\vec{x}) = \frac{1}{1 - s_f(\vec{x})} \sum_{i=1}^{N} \bar{a}_i(d(\vec{x})) \left[w_{avg} * [\bar{B}_i(z) z] \right](p)$$

where w_{avg} is an averaging kernel. Refer to the work by Annen et al. [2] for a complete derivation of the equation.

Conclusions

CLUSTERING: The clustering of VPLs to VALs is used to reduce the number of shadow maps to one per VAL, but omits the obvious use of the same information to perform indirect lighting through a VAL approximation. The authors also found a simple way ensure a coherent clustering result in dynamic scenes by starting the clustering in each frame from the same initial cluster distribution. This allows them to trivially circumvent the problem of vanishing clusters that can occur when reusing clustering information from the previous frame.

CSSM: In contrast to the work by Ritschel et al. [39, 41], a far smaller amount of shadow maps has to be generated, while the rendering cost of each single shadow map is however much higher. Furthermore, the pre-convolved depth maps have the drawback that they are only an approximation of the original depth function. On the other hand, the possibility to use simple and fast filtering techniques allows for efficient calculation of soft shadows. Typical artifacts [2, 11] caused by the reconstruction of the depth function (ringing, curving, MIP discretization, etc.) are not as noticeable in the case of indirect illumination effects.

3.2.6 Reflective Shadow Map Clustering

In 2012, another clustering technique for indirect illumination systems was presented. Inspired by Dong et al.'s work, Prutkin et al. [36] also use a clustering algorithm inspired by k-Means clustering. Their work focuses solely on the clustering and reduction of the number of VPLs that are required to illuminate the scene. To this end, they accept a few concessions such as completely omitting the visibility term for indirect light sources and limiting their method to

Figure 3.10: *In this comparison, the Crytek Sponza scene is used to highlight differences between a reference picture rendered with* 256.000 *VPLs (far left) and the method presented by the authors using 960 disk-shaped area lights (2nd from left). The third picture shows a pixel-wise difference between the two methods and the last one gives a representation of the distribution of the VALs (depicted as hexagons) in the scene. Image courtesy of Prutkin et al. [36].*

single-bounce diffuse indirect illumination. See Figure 3.10 for results the authors achieved with their method.

Outline

The setup is similar to most recent methods, where an RSM is generated from the light source's point-of-view. Afterwards, the clustering process is initialized on the RSM using bi-directionally importance-sampled seed positions [41]. Bi-directionality is ensured by accounting for visibility from the main camera such that surface areas that contribute to the main view gain higher importance. There are three main differences to the clustering method described in the previous section:

1. The clustering has been enhanced by also incorporating the flux into the distance metric. Reusing the notation of Equation (3.7) we get

$$\mu(c,\vec{x}) = \mathring{w}_{\vec{x}}\Delta_{\vec{x}} + \mathring{w}_{\alpha}\Delta_{\alpha} + \mathring{w}_{\Phi}\Delta_{\Phi} \tag{3.8}$$

   where Δ_{Φ} is the squared vector distance $\|c_{\Phi} - \Phi\|^2$ between the color values at the cluster center c_{Φ} and at the current position Φ. The influence of which is weighted by $\mathring{w}_{\Phi}$.

2. This method only performs a single k-Means iteration in each frame, instead of performing many iterations in a single frame until convergence. Thus, temporal coherence is achieved by reusing the clustering information from the previous frame. The phenomenon of vanishing clusters is counteracted by simply re-seeding vanished clusters in the next frame.

3. Instead of disregarding the clustering information during illumination, the clusters in this work are also used to perform VAL illumination using point-to-disk or point-to-polygon form factors.

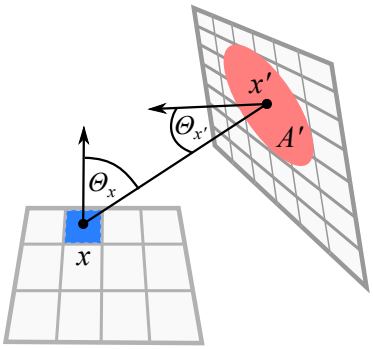


Figure 3.11: *Point-to-disk form factor for VAL illumination as suggested by Prutkin et al. [36].*

VAL Illumination

Because the clustering method is very similar to *Clustered Visibility*, our focus lies on the newly introduced idea of using the VALs for illumination purposes. These clusters provide an easy way to calculate an approximate area per cluster: a simple point-to-disk form factor [46] can be used to approximate disk-shaped area light sources. Using the same notation as for Equation (3.6), the form-factor can be rewritten as

$$F(x,x') = \int_{A'} \frac{G(x,z)}{\pi} \, \mathrm{d}A_z \approx \frac{\cos\theta_x \cdot \cos\theta_{x'}}{\pi \cdot \|x-x'\|^2 + A'} \tag{3.9}$$

where A' is the area of a cluster centered at x', θ_x is the angle between surface normal at x and the incoming ray direction from the cluster at x'. $\theta_{x'}$ is the angle between the surface normal at cluster center x' and the outgoing ray direction in direction of x (Figure 3.11). The similarity to Equation (3.6) is immediately visible. However, notice the change in the originally worrisome denominator, $1/(\|x-x'\|^2 + A')$, where the addition of A' prevents the denominator from growing without bounds at small distances between x and x'.

Conclusions

CLUSTERING: When compared to the variant used by Dong et al., employing a frame-by-frame iterative k-Means algorithm allows a more predictable and steady frame-time. As a drawback, certain issues such as the need for reseeding of individual clusters or maintaining a continuous, uniform cluster distribution have to be handled explicitly.

VAL ILLUMINATION: Using an already available clustering to perform area light illumination through the use of simple point-to-disk form factors is quite obvious and exhibits the following additional advantages: a) In comparison to using the original form factor, this variant provides an elegant solution to the singularity problem. b) In contrast to naive clamping, this solution is energy-conserving. c) The additional computational cost for using this form factor is negligible.

CHAPTER 4

Adaptively Clustered Reflective Shadow Maps

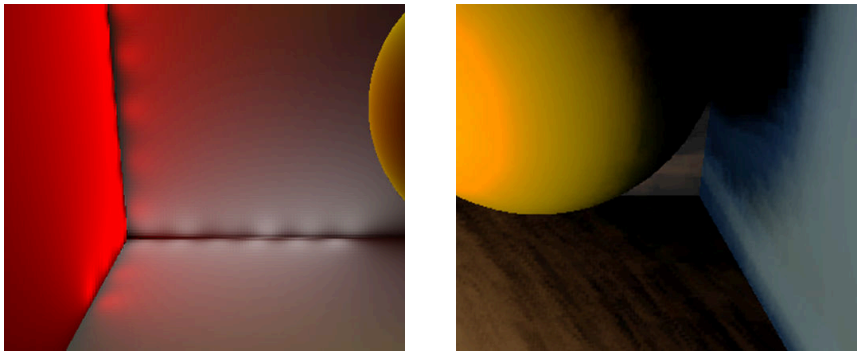
In this chapter we propose our new algorithm, *Adaptively Clustered Reflective Shadow Maps* (ACRSM) that is based on the concepts presented in the previous chapters. The focus lies on our novel *adaptive clustering* that aids in achieving temporally coherent indirect illumination at real-time frame rates for arbitrarily complex scenes.

Our algorithm is based on the concept of *Instant Radiosity* using *Reflective Shadow Maps* (RSMs). The idea is to simulate the interreflection of surfaces that receive light from a primary light source by using additional *virtual* light sources (VLs). We use the information stored in the RSM to generate VLs at positions where lights rays emitted from the primary light source intersect with scene geometry. In order to simulate indirect illumination, the virtual lights should then emit light colored according to the reflection characteristics of the underlying surface. To account for shadows from these VLs we employ *Imperfect Shadow Maps* (ISMs) in combination with a more sophisticated shadow sampling technique called *Percentage-closer Soft Shadows* (PCSS).

4.1 Motivation

While the techniques reviewed in Chapter 3 mainly describe methods to simulate global illumination, various problems arise in different areas. To some of them there already exist suitable solution approaches, others require new or improved methods to elevate the rendering quality and performance to an acceptable level:

§1 The use of point light sources [23, 25, 39, 41] can lead to point singularities, which occur when point lights end up at surface locations that are very close to receiving geometry they illuminate, as is typically the case in corners (see Figure 4.1(a)). This is often tackled by simply clamping the geometry term, which however also leads to energy loss.



(a) (b)

Figure 4.1: *(a) Illumination singularities appearing as "light sparks" that stem from a simple VPL form factor when the distance between light emitter and receiver is very small. (b) Example of reduced shadow quality when using an insufficient amount of ISMs.*

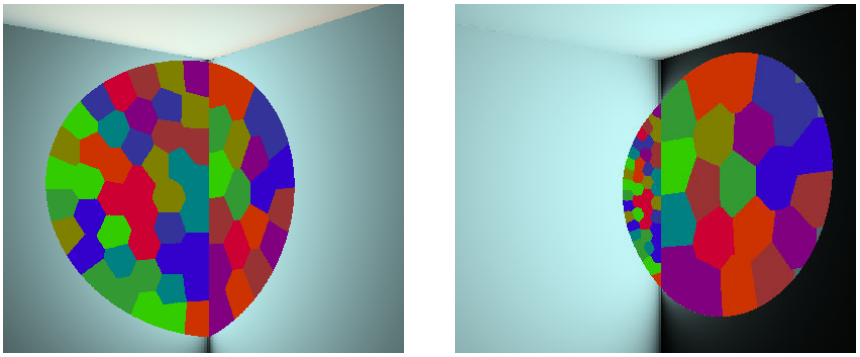
Prutkin et al. [36] suggest to cluster the RSM to disk-shaped *virtual area lights* (VALs) instead (outlined in Section 3.2.6) which are able to reduce these kinds of artifacts. This also comes with a considerable performance improvement because much less virtual light sources are required and the increase in computational effort is minimal.
Other forms of VALs for use in real-time applications such as Virtual Spherical Lights (VSLs) have been proposed by Hašan et al. [18], while Luksch et al. [28] generate Virtual Polygon Lights for use in GPU-aided light-map computations.

§2 The point lights are usually reseeded every frame using a pseudo-random initialization [23, 39, 41] that causes high frequency changes in the lighting, which can lead to flickering artifacts as well as erratic behavior in the resulting indirect shadows.
In order to reduce these artifacts, clustering approaches [11, 36] of similar fashion have been suggested (Sections 3.2.5 and 3.2.6) that facilitate temporal coherency.

§3 Clustering solutions [11, 36], however, introduce other problems in dynamic scenes that stem from the distance metrics μ as given in Equations (**3.7**) and (**3.8**):

a) As light sources move through the scene and pixels are assigned to clusters according to the distance metrics, it can happen that some clusters will not be assigned any pixels. Hence, the clusters *vanish* (Figure 3.9), which needs to be handled by the algorithm.

b) Furthermore, the distance metrics can prevent clusters from moving between objects. In Figure 4.2 a spot light is moving from one wall to another. The corner between the two walls poses a problem for the clustering due to the strong dissimilarity of their surface normals, which influences the distance metric μ. Hence, only a handful of clusters transition between the walls, while the majority of clusters are not transitioning, which therefore produces an *unbalanced cluster distribution*.



(a) (b)

Figure 4.2: *A spotlight transitioning smoothly from the position in (a) further to the right in (b) over the course of a few seconds. The clustering of the lit region is represented by the Voronoi-shaped, colored cells in each picture. Notice, how the cluster distribution becomes unbalanced in (b), because most clusters do not move over the* clustering-boundary *(sharp edge) between the two walls.*

Dong et al. [11] do not reuse clustering information from the previous frame, which allows them to easily prevent the vanishing of clusters as well as unbalanced cluster distributions. Prutkin et al. [36] simply reseed vanished clusters in the next frame but do not propose a solution to the problem of unbalanced cluster distributions.

§4 Achieving soft indirect shadows normally requires a large number of indirect light sources and corresponding low-quality shadow maps, which considerably increases the rendering cost for the indirect illumination. Therefore, simply reducing the amount of ISMs to improve performance leads to bad shadow quality (Figure 4.1(b)).
Dong et al. [11] suggest the use of Convolution Soft Shadow Maps [2] to gain higher quality shadows from a smaller bunch of clustered VPLs, improving the shadow quality while also reducing the computational cost. An alternative soft-shadowing technique is presented by Fernando [14] as discussed in Section 3.1.1.

§5 The low-resolution shadow maps used for indirect shadows by Ritschel et al. [39, 41] rely on a point representation of the scene, which would normally require an offline pre-process to sample the geometry. This, however, does not allow for fully dynamic scenes since the static, precomputed point samples cannot be easily remapped to.
An approach by Barák et al. [4] circumvents this by employing the tessellation hardware of modern GPUs to dynamically create point samples for the scene geometry at run-time.

4.2 Contribution

The methods discussed above all target individual aspects of the challenging task of performing high-quality indirect illumination in real-time rendering, but have so far never been combined in one unified, integrated system. In this thesis, we therefore describe a new algorithm that builds on the strengths of the methods above, combines them in a new way and additionally employs a number of novel ideas to tackle remaining problems outlined below:

- By using disk-shaped virtual area lights [46, 36] (see §1) it is possible to reduce the singularity artifacts that appear in indirect illumination with virtual point lights (VPLs), while circumventing energy loss. Furthermore, VALs offer the possibility to reduce the number of virtual light sources themselves, which in turn reduces shading costs and hence achieves higher performance (Section 3.2.6).

- A lower number of virtual light sources also results in a lower number of indirect shadow maps. This, however, means that simple point-based shadow maps like Imperfect Shadow Maps (ISMs) [39, 41] are not suitable, as their low-quality nature depends on a huge number of ISMs being superimposed to produce a smooth shadow result (see §4). To address this problem we use a more sophisticated shadow-map sampling that is based on *Percentage-closer Soft Shadows* [14] (Section 3.1.1).

- To avoid indeterministic illumination flickering artifacts due to incoherently changing virtual light source locations (§2), we employ a clustering method inspired by Dong et al. [11] and Prutkin et al. [36] that ensures a perceptually temporally coherent illumination. In line with findings by Dong et al. (Section 3.2.5), our experiments show that simply reusing the clustering information from the previous frame can lead to problems (§3) such as vanishing clusters (Figure 3.9) and/or an unbalanced cluster distribution (Figure 4.2). Our solution therefore expands upon this idea by introducing an *adaptive clustering* method that 1) re-seeds vanished clusters and 2) evaluates the clusters in each frame to decide whether a cluster needs to be repositioned in order to obtain a more balanced distribution (Section 4.8).

Combining all these approaches and contributions allows us to achieve a technique that is both, comparable in visual quality to existing methods and reduces the computation costs, therefore gaining higher performance in real-time applications.

4.3 Overview

Our algorithm (see figure on the right) works on the scene representation stored in *G-Buffers* generated from the point-of-view of the viewer (in other words, the camera G-Buffer) and the light source (i.e., the RSM). To efficiently generate VALs, we employ a clustering algorithm that works directly in the Reflective Shadow Map's image-space. Our method is therefore more similar to the work by Prutkin et al. [36], while other related clustering techniques [11, 28] work on VPLs.

In the beginning, cluster centers (or centroids) are positioned quasi-randomly (SEEDING & IMPORTANCE WARPING). Then, clusters are formed around the centroids by grouping pixel regions inside the RSM based on similarity (MAPPING). To that end, weighted distances between cluster centers and nearby pixels are computed based on Euclidean distance from the cluster center, similarity of surface-normal, -color, -reflectivity, etc. This is achieved by performing a single k-Means iteration in each frame (AVERAGING). An additional EVALUATION is performed with each iteration to verify the current cluster distribution and, if necessary, perform a re-seeding of a subset of clusters to re-balance the distribution. In the end, each cluster is defined by its center position and a radius, thus forming a disk-shaped virtual area light (VAL).

Low-quality imperfect shadow maps are generated for each VAL cluster from the scene geometry either in a pre-process or on-the-fly using tessellation. To improve the coverage of these point-based shadow maps, a pull/push process is performed to close smaller gaps.

In order to reduce shading costs, we perform VAL illumination including soft shadows generated from imperfect shadow maps via tile-based interleaved shading. The low-resolution tile-based indirect illumination is in the end filtered using a geometry-aware blur and ultimately combined with the direct light source's high-resolution illumination.

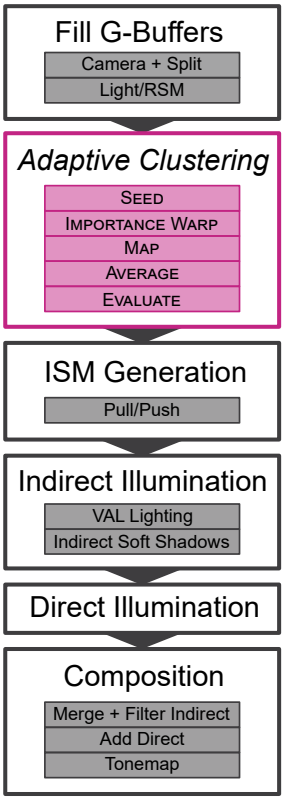


Figure 4.3: *Overview of the algorithm's pipeline. In this chapter we emphasize on our* adaptive clustering *as discussed below.*

Adaptive Clustering

As highlighted in Figure 4.3, we want to emphasize one of the main contributions of this work, the *Adaptive Clustering* approach described in depth in the next few sections. Our approach is inspired by k-Means clustering [29], which has been used in a similar fashion by both Dong et al. [11] (object-space VPL clustering) and Prutkin et al. [36] (image-space clustering). k-Means is a popular choice whenever spatial data has to be partitioned based on similarity characteristics. The goal of the algorithm is to create k partitions in such a manner that the sum of the squared distances between the clusters $C = \{C_1, C_2, \ldots, C_k\}$ and the d-dimensional input data $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$ (in other words, the *variance*) is minimized. This problem can be formalized

as

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} \|\vec{x} - c_i\|^2$$

where c_i is the mean of points in C_i. Visually, the result of the clustering represents a *Voronoi diagram* [3] generated by the means (Figure 4.2). In our case, each cluster C_i represents a virtual area light (VAL) and X are pixels inside the RSM. The properties *position*, *normal* and *flux* stored in the RSM make up the d-dimensional vector data that is used to minimize the distance. Following Lloyd's [27] definition, k-Means performs the following three steps:

1. Choose k random centroids c_i from the data set.
2. Assign each input vector $\vec{x}$ to the cluster that has the least increase in variance.
3. Update the current cluster centroids by averaging the data from the assigned vectors.

Steps 2-3 are repeated until convergence.

We adapted this procedure to better fit the requirements of our application and made changes with respect to

Centroid initialization: Instead of choosing random positions for the initial cluster distribution, we employ an importance-based seeding. This way we can skew the initial distribution into a more favorable state for the task of forming VAL clusters.

Time-criticalness: In order to be applicable in a real-time application with steady performance, we only apply a single iteration of steps 2 and 3 in each frame. Inherently, we are reusing the clustering information from the previous frame and thus, we usually achieve convergence over the course of a few frames.

Clustering optimization: Especially in dynamic scenes, reusing data from the previous frame can have its drawbacks. As depicted in Figure 4.2, a good initial cluster distribution might become suboptimal over time due to a moving light source or dynamic objects. In this case, we need to *adapt* our clustering dynamically in order to re-balance the cluster distribution. Since such a step is not present in standard k-Means variants, our adapted algorithm outlined below contains an additional EVALUATION step.

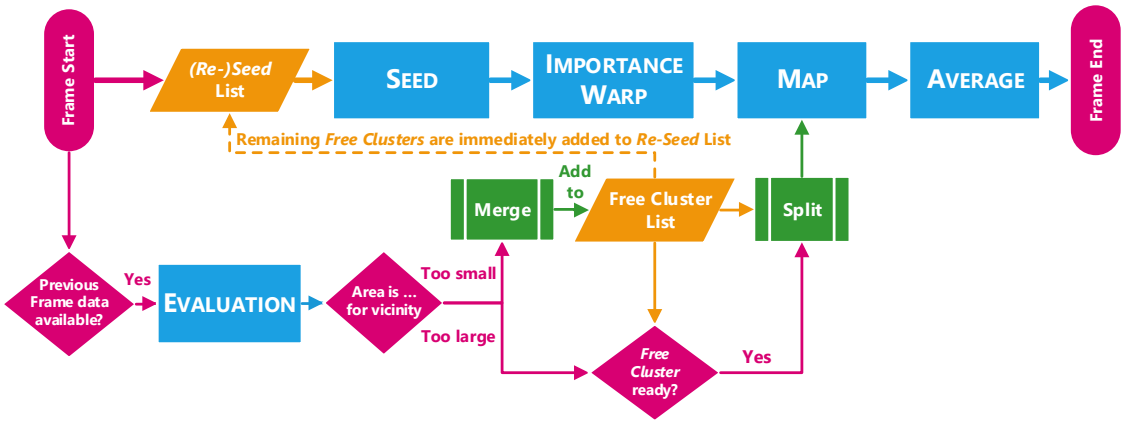


Figure 4.4: *Basic sequence of our clustering algorithm. Start- and End-Nodes are represented by round-border rectangles and also denote the beginning and end of a single frame. Decisions are given by the diamond-formed shapes. Rhombuses denote important data structures that feed the processes, which in turn are represented as simple rectangles. Inter-Frame flow is given by dashed-lines.*

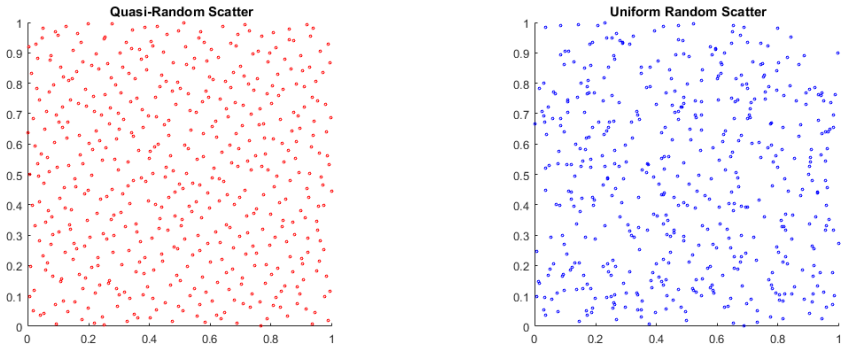
Hence, our final adaptive clustering procedure (Figure 4.4) consists of the following five steps:

SEEDING	Initializes the cluster centers at quasi-random positions (Section 4.4).
IMPORTANCE WARPING	Re-distributes the randomly positioned clusters according to the surfaces' reflection characteristics (Section 4.5).
MAPPING	Assigns individual pixels to the corresponding nearest cluster according to a specific distance metric (Section 4.6).
AVERAGING	Performs an adjustment of the cluster properties according to their currently mapped region (Section 4.7).
EVALUATION	Re-distributes the clusters to ensure a balanced cluster distribution through neighborhood evaluation (Section 4.8).

4.4 Seeding

The first step of the clustering process is the *seeding* of the cluster centers among the lit surfaces in the RSM. The *(Re-)Seed*-List stores the current set of clusters that need to be seeded and their initial seed positions. Usually, we generate these seed positions in 2D texture space using a quasi-uniform distribution given by a low-discrepancy Halton-sequence [16] (refer to Figure 4.5).

However, it is probable in dynamic scenes that in the following frames, the clusters are moving and changing, which leads to additional scenarios where elements are queued to the re-seed list:



(a) (b)

Figure 4.5: *500 random points generated with the low-discrepancy Halton-sequence (a) and with Monte-Carlo sampling (b). Images courtesy of The MathWorks, Inc.*

1. Some clusters will not be assigned any corresponding pixels of the RSM by the MAPPING procedure. These clusters *vanish* and have to be re-seeded, which is discussed in greater detail in Section 4.6.

2. Additionally, too large or too small clusters can be detected by the EVALUATION step. In this case, clusters are relocated to different positions utilizing the re-seed list. We discuss these scenarios in Section 4.8.

It is interesting to point out that the usage of a low-discrepancy sequence for initialization stems from related work [23, 25, 39] where VPL positions are re-seeded in each frame. This is required to guarantee a more uniform distribution without generating recurring patterns as these are known to produce sampling artifacts. Although our algorithm is less prone to exhibit these artifacts due to its progressive nature, a good initial distribution is still important in order to ensure a faster convergence.

4.5 Importance Warping

In general, instead of uniformly distributing the clusters among the lit surfaces, it is beneficial to bias the distribution according to an informed decision-making process. To that end, we assign a so-called *importance* value $\rho(\vec{x})$ to each point $\vec{x}$ of the RSM beforehand. This way, we can assign a higher importance to clusters that have more complex reflection characteristics, such as specular surfaces. Modifying the sampling density of clusters based on this importance value allows us to place more clusters in areas with reflection characteristics of higher frequency, and less in areas of low frequency (*importance sampling*). In turn, this results in a loose coupling between the area of a cluster and the importance of the underlying surface. We derive a two-tiered importance value, based on a

BRDF-DERIVED value ρ_f that is based on the observation that specular materials usually create more complex lighting effects than rougher materials. Thus, we base our importance

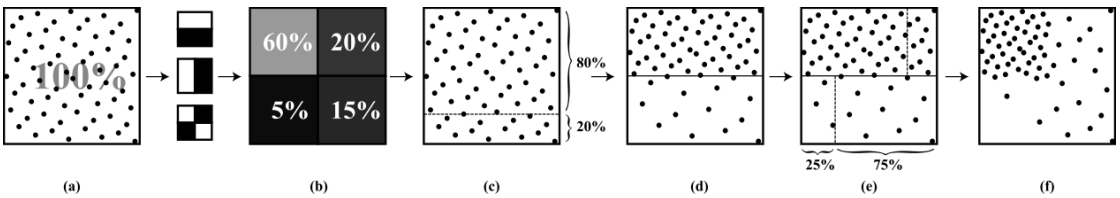


Figure 4.6: *Illustration of importance warping on a single hierarchy level. Image courtesy of Clarberg et al. [7].*

calculation on diffuse and specular surface reflectivity, deeming surfaces with higher specularity as more important than those surfaces with less specular reflectivity. In a more formalized way, we derive a normalized value in the range $[0,1]$ from a simplified BRDF-term $f_r(\vec{x})$ such that

$$\rho_f(\vec{x}) = f_d(\vec{x}) \cdot w_d + f_s(\vec{x}) \cdot (1.0 - w_d) \tag{4.1}$$

where $f_d \approx I_d$ is the diffuse reflection component and f_s is a combination of specular surface reflection properties. Both values are implementation specific and thus detailed in Chapter 5. Finally, w_d is a variable that allows to shift the importance contribution between the diffuse and specular components. Additionally, a

VIEW-DEPENDENT value ρ_v as proposed by Ritschel et al. [41] is employed, to further refine the importance. This is achieved by also incorporating additional information about the currently viewed region in camera space. To this end, we compute the influence of each lit pixel on the visible region by taking up to M random samples from this region and deriving an averaged influence based on Lambert's law. Hence, RSM regions with higher influence on the randomly sampled positions in view-space get a higher importance. We can formalize this as

$$\rho_v(\vec{x}) = \frac{\sum_{i=1}^{M} \left\langle \hat{\mathbf{n}}_i | \hat{\boldsymbol{l}}_i \right\rangle}{M} \tag{4.2}$$

where, $\hat{\mathbf{n}}_i$ is the surface normal at the sampled view region and $\hat{\boldsymbol{l}}_i$ is the normalized directional vector between the sampled position in the view region and the lit position at $\vec{x}$.

These two values are then combined to a final *bi-directional importance* [41] value:

$$\rho(\vec{x}) = (1.0 - w_v) \cdot \rho_f(\vec{x}) + w_v \cdot \rho_v(\vec{x}) \tag{4.3}$$

where w_v allows to weigh the contributions of the BRDF- and view-dependent terms. This value is then stored as part of the RSM to represent an *importance map*.

After the initial cluster positions have been seeded, we apply an *Importance Warping* [7] on these seed positions. The aim of this process is to *warp* (re-distribute) the cluster positions according to the importance map. This process is illustrated in Figure 4.6. It is a hierarchical

process employing an image pyramid[1] of the importance map. For each cluster center (black dots in the figure), we start from the coarsest level of the image pyramid of size 2×2. Each of the quadrants (pixels) of that sub-sampled image gives a measure of importance for its encompassing region (Figure 4.6b). The warping procedure then begins by first warping vertically, thus partitioning the image into two rows: a top row, consisting of the top-left and -right quadrants, as well as a bottom row, consisting of the bottom-left and -right quadrants respectively. The individual importance values are summed up for each row to get percentages in the top row p_t and in the bottom row p_b respectively. To perform vertical warping, we first classify each cluster's y coordinate if it lies within the top p_t percent of the row or the bottom p_b percent (Figure 4.6c). According to this classification, the y coordinates are then rescaled so that $p_t\%$ of the clusters are in the top row and $p_b\%$ in the bottom row (Figure 4.6d). Afterwards, horizontal warping is achieved in a similar fashion by scaling the cluster's x coordinate to fit within one of the two child regions of the respective row (Figure 4.6e). Finally in Figure 4.6f, we arrive at the warped position for the current cluster at the current level of the image pyramid. This process is repeated at the next finer level of the image pyramid within the 2×2-quadrant the cluster was scaled to and continues until a specific level of the mip-map has been reached. Usually, it is not required to continue this process up to the finest level of the image pyramid (i.e., the original image). Instead the process can be stopped after a few levels as the impact on the warped positions becomes smaller with each finer level. Pseudocode of this algorithm is given in Chapter 5, Algorithm 5.3.

4.6 Mapping

After the cluster centers are warped into position according to the importance values, the surrounding areas have to be *mapped* to their nearest cluster center (refer to Figure 4.7). This is a standard procedure of the k-Means algorithm. To that end, we define a square-region of the same size around each cluster center, which yields an irregular grid of clusters where the cluster areas can overlap. Each smallest addressable area inside such a square-region (in an image, usually a pixel) will be assigned a normalized distance to the respective cluster center according to the following distance-metric:

$$\mu(c_i, \vec{x}) = \Delta_{\vec{x}} \cdot \mathring{w}_{\vec{x}} + \Delta_{\hat{\mathbf{n}}} \cdot \mathring{w}_{\hat{\mathbf{n}}} + \Delta_{\rho} \cdot \mathring{w}_{\rho}. \tag{4.4}$$

Similar to Prutkin et al., we use an additional parameter besides the distance $\Delta_{\vec{x}}$ and the normal difference $\Delta_{\hat{\mathbf{n}}}$. However, instead of using the flux as in Equation (3.8) we employ the importance ρ as an additional parameter. This decision is based on the observation where the inclusion of the flux often leads to oscillation during mapping whenever finely structured albedos are present. The various distances Δ are given as follows:

$\Delta_{\vec{x}}$ A normalized Euclidean distance $\|\vec{x} - c_i\| / b_{max}$ between the cluster center c_i and the current pixel position $\vec{x}$. Normalization is performed relative to the diameter of the world-space bounding box b_{max}.

[1] In computer graphics, an image pyramid is created through a method called *mip-mapping*. Starting from a square base image of side-length 2^n, multiple sub-sampled, smaller images are created. Each smaller image is down-sampled by a factor of two, effectively halving the side-length with each coarser level in the hierarchy.

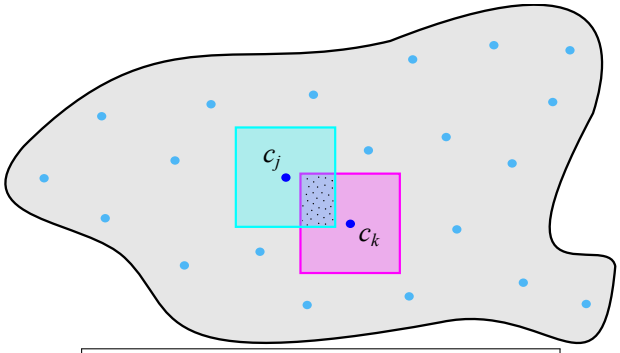


Figure 4.7: *Mapping of pixels to clusters for a given surface region (gray) with multiple cluster centers (blueish dots). Two randomly selected clusters c_j and c_k with their highlighted mapping regions (cyan and magenta) are candidate clusters for the intersecting area (dotted region in the middle). Each smallest quantifiable area $\vec{x}$ within the dotted region will be assigned to the cluster $C_{min}^{[j,k]}$ to which it has the smallest distance μ.*

$\Delta_{\hat{\mathbf{n}}}$ A distance derived from the angle between the normal $c_{\hat{\mathbf{n}}}$ at the corresponding cluster center and the normal $\hat{\mathbf{n}}$ at the current pixel, given by $1.0 - |\langle c_{\hat{\mathbf{n}}} | \hat{\mathbf{n}} \rangle|$.

Δ_ρ The absolute difference $\left| c_\rho - \rho(\vec{x}) \right|$, between the importance value stored at the cluster center c_ρ and at the current pixel $\rho(\vec{x})$.

Combined, they form a convex sum of distances, where each component is weighted by a corresponding weight $\mathring{w}$. RSM pixels $\vec{x}$ where the areas A of multiple clusters $C = \{C_1, C_2, \ldots C_k\}$ overlap, are assigned to the cluster C_{min} with the lowest overall distance μ, such that

$$C_{min} = \underset{C_i \in C | \vec{x} \in A_i}{\arg\min}\, \mu(c_i, \vec{x}) \tag{4.5}$$

where c_i is the cluster center of C_i. This definition does not guarantee that every cluster in C will have mapped pixels. Clusters without any mapped pixels cannot partake in the following steps of the algorithm. Thus, in order to prevent the permanent *vanishing* of unmapped clusters, they are added to the re-seed list to be reseeded in the next frame.

4.7 Averaging

After the mapping of addressable areas to clusters has been established, we have to adjust the cluster centers to reflect the new mapping. We generally call this *averaging*, because usually most cluster properties are calculated anew by averaging the properties from the entirety of all contributing pixels. In our case, most properties are updated according to their averages: cluster-position, -normal and -importance. The odd cases are:

- The flux of the cluster center, which instead of being averaged, is read normally from the newly updated cluster position. Thus, it always reflects the flux at the center of the cluster.
- The area of the cluster, which instead of being averaged, has to be recomputed to match the number of newly assigned pixels. Since we employ a point-to-disk form factor later during indirect illumination, we estimate a radius r_c for the cluster as the distance between the cluster center and the mapped pixel with the largest distance to it. Based on this radius, the area of a circle is given by $r_c^2\pi$.

4.8 Evaluation

Usually, in standard k-Means implementations, once the averaging step is done, the mapping process begins again and the result is refined over the course of a few iterations. In static scenes, the combination of pseudo-random cluster seeds with the following mapping and averaging steps quickly converges towards an optimal solution. However, in dynamic scenes with moving objects and light sources, these changing boundary conditions can lead to situations where the current cluster distribution becomes suboptimal. Instead of converging towards a global optimum, the current clustering keeps converging towards a disadvantageous local optimum. Refer again to Figure 4.2 where a moving light source causes an unbalanced cluster distribution.

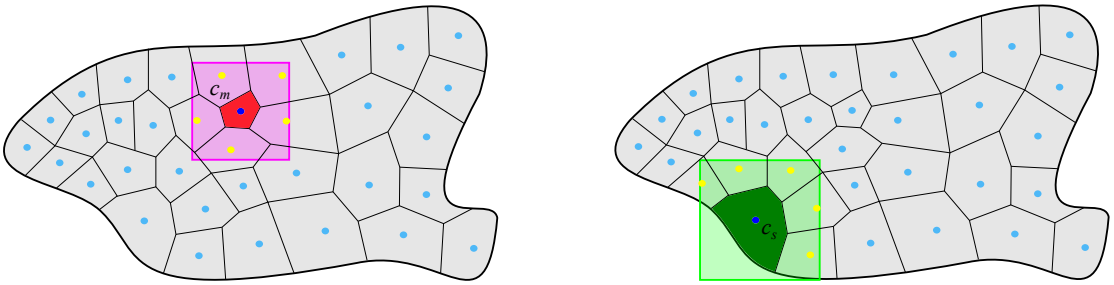
Following the concepts of optimization theory, we propose an EVALUATION step between the averaging step and the re-initialization/-seeding step of the next frame. Its aim is to reintroduce some guided randomness into the clustering algorithm to allow it to leave the unfavorable state of a local optimum. To that end, the EVALUATION heuristic performs a validation on a random subset of all clusters that determines whether a cluster is a) too *small* or b) too *big* in relation to the average size of the surrounding clusters. In the first case, the cluster c_m is *merged* by dropping it from the list of currently active clusters and added to a free cluster list L_f. In the latter case, a cluster c_s is *split* only if a free cluster is available in L_f. In this case, the free cluster c_m is reseeded at a random position within the cluster c_s to be split. Consequently, the importance-mapping process for c_m is circumvented in this scenario because its desired position has already been established. After the splitting operations have been performed, we check if there are any remaining free clusters in L_f. In order to ensure that the cluster budget of k active clusters is used up in each frame, those free clusters are injected into the re-seed list in order to be reseeded immediately.

Since area c_A and importance c_ρ of a cluster are loosely coupled through the importance mapping, using these two values to derive a metric for evaluation makes the most sense:

- A cluster with a low importance tends to have a large area and vice versa.
- Therefore, the comparison of the metrics between a small cluster with a high importance and a large cluster with low importance yields comparable results.

Thus, the comparison metric δ_c is derived from these properties as

$$\delta_c = \frac{c_A}{A_{max}} \cdot c_\rho \tag{4.6}$$



(a) Merging: $\delta_{c_m} - \bar{\delta} < -\varepsilon_m$ (b) Splitting: $\delta_{c_s} - \bar{\delta} > +\varepsilon_s$

Figure 4.8: *A theoretical scenario of how the evaluation step might perform for the given clustering. Clusters are represented by blue dots, their mapped regions are represented as Voronoi cells. The square-shaped regions around the selected clusters represent the area of the neighborhood-search. Yellow dots denote clusters found by the neighborhood-search and are used to compute* $\bar{\delta}$.

where A_{max} is the area of the biggest cluster. The basic principle of the EVALUATION step is depicted in Figure 4.8, which is similar to the one of the MAPPING procedure: Starting from a cluster center c, we look at a surrounding area of at least twice the size of the cluster to find neighboring clusters $C_n \subset C$. Subsequently, a weighted arithmetic mean $\bar{\delta}$ over the comparison metrics δ_{c_i} of the neighboring clusters $c_i \in C_n$ is computed as

$$\bar{\delta} = \frac{\sum(\delta_{c_i} \cdot \delta_{c_i})}{\sum \delta_{c_i}}, \tag{4.7}$$

where the data is simply weighted by itself. By way of their definition, equations (4.6) and (4.7) always return values in the range $[0,1]$. Thus, the evaluation heuristic is performed by comparing the values δ_c of the current cluster (refer to c_m and c_s in Figure 4.8) with $\bar{\delta}$ of the neighborhood C_n, which can result in three possible outcomes:

$$\delta_c - \bar{\delta} \begin{cases} < -\varepsilon_m & \text{merge,} \\ > +\varepsilon_s & \text{split,} \\ \text{otherwise} & \text{do nothing.} \end{cases} \tag{4.8}$$

Here, the thresholds ε_m and ε_s are *merge*- and *split*-thresholds that decide when a splitting or merging process occurs. A practical range for ε_m is around 0.35–0.5. This can be interpreted as *"if a cluster is* ε_m*% smaller than the sampled neighborhood it is merged"*. In turn, since our splitting process relies on free clusters that must be generated by the merge-process right beforehand, we usually have a lower split-threshold that lies around 0.1–0.2. Remaining free clusters generated by the merge process are immediately added to the re-seed pipeline and will be processed in the current frame as the algorithm continues with SEEDING.

CHAPTER 5

Implementation

In this chapter, we give an in-depth view on how the algorithm can be implemented into an existing deferred rendering pipeline. We use `OpenGL 4` in conjunction with `GLSL` shaders to implement this work as a plugin for a `C++`-based rendering framework. Please note that our description might therefore use terminology specific to those programming languages and APIs. From a technical point of view, our implementation consists of various pre-processing steps, a diverse range of buffers (textures, arrays, etc.) to hold our data, a large number of shaders to either create or visualize the results, and surrounding framework-level code to put it all together. All of this is then combined in our *ACRSM* algorithm, which achieves fast and efficient diffuse single-bounce global illumination. In the following sections, we outline the most important aspects of the implementation with emphasis on technical details to make our method reproducible.

5.1 Buffers & Data Structures

In the next few sections we outline the basic setup of the most important buffers in our implementation, from G-Buffers to cluster-data.

5.1.1 G-Buffers

G-Buffers are used in *deferred shading* setups to store scene information projected to screen space when the actual shading is intended to happen not at rasterization time, but in a *deferred* post-processing pass. These setups are widely used in recent rendering pipelines and became very popular due their capability to decouple geometric complexity from lighting computations. In order to compute lighting in common *forward shading* renderers, each object O has to be rendered again for each light source L that it affects, resulting in $\mathcal{O}(O*L)$ complexity. A deferred renderer, on the other hand, renders the geometry once and stores information that is relevant during illumination such as position, albedo and normals in textures. During illumination, these

textures are then sampled to perform deferred lighting computations for each light in a separate pass, yielding a much more lenient complexity of $\mathcal{O}(O+L)$. This decoupling is especially beneficial when dealing with large amounts of light sources, as we do in our approach.

There are different ways to set up a G-Buffer layout, which mostly depends on the needs of the application and the information they store. In our approach, we use a *slim* G-Buffer setup, which relies on 32bit-wide textures. On the one hand, this reduces the amount of expensive texture-memory and -bandwidth, but on the other hand, this tends to be more computationally expensive, because values like normals and positions have to be stored in a packed way to fit inside such limited storage.

	32bit			
	8bit	**8bit**	**8bit**	**8bit**
DIFFUSE k_d	R	G	B	–
SPECULAR k_s	R	G	B	Power
NORMAL $\hat{\mathbf{n}}$	X		Y	
DEPTH $\tilde{z}$	Hardware-Depth			

(a) *Camera: Variable resolution.*

	32bit			
	8bit	**8bit**	**8bit**	**8bit**
REFL. FLUX Φ	R	G	B	–
NORMAL $\hat{\mathbf{n}}$	X		Y	
IMPORTANCE ρ	Bi-dir. Importance Map			
DEPTH $\tilde{z}$	Hardware-Depth			

(b) *Light/RSM:* 512×512 *pixels.*

Table 5.1: *Slim G-Buffer layouts as used in our implementation.*

We use two G-Buffers, one from the point-of-view of the camera that has a variable resolution of $width \times height$ plus an additional tiled variant that is split according to our tiled mesh geometry (Section 5.1.4). This *Split G-Buffer* is used during indirect illumination while the original is employed for direct illumination. We discuss both cases at the beginning of Section 5.3.3. The other G-Buffer, our RSM, is rendered from the PoV of the light source with a static resolution of 512×512 pixels. This is sufficient since we use a spot light source for illumination. Both G-Buffers use similar layouts as can be seen in Table 5.1. For convenience, we use the Light-G-Buffer's attached depth buffer directly for shadow-mapping as well. Because of the slim layout, the following values have to converted, packed and unpacked when writing and reading them, respectively:

NORMAL $\hat{\mathbf{n}}$: Since 8-bit floats are not sufficient for storing the three normal components, we opted to store the normals in view-space using spherical mapping [31]. To that end, during G-Buffer creation, only the x, y components of the compacted normal are stored as

$$\hat{\mathbf{n}}'_{xy} = \texttt{normalize}\,(\hat{\mathbf{n}}_{xy}) \cdot \sqrt{\hat{\mathbf{n}}_z \cdot 0.5 + 0.5}.$$

When reading from the G-Buffer, we restore the normal components $\hat{\mathbf{n}}_z$ and $\hat{\mathbf{n}}_{xy}$ separately as

$$\hat{\mathbf{n}}_z = -(\langle \hat{\mathbf{n}}'_{xy} | \hat{\mathbf{n}}'_{xy} \rangle \cdot 2.0 - 1.0)$$

$$\hat{\mathbf{n}}_{xy} = \texttt{normalize}(\hat{\mathbf{n}}'_{xy}) \cdot \sqrt{1.0 - \hat{\mathbf{n}}_z \cdot \hat{\mathbf{n}}_z}$$

DEPTH $\tilde{z}$: We use a hardware depth buffer, which is computed using the standard model-view-projection transformation in the vertex shader through setting `gl_Position`. Restoring

the view-space position, on the other hand, is a bit more involved than when using a linear depth-buffer. By drawing a full-screen quad with coordinates given in projection-space $[-1,1]$ and multiplying with the inverted projection-matrix we end up with view-space x,y coordinates per pixel. To further reconstruct the view-space z-component from $\tilde{z}$, we use

$$z = \frac{z_{near} \cdot z_{far}}{\tilde{z} \cdot (z_{far} - z_{near}) - z_{far}}$$

where z_{near}, z_{far} are the near- and far-clipping plane values in view-space.

REFLECTED FLUX Φ: As specified in the beginning, the flux defines the radiant power. Depending on the type of light source, there are different ways to compute it. For a uniform directional light, this is a constant value. For a uniform spotlight, which is the type of light we use, defined through its direction $\hat{\boldsymbol{l}}_s$ and a half-opening angle ω_s, the flux decreases with the cosine $\cos\theta$ to the spotlight's direction. The *reflected* flux incorporates the material reflection coefficient k and is computed as

$$\Phi = k \cdot \frac{\cos\theta}{\cos\omega_s}$$

where θ defines the angle between $\hat{\boldsymbol{l}}_s$ and the position vector $\vec{x}$ of the point to shade as depicted in Figure 5.1.

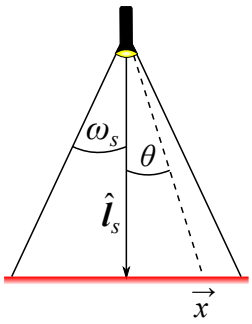


Figure 5.1: *Spotlight geometry. The computed flux at $\vec{x}$ decreases with the cosine ($\cos\theta$) to the spotlight's direction $\hat{\boldsymbol{l}}_s$ and vanishes completely at $\cos\omega_s$.*

IMPORTANCE ρ: We have already defined ρ in Equation (4.3), which is based on a BRDF-component ρ_f (Eq. (4.1)) and a view-dependent component ρ_v (Eq. (4.2)). In our implementation we define the parameters f_d and f_s of the BRDF-dependent part as

$$f_d(\vec{x}) = \max_{rgb}\left(k_d \cdot l \cdot \frac{\cos\theta}{\cos\omega_s}\right) \tag{5.1}$$

$$f_s(\vec{x}) = \max_{rgb}\left(k_s \cdot l \cdot \frac{\cos\theta}{\cos\omega_s}\right) \cdot \alpha \tag{5.2}$$

where the $\max_{rgb}$-function returns the maximum of the three RGB values, l defines the RGB light color, $\cos\theta$ defines the spotlight falloff as above, k_d and k_s define the diffuse- and specular-reflection coefficients, respectively. and α is the specular exponent.

5.1.2 Cluster Textures

Information for the clustering process is stored twofold:

1. *Per-pixel information* is stored in a 2D-texture of the same size as the RSM. We use an integer-based texture with two (red & green) 32-bit wide components. The red component stores for each pixel the index of the cluster to which it belongs. The green component is only sparsely filled. It only stores the current cluster index at the pixel where the cluster has its center, which is useful later during EVALUATION when we want to find neighboring clusters.

2. *Per-cluster information* is stored in nine separate 1D-textures. They are comprised of the overall number of pixels belonging to a cluster c_K, its orientation $c_{\hat{\mathbf{n}}}$, view-space position c, position in texture coordinates c_{uv}, flux c_Φ, area c_A, importance c_ρ, irradiance c_E and verification measures c_δ. Note that in order to store additional helper data, our implementation uses more textures than the minimum set of relevant properties we mentioned previously in Section 4.3. Most of the additional data could be computed on the fly, but temporarily storing them has proven to be more efficient. The cluster textures are defined as given below:

Texture	**component(s)**		**channels**		
	# of	**bits per**	**R**	**G**	**B**
POSITION c	3	32bit	$\vec{x}_x$	$\vec{x}_y$	$\vec{x}_z$
NORMAL $c_{\hat{\mathbf{n}}}$	3	32bit	$\hat{\mathbf{n}}_x$	$\hat{\mathbf{n}}_y$	$\hat{\mathbf{n}}_z$
FLUX c_Φ	3	8bit	Φ_r	Φ_g	Φ_b
VERIFICATION c_δ	2	32bit	$\sum \delta_c^2$	$\sum \delta_c$	–
TEX COORD c_{uv}	2	32bit	u	v	–
AREA c_A	1	32bit	current cluster area		
COUNT c_K	1	32bit	# of pixels belonging to cluster		
IMPORTANCE c_ρ	1	32bit	ρ at cluster center		
IRRADIANCE c_E	1	32bit	E at cluster center		

Table 5.2: *1D-textures for cluster information, specifying layout and usage.*

Each of these textures is $1 \times M$ pixels in size. However, the amount of clusters in use at any time can vary dynamically at run-time, based on a user-specific parameter. Thus, the active amount of clusters always lies in the range $m \leq k \leq M$, where m, M define the lower- and upper-bound for manageable clusters in our algorithm. In our implementation, we define them as 16 and 512, respectively.

5.1.3 Lighting and Shadow Buffers

ISM Atlas

The ISM atlas is a single, large texture buffer of size 4096×4096 pixels, consisting of a single 32-bit wide component. It is used to store an array of smaller, point-based shadow maps, where

each of those is of size 128×128 (see Figure 3.7). This yields a theoretical maximum of up to 1024 imperfect shadow maps that can be stored inside the atlas.

Lighting Buffers

Various textures of the same size as the *Camera G-Buffer* are also needed during the illumination steps of the algorithm:

a) One texture to hold the *direct* light contribution.

b) Three 16-bit per-channel textures for the *indirect* light contribution. The first texture holds the *Split G-Buffer* (i.e., a representation of the Camera G-Buffer, but split into $n \times m$ tiles), which is lit using tile-based interleaved shading. The remaining two textures keep the merged and filtered contributions of indirect light.

c) One texture to store the *combined* output of direct- and indirect-light contribution.

Additionally, a *luminance* texture of the scene of size 1024×1024 is used to perform tonemapping during rendering into the back-buffer.

5.1.4 Interleaved Sampling Geometry

In order to accumulate the lighting contributions of many light sources, an additive blending operation has to be performed by the graphics hardware, which involves a lot of read/write locks (instead of just a simple write). Naturally, this puts a lot of stress on the hardware's memory bus and leads to stalls in the pipeline.

In a deferred rendering pipeline, the most trivial, but also slowest solution is to simply perform a fullscreen-pass for each individual light source in order to compute its illumination contribution to every pixel of the scene. Assuming L light sources and K pixels to shade, this amounts to a runtime complexity of $\mathcal{O}(L*K)$ (as mentioned above in Section 5.1.1). Since graphics shaders are usually capable of executing with high parallelism, such a runtime complexity is normally feasible. However, in this case, a read/write lock is instated each time the value has to be updated, so that only one shader thread can read or write a pixel at the same time to ensure correct blending results. This locking behavior is thus detrimental to the high parallelism.

To minimize the bottleneck that appears due to additive blending, we therefore employ *tile-based interleaved shading*. The idea is to split the above problem into multiple smaller problems of the same complexity. Inspired by Laine et al. [25] (Section 3.2.3), we create a tiled representation of the original frame, which is split into $n \times m$ smaller tiles. Each of those smaller tiles will represent a full, but sub-sampled version of the G-Buffer. Thus, instead of rendering one fullscreen quad per light source, we create a structure that consists of $n \times m$ smaller tiles, where each tile is later on only lit by exactly one of the lights in L. The complexity is therefore reduced

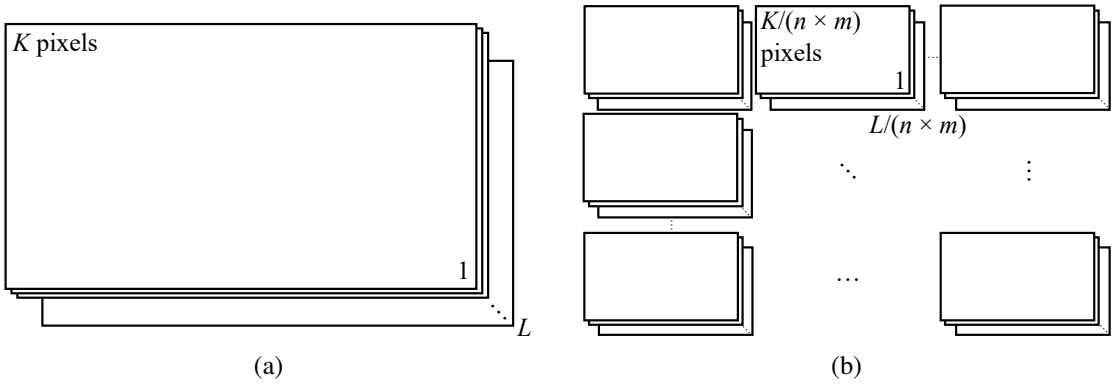


Figure 5.2: *(a) represents the trivial approach where for each light in L, K pixels are shaded. In (b) on the other hand, the basic principle of tile-based interleaved shading is shown: Each quad represents a full but sub-sampled version of the scene, which means less pixels have to be shaded per light, as each light is only used to illuminate a single, small quad instead of the whole scene.*

to a sum over similar complexities

$$\sum_{i}^{n\times k} \mathcal{O}\left(\frac{L\cdot K}{(n\cdot m)^2}\right).$$

This allows the GPU to achieve a higher degree of parallelism, since only $L/(n\times m)$ lights are requesting read/write permissions per quad. Also, the number of pixels to shade per light has also been reduced to $K/(n\times m)$. Figure 5.2 gives a visual explanation of the change and improvement.

Thus, we create a helper structure, the *tiled mesh geometry*, to facilitate interleaved rendering. For our implementation, we generated a structure with a predefined amount of 3×3 tiles in a pre-process on the CPU. This seems to give the most benefit in terms of performance and quality for the amount of indirect light sources we are aiming for. Note that it should also be possible to create this helper structure on the fly from a single full-screen quad using a geometry shader that emits newly created vertices for the specified number of rows and columns. This would yield a more dynamic solution that allows changing the number of tiles during runtime.

The created helper structure is facilitated later during indirect illumination in Section 5.3.3, while the creation of the required buffers that describes how the original pixels are distributed among the tiles has already been summarized in Section 3.2.3 and we describe it in more detail in Section 5.1.

It is also worth noting that any other way of minimizing overdraw during illumination could be used additionally. Among them are, using bounding volumes representing the light sources to limit the region of influence while shading. For instance, one could render a sphere for a point-light source, where the radius is derived from the attenuation factor and only shade pixels that

are within this bounding volume. Other bounding volumes could be utilized for different types of lights (i.e., paraboloid for directional light, pyramid for spot light, etc.). More recent methods [17, 33] use a similar approach to our tile-based interleaved shading, but use much more and smaller tiles including bounding volume calculations computed entirely on the GPU.

5.1.5 RSM Sampling Geometry

During various stages (especially MAPPING and EVALUATION) of our clustering process, we need an array of vertices that can be drawn in order to verify each pixel of a quadratic grid of the RSM and then project it to an arbitrary position inside a 1D-texture. This way, we can write per-cluster data into a 1D-texture, such as the number of pixels belonging to a cluster, which is required during k-Means averaging (discussed in greater detail in Section 5.3.1).

Figure 5.3 shows the layout of the grid that we want to render utilizing an array of sequential points. Each pixel in the grid can be assigned a number i from 1 to M, where M is the size of the RSM and i the size of the squared region inside the RSM that we want to sample. Thus, the numbers 1 through i represent all pixels that have to be rendered in order to yield this quadratic sample area inside the RSM. Each pixel in the grid stores a corresponding (u,v) texture coordinate in RSM texture space as $(u,v) = (x,y)/M$, where (x,y) are the pixel coordinates of the RSM. We flatten the entire grid of size $M \times M$ into a sequential array starting with the (u,v)-coordinates for $i = 1$, then adding the three (u,v) values for $i = 2$, then the additional 5 values for $i = 3$, etc. until we reach $i = M$ adding the final $2i-1$ values to the array. Hence, in order to draw a quad that allows us to sample each pixel of the quadratic region of size $i \times i$ inside the RSM we draw the first i^2 vertices of the array by calling `glDrawArrays(clusterGeomVBO, i * i)`. Algorithm 5.1 outlines the exact creation of the array `clusterGeomVBO`.

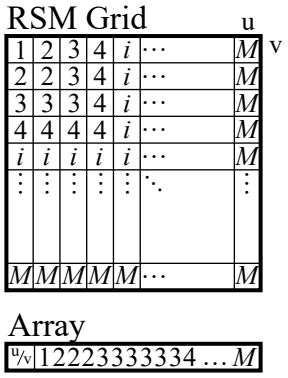


Figure 5.3: *The top represents a grid of quadratic size where the numbers 1 through i represent the pixels that have to be rendered for a quadratic region of width/height i inside the RSM with corresponding size $M \times M$. For this to be easily renderable, we need to create an array as layed out at the bottom.*

Algorithm 5.1 *Generating a vertex geometry for sampling each pixel of a quadratic region of arbitrary size.*

```
1:  procedure CLUSTERMAPGEOMETRY
2:      Initialize clusterGeom array that will hold points
3:      halfTexel ← 0.5/M
4:      for i ← 0 through M do
5:          n_i ← 2 · i − 1
6:          Init x, y and u, v
7:          u ← ((i − 1)/M) + halfTexel
8:          for y ← 0 through n_i/2 do
9:              v ← (y/M) + halfTexel
10:             clusterGeom.Add(u, v)
11:         v ← u
12:         for x ← 0 through n_i/2 do
13:             u ← (x/M) + halfTexel
14:             clusterGeom.Add(u, v)
15:     Create clusterGeomVBO from clusterGeom
```

5.2 Point Generation

As stated previously, our algorithm is designed to work on arbitrary scenes consisting of triangle meshes. However, since we rely on a point-based representation of the scene for use with the Imperfect Shadow Maps, a point-generation scheme based on tessellation is required to create that point-based model. By hooking into the model-loading methods, we gain access to the raw mesh data (vertices, faces, etc.), which we analyze in order to find a few scene-describing properties:

A_{min}, A_{max} Smallest and largest triangle area, as well as

e_{min}, e_{max} shortest and longest triangle edge.

We also supply user-definable `min`- and `max`-tessellation factors f_{min}, f_{max} to keep the tessellation levels within reasonable bounds. To define a combination of minimum- and maximum tessellation factors in a single term, we use the notation $f_{[min,max]}$. From these values we derive *inner*- (t_A) and *outer*-tessellation factors t_e:

$$t_A = \texttt{max}\left(\texttt{ceil}\left(\texttt{smoothstep}(A_{min}, A_{max}, A) \cdot f_{max}\right), f_{min}\right) \tag{5.3}$$

$$t_e = \texttt{max}\left(\texttt{ceil}\left(\texttt{smoothstep}(e_{min}, e_{max}, e) \cdot f_{max}\right), f_{min}\right) \tag{5.4}$$

These equations compute tessellation factors along the edges and inside the face of a triangle such that both the longest triangle edge and the largest triangle face in a scene correspond to the maximum tessellation factor f_{max}, and the shortest edges and smallest faces correspond to the smallest possible tessellation factor f_{min}, respectively. All other faces and edges will be assigned

interpolated tessellation factors that lie in the range $f_{[min,max]}$. The `smoothstep`(l,u,t)-function is a higher order Hermite interpolation polynomial commonly used in computer graphics where $l \leq t \leq u$. It is a standard function available in both `GLSL` and `HLSL` shading languages that performs smooth interpolation in the range $[0,1]$ through a 3^{rd} order polynomial given below as

$$3^{rd} \text{ order: } 3x^2 - 2x^3$$
$$5^{th} \text{ order: } 6x^5 - 15x^4 + 10x^3$$

where $x = (t-l)/(u-l)$. Also given is a 5^{th} order variant of the `smoothstep` function, while even higher order variants are known to exist, which further minimize the slope of the interpolation function. The tessellation factors created from Equation (5.3) and (5.4) feed the two interchangeable methods of point generation used by our algorithm:

STATIC TESSELLATION: Using the CPU we create a static point-representation at the start of the application after the entire scene has been loaded. We iterate all triangles within the scene and compute inner- and outer-tessellation factors according to the equations given above. For each triangle, step-sizes s_a, s_b, s_c along their edges a, b, c are computed as demonstrated in Algorithm 5.2. To generate additional points on the triangles, we employ a custom iteration scheme which is based on barycentric coordinates. The barycentric weights are incremented in each iteration using the step-sizes computed before. Using this static tessellation method, simple dynamic scenes that support translation, scaling and rotation of whole objects are possible.

Algorithm 5.2 *Point Generation on static mesh geometry*

Require: $A_{min}, A_{max}, e_{min}, e_{max}$, Meshes
1: **procedure** POINTGEN
2: $f_{max} \leftarrow 64, f_{min} \leftarrow 5$ ▷ Tessellation factor range $f_{[5,64]}$
3: **for each** *mesh* in Meshes **do**
4: **for each** *triangle* in *mesh*.Triangles **do**
5: $triangle.t_A \leftarrow$ Calc Equation (5.3)
6: **for each** *edge* in *triangle*.Edges **do**
7: $edge.t_e \leftarrow$ Calc Equation (5.4)
8: $A, B, C \leftarrow$ the three triangle positions
9: $a, b, c \leftarrow$ the three edges opposite of the positions
10: $s_a \leftarrow 1.0/(a.t_e + triangle.t_A)$ ▷ Define stepsizes along the three edges
11: $s_b \leftarrow 1.0/(b.t_e + triangle.t_A)$
12: $s_c \leftarrow 1.0/(c.t_e + triangle.t_A)$
13: **for** $\gamma \leftarrow 0.0$ **through** 1.0 **step** s_c **do** ▷ Generate points on triangle
14: $\alpha \leftarrow 1.0 - \gamma, \beta \leftarrow 0.0$ ▷ Use barycentric coordinates α, β, γ
15: **while** $\alpha > 0.0$ **do** ▷ to iterate over triangle area using step-sizes s_a, s_b, s_c
16: *mesh*.Positions.Add$(A * \alpha + B * \beta + C * \gamma)$
17: $\alpha \leftarrow \alpha - ((1.0 - \alpha) * s_a + (1.0 - \beta) * s_b + (1.0 - \gamma) * s_c)$
18: $\beta \leftarrow 1.0 - \alpha - \gamma$

DYNAMIC TESSELLATION: In order to support fully dynamic objects that are dynamically created or morphed on a per-vertex basis, for instance, surfaces defined by dynamic implicit functions ("blobby objects", liquid surfaces in smoothed particle hydrodynamics, etc.), we employ dynamic tessellation using the graphics hardware's built-in tessellation unit. Here we use the same basic principle as above and compute equations (5.3) and (5.4) inside the tessellation control shader by assigning `gl_TessLevelInner` $\leftarrow t_A$ and `gl_TessLevelOuter` $\leftarrow t_e$. The barycentric coordinates for the new points on the surface are computed by the hardware, which eliminates the need for a custom iteration as in the static tessellation case. Hence, we only have to interpret the given barycentric coordinates in order to translate the created points on the face of the triangle. More information on this specific technique is given in Section 5.3.2.

5.3 Rendering Loop

In this section, we describe the main rendering loop, which consists of the individual steps to be executed in each consecutive frame. An overview of the various steps is given in Figure 5.4 to the right (adapted from a similar figure in Chapter 4), while the following sections describe the more advanced steps in greater detail. All of the below discussed methods are designed to be executed on the GPUs programmable shader pipeline.

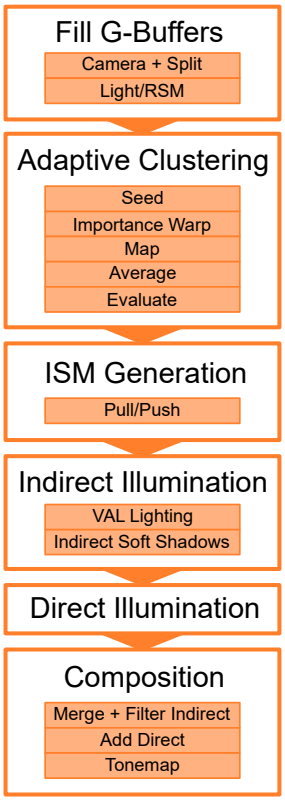


Figure 5.4: *Overview of the algorithm's pipeline.*

5.3.1 Clustering

Since generating the G-Buffers according to the layout discussed in Section 5.1 is rather straight-forward, we get right to the clustering itself, which as discussed in Chapter 4, can be subdivided into five substeps.

Distribution

We start by seeding quasi-uniformly distributed clusters inside our RSM texture using positions generated from a 2D-Halton sequence [16] (described in Section 4.4). This gives us pairs of values (u,v) in texture coordinates which are stored in a *Vertex Buffer Object* (VBO) of size M, containing two data streams: a) A single integer value containing the index of the cluster and b) the generated (u,v) coordinates of the cluster.

This stage is fed by two data sources: The first is the normal SEEDING pipeline utilizing the *(Re-)Seed List* (refer to Figure 4.4) that checks for each cluster whether the number of assigned pixels is zero. If so, the cluster will be reset by relocating it to a new position from the Halton sequence. This usually happens in the very first frame for all clusters, and then occurs sporadically throughout the algorithms lifetime whenever a cluster is eliminated by the process (i.e., no pixels

are assigned to a cluster). We use the *transform feedback* capabilities of the GL-device to return the indices of the affected clusters and create new positions for them.

The second mechanism feeding into this stage is the EVALUATION process (described later), during which a cluster can be identified as either too small or too large for its vicinity. Clusters that are too small will be eliminated by manually resetting their pixel-count to -1 and adding them to the *Free-Cluster List* L_f. Clusters that are too large will be split in half by taking a cluster from L_f and re-inserting it into the distribution within that split cluster using a random offset. If any free clusters are still remaining after the evaluation process, they are injected into the re-seed pipeline and will be assigned new positions according to the Halton sequence. These two mechanisms ensure that all clusters are in use in each frame.

Importance Warping

Through the process of importance warping, our initial seed positions are then re-distributed according to the importance map ρ, which is stored in the RSM. The idea behind this is to give higher importance to surfaces that we deem more important than others: Surfaces with reflection characteristics favored by our BRDF-dependent term ρ_f using Equations (5.1) and (5.2). Additionally, surfaces that have a greater effect on the currently viewed part of the scene according to ρ_v as defined in Equation (4.2). From a technical point of view, we perform importance warping as described in Section 4.5 on the GPU using a transform-feedback loop. The pseudo-code is given in Algorithm 5.3.

Algorithm 5.3 *Performing importance warping on a single point using a transform-feedback loop, where $\vec{x}$ is the original seed position, ρ_{tex} defines the importance map texture and n is the number of MIP-levels to traverse. The result is returned as $\vec{x}_{warped}$.*

Require: $\vec{x}, \rho_{tex}, n$

1: **procedure** IMPORTANCEWARPING
2: **for** $i \leftarrow 1$ **through** n **do** ▷ iterate through the MIP-levels starting at the 2nd coarsest
3: $\rho_q \leftarrow$ `sample`(ρ_{tex}, at each of the 4 quadrants q)
4: $percent_q \leftarrow 1.0 / \sum \rho_q$
5: $tile_{range} \leftarrow q_{width} \cdot 2$
6: **if** $\vec{x}_y$ in $q < tile_{range} \cdot percent_{bottom}$ **then** ▷ point in lower-half
7: $pos_y \leftarrow \vec{x}_y$ in $q / percent_{bottom} \cdot 0.5$ ▷ warp vertically
8: $percent_{left} \leftarrow percent_{bottom\text{-}left} / percent_{bottom}$
9: **if** $\vec{x}_x$ in $q < tile_{range} \cdot percent_{left}$ **then** ▷ warp horizontally
10: $pos_x \leftarrow \vec{x}_x$ in $q / percent_{left} \cdot 0.5$
11: **else**
12: $pos_x \leftarrow tile_{range_x} - (tile_{range_x} - \vec{x}_x \text{ in } q) / (1.0 - percent_{left}) \cdot 0.5$
13: **else** ▷ point in upper half
14: Perform similar to lines 7-12 for upper half
15: $\vec{x} \leftarrow \vec{x} + pos_{xy}$
return $\vec{x}_{warped} \leftarrow \vec{x}$ ▷ return the warped position

After the warping procedure is done, the clusters are initialized by writing the now finalized cluster-attributes to the 1D-texture map (refer to Table 5.2): The POSITION c, NORMAL $c_{\hat{\mathbf{n}}}$, FLUX c_Φ, TEX COORD c_{uv} and IMPORTANCE c_ρ can be initialized, since they are defined from the properties of the cluster center.

Mapping

After our cluster centers have been warped to their final position, we can continue to assign each pixel of the RSM to the cluster to which it belongs. To that end, we render a quadrilateral area of size $i \times i$ pixels, centered around each cluster's midpoint and aligned with the texture's uv-axes (i.e., non-rotated). Here, i defines the maximum size of a cluster, which is a user-definable value usually in the range $[64, 256]$ depending on the amount of clusters in use.

In the fragment shader, we compute the distance $\mu(c_i, \vec{x})$ between the current fragment $\vec{x}$ and the cluster center c_i according to Equation (4.4). The cluster's attributes are read from the 1D-texture maps initialized in the previous step and the attributes at the current position $\vec{x}$ are reconstructed from the RSM. We write the index of the currently rendered cluster into the red channel of the cluster-map (see enumeration item 1 in Section 5.1.2) at position $\vec{x}$. The result for μ is written to `gl_FragDepth`. This way, we make use of the GPUs depth-testing capabilities to efficiently decide which cluster has the smallest distance to the current pixel $\vec{x}$ (see Figure 5.5).

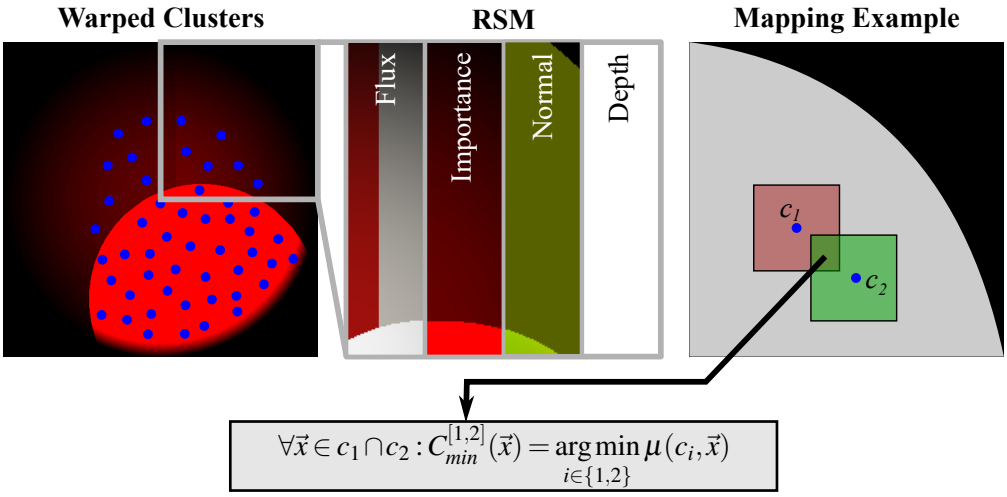


Figure 5.5: *Illustration of the cluster mapping process. After importance warping, each pixel of the RSM is assigned to its nearest cluster c_i according to Eq. (4.5) using distance metric μ defined in Eq. (4.4).*

After this process is completed, we know for each pixel of the RSM to which cluster it is currently assigned. In a final pass we utilize the cluster map geometry (Section 5.1.5) by iterating over each pixel of the cluster map in order to compute the missing values for AREA c_A and COUNT c_K of the 1D-texture map. The AREA of the cluster is estimated as the area of a

maximum bounding circle that fully encompasses the cluster. This is achieved using `GL_MAX`-blending on the result of $r_c^2\pi$, where $r_c = \texttt{dist}(c,\vec{x})$, the Euclidean distance between the cluster center and the currently sampled position $\vec{x}$. This usually leads to overestimated cluster areas, that will also most likely intersect one another. The areas are therefore not mathematically exact, however, they serve their purpose without negatively impacting the result of the algorithm.

COUNT, on the other hand, is calculated via additive-blending, by simply adding 1 to the cluster's counter for each occurrence of the cluster's index.

Averaging

The next step in the clustering pipeline, inspired by *k*-Means clustering, is the *averaging* process. Since we now know for each pixel to which cluster it belongs, we recompute each cluster's center by averaging the values of the cluster-position, -texture-coordinates, -normal and -importance. These are computed in the vertex shader, utilizing the *cluster-map geometry* introduced in Section 5.1.5, by dividing the current values by the COUNT c_K. The final values are again achieved through additively blending them together in the fragment shader stage, which is possible according to distributivity as $\sum_i \frac{c_i}{c_K} \equiv \frac{\sum_i c_i}{c_K}$. This works analogously for $c_{uv}, c_{\hat{\mathbf{n}}}$ and c_ρ.

c_Φ, as well as c_E are exempt from the averaging. Both of these values are computed anew in a separate pass, where c_Φ is simply read from the RSM at the new, averaged cluster center and c_E is recomputed according to Equation (2.1).

Evaluation

Cluster evaluation (or verification) is actually performed at the beginning of the clustering pipeline. However, in order to cover the prerequisites for understanding how the clustering works without evaluation, we describe it here at the end. Thus, you are now familiar with the basic principle of our frame-by-frame iterative *k*-Means clustering and its nomenclature, which makes it easier to understand how the evaluation step adjusts the results of clustering.

As a first step, the cluster map's green channel is updated, by only storing the index of the cluster at the pixel that represents its center. This is necessary, since with our next step, we want to find neighboring clusters. We do this very similarly to the process we described above in Mapping, where we render a quadrilateral centered around the cluster's midpoint. This time, however, the quad has a size of $2i \times 2i$ to make sure it is large enough to find neighboring cluster centers. Again, we utilize the cluster-map geometry to render $2i \times 2i$ vertices, such that we are able to sample each relevant individual pixel of the cluster map. In this case, we look in the green-channel of the cluster map (which we filled previously) for other cluster centers and using the geometry shader, only emit these vertices to the fragment stage. For them we simply compute δ_c and δ_c^2 according to Equation (4.6). The values for all found neighboring clusters are additively blended together and stored in the 1D-texture map as VERIFICATION c_δ to form the sum of these values. They act as the base for evaluating whether a cluster is too large or small for its neighborhood.

After these pre-processing steps are completed, we employ three *compute shaders* to perform merging and splitting (pipeline depicted in Figure 5.6) of the clusters according to the verification metric (4.8). From the active amount of clusters k, we take a random subset of usually $k/3$ clusters and run the verification pipeline on these selected clusters only. The process begins with

MERGING, where each randomly selected cluster is evaluated by computing the weighted arithmetic mean $\bar{\delta}$ according to Equation (4.7) from the values stored in the 1D-texture VERIFICATION c_δ. Additionally we calculate δ_c for the current cluster and compare it to $\bar{\delta}$. According to the decision in Equation (4.8) we merge a cluster if $\delta_c - \bar{\delta} < -\varepsilon_m$, in which case, we add the cluster-index to a free-cluster list L_f and increment the free-cluster counter c_f.

At the end of this step, we end up with a free-cluster list of size $0 \leq c_f \leq k/3$ and continue with

SPLITTING, which only runs in case of $c_f > 0$. Otherwise, L_f is empty and no clusters are available that can be used for splitting. However, if some clusters have been merged beforehand, we take the same randomly chosen $k/3$ clusters and check for each cluster c if $\delta_c - \bar{\delta} > +\varepsilon_s$. If this comparison evaluates as `true` and a free cluster is available ($c_f > 0$), we take the last (i.e., at position c_f) cluster from L_f and initialize its seed position within the current cluster's radius. Afterwards, we decrement c_f by 1. This process is continued for all $k/3$ clusters.

As mentioned previously, in this case, we circumvent the normal seed-pipeline that is based on Halton-samples and instead initialize the position of the cluster taken from the free-cluster list to a random position near the cluster that is to be split.

In the end, we perform a *clean-up*, which ensures that any remaining free clusters (i.e., $c_f > 0$) are handled by our normal re-seed pipeline through setting their COUNT $c_K \leftarrow 0$.

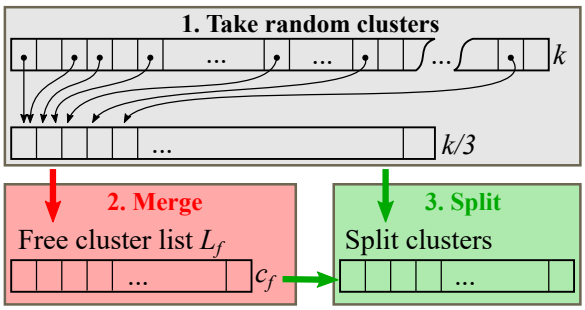


Figure 5.6: *The evaluation pipeline checks $k/3$ random clusters and performs merging on eligible clusters by adding them to a free-cluster list of size c_f. The random clusters in combination with the free-cluster list are in turn used in the third step to feed the splitting process.*

5.3.2 ISM Generation

After the clustering pipeline has finished, we generate the imperfect shadow maps using a point representation of the scene that is rendered into a single large texture, called the ISM Atlas.

To reiterate, ISMs [39, 41] are sparsely filled shadow maps that are generated by splatting point samples representing the scene into a large amount of small shadow maps. In our case, each cluster representing a virtual area light uses exactly one corresponding ISM, yielding $m \leq k \leq M$ imperfect shadows maps. Reusing our nomenclature, m and M define the lower- and upper-bound for manageable clusters and k is the amount of currently active clusters.

As mentioned previously, we employ two different splatting strategies: One uses a static point representation of the scene generated in a pre-process, the other one is based on a dynamic representation utilizing the graphics hardware's tessellation capabilities.

Static Representation

For the static representation, we use the vertex buffers that were generated during the start-up of the application. We use a scene graph where each node is assigned a child model that represents the point cloud sampled from its surface. By attaching it as a child, we ensure that each transformation (i.e., translation, rotation, scaling) applied to its parent polygonal mesh is also applied to the point representation.

During ISM rendering, we traverse the scene graph, but only render the point clouds, ignoring all other mesh types. In this case, we employ three shaders: *vertex*-, *geometry*- and *fragment*-shaders. The vertex and fragment shaders are straight forward, with the former simply passing the current vertex $\vec{x}$ and an integer i used to compute the receiving cluster to the geometry stage. The latter writes the computed depth-value to the ISM atlas texture. The vast bulk of the computation is handled by the geometry shader (i.e., it lies in between the vertex- and fragment-stages). For the integer passed from the vertex stage we chose to use the built-in `gl_VertexID` variable. We compute the index of the receiving cluster as $i \bmod N$, thus assigning point splats in an interleaved manner to the ISM atlas. Based on that index, we can perform lookups into the 1D-texture maps containing cluster information, such as POSITION c and NORMAL $c_{\hat{\mathbf{n}}}$. From these values we compute a view-matrix, which allows us to transform and project a point sample to an ISM viewed from a given cluster center using paraboloid mapping [6]. By further computing the distance between the point sample and the cluster center, we can estimate a *projected* size (`gl_PointSize`) for the point such that far away points yield smaller splats, while those close to the cluster center produce larger splats. Additionally, by looping the geometry-shader code, we render each point s times, controlled by the splat factor s_f, but into different clusters through adding the loop's iteration counter to i before applying the modulo operation.

Dynamic Representation

The dynamic point representation is handled very similarly. Since we use the GPUs tessellation hardware to dynamically generate point clouds from polygons, the scene graph is traversed normally and only the polygonal meshes are drawn (i.e., the static point clouds are ignored). To

this end, two additional shaders are attached after the vertex stage: the *tessellation-control* and *-evaluation* shaders. In the control-shader we set the tessellation factors t_A and t_e in exactly the same way as we did during the static point mesh generation (according to Equations (5.3) and (5.4)). The evaluation shader replaces the output of the vertex stage, by computing a random cluster index i from a primitive identifier and the barycentric weights as

$$\begin{aligned} i = & \texttt{gl\_PrimitiveID} + \\ & \texttt{gl\_TessCoord.x} \cdot 7 + \\ & \texttt{gl\_TessCoord.y} \cdot 89 + \\ & \texttt{gl\_TessCoord.z} \cdot 991 \end{aligned}$$

By multiplying the barycentric weights with prime numbers we achieve additional randomness for the computed index.

The barycentric coordinates stored in `gl_TessCoord` are used to generate the new interpolated vertex position on the triangle. The new position on the triangle and the computed receiving cluster index are then sent to the geometry stage and subsequently the fragment shader, which remain unchanged from the description given in Static Representation.

Pull-Push

With large amounts of ISMs, rendering proportionally larger amounts of point splats quickly becomes a performance bottleneck in the rendering pipeline. Thus, similar to the original implementation [39], we always render the same amount of point splats but into varying amounts of ISMs. Since the so rendered ISMs can become very sparsely filled, one way to accomplish a more dense scene representation is by performing a *pull-push* operation on the ISMs in image space. To this end, a custom image pyramid is created, where we start from the finest resolution level $l \leftarrow 0$ and perform j iterations of the *pull*-phase: In each iteration, a median-of-four is computed using data from the l^{th} level of the ISM texture. Each quadruple of values from level l forms a single new value – the median – in level $l+1$ (i.e., the next coarser level), thus *pulling* data down in the hierarchy. l is incremented in each iteration and the process continues until $l \equiv j$.

The *push*-phase works in the opposite direction: Starting from level j, we *push* data up the hierarchy, and use the pushed depth values to fill pixel that contain no data. This way we fill the holes in each finer level. Again, this iterative process is continued until we reach the finest level in the image pyramid. The result of such an operation is depicted in Figure 5.7.

Figure 5.7: *Two iterations of Pull-Push performed on a single ISM. Smaller holes (top) are filled by the algorithm (bottom).*

5.3.3 Indirect Illumination

From the information created by the clustering pipeline and with the ISM rendering completed, we now possess all necessary data to perform indirect illumination. We employ tile-based in-

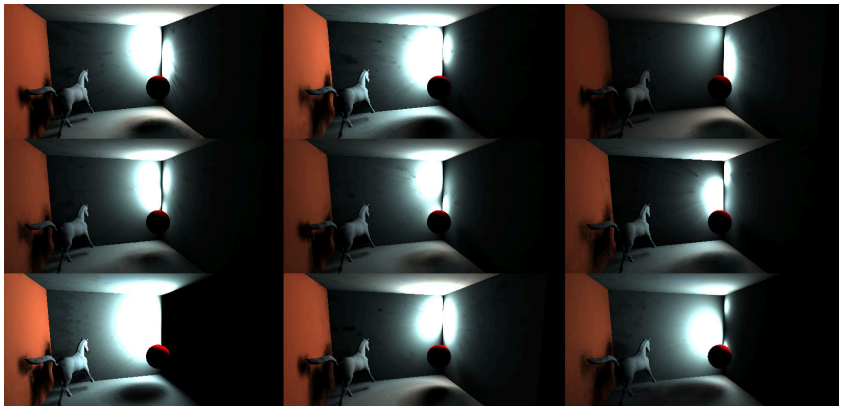


Figure 5.8: *The split G-Buffer representation after indirect illumination using tile-based interleaved shading. Each tile is lit by a different subset of VAL clusters using the tiled mesh geometry.*

terleaved shading via our tiled mesh geometry (refer to Section 5.1.4) using a single tile per VAL cluster. As shown in Figure 5.2(b), multiple VALs will render into the same tile, which means we have to additively blend the indirect illumination results together into the first indirect illumination buffer containing the *split indirect illumination* (see Figure 5.8).

Indirect illumination itself is computed using the data in the 1D-textures for each cluster. The basic principle for computing the illumination is the same as with any other light source. The main difference is that in this case we deal with area light sources. Plugging our predefined variables into Equation (3.9) we get

$$F(\vec{x},c) = \frac{\cos\theta_{\vec{x}} \cdot \cos\theta_c}{\pi \cdot \|\vec{x}-c\|^2 + c_A} = \frac{\left\langle \hat{\mathbf{n}} | \hat{\boldsymbol{l}} \right\rangle \cdot \left\langle c_{\hat{\mathbf{n}}} | -\hat{\boldsymbol{l}} \right\rangle}{\pi \cdot \|\vec{x}-c\|^2 + c_A} \tag{5.5}$$

to compute the form factor between our disk-shaped light source c and the receiving pixel $\vec{x}$, where $\hat{\boldsymbol{l}}$ is the directional vector from $\vec{x}$ to c. The single-bounce diffuse light intensity for a single cluster is then calculated as

$$I_d(\vec{x},c) = k_d \cdot F(\vec{x},c) \cdot V(\vec{x},c)$$

where k_d represents the diffuse reflection characteristics and $V(\vec{x},c)$ denotes the soft shadow term discussed below. The form factor arithmetics are visualized in Figure 5.9.

Soft Shadows from Indirect Light

One important thing missing in the diffuse lighting term I_d given above is the visibility component $V(\vec{x},c)$. The original ISM implementation relies on having a large amount of ISMs to sample from. By virtue of this, smooth indirect shadows are achieved through blending the result of many binary shadow lookups together. As mentioned previously, we however employ percentage-closer soft shadows (discussed in Section 3.1.1), a technique that is usually combined with standard shadow mapping. In our case, we use the percentage-closer filtering (PCF)

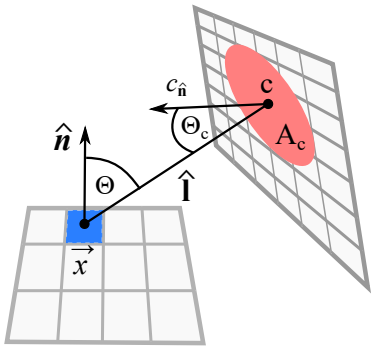


Figure 5.9: *Arithmetics of the form factor in use in our algorithm. The intensity of light reflected at the blue pixel $\vec{x}$ is dependent on $\cos\theta_{\vec{x}}, \cos\theta_c$, the cluster size c_A and the distance between the point and the cluster center, given as $\|\vec{x}-c\|^2$.*

for imperfect shadow maps, in order to reduce the amount of required virtual lights and thus gain higher performance.

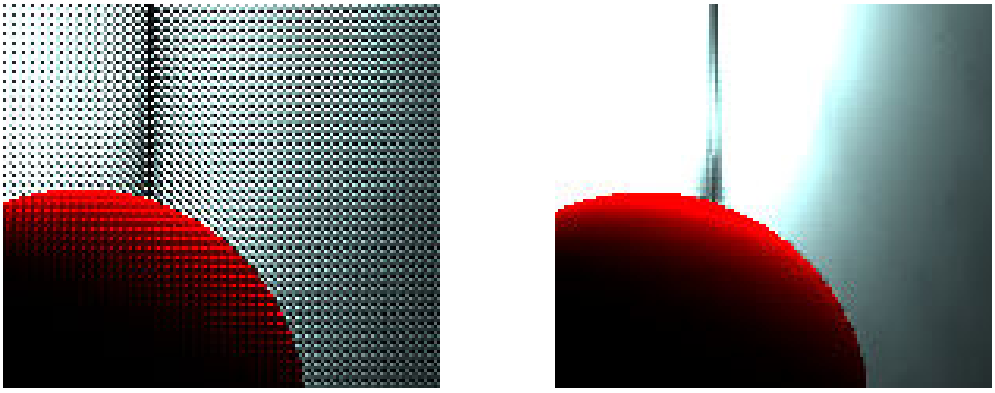
There is actually little difference between applying PCSS on a normal shadow map or on imperfect shadow maps since the filtering itself works practically the same. We just have to make sure to transform a current pixel into the paraboloid space used by the ISM and then continue with the PCSS sampling as usual. Two important remarks have to be made:

1. At this point, we also remember that the penumbra size depends not only on the distance between shadow blocker and receiver (i.e., the closer the blocker is to the receiver the smaller the penumbra and vice versa), but also on the size of the light source. Refer again to Figure 3.1(b). This is where the combination of our algorithm with PCSS especially benefits: since our indirect lights are area lights of specific sizes, we can accurately estimate a penumbra region.

2. Instead of varying the kernel size itself, as suggested in Section 3.1.1, it is more practical to keep the kernel size static and instead scale the corresponding lookup coordinates according to the estimated penumbra size. To that end, we employ a static kernel of size 16, that is initialized with Poisson-disk samples [12] on the unit sphere. These values are then, upon lookup, rescaled to the corresponding lookup region as computed by the penumbra size estimation given in Equation (**3.2**). Using these rescaled coordinates, we perform 16 shadow lookups to compute the final, smooth visibility term $V(\vec{x}, c)$.

Merging and Filtering the Split Indirect Illumination

After the gathering of the indirect light into the split representation (see Figure 5.8) is completed, we need to merge the split representation back into a single image. This is achieved by reversing the split operation introduced in Tiled G-Buffer Illumination in Section 3.2.3. For a given screen coordinate (x_m, y_m) in the merged representation, we compute the tile (i, j) from which to gather



(a) (b)

Figure 5.10: *(a) A mosaic pattern emerges after the join operation is performed on the split illumination. (b) Through application of a geometry-aware box-blur the mosaic pattern vanishes almost completely.*

the split data as

$$\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} x_m \\ y_m \end{pmatrix} \bmod \begin{pmatrix} n \\ m \end{pmatrix}$$

where (n,m) is the number of tiles in horizontal and vertical dimensions. The final sample position in screen coordinates (x_s, y_s) for the split texture is then calculated by

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix} \cdot \begin{pmatrix} w \\ h \end{pmatrix} + \begin{pmatrix} x_m \\ y_m \end{pmatrix} / \begin{pmatrix} n \\ m \end{pmatrix}$$

where the vector (w,h) gives width and height of a single tile in screen-space. Subsequently, (x_s, y_s) can simply be used to lookup the merged value at (x_m, y_m). This process results in a mosaic pattern on the merged image that appears due to the interleaved shading, which can be removed through an additional step: A geometry-aware filtering technique as presented in Section 3.2.3 performs filtering based on depth-discontinuity Δ_α, as well as normal-similarity Δ_β, using a filter-kernel of size $n \times m$. This convolution uses `step`-functions in combination with two thresholds α and β to determine whether a sample s is similar enough to the current position $\vec{x}$ and normal $\hat{\mathbf{n}}$ to include it in the filtering:

$$\Delta_\alpha = 1.0 - \texttt{step}(\alpha, \texttt{abs}(\vec{x}_z - s_z))$$
$$\Delta_\beta = \texttt{step}(\beta, \langle \hat{\mathbf{n}} | s_{\hat{\mathbf{n}}} \rangle)$$

Only in the case when both similarity conditions are met, i.e., $\Delta_\alpha = \Delta_\beta = 1$, the current pixel is included in the computation of the filtered value. The results of the merging process and its final filtered version are shown in Figure 5.10.

5.3.4 Direct Illumination & Image Composition

Our scene's main light emitter is a spotlight with angular falloff as depicted in Figure 5.1. As the BRDF for our scene we employ Lambert's law in combination with the Blinn specular model

(see Section 2.2.3). To account for visibility we perform shadow mapping by reusing the depth information already stored inside the RSM. This way we can circumvent another costly object rendering pass that would only be used for shadow-map generation. The downside to reusing the already available depth information is, however, that the RSM is a rather low resolution texture (512×512 pixels), where the contained depth information is also rather prone to aliasing and stair-stepping artifacts. In order to counter these artifacts, we employ 25-tap PCF filtering during shadow lookup, which achieves sufficiently soft shadows for use in a real-time application.

Image Composition

The final step in our pipeline before rendering into the back-buffer is to combine the *filtered indirect illumination* contribution with the *direct illumination* into a 16-bit wide final-gather texture. In order to present the information on the screen, we have to ultimately rescale the 16-bit values to an 8-bit per-channel range through means of a tonemapper [38]. An example of a scene rendered with our algorithm is shown in Figure 5.11.

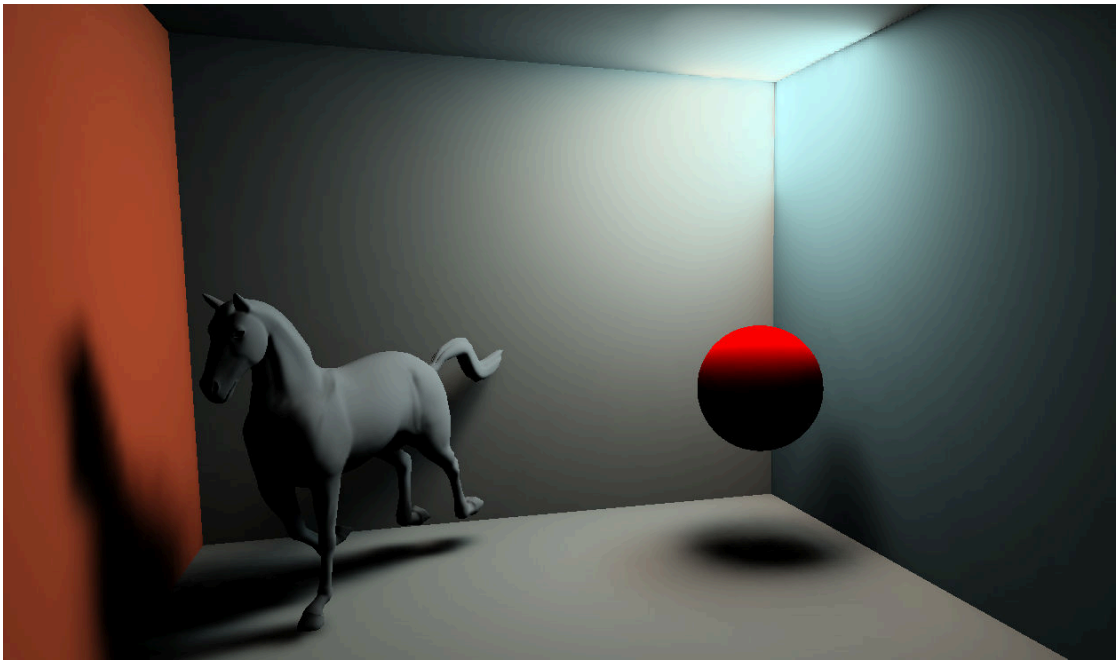


Figure 5.11: *A Cornell-box scene rendered with our algorithm. The spotlight in the upper-right corner of the box acts as the area light that is indirectly illuminating the scene by means of 128 individual clusters. Shadows from the horse and sphere are rendered for the indirect lights through PCSS-sampling of their ISMs.*

CHAPTER 6

Results

In the previous chapters we introduced our new method called *Adaptively Clustered Reflective Shadow Maps* (ACRSM), and discussed the theoretical background and practical implementation. In this chapter we present our results by highlighting the improvements we made with respect to global illumination (see Section 6.1), ISM generation (refer to Section 6.2) and especially our adaptive clustering algorithm (in Section 6.3). We round out our results chapter with some performance measurements in Section 6.4.

6.1 Global Illumination

The results regarding global illumination are split into two separate sections, by first highlighting the differences regarding indirect lighting itself (see Section 6.1.1). Secondly, we present results for the indirect shadows in Section 6.1.2. Finally, Figure 6.3 shows results using three different scenes rendered with our method and provides detailed comparisons between our method and a reference implementation using VPLs and standard ISMs.

6.1.1 Indirect Lighting

Remember that one of the alleged improvements is supposed to stem from the fact that we interpret the virtual lights that we use for indirect illumination as *clusters* with a corresponding area, thus allowing us to use a slightly more advanced form factor (see Equation (5.5)) to simulate *virtual area lights*. This should in comparison provide a significant improvement over trivial *virtual point light* sources in certain scenarios where point light sources have proven to be problematic. A simple Cornell-Box is used, since an unoptimized clustering algorithm may have severe trouble with the many "hard" boundaries in this scene, which can easily lead to bad virtual-light distributions with a moving spot light. Depending on the illumination type (i.e., VAL or VPL) used for the indirect light, image quality can degrade severely in these cases. Comparisons are given in Figure 6.1, which showcases pure indirect illumination results (i.e., without indirect shadows) when using VPL or VAL lighting for good and bad virtual light distributions. From

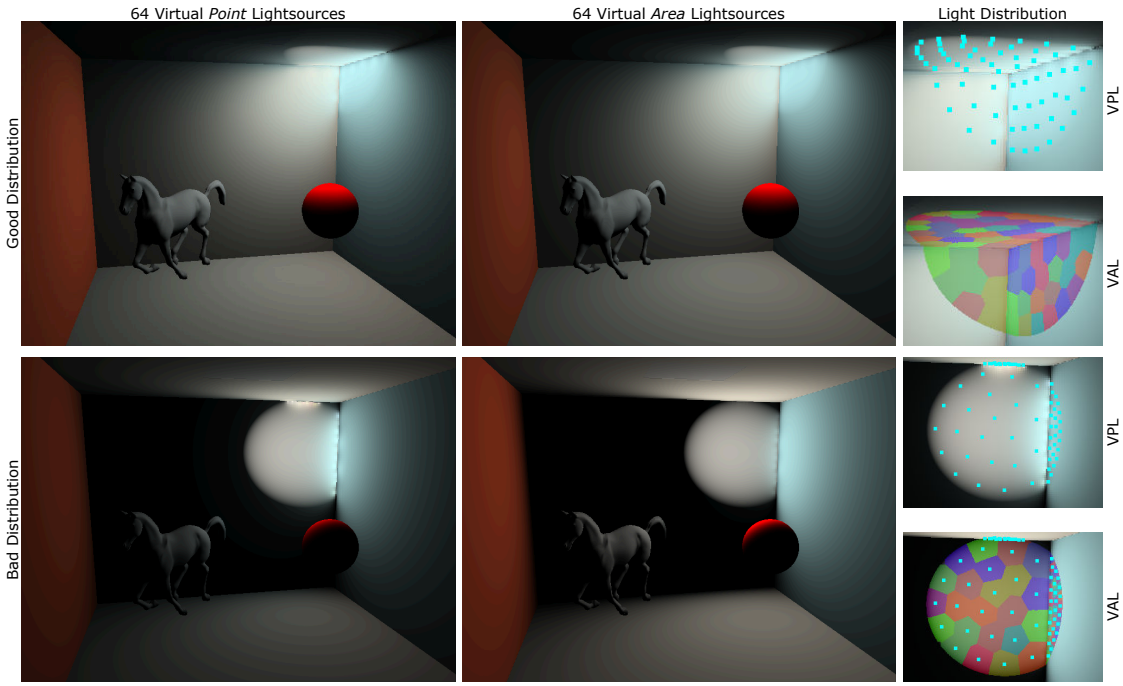


Figure 6.1: *The same scene rendered using* 64 *indirect light sources under different lighting conditions and with either VPL or VAL lighting. Shadows from indirect light sources are disabled in this comparison to emphasize the differences in pure illumination. In the top, an ideal light distribution is used: The resulting illumination is nearly identical in both cases. Slight differences only occur due to more pronounced virtual light singularities in the case of VPLs. In the bottom, a bad distribution of virtual lights is used. This time, the differences are more extreme: In the VPL case, the lighting sparks from the singularities become disturbingly implausible, while the VALs' singularities on the other hand remain contained. Another, more pronounced difference is in the rest of the scene's illumination: Most of the VPLs' energy is spent on the small regions of the spot light where a large amount of the virtual lights are distributed, thus the scene appears unnaturally dark as the large spot light area on the back wall contains a rather limited amount of the overall VPLs. In the VAL case, however, each cluster is aware of its individual size with respect to the overall spotlight's area, which allows us to rescale the amount of energy reflected by each virtual light source according to that size. The illumination remains more plausible, with the large spotlight area on the back wall reflecting more energy than the small regions on the ceiling or the right wall although more clusters have been distributed there.*

these comparisons we can easily see that the VAL illumination we employ in ACRSM is superior in all cases when compared to a standard VPL illumination.

6.1.2 Indirect Shadows

Another goal of our approach is to achieve smoother soft shadows from a smaller amount of virtual light sources via the use of a more sophisticated shadow sampling technique. Using the percentage-closer soft-shadow technique we circumvent the problem of under-sampling that would usually occur due to using less imperfect shadow maps. In Figure 6.2 we compare the indirect shadow terms of our ACRSM method versus a standard ISM implementation and provide detail comparisons of the final composition as well as the sole shadow term. Note that all

comparison shots in this section are rendered using dynamic point generation with $f_{[5,32]}$.

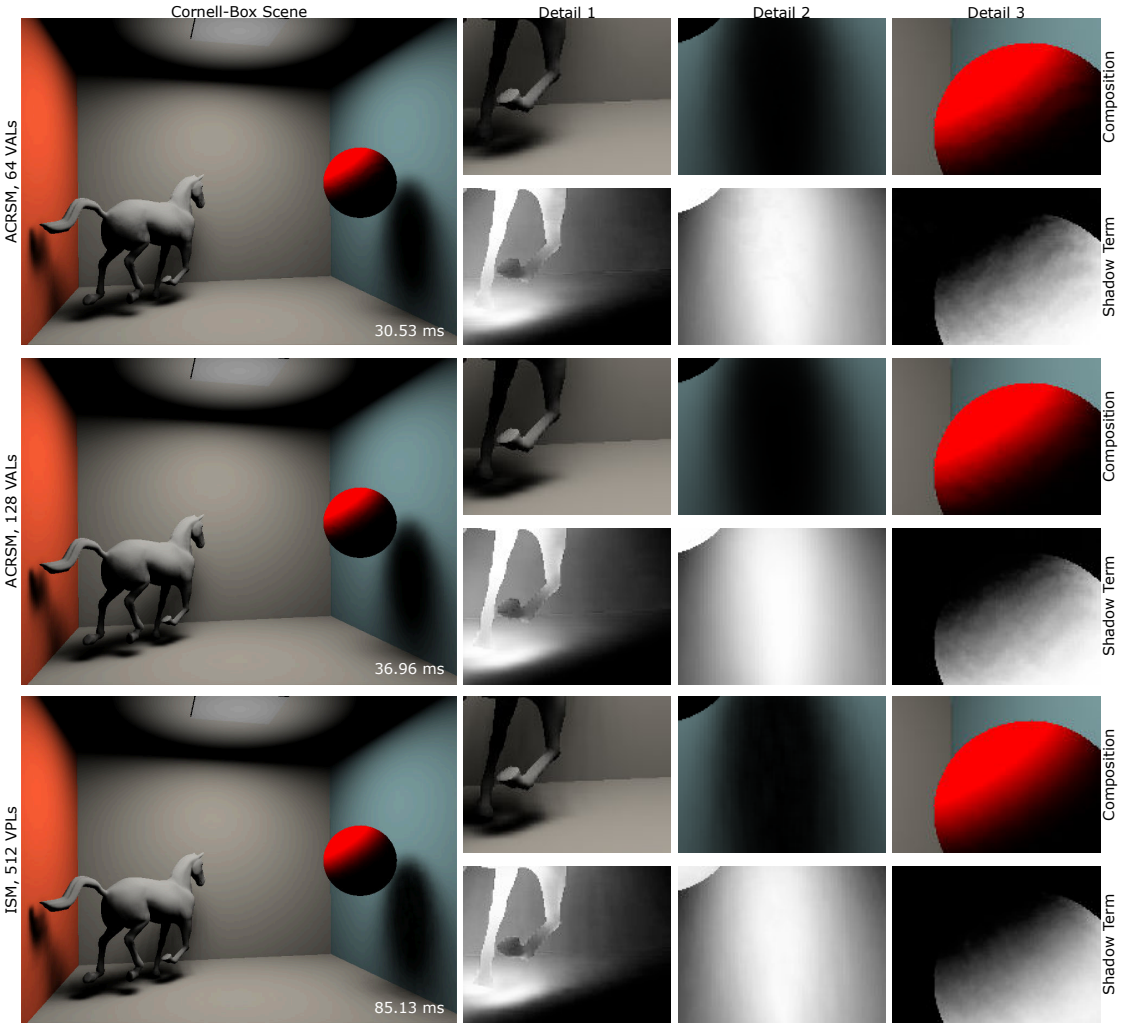


Figure 6.2: *In this comparison we show the same scene using different ISM sampling configurations. The two scenes in the top are rendered using our ACRSM method with 64 and 128 VAL clusters and percentage-closer soft-shadow sampling enabled. The bottom scene acts as a frame of reference using 512 ISMs with default shadow sampling. Comparing the final composited scenes (images on the left), we can see that our method is able to render perceptively comparable results to the reference ISM implementation using only a fraction of the originally required ISMs. The detail comparisons show the final composition and the shadow term (white = shadow, black = no shadow). These reveal that our method slightly overestimates shadow regions of finely detailed structures (*Detail 1*), but has no trouble providing smooth shadows for large, homogenous regions (*Detail 2*). We can define further problematic regions on areas with smooth geometric changes and their self-shadowing (*Detail 3*). This is one instance where the point-based nature of the ISMs is not able to faithfully reproduce the fine changes in the geometry and even our more advanced sampling technique is not able to achieve better results. However, in this case the more pronounced artifacts on the back of the sphere are not visible in the final composition, since the problematic areas are part of the unlit region where Lambert's law already negates any impact from the light sources. A second-bounce indirect illumination would probably make these artifacts more visible.*

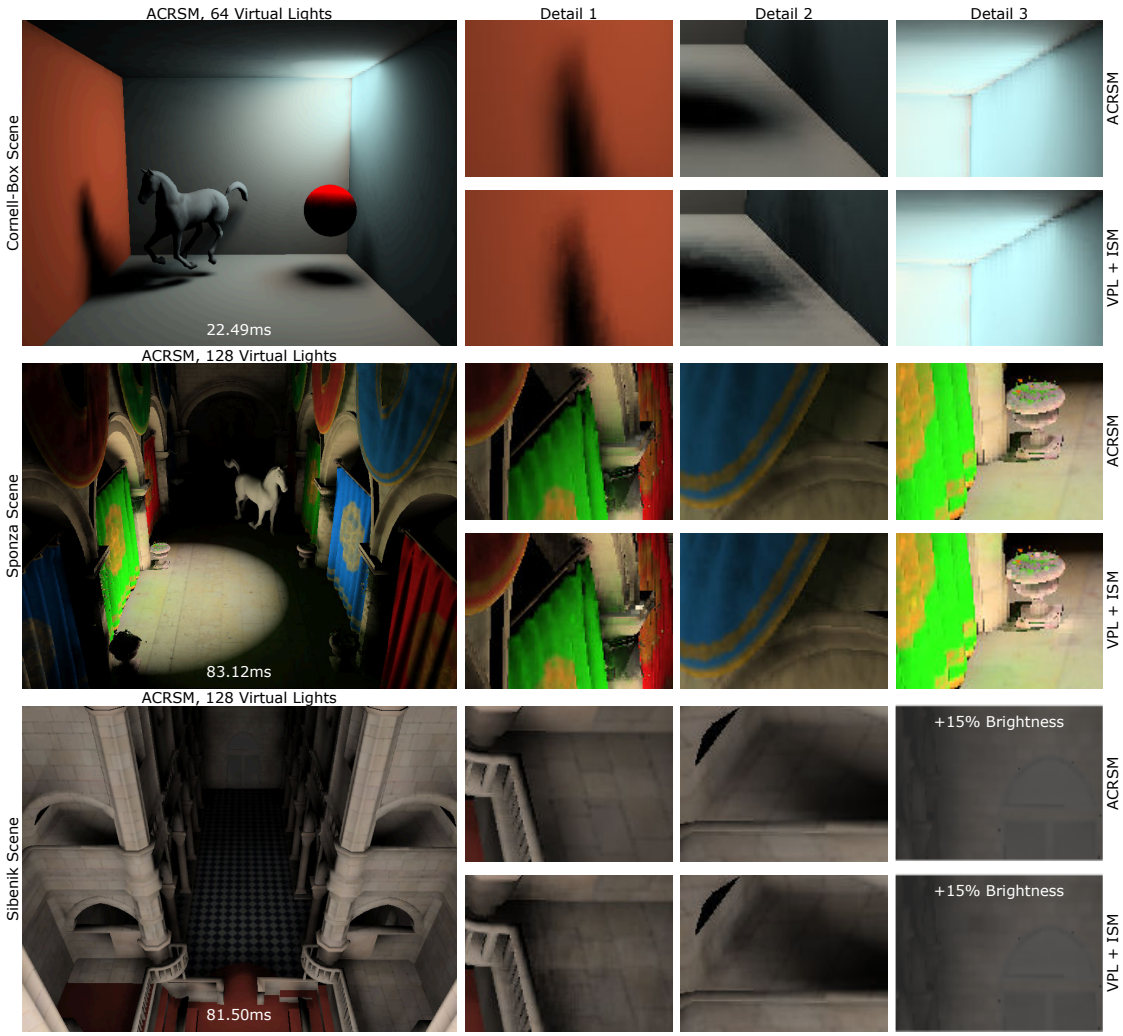


Figure 6.3: *Three different scenes rendered with our method and compared to a default ISM implementation using the same amount of virtual lights. Throughout all comparisons, our algorithm is able to achieve smooth soft shadows with even small amounts of VALs, while in the standard ISM implementation using the same amount of VPLs the image quality degrades noticeably: Shadows appear more aliased and blocky (compare* Detail 1 *and* Detail 2 *shots of the* Cornell-Box *and* Sibenik *scenes) and VPLs tend to produce more noticeable singularity sparks (i.e.,* Cornell-Box *and* Sponza *scenes* Detail 3*).*

6.2 ISM Generation

Another important observation to make is with regards to the ISM generation schemes which we outlined in Section 5.3.2. One important metric that has to be considered when generating ISMs is the coverage (i.e., how sparse or dense the ISMs are filled with point splats) and how this influences the shadows in the final composition. Hence, Figure 6.4 focuses on differences between static vs. dynamic scene representations, their advantages and disadvantages and how the Pull-Push method fits into both scenarios to help tackle the coverage problem.

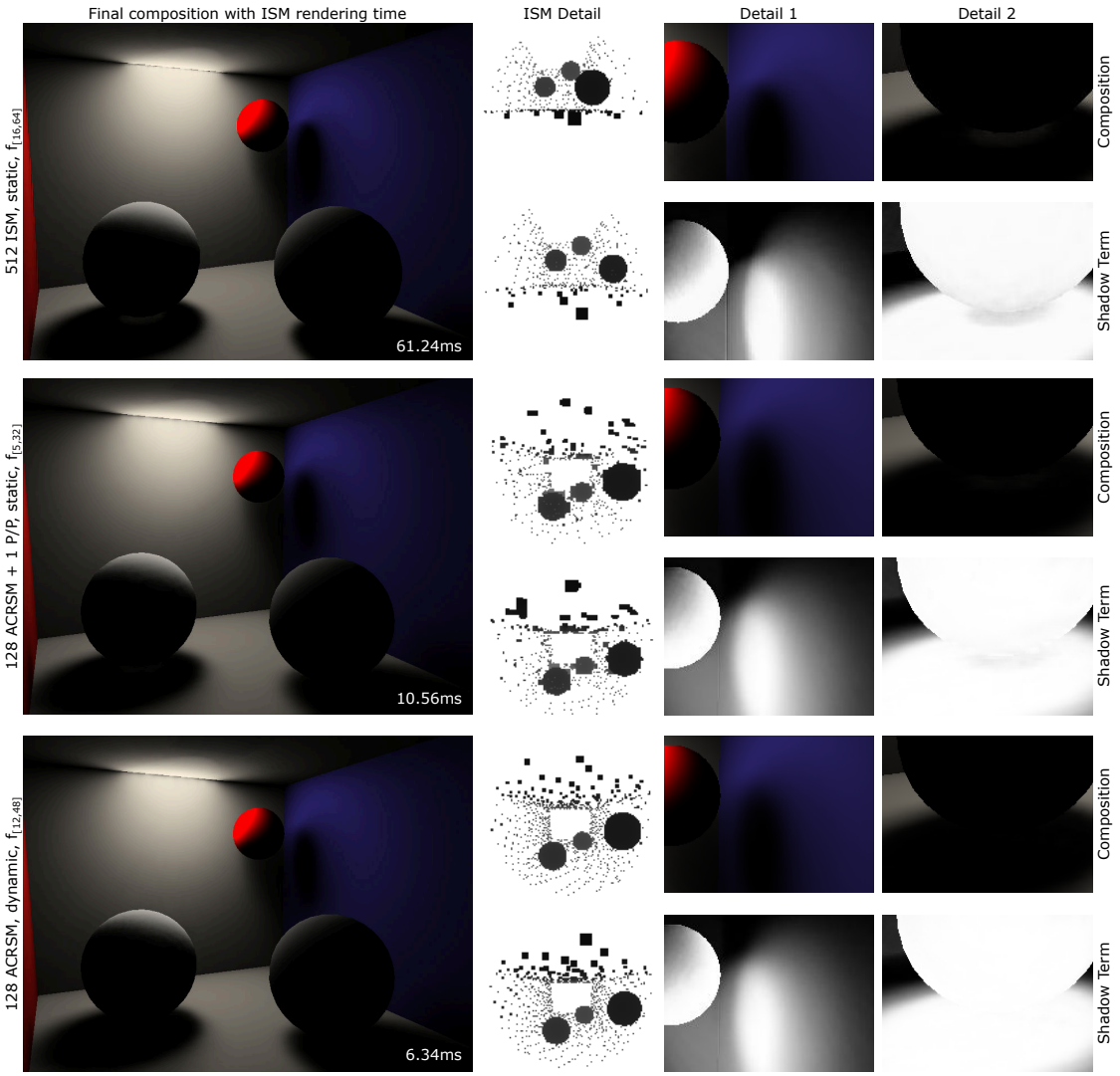


Figure 6.4: *Here we explore the visual differences between various ISM generation settings. The top row is a reference rendering produced using 512 standard ISMs and a static point generation scheme with tessellation factors* $f_{min} = 16$ *and* $f_{max} = 64$ *yielding approximately* 1 000 000 *point splats. The rows below employ our ACRSM method. In the middle, we use static point generation and tessellation factors* $f_{[5,32]}$ *in combination with a single iteration of pull/push. In the last row, we employ dynamic point generation with* $f_{[12,48]}$*. The* ISM Detail *column reveals sufficient coverage of the spherical objects in the scene for all configurations. However, the middle row due to activated pull/push generates more blocky spheres. Overall, the last row appears to provide the best mix of coverage and detail throughout the scene and also eliminates the slight light leakage visible in the* Shadow Term *of* Detail 2 *on the floor below the sphere.*

It is important to point out that there is a clear correlation between the tessellation factors (i.e., the number of points splats) and the amount of ISMs in use: The more ISMs that have to be rendered, the less coverage is achieved with the same number of point splats. Thus, larger amounts of ISMs require massively larger numbers of point splats to achieve sufficient coverage.

Since our algorithm needs fewer ISMs to produce comparable indirect shadow results, we also reap the benefit of requiring smaller point counts. Both these factors have a large influence on performance since tessellation factors as well as number of ISMs/VALs are reduced significantly, hence achieving higher performance. The performance difference for the ISM generation stage can be substantial, as reflected in Figure 6.4. From additional measurements in Table 6.1 and Figure 6.5 we can see that the results for dynamic tessellation are highly dependent on the number of meshes, and performance can therefore vary greatly depending on scene setup (refer to the SIBENIK and SPONZA results), whereas pull/push adds a largely static amount of time to the rendering pipeline, hence allowing more predictable changes in rendering times.

	method	**128 ACRSM**							**512 ISM**
	point generation	**static**			**dynamic**				**static**
	configuration	$f_{[5,32]}$	1 P/P	2 P/P	$f_{[5,32]}$	1 P/P	$f_{[12,48]}$	$f_{[16,64]}$	$f_{[16,64]}$
Scene	CORNELL B.	2.91	10.56	12.22	2.58	10.23	6.34	10.93	61.24
	SIBENIK	34.78	42.41	44.79	35.33	43.20	113.85	203.44	245.19
	SPONZA	51.56	59.65	61.69	54.73	62.88	185.22	347.93	945.09

Table 6.1: *Timings in milliseconds for the ISM rendering stage in different scenes for various configurations. The value $f_{[x,y]}$ defines the min- and max-tessellation factors in use. Consecutive columns with z* P/P *additionally employ z iterations of the* pull/push *algorithm using the previously defined tessellation factors. The splat factor is scene-dependent and set to $s_f = 1$ for the* CORNELL BOX *and* SIBENIK *scenes and $s_f = 0.3$ for* SPONZA.

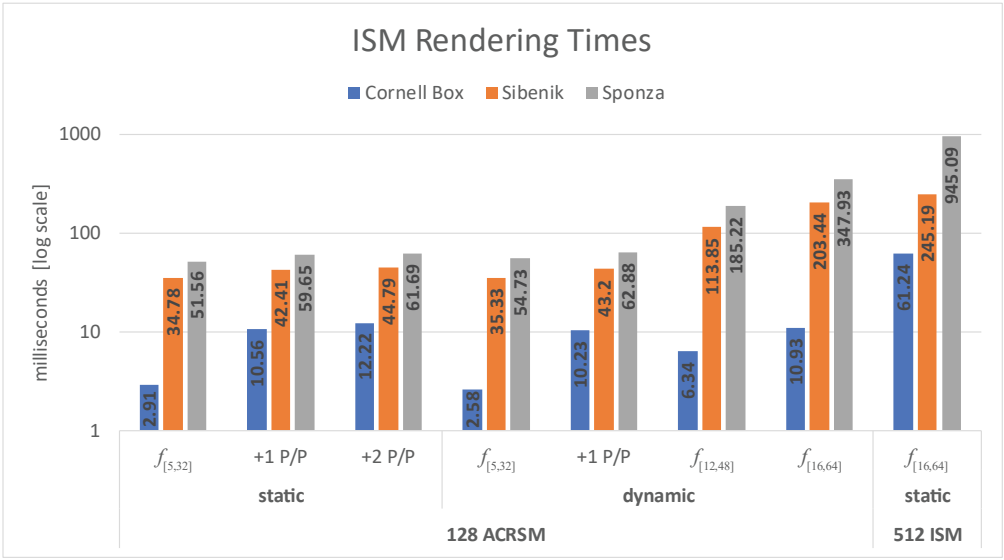
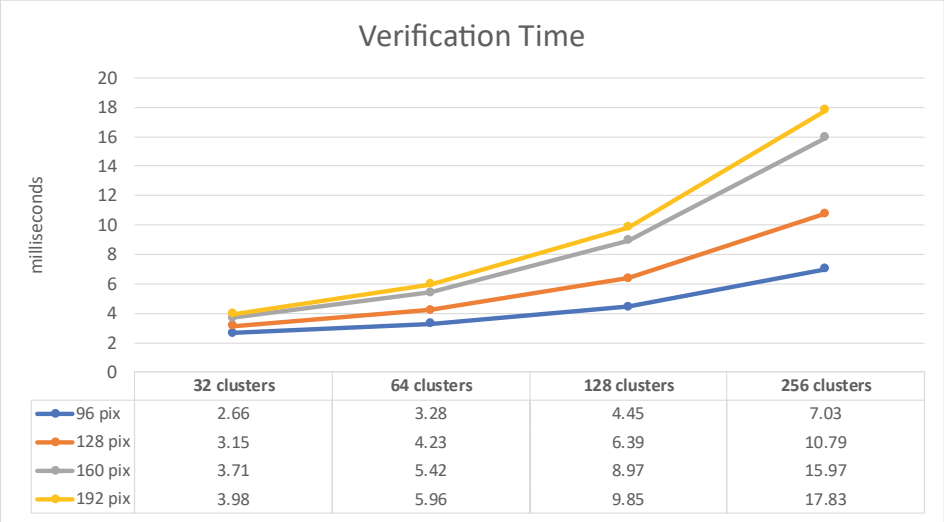


Figure 6.5: *Visualization of Table 6.1 on a logarithmic scale depicting ISM rendering time in milliseconds for the three different scenes under various tessellation configurations. The numbers in the bars denote the exact render times in ms.*

6.3 Adaptive Clustering

The main contribution of this thesis is the adaptive clustering process (refer to Chapter 4) that is meant to tackle the problem of cluster distribution in dynamic scenes (remember Figure 4.2 where a spotlight is moving from one wall to another over a clustering boundary). Timings in milliseconds for the evaluation procedure are given Figure 6.6 for different amounts of clusters and at varying cluster sizes. The cluster size has a direct influence on the timings since the verification region scales linearly with the cluster size, thus larger regions have to be evaluated.



	32 clusters	64 clusters	128 clusters	256 clusters
96 pix	2.66	3.28	4.45	7.03
128 pix	3.15	4.23	6.39	10.79
160 pix	3.71	5.42	8.97	15.97
192 pix	3.98	5.96	9.85	17.83

Figure 6.6: *Timings in milliseconds for our adaptive cluster evaluation process for 32, 64, 128 and 256 clusters at four different cluster sizes of 96, 128, 160 and 192 pixels.*

In the following paragraphs we outline the improvements we have been able to achieve using our adaptive approach when compared to non-adaptive clustering by means of time series comparisons. In Figure 6.7 a comparison of the problematic Cornell-Box scene as discussed previously is given, where a spotlight moves smoothly from the back wall to the right-side wall over the course of ten seconds. On the left side, the time series for *non-adaptive* clustering is shown, while the *adaptive* variant is shown right next to it. We also provide detail-comparison shots to highlight differences in shadow quality at the current point in time. Focusing on the clusters, we can see that their distribution remains more uniform with adaptive clustering enabled. The benefits of this can be seen especially in the 3^{rd} and 4^{th} row, where a smoother shadow term is but one of the positive side effects that can be gained. Having a better distribution automatically yields a softer shadow term since a more uniform sampling is achieved. Conversely, a bad distribution leads to more visible artifacts. Most of the other negative side effects that could be caused by the non-uniform distribution like VPL singularities and unexpected lighting contributions (refer again to Section 6.1.1) stemming from the bad distribution are mostly eliminated through the usage of VAL lighting, which is active in both scenarios.

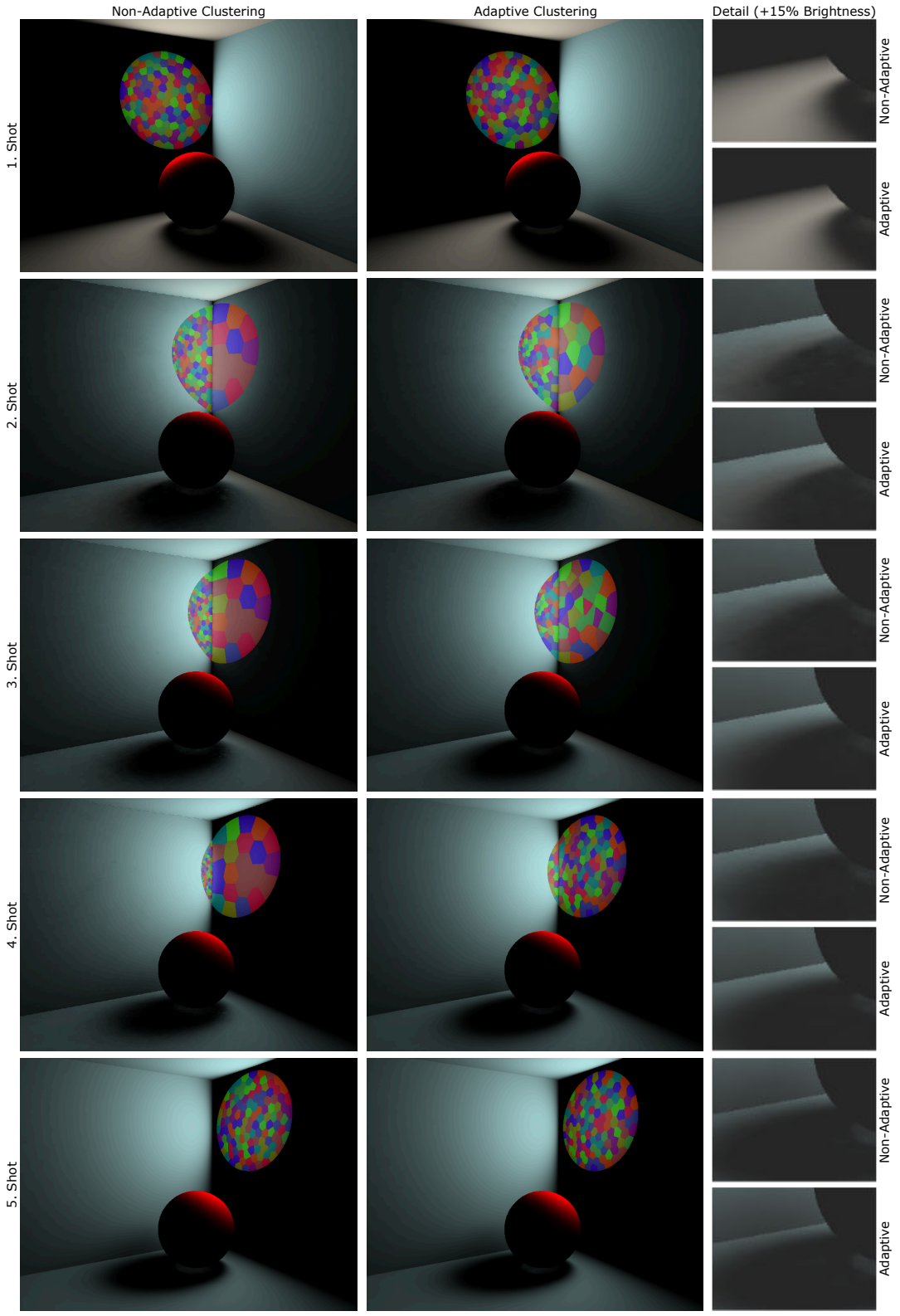


Figure 6.7: *A time series comparison for a moving light source comparing non-adaptive and adaptive clustering from top (beginning) to bottom (end). In this comparison 128 clusters are in use. Their size and distribution is visualized by the Voronoi-shaped colored cells.*

6.4 Performance

In this final section we give detailed performance measurements of our implementation measured in milliseconds for three test scenes at varying cluster counts. Our test system consists of an Intel XEON X3350 processor clocked at 2.66 GHz with an AMD Radeon R9 380 graphics card. Measurements are taken at a framebuffer resolution of 800×600 pixels. The various stages are timed using a high-performance counter provided by the `Qt` framework and are carried out through the usage of the `glFinish` operation.

Beginning with the simple Cornell Box scene (see Figure 6.8), we provide exact timings for the *cluster evaluation*, *cluster averaging*, *ISM rendering* and *indirect lighting* stages via the green, blue, yellow and orange stacks in the bar charts. The greyish stacks denote various other stages of the algorithm that are largely independent of the number of clusters: a) G-Buffer performance is largely dependent on scene complexity, while b) cluster initialization happens only sporadically throughout the algorithm and its impact on the performance is negligible. c) Cluster mapping depends on the cluster size, which is fixed to 128 pixels for these leading to static and minimal contribution to the frame timings. d) Direct lighting and tonemapping scale linearly with the rendering resolution and are therefore not affected by changes to the number of clusters in use by our algorithm.

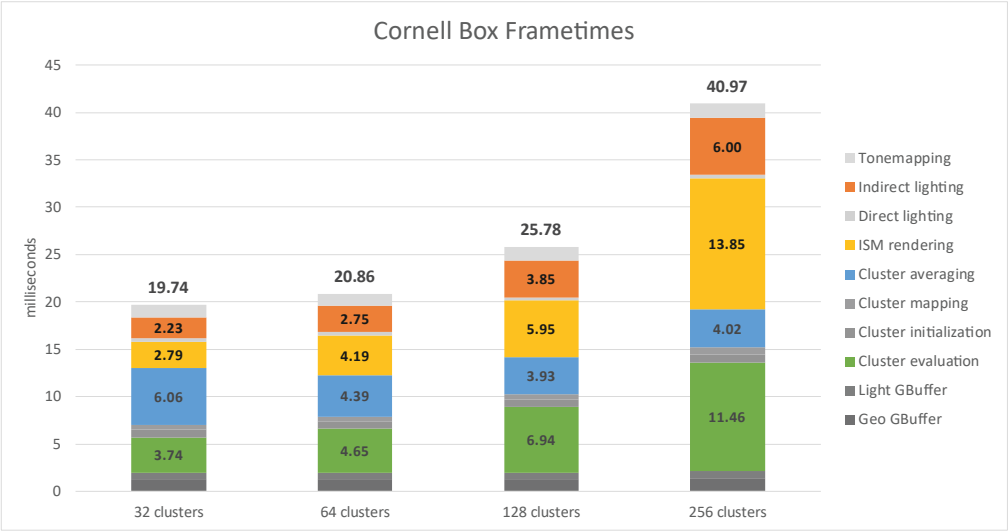


Figure 6.8: *Complete frame timings for the Cornell Box scene, rendered with 32, 64, 128 and 256 clusters. For this scene we use dynamic tessellation with factors* $f_{[10,32]}$ *and* $s_f = 1$.

Apart from the expected measuring inaccuracies, the values for the clustering verification stage match up with our earlier measurements as shown in Figure 6.6. Once again, it becomes clear that this process scales with the number of clusters in use.

The cluster averaging, on the other hand, is much more interesting as there is no clear pattern to discern at first. As mentioned earlier, the averaging process is an additive blending operation (i.e., an alternating read/write procedure) that puts a lot of stress on the memory subsystem of the graphics hardware. The reason why the averaging process is more costly for fewer clusters

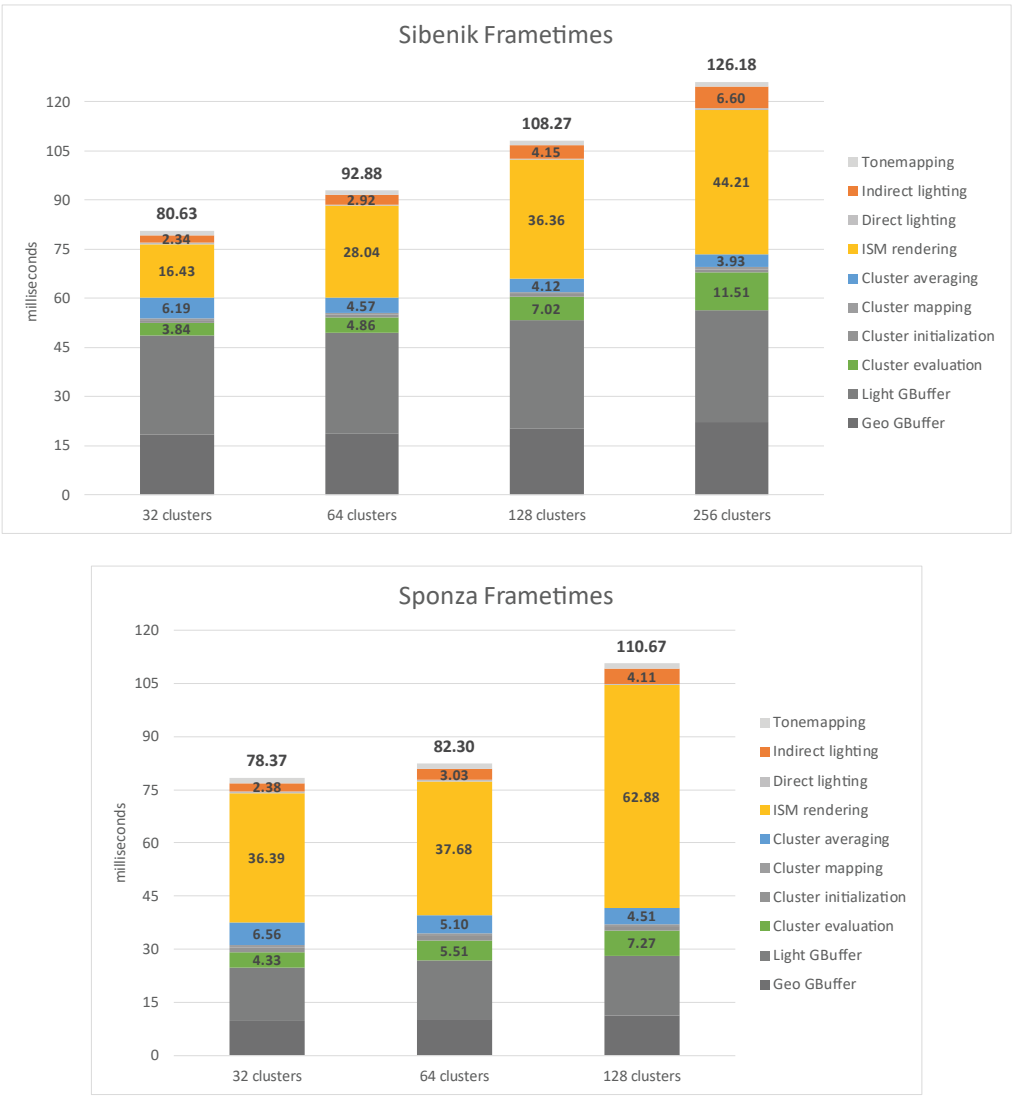


Figure 6.9: *Complete frame timings in milliseconds for the Sibenik scene (top) with $s_f = 1$ and the Sponza scene (bottom) with $s_f = 0.3$. Both scenes use dynamic tessellation with factors $f_{[5,32]}$.*

(32 and 64) is because in these cases, each individual cluster covers a larger area of the RSM, thus more individual pixels play a role in the averaging process of a single cluster, which in turn equals more read/write interlocks. At 128 clusters we appear to have hit some kind of sweet spot for our algorithm where the tradeoff between number of clusters and average size of a single cluster allows the operation to perform at an optimal pace. At 256 clusters the tide is beginning to turn again (although the difference appears to be within the margin of error for our measurement technique). Further measurements at 512 clusters (not visualized here) confirm that this is indeed the case and cluster averaging becomes more costly again.

ISM Rendering and indirect lighting also scale with the number of clusters. The former has a higher performance penalty starting to be visible at 256 clusters. We attribute this specific behavior to a combination of the tessellation and geometry shader stage. The latter does all the heavy lifting (i.e., paraboloid mapping, replicating vertices to be rendered into multiple ISMs) and it appears that our hardware is hitting some kind of limitation that causes a higher than expected increase in ISM rendering times. Indirect lighting, on the other hand, scales largely as expected and is also influenced by an additive blending operation (as described in Section 5.1.4). Note that these results may also be dependent on the type of hardware in use.

In Figure 6.9 (top) the results for the Sibenik scene are visualized. The pattern described previously emerges again. Obvious from the stacked bars is also that this scene is much more heavy during GBuffer creation, owing to its higher complexity thus requiring a higher number of draw calls. Notable differences can be seen in the ISM Rendering stage, which scales more linearly as in the Cornell Box scene and is also dependent on the scene complexity. Furthermore, the cluster averaging appears to come in at a lower measurement when comparing with the 256 cluster results from the previous figure, which might again be attributed to the margin of error in our measurement.

Results for the Sponza scene are given in Figure 6.9 (bottom). Similar to the Sibenik scene, this scene has a higher geometric complexity but requires fewer draw calls, thus the GBuffer creation is a bit faster when compared to the previous scene. This is, however, contrasted by the results of the ISM rendering stage, which is exorbitantly higher as when compared to the Sibenik scene and again the scaling is not really as expected. Once more, we attribute this to the tessellation and geometry shading stages due to the layout of the scene, which consists of

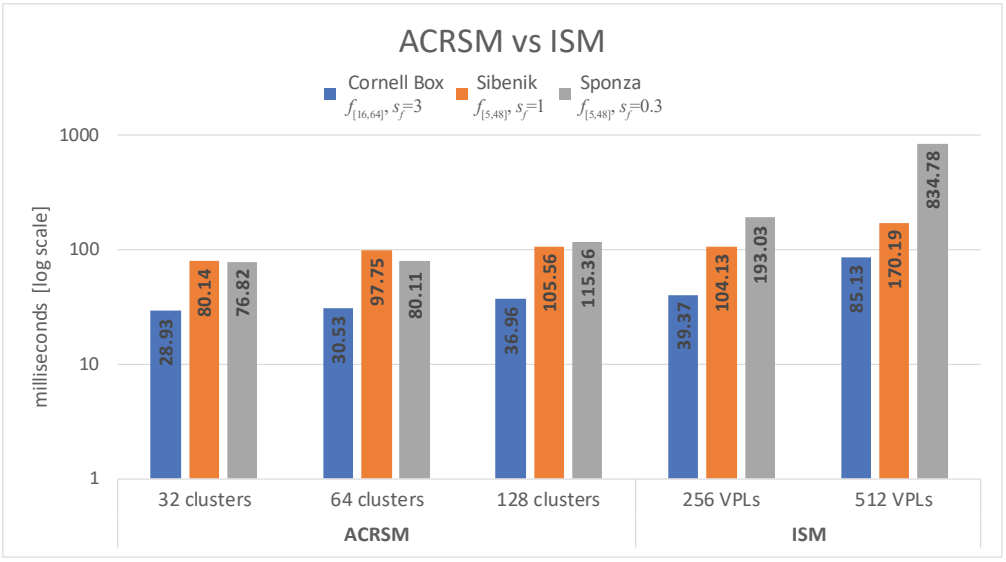


Figure 6.10: *Complete frame timings of ACRSM compared with standard ISMs at virtual light counts that produce qualitatively similar visual results (Fig. 6.2). For better comparability, both variants employ static tessellation .*

much larger triangle areas, thus requiring higher tessellation factors on average, putting additional stress on the knowingly weak AMD tessellator and the following geometry stage. We omit the 256 cluster result for this scene entirely as the ISM rendering stage alone takes up ~160 milliseconds, dwarfing the other results and thus making the remaining results confusing.

Finally, we provide a simulated comparison between our algorithm and the classic ISM approach. Since we are not able to fully replicate the classic ISM method with our approach due to the many differences with respect to how VPLs are seeded and updated, we estimate their results by cutting out the timings of the cluster verification, mapping and averaging steps. We only leave the cluster initialization intact, since this step is very similar to what the classic ISM approach does (i.e., seeding VPL positions and performing importance warping). Additionally, we change the indirect lighting computation and the shadow-lookup algorithm for the ISM variant to employ VPL-based illumination and simple shadow-map sampling (instead of VAL illumination and PCSS sampling as in the ACRSM case). These simulated results are given in Figure 6.10.

CHAPTER 7

Conclusion

In this thesis we have presented *Adaptively Clustered Reflective Shadow Maps* (ACRSM): An efficient real-time single-bounce diffuse global illumination method that is based on clusters which form Virtual Area Lights (VALs), a fast method to provide shadow information for them and an improved soft shadow-map sampling technique to achieve visually acceptable results from lower amounts of Imperfect Shadow Maps (ISMs). Our main contribution is a progressive VAL clustering algorithm that works in the Reflective Shadow Map's (RSM) image space. To maintain a balanced VAL distribution in dynamic scenes we have proposed a new verification procedure inspired by methods from optimization theory.

The result of our work is a real-time global-illumination rendering framework that extends state-of-the-art indirect illumination and shadowing techniques by a novel, adaptive VAL clustering approach (Chapter 4). While a simple k-Means based clustering of VALs is sufficient for static scenes, its usefulness for dynamic scenes proved to be strongly limited, because the convergence of the clusters is highly dependent on their initial seed positions. This means the clustering will converge towards a local optimum that has been defined at the start of the algorithm, and does therefore not account for changing scenes with moving light sources and objects.

To deal with these problems, we have extended the clustering pipeline by an extra *cluster evaluation* pass, which evaluates the VALs produced by the k-Means clustering and identifies degenerated clusters that became too large or too small for their local neighborhood. To that end, we have introduced a custom measure that verifies whether the geometric properties of a single cluster still contribute to a balanced cluster distribution within its neighborhood, or, if not, requires removing and re-seeding the cluster at a better suited location.

Chapter 5 elaborates on the technical implementation of this technique into a deferred rendering pipeline as is commonly found in today's graphics engines. The presented method renders dynamic scenes entirely on the GPU in real-time by employing the various shader stages available on modern graphics hardware.

The results presented in Chapter 6 show that our ACRSM is more efficient and produces higher-quality results compared to a standard ISM implementation. Using VALs instead of

Figure 7.1: *The Crytek-Sponza scene rendered with our algorithm using 256 VAL clusters.*

VPLs for illumination reduces singularity artifacts (i.e., light sparks) considerably at little to no extra cost in performance. We also require far smaller amounts of imperfect shadow maps to achieve perceptually similar or even better results with regards to indirect shadow quality. Since we distribute the point splats in an interleaved manner to the ISMs, requiring less ISMs has the additional benefit of gaining a higher point-splat coverage per ISM. This enables further performance gains because the algorithm does not require to perform pull/push operations on the ISMs in order to fill large gaps in them. A time series comparison shows that the adaptive clustering effectively maintains a balanced VAL distribution, thereby optimizing the indirect illumination quality at a constant virtual lights budget. Furthermore, it has a beneficial influence on the visual quality of the rendered GI image, due to the improved distribution of virtual area lights. While our adaptive clustering requires additional evaluation computation compared to a naive re-seeding approach, the amount of performance gained by requiring far less point splats greatly outweighs its costs.

Future Work

Despite the aforementioned contributions, there are still some limitations that could be improved upon in the future to increase the overall quality and possibly performance of global illumination rendering:

MULTIPLE BOUNCES Currently, our method is a radiosity solver limited to only a single bounce of indirect illumination. The approach could be easily extended to include a second light bounce. One way would be to replace the ISMs by RSMs, where each point splat stores additional information such as flux and orientation. This would result in *imperfect reflective shadow maps*

(IRSMs), which Ritschel et al. [39] already envisioned in their original work. Based on these IRSMs, a second set of virtual lights could then be seeded and used for another indirect illumination bounce. Alternatively, a screen-space method such as *Screen-Space Directional Occlusion* (SSDO) [40] could be employed to approximate a second illumination bounce in screen space only.

SPECULARITY Another possible extension of our GI implementation is the additional handling of specular light reflections. However, to faithfully reproduce high-frequency specular effects would again require much larger amounts of virtual lights. In some way, this would counteract our current intention to reduce the number of virtual lights while keeping image quality intact. A feasible approach could be to keep the current clustering and ISM generation to obtain an initial set of VALs and supplement each cluster with an additional set of VPLs based on a cluster's specularity using a rotated Poisson-disk for seeding within the bounds of the cluster. Highly glossy surfaces would gain larger amounts of additional VPLs than less glossy or diffuse reflectors. A cluster's ISM could then be shared with each VPL to incorporate visibility information similar to Dong et al. [11].

INDIRECT LIGHTING OPTIMIZATION For accumulation of the contributions of indirect light, more sophisticated methods than the rough tile-based interleaved sampling used by our current implementation could be considered. There exist methods that are designed to reduce overdraw by either clustering pixels into much smaller tiles and performing lighting computations in the compute shader [17, 33] or by splatting spherical objects into the G-Buffer, adapting their shape based on the reflection characteristics [10]. These extensions would be especially helpful when introducing specular effects as mentioned above, since they could handle the higher number of virtual lights more efficiently.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz. Convolution Shadow Maps. In J. Kautz and S. Pattanaik, editors, *Rendering Techniques*. The Eurographics Association, 2007. ISBN 978-3-905673-52-4. 20, 21, 33

[2] T. Annen, Z. Dong, T. Mertens, P. Bekaert, H. Seidel, and J. Kautz. Real-time, all-frequency shadows in dynamic scenes. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 34:1–34:8, New York, NY, USA, 2008. ACM. ISBN 978-1-4503-0112-1. 31, 33, 39

[3] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991. ISSN 0360-0300. 42

[4] T. Barák, J. Bittner, and V. Havran. Temporally coherent adaptive sampling for imperfect shadow maps. In *Proceedings of the Eurographics Symposium on Rendering*, EGSR '13, pages 87–96, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association. 28, 29, 39

[5] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 192–198. ACM, 1977. 10

[6] S. Brabec, T. Annen, and H. Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Proc. of Computer Graphics International*, pages 397–408, 2002. 26, 30, 65

[7] P. Clarberg, W. Jarosz, T. Akenine-Möller, and H. Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Transactions on Graphics (TOG)*, 24(3):1166–1175, 2005. 45

[8] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, 1(1):7–24, 1982. 11

[9] C. Dachsbacher and M. Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231. ACM, 2005. 24

[10] C. Dachsbacher and M. Stamminger. Splatting indirect illumination. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 93–100, New York, NY, USA, 2006. ACM. ISBN 1-59593-295-X. 85

[11] Z. Dong, T. Grosch, T. Ritschel, J. Kautz, and H. Seidel. Real-time indirect illumination with clustered visibility. In *Vision, Modeling, and Visualization Workshop 2009*, 2009. 17, 31, 32, 33, 38, 39, 40, 41, 85

[12] D. Dunbar and G. Humphreys. A spatial data structure for fast poisson-disk sample generation. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 503–508, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. 68

[13] E. Eisemann, U. Assarsson, M. Schwarz, and M. Wimmer. Casting shadows in real-time. In *ACM SIGGRAPH ASIA 2009 Courses*, SIGGRAPH ASIA '09, New York, NY, USA, 2009. ACM. 17

[14] R. Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, page 35. ACM, 2005. 18, 31, 33, 39, 40

[15] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, 18(3):213–222, Jan. 1984. ISSN 0097-8930. 13, 15

[16] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, Dec. 1964. ISSN 0001-0782. 23, 43, 60

[17] T. Harada, J. McKee, and J. C. Yang. Forward+: Bringing Deferred Lighting to the Next Level. In C. Andujar and E. Puppo, editors, *Eurographics 2012 - Short Papers*. The Eurographics Association, 2012. 57, 85

[18] M. Hašan, J. Křivánek, B. Walter, and K. Bala. Virtual spherical lights for many-light rendering of glossy scenes. *ACM Transactions on Graphics (TOG)*, 28(5):143, 2009. 14, 38

[19] P. S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *SIGGRAPH Comput. Graph.*, 24(4):145–154, Sept. 1990. ISSN 0097-8930. 14

[20] H. W. Jensen. *Realistic image synthesis using photon mapping*, volume 364. Ak Peters Natick, 2001. 16

[21] J. T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM. ISBN 0-89791-196-2. 3, 12, 16

[22] B. Karis. Real shading in unreal engine 4. In *ACM SIGGRAPH 2008 Presentation*, 2013. 11

[23] A. Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997. 2, 22, 37, 38, 44

[24] A. Keller and W. Heidrich. Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276. Springer-Verlag, 2001. 27

[25] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, and T. Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering*, pages 277–286, 2007. 17, 25, 26, 27, 37, 44, 55

[26] J. H. Lambert. *Photometria*. 1760. 10

[27] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28 (2):129–137, 1982. 42

[28] C. Luksch, R. F. Tobler, R. Habel, M. Schwärzler, and M. Wimmer. Fast light-map computation with virtual polygon lights. In *Proceedings of ACM Symposium on Interactive 3D Graphics and Games 2013*, pages 87–94. ACM, Mar. 2013. ISBN 978-1-4503-1956-0. 38, 41

[29] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967. 41

[30] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient point-based rendering using image reconstruction. In *Symposium on Point-Based Graphics*, pages 101–108, 2007. 30

[31] M. Mittring. A bit more deferred–cryengine 3. In *Triangle Game Conference*, volume 4, 2009. 52

[32] F. E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *Appl. Opt.*, 4(7):767–775, Jul 1965. 7

[33] O. Olsson, M. Billeter, and U. Assarsson. Clustered deferred and forward shading. In *HPG '12: Proceedings of the Conference on High Performance Graphics 2012*, 2012. 57, 85

[34] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975. 10

[35] R. Preiner. Real-time global illumination in point clouds. In *Proceedings of the 14th Central European Seminar on Computer Graphics*, May 2010. 28

[36] R. Prutkin, A. Kaplanyan, and C. Dachsbacher. Reflective shadow map clustering for real-time global illumination. In *Eurographics 2012-Short Papers*, pages 9–12. The Eurographics Association, 2012. 17, 33, 34, 35, 38, 39, 40, 41

[37] W. Reeves, D. Salesin, and R. Cook. Rendering antialiased shadows with depth maps. *ACM SIGGRAPH Computer Graphics*, 21(4):283–291, 1987. ISSN 0097-8930. 19

[38] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics (TOG)*, 21(3):267–276, 2002. 70

[39] T. Ritschel, T. Grosch, M. H. Kim, H. P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph. (Proc. of SIGGRAPH ASIA 2008)*, 27(5), 2008. 17, 28, 29, 33, 37, 38, 39, 40, 44, 65, 66, 85

[40] T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, pages 75–82, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-429-4. 85

[41] T. Ritschel, E. Eisemann, I. Ha, J. Kim, and H. Seidel. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. In *Computer Graphics Forum*. Wiley Online Library, 2011. 17, 28, 33, 34, 37, 38, 39, 40, 45, 65

[42] C. Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246, 1994. 11

[43] N. Schulz. The rendering technology of ryse. In *GDC 2014 Session*, 2014. 11

[44] E. Tabellion and A. Lamorlette. An approximate global illumination system for computer generated films. *ACM Trans. Graph.*, 23(3):469–476, Aug. 2004. ISSN 0730-0301. 2, 17, 28

[45] E. Veach and L. J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 16

[46] J. R. Wallace, K. A. Elmquist, and E. A. Haines. A ray tracing algorithm for progressive radiosity. *SIGGRAPH Comput. Graph.*, 23(3):315–324, July 1989. ISSN 0097-8930. 35, 40

[47] G. J. Ward. Measuring and modeling anisotropic reflection. *SIGGRAPH Comput. Graph.*, 26(2):265–272, July 1992. ISSN 0097-8930. 11

[48] C. Weinzierl-Heigl. Shadows in real-time applications. Bachelor's Thesis, 2011. 17

[49] C. Weinzierl-Heigl. Efficient val-based real-time global illumination. In *Proceedings of the 17th Central European Seminar on Computer Graphics*, CESCG '13, pages 17–24, Favoritenstraße 9-11/186, 1040 Vienna, Austria, Apr 2013. Vienna University of Technology. 3

[50] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980. 16

[51] L. Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 270–274, New York, NY, USA, 1978. ACM. 18, 28