FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Using Perception-Based Filtering to Hide Shadow Artifacts

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Felix Kreuzer, BSc
Matrikelnummer 0827433

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Univ.Ass. Dipl.-Ing. Michael Hecher

Wien, 24. August 2017

Felix Kreuzer	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Using Perception-Based Filtering to Hide Shadow Artifacts

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media Informatics and Visual Computing

by

Felix Kreuzer, BSc
Registration Number 0827433

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Univ.Ass. Dipl.-Ing. Michael Hecher

Vienna, 24th August, 2017

Felix Kreuzer	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Felix Kreuzer, BSc
Rasumofskygasse 12/8, 1030 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.

Wien, 24. August 2017

Felix Kreuzer

Acknowledgements

I would like to thank my friends and family for their love and support. I would also like to thank my supervisors Michael Hecher and Michael Wimmer for their support. And thanks to all the people who took part in the user study.

And special thanks to Jan from craftsmanspace.com for allowing me to use one of his models for rendering the benchmark results.

Kurzfassung

Schatten helfen dabei, räumliche Zusammenhänge zu begreifen, und sind aus modernen Echtzeitvisualisierungen nicht mehr wegzudenken. In der Echtzeitgrafik werden häufig Shadow-Mapping Verfahren in Kombination mit Tiefpassfiltern verwendet um visuell ansprechende Schatten zu erzeugen. Der weichzeichnende Effekt dieser Filter hat den Vorteil, dass Fehler, die bei beim Abtasten der Szenengeometrie entstehen, versteckt werden können, was wiederum die visuelle Qualität der erzeugten Schatten erhöht.

Ziel dieser Arbeit ist es, diese Filter auszunutzen, um eine Funktion aus dem Radius des Weichzeichnungsfilters und der Abtastrate der Shadow Map herzuleiten, die es ermöglicht, Rechenleistung einzusparen, während der optische Eindruck der Schatten erhalten bleibt. Im Verlauf dieser Arbeit werden wir ergründen, wie diese Fehler entstehen und wie man sie vermeiden kann. Eine Studie hilft uns dabei, ein optimales Verhältnis zwischen der Abtastfrequenz der Shadow Map und dem Radius des Filters zu finden. Aus den Ergebnissen der Studie leiten wir dann eine Formel ab und entwickeln einen Algorithmus, mit dem sich bestehende Shadow-Mapping Algorithmen erweitern lassen können. Anhand der Implementierung eines Prototyps können wir zeigen, dass sich mit unserer Methode in vielen Fällen die Sampling Frequenz reduzieren lässt und Rechenleistung eingespart werden kann.

Abstract

Shadows are an indispensable aid for understanding spatial relations of objects in natural scenes, which is why they are very important for real-time rendering applications. Combining filtering techniques with shadow mapping is a common tool to simulate visually pleasing shadows in interactive applications. A positive effect of such approaches is that filtering blurs aliasing artifacts caused by sampling the discretized geometric data stored in the shadow map, thereby improving the visual quality of the shadow.

The goal of this thesis is to exploit common filtering algorithms in order to find a function of blur radius and shadow-map sampling frequency, which allows for optimized computational performance while mostly preserving the visual quality of the shadow. In the course of this work, we investigate how shadow artifacts arise and how to hide them. We set up and execute a user study to find the optimal relation between the shadow-map sampling frequency and the filter radius. From the results of the user study, we derive a formula and develop an algorithm that can be incorporated into existing shadow-mapping algorithms. We evaluate our results by applying the algorithm to a custom-made rendering framework and observe an increase in processing speeds.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Shadows are an indispensable aid for understanding spatial relations of objects in natural scenes. While we are used to them in nature, without shadows it would be hard to infer an object's position in a virtual scene projected onto an image. Figure 1.1 illustrates this by showing the same scene rendered with different shadow features. Our ability to interpret spatial relations and sometimes shapes from them also helps artists to transport information and aesthetics. Hence, shadows are crucial for realistic image synthesis.

"Where there is light, there must be shadow ..." [Mur12]

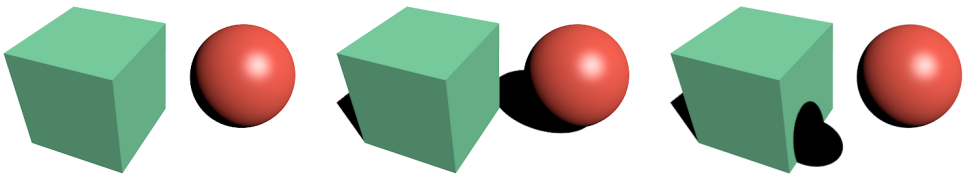


Figure 1.1: *Shadows help us infer spatial relations between objects. While shading helps us understand what he objects' surfaces look like (e.g., that the circular shape is a sphere), the shadows indicate the objects' elevations relative to the other surfaces.*

The concept of shadows is very simple to grasp since it is intuitively just the absence of light. Despite their simple nature, they still pose a challenging problem in real-time rendering applications.

In computer graphics we generally distinguish between simulating physically correct light transport and emulating the visual phenomena caused by light. If we would simulate light rays, shadows would appear automatically due to the lack of light rays reaching a surface. Unfortunately, light transport simulations (the interested reader might take a look at Pharr's excellent book on *physically based rendering* [PH04]) are very complex and infeasible to compute in real-time.

When we emulate phenomena caused by light in real-time rendering, we distinguish between *shading* and *shadowing*. Shading techniques like the widely employed *"Blinn-Phong shading" [Bli77]* basically determine a surface's color based on the its orientation in relation to viewer and light source. This means that light is not treated as rays but rather as an environmental feature affecting the color of each visible surface. Since each surface is treated independently, shading approaches do not offer capabilities to detect occlusions of light by other surfaces. In order to emulate shadows, shadowing calculations have to be performed separately.

Attempts at emulating plausible shadows in computer graphics have kept many researchers busy over the past decades and are still a challenging research field today. In the next paragraphs, we give a short historic introduction to this topic and dig deeper in the upcoming chapters, as real-time shadowing is in the focus of this thesis.

Shadow rendering dates back to 1968, when Appel et. al. [App68] created an algorithm which used ray casting to compute hard shadows. This algorithm was aimed at working with digital plotters and used plus signs ("+") to draw shadows. Two years later, Bouknight and Kelley [BK70] introduced a rasterization-based algorithm to draw scenes on CRT displays. This approach was already very similar to modern *shadow mapping* algorithms as it combined rasterizing operations for the light source and viewer in order to evaluate whether a polygon lies in shadow or not.

20 years ago, when real-time 3D applications started to gain popularity (especially through computer games), developers and researchers came up with simple solutions which were not strongly focused on realism, but were aimed at assisting the user in spatial orientation. One of the earliest wide-spread approaches were *approximate shadows* (often called *blob shadows*, see Figure 1.2a), where the shadows of dynamic objects were approximated by very simple geometry (e.g., a circle drawn on the surface beneath the player). Other applications used *projected shadows* [Bli88], which projected occluder geometry from the light source onto the receiving plane, thus covering the shadowed area. However, the projective shadow approach is not capable of computing self shadowing and was – due to the geometric processing routines involved and the lack of dedicated hardware to compute them efficiently at the time – only practical for scenes with very few and large receivers. Also, it potentially produces wrong shadows when objects lie behind the light source or behind the receiving surface.

(a) *Super Mario 64 (1996)* ***(b)*** *Doom 3 (2004)*

(c) *Grand Theft Auto V (2013)*

Figure 1.2: *Evolution of shadows in the last two decades. 20 years ago shadows in real-time rendering were primarily focused on spatial cues like* blob Shadows *in "Super Mario 64" (a). Only a few years later they evolved to become much more complex. "Doom 3" (b) employed* stencil Shadows *to feature shadows accurately representing object silhouettes. Recently* soft shadows *as in "Grand Theft Auto V" (c) are employed more frequently with the focus on adding realism to shadows.*

(a) *Low resolution* ***(b)*** *High resolution* ***(c)*** *Low resolution, filtered*

Figure 1.3: *Side-by-side comparison of the same shadow silhouette rendered with low and high shadow map resolution. The blocky artifacts (a) can be mitigated by using a higher resolution (b) or proper filtering (c).*

With the rise of more powerful graphics processing hardware, more sophisticated approaches based on *shadow volumes* [Cro77](sometimes referred to as *stencil shadows*, see Figure 1.2b) and *shadow mapping* [Wil78] were developed. In this thesis, we are going to cover *shadow volumes* briefly in Chapter 2 and focus on the more wide-spread and versatile algorithms based on shadow mapping. We start by introducing the method and our contributions to it.

1.2 Problem Definition

Currently, the most widely used techniques for computing shadows in interactive applications are based on *shadow mapping* [Wil78], which is intended to provide a reasonably good trade-off between computational performance and realism. The basic principle behind shadow mapping is to render the scene from the point of view of the light source and save the distance a light ray travels before it hits a surface in a buffer, called the *shadow map*. Later in the rendering process, this information is used to determine whether a point on a surface seen by the viewer was ever hit by a light source or lies in shadow. Although the size of a shadow map is limited, proper filtering techniques compensate lack of detail to some degree and allow softening of shadow silhouettes which further support realistic shadow appearance. The improvement in realism simply occurs based on the fact that in nature no light sources with an infinitesimally small area exist, hence there are no point light sources and no strictly hard shadow silhouettes.

All shadow-mapping approaches rely on a proper selection of the underlying shadow-map resolution. If the resolution is chosen too low, staircase artifacts appear. Choosing a high resolution prevents such artifacts, but also increases the computational effort. Fortunately, proper filtering can hide artifacts with the downside of reducing silhouette detail. Figure 1.3 illustrates the visual impact of increasing the shadow-map resolution versus a growing filter radius.

In general terms, a good shadow map is one that has a small resolution, while being perceived as artifact-free by the user. We want to exploit the low-pass filtering property of soft shadow outlines to introduce a function that allows us to compute the minimal shadow-map size that produces artifact-free results.

1.3 Contributions

In this thesis we want to find out which shadow-map resolution is sufficient to create perceptually plausible shadows in real-time rendering. In order to reach our goal, we investigate common shadow-map filtering algorithms and make the following contributions to this field of research:

- We investigate the relatively complex problem of artifacts generated by arbitrarily aligned shadow maps in soft-shadow algorithms and break down the huge parameter space, which can be hardly investigated in a user study, into a simplified version.
- We design a perceptual user study that allows us to analyze the impact of under-sampling artifacts on shadow perception.
- We develop a novel approach to dynamically adjusting shadow-map sizes for real-time soft shadowing algorithms. By reducing the number of depth samples in a shadow-map, we can increase performance in shadow-map generation since there are less fragments to process and fewer texture lookups. This also improves cache efficiency because shadow samples are tightly packed and redundant samples are being avoided.
- Our generic method is adaptable and can be applied to several existing soft shadow-mapping algorithms.

1.4 Overview

The aim of this thesis is to investigate and optimize existing shadow-map filtering algorithms. We therefore will go on and discuss literature related to real-time shadowing in Chapter 2 which serves as the foundation for our research.

We continue with an introduction to real-time shadow rendering, with the focus on shadow-mapping techniques and soft shadow extensions related to our work. While only being a small subset of the existing research in this field, Chapter 3 supplies the theoretical foundations which are necessary to understand our methods.

In Chapter 4 we will introduce our own research by investigating the behavior of shadow mapping with respect to visual artifacts. Therefore, we will analyze the impact of shadow-map resolution and filter sizes on the perceived visual quality.

Since defining visual quality is a perceptual problem, we argue for setting up an empirical experiment based on a user study. We will use this experiment to find out which shadow-mapping configurations successfully hide visual artifacts from the participants. Therefore, we will discuss the relevant variables involved in order to set our user study up and describe how we evaluated the user study results.

We continue with discussing the study results in Chapter 5. They allow us to formulate a generic method to estimate shadow-map resolutions which perceptually hide artifacts. Afterwards, we provide a detailed description on the algorithmic approach used in our implementation and provide results and benchmarks.

We then conclude by summarizing the major findings and drawbacks in this thesis and present an outlook on possible future research directions.

CHAPTER 2

Related Work

In this chapter we are going to take a look at literature dedicated to real-time shadow rendering. The literature we present is quite extensive and provides an overview of research done in this field. Subsequently, Chapter 3 will explain the most relevant subset of theoretical foundations needed for our research.

2.1 Shadow Mapping

A vast amount of publications describing real-time shadow algorithms have been published in the last years, especially during the last two decades.

Most of these algorithms are extensions to *shadow mapping* which was first introduced by Williams [Wil78] in 1978. They all follow the same principles (already outlined in Section 1.2) of projecting the light-source's field-of-view onto a discretized texture. This texture (i.e. the *shadow map*) is then used to determine whether a point on a surface is illuminated by the light source or not by comparing the distances from both points-of-view (we refer to this as the *shadow-test*).

Nowadays a variety of filter based extensions to the traditional shadow mapping algorithm exist. Not only are they aimed at softening the rough and jagged silhouettes produced by the original approach, they also increase realistic look of shadows by emulating a *penumbra* region (where the light is partially visible). The added realism is due to the fact that in nature nearly all shadows have soft boundaries because there is no such thing as a point light source. In the following paragraphs we will name popular extensions and briefly address their porpuse. We will discuss details about these approaches in the next chapter.

Percentage Closer Filtering (PCF) [RSC87] addresses the problem of anti-aliasing in shadow maps which we saw in Figure 1.3(a). The goal of PCF is to get a smooth transition on the shadows silhouette by blurring the binary result of the shadow test.

However, traditional shadow maps contain depth information, hence pre-filtering cannot be achieved directly. The solution is a screen space averaging approach.

Variance Shadow Maps [DL06] approximate the depth values by storing mean and variance of the depth distribution. Instead of averaging multiple samples like in PCF, the probability of a fragment being lit is calculated by approximating the depth's distribution through the moments. Storing mean and variance of the depth distribution instead of the actual depth values allows pre-filtering of the shadow map.

Convolution Shadow Maps [AMB+07] use Fourier expansion to store and reconstruct depth values. This approach allows shadow maps to be pre-filtered but requires a lot of memory and expensive memory transfers to retrieve the Fourier coefficients.

Exponential Shadow Maps [AMS+08] adopt an exponential function to approximate the shadow test. The main benefits are the possibility to pre-filter and cheap memory- and computational costs.

2.2 Soft Shadows

Some algorithms can further increase the physical plausibility by trying to take the distance from the light source to the occluder into account thereby generating variable penumbra sizes. These variable sized shadows are often referred to as *Soft Shadows*.

*Percentage Closer Soft Shadowing (*PCSS*)* [Fer05] extends the capabilities of PCF by evaluating the filter radius for each fragment based on the distance from the shadow occluder to the receiver. This approach features a more plausible penumbra behavior in regions where occluder and receiver merge (*contact hardening*).

Schwärzler et. al. [SMSW12] compute Soft Shadows by using use multiple shadow maps distributed over the area of the light source. They employ a custom strategy to find important sampling points and blend the sample together. While their method is very accurate it is far more costly than less accurate methods such as PCSS.

Several papers [GBP06], [GBP07], [AHL+06], [ASK06], [SS07] have been published, which propose variants of a technique called *backprojection*. The idea is to render a discretized representation of the scene from the light source's view and project the resulting texels into world space. When the scene is drawn from the viewer's point of view the corresponding texel gets reprojected onto the light source where the amount of occlusion can be estimated. While these approaches potentially produce more accurate results than PCSS they are prone to artifacts due to overlapping occluders, objects being too close to the light source or when the penumbra is extremely large. In addition to that, it can become very costly in terms of computational effort when a huge number of texels have to be reprojected.

In *Penumbra Maps* [WH03], in addition to a shadow map a so-called *Penumbra Map* is generated by analyzing the objects silhouettes from the position of the light source. This allows a penumbra region to be estimated during the shading pass.

2.3 Shadow Volumes

The approaches presented in the preceding paragraphs were image based. In contrast, Shadow Volumes [Cro77] rely on a geometric approach for solving the shadowing problem. Instead of transforming the scene into a shadow map, Shadow Volume algorithms describe shadows by attaching new geometric elements to the scene. These algorithms basically extrude geometry, which makes them ineffective with increasing scene complexity under dynamic lighting conditions. Since Shadow Volumes are geometric entities they produce very crisp silhouettes. Penumbra Wedges [AAM03], [FBP06] offer support for more realistic looking shadows by providing penumbra effects.

2.4 Comparisons and State of the Art Reports

Hecher et al. [HBM+14] present a comparison of some of these algorithms using a comprehensive perceptual study. They have come to the conclusion that Percentage Closer Soft Shadows provide a sufficient amount of realism for most users and does not perform significantly worse among experts.

For further reading the reader is referred to the excellent book on real-time shadows by Eisemann et. al. [ESAW11]. It covers many aspect of real-time shadow computation and extensively analyzes the shadow mapping algorithms and its various extensions.

Insight on the history and evolution of shadowing algorithms can be found in the overview by Woo et. al. [WPF90] and in the outdated but more recent report by Hasenfratz et. al. [HLHS03] which focuses on real-time shadowing.

2.5 Summary

In this thesis we will focus on PCF and PCSS as representative examples of filter based methods, but our findings can potentially be applied to any of the filtering techniques named above. We chose PCF based shadow map filtering approaches because they are widely used in modern applications, provide good quality, are very robust and are fairly easy to implement. Ashu Rege, who is currently vice president of engineering at NVIDIA's computer vision department, stated back in his 2004 whitepaper on *Shadow Considerations* [Reg04] that, when comparing shadow volumes with shadow mapping algorithms: *"Long term, however, we expect shadow maps to be more widely used"*. This outlook proved to be valid, since currently all major industrial game engines use filtered shadow maps.

CHAPTER 3

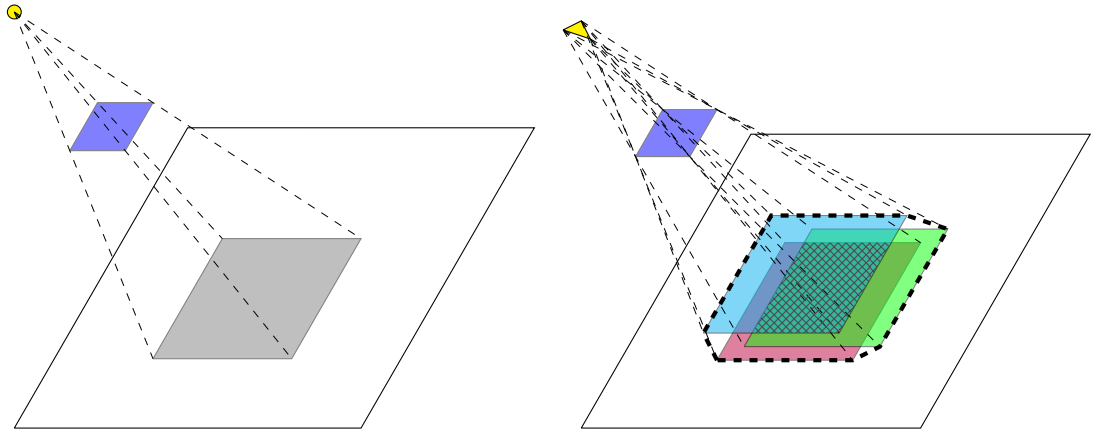
Theoretical Foundations

Recall our goal: We want to investigate the relation between resolution and filtering of shadow maps and their impact on the visual quality. This will help us find the optimizations we claim to contribute. That's why it is required to know the technical background of shadow mapping and its basic filtering techniques. In this chapter we will recap all knowledge necessary. We will talk about generating shadow maps and how to evaluate them. Afterwards, we present the basic filtering techniques necessary to reproduce our research work.

3.1 Shadows in Nature

Shadows are an omnipresent phenomenon in nature. In order for us to emulate them in software applications, we need to understand their basic concepts and how they appear and effect us.

What is a shadow? Consider a scene consisting of a light source and receivers. Receivers are surfaces in the scene which are potentially illuminated by the light source. A point on a receiver is considered to be in the *umbra* if it does not receive any light directly from the light source. In contrary a point on a receiver is considered to be in the *penumbra* if only receives part of the light emitted from the light source. To better understand what this means imagine a solar eclipse where you can only see part of the sun's surface. In this imaginary model you are the point on the earth's surface and the sun is the light source. During the solar eclipse a part of the light source is occluded by another object (i.e. the moon). In this case you are standing in the penumbra region of the shadow cast by the sun and the ground you are standing on is not completely dark, but dim lit. What we generally call a shadow is the union of the umbra and penumbra and objects that hide a point from the light source are called *occluders*. Figure 3.1 illustrates the concepts of umbra and penumbra.



(a) Point Light Source *(b) Area Light source*

Figure 3.1: *The left image illustrates how a hard shadow forms from a point light source. No point on the gray surface has a direct line of sight to the light source, because they are blocked by the occluder (in blue). A more sophisticated concept is to model the area light sources. The image on the right illustrates this concept. Points on the inner (crosshatched) shadow region have no line of sight to the light source and therefore completely lie in shadow. This region is referred to as* the umbra*. However, Points in the outer shadow region (dashed outline) have direct line of sight to a fraction of the light sources area. This region is referred to as* the penumbra*.*

Importance of shadows Definitions and analysis of the geometry of shadows date back to research by Da Vinci (see Figure 3.2) and further. Since Da Vinci's times, not only geometry has been analyzed deeper but also the importance of shadows in human perception. Today we know, that shadows impact our our perception in multiple ways, most importantly they give us visual cues about a scene. They convey important information about the scene such as the position and size of the occluder, geometry of the occluder and about geometry of the receiver. They also help us in understanding spatial relations between occluder and receiver. This is especially useful for our ability of estimating the movement trajectories of occluders in animated scenes.

Hard shadows vs. soft shadows Probably the easiest way to think of a shadow is by considering a binary status, if a point is either in shadow or not. This corresponds to a *hard shadow* which is a shadow that has no penumbra. Such a shadow could theoretically be cast by the light emitted from the infinitely small surface of a point. Infinitely small surfaces do not exist in nature, as light-bulbs, the sun and even fireflies have a considerable surface emitting light. And because they are unnatural we get a unrealistic feeling from them. Still these hard shadows play a big role in computer-graphics, as they are fairly easy to compute which we will see in Section 3.3.

Realistic light sources have a finite surface and realistic shadows have a penumbra. The

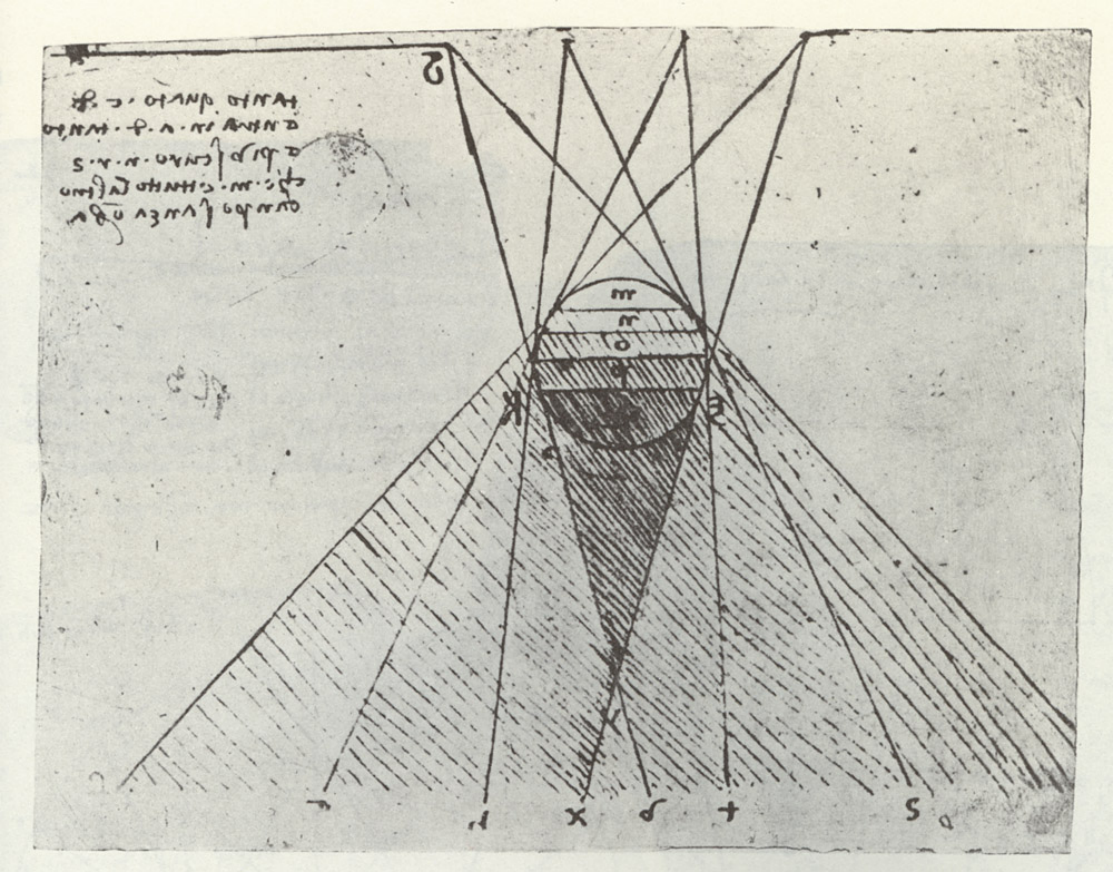


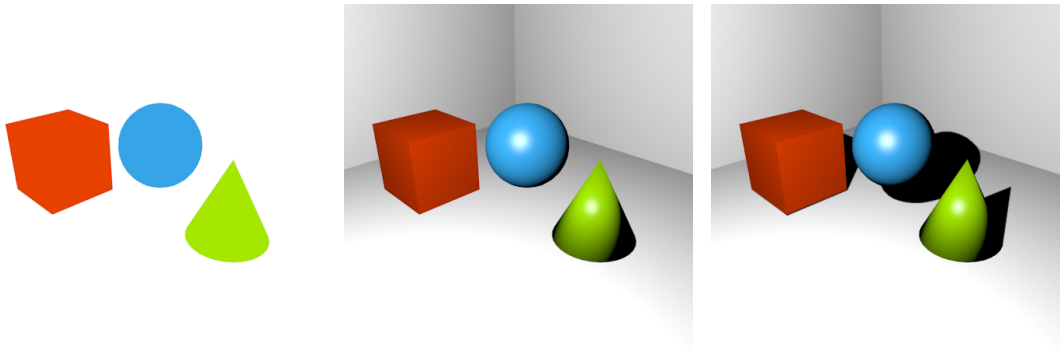
Figure 3.2: *Da Vinci observed the gradation of shadow very carefully. His sketch demonstrates how the light on the surface turned to the light gradually turns into shadow. Further, it demonstrate the shadow/light gradation on surfaces beneath the sphere. (Source: Leonardo da Vinci, Notebooks, drawing of gradation of light and shadow)*

penumbra can in the simplest form be interpreted as a blurred (soft) outline, which is why they are called *soft shadows* in general. These soft shadows are obviously more realistic than hard shadows (see Figure 3.4 for an example of a natural shadow) but they are more complicated. In general the degree of blurring varies considerably with the distances involved between light source, occluder and reciever. We will demonstrate a common method for the computation of soft shadows later in this chapter (Section 3.4.3).

3.2 Real-Time Shadows

Simulating physically realistic shadows, as they appear in nature, requires very complex computations and is impractical for real-time use. The aim of real-time rendering is to provide fast methods that are not necessarily physically accurate, but convey the impression of physical accuracy to the user as good as possible. Projecting geometry

onto a view plane alone is usually not sufficient for creating a human interpretable scene. Rather, humans get information about complex surface compositions from light being transported through the scene and it's interactions with surfaces. While physically-based rendering (the interested reader is referred to *Pharr's book* [PH04]) simulates the flow of light from the light source via surface interactions to the camera, in real time rendering we are computationally bound to keep the effort for light computation very low. The *Blinn-Phong Shading Model* [Bli77] is a representative method for calculating a light-source's impact on a surface, based on the light's direction and its distance relative to the light source. However, in real-time rendering, shading does not cover higher order reflections, refractions, nor light occlusions. Instead, these complex interactions between light and surfaces are being split into separate approximations as it is illustrated in Figure 3.3.



(a) *Material only* ***(b)*** *Shaded scene* ***(c)*** *Shadowed scene*

Figure 3.3: *In Real time rendering, complex lighting simulations are calculated step-by-step. While the color (3.3a) conveys information about the object's material, we gain information about it's geometric composition through shading (3.3b) and the spatial relations between objects through shadowing (3.3c).*

In the following section we will start with a very simple but efficient solution to create shadows called shadow mapping. Shadow mapping produces simple shadows, with hard silhouettes on arbitrary geometry. However, if we observe shadows in nature, we see that shadow silhouettes tend to be softer the further the receiver is away from the occluding surface. Figure 3.4 illustrates this phenomenon. This leads us to more sophisticated shadow mapping extensions (so called "soft-shadow" algoritms), such as *Percentage Closer Soft Shadows*, which will be discussed in more detail later in this chapter.

3.3 Shadow Maps

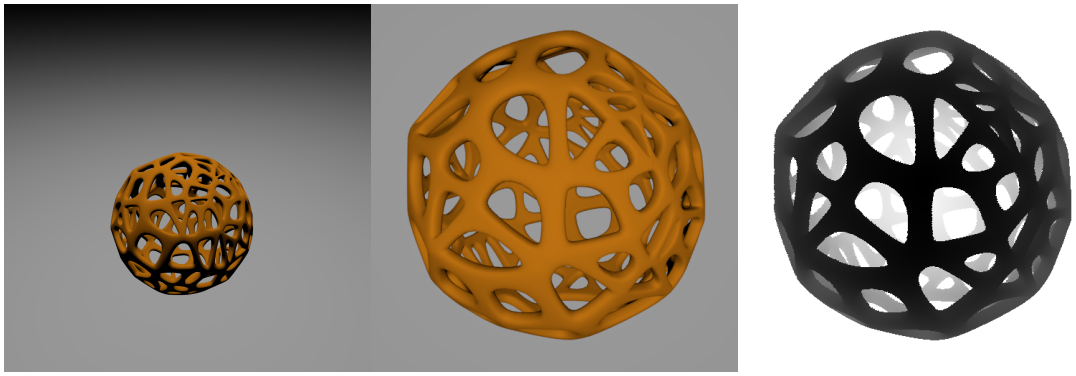
3.3.1 Basic Principle

Shadow mapping is based on a very simple observation: What the light source cannot see should not be lit by it. This implies that if we have a representation of the scene

Figure 3.4: *In nature purely hard shadows do not exist. Instead, shadow silhouettes feature complex transitions. This figure illustrates that shadow transitions are very slim at the base of the occluding geometry and extend when further away.*

as it is seen by the light source we can determine which surfaces are lit. Hence, the remaining unseen parts lie in shadow. Fortunately modern consumer graphics processing hardware is designed for massive parallel processing of geometric transformations. They allow us to project objects from a three dimensional Cartesian coordinate space into a projective space which can be rasterized (or rather discretized) onto a view plane (i.e. the computer's screen) in real-time. Shadow mapping utilizes this feature by treating the light source as a camera and uses the graphics pipeline to draw a discretized representation of the camera's view into a buffer. This buffer is called shadow map and stores depth values which allow us to infer how far a light ray reaches into the scene before it hits a surface (see Figure 3.5).

When the scene is drawn from the point of view of the eye we can transform the depth of each fragment from the viewer's perspective into the light's perspective space and compare its depth values. In order to determine whether the light reaches a fragment or not, we simply compare the depth of the surface point we see with the depth of the light ray shot in the direction of the surface point. If the depth of the light ray is smaller than



(a) Camera View *(b) Light's View* *(c) Shadow map*

Figure 3.5: *The same process which is used for rasterizing a scene from the viewer's point of view (a) can be used to draw the scene from the light source's point of view (b). The depth values resulting from such a projection convey information about how far the light rays reach into the scene before they hit a surface. This discretized-depth representation is called a* shadow map *(c).*

the depth of the fragment the ray must have hit a surface before it reached the fragment, hence the fragment must lie in shadow (see Figure 3.6).

In the following paragraphs we will describe the steps necessary to create and evaluate a basic shadow map. Afterwards, we will introduce selected extensions to the basic shadow mapping algorithm which allow us to influence the shadow's silhouettes and enhance the visual quality. These extensions will later be used in our experiments.

3.3.2 Generating a Shadow Map

Graphics hardware allows us to generate shadow maps at very little cost because we can use the same technique which is used to resolve visibility during standard rendering. Just like in the standard rendering process we use a camera matrix $M^L_{VP} = M^L_P M^L_V$ (where M^L_P is the projective transformation matrix and M^L_V transforms the scene into a coordinate frame originating at the light source) to transform the scene's geometry.

Usually a perspective camera (see Figure 3.8) is used to draw a shadow map for spotlights (in order to emulate point lights six cameras facing all axes have to be set up). Such a perspective matrix can be set up by defining the shape of the view frustum. The view frustum can be specified using near- and far plane distances (z_{near} and z_{far}) as well as the screen's extents (*height* and *width*), as referred in Equation 3.1. Figure 3.7a illustrates the geometric properties associated with such a matrix.

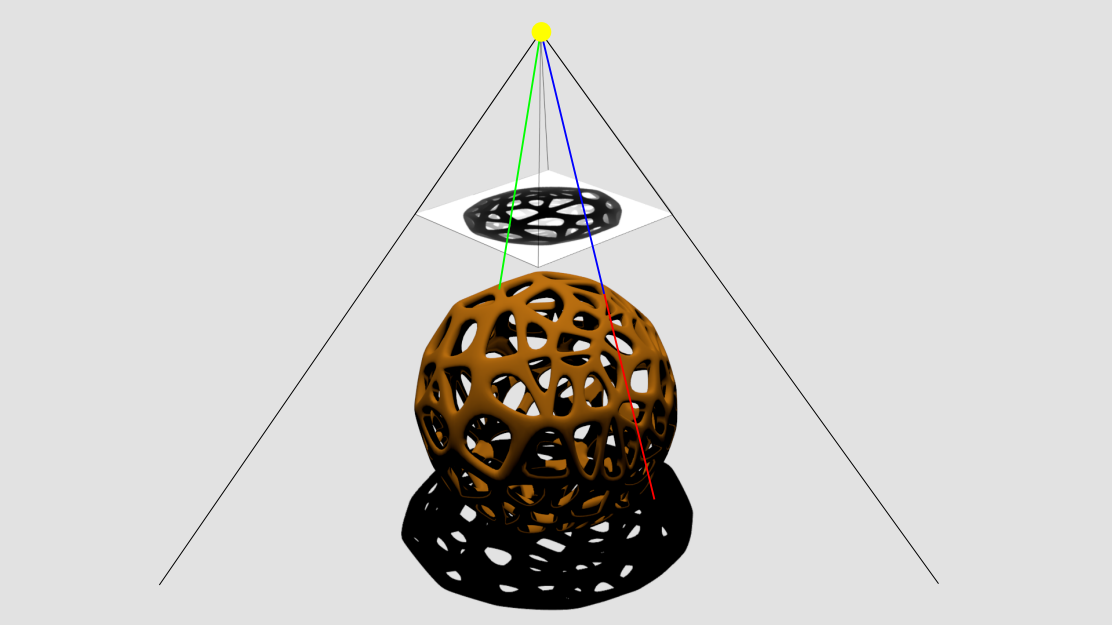


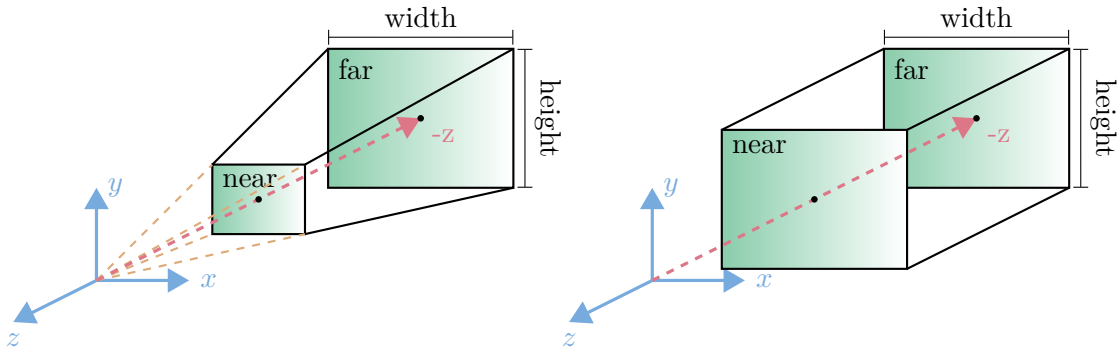
Figure 3.6: *When the scene is drawn from the viewer's point of view, we use each fragment's distance to the light source and compare it to its corresponding depth texel in the shadow map. The fragment is in shadow if its depth is greater than the shadow-map's depth (red ray) or lit if less or equal (green ray).*

$$M_P = \begin{bmatrix} \frac{2z_{near}}{width} & 0.0 & 0.0 & 0.0 \\ 0.0 & \frac{2z_{near}}{height} & 0.0 & 0.0 \\ 0.0 & 0.0 & -\frac{z_{far}+z_{near}}{z_{far}-z_{near}} & \frac{2z_{far}z_{near}}{z_{far}-z_{near}} \\ 0.0 & 0.0 & -1.0 & 0.0 \end{bmatrix} \tag{3.1}$$

An orthographic camera (Equation 3.2, Figure 3.7b) may be used for directional lights.

$$M_O = \begin{bmatrix} \frac{2}{width} & 0.0 & 0.0 & 0.0 \\ 0.0 & \frac{2}{height} & 0.0 & 0.0 \\ 0.0 & 0.0 & -\frac{2}{z_{far}-z_{near}} & -\frac{z_{far}+z_{near}}{z_{far}-z_{near}} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{3.2}$$

Once the camera matrix is set up properly, the scene geometry can be drawn. A vertex shader transforms all vertices into the lights's perspective space. The GPU's rasterizer then automatically generates a discretized depth-map (a.k.a. the shadow map) of the scene from the light's point of view. Each pixel of this depth-map then holds the distance from the light source to the first visible surface.



(a) Perspective Frustum *(b) Orthogonal Frustum*

Figure 3.7: *The Frustum of the (light's) camera defines which part of the scene is projected onto the view plane. While a perspective projection (a) is usually used for drawing spotlights, an orthographic projection (b) is suited for directional lights.*

3.3.3 Evaluating a Shadow Map

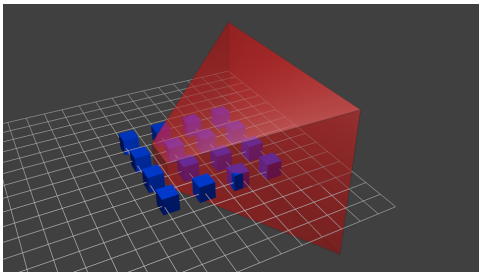
The shadow map is evaluated during the lighting process when drawing the scene from the eye's point of view. When vertices are transformed in the vertex processing stage of the drawing pipeline we compute an additional vertex position in the light-source's clip space $v' = M_P^L M_V^L v$ using the same transformation matrix used to generate the shadow map. This will later aid us in looking up the corresponding surface positions in the shadow map. Light clip-space vertex positions have to be shifted from $[-1, 1]^2$ to $[0, 1]^2$ in order to match the texture coordinates of the shadow map. This can be done by pre-multiplying with a shifting matrix resulting in Equation 3.3.

$$v' = \begin{bmatrix} 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} M_P^L M_V^L v \tag{3.3}$$

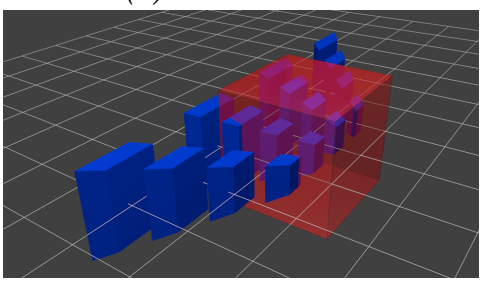
Once we process the viewer's fragments we homogenize it's light clip space position p^L by multiplying it with $\frac{1}{v_w}$ resulting in $\hat{p^L} = p^L \frac{1}{v_w}$. We then use $\hat{p_{xy}^L}$ to look up the depth samples stored in the shadow-map and compare them to $\hat{p_z^L}$. If $\hat{p_z^L}$ is greater than the stored value, the fragment is occluded by another surface closer to the light source, meaning that it is in shadow.

3.4 Shadow Aliasing

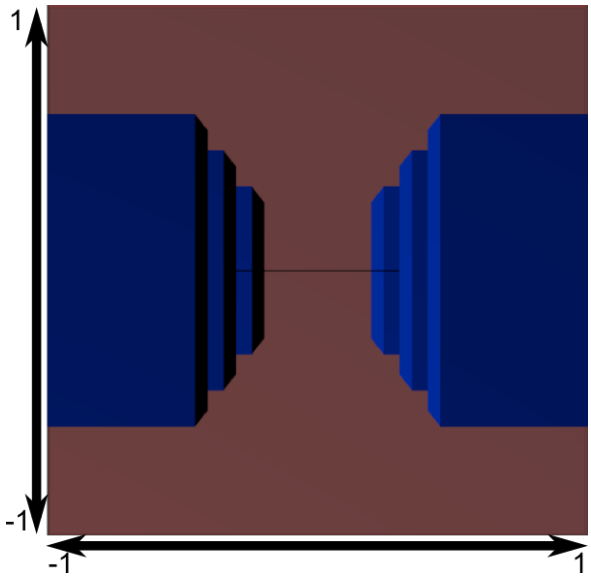
The image-based depth representation used in shadow mapping has some serious drawbacks: One of them is, that due to the shadow map's limited resolution, artifacts



(a) Frustum in Scene



(b) Perspective projection



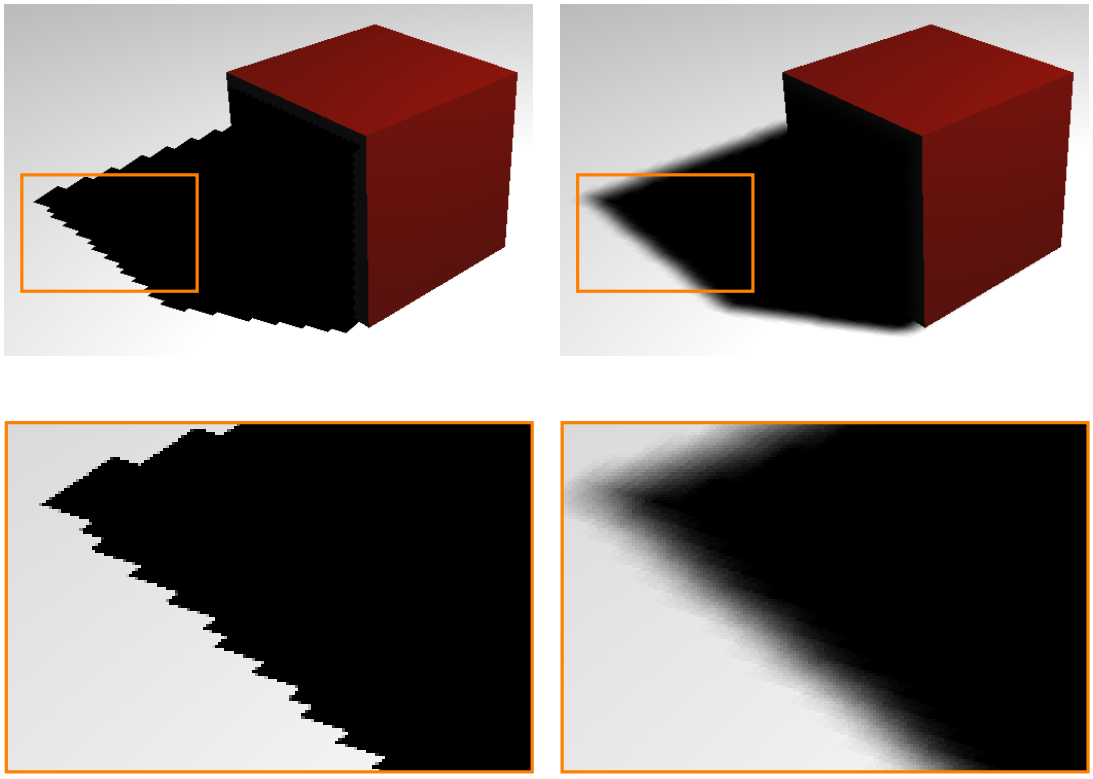
(c) Screen space view

Figure 3.8: *A perspective transformation matrix transforms scene geometry into a projective coordinate system. Geometry that was contained in the frustum (a) is now contained in a unit cube (b). This allows us to directly draw the transformed scene onto the view plane (c). Image Source: http://www.opengl-tutorial.org*

can appear, which means that shadow boundaries will often contain visible jagged-line artifacts. Figure 3.9 gives an example of such artifacts.

One major cause for these artifacts is called *undersampling*. It means that several view samples project into the same shadow map texel and they all receive the same shadow response. While this is no problem for areas inside of the shadow (they are all unlit anyway), the falsely shadowed pixels are well visible on the shadow's silhouette. Figure 3.10 demonstrates this unwanted effect by showing a shadow map being projected onto the screen. The shadow map samples the shadow caster's contour. When projected to the screen, the sample size of the shadow map will most likely not correspond to the sample size of the screen. Once the shadow tests are finished, the resulting shadow silhouette hardly corresponds to the original curve sampled by the shadow map.

The most obvious approach of limiting this problem is by increasing the shadow map's resolution. This would reduce the amount of view samples covering a projected shadow-map texel. For arbitrary scene setups however, the required resolution would grow towards infinity. Also, increasing the shadow-map size comes at a high cost in terms of memory consumption and an increasing amount of computations. One way to limit these artifacts is to filter the computed shadow. In general, shadow-map filtering algorithms can be categorized into *pre-filtering* and *post-filtering* algorithms. While pre-filtering algorithms filter the entire shadow map right after its generation, post-filtering algorithms



Jagged-line artifacts *Anti-aliased shadows*

Figure 3.9: *Jagged-line artifacts appear due to the shadow map's limited resolution (a). After filtering is applied the picture (b) is more appealing due to its smooth anti-aliased shadow boundaries.*

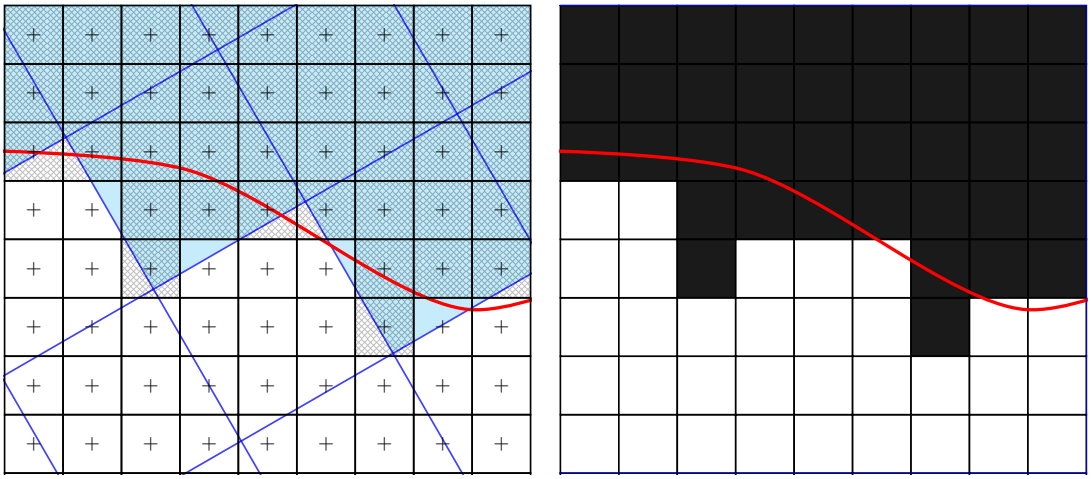
apply filtering during each shadow evaluation of a viewport fragment. In this section we are going to discuss one common approach for each category. The interested reader is referred to the book by Eisemann et. al [ESAW11], which dedicates a whole chapter on shadow-map aliasing and discusses the topic extensively.

3.4.1 Shadow Mapping as a Signal-Reconstruction Process

To understand and analyze shadow-map aliasing in detail, it is helpful to view the process from a signal-processing point of view:

The signal-reconstruction pipeline starts with a function of the depth of the scene geometry (transformed to the light-source's view plane) as a continuous input signal. The goal is to store the input signal in a discrete form (shadow map), which can later be reconstructed and compared to the depth function of the geometry seen by the camera.

The first stage of this process is referred to as "initial sampling". In traditional signal reconstruction, the input signal would be bandlimited first, and then sampled. Ban-



(a) *Shadow map projected to viewport* ***(b)*** *Shadow test result*

Figure 3.10: *Simplified setup of a shadow-map being projected onto the screen. Here the shadow-camera's view direction is assumed to be perpendicular to the screen's view direction. In the common case in which the camera orientations are not perpendicular the projected shadow map pixels would also be perspectively distorted. The shadow map (illustrated as a blue grid) samples the shadow casters contour which is represented by the red curve. When projected onto the screen the shadow-map's pixels will most likely not correspond to the sample size of the screen pixels. Figure (b) shows that many boundary pixels are shadowed wrongly, resulting in a silhouette that does not preserve the original shadow-caster's contour well.*

dlimiting removes frequencies higher than half the sampling frequency, which allows accurate reconstruction of the (bandlimited) signal (Nyquist-Shannon sampling theorem). Later the sampled signal is resampled for a particular output resolution: The signal gets reconstructed by interpolating the samples from its discretized representation and transforming it to the output domain. After that, the signal is bandlimited to accommodate the output resolution and finally gets resampled.

In shadow mapping, however, since we sample and reconstruct geometric signals, bandlimiting cannot be used, because it would reshape and obfuscate the geometry. This lack of bandlimiting introduces aliasing when the signal is reconstructed. The signal-reconstruction pipeline can be used to define various types of errors:

Initial Sampling Error Since a signal with unlimited frequencies is discretized and no opportunity exists to bandlimit the signal, the discretization introduces artifacts.

Undersampling Occurs when the initial sampling frequency is lower than the screen sampling frequency. This means that details can be lost completely and the inital sampling error gets magnified.

Reconstruction Error Occurs due to incorrect interpolation between the sampled values. This error can be avoided by utilizing a proper filter kernel.

Resampling Aliasing This error occurs due to the lack of bandlimiting when the frequency of the shadow-map samples is higher than the screen sampling frequency. Reconstruction filters (which will be described in Section 3.4.3) can be used to reduce resampling aliasing. For pre-filtering algorithms, mip-mapping can be used.

In the following sections, we introduce basic strategies to lessen the impact of these errors.

3.4.2 Reducing the Initial Sampling Error

Apart from increasing the sampling resolution, there are other ways to reduce the initial sampling error. The methods described in this section aim at maximizing the use of available resolution and depth precision. *Fitting* minimizes the light's viewing frustum by fitting it exclusively to geometry which is visible by viewer and light source. *Warping* distorts the geometry drawn to the shadow map in order to cover more detail in close proximity to the viewer's near plane and less detail for distant geometry. Finally, *partitioning* algorithms combine both aspects of fitting and warping by dividing the light's view frustum into separate partitions depending on their distance to the viewer.

Fitting A powerful way of approaching the problem is to optimize the distribution of the shadow samples. We can achieve this by adjusting the light's view frustum by fitting it to expand across necessary data only.

This idea was introduced by Brabec et. al. [BAS02]. Their approach is to extract a bounding rectangle from the scene's view camera which is then used to determine the light source's view frustum. While Brabec's method results in a very accurate light view-frustum it suffers from negative performance impacts. This is due to the circumstance that they need an extra rendering pass, an extra buffer and a costly memory transfer when reading out the buffer.

Eisemann et. al. [ESAW11] propose a basic fitting method which efficiently fits the view frustums extends to the geometry involved in shadow computation. They begin by finding a volume enclosing all potential shadow receivers. The volume of potential shadow receivers (PSR) can be described with equation 3.4, where S is a bounding box of the objects contained in the scene, L is the light's cone of influence and V represents the observer's view frustum.

$$PSR = L \cap V \cap S \tag{3.4}$$

This volume's projective space bounding box can be used to resize the projective frustum. Appending the fitting matrix F (Equation 3.5) to the view-projection matrix allows us to

map the bounding rectangle to $[-1, 1]$ post projective space. By doing so all irrelevant polygons are clipped and the shadow map fits the potential shadow receivers. Due to the circumstance that in post-projective space all light rays are parallel to the z-axis, we automatically include all potential shadow casters.

$$F = \begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

The frustum now fits the PSR's bounding box in x- and y direction.

While this allows us to optimize the distribution of the depth samples available (which is governed by the resolution of the shadow map) it does not affect the samples' precision. In order to optimize precision, we need to minimize the range in which depth values can appear. This range is governed by the distance between near and far plane. The far plane's distance can be extracted from the PSR's z-extents. Finding a suitable value for the near plane's distance is more complicated, since shadow casters might lie outside of the PSR-volume. By computing a volume of potential shadow casters (PSC) one can extract the near plane's distance. Wimmer et. al. [WS06] describe how this can easily be done suing Equation 3.6.

$$PSC = (PSC + L) \cap S \tag{3.6}$$

Nowadays the computation of the PSC volume can be omitted by enabling the *Depth Clamping* feature supported by graphic processors. Instead the PSR's z-extents can be used for the near plane as well. Due to Depth Clamping, shadow casters will still be drawn with their depth clamped to the near frustum. This guarantees that shadow casters are evaluated as being closer to the light source than the receivers.

Warping A category of techniques called *warping* goes one step further: in addition to fitting the light's view frustum warping aims at reducing the error originating from the eye's perspective projection by varying the sampling density. This idea grounds on the observation that due to perspective projection more detail is needed near the camera's near plane, than further away.

The idea was first introduced and used by Marc Stamminger and George Drettakis in *Perspective Shadow Maps* [SD02]. The approach used in this method is to perspectively distort (hence the term warp) the scene before drawing the shadow map. While living up to its terms in reducing the perspective error, PSM had the downside of changing the lights source's intended direction. The original perspective shadow mapping approach made way for more research in this field and more variations appeared, namely *trapezoidal shadow maps* [MT04] (TSM) and *light space perspective shadow maps* [WSP04] (LiSPSM).

Fortunately, these techniques can be implemented efficiently and offer great potential for optimizing shadow maps in complex scene setups. Equation 3.7 shows how the shadow matrix is build up in the case of LiSPSM. The right-hand side of the equation is already known from the standard shadow-mapping algorithm: At first, the model matrix M is transformed into perspective light space by multiplying with the light's view matrix M_V^L and its projection matrix M_P^L. In order for the warping to be effective, the orientation of the shadow map (x- or y-axis) has to be aligned with the viewers orientation (z-axis). This alignment is then described using a rotation matrix L_R. The warping matrices W_V and W_P describe a perspective projection. The projections parameters for near and far plane can be calculated by transforming the PSR bounding box Fitting matrix as described in the previous section. However, finding a good focal point of the warping matrix is not trivial and the LiSPSM article proposes various methods for it. Finally, a fitting matrix as described in the previous section is used to fit the shadow map to the view-port.

$$S_W = F W_P W_V L_R M_P^L M_V^L M \tag{3.7}$$

Warping can be beneficial in many cases, but unfortunately it has some downsides as well. The main problem of algorithms like LiSPSM is that their effectiveness depends heavily on the relative alignment of the viewer's and the light's frustums. Using LiSPSM is not always possible (e.g. light source parallel to view direction) or suitable (e.g. lightsource is behind the viewer). The partitioning techniques described in the next section offer a robust alternative to warping.

Partitioning Another promising and very practical approach to reduce the initial sampling error is to use multiple equally sized shadow maps to represent a single light source. This can be achieved by subdividing the view frustum along its z-axis and by then calculating a shadow map for each sub-frustum. This partitioning idea has been proposed by different authors as *cascaded shadow maps* [Eng06] (CSM), *parallel split shadow maps* [ZSXL06] (PSSM) and z-partitioning [LTYM06]. Using partitioning the sampling density decreases over subsequent partitions. Figure 3.11 demonstrates that the shadow map partition next to the near plane covers a smaller area than the partition next to the far plane. Therefore, the sampling density and the perceived resolution is higher near the viewer, which is an effect similar to warping.

The algorithm for constructing these split shadow maps is fairly simple. According to a split rule, the view frustum is split into n partitions along the z-axis. For each of the partitions a shadow view frustum is fit in the same manner as explained in Section 3.4.2. In order for partitioning to work well, the placement of splits is very important. Lloyd [LTYM06] argues, that an optimal warping function would be logarithmic and therefore proposes Equation 3.8 to define split positions C_i^{log} (where n is the number of

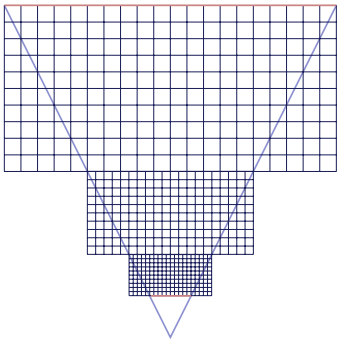


Figure 3.11: *Cascaded shadow maps work by dividing the light-source's view frustum into different sections aligned with the camera's view frustum. This results in more detail close to the viewer and less detail in far away regions.*

partitions, i is the partitions index, and z_{near}, z_{far} are the view frustum planes).

$$C_i^{log} = z_{Near}\left(\frac{z_{Far}}{z_{Near}}\right)^{\frac{i}{n}} \tag{3.8}$$

The logarithmic splitting scheme allocates most resolution to an area close to the near plane. In practical scenarios this area is often not populated with objects. Two notable extensions exist in order to counter this problem: Zhang et. al. [ZSXL06] propose *practical splits* where the split positions are computed as a weighted average between a linear- and the logarithmic split distribution:

$$C_i^{pract} = \alpha C_i^{log} + (1-\alpha)\left(z_{Near} + \frac{i}{m(z_{Near} - z_{Far})}\right)$$

Lloyd et. al. [LGQ+08] suggest to replace the distance of the near plane z_{Near} in Equation 3.8 with an adapted value z'_{Near} that is adjusted to the PSRs for all splits except the first.

Depending on the targeted hardware the partitioned shadow maps can either be stored in separate textures (old hardware support), texture arrays (faster) or in a texture atlas (often impractical because of limited resolution).

Special care has to be taken for filtering transitions from one partition to the next as they will be visible in most cases. This effect can be reduced by combining z-partitioning with warping.

3.4.3 Reducing Undersampling and Reconstruction Errors

Undersampling occurs when the (initial) sampling frequency of the shadow-map samples is lower than the screen-space sampling frequency (assuming the shadow map is projected to the screen). This means that the pixel structure of the shadow map will be visible

Figure 3.12: *Side-by-side comparison of the same shadow silhouette rendered with different shadow map resolutions. The blocky artifacts tend to disappear with increasing shadow-map resolution.*

Figure 3.13: *Side-by-side comparison of the same shadow silhouette rendered using different filter radii. The radius increases from left to right. The blocky artifacts resulting from the resampling error can be blurred with a filtering radius.*

as blocky staircase artifacts, i.e., the initial sampling error is revealed. In the worst case, details of the original shadow caster's structure can be lost entirely. Figure 3.12 demonstrates the impact of increasing the resolution of the shadow map (i.e., the initial sampling frequency). We can see that staircase artifacts shrink with increasing resolution up to a point where they are not perceivable anymore.

Increasing the sampling frequency, however, comes at a high cost in terms of computational effort, memory- and bandwidth usage. Using a properly chosen filter algorithm to reconstruct the shadow's silhouette, like Percentage Closer Filtering, helps to blur staircase artifacts caused by undersampling. Figure 3.13 shows the impact of different filter radii on the shadow's silhouette.

Percentage Closer Filtering

Ordinarily, one would filter a texture (or any image) by accessing a region of it and applying some sort of filter kernel. Unfortunately, this process is more complicated with

shadow maps since they contain spatial information. While properly filtered color values can be used without drastically altering shapes or other features of images, the geometric meaning of a filtered depth value might bear no relation to the original geometry of the scene (image blurring the geometry of a cube). Another problem arises during the shadow test, where the filtered depth value stored in the shadow map is compared against a depth value of a point on the surface. The shadow test results in a binary value indicating whether the pixel is lit or not. This binary result would make soft anti-aliased shadow contours impossible because there would be no gradations between a point on the surface being lit or unlit. This means that unlike color textures, shadow-map textures cannot simply be pre-filtered to remove aliasing.

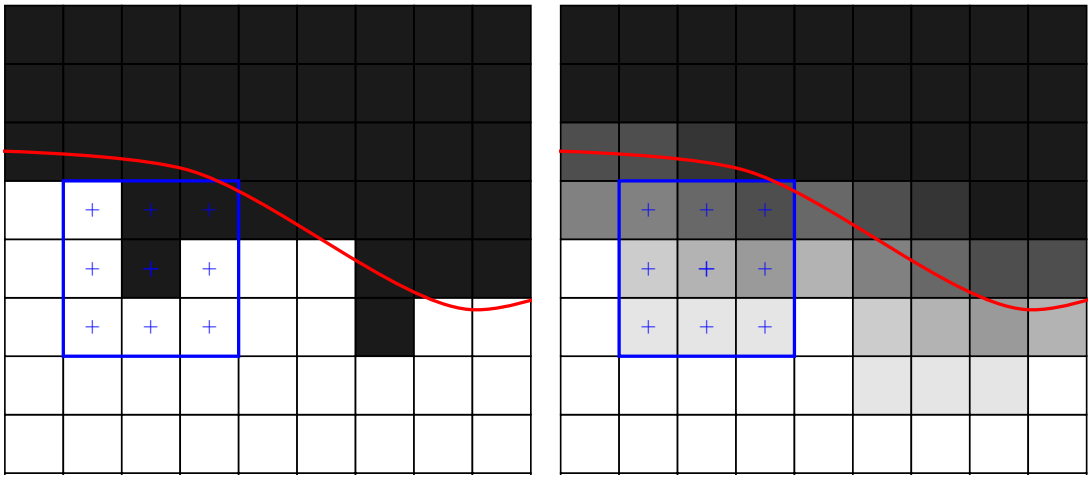
Instead of filtering the depth values, the desired softening effect can be accomplished by performing multiple shadow-map comparisons per pixel and averaging the binary results, resulting in nuanced shadowing factors. This technique is called *Percentage-Closer Filtering* (PCF) because it calculates the percentage of the surface near a point that is closer to the light and, therefore, not in shadow. Figure 3.14 demonstrates the principle of PCF by applying a 3×3 averaging kernel to the results of the shadow test. While the resulting image still cannot reconstruct the contour of the shadow caster in full detail, it offers an improvement by suppressing the artifacts.

Besides defining the density and size of a filter window, the method of accessing samples is also relevant. In the example shown in Figure 3.14, the spacing between neighboring samples matches the texel size and *nearest-neighbor sampling* can be applied. However, for arbitrary kernel sizes, *bilinear sampling* improves visual results. In contrast to nearest-neighbor sampling, bilinear sampling linearly interpolates values of the neighbors adjacent to the sample center. Fortunately, modern graphics processing units are capable of performing bilinear sampling without negatively impacting lookup performance. They even support bilinear PCF sampling, where the user supplies the transformed depth and the process of lookup, comparison and bilinear combination is performed by the hardware.

Filter Kernels

The choice of the filter kernel is crucial and allows for powerful improvements in the reconstruction of shadow contours and in suppression of artifacts. Figure 3.15 demonstrates the visual effects of using different filter kernels. In general, filter kernels define the neighborhood incorporated in a sampling operation. Just like in image processing, filter kernels can be chosen in arbitrary shapes, sizes and sample distributions. However, unlike the countless applications of filtering in image processing, shadow filtering solely aims at achieving smooth and plausible transitions between lit and unlit surfaces. For PCF this means that filtering operations are limited to blurring (low pass) operations. These are performed by averaging sample values and optionally weighting them by the distance to their center.

When designing a kernel used for PCF filtering, three parameters are relevant: *filter radius*,



(a) After shadow test *(b) Shadow after filtering*

Figure 3.14: *In this example PCF is performed by applying a 3×3 filter kernel to the shadowed pixels after the shadow test. The effects can be seen by comparing both shadow contours to the actual shadow caster's boundary (red curve): Artifacts are being suppressed and the shadow's contour has a closer resemblance to the actual shadow caster's contour.*

sample density and *sample distribution*. *radius* of the kernel allows the adjustment of the blur range and is usually chosen in shadow-map texture space for signal reconstruction. Changing the *sampling density* allows the adjustment of the blur granularity.

In Figure 3.14, a 3×3 filter kernel which consisted of nine samples evenly distributed on a regular grid was averaged for each pixel.

Using a uniform, regular distribution of samples might lead to repeating visual patterns in the resulting image. However, repeating patterns do not appear often in nature and are therefore rarely perceived as natural by human observers. Unlike regular sampling techniques, non-uniform sampling techniques help turning regular aliasing patterns into less structured transitions. Hence, in computer graphics, non-uniform sampling is often preferred over uniform sampling as its results are more representative of the unstructured geometry in natural phenomena and therefore more appealing.

A simple and efficient non-uniform sampling technique is *Poisson-disk sampling*. Unlike regular sampling strategies, Poisson-disk sampling results is an even, but random distribution of samples. In a Poisson disk kernel samples are distributed on a unit disk and are tightly packed, meaning that the distance between neighboring samples is uniform and minimal for all samples. Figure 3.16 visualizes these properties and compares a regular kernel with a Poisson-disk kernel.

Poisson disks can be created naively using a "dart-throwing" technique introduced by Cook [Coo86]. The algorithm continuously calculates uniformly distributed points and only keeps them if they satisfy a minimum distance to all of the already existing points.

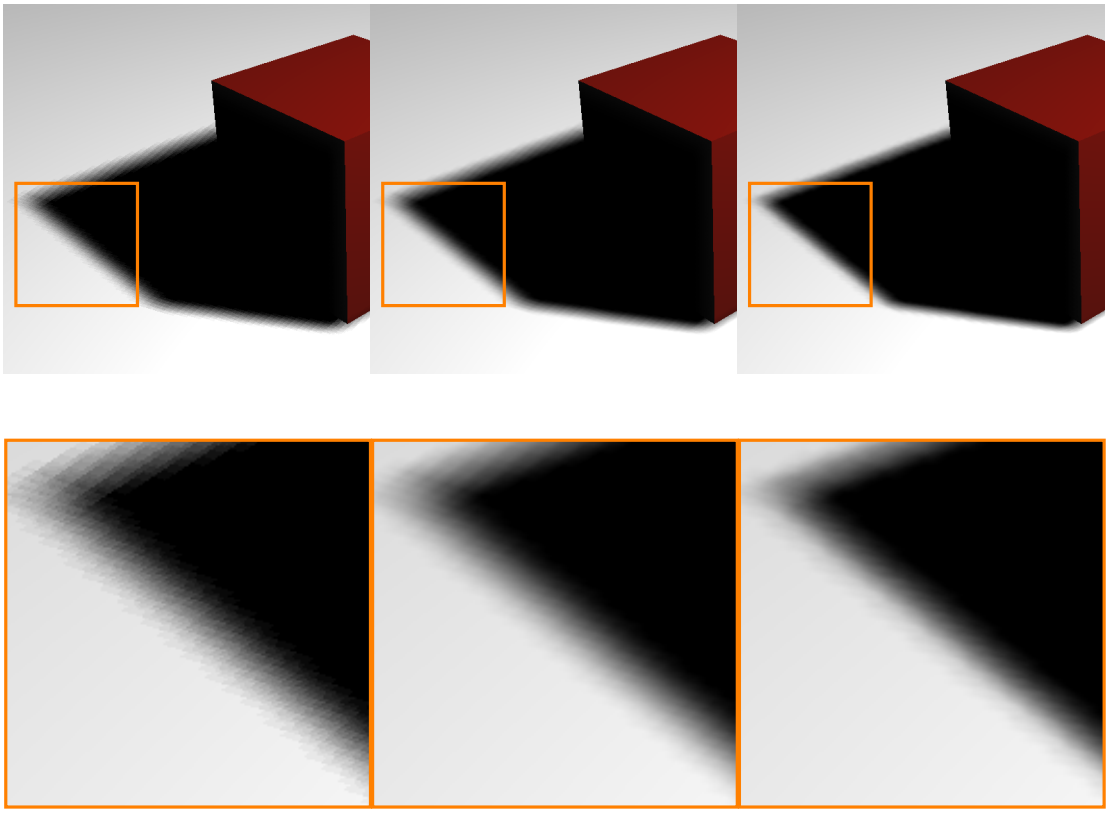


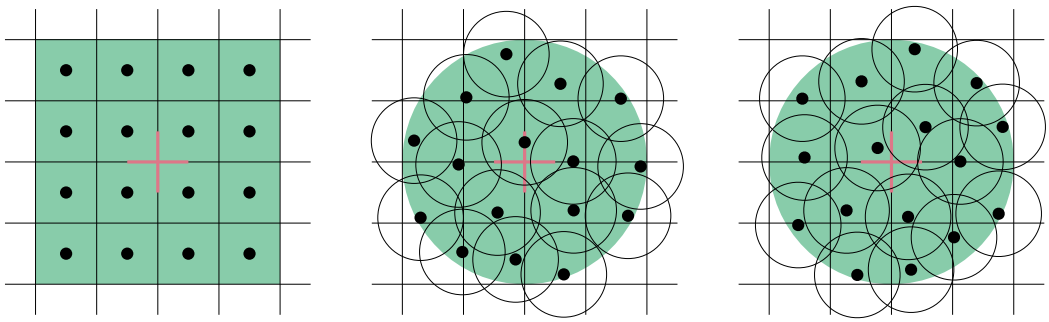
Figure 3.15: *The choice of a proper filter kernel and sample size allows for a broad range of quality and performance improvements.*

Besides being slow (speed improvements exist [DH06, Jon06]), this algorithm has many downsides. For example, if aborted prematurely, the distribution of the points might be non-uniform. The outcome of the algorithm can only be defined by the desired distance between points. Since we need to control the amount of samples taken by a kernel, this algorithm is usually unfavorable. Despite all of its disadvantages the algorithm's natural way of generating a Poisson-disk makes it an often-used candidate for a ground-truth set.

A fairly simple extension to the dart-throwing approach is called "*Relaxation Dart Throwing* [MF92]. It starts by dart throwing points at a large radius and – after a large number of attempts fail – reduces the radius by some fraction. The relaxation algorithm is faster, termination is guaranteed and, most importantly, the number of points can be controlled.

Over the last three decades, many more algorithms were proposed. However, it lies beyond the scope of this thesis to investigate all of them. The interested reader is referred to the comprehensive comparison of Poisson disk algorithms by Lagae et. al. [LD08].

Instead of calculating a unique, separate kernel for each processed pixel, shadow filtering



(a) Regular Kernel *(b) Poisson Disk Kernel* *(c) Rotated Poisson Disk Kernel*

Figure 3.16: *A regular sampling kernel (Figure 3.16a) distributes samples evenly over a rectangular area. Due to its rectangular area of influence the regular kernel produces banding artifacts. In contrast to this a Poisson disk kernel (Figure 3.16b) distributes samples randomly and evenly spaced inside a circle. Repetitive patterns in the filtered image can be mitigated by randomly rotating the filter kernel (Figure 3.16b).*

applications require a more practical way of using Poisson-disk kernels. A practical and efficient method of sampling non-uniformly is to pre-calculate the Poisson-disk kernel, and randomly rotate it every time it is utilized.

Poisson-Disk sampling as well as other techniques used in PCF are foundations for many other shadow-filtering methods. The shadow-filtering methods described in the next pages benefit from the same considerations in terms of kernel choice.

Variance Shadow Maps

PCF's filtering performance is limited because the kernel window has to be sampled for each viewport fragment. While this performance issue is relatively insignificant for small kernel windows with few samples, it increases exponentially when more samples are required. Instead of sampling lots of shadow-map samples as in PCF, Donnelly and Lauritzen [DL06] observed that the depth sample partitioning can be estimated using simple statistic measures and then be used to determine the amount of light reaching a point on the surface. They suggest a different approach to shadow mapping called *Variance Shadow Mapping* (VSM). Their idea is to store information in one shadow-map sample that allows for restoring mean and variance estimates of the depth's sample distribution. The key benefit over PCF is that the variance shadow map can be pre-filtered like a normal texture. This ability to pre-filter allows for a feasible bandlimiting option, using the GPU's hardware mipmaping [Wil83] capabilities.

In addition to storing depth values, a variance shadow map also stores squared depth. By applying a simple box (or gaussian-) filter, the resulting texture then contains the first moment M_1 and the second moment M_2 for the underlying depth distribution. These

moments allow us to estimate mean ($\mu = M_1$) and variance ($\sigma = M_2 - M_1^2$). We can easily vary the size of the filter window to adapt the shadow transition-radius, because the moments can be averaged while still preserving the original depth distribution. By choosing an appropriate window size, VSM can achieve very similar results as PCF while only taking a single sample from the pre-filtered shadow map for determining whether a fragment lies in shadow. In order to perform shadow testing, Equation 3.9 is used to compute an upper bound for the probability that a receiver point with light-space depth $\tilde{z}$ is smaller than a randomly drawn depth value from the distribution. This probability $P(\tilde{z})$ is then used to vary the shadows intensity. A visual comparison between PCF and VSM is demonstrated in Figure 3.17. Unfortunately, in cases where the variance is high VSM suffers from an effect which the author calls *light bleeding*. The author proposes an effective solution named layered VSM [LM08] to counter such situations.

$$p(\tilde{z}) = \frac{\sigma^2}{\sigma^2 + (\tilde{z} - \mu)^2} \tag{3.9}$$

Percentage Closer Soft Shadowing

Methods like PCF and VSM allow us to generate smooth shadow silhouettes, but if we recall the natural scene from Figure 3.4 we see that shadow boundaries become sharper as objects get closer together and softer as they grow apart, because the light is emitted from a finite area, rather than from a point. This phenomenon is referred to as soft shadowing and the previously mentioned approaches do not cover it, because they soften the shadows boundary solely based on a predefined size of the filter window.

Fernando [Fer05] presents a method to generate perceptually accurate *soft shadows*, called *Percentage Closer Soft Shadows (PCSS)*. In contrast to fixed kernel size algorithms, PCSS takes the area of the light source into account. In Figure 3.18 we can easily see that increasing the area of the light source increases the penumbra width. Also, it is easy to imagine that the penumbra shrinks the closer the occluding surface (i.e. the Blocker) moves to the receiving surface (i.e. the receiver).

PCSS is based on this observation and for the sake of reducing computational complexity, it assumes that blocker and receiver are parallel to each other. The actual filter window size is computed per fragment using the relation between the fragment's (receiver-) depth, the average blocker depth and the with of the area light source. While receiver distance $\tilde{z}$ and the width of the light source are already known, the average blocker depth $z_{Blocker}$ has to be computed by averaging the neighboring shadow map samples. The search radius r_{Search} for the lookup window can be computed using Equation 3.10 which is derived from the geometric relation between near plane and receiver shown in Figure 3.19a using the parallel planes equation.

$$r_{Search} = w_{Light} \frac{z_{Receiver} - z_{Near}}{z_{Receiver}} \tag{3.10}$$

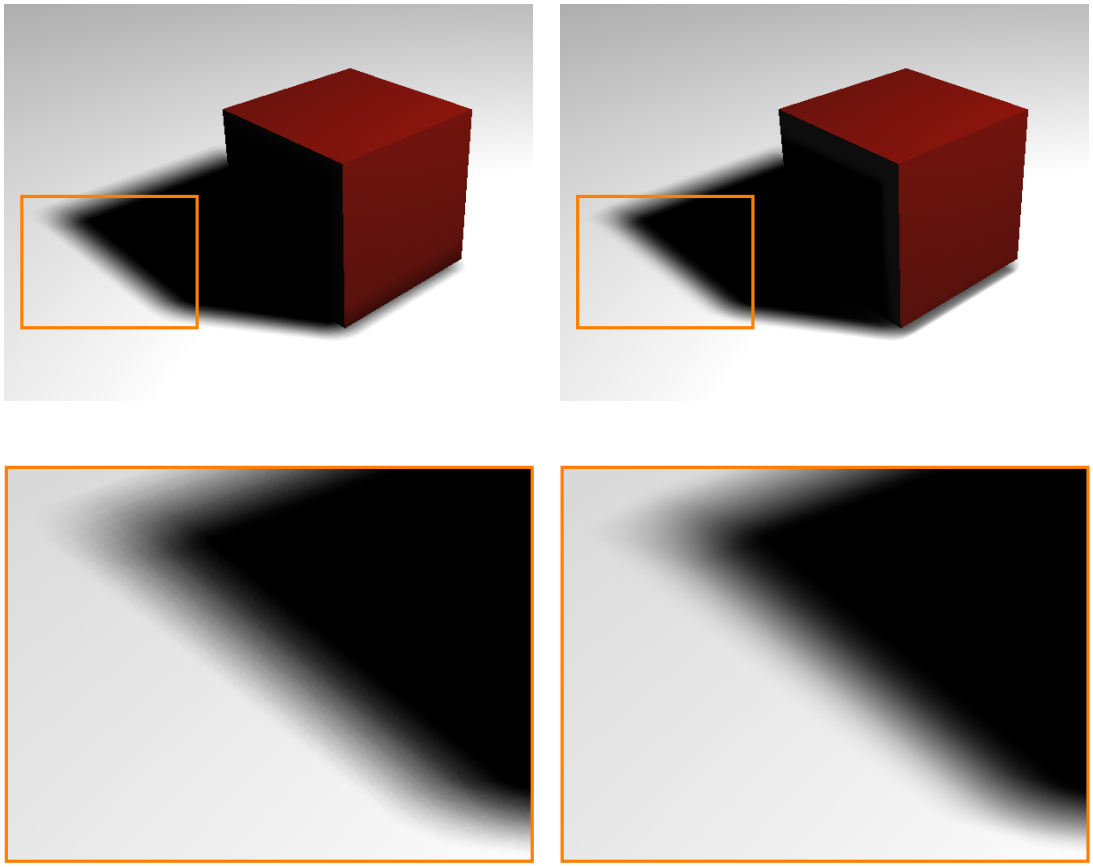


Figure 3.17: Variance Shadow Mapping *(VSM) produces very similar results as* Percentage Closer Filtering *(PCF) while being less computationally expensive for large filter kernels.*

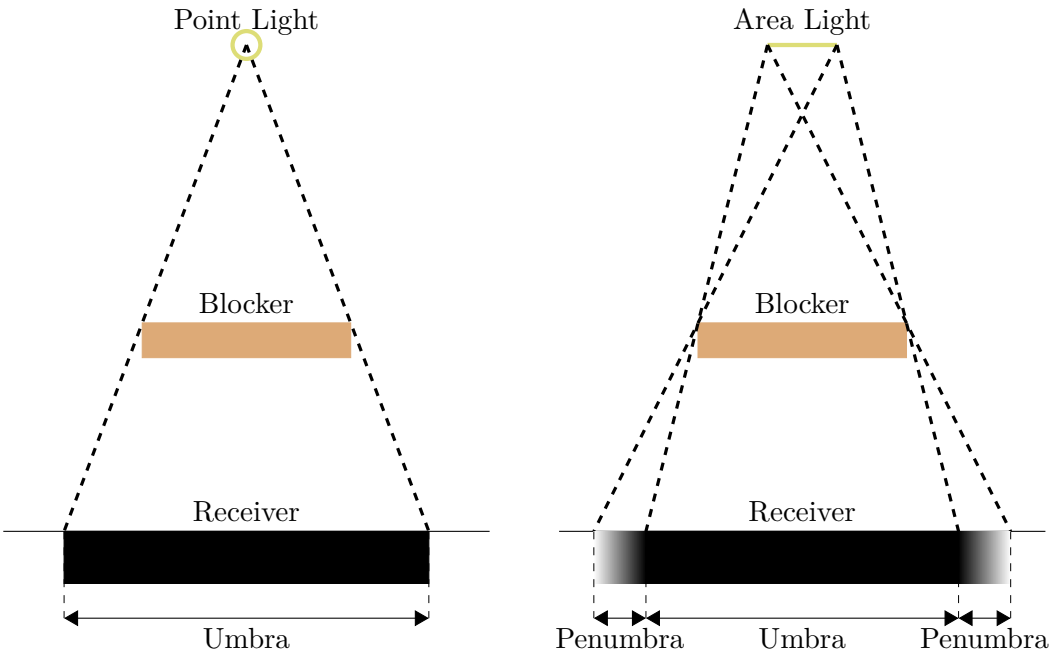
Once the average blocker depth $z_{Blocker}$ is calculated the penumbra width $w_{Penumbra}$ can be determined by using Equation 3.11 derived from the relations depicted in Figure 3.19b.

$$w_{Penumbra} = w_{Light} \frac{z_{Receiver} - z_{Blocker}}{z_{Blocker}} \tag{3.11}$$

Once the penumbra size is estimated, PCF can be used for the actual filtering process. A visual comparison between PCF and PCSS is demonstrated in Figure 3.20.

The interested reader might want to have a look at *Summed Area Tables Variance Shadow Mapping* [LM07], which is an effective method for integrating variance shadow maps into PCSS. Also, there is an excellent presentation giving an overview on soft shadow algorithms by Bavoil [Bav08].

For a detailed description on how to implement the PCSS algorithm see Chapter 5.



(a) *Hard Shadow* ***(b)*** *Soft Shadow*

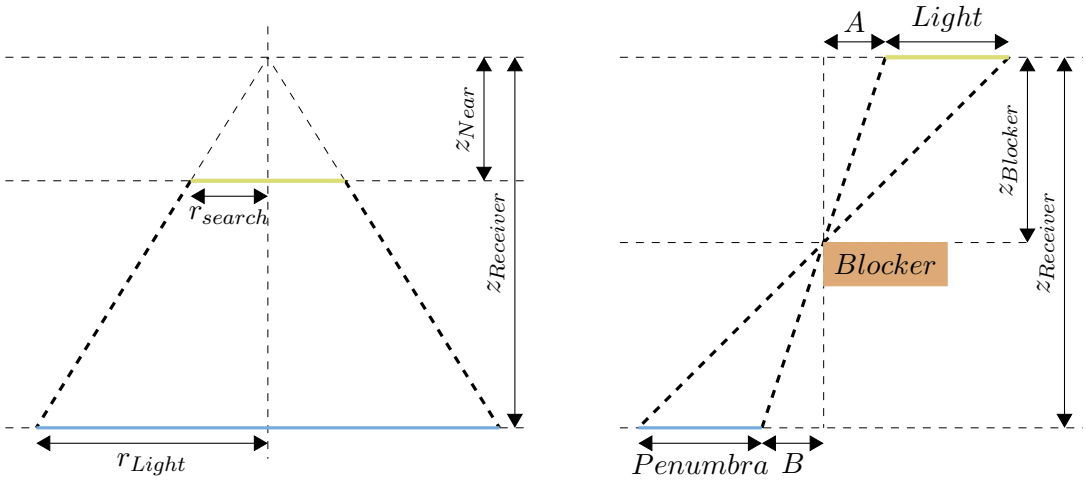
Figure 3.18: *Hard Shadows vs. soft shadows. The decision whether a point lies in shadow can be answered by "yes" or "no" only for point lights. For area light sources the decision is more complicated. The region of the shadow's silhouette where only parts of the light source are occluded is called* the penumbra.

3.4.4 Incorrect Self-Shadowing

Recall that a shadow map is a discretized representation of the scene from the light's point of view. Because of the shadow-map's limited resolution, we will encounter some kind of error in almost all cases when we compare a screen fragment's depth to a shadow map sample. When sample points from the viewer are compared to shadow map samples the view sample will rarely project exactly to the location that was actually sampled in the shadow map, but will instead compare to the nearest neighboring points. As an example imagine a shadow being cast on a tilted plane. Within each shadow map pixel depth values of the view samples covering that pixel would vary, some of them being higher an some of them lower. Those being higher would then be declared to lie in shadow while the others are being lit causing unpleasant Moiré patterns in shadows. Figure 3.22 illustrates this problem.

If we were to apply a depth bias to all of our view samples, the depth comparisons would become more robust. Figure 3.21 illustrates this technique which is often called *z-Bias*.

Instead of finding a suitable constant bias only (which would be impossible for most

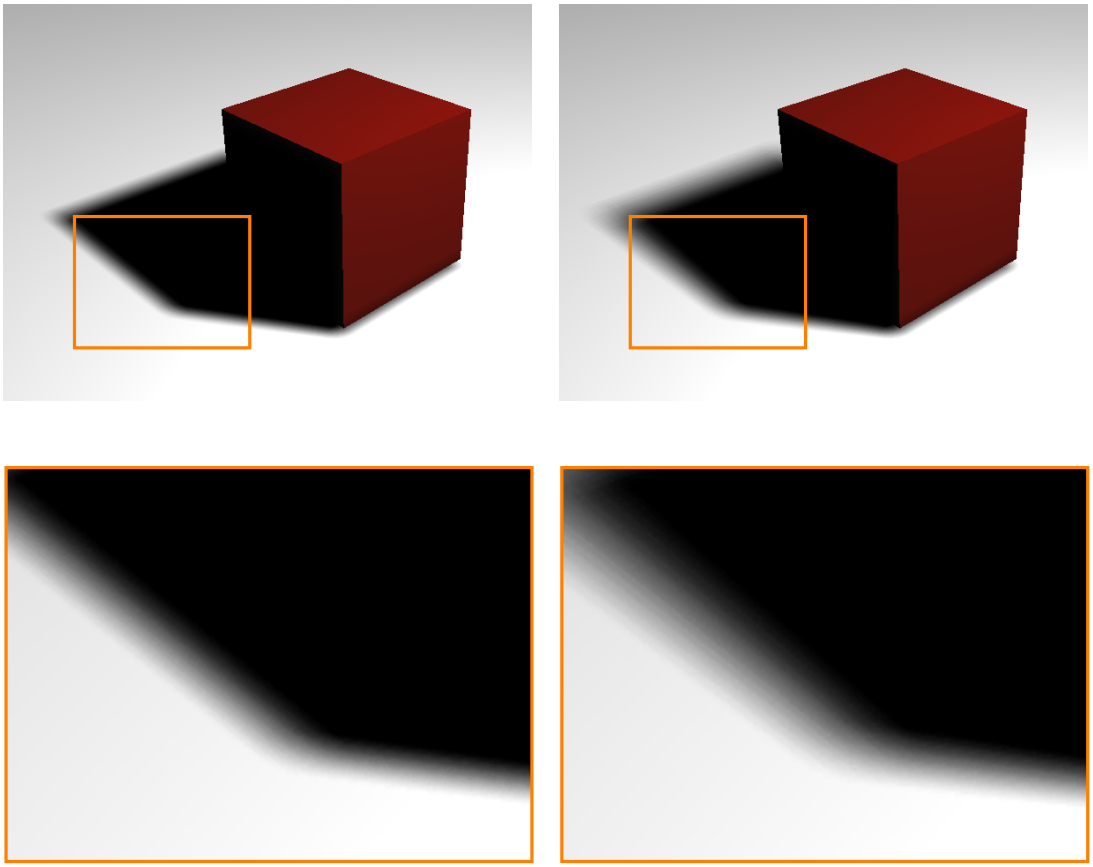


(a) *Blocker Search* ***(b)*** *Penumbra Estimation*

Figure 3.19: *In* Percentage Closer Soft Shadows *the penumbra width varies for each fragment. The figure on the right shows that the penumbra width* $w_{penumbra}$ *depends on the average blocker distance* $z_{blocker}$*, the fragment's distance* $z_{receiver}$ *and the width of the light source. The average blocker distance itself has to be determined by examining the shadow map neighborhood of the search radius* r_{search} *defined by the relation shown in the figure on the left.*

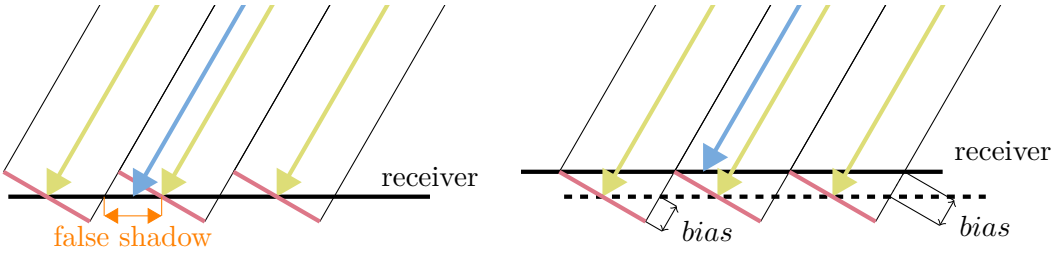
scenes) we apply an additional bias that depends on the slope of the fragment's surface. If the bias is chosen too high, shadows get shifted away and get disconnected from their casters or might disappear entirely. This problem is also known as *Peter Panning* (see Figure 3.23).

Fortunately, OpenGL and DirectX both support depth-biasing natively. In case of OpenGL the biasing can be enabled by enabling the *GL_POLYGON_OFFSET_FILL* functionality. Its behavior can be controlled by setting the *glPolygonOffset* parameters. While the first parameter operates on a surface's change in depth and scales the maximum Z slope, the second parameter scales the minimum resolvable depth value. The actual depth offset is added before the GPU performs the depth test, which allows us to resolve z-fighting issues during shadow computation. In practice the parameters have to be chosen manually. Adjusting near and far plane of the light's frustum and using linearized depth in shadow maps helps to decrease the need for large biases by maximizing the usable depth resolution.



PCF *PCSS*

Figure 3.20: Percentage Closer Soft Shadows *(PCSS) produces more realistic shadows as* Percentage Closer Filtering *(PCF). While PCF produces a penumbra of constant radius, in PCSS the penumbra radius grows with distance to the occluder.*



(a) *without z-bias* **(b)** *with z-bias*

Figure 3.21: *Due to the limited resolution of a shadow map, sampling depth at arbitrary positions leads to erroneous results. Picture (a) illustrates a naively sampled shadow map. The shadow map stores the depth as evaluated at the texel center (yellow arrow). If we were to sample the depth in the region marked with "false shadow" (blue arrow) the shadow test would result in the region being falsely occluded. This happens because the depth value stored in the shadow map would be closer to the surface than the depth value obtained by the projection to light space. In picture (b) this can be avoided by introducing a bias (i.e. by shifting all values stored in the shadow map).*

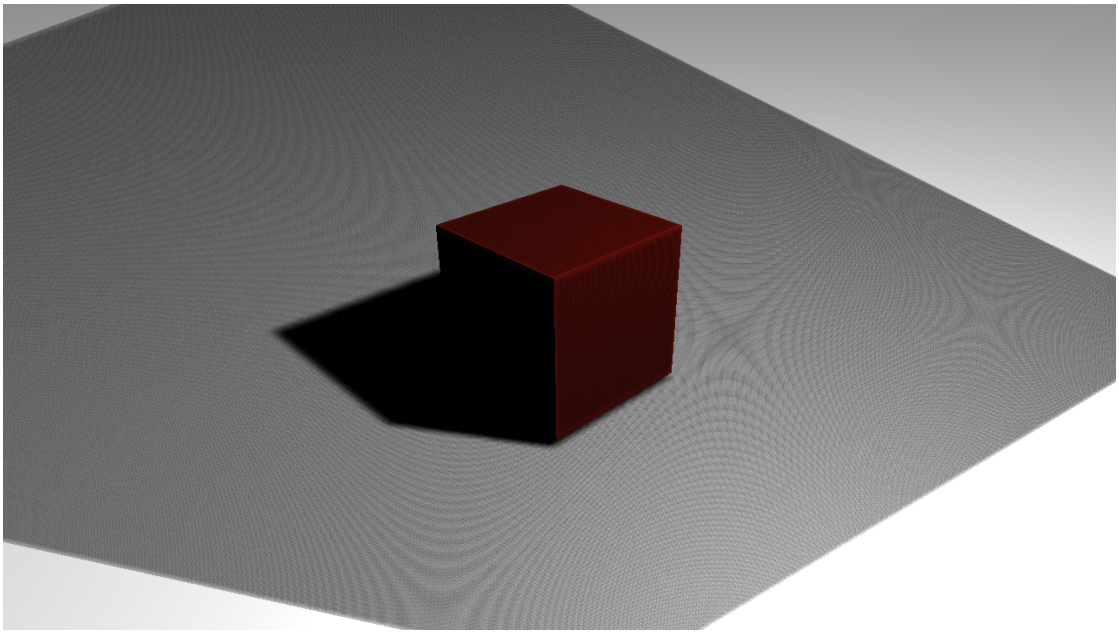


Figure 3.22: *Moiré patterns known as* Shadow Acne *appear due to limited shadow map resolution and numerical errors. They can be removed by using a depth bias.*

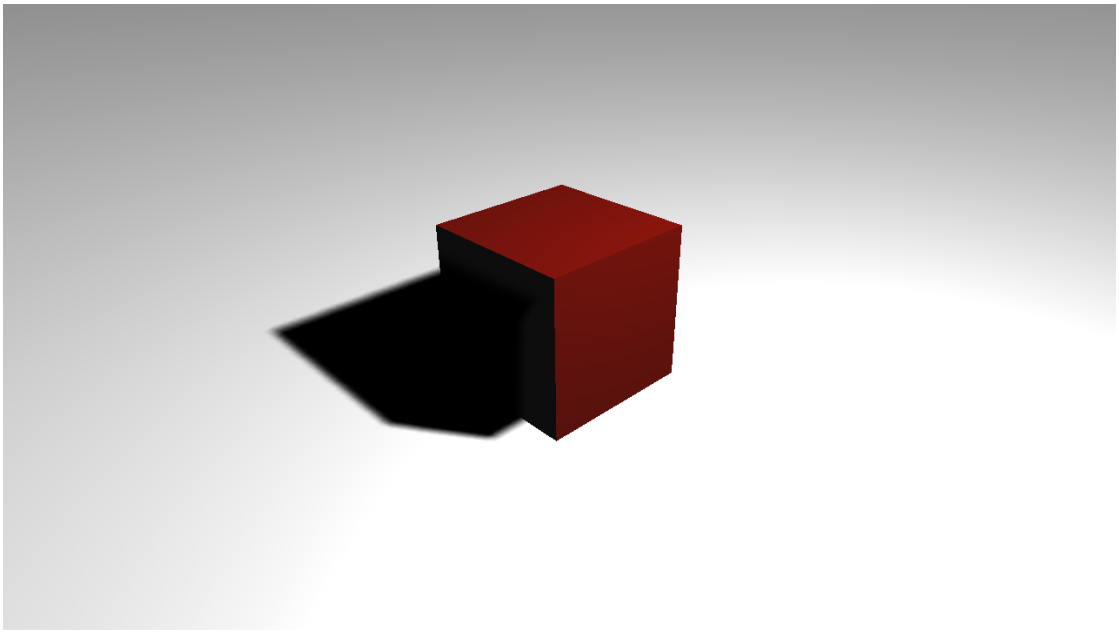


Figure 3.23: *While choosing a higher depth bias removes* Shadow Acne *it has to be chosen carefully, because shadows might get disconnected from their casters if it is chosen too high. This problem is also known as* Peter-Panning.

CHAPTER 4

Experiment

Now that we discussed the theoretical foundations for shadow-mapping algorithms and its common problems and remedies, we can further investigate the behavior of shadow-map filtering. Let us therefore recall the shadow-mapping process described in the last chapter:

Shadow maps are generated per frame by sampling depth values of a scene from the light source's perspective. The sampling process makes use of the hardware graphics pipeline by transforming scene geometry into the perspective view space of the light source and storing the nearest depth values per pixel fragment.

Later, when the scene's per-pixel lighting is calculated, the fragment shader projects each fragment to light space and queries the shadow map to compare depth values. In case of filtered shadows, multiple queries are performed in the vicinity of the fragment in question and the comparison results are filtered by averaging in order to achieve a blur effect on shadow borders.

If configured properly, this blurring filter can be used to hide undersampling artifacts on the shadow boundary by substituting missing bandlimiting of the original signal with a blur during reconstruction. This leads us to the hypothesis that with increasing softness of the shadow (i.e., a larger filter size) a less detailed shadow-map resolution (i.e., a lower initial sampling-frequency) is required to produce visually sound results.

Figure 4.1 demonstrates this observation by comparing the visual impact of varying filter sizes with different resolutions of the same shadow silhouette. We can see that the resolution required for displaying a perceptually sound shadow silhouette seems to be directly affected by the size of the reconstruction filter.

These observations suggest that an optimal ratio between shadow-map resolution and filter radius exists. Since optimizing these two parameters is supposed to result in an optimal visual quality, we have to analyze the correlation between them from a perceptual point-of-view. Therefore, we will set up a perceptual user study in which the participants

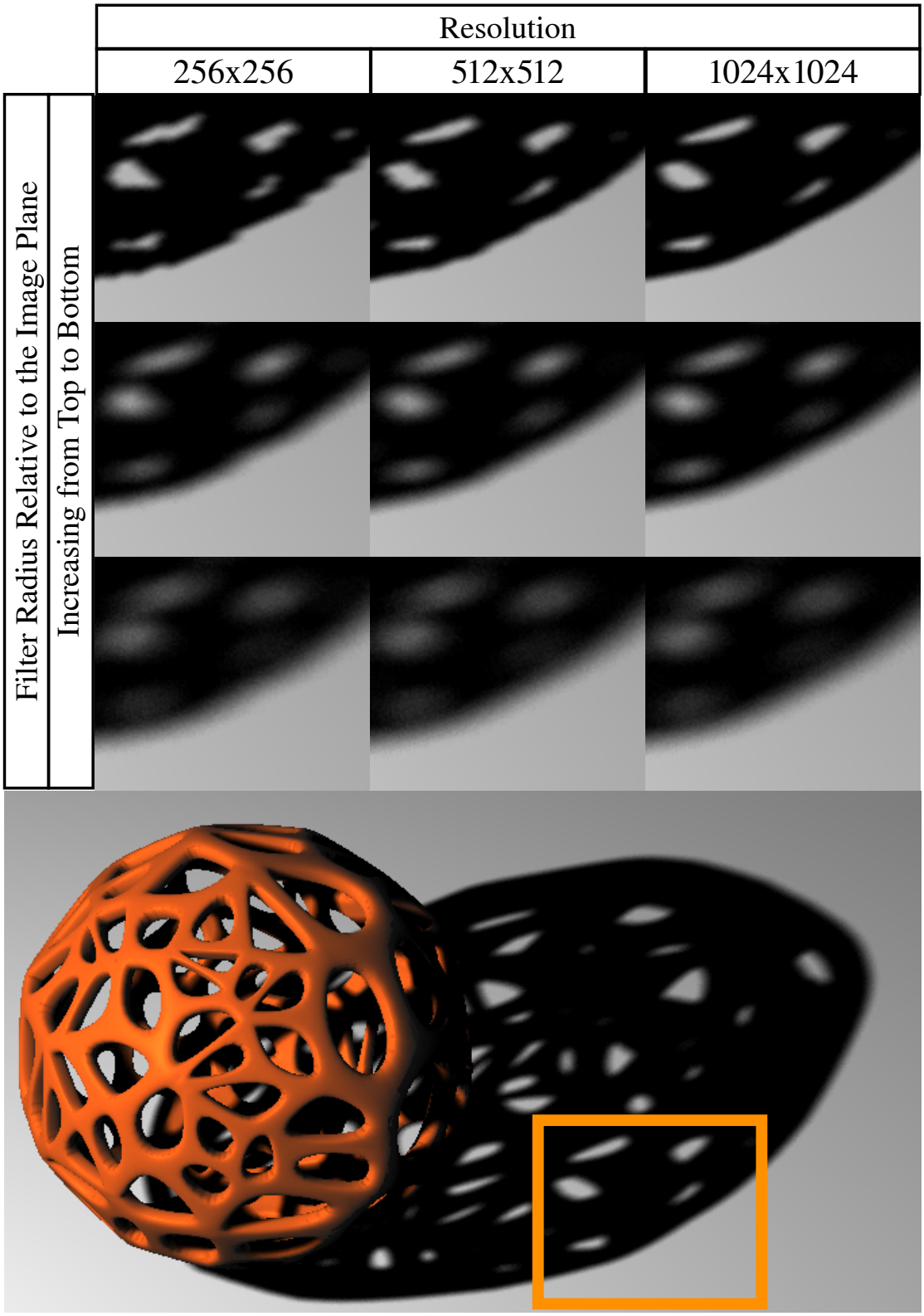


Figure 4.1: *By doubling the size of the reconstruction filter we can reduce the shadow map's resolution by half without noticeable negative impact on the visual quality.*

have to evaluate the visual impact of undersampling artifacts. This study allows us to investigate how the reconstruction-filter radius and the initial-sampling frequency have to be chosen in order to hide undersampling artifacts from the human observer. In this chapter we are going to investigate the parameters involved to set up such a user study. With the results obtained from the user study we should then be able to formulate a reliable function describing the optimal ratio, which enables the user to select a resolution for a given blur size or vice-versa.

4.1 Designing the Study

In order to properly set up a user study, we need to discuss our goal and the parameters we want to test.

We already know the two parameters we want to investigate (shadow-map resolution and filter radius) and that our goal is to optimize them while preserving the perceptual quality of shadow silhouettes. In order to set up a proper user study, we need to consider other parameters that could influence our findings.

4.1.1 Parameter Space

In Chapter 3 we saw that shadow-mapping algorithms are a versatile tool, as they can be applied to almost every scene setup imaginable. The visual result on the screen depends on more parameters than simply a combination of shadow-map resolution and reconstruction-filter radius.

In the next paragraphs we will enumerate parameters which could potentially influence the visual appearance of filtered shadows.

Shadow-Map Resolution The shadow map is a discretized representation of the part of the scene as it is seen from the light-source's point-of-view. Its resolution corresponds to the initial-sampling frequency and it governs the amount of detail captured by the shadow map. The size of the projected undersampling artifact depends on the resolution, since it corresponds to a cell in the shadow maps's pixel grid.

Reconstruction-Filter Radius The filter radius governs the radius used for reconstructing a shadow signal from the discrete shadow map. It is relative to the size of the shadow frustum's view plane. If the filter radius is chosen big enough, sampling errors caused by the shadow map's discretization can be hidden (at the cost of increased blur).

Scene Setup We need to consider several important parameters related to the viewer's camera, placement and orientation of the light source, objects in the scene casting or receiving shadows etc. If we take a filtered shadow-map texel projected onto a surface for example, we would have to consider its orientation with respect to the viewer and its orientation with respect to the light source.

Contrast The shadow quality might also be affected by the hue of the surface's texture and the intensity of light reflected by the surface. If the contrast between the shadow's silhouette and the surrounding surface is high, artifacts become more emphasized.

4.1.2 Refining the Parameter Space

Before we can define the user study in detail, we need to determine which parameters are relevant.

Shadow-map resolution and filter radius are obviously indispensable, since we focus our investigation around them.

Since we want to analyze properties of filtering artifacts, we will approach our investigations by looking at the way filtered artifacts can be propagated through the shadow-mapping process.

Contrast and Scene Setup Changing the scene setup might impact the perceivability of shadow artifacts, as it can influence the projected artifact in the scene (by changing light source, shadow caster and/or shadow receiver position), the projection of the artifact onto the viewer's image plane (by changing the viewer's position) or the contrast of the produced artifacts (by changing the surface color and/or intensity of the light source).

So the problem can be separated into three parts. The first part involves the artifact projected from the light source into the scene, the second part how the artifact is projected onto the image plane of the viewer and the third part the contrast and color of the artifact.

1. Looking at the first part from an analytical standpoint, we make the following observation: The result of computing the shadow from a shadow map using filter-based approaches can be basically seen as a set of iso-contours representing the filtered hard shadow. Figure 4.2 shows an example of these iso-contours. While the artifact-free curve is perfectly straight, the artifact-prone image has an uneven curve, which results from discretized shadow-map pixels being filtered with a radius being slightly too small. The patterns formed from these iso-contours depend on the shape of the occluding objects, the angle of the light source and the structure of the shadow-receiving surface. Artifact-prone and artifact-free solutions will have different contour patterns and we argue that the user evaluates the dissimilarity in their curvature to identify artifacts. We can use the iso-contours as an indicator for how clean straight features will appear to the user. Since the orientation of viewer and receiving surface only affect the projection of these iso-contours, we assume that it does not make a difference to the observer, from which angle the projected, filtered shadow is viewed.

2. For the observer, a planar shadow receiver exposes insufficiently filtered artifacts the most. Imagine a shadow being cast on a stair: the uneven structure of the stair

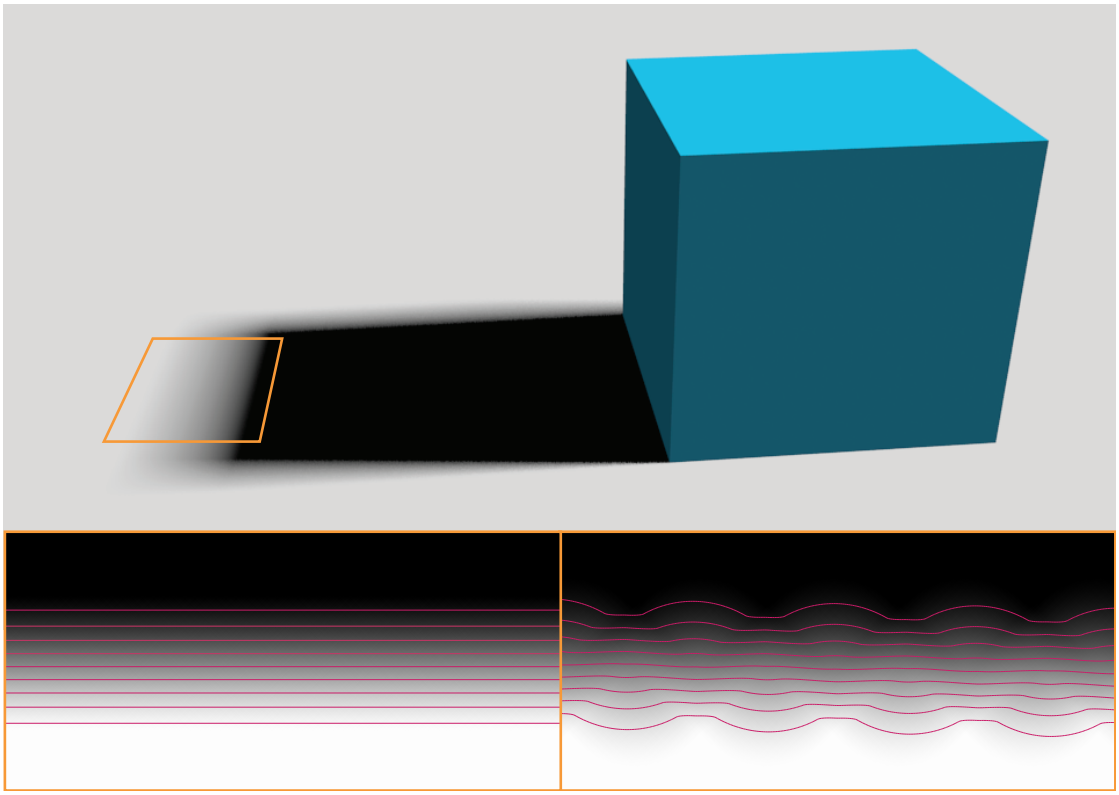


Figure 4.2: *A comparison of soft-shadow iso-contours produced by an artifact-free shadow map (left) and a shadow map with a regular artifact pattern (right).*

makes it harder to evaluate the shape of the cast object. Therefore we assume that it is sufficient to use a planar receiver for our experiment.

3. Since shadows represent the absence of light, we can safely assume that their color is very dark and close to black. We can make maximize an artifact's distinguishability by drawing a black shadow on a perfectly white surface.

Conclusion We argue that we do not need to consider an arbitrary scene setup for investigating the perceivability of filtered shadow-artifacts. It is sufficient to consider a simplified setup, where the viewer is parallel to a white surface onto which the filtered shadow is projected in black. If the relation between filter radius and shadow-map resolution is indeed linear (recall Figure 4.1), the projected iso-contours of the same artifact will always be proportional in size, no matter how the scene is set up.

4.1.3 Dependent and Independent Variables

Now that we were able to greatly simplify the problem, we have to sample the remaining parameters in a meaningful way.

Filter Size We treat the filter size as the dependent variable in our experiments. Our goal is to understand how much an artifact has to be blurred so users cannot recognize it anymore.

Artifact Size / Shadow-Map Resolution Using the scene considerations from Section 4.1.2, we see that the artifact size corresponds to a shadow-map texel and is therefore indirectly coupled to the shadow-map resolution. In order to confirm and identify the linear behavior we expect from the results, we need to test more than one artifact size. Selecting meaningful artifact sizes is actually not that trivial, as we have to consider that the monitors the experiments will be conducted on have a specific pixel resolution. In order not to influence the study, the screen resolution has to be much higher than the artifact size. Choosing the artifact size too big or too small can bias the user in his or her decision in whether the original stimulus actually was an artifact (e.g., if the artifact is below pixel size or so big that the filter necessary to hide it needs to be bigger than the screen). We therefore choose the minimal artifact size to be at least five pixels on the screen and at most 5% of the screen size (in our case 30 pixels). In-between we set two additional sample points at 10 and 20 pixels, resulting in four conditions in total (In the final study, the largest artifact sample was discarded due to time restrictions).

Silhouette Patterns The patterns formed by the shadow map depend on the angles of the object's silhouette on the shadow map. If a silhouette is horizontally or vertically aligned with the shadow map, artifacts are not visible. In the case of a diagonal 45 degree silhouette, artifacts are visible at regular intervals, to which we will refer as *stair pattern*. Cases in-between result in irregular or a mixture of irregular and regular patterns. We decided to investigate cases where a single artifact (which we will call *step pattern*) is generated and the 45 degree case. Figure 4.3 illustrates these patterns.

Light-Shadow Contrast The last independent variable we want to investigate is the contrast between lit and shaded areas. We decided to include the worst-case scenario, which is the contrast between completely black and white screen pixels. Additionally, we reduce the intensity of the lit part by 50% to have another sample case.

To summarize, we need to find the right filter size for all artifact size, artifact pattern and light-shadow contrast combinations (12 in total).

4.2 Task Definition

Finding the "right size" means that we need to investigate these physical stimuli and determine the thresholds for our dependent variable (the filter size) at which they are perceived as artifact-free. Such a threshold can be found by conducting an adaptive psychophysical experiment.

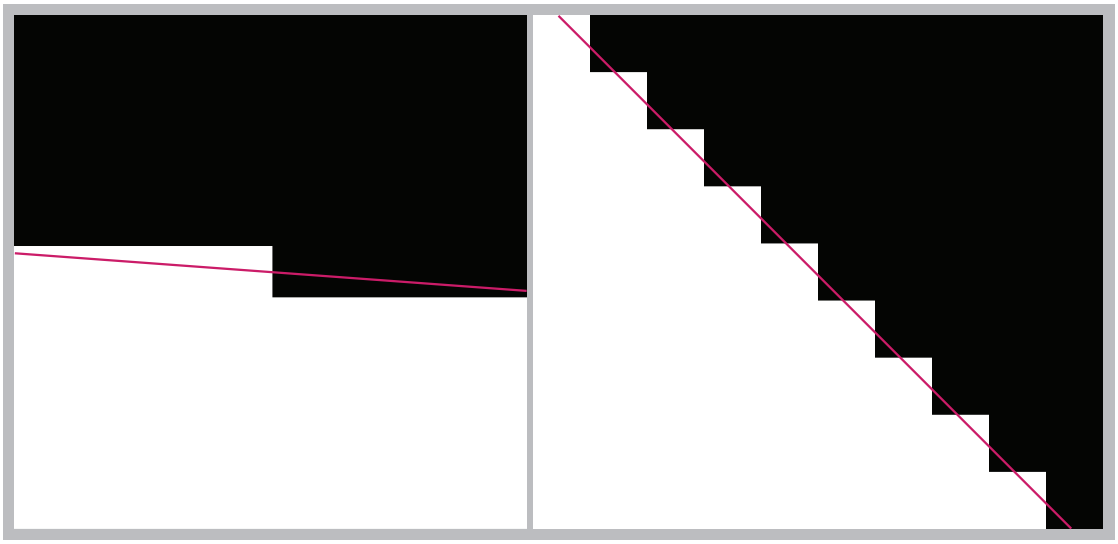


Figure 4.3: *An illustration of the two silhouette artifact patterns used in our user study. The left image shows a single* step*, the right image a regular* stair *pattern. The red lines represent the actual silhouette of the sampled geometry.*

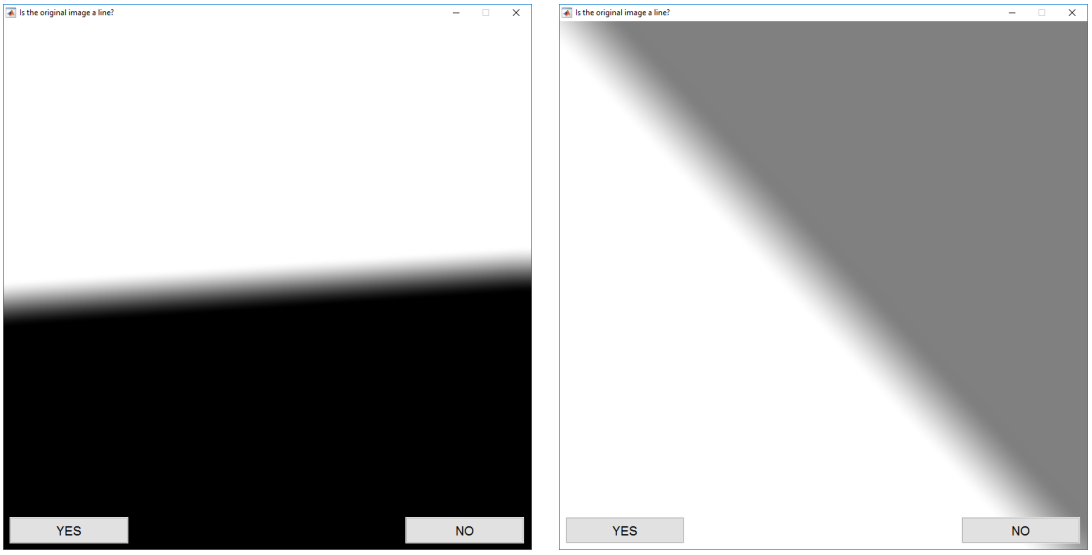
We decided to use the QUEST procedure [WP83] to find the threshold at which users can no longer infer from the filtered stimuli whether the original had artifacts in it or not. We used the Matlab Psycho Toolbox to control the QUESTs. The threshold guess was set to 3% of the artifact size, which is also used as an initial guess and the standard deviation guess. As a probability threshold we used 0.82. These parameters were optimized by the authors performing the experiment in advance. As recommended in [WP83], the *gamma* parameter was set to 0.5 (for a two-alternative forced choice), *delta* to 0.01 (the probability of a mistake by the user) and for *beta* we used 3.5 (controlling the slope of the psychometric function).

4.3 User Study Execution

Due to limited time and resources, the study was conducted with a relatively small population of ten users (nine male, one female). All of them were experts in computer graphics, aged 28.2 years on average (standard deviation 3.1 years).

Environment All participants were trialed under the same conditions. The trials took place in the students' laboratory at the *Institute for Computer Graphics and Algorithms*. During each trial the room was sealed off from sunlight and lit by dim light emitted by a common LED lightbulb. Each of the trials took place on the same monitor.

Software Interfaces We designed a minimalistic graphical user interface for study participants featuring the question "Is the original image a line?", a big picture of the



(a) User Screen High Contrast *(b) User Screen Low Contrast*

Figure 4.4: *Two examples of the tasks presented to the participants. The task "Was the original image a line?" is visible in the heading during the entire trial.*

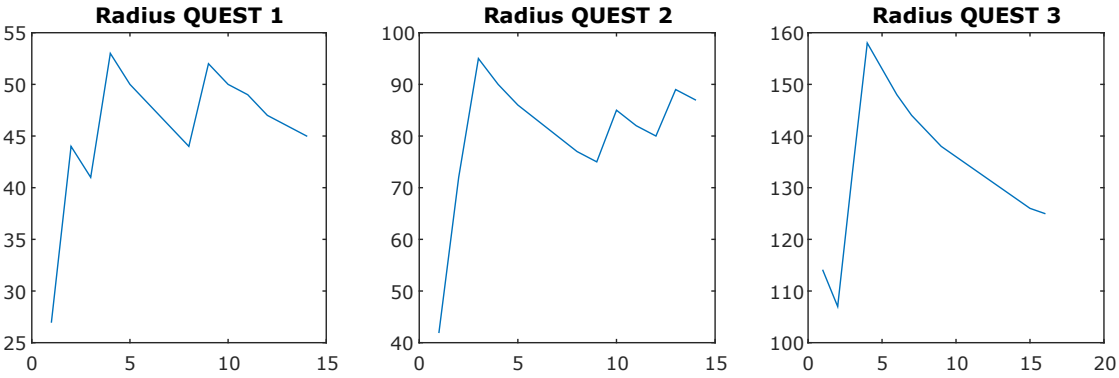
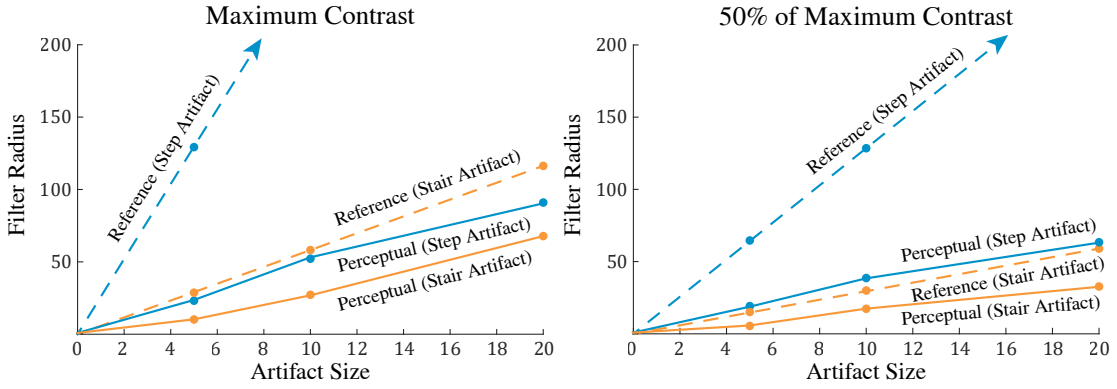


Figure 4.5: *The administrator's interface allows us to track progress of the study during the trials. The three charts indicate the status of convergence for each quest. In these charts the horizontal axes represent the number of trials which were already completed by the participant and the vertical axes represent the filter radius applied in each trial.*

trialed shadow contour and two clickable buttons for "yes" and "no" answers. The user interface for the study administrator was shown on a seperate monitor and was not visible to the participant. See Figure 4.4 and Figure 4.5.

Execution Trials were conducted in four rounds per participant. Each round tested for a different combination of artifact type (i.e., "stair" and "step") and shadow contrast (i.e., 100% and 50%). Per round, we observed three quests for three different artifact



(a) High Contrast *(b) Low Contrast*

Figure 4.6: *A comparison between reference filter size and perceptual filter size for the investigated artifact patterns (stair and step). The filter size necessary to hide artifacts from users (perceptual filter) is significantly lower than for the reference filter size. Reducing the contrast (b) makes it slightly harder to spot artifacts as can be seen in comparison to the results in Figure (a)*

sizes. In total a full trial lasted about 25 minutes.

During a trial the administrator carefully examined the progress of the three quests, and finished the round once all quests were tried at least 10 times each and all three quests converged.

4.4 User Study Evaluation

After all trials ended, mean filter radii for each combination of artifact type and artifact size were computed from the collected data. A graphical representation of the results is provided in Figure 4.6.

Based on the user study's results, we can analyze the impact of the different parameters on the optimal filter size.

Impact of Artifact Size As already indicated by our observations (recall Figure 4.1), the artifact size proportionally corresponds to the filter radius that is required to hide them. Figure 4.6 shows a dependence between filter size and artifact size that is nearly linear. While the stair artifacts of size 10 are hidden using a filter size of ~ 25 pixels or larger, the 20-pixel artifacts, on the other end, require a filter of at least 50 pixels.

Since the function is linear it reinforces our claim from Section 4.1.2 that the relation of artifact size to filter size can be evaluated independently of the scene setup: If the artifact seen by the observer on the screen requires a certain blur radius, an artifact that appears larger on the screen due to projective transformations requires a proportionally

larger blur radius, which, when projected back to shadow-map space, equates to the identical artifact size and blur radius. In other words, the filter size spans a constant amount of shadow-map texels. In our case, the filter radius is 3.47 shadow-map texels. It is important that the blur radii selected by the users correspond to the linear relationship implied by scene transformations, since this allows us to ignore changes in the scene setup, in particular the projected size of an artifact to the screen, and choose the filter radius solely depending on the shadow-map resolution, i.e., proportional to the texel size.

Impact of Patterns The study shows that the step pattern needs a larger filter size compared to the stair pattern to be hidden from the user. Figure 4.6 shows that the stair patterns of size 10 pixels are perceived as a straight contour when the filter size is at or above 25 pixels. Using the same filter size, single-step artifacts of the same size are still identifiable by the users. We assume that the regularity of the 45 degree pattern is beneficial to the user's perception of a straight contour.

Comparison to Reference Filter Size In order to measure the actual benefits of the perceptual approach, we need to compare it to a reference solution. We decided to use the same setup we employed in our user study, with the assumption that in the worst case, artifacts will still be noticed if at least one pixel differs from the expected outcome. In other words, if the rasterization of two filtered solutions, one with and one without artifacts (drawn using Bresenham's line algorithm at high resolution), produces the same image (assuming a typical 8-bit representation for intensities), even a perfect user will not be able to spot any artifacts. This corresponds to finding the minimal filter size where this condition is met. We will refer to this filter size as the *reference filter size*.

Impact of Light-Shadow Contrast Reducing the contrast between the lit- and the umbra region makes it slightly harder for the user to identify artifacts, as can be seen in Figure 4.6b.

In the next chapter we will use the data gathered by the user study to fit a linear function that describes the perceptually optimal relation between filter- and artifact size.

CHAPTER 5

Algorithm

In this chapter we are going to propose the steps necessary in order to integrate our findings into a real-time setup based on our own implementation. Furthermore, we will demonstrate visual results and benchmarks generated using our approach.

While the previous chapters investigated the impact of changes of given parameters on the optimal filter radius, real-time shadowing setups often need to approach the problem from the opposite direction, that is, to calculate the optimal parameters for a certain filter radius. Usually, designers want to define the visual appearance, i.e., the general softness, of a shadow by adjusting the filter radius (in case of PCSS, the designer adjusts the light size, which also affects the filter radius).

5.1 Integrating our Results into Shadow-Mapping Algorithms

5.1.1 Filter to Resolution Relation

Given an arbitrary scene, we have to assume the worst cases of artifact patterns and the worst cases of light-shadow intensity to appear, hence the only parameter left to find is the right artifact size. During shadow-map evaluation, the shadow map gets projected into the scene. If we recall our findings from Chapter 4, we can see that we are trying to maximize the projected shadow-map texels' size to a degree where we still get pleasant results after blurring it with a specific filter size. Since we are investigating undersampling artifacts, the artifact size is governed by the shadow map's sampling resolution (i.e., the initial sampling frequency).

Fitting a linear function to the results shown in Figure 4.6 (blue line) we get Equation 5.1, which allows us to calculate the optimal filter radius r for a shadow map by multiplying the artifact size a with the slope c of the linear function. More intuitively, this can be

interpreted as calculating the filter radius proportional to the texel size of the shadow map.

$$r = a \cdot c \quad (c = 3.47) \tag{5.1}$$

Based on our assumption from Section 4.1.2 that the function is independent of the scene (i.e., camera) setup, we can incorporate it into shadow mapping algorithms. In this section, we want to compute the shadow-map resolution for a user-defined filter radius, so we have to invert Equation 5.1.

There are two ways how to calculate the shadow-map resolution, depending on which space the filter radius is defined in:

Assuming that the desired blur radius r_{UV} is already defined in shadow-map texture space, we can calculate the artifact size a_{UV}:

$$a_{UV} = \frac{r_{UV}}{c} \tag{5.2}$$

Since we want to be able to hide one shadow-map texel, we substitute the relative artifact size for the reciprocal of the shadow-map's resolution, yielding equation 5.3:

$$s_{res} = \frac{1}{a_{UV}} = \frac{c}{r_{UV}} \tag{5.3}$$

If, on the other hand, the desired blur radius is defined in world space as r_{WS}, we can calculate the artifact size a_{WS} (in world space):

$$a_{WS} = \frac{r_{WS}}{c} \tag{5.4}$$

Using the extents of the light's view plane L_{WS}, we calculate the artifact size relative to it:

$$a_{rel} = \frac{a_{WS}}{L_{WS}} \tag{5.5}$$

$$s_{res} = \frac{1}{a_{rel}} = \frac{L_{WS}}{r_{WS}} c \tag{5.6}$$

Equations 5.3 or 5.6 can be directly incorporated into algorithms which support a constant filter size (such as PCF or VSM).

Soft-shadow algorithms require additional considerations which will be discussed in Section 5.1.3.

5.1.2 Shadow-Map Generation

Before computing the shadow map, we limit the view frustum of the shadow-casting light source. By doing so we can concentrate our shadow samples to surfaces which potentially cast shadows and get rid of redundant depth information.

The fitting algorithm described in Section 3.4.2 fulfills this purpose. However, there are a few things to consider when implementing shadow-map fitting:

Fitting scales the portion of the scene that is projected to the viewport. We want the filter radius to cover the same area of the scene as it would do without fitting. So we need to account for the scaling. We can do so by incorporating the scaling coefficients s_{xy} of the fitting matrix into our computations. Since the fitting matrix enlarges the portion of the viewport we need to scale the filter radius by multiplying it by s_{xy}:

$$r_{UV} = \frac{r_{WS}}{L_{WS}} s_{xy}$$

For soft shadows we need to adapt the light source's area in a similar fashion:

$$w_{Light} = \frac{w_{LightWS}}{L_{WS}} s_{x,y}$$

Later on, we can use w_{Light} to scale the filter radius independently from the light's actual view-plane size.

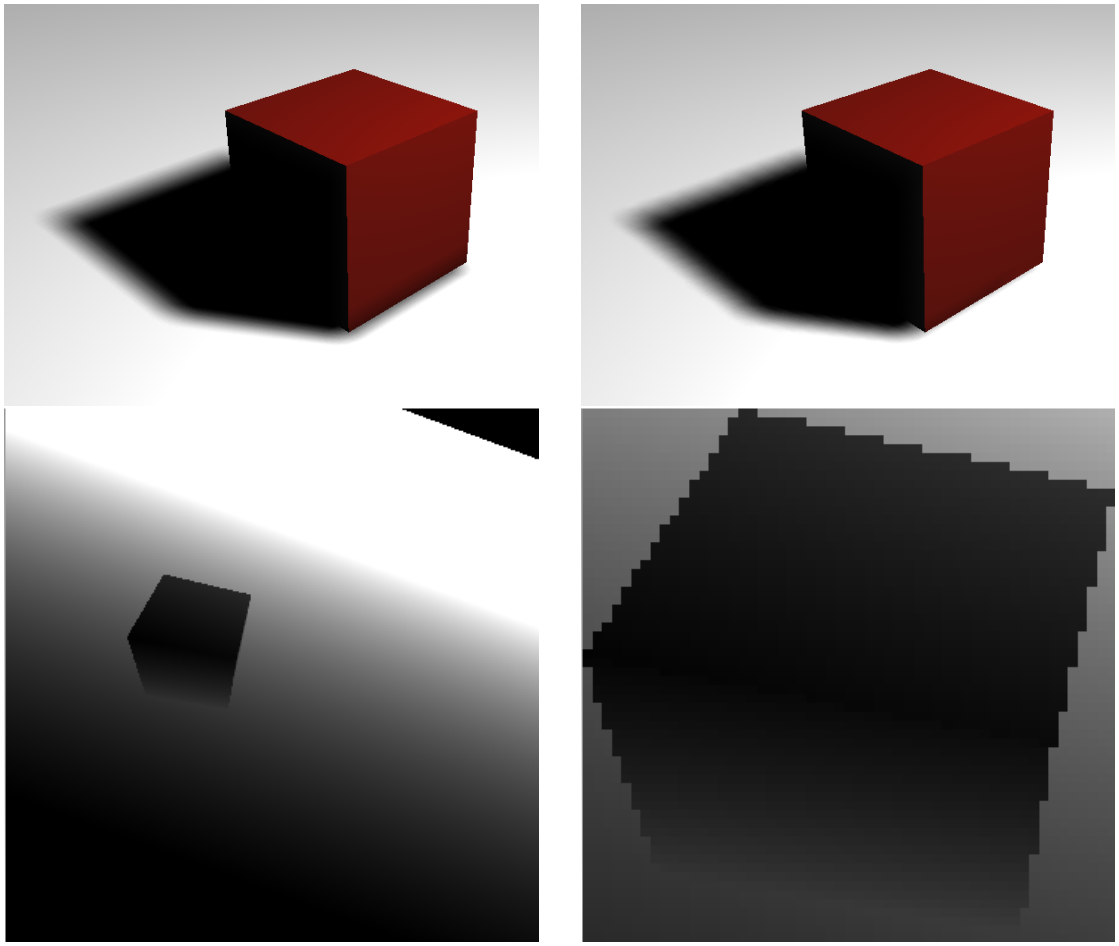
Figure 5.1 demonstrates the effects of shadow-map frustum fitting. Recall Equation 5.1: by increasing the filter radius, we can further reduce the shadow map's resolution. Therefore, fitting has great potential in combination with our method. These benefits can be seen in Figure 5.1, where the shadow-map size for the fit frustum is substantially lower than in the unfit case. The visual result is very similar despite the reduced resolution.

The scene can now be rasterized using the previously defined adaptive view frustum. The target framebuffer's depth attachment then holds the shadow map. Its depth values range from 0 to 1, where 0 is mapped to the near plane and 1 is mapped to the far plane.

5.1.3 Shadow-Map Evaluation

Now that the shadow map contains the sampled distances from the light source, the scene is drawn from the viewer's camera. As explained in Section 3.3.3, we additionally need to project each vertex into shadow-map space using the view projection matrix from the previous section and shift the resulting values into the same $[0, 1]$ interval as the shadow-map values are ranging in.

During fragment shading, the perspective division (Equation 5.7) is performed to map the fragment into shadow map space.



(a) *unfit at 237x237 px* ***(b)*** *fit at 55x60 pixels*

Figure 5.1: *We use fitting to focus the shadow map on relevant geometry only. By doing so the filter radius grows with respect to the near plane size. This allows us to further reduce the shadow map's resolution.*

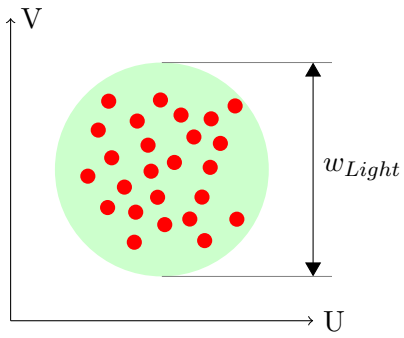


Figure 5.2: *When* Poisson disk sampling *is used, sampling points are distributed inside a circle. Randomly rotating the disks per fragment adds noise to our filter, which further obfuscates sampling artifacts.*

$$P_{shadow} = \frac{P_{shadow}.xyz}{P_{shadow}.w} \tag{5.7}$$

At this point, a simple binary shadow evaluation could already be performed by comparing the fragment's projected shadow map space depth P_{shadow} to the corresponding value stored inside the shadow map.

Percentage Closer Filtering

The results obtained by Percentage Closer Filtering depend on the filter size, which is governed by the desired blur width (corresponds to light with r_{UV} or r_{WS} defined in Section 5.1.1) and the sampling kernel used.

A common approach to reduce the amount of shadow-map samples needed to compute smooth shadow contours is to utilize randomly rotated Poisson disk kernels (an example is shown in Figure 5.2). We will shortly discuss why we expect our findings to be usable in Poisson disk based algorithms as well.

Let's first consider that Poisson disk sampling introduces noise in the soft shadow, which makes it harder for the user to perceive the iso-contours discussed in Section 4.1.1 (the noise obfuscates the contours). We therefore expect our findings to be compatible with such algorithms, as artifacts should be even less noticeable when they are used.

We used a sample size of 25 to produce the results shown in Figure 5.7 and Figure 5.8.

Percentage Closer Soft Shadows

We also aimed to apply our adaptive resolution to the Percentage Closer Soft Shadows (PCSS) algorithm. The PCSS algorithm determines the shadow's penumbra size by relating the distance from an area light source to the surface point blocking the light and the point on the surface receiving the light as shown in Figure 3.19b. Since the blocker

distance is unknown, it has to be estimated using the mean distance from samples taken from the shadow map inside a potential blocker range as shown in Figure 3.19a. In order to achieve this, distances inside a region of r_{search} (see Equation 5.8) have to be averaged as show in Equation 5.9.

$$r_{search} = w_{Light} \frac{z_{Receiver} - z_{Near}}{z_{Receiver}} \tag{5.8}$$

$$z_{Blocker} = \frac{\sum z_{Sample}}{n_{Samples}} \tag{5.9}$$

The resulting average blocker distance $z_{Blocker}$ can be used to estimate the penumbra radius $r_{Penumbra}$ (see Equation5.10) for the point on the receiving surface.

$$r_{Penumbra} = r_{Light} \frac{z_{Receiver} - z_{Blocker}}{z_{Blocker}} \tag{5.10}$$

Subsequently, the amount of light received can be determined conventionally by applying PCF with a radius of r_{filter} as shown in Equation 5.11.

$$r_{Filter} = r_{Penumbra} \frac{z_{Near}}{z_{Receiver}} \tag{5.11}$$

Due to the penumbra size being variable, our optimizations cannot be applied without considering some caveats. Consider the typical case of a three-dimensional object being placed on a flat surface and a light source that shines light in a slanted angle to the surface. The shadow would start out with an infinitesimally small penumbra and grow with increasing distance. Obviously, our algorithm cannot deal with a penumbra radius tending towards zero because the resolution would skyrocket towards infinity. By limiting the penumbra radius using a minimum threshold, we are still able to apply our optimizations. This has the downside of losing sharp shadow silhouette features in those situations where surfaces are very close to each other and especially when the angle between light source and shadow receiver is very shallow.

We can also compute the truly required minimum resolution using the *AtomicMin* operations supported by modern GPUs to evaluate Equation 5.12 on a per-pixel basis.

$$res_{Minimal} = \max_{Fragments} \left| \frac{c}{r_{Filter}} \right| \tag{5.12}$$

5.2 Evaluation Framework

In order to conduct a practical evaluation of our observations, we implemented a real-time rendering environment and applied Equation 5.1 to dynamically resize the resolution of a shadow map. We demonstrate the effectiveness of the reduced shadow-map resolution by applying PCF as well as PCSS filtering.

5.2.1 Software Requirements

The software was written in Visual C++ 14 with the following aspects in mind:

Scene Content The application needs to offer functionality to load meshes and textures, cameras and lights from file. We achieved this by using well-established third-party software. Models and materials were created using the modeling software *Blender* [Ble15]. *Assimp* [AG15] was used for reading the data produced by Blender in our framework.

Shadow Mapping In order to test our algorithm, the application needs to support custom shadow-map filtering algorithms (i.e., none, PCF, PCSS, VSM). We therefore implemented a custom rendering engine utilizing the OpenGL 4.4 API.

User Interface Mouse and Keyboard can be used for interactive placement and orientation of cameras and light sources. The *AntTweakBar* library [Dec13] is used to supply a user interface for changing the filtering algorithm and for tweaking its parameters.

1. Additional light-source parameters
    - Enable or disable shadow-map fitting.
    - Select which shadow-mapping algorithm to use.
    - Manipulate parameters specific to the selected shadow-mapping algorithm (i.e., blur radius, light size, etc.).
2. Data visualization
    - Display current shadow-map resolution.
    - Run benchmark (including average frame-generation timings) and print the results to a file.
    - Overlay a visualization of the shadow-map contents.

Benchmarking Using Opengl timer queries, the application offers an accurate benchmark environment to measure GPU computation times and cache efficiency.

5.2.2 Application Showcase

The application we developed includes all the functionality discussed above. Figure 5.3 shows a screenshot of the actual evaluation framework.

All 3D-rendered images depicting real-time shadows used in this thesis were generated using this application.

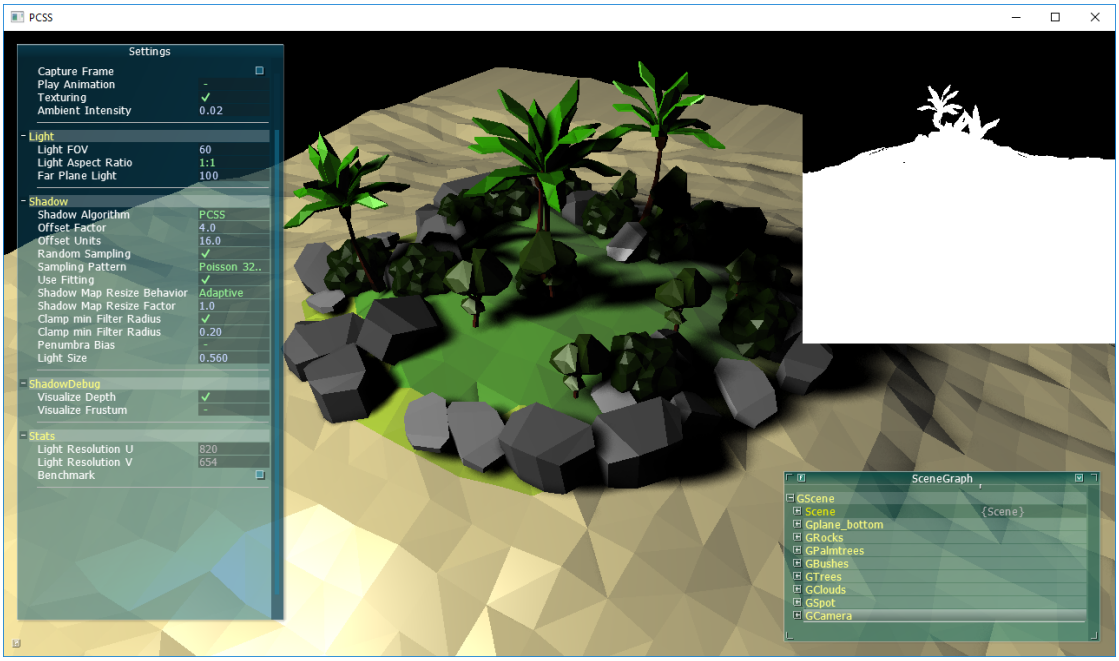


Figure 5.3: *Screenshot of the application used to evaluate our algorithm. The user interface offers access to all the important parameters.*

5.2.3 Quality and Performance Evaluation

Results rendered with the new algorithm are shown in Figures 5.7 and 5.8. For each filtering method we show two situations, in particular small and large filter size, and compare the reduced shadow-map resolution rendering to the unreduced reference rendering side-by-side. While the results look almost identical, we observe an increase in frame-rates on a Geforce GTX 960 GPU by up to 100%.

Diagram 5.4 shows detailed GPU computation times (times for shadow-map generation and times for drawing the image) at different resolutions and radii for our algorithm.

We can observe an overall improvement in computation times using the reduced resolution in our prototype, especially during the shadow evaluation. This benefit also increases for larger filter sizes. In the following paragraphs, we will analyze the causes for the performance gains.

Shadow-mapping computations take place almost entirely on the dedicated graphics processing hardware (a.k.a. the GPU). The sole responsibility of the client program running on the CPU is issuing memory allocations, draw calls and to set parameters on the GPU. This narrows the investigation of performance impact down to processes run on the GPU.

The first task issued to the GPU by our algorithm is to allocate memory for storing the shadow map. For PCSS, we expect our shadow-map size to vary for dynamic scenes

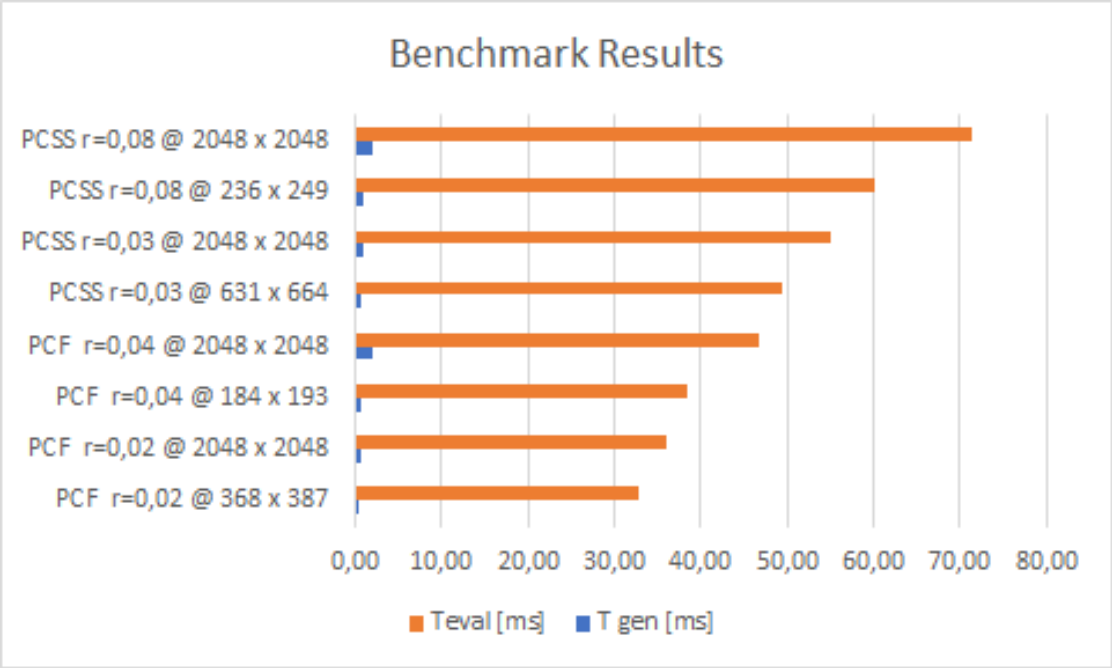
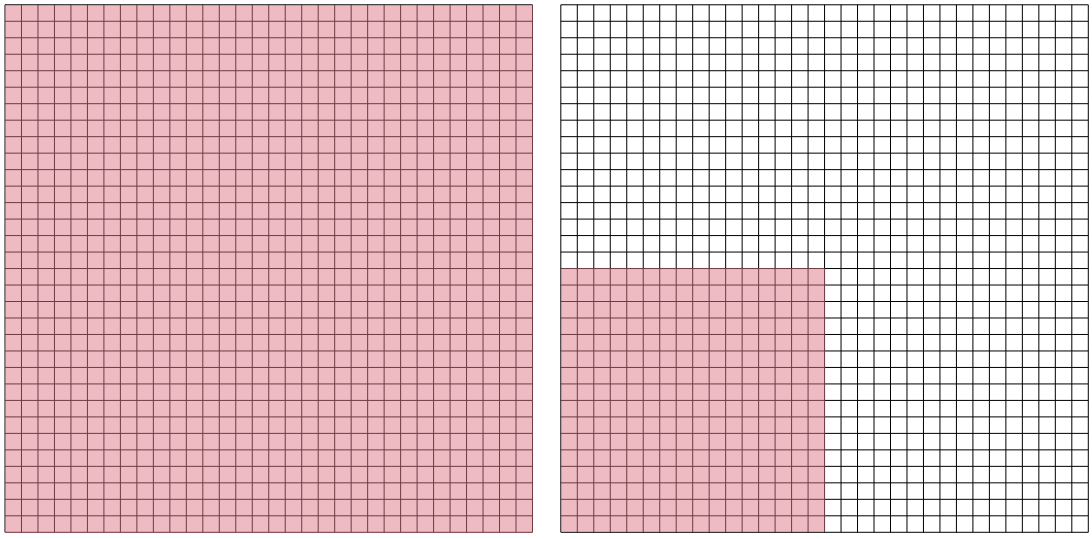


Figure 5.4: *Detailed benchmark results for computing the images depicted in Figures 5.7 and 5.8 from a slightly different light angle. Notice, that due to the fitting algorithm and a different positioning of the light source, the resolutions slightly differ from the results in figures 5.7 and 5.8 A 25-tap Poisson-disk kernel was used in all instances. Since timings slightly differ from one frame to another, the timings of 10 sequential frames were averaged.*

because the receiver and blocker distances change each time the camera of either the viewer or the light source is moved. If the fitting algorithm is used, frequent changes in the shadow map's resolution are expected for PCF as well. This means that the memory footprint of the shadow map potentially changes from one frame to another. Theoretically one could reallocate shadow-map memory each time the PCF filter radius changes. For real-time applications, however, we prefer processing performance over memory consumption and therefore we allocate a generous amount of memory beforehand and just fill up the chunk of data as needed. Figure 5.5 demonstrates how we adjust the shadow-map resolution by using a portion of the reserved memory.

Because of the preliminary memory allocation, we would not actually gain memory that could be used otherwise by resizing the shadow map. Smaller shadow maps can theoretically be computed faster than larger ones, because less fragment shader operations are needed when they are generated. Although the same amount of operations are needed during the evaluation stage, the key benefit of a smaller shadow map lies in the PCF-filtering step (plus blocker search for PCSS). The PCF and PCSS filter samples and averages multiple shadow-map texels per fragment. This causes lots of memory



Full memory footprint *Optimized memory footprint*

Figure 5.5: *Reducing the resolution of a shadow map potentially yields considerable performance benefits. When a shadow map is generated, a shader routine has to be issued for each pixel in the shadow map. By reducing the size of the shadow map, we reduce the number of pixels to be written into the buffer and therefore reduce the number of shader invocations.*

transfers on the GPU's memory-pipeline. When a memory fetch is requested from a certain shadow-map location, the GPU transfers not only the one texel requested, but additionally fetches a whole bunch of neighboring texels. This process is called *texture caching* and is designed to offer fast access to adjacent values, which is exactly what we need for filtering applications (see Figure 5.6).

This caching strategy assumes that multiple fragments that are processed by the GPU in parallel request data from memory regions in close proximity. Here lies the benefit of small buffers. We can store relevant data densely and thereby increase the cache efficiency and reduce memory transfers (see Figure 5.6).

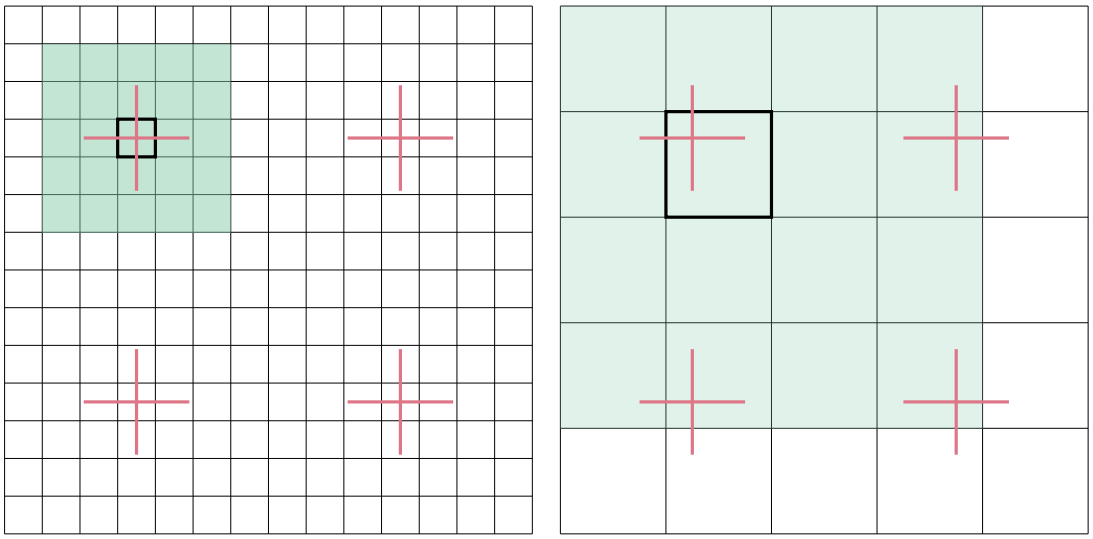
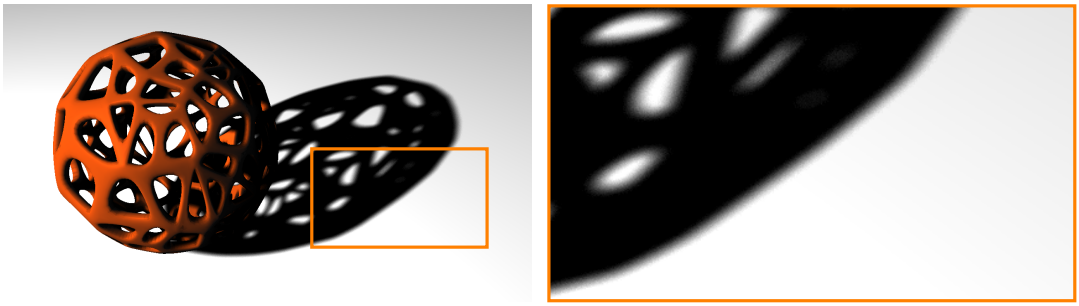
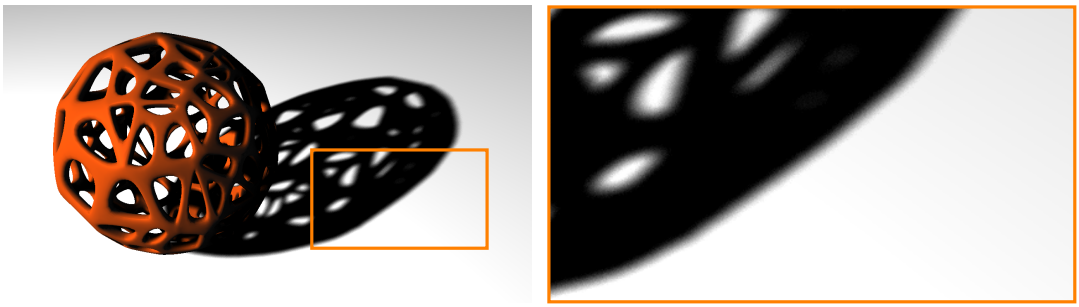


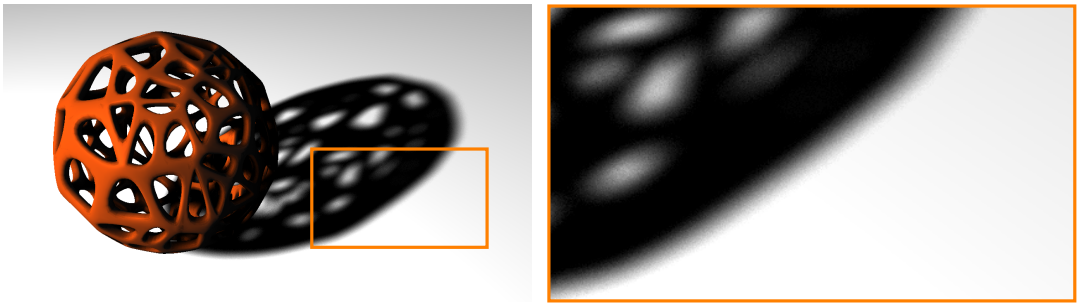
Figure 5.6: *During the shadow-evaluation stage, shader routines constantly fetch samples from the shadow map. The GPU internally manages cached memory (usually a block of adjacent pixels in a 2D buffer), which allows a much faster access to the required data. Performance is drastically increased when cached data is hit more often. Therefore, reducing the shadow map's density, we can increase the probability of cache hits because the chance of a requested sample being adjacent to an already cached sample is higher.*



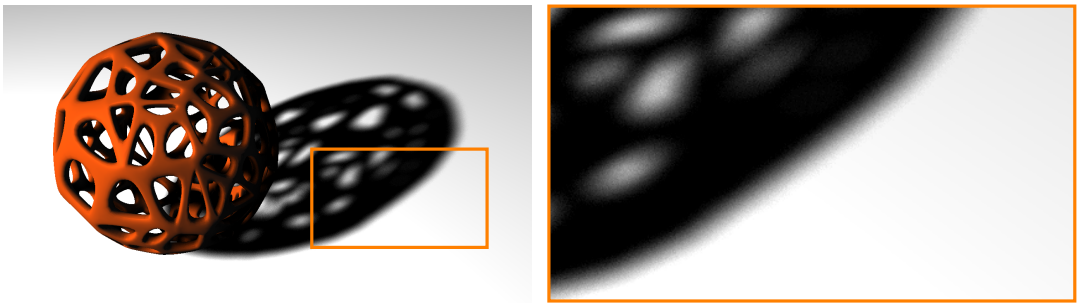
PCF 2048 × 2048, 0.44*ms and* 7.47*ms*



PCF 368 × 387, 0.17*ms and* 3.62*ms*

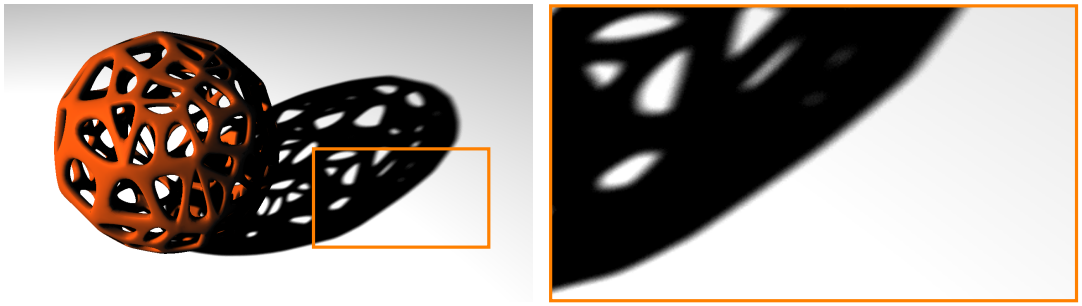


PCF 2048 × 2048, 0.40*ms and* 10.94*ms*

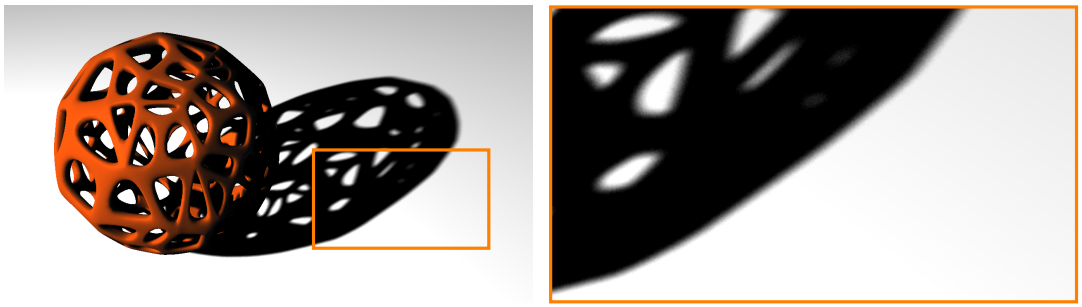


PCF 184 × 193, 0.20*ms and* 3.59*ms*

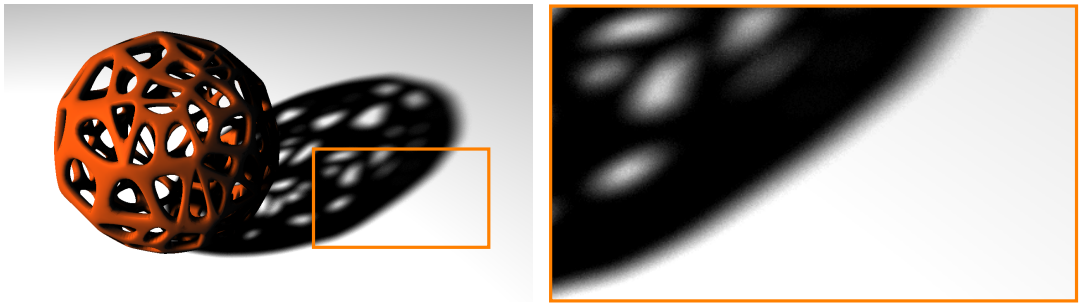
Figure 5.7: *Side-by-side comparison of the same scene rendered with large and reduced shadow-map resolution using two different blur sizes. The shadow-map resolutions and achieved frame rates can be seen in the respective captions.*



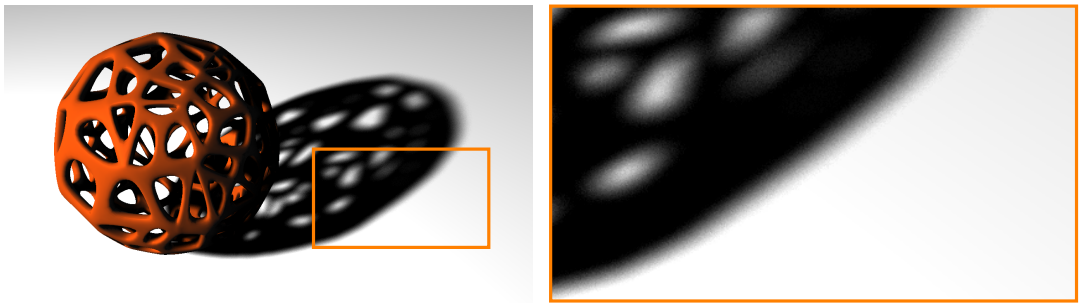
PCSS 2048 × 2048, 0.48*ms and* 8.51*ms*



PCSS 734 × 773, 0.23*ms and* 4.01*ms*



PCSS 2048 × 2048, 0.47*ms and* 10.68*ms*



PCSS 271 × 285, 0.14*ms and* 3.83*ms*

Figure 5.8: *Side-by-side comparison of the same scene rendered with large and reduced shadow-map resolution using two different light sources with different surface area. The shadow-map resolutions and achieved frame rates can be seen in the respective captions.*

CHAPTER 6

Conclusion

6.1 Synopsis

In this thesis, we demonstrated how shadow filtering can be exploited to hide shadow mapping related artifacts.

Shadow artifacts appear due to various reasons and are influenced by many parameters (such as position, orientation and material of the surface, position, orientation and intensity of the light source, position and orientation of the viewer, shadow-map resolution, filter size). We were able to break down the huge parameter space, which can be hardly investigated in a user study, into a simplified version consisting of four features that impact the shadow's perceptual quality. Reducing the complex feature space of shadow perception allowed us to design a user study which helped us to find out at which point filtered artifacts become unnoticeable to users.

By interpreting the results of the user study, we were able to describe the connection between shadow filter width and shadow-map resolution from a perceptual point-of-view. The resulting linear function can be used in practical shadow-mapping setups. It can be used to dynamically adjust shadow-map resolution in real time, or to calculate a feasible shadow-map resolution tailored to a desired penumbra width. By reducing the number of depth samples in a shadow map, we can increase the performance of shadow-map generation, since there are fewer fragments to process. Also, the performance of shadow-map evaluation is increased, because reducing the number of depth samples also increases the performance of shadow lookups, since it increases cache efficiency because shadow samples are tightly packed. Due to its generic nature, our method is adaptable and can be applied to several existing shadow-mapping algorithms (such as PCF, PCSS, VSM, CSM, etc.).

In order to demonstrate our method's effectiveness, we applied it to common shadow filtering algorithms. We offer a detailed description of a practical implementation. We

were able to save resources by using our perceptually optimized algorithm (as can be seen by comparing the shadow-map resolutions and *fps*-timings in Figures 5.7 and 5.8).

6.2 Limitations and Future Work

Although our implementation shows promising results, there are some noticeable limitations.

In cases where the blur width is large, high-frequency geometry details might be omitted. An example is shown in Figure 6.1. While this loss in detail is partially owed to the low-pass filtering characteristic of the blur, the reduced sampling-frequency might miss entire details such as small gaps in the geometry.

In case of PCSS, our method also requires the programmer to define lower bounds for the penumbra size. In cases where the penumbra width is very small, i.e., for contact shadows, the penumbra width has to be enlarged, because otherwise the required resolution would tend to be infinitely large. One possible solution could be the adaptive light-source subdivision approach, described by Schwärzler et al. [SMSW12]. They dynamically subdivide the light's view plane and compute multiple shadow maps of varying sampling density to cover it. While this approach would cover the problems on contact shadows, it is not well suited for real-time applications.

Due to limited resources, we conducted our user study on a small group of ten expert users. However, we do not expect significant differences to our linear fitted function if more users were to be questioned.

Adjusting the shadow map's resolution in-between frames introduces problems with temporal coherence, making shadow silhouettes appear wobbly during animations. We tried to alleviate this problem by restricting the rate of resolution change over time.

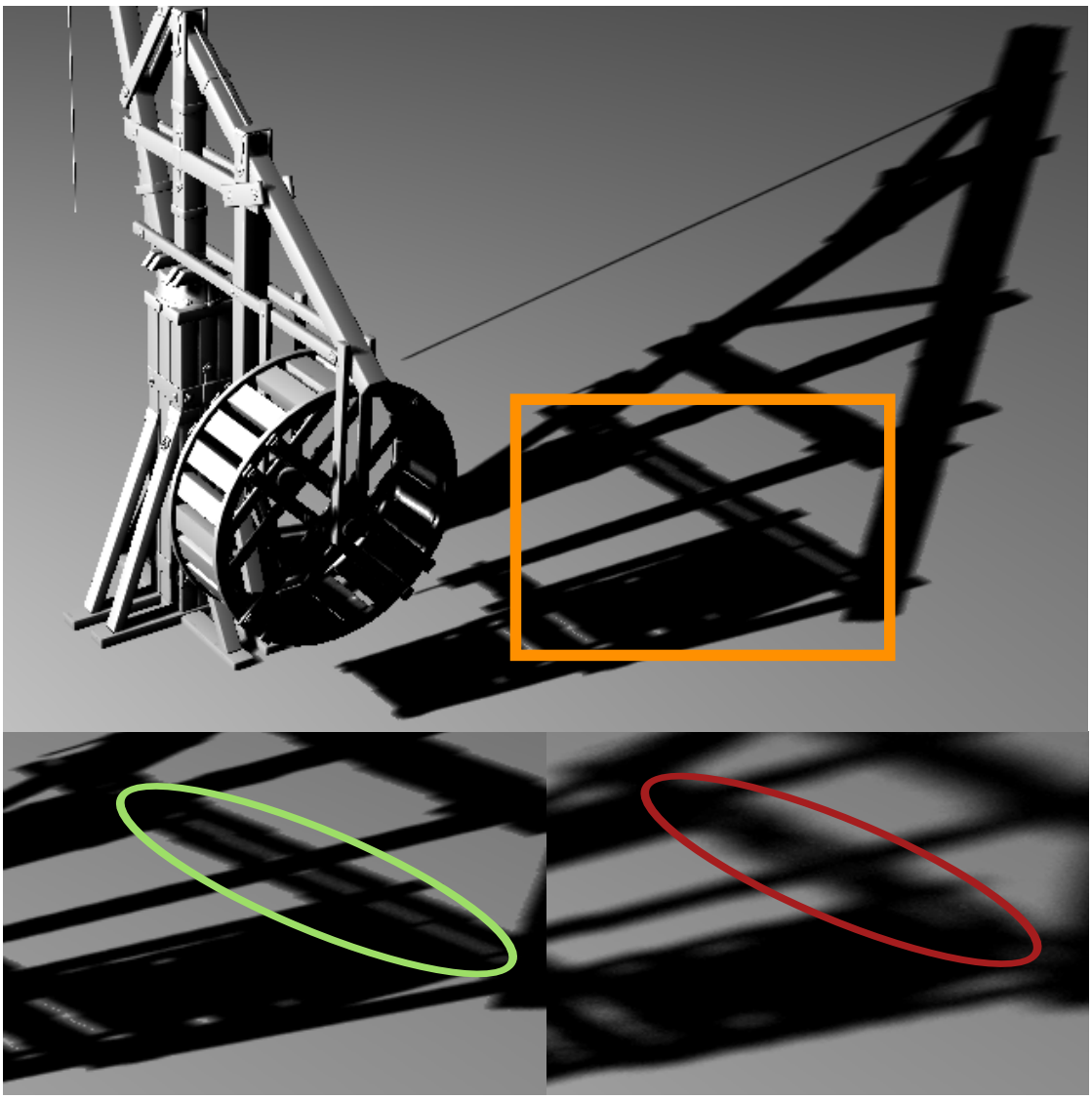


Figure 6.1: *Lowering the resolution might have the unwanted side-effect of detail being omitted. In the lower-left detail, the shadow map was sampled at a higher resolution than in the lower-left detail. The gap in the crane's geometry features high-frequency detail, which gets lost when using a larger blur radius and a lower sampling frequency.*

Bibliography

[AAM03] Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (TOG)*, 22(3):511–520, 2003.

[AG15] Kim Kulling David Nadlinger Alexander Gessler, Thomas Schulze. Assimp - open asset import library. `http://www.assimp.org/`, 2006-2015.

[AHL+06] Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Charles Hansen, and François X Sillion. Soft shadow maps: Efficient sampling of light source visibility. In *Computer Graphics Forum*, volume 25, pages 725–741. Wiley Online Library, 2006.

[AMB+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 51–60. Eurographics Association, 2007.

[AMS+08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proceedings of graphics interface 2008*, pages 155–161. Canadian Information Processing Society, 2008.

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM, 1968.

[ASK06] Barnabás Aszódi and László Szirmay-Kalos. Real-time soft shadows with shadow accumulation. *Short Paper Eurographics*, 2, 2006.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. *Journal of Graphics Tools*, 7(4):9–18, 2002.

[Bav08] Louis Bavoil. Advanced soft shadow mapping techniques. In *Presentation at the game developers conference*, volume 2008, page 11, 2008.

[BK70] J Bouknight and K Kelley. An algorithm for producing half-tone computer graphics presentations with shadows and movable light sources. In *Proceedings*

of the May 5-7, 1970, spring joint computer conference, pages 1–10. ACM, 1970.

[Ble15] Blender Online Community. Blender - a 3d modelling and rendering package. `http://www.blender.org/`, 2015.

[Bli77] James F Blinn. Models of light reflection for computer synthesized pictures. In *ACM SIGGRAPH Computer Graphics*, volume 11, pages 192–198. ACM, 1977.

[Bli88] Jim Blinn. Me and my (fake) shadow. *IEEE Computer Graphics and Applications*, 8(1):82–86, 1988.

[Coo86] Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, 1986.

[Cro77] Franklin C Crow. Shadow algorithms for computer graphics. In *Acm siggraph computer graphics*, volume 11, pages 242–248. ACM, 1977.

[Dec13] Philippe Decaudin. Anttweakbar. `http://anttweakbar.sourceforge.net/`, 2005-2013.

[DH06] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Transactions on Graphics (TOG)*, 25(3):503–508, 2006.

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165. ACM, 2006.

[Eng06] Wolfgang Engel. Cascaded shadow maps. *Shader X5-Advanced Rendering Techniques*, pages 197–206, 2006.

[ESAW11] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-time shadows*. CRC Press, 2011.

[FBP06] Vincent Forest, Loïc Barthe, and Mathias Paulin. Realistic soft shadows by penumbra-wedges blending. In *SIGGRAPH/EUROGRAPHICS Conference On Graphics Hardware: Proceedings of the 21 st ACM SIGGRAPH/Eurographics symposium on Graphics hardware: Vienna, Austria*, volume 3, pages 39–46, 2006.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, page 35. ACM, 2005.

[GBP06] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by backprojection. In *Rendering techniques*, pages 227–234, 2006.

[GBP07] Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. High-quality adaptive soft shadow mapping. In *Computer Graphics Forum*, volume 26, pages 525–533. Wiley Online Library, 2007.

[HBM+14] Michael Hecher, Matthias Bernhard, Oliver Mattausch, Daniel Scherzer, and Michael Wimmer. A comparative perceptual study of soft-shadow algorithms. *ACM Transactions on Applied Perception (TAP)*, 11(2):5, 2014.

[HLHS03] J-M Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Computer Graphics Forum*, volume 22, pages 753–774. Wiley Online Library, 2003.

[Jon06] Thouis R Jones. Efficient generation of poisson-disk sampling patterns. *Journal of graphics, gpu, and game tools*, 11(2):27–36, 2006.

[LD08] Ares Lagae and Philip Dutré. A comparison of methods for generating poisson disk distributions. In *Computer Graphics Forum*, volume 27, pages 114–129. Wiley Online Library, 2008.

[LGQ+08] D Brandon Lloyd, Naga K Govindaraju, Cory Quammen, Steven E Molnar, and Dinesh Manocha. Logarithmic perspective shadow maps. *ACM Transactions on Graphics (TOG)*, 27(4):106, 2008.

[LM07] Andrew Lauritzen and Michael Mccool. Summed-area variance shadow maps. In *GPU Gems*. Citeseer, 2007.

[LM08] Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *Proceedings of graphics interface 2008*, pages 139–146. Canadian Information Processing Society, 2008.

[LTYM06] Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Rendering Techniques*, pages 215–226, 2006.

[MF92] Michael McCool and Eugene Fiume. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface*, volume 92, pages 94–105, 1992.

[MT04] Tobias Martin and Tiow Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. *Rendering techniques*, 2004:15th, 2004.

[Mur12] Haruki Murakami. *1Q84*. Random House, 2012.

[PH04] Matt Pharr and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.

[Reg04] Ashu Rege. Shadow considerations. *NVIDIA Developer Technology Group*, 2004.

[RSC87] William T Reeves, David H Salesin, and Robert L Cook. Rendering antialiased shadows with depth maps. In *ACM Siggraph Computer Graphics*, volume 21, pages 283–291. ACM, 1987.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 557–562. ACM, 2002.

[SMSW12] Michael Schwarzler, Oliver Mattausch, Daniel Scherzer, and Michael Wimmer. Fast accurate soft shadows with adaptive light source sampling. In *Proceedings of the 17th International Workshop on Vision, Modeling, and Visualization (VMV 2012)*, pages 39–46. Eurographics Association, nov 2012.

[SS07] Michael Schwarz and Marc Stamminger. Bitmask soft shadows. In *Computer Graphics Forum*, volume 26, pages 515–524. Wiley Online Library, 2007.

[WH03] Chris Wyman and Charles D Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Rendering techniques*, pages 202–207, 2003.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM, 1978.

[Wil83] Lance Williams. Pyramidal parametrics. In *Acm siggraph computer graphics*, volume 17, pages 1–11. ACM, 1983.

[WP83] Andrew B. Watson and Denis G. Pelli. Quest: A bayesian adaptive psychometric method. *Perception & Psychophysics*, 33(2):113–120, 1983.

[WPF90] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *Computer Graphics and Applications, IEEE*, 10(6):13–32, 1990.

[WS06] Michael Wimmer and Daniel Scherzer. Robust shadow mapping with light space perspective shadow maps. In Wolfgang Engel, editor, *ShaderX 4 – Advanced Rendering Techniques*, volume 4 of *ShaderX*. Charles River Media, March 2006.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. *Rendering Techniques*, 2004:15th, 2004.

[ZSXL06] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 311–318. ACM, 2006.