Shadow Algorithms for Real-time Rendering

Eurographics 2010 Tutorial Notes

Elmar Eisemann
Télécom ParisTech/Saarland University/MPI Informatik

Ulf Assarsson
Chalmers University

Michael Schwarz
MPI Informatik

Michael Wimmer
Vienna University of Technology

About the Authors

Elmar Eisemann

Associate professor, Télécom ParisTech, France

Before being an associate professor at Télécom ParisTech, Elmar Eisemann was a senior scientist in the Cluster of Excellence (MMCI), Saarland University/Max-Planck-Institute, Germany and head of the research group ECLEXIS until December 2009. He has a “Vordiplom” in Mathematics from Cologne University and studied at the École Normale Supérieure Paris (2001). He obtained Master (2004) and PhD. (2008) in Mathematics/Computer Science from Grenoble Universities, advised by Xavier Décoret and Francois Sillion at INRIA. He worked abroad and collaborated with Frédo Durand (MIT, 2003), John C. Hart (UIUC, 2006), David Salesin (Adobe (Seattle), 2007) and Sylvain Paris (Adobe (Boston), 2008).

Ulf Assarsson

Associate professor, Department of Computer Science and Engineering, Chalmers University of Technology, Sweden

Ulf Assarsson received his M.Sc. in Engineering Physics in 1997 and Ph.D. in Computer Graphics in 2003, and he is now head of a research group focusing primarily on real-time and non-real-time soft shadows as well as ray tracing, GPU-techniques and global illumination.

Michael Schwarz

Postdoc, MPI Informatik, Germany

Michael Schwarz received a Diploma in 2005 and a Ph.D. in 2009 from the University of Erlangen-Nuremberg. His research interests include real-time computer graphics, GPU techniques, global illumination, level-of-detail approaches and perception-aware graphics.

Michael Wimmer

Associate professor, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria

Michael Wimmer received an M.Sc. in 1997 and a Ph.D. in 2001. His current research interests are real-time rendering, computer games, real-time visualization of urban environments, point-based rendering and procedural modeling. He has coauthored many papers in these fields, and was papers co-chair of EGSR 2008.

General Information

Abstract

Shadows are crucial for enhancing realism and provide important visual cues. In recent years, many important contributions have been made both for hard shadows and soft shadows. Often spurred by the tremendous increase of computational power and capabilities of graphics hardware, much progress has been made concerning visual quality and speed, making high-quality real-time shadows a reachable goal. But with the growing wealth of available choices, it is particularly difficult to pick the right solution and assess shortcomings. Because currently there is no ultimate approach available, algorithms should be selected in accordance to the context in which shadows are produced. The possibilities range across a wide spectrum; from very approximate but really efficient to slower but accurate, adapted only to smaller or only to larger sources, addressing directional lights or positional lights, or involving GPU- or CPU-heavy computations. This tutorial tries to serve as a guide to better understand limitations and failure cases, advantages and disadvantages, and suitability of the algorithms for different application scenarios. We will focus on real-time to interactive solutions but also discuss offline approaches if needed for a better understanding.

In the future, we will provide you with more information via our webpage:
`http://www.mpi-inf.mpg.de/resources/ShadowCourse/`.

Prerequisites

Notions of geometry and linear algebra are of advantage. Some working knowledge about GPU programming is helpful to use the presented algorithms in practice, but the tutorial will also be informative for people with very basic GPU experience.

Contents

Overview

This document is intended to provide a deep understanding of real-time shadow algorithms. It contains descriptions of the most common techniques, gives advice and provides the theoretical background of this topic. In general, only few prerequisites are needed to understand the content. Some notions of linear algebra, calculus, and simple knowledge of hardware and programming should be sufficient to understand the text.

The document is structured as follows:

Chapter 1 will concentrate on the mathematical definitions. This is important because current literature often uses many terms in ambiguous ways. Therefore, for instance, the frequently employed attribute "physically-based" does not allow any deductions on the behavior or quality of the algorithm.

In order to provide a clean classification, we will show what approximations are usually done when computing soft shadows and propose a nomenclature that allows distinguishing between various algorithms.

We will then analyze different kinds of shadow algorithms. Starting by laying a groundwork with the most basic and widely spread shadow methods in Chapter 2, we continue with hard shadow methods in image and object space in Chapter 3.

We will then discuss filtered hard shadows in Chapter 4, where the shadow boundary is smoothed in order to combat aliasing. The difference to soft shadows is that such smoothing of the shadow boundaries is approximating the physical model too much by ignoring the configuration of light source, shadow caster and shadow receiver.

Chapter 5 represents the major part of this tutorial and will deal with soft shadow algorithms. Here the transition between lit and shadowed regions is modified to mimic the underlying physical behavior, which is a significant challenge. Further, we will make distinctions between geometry- and image-based solutions, as well as methods that deliver accurate results for certain configurations and those that are always approximate.

Towards the end of the tutorial, in Chapter 6, we will give a short summary of shadow methods in the context of environmental illumination. A special such case is ambient occlusion. Here, visibility is seen as accessibility and is closely related to the assumption that light is coming from all directions.

We conclude this document in Chapter 7 by giving the reader some hints on when and under what conditions an algorithm is of interest. Developers can hence find out which approaches might be well-suited for a certain problem and find all the details about the algorithm in the corresponding section in our document.

How to Read These Notes

Depending on the type of reader, it makes sense to access this document differently. If you are less interested in the theory behind shadows and the basic approximations, it is possible to skip the introduction and start with the algorithmic descriptions. Only Equations 1.3 and 1.4 are essential, but can be understood without context.

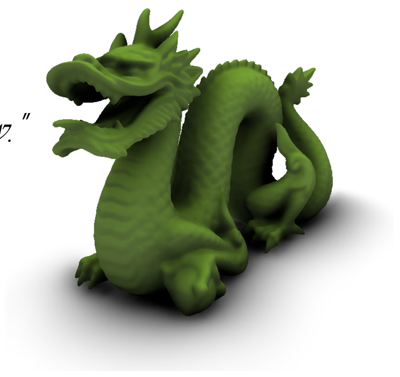
If interested in specific types of algorithms, one can follow our classification into sections. Each can be read independently, but for an exhaustive overview, it makes sense to start reading the algorithmic sections in order.

Finally, the rather practically oriented reader might consider jumping ahead to Chapter 7 which gives an overview of the most suited techniques for various scenarios. Given the particular algorithm of interest it is then possible to read the corresponding section in the main body of the document.

This tutorial is obviously focused on shadows, but provides some more general outlooks and discussions of other related topics. These excursions are indicated with oval boxes and can always be skipped safely. The information therein is not crucial for the understanding of this document, but often provides a deeper insight, which might be considered of value by the interested reader.

"Beware lest you lose the substance by grasping at the shadow."

Aesop (~600 BC) - *The Dog and the Shadow*



1 Introduction

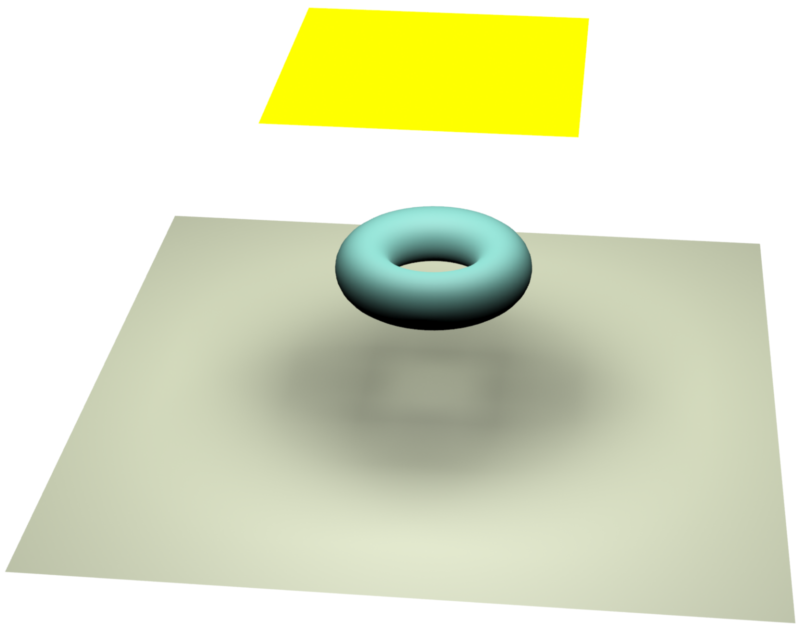


Figure 1.1 A very large source leads to soft shadows. All points on the floor are actually lit to some degree.

1.1 Definition

What is a shadow? This is a good question and because of the fuzziness of the term even dictionaries have trouble giving an accurate definition. WordNet [Princeton University, 2009] states: *Shade within clear boundaries* or *An unilluminated area*. By looking at Figure 1.1, one realizes rapidly that this definition is not accurate enough. The same holds for other definitions which try to capture the common notion that a shadow is often attributed to a certain object; for instance Merriam-Webster [2009] states:

> The dark figure cast upon a surface by a body intercepting the rays from a source of light.

A better definition is given in the American Heritage Dictionary of the English Language [Pickett et al., 2000]:

> An area that is not or is only partially irradiated or illuminated because of the interception of radiation by an opaque object between the area and the source of radiation.

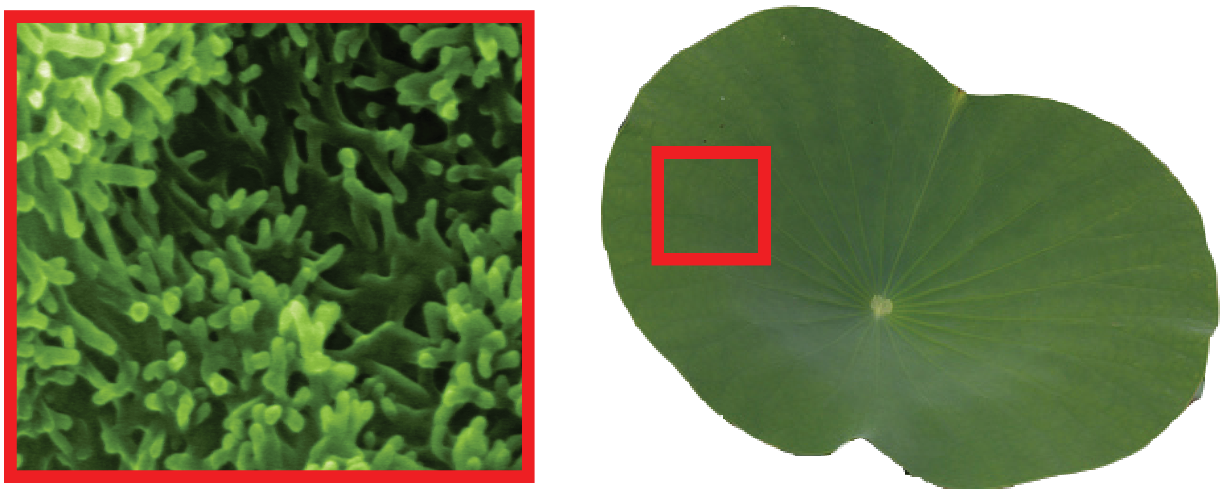


Figure 1.2 What we define as *shadow* depends upon the scale at which we look at objects. In the real world, the definition is thus very ambiguous; in a virtual world, described by a mathematically accurate framework, precise definitions are possible and meaningful. Left: Courtesy of Prof. U. Hartmann, Nanostructure Research and Nanotechnology, Saarland University. Right: Courtesy of [flickrPrince, 2007]

This definition brings us closer, and coincides more readily with the one provided by Hasenfratz et al. [2003]:

> Shadow [is] the region of space for which at least one point of the light source is occluded.

There are two catches though. First, this only considers direct lighting; light bouncing off a surface is ignored. Second, occluders are considered opaque, which is not necessarily the case in the real world.

But even when restricting ourselves to opaque objects and direct light, the definition for the "real world" is not as simple as the above descriptions lead us to believe. Shadows are different from shading, that is, different from situations where incident light is only partially reflected from a surface. Take a look at Figure 1.2 (left): do we see shadows in this picture? Without exactly knowing what is depicted, most people would say yes. However, this picture actually shows a microscopic zoom of a leaf just like the one in Figure 1.2 (right). If one presents solely this latter picture, most people will tend to argue that there is no shadow. The underlying principle is that what we see and how we interpret it depends highly on the scale at which we look at things. There is hence a fine line between shading and shadows [Heidrich et al., 2000] as a surface's reflection behavior is partially influenced by micro-scale light blocking.

In our artificial world, details are usually omitted, but unfortunately their impact on appearance can be enormous. A CDROM is a typical example of this: if you look at its back you see a rainbow of colors due to diffraction of light caused by the fine surface structure that is used to store data. In practice (in our virtual reality), we cannot work at the scales necessary to capture these effects, and as a consequence approximations are necessary. Many approaches modify the appearance of a surface using techniques that simulate detail which otherwise would be lost due to the coarse representations of our models. Even more, virtual objects are often just boundary rep-

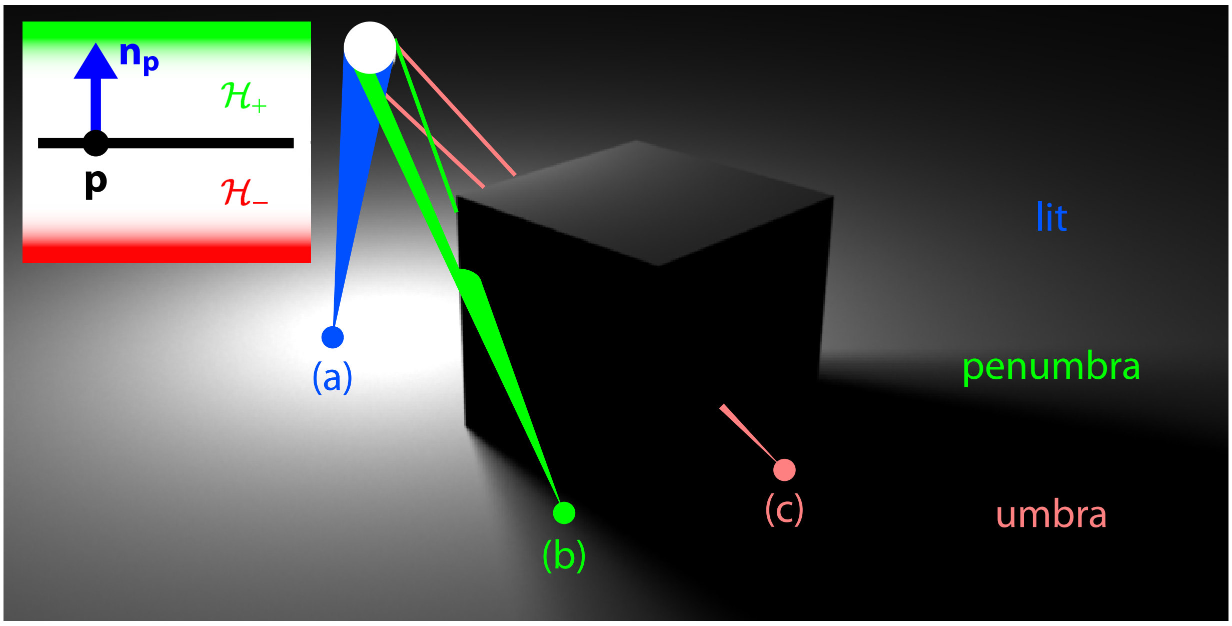


Figure 1.3 Notations for the derivation (Inlay). A point is either lit (a) or shadowed (b, c). In the latter case, we further distinguish between penumbra (b) and umbra (c), depending on whether the light source is only partially or completely hidden.

resentations (at least in case of triangular meshes). Real-world objects are much more complex and many effects take place underneath the surface; light is scattered, attenuated, or diffracted. To overcome this limitation, a great deal of research focuses on simulating these interactions approximately on the surface. Example solutions include texturing, bump and normal mapping, as well as general distribution functions like BRDFs (see below), BTDFs and BSSRDFs. More advanced reflectance models such as Cook–Torrance [1982] approximate the surface by a distribution of microfacets and explicitly account for the visibility of these small-scale details.

This leaves us with an interesting situation. In the real world, shadows might have all kinds of ambiguities. By contrast, in our artificial universe, details are limited, and shadows are described independently of scale and purely in terms of visibility. A definition such as the one given by Hasenfratz et al. [2003] is mostly sufficient; at least as long as only opaque objects and direct lighting are considered. Completely general real-time algorithms, going beyond these restrictions, are probably a challenge to still occupy future generations. Hence we will assume opaque objects and direct light for the remainder unless otherwise stated.

In order to provide the reader with a mathematically sound specification, we will derive a clean definition in the following. The experienced reader might want to skip this part and solely take a look at Equations 1.3 and 1.4 which will be referred to hereafter.

From the rendering equation to soft shadow approximations

Notations Figure 1.3 illustrates the main notations. With $\mathcal{H}_+(\mathbf{p})$ denoting the open positive half-space defined by a point $\mathbf{p}$ and its normal $\mathbf{n_p}$, the point $\mathbf{p}$ lies in *shadow* if and only if there exists an open segment in $\mathcal{H}_+(\mathbf{p})$ from $\mathbf{p}$ to a source sample $\mathbf{q}$ on the light[1] $\mathcal{L}$ which intersects the scene geometry $\mathcal{S}$. Let $\text{occluded}_{\mathcal{L}}(\mathbf{p})$ be the occluded parts of the light source, i.e., the endpoints on the light source of such segments, then this definition is equivalent to

$$\text{occluded}_{\mathcal{L}}(\mathbf{p}) = \{\mathbf{q} \in \mathcal{L} \mid (\mathbf{p}, \mathbf{q}) \cap \mathcal{H}_+(\mathbf{p}) \cap \mathcal{S} \neq \emptyset\} \neq \emptyset. \tag{1.1}$$

Note that this definition assumes that light travels along straight lines (even though in some situations this is not a valid approximation, e.g. in case of atmospheric diffraction, or near black holes). Let $\mathbf{p}$ lie in shadow; then $\mathbf{p}$ is said to be in the *umbra* if $\text{occluded}_{\mathcal{L}}(\mathbf{p}) = \mathcal{L}$ (i.e., if the whole light source is blocked by the scene geometry) and in the *penumbra* otherwise.

Generally, we refer to an object that can intersect segments from $\mathbf{p}$ to the light as *occluder* (or equivalently *blocker* or *shadow caster*) for $\mathbf{p}$. Elements containing such points in shadow, i.e. onto which a shadow is cast, are called *receivers*. There are situations where receivers and blockers are distinct, or where each receiver is only shadowed by a subset of casters. Notably, some algorithms do not allow self-shadowing (caster and receiver are the same object).

So far, we have clarified where we can find shadows. Now, we will discuss their actual influence on the appearance of a scene. We will make use of one of the fundamental equations in computer graphics, the so-called *rendering equation* introduced by Kajiya [1986]:[2]

$$L_o(\mathbf{p}, \omega) = L_e(\mathbf{p}, \omega) + \int_{\Omega_+} f_r(\mathbf{p}, \omega, \hat{\omega})\, L_i(\mathbf{p}, \hat{\omega})\, \cos(\hat{\omega}, \mathbf{n_p})\, d\hat{\omega}, \tag{1.2}$$

where $\mathbf{p}$ is a point and $\mathbf{n_p}$ the surface normal at $\mathbf{p}$, ω denotes a direction, and Ω_+ is the hemisphere above the surface at $\mathbf{p}$. The equation puts the following functions into a relation:

- L_o describes the outgoing radiance (light energy per unit time and unit solid angle and unit projected area) as a function of position $\mathbf{p}$ and direction ω. Simply put, the light (direct and indirect) leaving a point in a given direction.
- L_e yields the emitted radiance. Put simply, the light produced at a given point for a given direction.
- L_i is the incoming radiance. We will see right hereafter that it directly relates to L_o.
- f_r is a BRDF, a bi-directional reflectance distribution function. Put simply, it describes the ratio of exitant to incident light for a given point and directions $\hat{\omega}$ and ω of ingoing and outgoing radiance. Note that this function can be very complex, but might also just be a constant for perfectly diffuse materials.

The rendering equation is physically-based and describes the equilibrium of energy in a scene. It is a good model of illumination exchanges, but solving the equation is analytically difficult (except for a few uninteresting cases).

Photo-realistic rendering aims at finding efficient ways to approximate and populate this equation. The equation inherently depends upon itself, which makes this task particularly difficult.

[1]Artificial constructs like environment maps can have *source samples* at infinity.

[2]Kajiya introduced the equation in a different formulation, but for our explanation this equivalent form is more appropriate.

Further Reading

A more detailed and accurate derivation of the rendering equation can be found in [Sillion and Puech, 1994]. It also contains an exhaustive presentation of the quantities and units of all function components. By contrast, this section aims at providing very high-level insights into what we compute and where it comes from originally.

Employing the notation $\mathbf{p} \to \mathbf{q} := \frac{\mathbf{q}-\mathbf{p}}{\|\mathbf{q}-\mathbf{p}\|}$, the following relationship holds:

$$L_{\mathrm{i}}(\mathbf{p}, \mathbf{p} \to \mathbf{q}) = L_{\mathrm{o}}(\mathbf{q}, \mathbf{q} \to \mathbf{p})$$

for mutually visible points $\mathbf{p}$ and $\mathbf{q}$, which means that $\mathbf{p}$ and $\mathbf{q}$ can be connected by a segment that does not intersect the scene and along this segment. Consequently, the outgoing illumination from one side is exactly the incoming illumination from the other side and vice versa.

The integration over the directions as denoted in Equation 1.2 can be reinterpreted. It corresponds to an integration over a sphere centered at $\mathbf{p}$ onto which all the surrounding geometry is projected as seen from $\mathbf{p}$. We can hence perform a change of variables and equivalently integrate over the surfaces of the scene instead of over directions, leading to:

$$L_{\mathrm{o}}(\mathbf{p}, \omega) = L_{\mathrm{e}}(\mathbf{p}, \omega) + \int_S f_{\mathrm{r}}(\mathbf{p}, \omega, \mathbf{p} \to \mathbf{q})\, L_{\mathrm{i}}(\mathbf{p}, \mathbf{p} \to \mathbf{q})\, G(\mathbf{p}, \mathbf{q})\, V(\mathbf{p}, \mathbf{q})\, d\mathbf{q},$$

where

$$G(\mathbf{p}, \mathbf{q}) = \frac{\cos(\mathbf{p} \to \mathbf{q}, \mathbf{n}_{\mathbf{p}}) \cos(\mathbf{q} \to \mathbf{p}, \mathbf{n}_{\mathbf{q}})}{\|\mathbf{p} - \mathbf{q}\|^2}$$

and V encodes a binary visibility function; it is one if $\mathbf{p}$ and $\mathbf{q}$ are mutually visible and zero otherwise.

For soft shadows, we are only interested in direct illumination. This removes the equation's dependency on itself. Consequently, for all points in the scene, the integral evaluates to zero except for those locations lying on a source. It follows that the term L_{e} can simply be omitted and added back later (In practice, this means the light source is simply drawn on top of the final image). Also, the additivity of the integral allows us to treat several lights sequentially, summing up their contributions. We thus assume that there is only one source in the scene, thereby obtaining the *soft shadow equation*:

$$L_{\mathrm{o}}(\mathbf{p}, \omega) = \int_{\mathcal{L}} f_{\mathrm{r}}(\mathbf{p}, \omega, \mathbf{p} \to \mathbf{q})\, L_{\mathrm{e}}(\mathbf{q}, \mathbf{q} \to \mathbf{p})\, G(\mathbf{p}, \mathbf{q})\, V(\mathbf{p}, \mathbf{q})\, d\mathbf{q}, \tag{1.3}$$

Another simplification is to assume that all surfaces in the scene are Lambertian (perfectly diffuse), which causes the BRDF to become independent of directions, i.e. $f_{\mathrm{r}}(\mathbf{p}, \omega, \hat{\omega}) = \rho(\mathbf{p})/\pi$ where $\rho(\mathbf{p})$ denotes reflectance. As a direct consequence, the outgoing radiance L_{o} also no longer depends on the outgoing direction. Therefore, we get the following equation:

$$L_{\mathrm{o}}(\mathbf{p}) = \frac{\rho(\mathbf{p})}{\pi} \int_{\mathcal{L}} L_{\mathrm{e}}(\mathbf{q}, \mathbf{q} \to \mathbf{p})\, G(\mathbf{p}, \mathbf{q})\, V(\mathbf{p}, \mathbf{q})\, d\mathbf{q},$$

It is widely considered to correspond to a physically-based soft shadow computation [Agrawala et al., 2000]. In practice, a still close result can be obtained when simplifying the equation further.

If the distance of the light to the receiver is relatively large with respect to the light's solid angle (the angle an object subtends in three-dimensional space for a given point), and the light's surface is well-behaved, then the geometric term G varies little. This allows for a separation of the integral:

$$L_o(\mathbf{p}) = \underbrace{\frac{\rho(\mathbf{p})}{\pi} \int_{\mathcal{L}} G(\mathbf{p},\mathbf{q})\, d\mathbf{q}}_{\text{Shading}} \underbrace{\int_{\mathcal{L}} L_e(\mathbf{q},\mathbf{q}\to\mathbf{p})\, V(\mathbf{p},\mathbf{q})\, d\mathbf{q}}_{\text{Shadow}}.$$

Basically this separation results in a decoupling of shading and shadows. This approximation depends on the correlation between the two functions and Soler [1998] delivers an error discussion in his dissertation.

It is interesting to know that for $\int_{\mathcal{L}} G\, d\mathbf{p}$ analytic solutions exist even for the case where $\mathcal{L}$ is a polygon and we integrate further over all $\mathbf{p}$ within another polygonal region [Schröder and Hanrahan, 1993]. This is typically the case for radiosity computations. On the other hand, the exact formula, found by Schröder and Hanrahan [1993], is often considered too complex in many practical applications. Although it is an important theoretical contribution that remained unsolved until 1993 (despite earlier attempts such as Lambert's in 1790). For complex BRDF's or visibility configurations, we are generally left with sampling as the only option (e.g. employing Monte Carlo techniques).

Furthermore, we typically assume that the light source has homogeneous directional radiation over its surface, causing L_e to simplify to a function of position $L_c(\mathbf{q})$ only, which in case of a uniformly colored source reduces to a constant $\bar{L}_c$ and can be taken out of the integral. (This case is very common and we will explicitly mention if $L_c(\mathbf{q})$ is not a constant and must remain in the integral.) The remaining *visibility integral* modulates the shading and represents the true shadow component in the equation:

$$\bar{L}_c \int_{\mathcal{L}} V(\mathbf{p},\mathbf{q})\, d\mathbf{q}. \tag{1.4}$$

Usually, for real-time applications, Equation 1.4 is meant when talking about soft shadows and most solutions aim at solving it. Nevertheless, we will see that some methods provide a correct visibility sampling and not just the integrated quantity. This information allows us to remount to Equation 1.3 at a supplementary cost. One interesting remark is that for point lights, both equations simplify to a binary visibility query.

In general, Equation 1.4 is not physically correct and the approximation can be quite different compared to a reference solution based on Equation 1.3. Only the amount of visibility is evaluated and not which part is blocked. Because of the term $G(\mathbf{p},\mathbf{q})$, the influence of the source on the point $\mathbf{p}$ is not uniform and falls off with distance and orientation, which is not captured via the separated integration. Nonetheless, often results are convincing though, which made Hasenfratz et al. [2003] even claim in their survey that methods like [Assarsson and Akenine-Möller, 2003; Soler and Sillion, 1998] pursuing this approximation are *physically* accurate for a convex occluder.

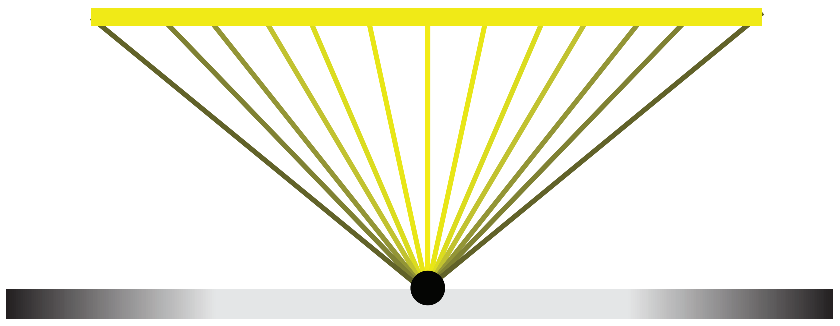
1.2 Why Should We Care About Shadows?

In general, people probably don't care much about shadows, except that we might want to avoid them to get tanned. But in graphics, shadows are of major importance: they provide clues con-

A Word on Direct Illumination

In many cases, the direct illumination integral is approximated by replacing $\mathcal{L}$ with a point light. Currently, this is typically combined with a simple lighting model like Lambert or Phong-Blinn. (For more details, we refer to the excellent survey by Schlick [1994] and for efficient solutions to map them to older graphics cards, we suggest consulting [Heidrich and Seidel, 1999]. Modern hardware often facilitates this task via shaders.)

$G(\mathbf{p}, \mathbf{q})$ is obviously related to a squared distance. (Light travels along straight lines and the energy is conserved. In consequence, the energy on the surface of a sphere around the light source should be constant.) Interestingly, OpenGL makes it possible to attenuate the light's power with distance using a general quadratic polynomial. This may sound strange, and makes many people smile when they hear about this option for the first time, but it actually makes some sense to include this supplementary degree of freedom.



The situation is depicted in the figure above. Far away source samples will have very little influence on the final result due to their orientation with respect to the receiving point. This is reflected in the $\cos(\mathbf{p} \to \mathbf{q}, \mathbf{n_p}) \cos(\mathbf{q} \to \mathbf{p}, \mathbf{n_q})$ term of $G(\mathbf{p}, \mathbf{q})$. In consequence—if the light is large with respect to the current receiving point—moving this source a little will have almost no impact on the received illumination. In other words, leaving the source at the same position and looking equivalently at adjacent receiver points, we will observe basically the same energy. From a certain distance on, orientation will be mostly constant, thus attenuation behaves in a quadratic manner as predicted. In-between, the behavior passes through a linear stage. The more general polynomial that OpenGL offers mimic some of this behavior.

cerning the spatial relationship of objects in the scene and the shape of a receiver, and reveal to some extent information hidden from the current point of view.

Several experiments underline the importance of shadows. For instance, Kersten et al. [1996] investigated the influence of shadows on perceived motion. In their many experiments, they also displayed a sphere above a plane, not unlike Figure 1.4 (left). Just as you can see in this image, the trajectory and position of the shadow influence the perceived position. If the shadow moves up in the image, the observer will have the impression that the sphere moves to the back of the box towards the ground. Interestingly, Kersten et al. [1996] found that soft shadows can lead to an even stronger motion cue than hard shadows.

Such perceptual results are often cited to stress the importance of shadows, and they seem to illustrate this point well. But it is arguable whether the conclusion is that we should aim at realistic

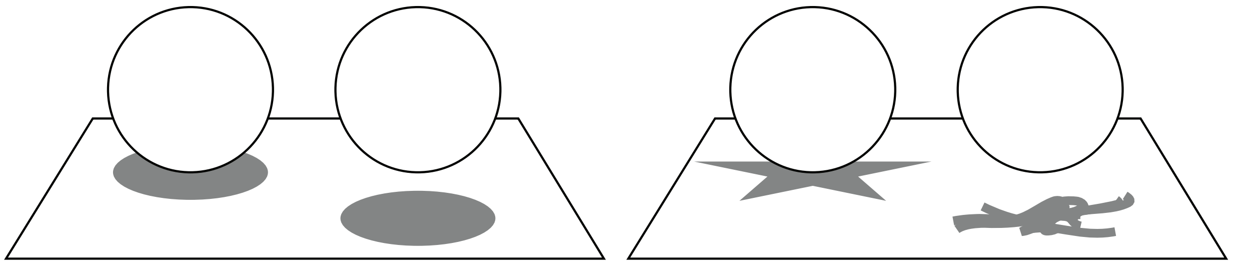


Figure 1.4 Shadows have an important influence on the interpretation of spatial relationships in a scene (left). Nevertheless, even coarse approximations can achieve the same effect (right).

shadows. In fact, even the most approximate shadows can often provide sufficient information to interpret the spatial relationships. Take a look at Figure 1.4 (right). We understand the scene just as before, but the shadows are far from realistic. Other experiments [Ni et al., 2004] illustrated that it actually suffices to add dark indications underneath the object. An observer automatically establishes the connection and accepts the *shadow*. In fact, this allowed the use of simple disc-shaped shadows underneath the characters in many older video games. The same principle also allowed the utilization of shadows to convey messages (like a famous advertisement of the Star Wars movie *Episode 1: The Phantom Menace*) and today shadow manipulations have also found their way into non-photorealistic rendering [DeCoro et al., 2007].

It is not necessary, then, to create accurate shadows to explain a scene. But the question is how far we can simplify while maintaining spatial information, and ultimately also realism. Unfortunately, this is very difficult to decide, even for artists. Cavanagh [2005] mentions several perceptual problems when approximating shadows and other physical phenomena. The work by Luca Signorelli *The Assumption of the Virgin with Saints Michael and Benedict* (late 1480) really underlines the fact that we are bad at estimating light directions and do not automatically realize problems of incoherent lighting. Only careful observation reveals that the ceiling's shadows are inconsistent with respect to the shadows cast by people in the scene.

Conversely, the moment that inconsistencies are in close spatial relation, as in Fra Carnevale's *The Birth of the Virgin* (1467), these shortcomings are evident. Two shadows overlap that have different gray levels and the artist simply decided to take the darker color in the overlapping region. Such evidenced problems, indicate the limitations of coarse approximations. Even though we obviously would like to benefit from the limited perceptual capabilities of the human visual system, this is very difficult for dynamic scenes where an acceptable, approximated configuration might change into an obvious visual deficiency. As a direction for future research, however, this is a promising field. This also concerns the degree to which approximations can be applied to shadows.

Finally, some artists exploit the fact that we often make unconscious assumptions concerning the caster's shape based on a shadow. Surprisingly, this can fail badly, as demonstrated by Shigeo Fukuda's installation Dirty White Trash (with Gulls). In fact, it is hard to *fake* shadows, especially if lights and objects are dynamic.

Incorrect shadows decrease the realism of an image dramatically which is problematic if a sufficiently realistic rendering is needed, for instance for architectural design. Here, light transport plays an important role and often even involves a complete global illumination computation of which soft shadows are just the first step. While an architect may be capable of imagining the final illumination in a building, a potential customer is not. Recent work on global dynamic relighting [Kristensen et al., 2005; Kontkanen et al., 2006; Lehtinen et al., 2007; Dachsbacher et al., 2007] underlines the importance of decoupling direct from indirect lighting. Due to its typically smooth variation, indirect lighting can often be coarsely computed and determined solutions be compressed well. By contrast, direct light consists of relatively high emitted energy usually resulting in high-frequency content. Consequently, approximations are more visible, and achieving a realistic composite of direct and indirect lighting necessitates an accurate direct lighting pass. For movie productions, even direct lighting is costly because the sources need very accurate sampling. This is especially true for shadows from the light sources because the transferred energy is relatively high and fewer approximations are possible than for indirect lighting.

Finally, realism can be important if the observer is investigating a realistic environment. Nevertheless, in this scenario, we should take advantage of the fact that accurate shadows are not needed to evoke the notion of realism. But we should be warned that inconsistencies can destroy the immersion in this virtual world. In some situations, accurate shadows might even be part of the game play, such as a player who casts a shadow around a corner, revealing her/his position. Furthermore, if the degree of realism is high enough, this might allow for the deduction of identity or equipment.

We should provide *sufficient* realism, not necessarily exactitude. The keywords in this context are *plausible* and *convincing*. Unfortunately, it is not easy to achieve this goal. Ultimately, only Equation 1.3 seems to be foolproof, but Equation 1.4 is sufficient in a large number of cases. Any further approximation is likely to fail in some common configurations. This is a major dilemma: we should compute approximate solutions, but in practice, only physically based shadows seem to be convincing in all situations. In the following, we will illustrate the main failure cases that make soft shadows such a challenging topic.

1.3 Why Is It Difficult to Compute Shadows?

Figure 1.5 shows how drastically soft shadows influence the appearance of a scene. A single hard shadow results in an unrealistic image. Even though a large amount of light is impinging in the room, the fruit basket casts a shadow that is overly dark and large. In nature, we would never encounter a small object that could block the light of the entire window. This is a situation where even a novice realizes that something is not quite right. This can be particularly disturbing in an animation because even small objects can block visibility of a point light, bathing the entire scene in darkness. The soft shadow image, on the other hand, does not exhibit these artifacts. Contact shadows stay sharp and the scene receives a realistic amount of direct light.

One difficulty of soft shadows is that treating occluders separately is not simple. Even if for each occluder an accurate scalar value is derived that indicates the blocking contribution for a particular object, it is generally not possible to derive a good estimate of the visibility integral. It is true that these values can be used to deliver upper and lower bounds for the exact visibility integral, but not more. Let $\{O_i\}$ be a set of objects and $\{B_i\}$ the corresponding set of visibility integrals for a given receiver point $\mathbf{p}$. Then, the following inequality holds:

$$\max_i B_i \leq \int_{\mathcal{L}} V(\mathbf{p}, \mathbf{q})\, d\mathbf{q} \leq \sum_i B_i. \tag{1.5}$$

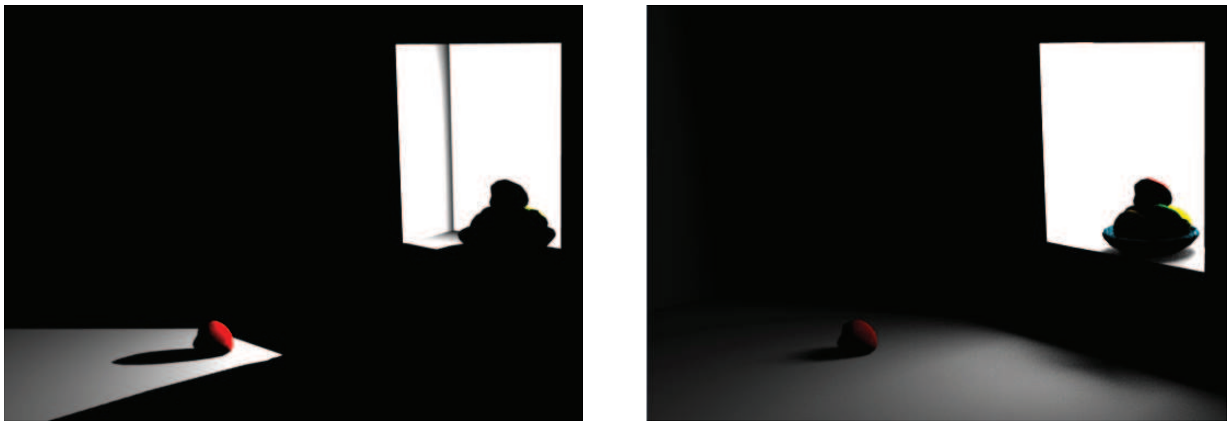


Figure 1.5 This scene shows one example of the importance of soft shadows in obtaining a convincing and realistic-looking scene. On the left is the result using a single hard shadow sample, whereas the right shows the outcome of a soft-shadow computation.

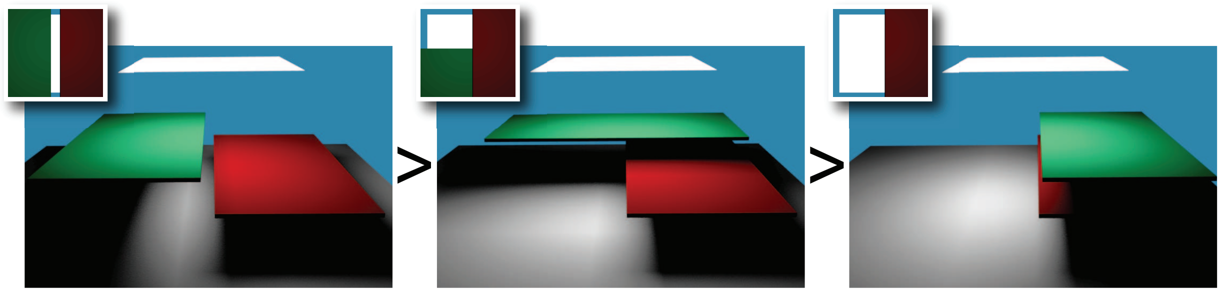


Figure 1.6 The figure shows different blocker-fusion scenarios for a view-sample in the center of the scene. On the left, both blockers occupy separate parts of the hemisphere and thus their occlusions should be summed. In the middle, they partially overlap; here, a multiplication is closer to the truth. The example on the right depicts one blocker being entirely hidden by the other. The maximum of both occlusion values would be the right choice. (Inlays show the source as seen from the center view-sample)

Figure 1.6 shows an illustration of different cases. The lower bound is produced if all blockers fall in the same part of the hemisphere; the upper bound if all their projections are disjoint. Many solutions have been proposed to combine different blockers, including the two extremities [Arvo et al., 2004; Assarsson and Akenine-Möller, 2003], as well as an average [Soler and Sillion, 1998]. None of these approximations are valid in all situations. That problem, as for visibility, is referred to as inaccurate *occluder fusion*. Figure 1.7 shows an extreme case for a very simple game character that illustrates how shadows can become very unrealistic if blocking contributions are not combined properly.

The intricate relationship between soft shadows and visibility introduces other implications. One cannot rely solely on objects visible from a single point on the source to compute shadows. This is illustrated in Figure 1.8. The right image is convincing, but on the left, only faces visible from the source's center intervene in the shadow computations (the computation for the visible geometry is carried out with highest accuracy). One can see significant problems. The shadow

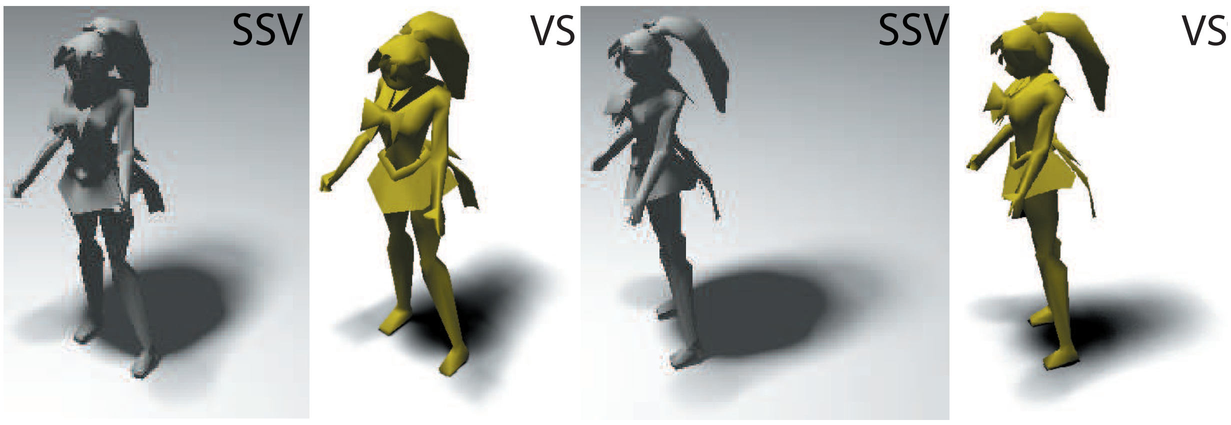


Figure 1.7 Even for typical and relatively simple game characters, classical approximations (silhouette from the center, additive occlusion) can cause noticeable artifacts (here the result with [Assarsson and Akenine-Möller, 2003] (SSV) is shown). The umbra is overestimated. In comparison, an accurate visibility sampling leads to convincing shadows (the reference (VS) was computed using [Eisemann and Décoret, 2007]).

on the sphere is lost and especially half of the shadow below the sphere seems to be missing. Although this part of the scene is not visible from the source's center, it has an important impact on the shadow. In a simple scene like this already four layers interact. Further, the notion of a layer is ill-defined for faces aligning with the source's center. During extraction these would usually be missed or captured inadequately. The same observation holds if one uses silhouette edges as seen from the center [Akenine-Möller and Assarsson, 2002; Assarsson and Akenine-Möller, 2003; Assarsson et al., 2003; Chan and Durand, 2003; Wyman and Hansen, 2003]. In consequence, artifacts and even temporal incoherence can arise.

On the one hand, it is surprising to see how much attention is needed when evaluating Equa-

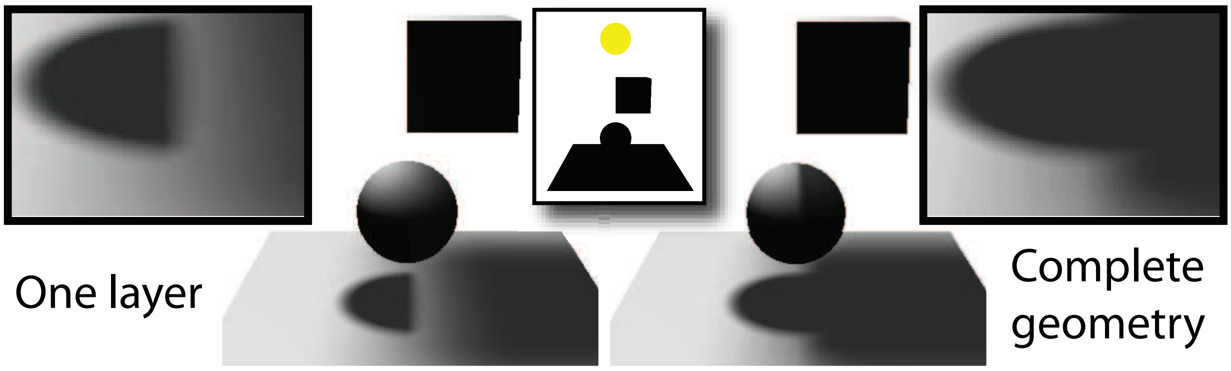


Figure 1.8 The first depth layer might not be enough for convincing shadows. Left: One depth layer and accurate shading. For accurate shadows, in this case, four layers would need to be extracted. Further, it has aligned the face with the light, which can be problematic when rasterized. Right: Accurate shadow solution.

Why not accumulate hard shadows?

The *XBox 360* has a fill-rate of about 16 billion pixels per second and can process up to 500 million triangles per second. This sounds largely sufficient, but let's assume that the resolution of our view is $512^2 = 262{,}144$ pixels. If we further assume that the evaluation of a shadow has approximately twice the cost of rendering this view, and multiply by a factor of 256 samples, we obtain 134,217,728 pixels. If we want 60 fps the fill-rate goes up to 8,053,063,680. This seems to work out, as it is approximately half the specification of the card, but we have not yet processed any geometry nor have we evaluated any of these maps. It is also worth noting that specifications, like for instance fill-rate and geometry-processing, are measured independently and in artificial conditions (e.g. non-shaded triangle strips, no blending operations, etc.). Further, the workload of the two processing stages (vertex/fragment shader) is no longer independent because the architecture relies on general stream processors. In practice, only a few frames per second are possible even on simple models.

tion 1.4. On the other hand, a robust solution is to simply sample the equation:

$$\int_{\mathcal{L}} V(\mathbf{p},\mathbf{q})\, d\mathbf{q} \approx \frac{1}{n+1} \sum_{i=0}^{n} V(\mathbf{p},\mathbf{l}_i), \tag{1.6}$$

where $\mathbf{l}_i \in \mathcal{L}$ are uniformly placed samples on the source.

We have already encountered the function V. It delivers the visibility between two points. So in Equation 1.6 where $\mathbf{p}$ and $\mathbf{l}_i$ appear, it encodes whether a point light source illuminates a point in the scene. Soft shadows can thus be computed by covering the area/volumetric source with sample lights. Consequently, the integration can be done by a one-by-one evaluation, where energy contributions are added. Even Equation 1.3 can be well approximated in this way. We thus have a direct link between hard and soft shadows: Shrinking an area light to a point leads to hard shadows and sampling an area light with point light sources results in soft shadows. Unfortunately, as we will see in the next section, even shadows for a single light remain an issue, and for convincing soft shadows a high amount of samples is needed. 256–1024 are standard for medium-sized sources, but large area lights might necessitate even more. Each sample will need to process the geometry of the scene. For 1,000 samples, the cost of a brute force computation will thus be roughly 1,000 times higher than for a single point light. Different solutions are necessary and we will describe them throughout the next sections.

1.4 General Information for the Reader

In order to facilitate the lecture of this document, let's quickly summarize the major elements that will reappear throughout this document.

We define several terms to describe scene entities:

- *light sample* or *source sample* – a point on the light;
- *(scene) point* – a point on the surface of the scene;
- *view sample* – a scene point visible from the current viewpoint.

Isn't ray tracing the answer to everything?

Ray tracing is currently emerging, but soft shadows will pose a significant thread in this context as well. According to reports by Intel regarding ray tracing, a single P4 3.2Ghz is capable of 100 million rays/sec on average models, but this mostly in unrealistic scenarios and necessarily static scenes. Even the latest eight-core systems usually achieve under 83 million rays per second on average-sized scenes (in demos, as of the end of 2007, shown by Intel themselves [Shrout, 2007]). 512^2 resolution times 60 fps leaves only 5 secondary rays per pixel. Even if one assumes that a more powerful solution with 450 million rays per second existed, only 28 shadow rays would be possible. Furthermore, the timings are usually measured without shading/texturing, which has a significant speed impact [Wald et al., 2006]. In the long run, ray tracing could be a solution, but it is unlikely that we have the needed processing power to compute enough secondary rays for sufficient quality any time soon.

Further, the formatting we use is as follows:

- $\mathbf{p}$ – a point;
- $f()$ – a function;
- B – the notation for a general object, e.g., a blocker;
- $\mathcal{N}$ – usually a set ($\mathcal{L}$ denotes the light source as it can be interpreted as an infinite set of points).

2 Basic Shadow Techniques

Over the last years, many contributions have been made in the field of shadows. And also many found their way into computer games. But at the time of this document, even though we have come a long way, accurate soft shadows cannot be obtained in sufficient quality in real time, and we are only on the verge of achieving pixel-accurate hard shadows for geometrically complex scenes at acceptable speed.

Both areas still leave room for improvement, especially as a scene with a single point light is not a very common scenario. In modern movie productions, hundreds of lights are placed by hand to create a wanted illumination.

In the following, we will give an overview of the two most common techniques used for hard shadow computation on graphics hardware, namely shadow mapping [Williams, 1978] and shadow volumes [Crow, 1977]. We will see that these algorithms are relatively well adapted to current hardware and are at the basis of many algorithms available today. Shadow mapping is particularly interesting because it requires only little information about the actual scene geometry and solely relies on an image that encodes the distance to the light source. Nevertheless, there are shortcomings and, over the years, much research has focused on improving the algorithms. For shadow maps and shadow volumes respectively, we will start by describing the main principles and follow up by some improvements.

2.1 Shadow Mapping

In this section we analyze a standard technique to compute shadows based on a point light, so-called hard shadows. The name stems from the fact that these shadows are binary in nature: Either a point is lit or it lies in shadow, creating a sharp, or *hard*, boundary.

Let's assume a very simple scenario, where we have a scene that consists of a receiver and a point light $\mathcal{L}$, as well as a distinct set of occluders that are placed in-between the two. If we create a binary image of the occluders, as seen from the light, shadow queries on the receiver become very simple. Following the definition in Section 1.1 (see also Equation 1.1), a point $\mathbf{p}$ is in shadow if the open segment between $\mathbf{p}$ and $\mathcal{L}$ intersects the scene. Such a segment degenerates to a point in this image as seen from the light. Consequently, a *single* texture lookup allows us to test for shadows: if the pixel containing the segment was filled while drawing the occluders, the point has to lie in shadow, otherwise it is lit. In fact, this binary texture can even be applied to the receiver via projective texture mapping.

Basic Algorithm The aforementioned technique [Akenine-Möller et al., 2008] is a simplified version of the probably most famous solution to compute shadows in real-time applications, namely *shadow mapping* [Williams, 1978]. This method no longer needs to separate occluders

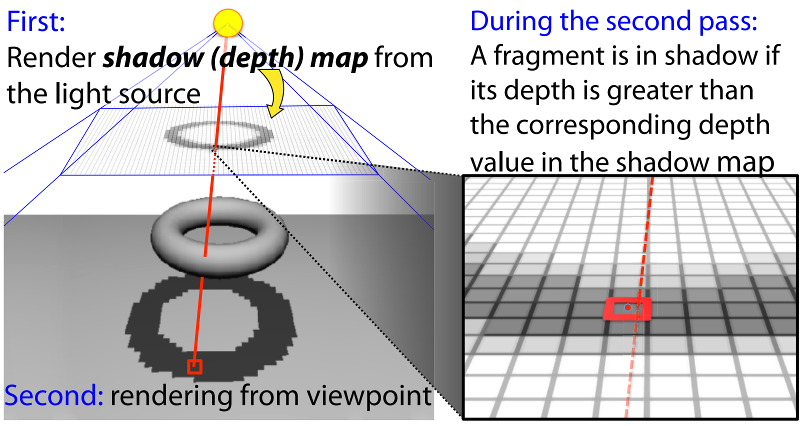

First:
Render ***shadow (depth) map*** from the light source
During the second pass:
A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map
Second: rendering from viewpoint

Figure 2.1 Illustration of the shadow map algorithm. A depth map is created from the light and then queried to determine shadows.

from receivers. Instead of a binary value, his shadow map builds upon the observation that the light *sees* all lit surfaces of the scene. Every hidden (unseen) element lies in shadow. To determine the visible surfaces as seen from the light, shadow mapping starts by creating an image from the light's position. In this image, the so-called *depth/shadow map*, each pixel holds the depth (i.e., the distance from the light) of the first visible surface. Graphics hardware supports the creation of such depth maps at very little cost because the same mechanism is used to resolve visibility during standard rendering. The second step of the algorithm performs a rendering of the scene from the actual viewpoint. For each rasterized fragment (which we will call *view-sample*), its (x, y, z)-coordinates are transformed into light space, in other words, if (x^s, y^s, z^s) are its transformed coordinates, then (x^s, y^s) are the position in the depth map to where the fragment would project when seen from the light and z^s is the distance of the fragment to the light source. To determine whether the fragment is visible from the light, it is sufficient to compare its depth value z^s to the value stored in the shadow map at position (x^s, y^s). If z^s is larger than the stored value, the fragment is necessarily hidden by some other surface nearer to the light and consequently lies in shadow, otherwise it is lit. This process is illustrated in Figure 2.1.

The technique is particularly interesting as it is usable with almost arbitrary input, as long as depth values can be produced. Further, the fact that both steps involve standard rasterization gives it a huge potential for acceleration on graphics cards. In fact, OpenGL provides extensions to perform the algorithm without shader intervention (today, most people would just use shaders, which is more convenient). Currently, shadow mapping and variants are the most popular techniques for creating shadows in games. Nevertheless, several problems are inherent to this method. The most important difficulties are the treatment of omni-directional sources (Section 2.1.1), imprecisions due to the depth test (Section 2.1.2) and aliasing artifacts arising from the pixel representation of the depth maps. We will analyze these problems more closely in the following.

2.1.1 Omni-directional Shadow Maps

The fact that the shadow map is produced via rendering makes it necessary to specify a light frustum, which in turn implies that this technique is mostly aiming at spot lights. The typical way to handle omnidirectional sources is to create, for example, six light cones (one for each side of a cube) that together cover the entire sphere of directions. This solution is currently standard, but implies that faces need to be rendered several times. Recently, geometry shaders can perform this projection on a cube map in a single pass, but the fact that geometry is duplicated for each face introduces a significant penalty. Instead of a cube, Brabec et al. [2002] point out that a parabolic mapping [Heidrich and Seidel, 1998] enables the extraction of the entire field of view with only two renderings. Furthermore, lookups in these maps are very cheap. The fact that the domain is not rectangular and that the sampling ratio might vary by a factor of four are two minor reasons why this technique has not yet received more attention. The main reason is probably that creating these maps is difficult. Lines need to be transformed to curves, which is incompatible with the standard rasterization pipeline. The solution in [Brabec et al., 2002] is to transform only vertices in a vertex shader to the correct position and assume that the scene is tessellated finely enough to provide the correct solution. Recently, Gascuel et al. [2008] proposed to compute the deformed and curved elements on graphics hardware, but the algorithm remains costly. Today, cube maps are still the most efficient solution when combined with a geometry-shader-based decomposition.

2.1.2 Depth Bias

Another problem of shadow mapping is imprecision (see Figure 2.2). The test whether a point is farther away than the reference in the shadow map requires some depth bias. Otherwise, numerical issues and insufficient sampling due to the limited shadow map resolution lead to so-called *z-fighting*. This results in visible shadow sparkles on lit surfaces. Introducing this depth bias is more problematic than it might seem. If a face mostly aligns with the light's view, a much larger bias can be necessary because, otherwise, the surface might shade itself. Hence, the standard approach supported by graphics hardware is to rely on polygon offsets. These modify the depth values output by the geometry. Usually, two parameters are available, a constant offset and an offset that depends on the alignment of the triangle with the light's view rays. Unfortunately, this solution needs to be hand-adjusted. E.g., for a very short triangle, too much offset would not make any sense, as then the depth would show no more correlation with the actual geometry. Even if this still sounds solvable by reducing the bias, another question arises: what about a tessellated planar region? A similar problem presents itself for curved surfaces. Finally, the depth buffer on graphics cards is non-linear. This makes sense for hidden surface removal because it puts more precision on nearby elements, but is not necessarily a good choice for shadow mapping. A region far from the light can actually be very close to the observer, and thus have limited precision where most precision is needed.

Decreasing the Need for a Bias

We will only quickly review the most famous suggestions to lower the bias concerns.

A straightforward solution to increase depth precision is to better fit the near and far plane of the light's frustum. This can be based on the scene's bounding box, but smarter solutions [Brabec

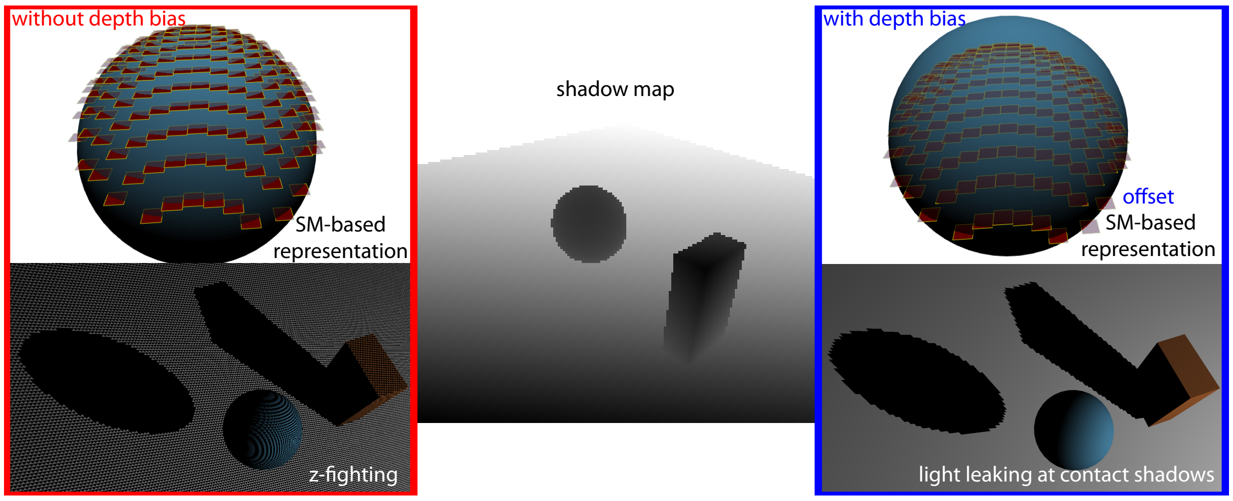


Figure 2.2 The Shadow Map (left) is a discretized representation of the scene. Each pixel can be considered a geometric quad that is situated in the scene at its according depth (2nd from left). This also explains the often jig-jaggy shadow appearance. Furthermore, this can create self-shadowing, everywhere where the quads extend beyond the object's geometry. The problems resulting from these imprecisions are referred to as z-fighting and lead to so-called surface acne (3rd from left). Offsetting the depth values in the depth map (4th from left) lifts many of the depth ambiguities (5th from left), but can also introduce light leaks. This happens when the depth values sink below the receiver's depth. This is most visible at contact shadows (right, at the cube's lower corner).

et al., 2005] will be presented in Section 3.2 that also address aliasing. Further, this paper describes a solution to linearize depth (an idea introduced in [Heidrich, 1999]). This further increases precision, but nowadays can be achieved directly in the shader; moreover one can rely on output textures of higher precision.

Polygon IDs A classical suggestion is to use indices for each polygon instead of depth samples [Hourcade and Nicolas, 1985]. This eliminates the bias because exact indices are compared. Nevertheless, today, this technique is particularly difficult to use because many triangles have sub-pixel size, but only one index can be stored per pixel. If alternative representations are used, the attribution of indices is difficult. Hybrid visibility determinations could be imagined, which use these indices to then initialize a search on the original mesh, but this becomes prohibitively slow.

Second Depth Shadow Mapping Wang and Molnar [1994] suggest using only the second layer of the shadow map which in its basic version only works for watertight scenes. Further, the discretization can lead to imprecisions when several view samples fall in the same shadow map texel, especially because this second depth layer might be far from the light. Therefore they suggest to resort to filtering operations we will talk about in Chapter 4.

Midpoint Shadow Maps A different solution, called *midpoint shadow maps*, has been presented by Woo [1992] that extract two layers of depth instead of just one. The shadow map can then be safely set to the average distance. The first layer cannot be shadowed, the second will still be shadowed due to taking the average. The method does required two passes to extract the first and second layers (see Figure 2.3). On modern GPUs it is possible to improve upon this for watertight scenes by redirecting front- and back-facing triangles to different depth buffers.

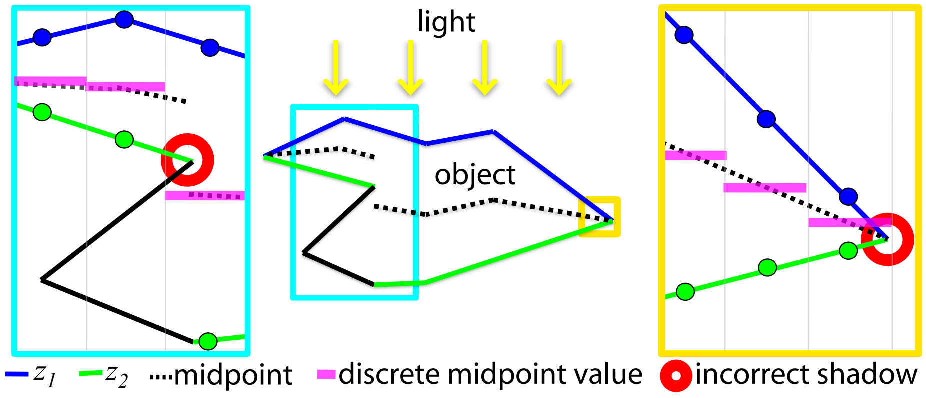


Figure 2.3 Midpoint shadow maps use the average of the two closest layers (marked in blue and green) as shadow map depth. This still results in problematic situations due to the discretization. Left: incorrect unshadowing because parts of the surface are closer to the light than the midpoint value in the shadow map (magenta). Right: incorrect self-shadowing because parts that are supposed to be lit (blue) are behind the midpoint value in the shadow map.

Dual Depth Layers Weiskopf and Ertl [2003] point out two failure cases of the midpoint shadow map (see Figure 2.3) where a carefully adjusted bias sometimes can help. The problem of midpoint shadow maps is that the difference between the first and second depth layer can be very large, leading to an overestimation of the needed depth bias. Weiskopf and Ertl's solution is to combine midpoint shadow maps with a bias by choosing the minimum of a fixed distance threshold and the actual depth distance. In such a way, the offset is always limited. In addition, they discuss the possibility of back-face culling for *watertight*[1] objects because back-faces are black due to shading.

2.1.3 Aliasing

Another drawback of working in image space is that the shadow map has limited resolution. This results in aliasing artifacts, which means that a cast shadow will reflect the discretization. The boundaries of shadows thus contain visible stair-stepping artifacts. The reason is that several view-samples can project into the same shadow map texel, hence resulting in a similar shadow response. To avoid this problem, many modern games use shadow map resolutions that exceed by far the window size: 4096^2 to 8192^2 are typical choices. The probability of two view-samples falling into the same shadow map texel is hereby reduced, but not removed.

This problem, being the major source of artifacts for this technique, has received much attention and therefore it will be analyzed in more detail in Section 3.

[1]Watertight is also sometimes referred to as manifold or closed. The definition we use here is that each triangle edge has one and only one neighboring triangle, the model separates an interior from an exterior region in space.

2.2 Shadow Volumes

Shadows have a relatively long history in the young science of computer graphics. One of the earliest methods, *shadow volumes* [Crow, 1977], was published in 1977. It nevertheless took more than 20 years before it was finally applicable for real-time rendering of average complexity scenes [Brabec and Seidel, 2003]. The algorithm in [Brabec and Seidel, 2003] was very fast for its time, based on many very particular representations in order to adapt the computation to a graphics card. Today, this solution is mostly historical. More direct and efficient implementations are possible on the latest generations of cards. But even today, the best solutions [Stich et al., 2007], which exploit a precomputed hierarchy on a static scene of 500,000 triangles, only achieve interactive rates.

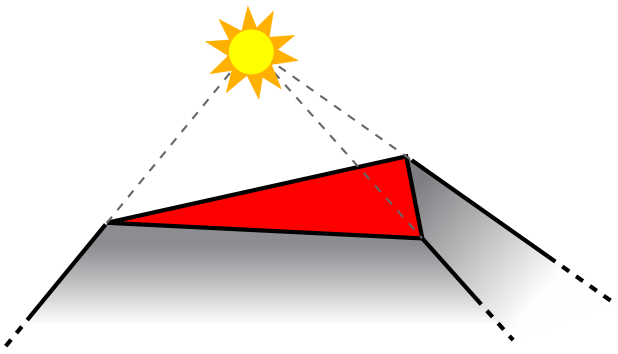


Figure 2.4 A Triangle's Shadow Volume

Basic Algorithm The basic algorithm for shadow volumes determines shadows in a scene by creating volumes representing the shafts arising from blocked illumination.

To simplify explanations, we will suppose that the model is *watertight* or *closed* (see footnote on page 19). Practically, this means that the object is volumetric, with a tessellated surface (typically triangulated) that does not have any cracks that would expose its interior. General, non-closed, models were originally discussed by Bergeron [1986] and an implementation is described in [Stich et al., 2007].

The shadow volume is the region in space that lies in shadow, i.e., all points hidden by the light. For a single triangle, this region is delimited by the triangle itself and faces defined by its extruded edges. The extrusions are constructed by four points each: the edge's two vertices, and these vertices' respective projection from the point light to infinity (see Figure 2.4). A point $\mathbf{p}$ lies in the triangle's shadow if it lies in this infinite volume. This coincides with the definition given in Section 1.1 because any segment connecting a point within the shadow volume with the source will necessarily intersect the triangle.

One realizes that adjacent triangles lead to an inner boundary that can be omitted (see Figure 2.5). This results in the well-known method of only extruding shadow volume quads for silhouette edges as seen from the light source. This is correct for closed objects, while general objects should have two quads for a silhouette edge that is shared by two triangles, and only one quad if the edge is open, i.e., only belongs to one triangle. This is discussed in more detail by Bergeron [1986].

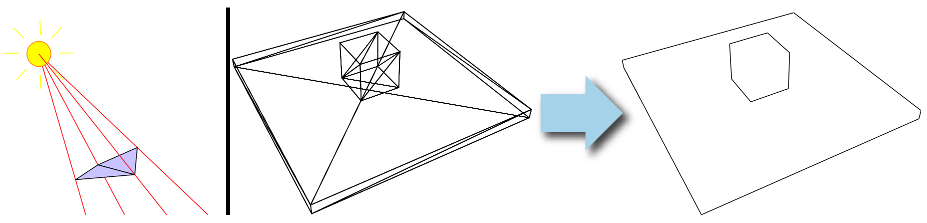


Figure 2.5 Left: Interior edge makes two quads which cancel out. Right: Finding the silhouette edges gets rid of many useless shadow volume quads.

To determine whether a point **p** is contained within a shadow volume and should be considered in shadow, a containment test is done. One possibility to perform this test is to shoot a ray from a *reference point* outside the shadow to **p**. A counter is incremented every time the ray enters a volume, and decremented when leaving. If the final number of intersections is even then **p** is lit else in shadow (see Figure 2.6). This is a direct consequence of the extension of the Jordan Curve Theorem to 3D. Basically, whenever a volume separates an interior from an exterior, any curve connecting a point in the exterior to a point in the interior, will have to intersect the volume's surface an impair amount of times. In our case, this means that a ray from outside the shadow volumes to a shadowed point (inside the shadow volumes), will intersect the volume an impair amount of times.

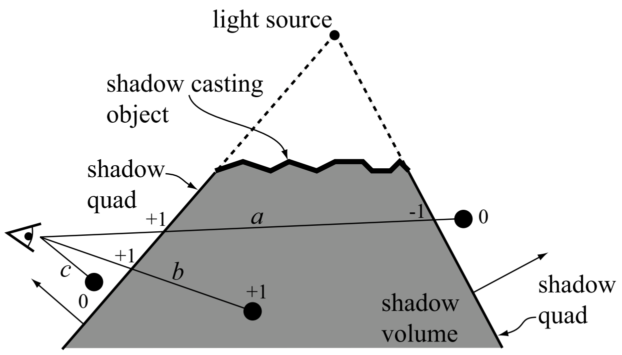


Figure 2.6 The standard shadow volume algorithm. Ray b is in shadow, since the stencil buffer has been incremented once, and the stencil buffer values thus is $+1$. Rays a and c are not in shadow, because their stencil buffer values are zero. (Courtesy of Tomas Akenine-Möller)

***Z*-pass** Efficient implementations Heidmann [1991] sends rays from the eye instead of an arbitrary center, making rasterization possible. The stencil buffer can then be used to count the volume intersections. In a first step, the depth buffer is filled from the viewpoint. Then the fragments of the shadow volume geometry increment/decrement the stencil values according to their orientation (front/back-facing) with respect to the eye. This serves as a parity counter for the intersections. Because the depth buffer blocks all shadow volume fragments further than the impact point **p**, counting only those that *pass* the z-test, the final stencil buffer entry correctly reflects the number of intersections from the eye up to **p**.

One major benefit was that geometric intersection tests were no longer necessary. E.g., earlier approaches clipped the geometry by planes along the shadow volumes to divide the model in lit

Heidmann's Stencil Shadow Volumes (Z-pass)

1st pass: Render the scene with just ambient lighting.
2nd pass: Turn off Z-buffer and Color writing (i.e., draw to stencil buffer only, with enabled depth test). Render front-facing shadow volume polygons to stencil buffer, incrementing the counter and render backfacing shadow volume polygons to stencil, decrementing the counter.
3rd pass: Render diffuse and specular where stencil buffer is 0.

and shadowed parts and often involved complex structures like a BSP tree (or even two [Chin and Feiner, 1990]) and although a moving object could be "efficiently" removed and reinserted [Chrysanthou and Slater, 1995], light position changes were almost infeasible for real-time usage.

Heidmann's stencil shadow volumes dramatically improved the usability and performance of the shadow volume technique and received strong support by hardware vendors who added an extension to increment and decrement the stencil buffer depending on a triangles orientation. This enabled the use of a single render pass. Today, a standard texture, alpha blending and a shader could be used to simulate the same counters. Nevertheless, there are several problems with the stencil solution.

Z-fail If the camera viewpoint is in shadow, the intersection count will be wrong. This can be corrected by initiating the count to the number of shadow volumes that the reference point, i.e., camera viewpoint, is located within. It, however, requires some extra geometrical tests to be performed by the CPU. These can be avoided by inverting the depth test [Bilodeau and Songy, 1999; Carmack]. In this situation, all shadow volume fragments are counted that lie behind $\mathbf{p}$ on the view ray from the eye. This technique is referred to as z-fail because counted shadow-volume fragments fail the z-test. By the same logic, the original method of counting from the eye, is often called z-pass. For the z-fail-method, the reference point is no longer the eye, but a point at infinity. The rational is that a point at infinity is always in light. In order for this to be true, the shadow volumes need to be closed. This is handled by closing the volume with the casters' triangles itself. A cap for the shadow volume is created on the one end by drawing the light front-facing triangles, as before, and on the other end by projecting the light back-facing triangles from the light to infinity.

Robust Z-fail Shadow Volumes There is still one problem with shadow volumes that so far has been overlooked in order to get a fully robust algorithm. Upon rendering, the shadow volumes may be clipped by the near plane and/or the far plane, in which case the stencil values becomes erroneous for the regions where clipping occurs. In order fix this problem in a simple and convenient way, Everitt and Kilgard [2002] suggest using an ad-hoc depth-clamping rasterization feature, that was added in graphics hardware by NVIDIA and now has become standard. Instead of clipping the polygons at the near- or far plane, this `NV_DEPTH_CLAMP` extension bounds z-values to the interval of $[0, 1]$. This ensures that no clipping happes at the near and far planes. Thus, the stencil buffer is correctly updated and vertices can simply be sent to a far plane at infinity using homogeneous coordinates.

ZP+ It turns out that *z*-fail, though more robust, is often slower than *z*-pass because it is more likely that fragments lie behind the first visible surface, leading to a large amount of updates to stencil buffer. The idea of ZP+ [Hornus et al., 2005] is to project the scene from the light onto the camera's near plane and thus initialize the stencil buffer from the view with the correct values to allow the application of *z*-pass (cf. Figure 2.7). The algorithm is cheap and theoretically simple, but numerical precision might lead to cracks for single pixels and the correction of these cracks is rather costly, involving a specialized vertex shader. Further, some special cases need to be tested, which makes the code more complex.

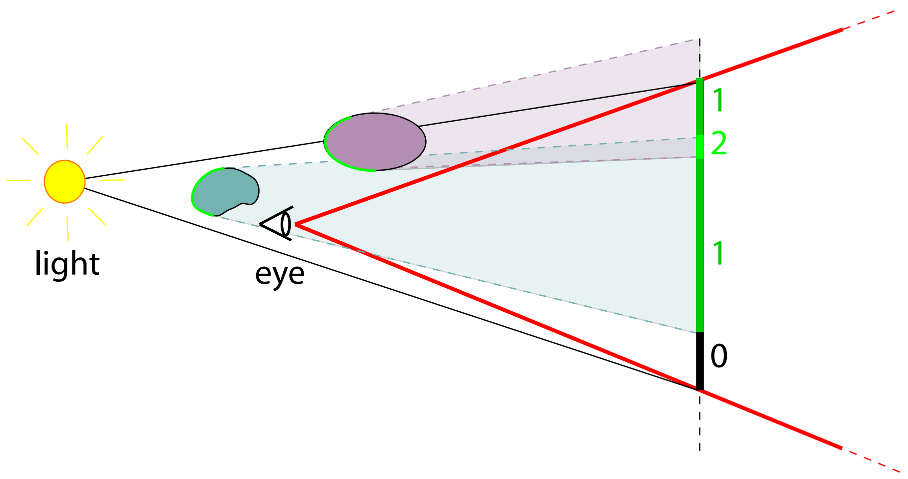


Figure 2.7 ZP+. The idea of the algorithm is to solve the problem with the *z*-pass method of having to initiate the stencil buffer with the number of shadow volumes that the eye are located within. This is done by rendering the near-capping triangles from the light's position, with the far plane identical to the near plane from the eye's position. This initiates the stencil buffer, which can then be used when continuing with the standard *z*-pass from the eye's position.

2.2.1 Improvements

Geometry processing and fill rate are the two bottlenecks of the shadow volume algorithm. In this section we will briefly mention a couple of techniques that have been proposed in order to improve the speed of the stencil updates.

2.2.1.1 Reducing the Geometric Overhead

The geometric overhead of the shadow volume algorithm can be reduced by culling shadow volumes that does not affect the visible end result on the screen.

Culling Lloyd et al.'s CC Shadow Volumes [Lloyd et al., 2004] reduce rendering by using culling of shadow casters that are located completely in shadow and elimination of shadow casters whose shadows are not visible to the eye. Furthermore, shadows that does not influence visible shadow receivers are culled. This is all tested by creating a shadow depth map, from the light's view. In addition, the shadow receivers are rendered (still from the light's view) to the stencil buffer setting the stencil value where the depth test fails, which identifies the shadowed

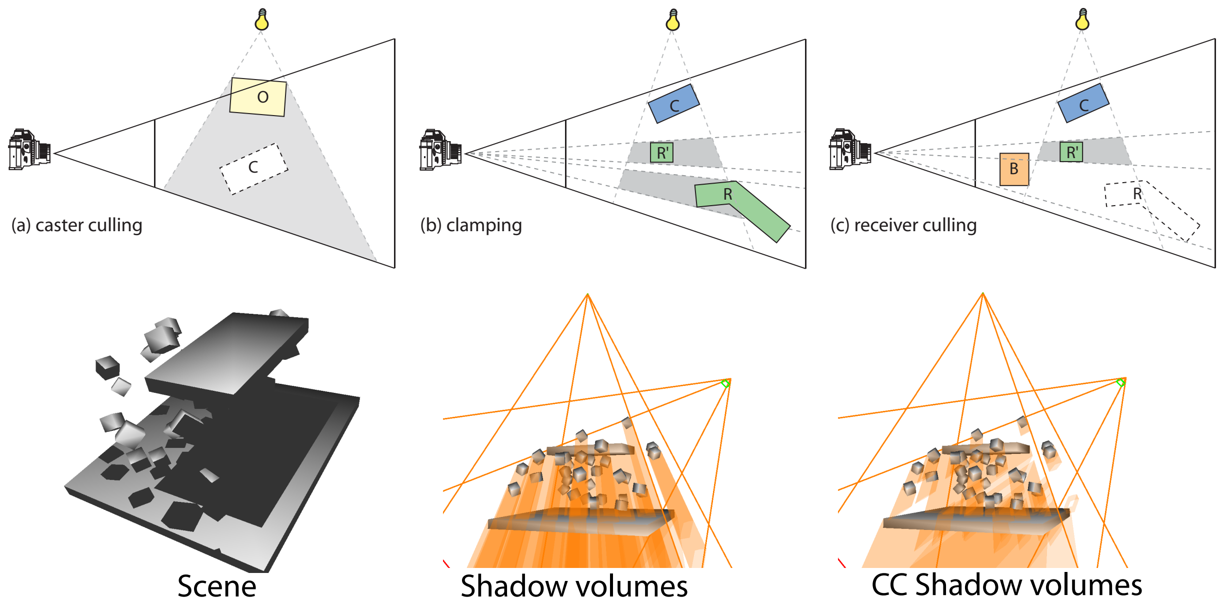


Figure 2.8 *Top:* The principle of shadow volume culling and clamping (a) shadow caster C is fully in the shadow of O so its shadow volume can be culled (b) the shadow volume for C need only to extend through regions containing shadow receivers (c) if a shadow receiver R is not visible from the viewpoint, the shadow volume for C does not need to be rendered around it. *Bottom:* Figure showing the overdraw from standard shadow volumes vs shadow volumes with culling and clamping. (Figure from Eisemann and Décoret [2006a]).

regions. Then, occlusion queries are used when rendering the bounding boxes of the shadow casters, from the viewpoint of the light, with the depth test plus the stencil test enabled (where the stencil test fails for set values). If no pixel is rendered, the shadow caster is either in shadow or does not influence any visible shadow receiver, in which case the caster's shadow volumes can safely be ignored.

A more recent approach to reduce the actual number of necessary shadow volumes is described in [Stich et al., 2007]. Here a scene hierarchy is exploited and only visible shadow volumes intervene in the shadow computations; this is conservatively estimated based on occlusion culling using bounding-volume hierarchies.

2.2.1.2 Improving Fill Rate

Split-shadow volumes The observation that speed can be gained by fewer stencil updates motivated split-shadow volumes [Laine, 2005]. The algorithm is based on the fact that from one *object* to the next, one can toggle between z-fail and z-pass if we assure that the stencil buffer is modified in a coherent way. The per-object toggle is decided on a per-pixel basis. The choice of using z-fail and z-pass is based on which one is likely to be the fastest. It is an attractive idea, but in practice, still lacking hardware support and thus it is currently not being efficient.

We will now explain the idea of the algorithm. Let's for the moment assume one single shadow volume. The stencil operations are set such that, if one side of the shadow volume lies in front and the other behind the first visible (impact) point $\mathbf{p}$, both methods (z-fail and z-pass) will result in a stencil buffer containing the value one. In other words: z-pass should increment the stencil for the visible/front-facing and z-fail for invisible/back-facing quads. If the shadow volume lies entirely in front or entirely behind **p**, both (z-fail and z-pass) lead to a value of zero in the stencil buffer.

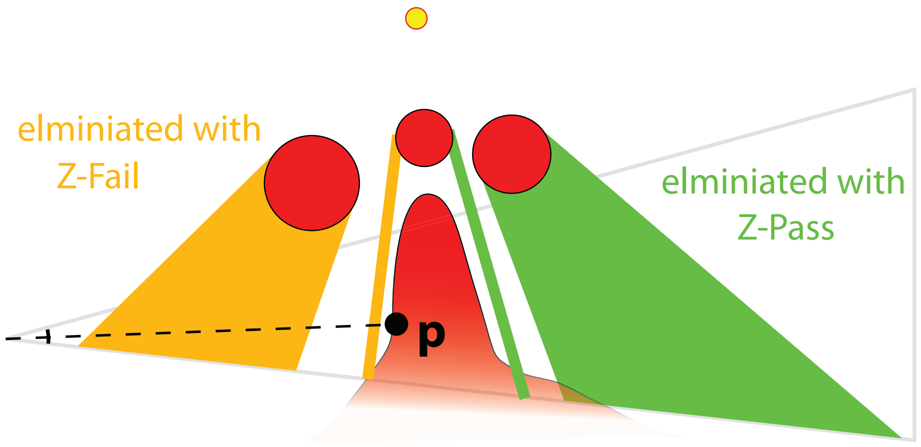


Figure 2.9 Split-Shadow Volume

Using early z-culling, it is possible to reduce unnecessary stencil updates. The observation is that for z-fail the stencil buffer is modified solely by fragments that lie behind **p**, whereas for z-pass the stencil buffer is modified only if fragments are in front of **p**. Therefore, the better choice of the two options is the one that is more likely to not perform any update for a given object. Which means that z-fail should be chosen if it is more likely that the entire volume lies in front of **p**. In the opposite case z-pass should be used.

To select an appropriate strategy, an additional value z_{split} is computed per pixel and object. The paper defines two ways to do this, but basically it just aims at drawing a quad that approximately splits the shadow volume. This quad defines a barrier that toggles between the two methods. Based on the depth of z_{split} and the depth of **p**, a per-pixel choice is made. If **p** is closer than z_{split}, the volume is likely to be behind **p**, and thus z-pass is chosen (eliminating all fragments behind **p**). In the opposite case, z-fail is used.

Of course, one would not gain much if z_{split} was rasterized at full view-resolution because then each object would draw one supplementary quad leading to an again increased fill-rate. But because the z_{split} values have no influence on the correctness of the result, only on performance, it is possible to render them in a much lower resolution buffer; Optimally, one whose resolution matches the highest level in the hierarchical z-buffer [Greene et al., 1993] to ensure the best early culling behavior.

Hybrid of Shadow Maps and Shadow Volumes Chan and Durand suggested to have two shadow passes. They first apply a shadow mapping algorithm where all pixels in the view will be marked if they lie on shadow boundaries (detected as discontinuities in the depth map). The marked pixels will be the only ones updated during the following shadow volume pass. To achieve this, the authors make use of the early Z capabilities of the graphics card. By setting the depth buffer to block all fragments outside the marked region, the hierarchical depth representation in the hardware takes care of discarding large areas of fragments. A variant has been

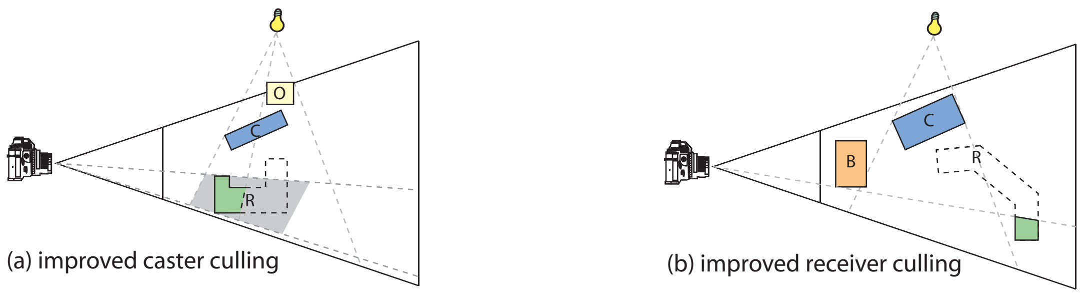


Figure 2.10 Cases that benefit from pixel-accurate culling. On the left, instead of just culling the shadow volume of C around the whole receiver R, the shadow volume can be clamped even tighter, since a part of R (the one dashed) is actually shadowed by another caster O. The right example shows a receiver that is visible by the observer but cannot actually reveive shadows, which also can be accounted for with pixel-accurate culling. (Figure from Eisemann and Décoret [2006a]).

presented by Aila and Akenine-Möller [2004] who compute intersections of the shadow volume with pixel tiles and mark them as boundaries. They then restrict shadow computation to one tile per region plus these shadow limit tiles. The solution is accurate, but currently slower.

Clamping In addition to using culling (see above) Lloyd et al.'s CC Shadow Volumes [Lloyd et al., 2004] also uses clamping of the shadow volumes to avoid unnecessary rasterization. They use two different and orthogonal methods, in the sense that they preferably should be used in combination. The first is called Continuous Shadow Clamping, where shadow volumes are clamped by using AABBs around the shadow receivers to achieve clamped shadow volumes only around the regions of interest. The second is called Discrete Shadow Clamping and clamps the shadow volumes to intervals defined by slicing planes that divides the view frustum into layers with slicing planes facing the light source and passing through the viewpoint. The rational is that for a given layer, the potential receivers are rendered with the two delimiting planes as clipping planes. In addition, the projection (from the light source) of each caster on the furthest of the two delimiting planes is rendered with a depth test. If no fragment passes the depth test (tested with an occlusion query), the shadow volume can be clamped for layers that lies further from the light source.

The same principle was improved upon by Eisemann and Décoret [2006a], who present a more efficient variant. Instead of rasterizing the entire scene multiple times, once for each slice, Eisemann and Décoret use a single pass voxelization technique to derive the layers. They further propose pixel-based instead of object-based culling to improve upon the culling and clamping behavior. Figure 2.10 shows some cases that benefit from pixel-accurate culling. They futher avoid rendering receivers into the slices, if their corresponding pixels are hidden from the viewpoint. This idea was based on the litmap approach [Décoret, 2005], but here it allows more shadow volumes to be clamped.

2.3 Summary

When choosing a shadow algorithm and trying to decide whether shadow maps or shadow volumes is the better choice, there are the following general characteristics to consider. Shadow maps are generally faster. The cost is roughly the same as the cost involved in rendering the image for the viewpoint. Shadow maps can also generate shadows from any rasterizable geometry, in contrast to shadow volumes, where the shadow casting geometry should be polygonal in order to be able to extract silhouette edges. The downside of shadow maps is the biasing issues, under-sampling artifacts in the form of jaggy shadow edges and the limitation to a single frustum, so that omnidirectional lights typically requires six shadow maps. Shadow volumes, on the other hand, produce perfectly sharp shadows, but are considered slow. They require three render passes, but more severely, the elongated quads of the shadow volumes cause a high fill-rate. In addition, extracting silhouette edges is often considered expensive.

3 Hard Shadows

Even though shadow algorithms have been around for almost as long as computer graphics itself, robust and efficient hard shadow generation is still not a solved problem. While geometry-based algorithms produce pixel-perfect results, they suffer from robustness problems with different viewer-light constellations, and are often slow due to the enormous overdraw involved in rasterizing shadow volumes.

Shadow map algorithms, on the other hand, are very fast as their complexity is similar to standard scene rendering, but they suffer from aliasing artifacts since the sampling of the shadow map and the sampling of the image pixels projected into the shadow map usually do not match up.

In this chapter, we will discuss several methods to reduce shadow map aliasing artifacts. First, we will analyze aliasing in more detail and show the different components of aliasing. Then we will show different strategies to reduce sampling error. Finally, we will give a cookbook to select an appropriate shadow algorithm in different situations.

3.1 Shadow Map Aliasing

The quality of shadow mapping is plagued by a number of different sampling errors. We first give an overview of these errors and common solution strategies, and then discuss two methods to characterize the most prominent error type, sampling error.

3.1.1 Different Types of Error

It is helpful to think about shadow mapping as a signal reconstruction process similar to texture mapping. Signal reconstruction has the following steps:

1. *Initially sample* an (ideal) input function, i.e., generate the shadow map using rendering. The signal in this case is the actual shadow in screen space, i.e., the projection of the (continuous) depth map onto the scene, seen from the camera view point. Since no bandlimiting is possible to avoid aliasing in the initial sampling phase, the sampling frequency should ideally be higher than the Nyquist frequency of the signal. However, since the actual shadow in screen space has sharp edges and therefore unlimited frequencies, the sampling rate should at least equal the screen sampling rate.

2. *Reconstruct* the signal from its sampled representation.

3. *Resample* the reconstructed signal at the final pixel positions

The main types of error are

- Undersampling, which occurs when the shadow map samples projected to the screen have a lower sampling rate than the screen pixels. This is due to a too low initial sampling frequency.
- Reconstruction error or staircase artifacts, which are due to nearest neighbor reconstruction.
- Oversampling, which happens when the shadow map samples projected to the screen have a higher sampling rate than the screen pixels. In this case, the classical aliasing known from texture sampling occurs.

Reconstruction error can be reduced by using a better reconstruction filter. In case of shadow mapping, percentage-closer filtering (PCF) [Reeves et al., 1987] (Chapter 4) is the equivalent to bilinear or higher-order reconstruction for texture mapping. Basically, PCF treats shadows as a projected texture by first evaluating the shadow function and then applying the filter kernel. Better reconstruction can also be achieved by changing the reconstruction algorithm itself (see sidebox).

Aliasing due to *oversampling* is usually avoided in image processing by band-limiting the reconstructed signal before resampling it. For texture mapping, prefiltering approaches such as mip-mapping are most common. However, this is much harder to do for shadow mapping since the shadow function is not linear, and therefore the bandlimiting step cannot be done before rendering. One option is to evaluate PCF with large filter kernels, however this is slow and does not scale. Recent research proposed clever ways to reformulate the shadow test into a linear function so that prefiltering can be applied (see Chapter 4).

However, by far the most research effort has been invested into fixing the problem at its root, namely the *initial sampling* error. Unlike texture mapping, where the resolution of the input image is usually predetermined, in shadow mapping there is significant control over the original sampling step. Therefore, it is possible to adapt the sampling so that the projected shadow map samples correspond much better to the screen space sampling rate than naive shadow mapping. In standard texture mapping, most of the burden lies on the reconstruction filter in magnification scenarios. In shadow mapping, this burden can be reduced by increasing the sampling rate and thus removing the magnification (or undersampling) from affected areas, so that even nearest neighbor reconstruction can sometimes give good quality. Furthermore, the need for prefiltering can be reduced by reducing the initial sampling rate in areas which appear small on screen.

Another error that needs to be taken into account in all shadow map approaches is temporal aliasing, which will appear especially for non-optimal reconstruction if undersampling occurs. This manifests itself in flickering artifacts if the rasterization of the shadow map changes each frame.

In the following, we will concentrate on sampling error introduced in the initial sampling phase. We first analyze this error in more detail, and then present strategies to avoid it.

3.1.2 Simplified Sampling Error Analysis

At the root of most algorithms to reduce shadow map sampling errors is an analysis of the distribution of errors in a scene. A simplified error analysis was first introduced by Stamminger and

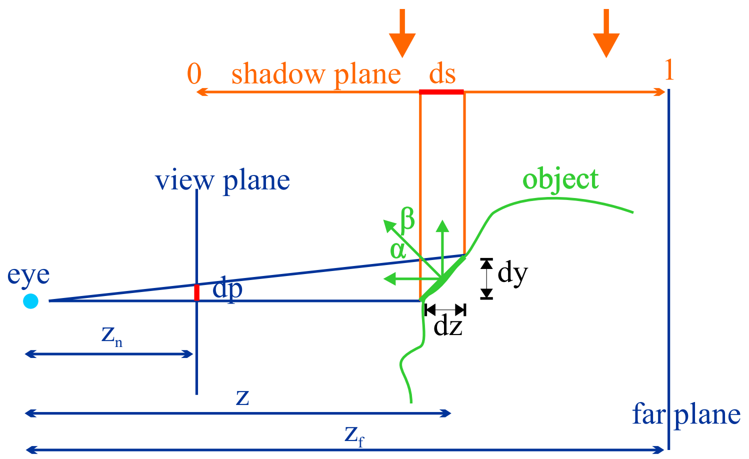


Figure 3.1 Aliasing in shadow mapping.

Drettakis [2002] for Perspective Shadow Maps, and the same formula has been used in many subsequent approaches. The analysis assumes an overhead directional light and looks at a surface element located somewhere on the z-axis of the view frustum. Figure 3.1 shows a configuration for a small edge.

A pixel in the shadow map represents a shaft of light rays passing through it and has the size $ds \times ds$ in the local parametrization of the shadow map. We assume a local parametrization of the shadow map which goes from 0 to 1 between near and far planes of the viewer – this already assumes that the shadow map has been properly focussed to the view frustum, not wasting any resolution on invisible parts of the scene (see Section 3.2.1). In world space, the shaft of rays has the length $dz = (z_\mathrm{f} - z_\mathrm{n})ds$ for uniform shadow maps as an example.

The shaft hits a small edge along a length of $dz/\cos\beta$. This represents a length of $dy = dz\frac{\cos\alpha}{\cos\beta}$ in eye space, projecting to $dp = dy/z$ on screen (assuming a near plane distance of 1). Note that we assume that the small edge can be translated along the z-axis. The shadow map aliasing error dp/ds is then

$$\frac{dp}{ds} = \frac{1}{z}\frac{dz}{ds}\frac{\cos\alpha}{\cos\beta}. \tag{3.1}$$

Shadow map *undersampling* occurs when dp is greater than the size of a pixel, or, for a viewport on the near plane of height 1, when dp/ds is greater than $\mathrm{res}_{\mathrm{shadowmap}}/\mathrm{res}_{\mathrm{screen}}$. As already shown by Stamminger and Drettakis [2002], this can happen for two reasons: *perspective aliasing* when $\frac{dz}{zds}$ is large, and *projection aliasing* when $\cos\alpha/\cos\beta$ is large.

Projection aliasing is a local phenomenon that occurs for surfaces almost parallel to the light direction. Reducing this kind of error requires higher sampling densities in such areas. Only approaches which adapt the sampling density locally based on a scene analysis can achieve this (Sections 3.2.4 to 3.2.6).

Perspective aliasing, on the other hand, is caused by the perspective projection of the viewer. If the perspective foreshortening effect occurs along one of the axes of the shadow map, it can be influenced by the *parametrization of the shadow map*. If a different parametrization is chosen, this will lead to a different sampling density distribution along the shadow map. The standard uniform parametrization has dz/ds constant, and therefore the sampling error dp/ds is large when

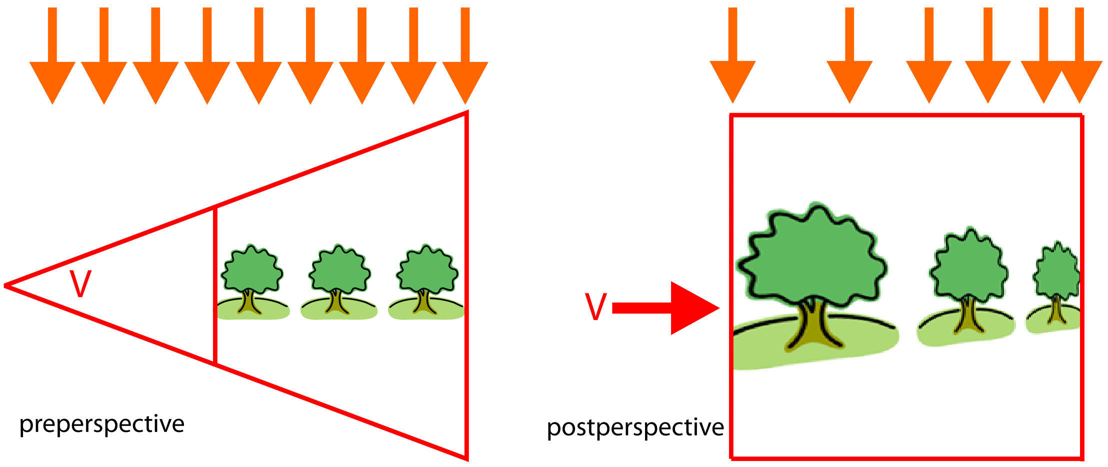


Figure 3.2 The uniform distribution of a shadow map in world space (left) degrades near the observer due to perspective foreshortening. This effect is visible in post-perspective space (right). Much less samples are spent on nearby elements.

$1/z$ is large, which happens close to the near plane, which is very visible (compare Figure 3.2). In order to reduce perspective aliasing, there are several approaches to distribute more shadow map samples near the viewer, either by using a different parametrization, or by splitting the shadow map into smaller parts (Sections 3.2.2 and 3.2.3).

3.1.3 Accurate Sampling Error Analysis

An accurate analysis of sampling error is somewhat involved and can be studied in Brandon Lloyd's article and thesis [Lloyd et al., 2008; Lloyd, 2007]. Here we just give the result. For a general configuration, the aliasing error m is

$$m = \frac{r_j}{r_t}\frac{dG}{dt}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\phi_e}\frac{\cos\psi_e}{\cos\psi_l}. \tag{3.2}$$

In this formulation (see also Figure 3.3),

- $\frac{r_j}{r_t}$ is the ratio of the screen and shadow map resolutions
- $\frac{dG}{dt}$ is the derivative of the shadow map parametrization (called dz/ds above)
- $\frac{W_l}{W_e}$ is the ratio of the world space widths of the light and eye viewports
- $\frac{n_e}{n_l}$ is the ratio of the near plane distances of eye and light
- $\frac{d_l}{d_e}$ is the ratio of the patch distances from the light and from the eye (d_e corresponds to z above)
- ϕ_l, ϕ_e are the angles of the light and eye beams from the image plane/shadow map plane normals

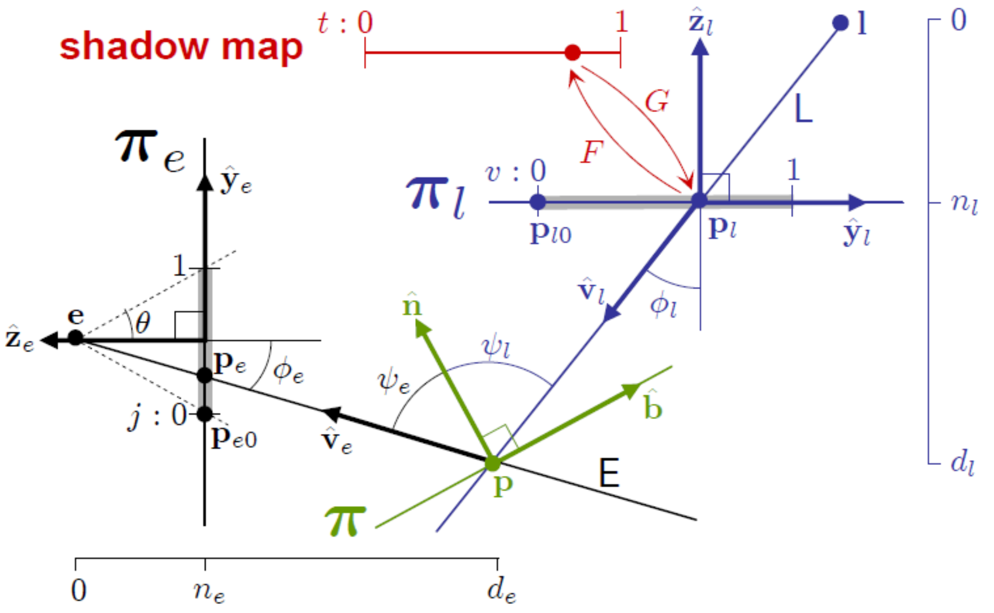


Figure 3.3 Notation used in the accurate aliasing description (image courtesy of Brandon Lloyd).

- ψ_l, ψ_e are the angles between light and eye beams from the surface normal of the patch (corresponding to α, β above).

In comparison to the simplified analysis, this formulation takes into account the variations in sampling error when the surface element is not in the center of the view frustum, and for arbitrary light positions or directions. It also correctly accounts for point lights. For directional lights, n_l/d_l converges to 1 and $\cos\phi_l$ will be constant. Shadow map undersamping occurs when $m > 1$.

An important point to consider is that these formulations only treat one shadow map axis (if you look at Figure 3.2, the axis orthogonal to the plane this paper is printed on is ignored). In practice, it is necessary to consider sampling error in both directions.

3.2 Strategies to Reduce Sampling Error

3.2.1 Fitting

One of the most straightforward ways in which the sampling rate can be improved is to make sure that no shadow map space is wasted. Especially in outdoor scenes, if a single shadow map is used for the whole scene, then only a small part of the shadow map will actually be relevant for the view frustum. Thus, fitting or focusing techniques, first introduced by Brabec et al. [2005], fit the shadow map frustum to encompass the view frustum.

The geometrical solution is to calculate the convex hull of the view frustum and the light position (for directional lights this position is at infinity) and afterwards clip this body with the scene bounding volume and the light frustum (see [Wimmer et al., 2004; Wimmer and Scherzer, 2006] for details). Clipping to the scene bounding volume is necessary because today very large view frusta are common and they frequently extend outside the scene borders. We call the resulting body the intersection body B (see Figure 3.4).

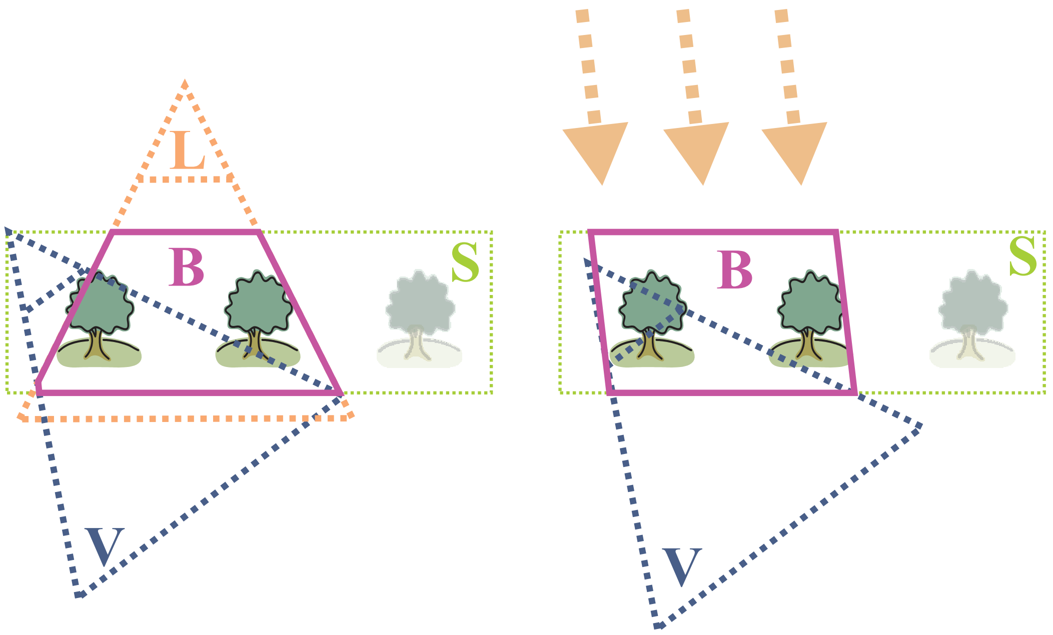


Figure 3.4 Intersection body B used for focusing. Left: point light. Right: directional light.

The intersection body can be further reduced using visibility algorithms. If, before the shadow map is created, a first depth-only pass is rendering with an online visibility algorithm like CHC [Mattausch et al., 2008], the far plane distance can be reduced to just cover the furthest visible object.

In general, fitting leads to temporal aliasing because the rasterization of the shadow map changes each frame. Especially when using visibility information, strong temporal discontinuities can occur, so this should only be used if a good reconstruction filter is used.

Temporal aliasing due to fitting can also be somewhat reduced by trying to keep texel boundaries constant in world space. First, the shadow map needs to maintain a constant orientation in world space in order to avoid projected shadow map texels to change shape whenever the viewer rotates. For this, the shadow map needs to be focused on the axis-aligned bounding box of the intersection body. To avoid aliasing due to translation of the view frustum in the shadow map view, the shadow map should be created with one texel border and only refit if the view frustum moves a whole texel. However, most viewer movements also lead to a scaling of the view frustum in the shadow map view, and this is more difficult to control without wasting much shadow map space, see [Zhang et al., 2009] for more details.

3.2.2 Warping

When projecting the view frustum into the shadow map, it becomes apparent that higher sampling densities are required near the viewpoint and lower sampling densities far from the viewpoint. In some cases, it is possible to apply a single transformation to the scene before projecting it into the shadow map sucht that the sampling densities is globally changed in a useful way. This was first discovered by Stamminger and Drettakis [2002] in used in Perspective Shadow Maps (PSM).

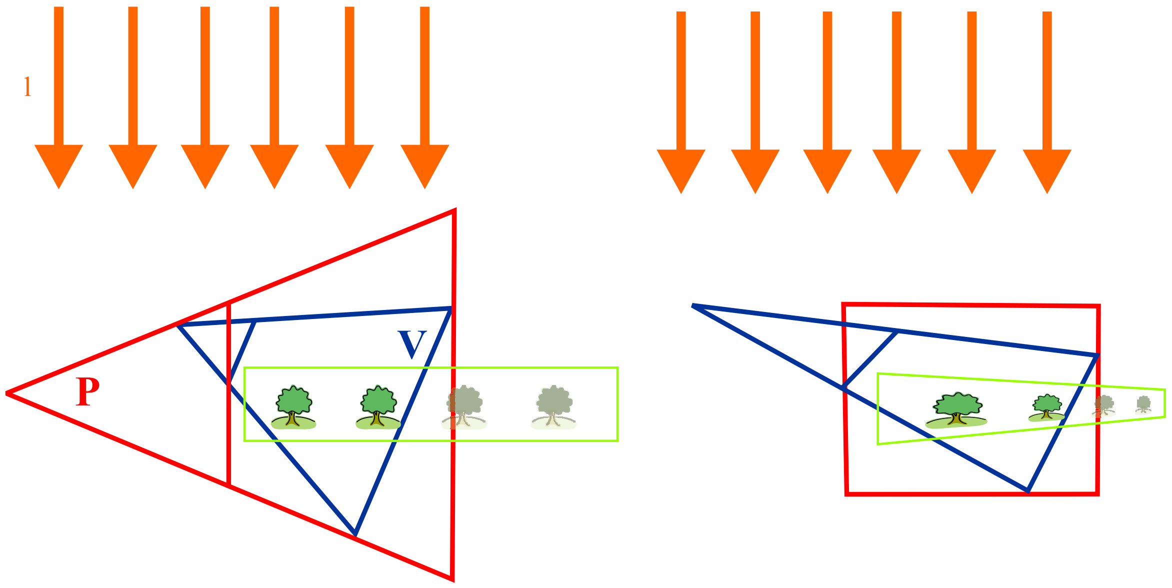


Figure 3.5 An example configuration of light space perspective shadow maps with view frustum V and the frustum defining the perspective transform P. Left: directional light, a view frustum V, and the perspective transformation P. Right: after the warp, objects near the viewer appear bigger in the shadow map and therefore receive more samples.

It can be shown that a logarithmic transformation along the z-axis of the viewer provides an optimal sampling rate for the whole depth range in the view frustum [Wimmer et al., 2004], however, this requires logarithmic rasterization, which is currently infeasible.

Practical warping schemes use perspective transformations to redistribute samples towards the near plane [Wimmer et al., 2004; Martin and Tan, 2004; Chong, 2003].

In the original algorithms, warping was applied to a single shadow map, however it has been later combined with partitioning algorithms to further improve sampling rates.

3.2.2.1 Light Space Perspective Shadow Mapping

As a representative of warping algorithms we discuss light space perspective shadow mapping (LiSPSM) introduced by Wimmer et al. [2004], which is mostly equivalent to Trapezoidal Shadow Maps (TSM) introduced independently by Martin et al. [2004]. In a very insightful work, Lloyd et al. [2006] proved that all perspective warping algorithms (PSM, LiSPSM, TSM) actually lead to the same *overall* error when considering both shadow map directions, but LiSPSM gives the most even distribution of error among the directions and is therefore advantageous.

The main idea of perspective shadow mapping is to apply a perspective transformation to the scene before rendering it into the shadow map. Thus, the distribution of shadow map samples is changed so that more samples lie near the center of projection, and less samples near the far plane of the projection. See Figure 3.5 for an illustration.

Perspective shadow mapping methods differ in the way this perspective transformation is set up. In the original PSM, this transform was chosen to be equivalent to the viewer projection. However, this changes the direction of the light or even the type of the light (from directional to point or vice versa), and also distributes the error in a non-optimal way. In LiSPSM, the

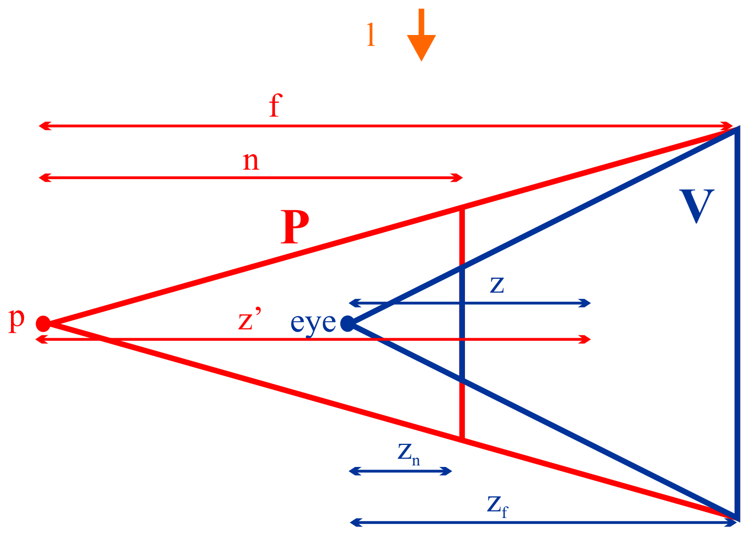


Figure 3.6 The parametrization of light space perspective shadow maps (shows the yz-plane in light space). The parameter n is free and can vary between z_n (perspective shadow mapping) and infinity (uniform shadow mapping).

perspective transformation is always aligned to the axis of the light frustum, and therefore lights do not change direction or type (see Figure 3.6). In order to deal with point lights, the projection of the point light is applied first, converting the point light to a directional light, and LiSPSM is done in the post-perspective space of the light. This transformation is the same as the one depicted in Figure 3.2, when interpreting the viewpoint as a light source.

There is one free parameter in perspective shadow mapping, namely the near plane distance n of the perspective transformation. A small distance leads to a stronger warp and more focus on nearby objects, a larger n leads to a less strong warp.

Trapezoidal shadow maps (TSM) are set up in exactly the same way as LiSPSM, but use a different choice for n.

3.2.2.2 Logarithmic Warping

Consider again the simplified error formulation shown in Equation 3.1. An *optimal parametrization* would make dp/ds constant (= 1 assuming equal screen and shadow map resolutions) over the whole available depth range. For the ideal case of view direction perpendicular to light direction, this is (constants notwithstanding) equivalent to [Wimmer et al., 2004]

$$ds = \frac{dz}{z}, \text{ i.e., } s = \int_0^s ds = \int_{z_n}^{z} \frac{dz}{z} = \ln \frac{z}{z_n}.$$

This shows that the optimal parametrization for shadow mapping (at least for directional lights) is logarithmic. In more recent work, Lloyd at al. [2008] have revisited the logarithmic mapping and combined it with a perspective warp (LogPSM). In a very involved mathematical treatise, they derive warping functions that approach the optimal constant error very closely, based on the exact sampling error formulation. They also consider fully general 3D configurations.

Unfortunately, such a parametrization is not practical for implementation on current hardware, but Lloyd et al. [2007] propose simple modifications to the rasterization pipeline to make it feasible.

The logarithm could be applied in a vertex program, however, pixel positions and all input parameters for pixel programs are interpolated hyperbolically. This makes graphics hardware amenable to perspective mappings, but not logarithmic ones. As a proof of concept, logarithmic rasterization can be evaluated exactly in the fragment shader by rendering quads that are guaranteed to bound the final primitive, but this is too slow for practical implementation.

3.2.2.3 Optimal Warping Parameter for Perspective Warping

As mentioned above, there is a free parameter for P in perspective warping methods, namely, the distance n of the projection reference point $\mathbf{p}$ to the near plane. This parameter influences how strong the shadow map will be warped. If it is chosen close to the near plane of P, perspective distortion will be strong, and the effect will resemble the original perspective shadow maps (where n is chosen the same as the view frustum near plane distance). If it is chosen far away from the far plane of P, the perspective effect will be very light, approaching uniform shadow maps. It can be shown that in the case of a view direction perpendicular to the light vector, the optimal choice for this parameter is [Wimmer et al., 2004]

$$n_{\mathrm{opt}} = z_{\mathrm{n}} + \sqrt{z_{\mathrm{f}} z_{\mathrm{n}}},$$

where z_{n} and z_{f} are the near and far plane distances of the eye view frustum. Figure 3.7 compares the aliasing error along the viewer z-axis for uniform shadow maps, perspective shadow maps with warping parameter as in the original PSM paper, and the optimal warping parameter. Note, however, that this analysis only treats errors in the shadow map direction aligned with the z-direction. Considering the x-direction, the PSM parameter actually leads to an optimal constant error, however, as can be seen in the plot, the error along the z-direction is very uneven and leads to very bad shadow quality when moving away from the viewer. The optimal LiSPSM paramter leads to an even distribution of error among the two axes.

When the viewer is tilted towards the light or away from it, n has to be increased, so that it reaches infinity when the viewer looks exactly into the light or away from it. In this case, perspective warping cannot bring any improvements, therefore no warping should be applied.

In the original LiSPSM paper, a falloff depending on the angle γ between the shadow map normal vector and the view plane normal vector was introduced, by $n'_{\mathrm{opt}} = n_{\mathrm{opt}} / \sin\gamma$. However, Lloyd [2007] later showed that this falloff is not fast enough once the angle passes the point where one of the view frustum planes becomes parallel to the shadow map normal. He proposes a different falloff function that avoids this problem, but is a bit too involved to reproduce here, so we refer the reader to Section 5.1.2.1 and 5.2.1 of [Lloyd, 2007] for the exact equations. Another falloff function has been proposed in [Zhang et al., 2006b], but it only takes error along the z-axis into account.

Another interesting extension is to use a different view frustum near plane distance for the computation of n. The rationale is that the nearest depth range (e.g., between 0.01 and 1) does not often show shadows, but a lot of precision is wasted on this range using the optimal warping parameter. Lloyd describes using a pseudo-near plane in his thesis in Section 5.1.9.

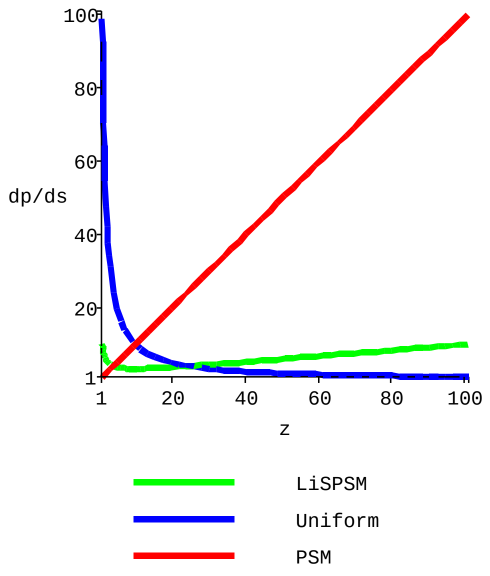


Figure 3.7 Perspective aliasing errors plotted against z-coordinate for different shadow mapping techniques for an overhead directional light.

3.2.3 Global Partitioning

While warping works very well in some configurations, especially if the light is overhead, there are other configurations where no benefit can be reaped, for example if the light is directly behind the viewer. In this case, one global perspective warp will not change the sampling densities along the z-axis of the viewer, and therefore warping degenerates to uniform shadow mapping. A much better alternative is to use more than one shadow map.

3.2.3.1 Z-Partitioning

The most prominent approach and one of the most practical algorithms is to subidivide the view frustum along the z-axis, and calculate a separate equal-sized shadow map for each sub-frustum. This algorithm goes by the names of Cascaded Shadow Maps (CSM) [Engel, 2006], Parallel Split Shadow Maps (PSSM) [Zhang et al., 2006a], or z-partitioning [Lloyd et al., 2006]. Figure 3.8 shows an example of a PSSM where the view frustum is split into three partitions, and the shadow map for the middle partition map is shown. Using this approach, the sampling density decreases for each successive partition, because the same number of shadow map samples cover for a larger and larger area.

In the most naive implementation, a PSSM scheme with n partitions requires n shadow rendering passes. Zhang et al. [2007] describe different methods to reduce the number of rendering passes, for example by using the geometry shader to replicate each triangle into each of the required shadow maps during the shadow rendering pass. On the other hand, they also show a multipass method that does not require shaders and runs on older hardware.

The most important question in this method is where to position the split planes. One way is to go back to the derivation of the shadow map resampling error. Each sub-shadow map could be

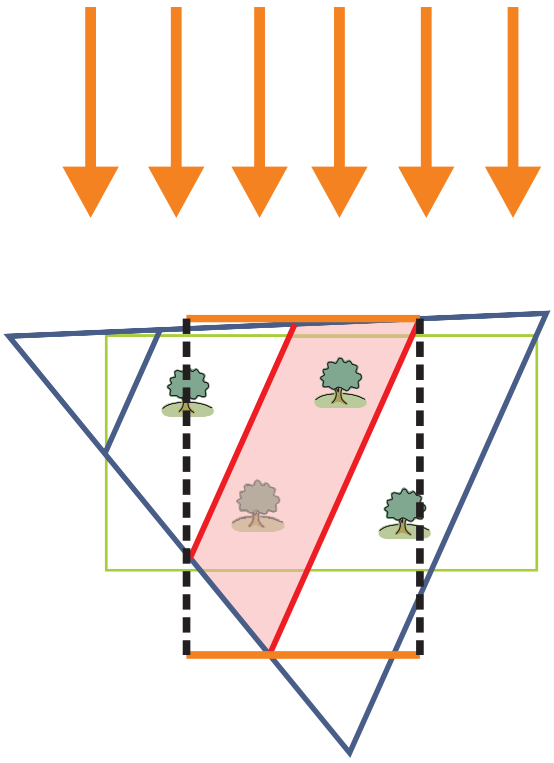


Figure 3.8 PSSM: the shadow map for the middle of three partitions of the view frustum (side view).

interpreted as a big texel of a global shadow map, so that z-partitioning becomes a discretization of an arbitrary warping function. We have shown before that the optimal warping function is logarithmic, therefore the split positions C_i should be determined as [Lloyd et al., 2006]:

$$C_i = z_\mathrm{n}\left(\frac{z_\mathrm{f}}{z_\mathrm{n}}\right)^{\frac{i}{m}}$$

where m is the number of partitions. However, as opposed to global warping schemes, the effect of z-partitioning is not limited to the axes of the shadow map, but even works when the light is directly behind the viewer. This is the main advantage of z-partitioning over warping approaches, and the reason why z-partitioning is much more robust in general configurations. Figure 3.9 shows the nearest and farthest partition in a situation with the light directly behind the viewer. The shadow map for the nearest partition covers a much smaller area, and therefore the perceived resolution is higher, just as is the case for the viewer projection.

Zhang et al. [2006a] note that the optimal partition scheme is often not practical because it allocates most resolution near the near plane, which is often not populated with objects. They therefore propose computing the split positions as weighted average between the logarithmic scheme and a simple equidistant split plane distribution. An alternative solution that better respects the theoretical properties of shadow map aliasing is to use a pseudo-near plane just as in warping. This approach is explained in Lloyd's thesis [2007] in Section 5.1.8.

Z-partitioning can be combined with warping by rendering each partition using LiSPSM. This increases quality especially for situations where LiSPSM works well (overhead lights). One special case of such a combination is to use one uniform shadow map and one perspective shadow

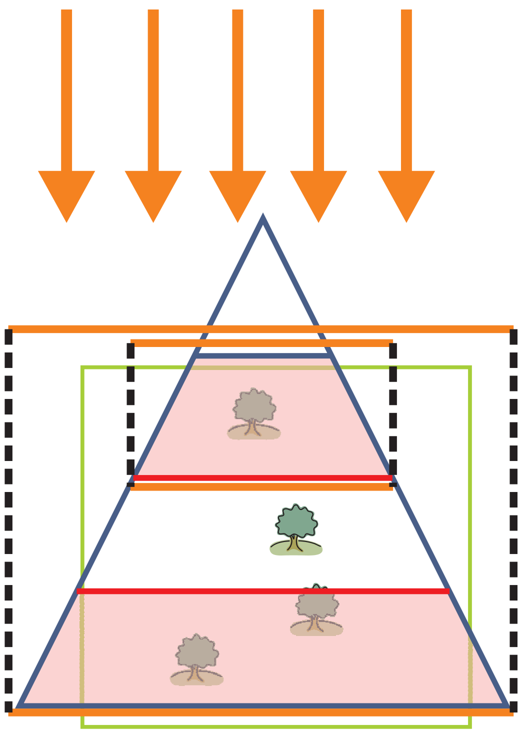


Figure 3.9 The nearest shadow map covers a smaller area than the farthest shadow map and therefore leads to higher resolution.

map and calculate a plane equation that separates areas where the one or the other provides the best quality [Mikkelsen, 2007].

Zhang et al. [2009] also discuss a number of practical issues related to z-partitioning, regarding flickering artifacts, shadow map storage strategies, split selection, computation of texture coordinates, and filtering across splits. An interesting observation is that in some cases, a point belonging to one partition should be shadowed using a shadow map generated for a different partition. This happens when the light is almost parallel to the view direction. In this case, the shadow maps for the partitions nearer the view point will provide better resolution.

3.2.3.2 Frustum Face Partitioning

Another alternative is to use a separate shadow map for each face of the view frustum as projected onto the shadow map plane, and use warping for each shadow map separately. This can also be interpreted as putting a cube map around the post-perspective view frustum and applying a shadow map to each cube face. Each frustum face can be further split to increase quality.

This scheme is especially important because it can be shown that it is optimal for LogPSM, i.e., the combination of logarithmic and perspective shadow mapping introduced by Lloyd et al. [2008]. However, we will not elaborate this scheme here because Lloyd et al. [2007] also showed that for practical situations, i.e., a large far plane to near plane ratio and a low number of shadow maps, z-partitioning (optionally combined with warping) is superior to frustum partitioning.

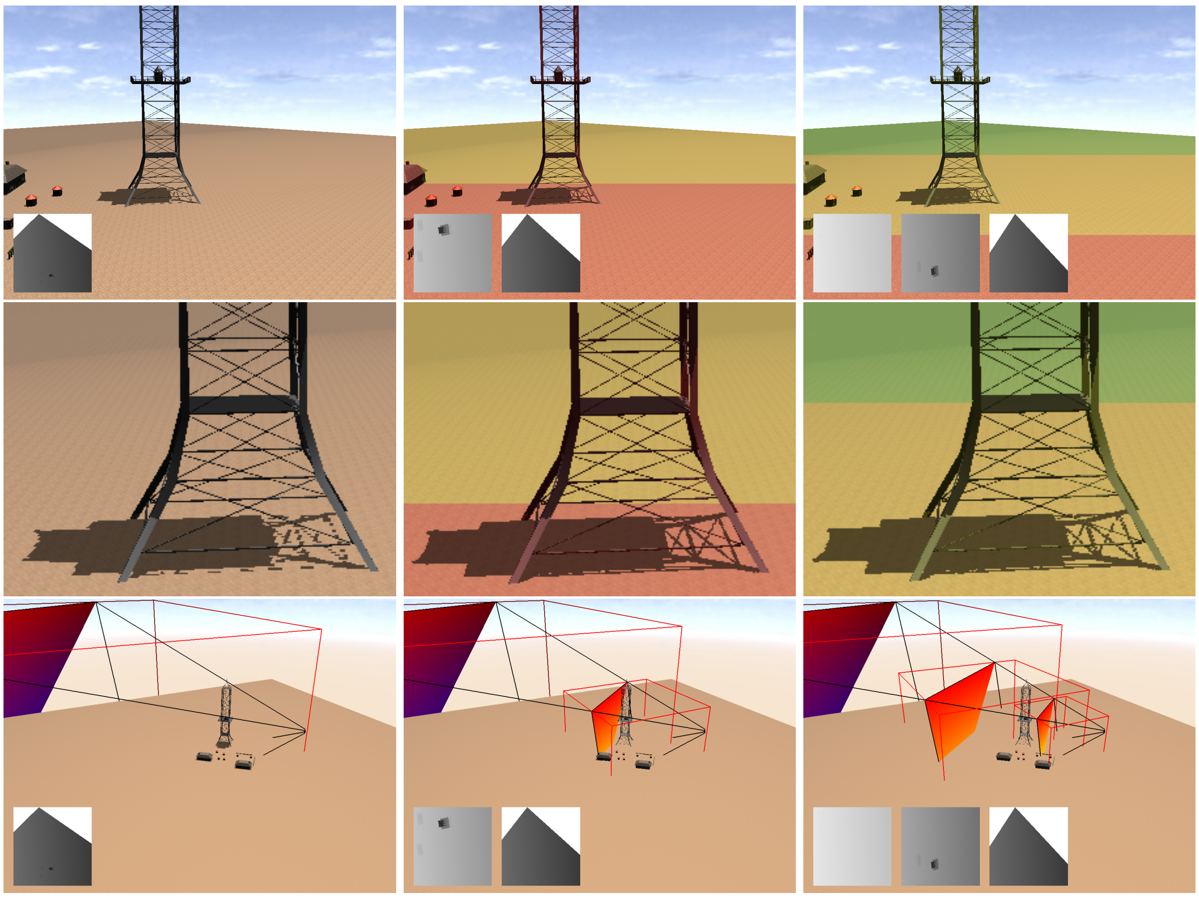


Figure 3.10 Examples of z-partitioning with 1 to 3 partitions. The shadow map used for each fragment is color coded. Top row: Camera view. Middle row: close-up of shadowed region. Bottom row: outside view showing the view frustum, the partitions, and the intersection bodies. Inlays show the depth maps.

3.2.3.3 Examples of Warping and Partitioning

Figure 3.10 shows the effect of z-partitioning. The split distances are chosen according to Zhang's practical split scheme. Figure 3.11 shows the effect of z-partitioning vs. warping and the combinations of the two. Warping alone has a similar effect as z-partitioning, and can further improve the quality of z-partitioning if used in combination. Figure 3.12 shows a case that is not amenable to warping due to a nearly overhead light, while z-partitioning still improves quality. All examples use a directional light source.

Figure 3.13 shows the overall error (here called storage factor), which takes into account error in both shadow map directions, of different schemes for different numbers of shadow maps for overhead lights (ideal for warping) and a light behind (no warping possible).

3.2.4 Adaptive Partitioning

The advantage of global partitioning algorithm is that they are very fast. On the other hand, they completely ignore surface orientation and can therefore not improve undersampling due to

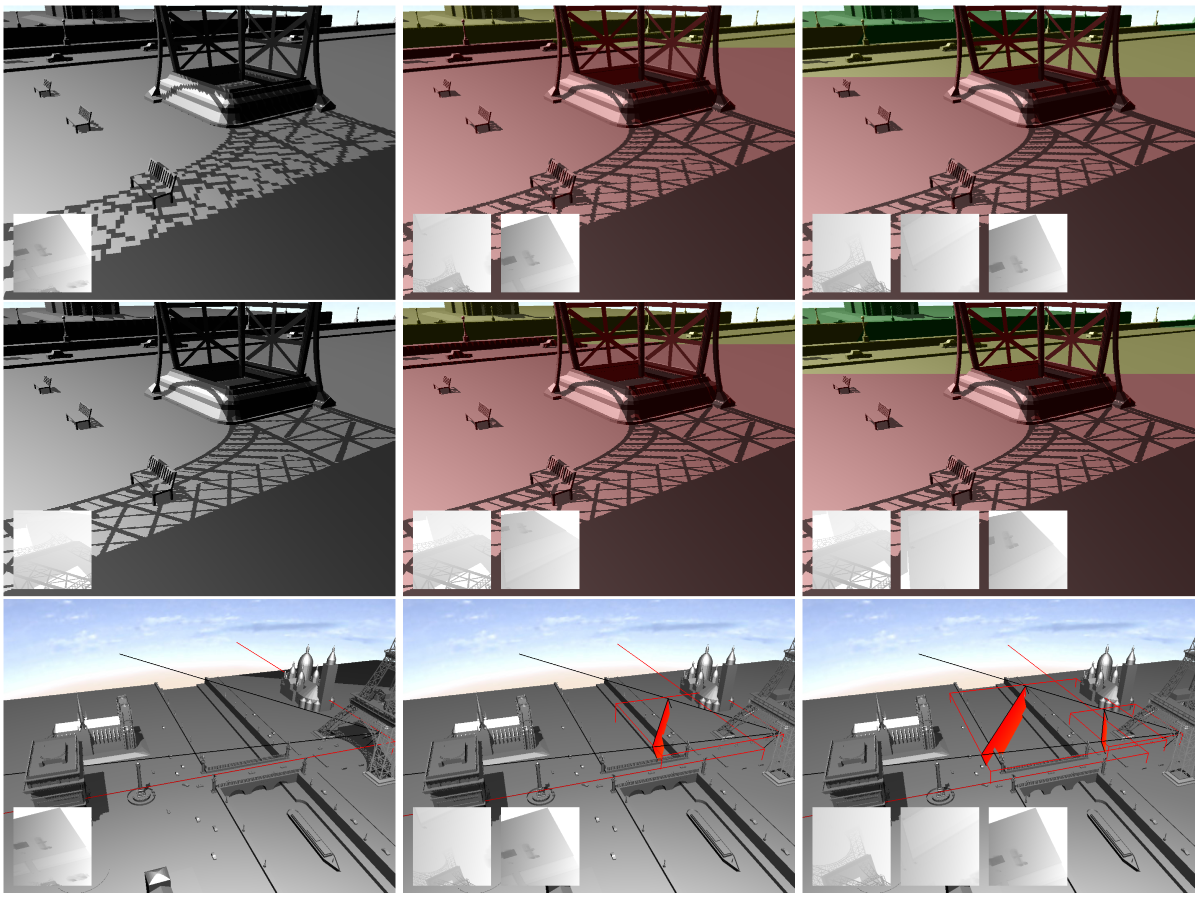


Figure 3.11 Examples of z-partitioning with and without warping for 1 to 3 partitions. The shadow map used for each fragment is color coded. Top row: Camera view, uniform. Middle row: Camera view, warping. Bottom row: outside view showing the view frustum, the partitions, and the intersection bodies. Inlays show the depth maps.

surfaces that are viewed almost edge-on by the light source (projection aliasing).

There are a number of algorithms that try to allocate samples in a more optimal way by analyzing the scene before creating the shadow map. This inevitably incurs some overhead due to the analysis step (which often necessitates a costly read-back), but leads to much better results in general cases. Prominent examples are Adaptive Shadow Maps (ASM) [Lefohn et al., 2005], Resolution Matched Shadow Maps (RSMS) [Lefohn et al., 2007], Queried Virtual Shadow Maps (QSM) [Giegl and Wimmer, 2007b], Fitted Virtual Shadow Maps (FVSM) [Giegl and Wimmer, 2007a], and Tiled Shadow Maps (TiledSM) [Arvo, 2004].

All of these approaches rely on a hierarchical data structure (usually a quadtree) to refine the shadow map. They differ mainly in the termination criteria, and the measures that are required to determine this termination criterion.

3.2.4.1 Queried Virtual Shadow Maps

Queried Virtual Shadow Maps (QVSM), introduced by Giegl and Wimmer [2007b], are maybe the simplest adaptive partitioning scheme to implement, because they do not require an expensive

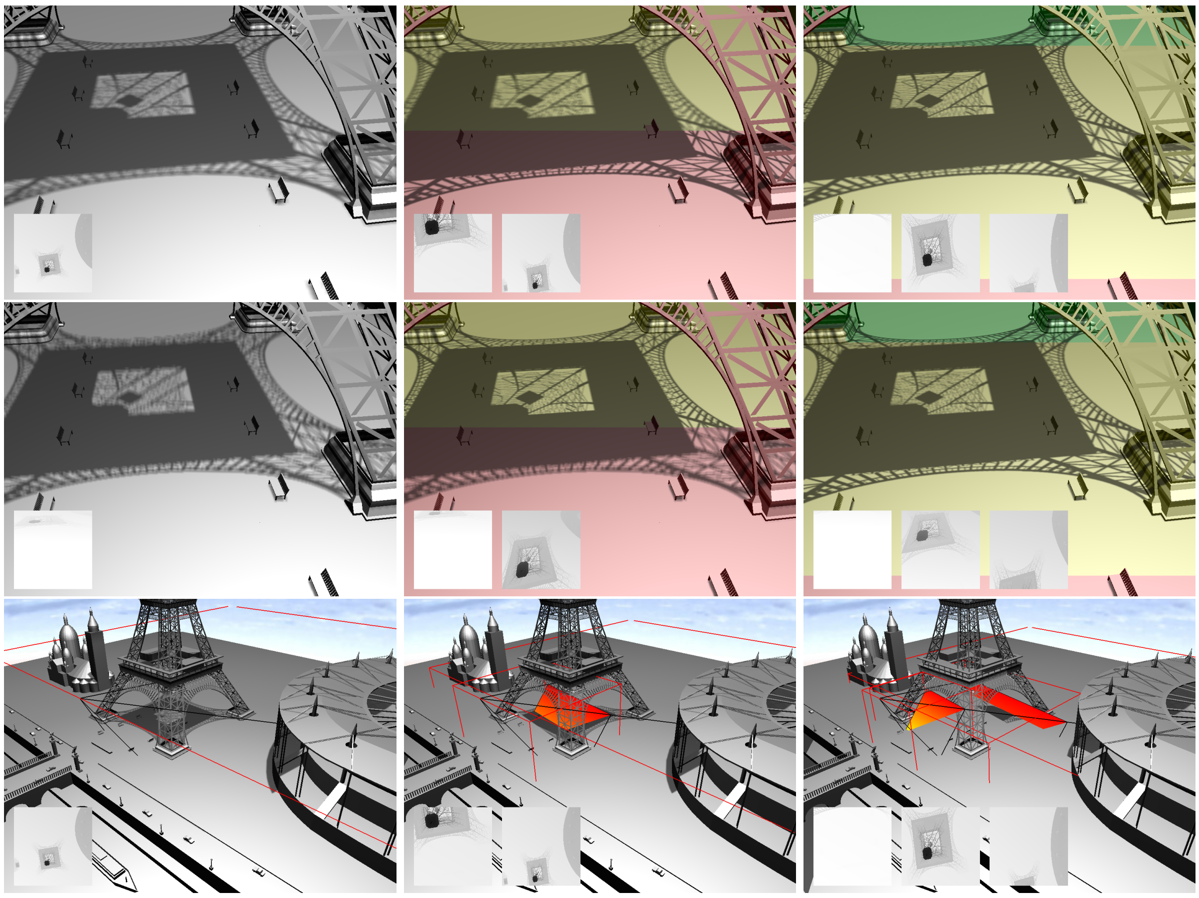


Figure 3.12 Examples of z-partitioning with and without warping for 1 to 3 partitions, here showing an example that is not amenable to warping. Using hardware PCF, the quality differences become more obvious. Top row: Camera view, uniform. Middle row: Camera view, warping. Bottom row: outside view showing the view frustum, the partitions, and the intersection bodies. Inlays show the depth maps.

readback to compute the termination criterion, and do not require implementing hierarchical data structures on the GPU. The idea is very simple: refine a shadow map hierarchy until the actual change observed in the shadow due to a refinement lies below a predefined threshold.

More exactly, starting from an initial shadow map (e.g., 2048x2048), this shadow map is split into 2x2 sub-tiles again with a resolution of 2048x2048 each. After each such refinement step, the scene is shadowed using the refined shadow maps, and the shadowing result is compared to the result of the previous step. If a certain threshold of changed pixels is exceeded in a tile, refinement continues.

The way to make this fast is to do all calculations on the GPU by using the occlusion query mechanism to count the number of pixels that differ when applying a more refined shadow map in comparison to the previous one.

QVSM require quite a high number of scene rendering passes, one for each refinement attempt. In order to avoid re-rendering the scene multiple times, the scene can be rendered into a linear depth-buffer first and each rendering pass just uses this buffer to calculate shadows (also called

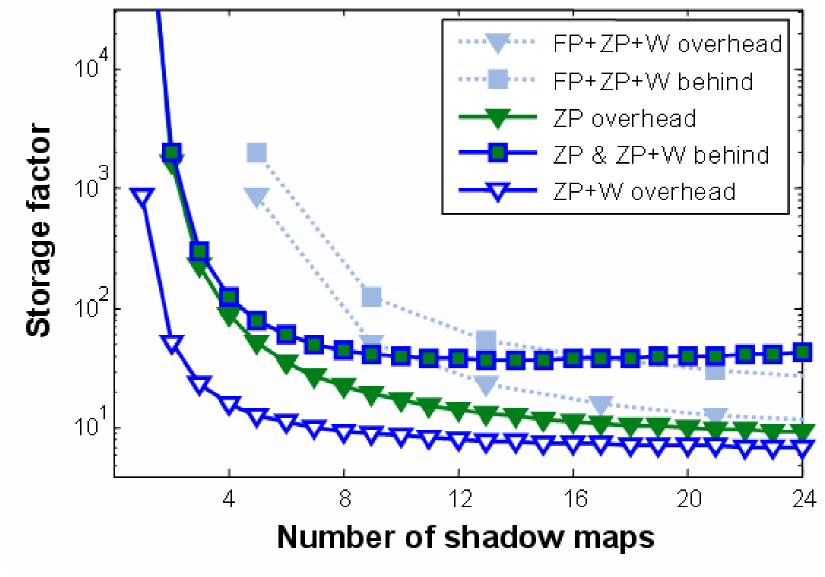


Figure 3.13 Total error of different schemes for varying shadow map numbers. FP is frustum face partitioning, ZP is z-partitioning, W is warping (figure courtesy of Brandon Lloyd).

deferred shadowing).

Figure 3.14 shows a comparison of a large standard shadow map with QVSM.

3.2.4.2 Fitted Virtual Shadow Maps

While QVSMs are simple, they suffer from a large number of shadow and scene rendering passes. In order to avoid this, Giegl and Wimmer [2007a] introduced Fitted Virtual Shadow Maps (FVSM) to try to determine beforehand what final refinement levels will be necessary in the quadtree. For this, the scene is rendered in a pre-pass, but instead of actually shadowing the scene, this pass just records the query location into the shadow map, as well as the required shadow map resolution at that query location. The resulting buffer is then transferred to the CPU. There, each sample of this buffer is then transformed into shadow map space and stored in a low-resolution buffer, utilizing the efficient scattering capabilities of the CPU. This buffer ultimately contains the required resolution in each area of the shadow map, and the quadtree structure can be derived from it. In order to avoid penalties due to readback, only a small framebuffer (e.g., 256x256) is rendered in the pre-pass.

In comparison to Adaptive Shadow Maps [Lefohn et al., 2005], both QVSM and FVSM are fast enough to evaluate the whole hierarchy for each frame anew and therefore work well for dynamic scenes, as opposed to ASM, which relies on an iterative edge-finding algorithm to determine refinement levels, and therefore needs to cache recently used tiles. Resolution Matched Shadow Maps (RMSM) [Lefohn et al., 2007] improve on ASMs especially for dynamic scenes, by avoiding the iterative step and calculating the required resolutions directly, somewhat similarly to FVSM. Both algorithms also mix data-parallel GPU algorithms [Lefohn et al., 2006] (like quadtree and sort) with standard rendering. In RMSMs, all steps are actually carried out on the GPU.

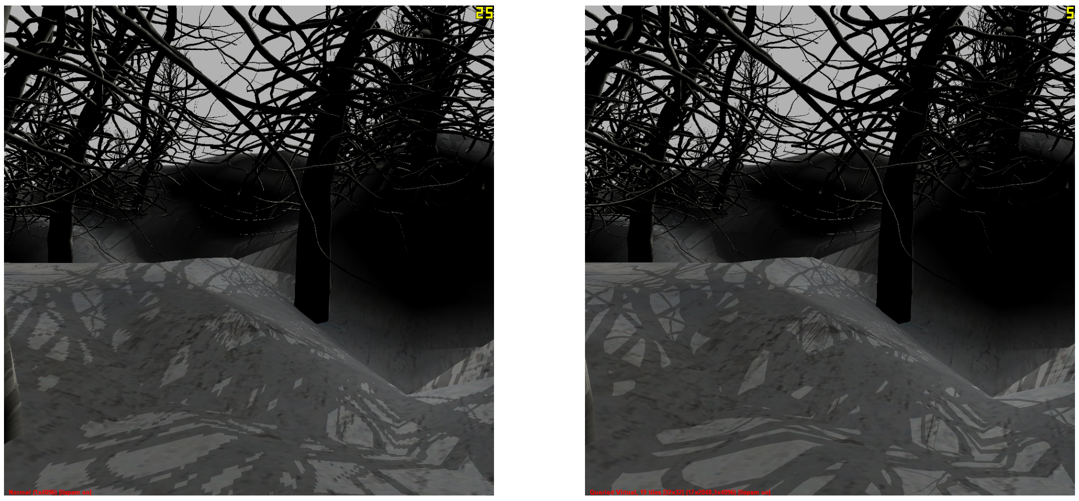


Figure 3.14 Left: standard 4096^2 shadow map. Right: QVSM with a maximum refinement level of 32x32, and 2048^2 tiles.

3.2.5 Irregular Sampling

The aliasing artifacts in hard shadow mapping stem from the fact that the shadow map query locations do not correspond to the shadow map sample locations. Ideally, one would like to create shadow map samples exactly in those positions that will be queried later on. Difficult as that may seem, it is actually possible and has been proposed independently by Aila and Laine [2004], and Johnson et al. [2005].

The algorithm proceeds by rendering an initial eye space pass to obtain the desired sample locations. These sample locations are then used as pixel locations for the subsequent shadow map generation pass. The challenge is that these new sample locations do not lie on a regular grid anymore. Therefore, view sample accurate shadow algorithms have to solve the problem of irregular rasterization. While Johnson et al. [2005] propose hardware modifications for this, Sintorn et al. [2008] manage to implement irregular rasterization on current GPUs using the NVIDIA CUDA framework. The authors report that they are only 3 times slower than standard $8{,}192^2$ shadow maps on a 512×512 viewport.

The basic idea of both algorithms is similar. After the eye-space pass, the generated view samples are projected into light space and stored in a light-space buffer. Each pixel of this buffer contains a list of view samples that fell into this pixel. Some lists will be empty, while some lists will contain several samples. These view samples are then tested for shadows by rasterizing the shadow casting triangles (typically all triangles in the scene) from the light's viewpoint. For each covered rasterized fragment, the fragment shader tests all view samples in the corresponding per-fragment lists of the light-space buffer. If a view sample is in shadow, this is indicated by flagging it. Finally, a screen-space quad is rendered in eye-space, where each fragment does a shadow query by testing its corresponding list entry for the shadow flag.

In Sintorn et al.'s implementation, the lists are realized with a constant memory footprint which is defined by the total number of view samples. The shadow status of each view sample is stored in a bit-mask. When testing a triangle, all view samples in the pixel are tested. In each pixel, a bitmask is stored such that one bit corresponds to one view sample. If in shadow, the view sample's bit is activated in the output buffer. Using the logical OR blending operation for the framebuffer, the occlusion flags set by all triangles are correctly accumulated. starting with the G80 chips, at least 128 bits can be written per output rendertarget. Using multiple render targets (currently 8 are possible) allows us to store and treat up to 1024 view samples per light pixel. The algorithm uses multipass rendering if more bits are needed. The final rendering is quick because each view sample knows its corresponding bitmask entry and simply fetches this value. The method is suited to be combined with reparametrization methods and in practice, the authors implemented a variant of [Brabec et al., 2005].

3.2.6 Temporal Reprojection

Finally, one way to increase the sampling rate is by reusing samples from previous frames through reprojection [Scherzer et al., 2007]. The main idea is to jitter the shadow map viewport differently in each frame and to combine the results over several frames, leading to a much higher effective resolution.

This method requires an additional buffer to store the shadowing result from the previous frame. In the current frame, the result of the shadow map lookup is combined with the result from the previous frame, which can be lookup up using reprojection. If a depth discontinuity between the new and the reprojected sample is detected, than the old result is not used since it is probably due to a disocclusion.

The reason why shadow quality actually converges to a pixel-perfect result using this approach is the choice of the weight between the current and the previous frame result. The weight is determined according to the *confidence* of the shadow lookup:

$$\mathrm{conf}_{x,y} = 1 - \max\left(|x - \mathrm{center}_x|, \left|y - \mathrm{center}_y\right|\right) \cdot 2,$$

where $\mathrm{conf}_{x,y}$ is the confidence for a fragment projected to (x, y) in the shadow map and $(\mathrm{center}_x, \mathrm{center}_y)$ is the corresponding shadow map texel center.

The confidence is higher if the lookup falls near the center of a shadow map texel, since only near the center of shadow map texels it is very likely that the sample actually represents the scene geometry.

Note that reprojection based approaches take a few frames to converge after quick motions. Also, they cannot deal very well with moving objects or moving light sources.

3.3 Cookbook

Finally, we want to give some practical hints which algorithms to use in what situations.

If the requirement is that only a single shadow map should be used, i.e., the algorithm should run at the same speed as standard shadow mapping, then Light Space Perspective Shadow Mapping, with the modification by Lloyd et al., is the best algorithm. This algorithm will achieve excellent quality in many configurations, especially in outdoor scenarios with roughly overhead lighting, however it can easily degrade to the quality of (focused) uniform shadow mapping. With the modification by Lloyd et al., it will never degrade below the quality of uniform shadow mapping.

Shadow Map Reconstruction

Zhang et al.'s forward shadow mapping [Zhang, 1998] splats the texels seen from the light to the view. This inverses the usual test which would look up values in the shadow map. Further, it allows us to apply some smooth degrading splat that leads to softer transitions between the samples. Currently, the practical relevance of this method is rather low, but it is an interesting idea to build upon, and in fact splatting of illumination has become a common solution for recent methods in the context of global illumination, e.g., [Lehtinen et al., 2008].

A method that reconstructs piece-wise linear shadow boundaries was presented in [Sen et al., 2003]. The method is relatively simple and it could be implemented on current graphics hardware. The idea is to use two maps: A shadow map that is shifted by half a pixel with respect to a *silhouette map*. The latter stores a point from the object's silhouette edges. To determine the shadow of a view sample it is projected in the silhouette map. Then, five samples are recovered: the center and its four neighbors. Virtual edges connecting the stored silhouette points are added between these positions. For each quadrant an evaluation of the shadow map decides on whether the region is considered shadowed or lit (this explains the half-pixel shift with respect to the shadow map). Finally the view sample is tested against these sectors. The idea to deform the grid locally based on centers stored in pixels has also been used in their follow-up work [Sen, 2004].

Major limitations are that each pixel only stores one center. In areas where two silhouettes project close to each other, noticeable artifacts can appear. Nevertheless, the quality of the method is better than for standard shadow maps, although the method remains resolution-dependent. The creation of this map involves a silhouette determination step and a rather costly rasterization that creates the silhouette map.

If more than one shadow map is allowed, i.e., some performance loss can be accepted, the best known tradeoff between efficiency and quality is achieved by z-partitioning (CSM, PSSM). The distribution of multiple shadow maps mimicks a very rough approximation of the optimal logarithmic shadow map reparametrization. Furthermore, each shadow map can adapt optimally to one part of the view frustum, thus improving the quality in each spatial dimension, independent of the orientation of the view frustum. It is possible to combine z-partitioning with reparametrization, however, temporal aliasing is increased by this approach, and the gain is not very high.

One major advantage of the afore-mentioned algorithms is that they are scene-independent, and thus do not require interaction (e.g., readback) with the scene. On the other hand, this limits these approaches to dealing with perspective aliasing only, while local aliasing effects due to different surface orientations, causing projection aliasing, cannot be improved. If higher quality is desired, then adaptive partitioning algorithms should be applied. In the future, even irregular sampling approaches, which really result in a pixel-accurate solution, might become feasible.

4 Filtered Hard Shadows

We have previously seen that image-based hard shadows are prone to aliasing. Several algorithms were suggested to overcome this problem by adapting the shadow map resolution or sampling patterns in various manners. A different approach to address the resolution-related issue is to smooth the shadow boundaries in order to make the aliasing artifacts less pronounced. At second glance, this becomes even more interesting because it results in shadow boundaries that resemble to some extent the appearance of physically-based soft shadows at a much lower computational effort. Nevertheless, none of the following methods in this section is physically accurate. In fact, there is very little physical meaning to them, but they address the aliasing problem very efficiently.

In practice, most of the techniques presented in this chapter, as well as their extensions we will discuss in Section 5.2.2 are standard solutions in game contexts and of high practical value. The simplicity of the implementation, the relatively good performance, the simple tradeoff between performance and quality, and the very reasonable outcome (at least for most configurations) makes them usually a good choice.

The interested reader is also referred to the talk by Bavoil at GDC08 [Bavoil, 2008]. This very good talk summarizes many of the practical implementation aspects and serves as a good overview for the techniques that will be presented in this chapter.

4.1 Filters and Shadow Maps

Filtering shadows sounds like a good idea to remove the jaggy appearance induced by shadow map shadows. In fact, simple upscaling techniques for low resolution images also rely on filters to remove the quad appearance induced by the pixels of the input image. Nevertheless, in the case of shadows, this step is less straightforward. In the following, we will investigate methods that employ a light-space blur over the shadow map in order to filter away the jaggy shadow edges.

4.1.1 Blurring Shadow Maps

We already mentioned that we want to filter hard shadows in light space. In other words, we will work in the space of the shadow map.

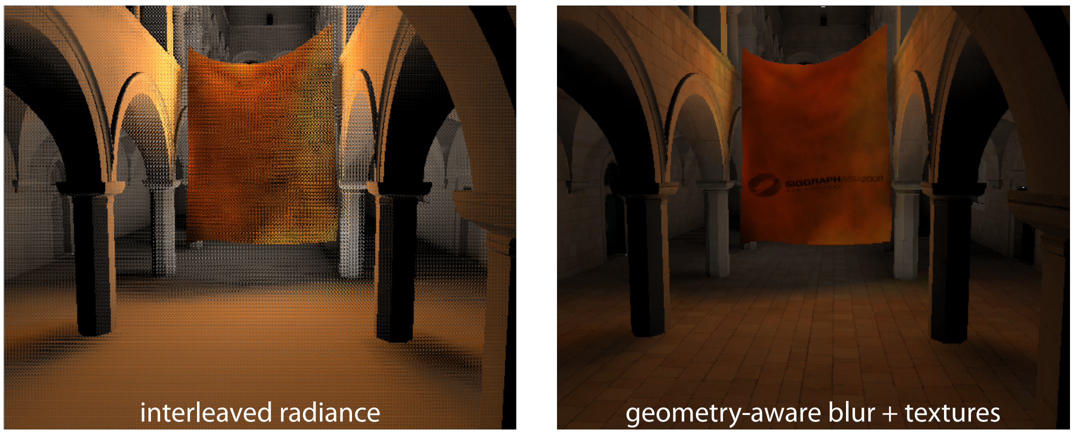
One first intuition could be to blur the depth values and then perform a standard shadow test. Unfortunately, such a blur does not make much sense. The depth values would be averaged, but the resulting shadows would still show the same aliasing artifacts because for each view sample, the shadow test still leads to a binary outcome.

To extend this idea, one might be tempted to not only replace the shadow map by a blurred version, but also the binary shadow test function with some smoother, continuous variation. This

Blurring in different spaces

Even though this chapter will focus on the standard solution of blurring in light space, there are actually several options with respect to the realm where the blur is performed.

The view frustum is one possibility, but the shadow resolution varies across the screen. Hence, it is not simple to rely on a constant-sized image-based filter. Furthermore, for larger filters, parts of the image that lie outside the view-frustum would need to be taken into account. This is one of the reasons, why view-frustum blur has been used rather little and should only be applied for small kernels or in very low frequency lighting situations (e.g., [Segovia et al., 2006; Sloan et al., 2007]). One particular application is a multi-light evaluation. A continuous radiance information is generated from an image where each pixel evaluated a different set of lights. The image below is such an example and was generated using the approach by Ritschel et al. [2008].



A surprising possibility is to blur in texture space. In [d'Eon and Luebke, 2007], one assumes that there is a one-to-one mapping between the scene and textures. This is not uncommon in a game setting, where each character can have their own texture image. If it is further possible to associate a world position to each texel, one can simply render a quad over each texture and use a fragment shader to write out the corresponding binary shadow information. A uniform filter is then applied to this texture, before mapping it back on the character. The resulting shadows have a smooth appearance for very little computational effort. In general, textures are often repeated, stretched or generally distorted which leaves such a technique unsuited for general cases.

solution has been used in [Luft and Deussen, 2006]. With some parameter tweaking those shadows can be visually pleasing for simple configurations, but can be very far from any physical reference. Furthermore, nearby superposing objects, as seen from the light, might lead to a bleeding. On the other hand, this method in [Luft and Deussen, 2006] was proposed in the context of non-photorealistic rendering in order to *abstract* shadows and give a visually smoother result. For this latter purpose, such an algorithm is well-suited, but it can hardly fool the observer into taking the shadows for real.

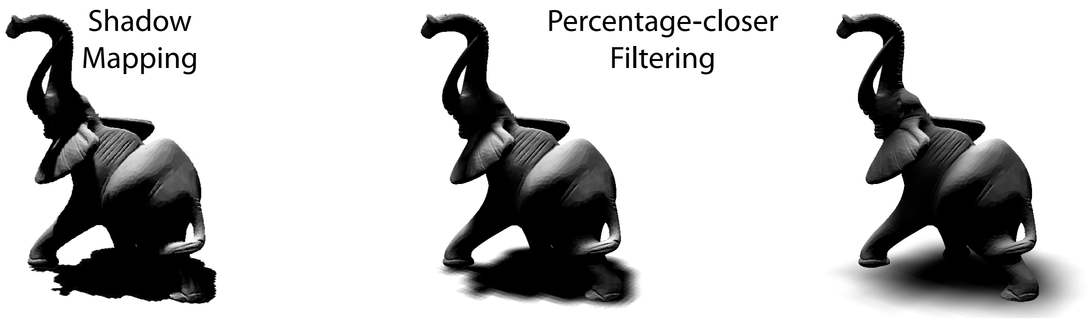


Figure 4.1 Percentage-closer filtering suffers less from shadow map aliasing (left) and results in reasonable shadows for smaller light sizes (middle). For larger sources some inconsistencies can occur (right, shadow under left foot).

4.1.2 Defining a Reference: Percentage-Closer Filtering

A more plausible outcome can be achieved using *percentage-closer filtering* (PCF) [Reeves et al., 1987]. It is a very early approach to hide the jaggy boundaries of shadow maps. Its simplicity and relatively satisfying quality makes this approach particularly interesting. Figure 4.1 shows some of the results obtained with this technique.

Standard shadow mapping compares a single depth sample to the depth of the view sample, the latter we will refer to as *reference depth*. PCF performs several such evaluations in a small window. The final shadow is the average of lit and shadowed results. A larger window leads to smoother shadows, a smaller to hard shadows. PCF has even been integrated on the hardware side to allow the application to sample 2×2 windows and to bilinearly interpolate between these neighboring shadow outcomes.

Formally, if $\mathbf{p}$ is the point to be shaded and $\mathbf{p}^s$ its projection into the shadow map, PCF computes:

$$\text{pcf}(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \cdot S(\mathbf{p}^s_z, z_{\mathbf{q}}), \tag{4.1}$$

where $\mathcal{N}(\cdot)$ is a neighborhood of shadow map texels, ω are weights from a filter kernel, S is the binary shadow test function comparing $\mathbf{p}^s_z$ (the depth of $\mathbf{p}$ in light space) to $z_{\mathbf{q}}$ (a depth in the shadow map at location $\mathbf{q}$). The entire algorithm is summarized in Figure 4.2.

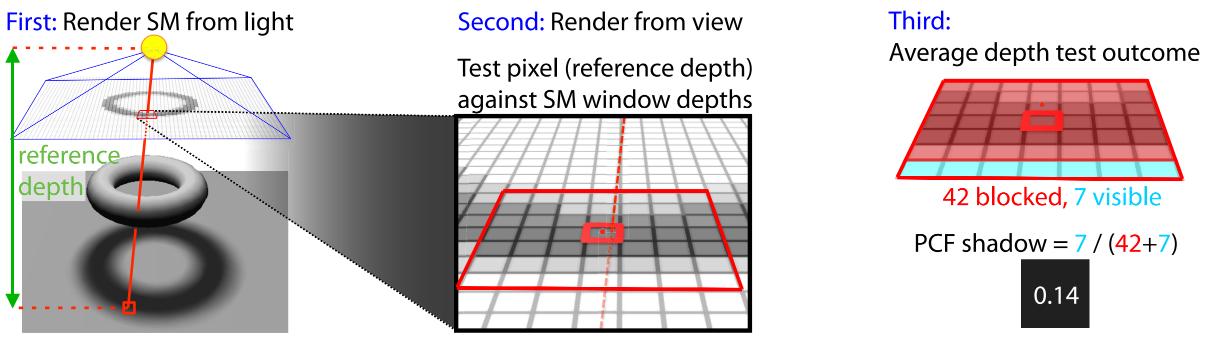


Figure 4.2 Overview of the PCF algorithm.

It has to be underlined again that the resulting shadows are not physically based. We saw in Section 1.3 that we should integrate rays over the light source. All these rays are emitted from the view sample we want to shade. For PCF, the evaluated rays do not leave from the view sample, but always from the light. This can be seen when looking at how a shadow map samples the scene. The light is located at the perspective center of the light frustum. Consequently, each pixel corresponds to a ray through the light itself. This is illustrated in Figure 4.3 (left). PCF averages shadow ray contributions together, that physically do not relate.

In practice, if the window remains of reasonable size, the shadows still look convincing because the overall look of the hard shadows is not altered too much, giving the impression of a very small, but not point, source. The eye-pleasing look that results from the blur has even been used in various movie productions, the first being the Pixar movie Luxo. From a today's point of view, one can easily see that the resulting umbra is too large and the penumbra too small, when compared to the shadows from a real large light bulb. Nonetheless, no aliasing is visible, leading to appealing and cheap shadows, which was the main goal.

Although, the cheapness of the method needs to be put into perspective. Even though at first glance, Equation 4.1 might resemble a convolution of the shadow map, it is not: The depth test needs to be performed *before* the filtering. This test implies a large performance penalty. For a small window (3×3), it might not seem very problematic, but for larger windows, the cost grows

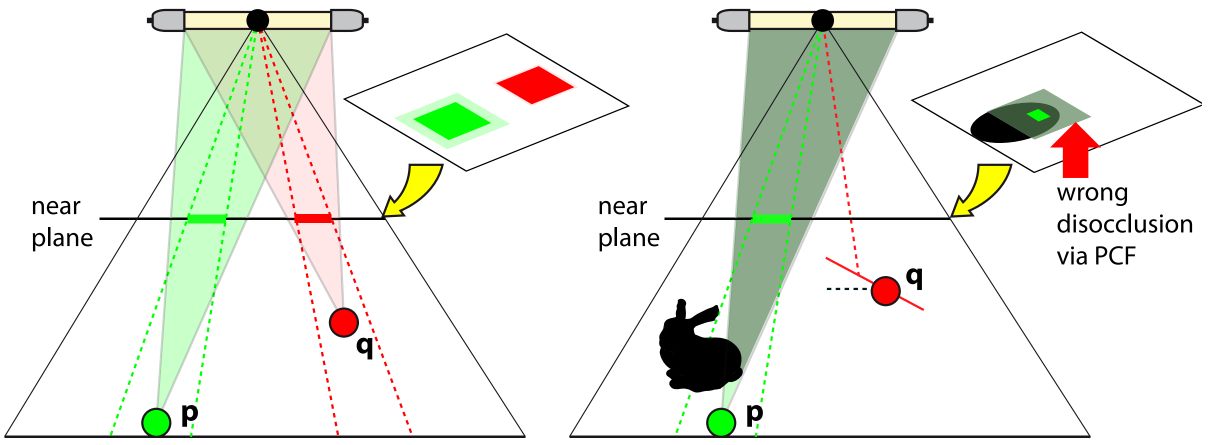


Figure 4.3 Percentage-closer filtering tests, for each point of the scene, depth samples from a constant window around the projection into the shadow map (left, dashed border). This is a coarse approximation. First, these windows should have different sizes depending on the distance to the light (left, **p**, **q**); second, the tested rays should leave from the impact point towards different locations on the light (left, transparent areas). Except for the shadow map texels that contains the point itself, none of the shadow map texels corresponds to a ray that originates at the impact point. Consequently, the shadow estimate is often approximate. Umbrae are introduced where a physically based shadow would still be partially lit, or, a point is considered to be in the penumbra, although it already lies in the umbra (right, **p**). Finally, performing the depth tests with respect to a constant reference depth can lead to self occlusions. A point **q** can be considered half in shadow even though it is fully visible to the light (right). If the surface is tilted, even though it is linear, a point can be shaded because half the surface will be considered closer than the reference depth.

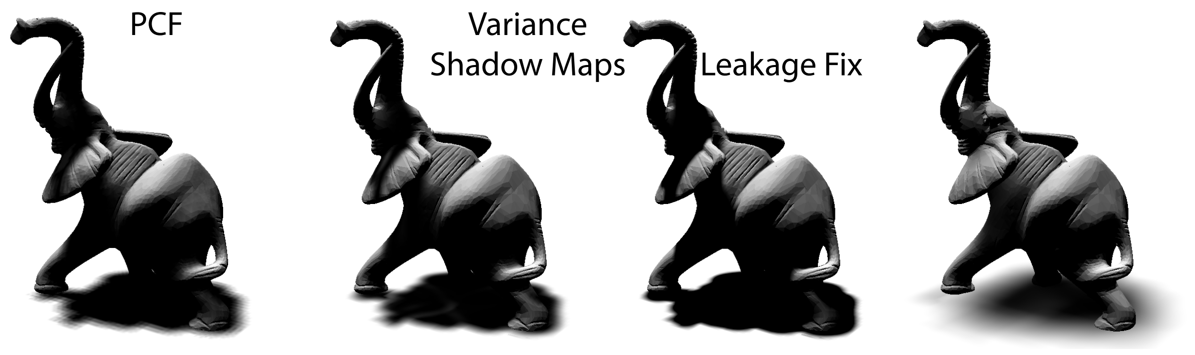


Figure 4.4 Variance shadow maps results in shadows that resemble PCF (left). The algorithms achieves much higher frame rates by approximating the computation. This may lead to slight light leaks (middle left), these can be addressed with simple solutions (middle right), but this affects the shadow smoothness. Especially for larger light sources (right, 5 times larger), the performance gain is tremendous.

rapidly and we will see in the following several ways to accelerate this computation.

4.2 Faster Than the Light (Space Tests)

An important drawback of a larger PCF window is that more samples need evaluation. This quickly becomes prohibitive.

One solution that performs well, if efficient branching is possible on the GPU and the scene is not too complex, is presented in [Uralsky, 2005]. In a first step, only a small set of samples is used. If the shadow result is not unanimous, only then are more samples evaluated. This leads to a strong speedup and samples are added mostly in the penumbra region, where they are needed. The second contribution was to choose varying sampling patterns on a per view sample basis. This avoids sampling correlations and introduces less objectionable noise [Uralsky, 2005]. Nonetheless, in order to ensure a continuous result, the first sample set should be representative of the samples in the depth window. This means again that a larger window implies more initial samples. Consequently, we are left again with our initial problem.

It would be so much easier if one could pre-blur the values for PCF. Unfortunately, this is impossible because of the comparison to the reference depth. But can such a filtering process be used to reveal some information about the depth values that would allow us to guess the outcome of the depth tests? This question motivated many of the following approaches.

4.2.1 Variance Shadow Maps

Interestingly, Donnelly and Lauritzen [2006] found a way to introduce meaningful pre-blurring operations. Their *variance shadow maps* (VSM) are a beautiful example of a simple mathematical formula that leads to a powerful algorithm. Figure 4.4 shows some results obtained with the method.

Instead of sampling each entry in the shadow map, Donnelly and Lauritzen observed that simple statistics can be used to estimate the depth sample repartition inside the PCF window.

They rely on the mean μ and variance σ of the depth samples.

Fortunately, mean and variance can be obtained very efficiently from a *linearized* depth map. Although, one might assume that depth values are typically non linear, in this chapter, we will always consider linearized depth values. In practice, this is often the case anyway because after having performed the modelviewmatrix product, the z value is linear and can be directly written into a 32 bit texture.

To find the mean μ and variance σ of the depth samples, one only needs to apply a filter of the size of the PCF window to the depth and a squared-depth map[1]. The resulting textures then hold what is called the first (M_1) and second momentum (M_2). The simple relationships $\mu = M_1$ and $\sigma = M_2 - M_1^2$ allow us to derive the mean and variance inside the PCF window and we avoid the usual high cost of evaluating all samples during the shadow computation.

Once they have this depth distribution information, they estimate the outcome of the shadow tests in the PCF window. In order to avoid any further assumptions on the distribution, they rely on the Chebyshev inequality, which provides a bound for the outcome. Precisely, given a shadow map depth value distribution with mean μ and variance σ, the probability $P(x \geq d)$ that a random depth value x drawn from this distribution is greater or equal to a given depth value d has an upper bound of

$$p(d) = \frac{\sigma^2}{\sigma^2 + (d - \mu)^2} \quad \geq \quad P(x \geq d). \tag{4.2}$$

This means that at most a percentage $p(d)$ of the shadow map region, the distribution is generated from, is not closer to the light than a receiver point $\mathbf{p}$ with light-space depth $d = \mathbf{p}_z^s$, and hence cannot occlude it. Note that the Chebyshev inequality is valid only for $d > \mu$. Hence, the value of p is set to 1 if $d \leq \mu$. This is a continuous extension of this function. The key is that this simple, rational function depends solely on the reference distance d and the mean/variance (μ, σ) of the depth distributions in a given window. This means, if both are accessible efficiently, an approximate shadow evaluation using the function p is direct.

Figure 4.5 illustrates what the computed shadow would look like for view samples placed at arbitrary locations in the scene. The gray levels indicate the blackness of the cast shadow approximation. Because the Chebyshev inequality is a bound, the correct shadow could be darker, but never brighter than this estimate. Unfortunately, if the variance is high (*one occluder on top of another*), light leaks can occur and the error might become very pronounced, making shadowed areas appear lit. This is somewhat coherent with the missing information from the depth map (*does the lower occluder have a hole where the upper is hiding it?*), but shows a clear limitation of the distribution estimate. Nonetheless, in the special case of a parallel planar receiver and planar occluder, one should point out that the bound is actually exact and becomes an equality.

The results generated with variance shadow maps are similar to percentage-closer filtering with large kernels. Performance-wise the method is ground-breaking in comparison to standard PCF and of very high practical value. The memory overhead is small. Typically, only one two channel 32 bit textures are used to represent the moments. The major disadvantage are the resulting light leaks for which we will discuss remedies in the following.

[1]Fast box-filter approximations can be applied and need significantly less lookups; e.g. [Hensley et al., 2005] (see box on page 55). Such an extension was proposed by Lauritzen [2007].

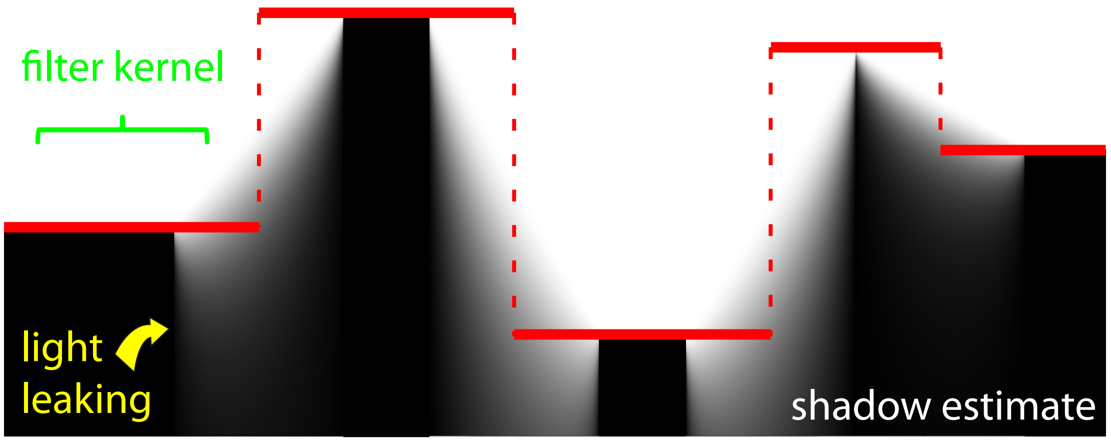


Figure 4.5 Variance shadow maps estimate the depth sample repartition inside the filter kernel window solely based on the variance and mean. Such an approximation leads to light bleeding. The light is coming from above, the shadow casters in the depth map are indicated in red, and the gray levels illustrate what the shadow value would be if a receiver were located at the corresponding position in space.

Fast-box Filter Approximations

In many shadow applications, it is necessary to find an average value of a region in a texture (e.g., [Soler and Sillion, 1998], [Donnelly and Lauritzen, 2006], [Eisemann and Décoret, 2006b]). Integrating these values inside a box region implies many texture lookups if this is implemented in a naive way. The most common solution to approximate this average value in a rectangular window are mipmaps, but other solutions exist.

To create a *Mipmap* pyramid [Williams, 1983] from a texture, in each level, four texels are grouped together to produce a texture of half the resolution. This is done by storing the average value of these four in the corresponding texel of the new level. This gives the box filter solution for kernels of size 2^n at each 2^n-th texel. Using linear blending approximate solutions for arbitrary kernels are possible. Mipmaps are hardware accelerated and very efficient to produce.

N-buffers [Décoret, 2005] are a similar structure, but instead of reducing the resolution, a pixel in an N-buffer of level l holds the maximum of all pixels in a window of size 2^l. Eisemann and Décoret [2006b] pointed out showed that it is also possible to store the average of this region. N-buffers are not accurate, but because they avoid the resolution reduction, the result is of higher quality. The construction can be done hierarchically, leading to an efficient algorithm.

Summed-Area Tables [Crow, 1984] are a one texture representation, that allows us to accurately determine the value in an arbitrary rectangular region by using only four lookups in this texture. In each texel $t(i, j)$, we store the sum $\mathrm{SA}(i, j) = \sum_{k \le i \wedge l \le j} t(k, l)$. Consequently, the average value A of a window defined by texels $t(i, j)$, $t(k, j)$, $t(k, l)$, and $t(i, l)$ with $i < k$ and $j < l$ is given by $\mathrm{SA}(i, j) - \mathrm{SA}(k, j) + \mathrm{SA}(k, l) - \mathrm{SA}(i, l)$. The construction is very efficient [Hensley et al., 2005], but a high precision is needed to avoid artifacts. For a binary input texture, the summed-area table should be stored in an integer texture which makes it simple to obtain robust results.

Depth Bias

We have mentioned in Section 2.1.2 that standard shadow mapping requires a careful depth bias adjustment. Even though we will not detail this problem for PCF shadows, it has to be pointed out that this problem can actually become even more pronounced. Imagine a point **q** on a tilted plane; using its depth to compare against the shadow map makes other points on this plane occlude **q** (Figure 4.3, right), which is physically impossible.

Brabec and Seidel [2002] propose to further use an object index map [Hourcade and Nicolas, 1985] to avoid taking depth samples from the same object, but a per-object index disables self-shadowing and a per-triangle index can result in similar problems as a depth test.

A different proposition is to estimate the derivative at the current view sample in order to define a reference plane for the shadow testing step [Schüler, 2006]. On today's hardware the plane can be easily estimated by applying the partial derivative operators `ddx`, `ddy`. Of course, planar approximations fail in areas of high curvature, but in practice it is often sufficient.

A similar approximation was used elegantly in the context of variance shadow maps [Lauritzen, 2007] when rendering into the VSM. Instead of considering the depth distribution to be constant over the entire shadow map texel, a local linear approximation is computed in form of a plane equation. Inside a texel, the surface is thus represented as:

$$f(u, v) := z + u\,\mathtt{ddx}(z) + v\,\mathtt{ddy}(z),$$

where z is the depth value encountered during rendering into this VSM texel, and `ddx` and `ddy` are derivative operators applied to this quantity. The first moment is given by $M_1 = z$; the second moment can be written as:

$$M_2 = z^2 + E(u^2)\,\mathtt{ddx}(z)^2 + E(v^2)\,\mathtt{ddy}(z)^2$$

because $E(u) = E(v) = E(uv) = 0$. Representing the pixel as a Gaussian distribution with half a pixel variance implies $E(u^2) = E(v^2) = \frac{1}{4}$ and consequently:

$$M_2 = z^2 + \tfrac{1}{4}(\mathtt{ddx}(z)^2 + \mathtt{ddy}(z)^2).$$

4.2.2 Layered Variance Shadow Maps

Initially, Lauritzen [2007] presented very simple solutions to deal with light leaks. One possibility is to map all values smaller than some threshold ϵ directly to black, hence removing subtle leaks. In order to maintain continuous shadow boundary variations, all shadow values bigger than ϵ need to be remapped to the range $[0, 1]$, e.g. using a `smoothstep`-function. Even though light leaking is reduced, such a tampering also makes shadows look darker and shrinks the smooth boundary regions. Another possibility is to clamp the variance to some maximum value, but this also affects the quality of the penumbra regions.

Layered Variance Shadow Maps (LVSM) [Lauritzen and McCool, 2008] aim at solving the light-leaking issue in a more elegant fashion. The observation is that strongly differing depth samples result in a large variance value, which in turn makes Equation 4.2 very unreliable. It would be much better, if the depth samples were close together. Unfortunately, these depth values depend on the scene and we cannot choose them the way we want.

The property that allows us to shift depth values together is that the outcome of a PCF does not depend on the actual depths, but rather on the comparison with the reference depth d. The exact same result is obtained if all depth samples closer to the light than d are moved to the depth $d - \epsilon$. The actual distance from which the depth samples are away from a receiver depth $d = \mathbf{p}_z^s$ is not interesting, only whether they are above or below. This insight led to the idea of a warping function that will adjust the depth values in order to make the approximation via Equation 4.2 more accurate.

Imagine all samples were warped by

$$\phi_d(x) = \begin{cases} 1, & \text{if } x \geq d \\ 0, & \text{if } x < d \end{cases}$$

then the result for a reference depth d using VSMs would be equivalent to the accurate result of the PCF. It would be costly to perform this warping per view sample. Basically, we would again have to go over all samples in the PCF window. Hence, instead the scene is sliced into depth intervals $\{[d_i, d_{i+1}]\}_{i=0}^n$. Inside each such layer, the original depth values are warped linearly, whereas outside they are clamped:

$$\phi_i(x) = \begin{cases} 1, & \text{if } x \geq d_{i+1}, \\ (x - d_i)/(d_{i+1} - d_i) & \text{if } x \in [d_i, d_{i+1}), \\ 0, & \text{if } x < d_i. \end{cases}$$

This leads to smaller variance estimates and, thus, better shadow behavior. In particular, the linear warping ensures that the quick momentum computation remains valid.

One could uniformly space the layer depth bounds d_i, but the authors also propose an algorithm to estimate the light bleeding based on the current depth map. The interested reader is referred to [Lauritzen and McCool, 2008] for more details.

The method is very efficient because, for evaluation, each sample only needs to test the layer into which it is projecting. The algorithm leads to much better results than standard VSMs, but to avoid light bleeding completely, many layers are needed and, hence, many filtering operations.

Further, many layers increase the memory usage, although it is possible to reduce the texture's precision with respect to VSM: 16 bit are sufficient, but often even 8 bit lead to an artifact-free solution. Besides memory advantages, a lower bit depth also allows us to fuse textures together into separate channels. This helps to reduce the number of filter passes.

4.2.3 Convolution Shadow Maps

Convolution shadow maps (CSM) [Annen et al., 2007] follow the spirit of variance shadow maps, their goal being a way to allow pre-filtering to approximate PCF. The main difference is that instead of basing the result on a statistical estimate of the depth distribution, Annen et al. develop the PCF evaluation in form of a Fourier expansion. In theory, this expansion has unlimited coefficients, but a truncation leads to an approximation of the final result. Because this result is just expressed in a different basis, a major difference with respect to variance shadow maps is that it converges towards the PCF solution when more Fourier coefficients are added. Even for for 16 coefficients this can lead to reasonable results as illustrated in Figure 4.6.

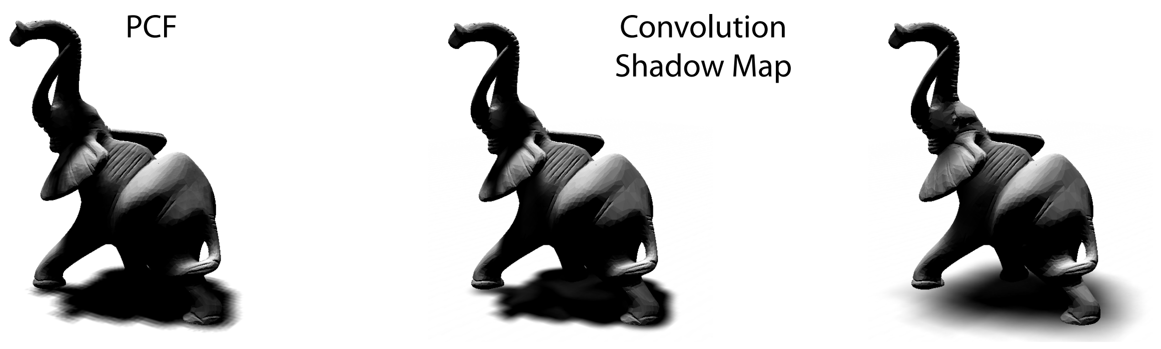


Figure 4.6 Convolution shadow maps lead to reasonable results (middle). The main problem are light leaks due to insufficient coefficients, such as under the foot. Nevertheless, the strong speed-up makes it an interesting alternative, especially for large sources (right).

Remember the PCF evaluation from Equation 4.1:

$$\mathrm{pcf}(\mathbf{p}) = \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \cdot S(\mathbf{p}^s_z, z_{\mathbf{q}}), \tag{4.3}$$

where $\mathbf{p}$ is the point to be shaded, $\mathbf{p}^s$ its shadow map projection, $\mathcal{N}(\cdot)$ is a neighborhood of pixels, the window in the shadow map in which we would perform the depth tests using PCF, ω are weights from a filter kernel, S is the binary shadow test function comparing $\mathbf{p}^s_z$ (the depth of $\mathbf{p}$ in light space, linearized to $[0, 1]$) to $z_{\mathbf{q}}$ (a depth in the shadow map at location $\mathbf{q}$). This equation was not a convolution of the shadow map because the depth test needs to be performed *before* filtering. Instead, it is a convolution of the outcome of the shadow test.

To transform the above equation into a convolution where no shadow testing is required prior to filtering, S is replaced by a combination of basis functions B_i:

$$S(\mathbf{p}^s_z, z) = \sum_{i=1}^{\infty} a_i(\mathbf{p}^s_z)\, B_i(z) \tag{4.4}$$

These basis functions *linearize* the depth test and separate the dependencies of the variables. Using Equation 4.4 one can conclude:

$$\mathrm{pcf}(\mathbf{p}) = \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \cdot S(\mathbf{p}^s_z, z_{\mathbf{q}}) = \sum_{i=1}^{\infty} a_i(\mathbf{p}^s_z) \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q})\, B_i(z_{\mathbf{q}}). \tag{4.5}$$

The resulting formulation is a weighted sum of convolutions. By looking at the terms, we see that it amounts to applying the basis function B_i to the depth map and blurring the result values with the ω kernel.

The big remaining question is how to choose B_i and a_i. The authors tested several possibilities and conclude that a solution via Fourier analysis is the best choice. $S(\mathbf{p}^s_z, z_{\mathbf{q}})$ can be seen as a step function $\mathrm{step}(z - \mathbf{p}^s_z)$ where

$$\mathrm{step}(x) = \begin{cases} 1, & x > 0, \\ 0, & x \le 0. \end{cases}$$

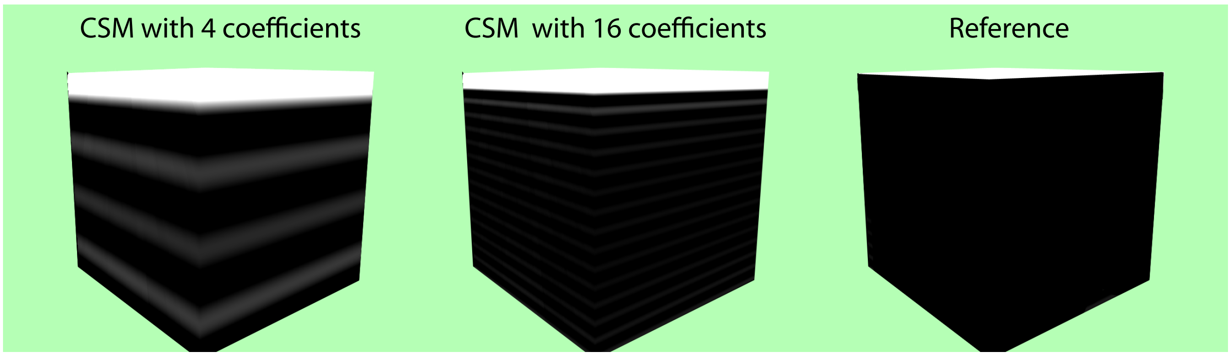


Figure 4.7 Convolution shadow maps can show light leaks due to an insufficient number of coefficients. The image illustrates the shadow result on a cube illuminated from the top scaled by a factor of four. Where four coefficients show strong ringing and a significant light leaking at the top of the cube (left), 16 coefficients result in a much better estimate (middle). Nevertheless, the difference to the reference (right) remains visible.

This function can in turn be approximated by a smooth function, e.g. $0.5\,\mathrm{sigm}(z-\mathbf{p}_z^s)+0.5$, where sigm is a sigmoid (s-shaped) function. A Fourier expansion leads to an equation of form (4.4). More precisely, they exploit the fact that the function is a real function. This allows a decomposition into cosine and sine terms. The final equation truncated to M terms is:

$$\mathrm{step}(z-\mathbf{p}_z^s) \approx \widehat{\mathrm{step}}(z-\mathbf{p}_z^s) \approx \frac{1}{2} + 2\sum_{k=1}^{M} \frac{1}{c_k}\cos(c_k\mathbf{p}_z^s)\sin(c_k z) - 2\sum_{k=1}^{M}\frac{1}{c_k}\sin(c_k\mathbf{p}_z^s)\cos(c_k z) \quad (4.6)$$

The precise coefficients c_k can be found in Section 3.1 of [Annen et al., 2007]. In practice, the sum is usually truncated at 16 coefficients stored using 8 bit values. Nevertheless, it should be realized, that $\sin(c_k z)$ and $\cos(c_k z)$ need to be stored separately, hence 16 coefficients result in 32 values to store.

Any Fourier truncation can lead to ringing artifacts which reflect the sinusoidal nature of the representation (Gibbs phenomenon). Figure 4.7 shows this effect on a real scene. The ringing artifacts propagate light into the shadow.

To hide the ringing artifact, the coefficients of higher frequencies are attenuated, but there is no guarantee that the ringing disappears. The red curve in Figure 4.8 shows the result of these damped coefficients for $M = 8$. There is still some significant light leaking, and shadow leaking. In particular, the approximation $\widehat{\mathrm{step}}$ always results in a C^∞ function, whereas a step function is not even C^0. This has important consequences. On the exact shadow boundary ($\widehat{\mathrm{step}}(0)$), we will encounter a value of 0.5 due to symmetry, leading to a reduced intensity for actually visible surfaces and more generally, to shadow and light leaking.

The authors decided that light leaking is less objectionable. Their solution is to virtually shift the function by applying an offset o: $\widehat{\mathrm{step}}_o(z) = \widehat{\mathrm{step}}(z - o)$ (Figure 4.8, left). A second improvement results from scaling $\widehat{\mathrm{step}}$ and clamping the shadow result (Figure 4.8, right). This gives a steeper approximation that sharpens shadows, and can potentially reintroduce aliasing. Using a scaling by 2 ensures that view samples no longer shade themselves.

Usually, the method needs to rely on at least four 8-bit RGBA textures (16 coefficients) to achieve a good quality. For this setting, the method is reasonably fast (around 60 Hz on a G80

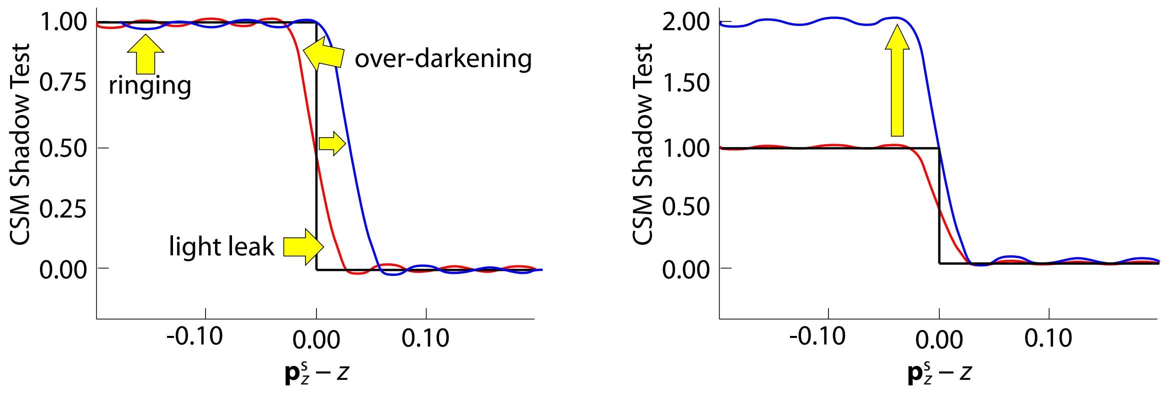


Figure 4.8 Damping the coefficients is not sufficient to avoid artifacts. The example shows the result of an approximation with 8 coefficients. To improve the solution, one can shift (left) or scale (right) the approximated step function.

GTX) for complex scenes, and gives good antialiasing because mipmapping or even anistropic filtering are meaningful when applied to the basis functions.

4.2.4 Exponential Shadow Maps

In concurrent work, Annen et al. [2008b] and Salvi [2008] (who developed a very similar solution that was presented slightly earlier at GDC 2008) proposed new basis functions for the convolution shadow maps approach. They suggested to replace the Fourier expansion by a simple exponential. This choice voids much of the storage requirements and thus addresses one of the major issues.

The main insight is that the depth map stores the nearest surface, thus any view sample or point in the scene should project on or behind, but never in front of any stored depth sample. With this assumption, $z - \mathbf{p}_z^s \leq 0$. Hence, Equation 4.7 (below) behaves almost like a step function which becomes steeper for an increasing constant c:

$$S(\mathbf{p}_z^s, z_\mathbf{q}) = \exp(-c(z_\mathbf{q} - \mathbf{p}_z^s)), \tag{4.7}$$

In practice, $c = 80$ seems to be a good choice. Larger values can lead to an overflow due to numerical inaccuracies. Applying the filtering operation to this function leads to:

$$\begin{aligned}\mathrm{pcf}(\mathbf{p}) &= \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \cdot S(\mathbf{p}_z^s, z_\mathbf{q}) \\ &= \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \cdot \exp(-c(z_\mathbf{q} - \mathbf{p}_z^s)) \\ &= \exp(c\mathbf{p}_z^s) \sum_{\mathbf{q}\in\mathcal{N}(\mathbf{p}^s)} \omega(\mathbf{p}^s - \mathbf{q}) \exp(-cz_\mathbf{q}).\end{aligned}$$

Thus the two terms are again separated, but, a single 32-bit information is sufficient this time.

Unfortunately, for a $\mathbf{q}$ coming out of a *neighborhood* of $\mathbf{p}^s$, the assumption $z_\mathbf{p} - \mathbf{p}_z^s \leq 0$ does not necessarily hold. As a consequence, large positive values can occur. This is a problem because it invalidates the summation: the exponential no longer behaves like a step function. Even though

Link to Image Processing

Convolution shadow maps show a resemblance to [Paris and Durand, 2006] focusing on image processing with the bilateral filter [Tomasi and Manduchi, 1998; Smith and Brady, 1995] that found many applications in computational photography. It is actually interesting to see that PCF can be linked to these filter accelerations and it gives a new way of looking at the problem.

The goal of [Paris and Durand, 2006] is to provide an approximation for the bilateral filter

$$\text{bilateral}(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in\mathcal{N}(\mathbf{x})} I(\mathbf{y})\, g(\mathbf{y}-\mathbf{x})\, h(I(\mathbf{y})-I(\mathbf{x}))}{\sum_{\mathbf{y}\in\mathcal{N}(\mathbf{x})} g(\mathbf{y}-\mathbf{x})\, h(I(\mathbf{y})-I(\mathbf{x}))},$$

where $I(\mathbf{y})$ describes the value of the image at position $\mathbf{y}$. g, h are usually Gaussian kernels. The filter thus combines pixels that are not only close in distance (g) but, further, have similar color (h). Paris and Durand compute the nominator and denominator separately. Let's first look at the denominator. It bears some similarity to [Annen et al., 2007]. We see that $g(\mathbf{y}-\mathbf{x})$ takes the place of ω. But if h is chosen to be a step function, then it is similar to $S(\mathbf{p}_z^{\mathrm{s}}, z_{\mathbf{q}}) = \mathrm{step}(z - \mathbf{p}_z^{\mathrm{s}})$. We thus obtain the same equation.

Paris and Durand then use a similar key insight to replace h by an approximation via basis functions:

$$h(I(\mathbf{y}) - I(\mathbf{x})) = \sum_i \delta(I(\mathbf{x}) - I_i) h(I_i - I(\mathbf{x})),$$

where δ is measuring the similarity between $I(\mathbf{x})$ and the fixed samples I_i. This equation is already of the same form as Equation 4.4. The computation is thus separable and both approaches perform similar computations. In a shadow context, this weight is already the final output. It describes how many pixels participate in the computation (let light pass). However, the derivation of the basis functions does differ. Paris and Durand rely on a linearization similar to previous work by Durand and Dorsey [Durand and Dorsey, 2002] which *slices* the function with respect to a set of reference values. It could be interesting to see whether one approach could benefit from the other's solution.

for many pixels this assumption does hold, the larger the kernel size, the less likely it is that artifacts can be avoided. This makes exponential shadow maps less usable for large kernels and it mostly finds application to smooth the boundary of hard shadows without achieving penumbra-like effects.

In order to address this robustness issue, Annen et al. propose to detect those pixels were the assumption might fail. For these, a standard PCF filtering is applied. This PCF filtering can still be performed with the exponential map by simply clamping each single value before the summation.

Two possibilities are presented to classify erroneous pixels. A first strategy tests if the filtered value exceeds one. This solution is very approximate, but fast. A better classification can be performed by creating a lookup texture that gives the maximum z-value in each filtering window. This could be done using a min/max mipmap approach. This is conservative, but costly in practice.

Although (using the approximate classification) this approach leads to an approximate speedup of 2–3 over [Annen et al., 2007], the performance depends highly on the scene and the kernel size. E.g., whenever silhouettes from the light are visible in the camera view, these will need to be evaluated with PCF. Grass on the ground, unstructured surfaces, or other fine geometry can result in many pixels being treated with PCF up to the point that the gained speed-up can completely vanishes. Memory costs, on the other hand, are always improved by an important factor of ≈ 8. For simpler models with less details, this method is a very good choice.

Exponential Layered Variance Shadow Maps Without Layers

Surprisingly, the concurrently published layered variance shadow map paper by Lauritzen and McCool [2008] proposes an alternative warping based on exponential functions and without the need for layers.

They suggest to use the variance shadow map approach (see Section 4.2.1) and apply it to depth maps that were warped by an exponential function. They propose to use two such warped depth maps, using the functions $-\exp(cz_{\mathbf{q}})$ and $\exp(cz_{\mathbf{q}})$. Both functions map the depth values monotonically, hence, allowing them to use the standard variance shadow map framework and the mentioned Chebyshev inequality related to Equation 4.2 gives valid bounds in both cases. Consequently, for both functions corresponding bounds p_-, p_+ are computed and the shadow is finally defined by $p := \min\{p_-, p_+\}$.

In practice, this method has little memory overhead, is the easiest to implement and performs very well. For the case of anti-aliased shadows, this seems currently a very good choice.

4.3 Summary

In this chapter we investigated the use of filtering hard shadows in order to create smooth shadow boundaries. Although not physically accurate, the methods can provide smooth transitions and visually pleasing results. In practice, these methods and their extensions to soft shadows that we will visit in Section 5.2.2 are very relevant for real-time applications.

One advantage we did not point out sufficiently is the fact that filtering approaches like convolution, variance or exponential shadow maps all support multisampling and can address aliasing due to foreshortening. When looking up values from the filtered textures, one can use anisotropy, supported natively as a texture filtering mode, to reflect the pixels projected footprint. This allows us to fight another source of aliasing.

The drawback is that the light source cannot be too large, otherwise the drawbacks of the constant filtering size become very obvious and contact shadows can suffer significantly. The

PCF Extension

Related, but more general than PCF, are multiple-depth shadow maps [Pagot et al., 2004], despite little practical relevance, the idea is intriguing. Instead of using all samples in a neighborhood of size l, a best selection of k elements is chosen (e.g., smallest/largest depth values). These samples are then evaluated with PCF. If k is small compared to l, then there is a potential gain. Unfortunately, the selection of the k elements is usually costly and the paper further limits the choice to $k = 2, 3$. Nevertheless, such a selection process is an idea to keep in mind for the future.

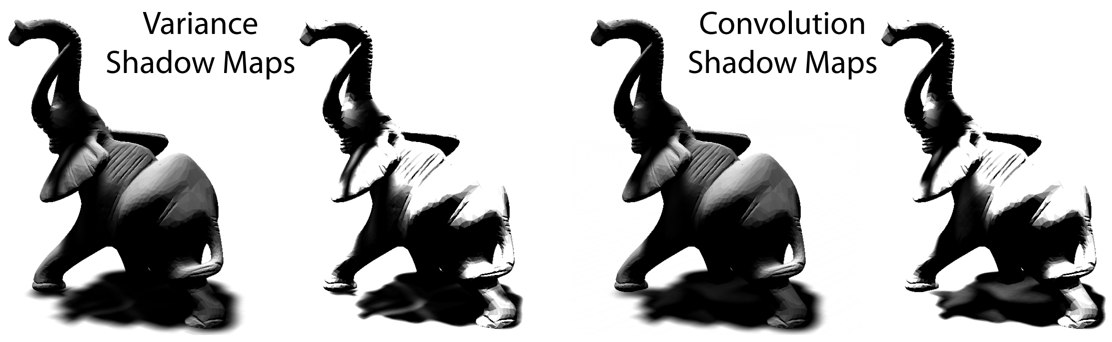


Figure 4.9 Although some light bleeding is present in approximate solutions (VSM: left , CSM: right, each original and scaled with a factor four), newer approaches show significant less artifacts and are a good choice when high performance is needed.

methods are really intended to fight aliasing, not to simulate soft shadows. The fact that the aliasing removal can result in a pleasing and soft-shadow-like appearance is a welcome side effect.

5 Soft Shadows

5.1 Introduction

While we concentrated on illumination by point light sources and the resulting hard shadows in the previous chapters, this chapter as well as the next one focus on soft shadows cast by extended light sources as encountered in the real world. Such extended light sources come in several flavors, ranging from distant environmental lights to area lights. In this chapter, we consider only nearby lights of simple shape, like rectangular or spherical light sources, not least because it is often such lights which are responsible for producing visually dominating and distinctly recognizable soft shadows. Environmental light sources like the skylight are subsequently treated briefly in Chapter 6.

Recall that, in contrast to hard shadows, soft shadows feature transition regions of partial illumination, the penumbrae, and may not necessarily comprise an umbra. Referring to the generic setting in Figure 5.1, straightforward geometric considerations show that as the size of a light source grows, a shadow's penumbra region increases, while its umbra shrinks and eventually even disappears. The part of the penumbra growing outwards with respect to the umbra at point light size is sometimes called *outer penumbra*, while the portion extending inwards and replacing the umbra is referred to as *inner penumbra*. Furthermore, keeping the light fixed but moving the occluder towards the receiver reduces the overall shadow size and decreases the relative penumbra portion. A direct consequence is that if an occluder touches a receiver, the cast shadow essentially only comprises an umbra at the region of contact but with increasing distance becomes

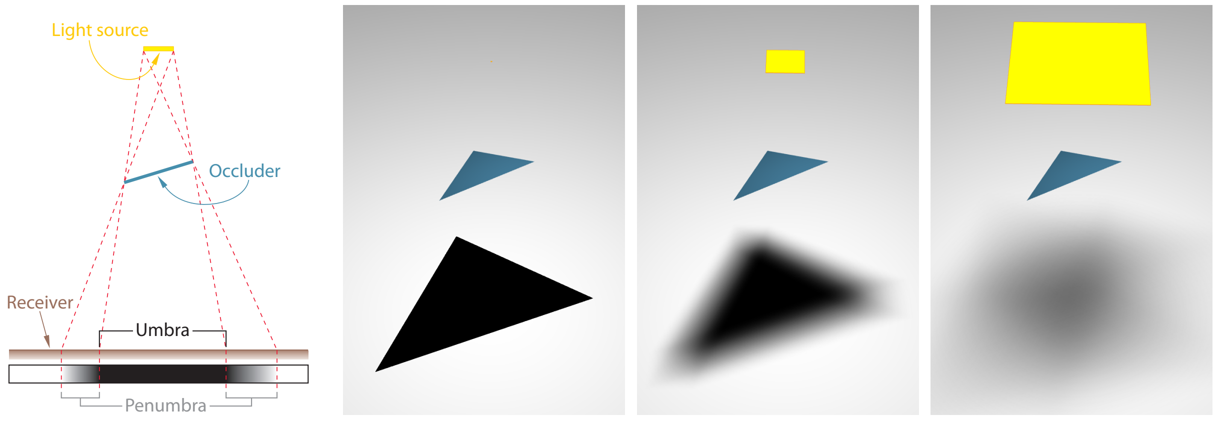


Figure 5.1 Simple setup illustrating the geometry of the soft shadow cast by a single occluder onto a planar receiver. Note that when enlarging the light size, the shadow becomes increasingly softer, ultimately losing its umbra.

wider and dominated by a growing penumbra which ultimately may supersede the umbra completely. Such shadow *hardening on contact* is an example of the important cues [Mamassian et al., 1998] provided by soft shadows and their significance for a realistic appearance. They convey information about the shape of objects, like for instance the size of lights, the silhouette of shadow-casting occluders, or the surface geometry of receivers.

As mentioned in Section 1.3, computing soft shadows is pretty challenging, especially if real-time performance is desired. Related algorithms hence typically focus on computing the visibility integral from Equation 1.4, i.e. they determine the fraction of the light source that is visible from a shadow-receiving point, and simply modulate the unshadowed shading accordingly.

5.1.1 Aspects of Practical Soft Shadow Computations

Generally, such soft shadow computations feature multiple aspects, for each of which various options with different consequences on generality, performance and accuracy exist. In the following, we briefly detail the most important ones, thus both highlighting an essential part of the algorithm design space and offering a classification scheme.

Providing Occluder Information Since visibility is a non-local problem, information about the scene geometry must be made available to the shaders computing the soft shadows. There are two general approaches to achieve this. On the one hand, an *explicit* scene representation may be constructed and provided to all fragments. In many cases, this is just a shadow map offering a point sampling of some of the occluders, but in principle one may also utilize a spatial structure like a kd-tree filled with the scene triangles or a subset of them.

Alternatively, scene information can be distributed *implicitly* by subsequently issuing primitives describing the scene geometry such that each causes the visibility data of the affected pixels to be updated accordingly. Examples include wedges corresponding to silhouette edges and polygonal footprints of scene triangles. Typically, each such primitive conservatively covers the influenced pixels, and for each generated fragment the stencil buffer is adjusted or the output of the triggered pixel shader gets blended into some buffer representing the current light visibility.

In finding a suitable way to convey global scene information, issues like generation time and required storage space but also ease of occluder processing have to be dealt with. Moreover, employing acceleration structures may be worthwhile if the gained savings exceed the involved building costs.

Visibility Computation Order Closely related to how occluder information is provided, explicitly or implicitly, two basic ways of processing this occluder information may be distinguished. Either each receiver sample *gathers* the blocking contribution of all relevant occluders by looping over them, or for each occluder the resulting occlusion is *scattered* to all receiver samples whose light visibility is affected by the occluder. Gathering requires an explicit scene representation and allows keeping intermediate occlusion information in shader registers. By contrast, scattering involves an implicit scene representation and necessitates storing intermediate occlusion information for all receiver samples in some buffer, which is successively updated. Moreover, to realize scattering via rasterization, an appropriate computational space must be chosen (see below).

Occluder Representation Another central aspect is the way occluder information is represented. One major option is utilizing an *image-based* representation of the considered occluding scene objects. By contrast, *geometry-based* methods employ an explicit geometric primitive for each processed occluder to derive the soft shadows. Due to the fundamental differences between these two approaches, we adopted this representation choice as our major line of distinction for structuring the soft shadow algorithms covered in the remainder of this chapter. We will first cover image-based approaches in Section 5.2 before treating geometry-based (and hybrid) methods in Section 5.3.

Image-based algorithms offer several advantages. First, current graphics hardware features direct support for both the generation and the query of shadow maps, which are pivotal to many such techniques. Second, they typically scale better with scene complexity than geometry-based approaches. Third, such methods can readily deal with geometry altered in the fragment stage via alpha masking, pixel kills or depth modifications. Especially the selective discarding of fragments is fundamental to many recent techniques for raycasting surfaces [Loop and Blinn, 2006; Stoll et al., 2006] and for adapting the silhouette in per-pixel displacement mapping algorithms [Oliveira and Policarpo, 2005; Chen and Chang, 2008]. On the other hand, an image-based representation involves sampling, leading to aliasing problems. In particular, fine structures may be missed and silhouettes are easily captured at a too coarse resolution. Moreover, biasing of the depth values recorded in a shadow map is required to avoid surface acne artifacts due to incorrect self-shadowing.

By contrast, geometry-based approaches avoid aliasing problems thanks to working with the exact occluding geometry. On the downside, such algorithms are typically slower than image-based methods.

Computational Space for Visibility Computation Especially for scattering approaches, the space in which receiver samples are considered for visibility computations is a major design choice. In *light space*, it is rather straightforward to determine a bounding region of the affected samples, and such regions are typically rather compact, avoiding the inclusion of a large amount of unaffected samples. Most approaches adopt an ordinary regular sampling of the light-space image plane, easily represented by a texture, and project the resulting visibility image to the screen space. This, however, restricts the supported blocker–receiver configurations and easily leads to aliasing. A better but more costly alternative is to adopt an irregular sampling like in the anti-aliased shadow map (cf. Section 3.2.5), where the sample positions correspond to pixels in the final image.

Alternatively, one may operate directly in *screen space*. While this circumvents the shortcomings of light-space approaches, it leads to looser bounding regions, often requiring processing many unaffected samples. A prominent example suffering from high fill-rate requirements and significant overdraw are shadow volumes and its derivatives.

Further Aspects Algorithms also differ in whether they treat all relevant occluders or focus only on a certain subset of them. For instance, one might solely consider those occluders which are directly visible from the center of the light source, e.g. as captured by a shadow map. Although such a choice easily misses relevant occluders and hence may lead to clearly incorrect soft shadows, it often yields satisfactory results, helps to significantly reduce occurring occluder fusions in the first place, and decreases the required computational effort.

Furthermore, various options exist for what kind of visibility information is determined. Apart from a scalar visibility factor, one may store visibility for selected sample points on the light source, or even regional information about the unoccluded parts of the light source. Note that the latter two approaches enable the evaluation of the soft shadow equation (Equation 1.3).

Partially depending on the adopted options for the discussed algorithmic aspects, developed soft shadow techniques have different quality and performance characteristics. Important quality properties are whether occluder fusion is handled correctly, whether the employed occluder representation is approximate (basically true for all image-based approaches) or exact, whether discretization or aliasing artifacts may arise, and ultimately whether the computed soft shadows are accurate.

In the following sections, we review mainly relevant algorithms but also briefly mention approaches which are now merely of historical interest, to better illustrate the vast range of the solution space. For a more exhaustive coverage of methods devised until 2003, the survey by Hasenfratz et al. [2003] is a good source.

5.2 Image-Based Approaches

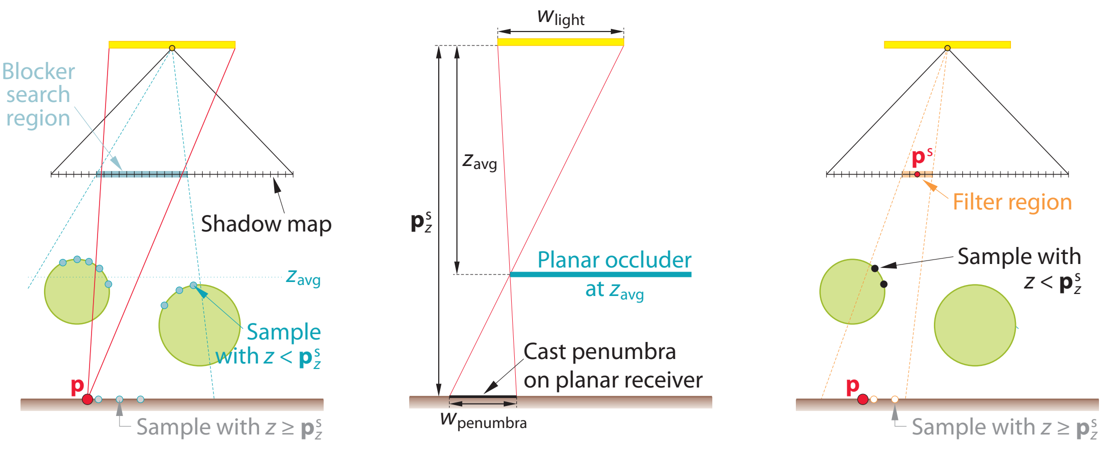
We start our overview of existing soft shadow techniques with image-based approaches. Their common and central characteristic is the use of some image-based representation of the occluders to derive soft shadows.

5.2.1 Plausible Faking by Adapting Hard Shadows

Nowadays being no longer of practical interest due to their severe limitations and rather low quality, many early techniques tried to fake soft shadows in a phenomenologically plausible way by introducing some kind of penumbra. They are often based on the idea of starting with the hard shadow obtained for a point light source placed at the center of the extended light and enlarge it outwards to generate a penumbra region [Parker et al., 1998].

Brabec and Seidel [2002] radially search a shadow map for the nearest texel containing depth information of an occluder and use both the distance to the corresponding texel and the read depth value to estimate a pixel's placement within the found occluder's soft shadow, thus adding outer penumbrae and replacing some hard shadows by inner penumbrae. In similar work, Kirsch and Döllner [2003] move all involved computations to the GPU but become further restricted to handle only inner penumbrae. They derive a shadow-width map storing the distance to the next unblocked pixel by iteratively filtering a black-and-white map of the occluders.

By contrast, Arvo et al. [2004] create outer and inner penumbrae by applying a modified flood-fill algorithm in screen space to the umbra regions obtained from a shadow map. At first, hard shadow borders are detected by executing a Sobel filter. Then, in successive render steps, the detected discontinuities are grown by a single pixel at a time which is added to the interior and exterior of these borders. During this shadow growing, the maximum blocking shadow map sample is propagated. The cost of the algorithm is directly related to the size of the penumbra on the screen, so it can be efficient for very small penumbrae. But depending on the viewpoint


Blocker search region
Shadow map
z_{avg}
Sample with $z < \mathbf{p}^s_z$
$\mathbf{p}$
Sample with $z \geq \mathbf{p}^s_z$
w_{light}
z_{avg}
$\mathbf{p}^s_z$
Planar occluder at z_{avg}
Cast penumbra on planar receiver
$w_{penumbra}$
$\mathbf{p}^s$
Filter region
Sample with $z < \mathbf{p}^s_z$
$\mathbf{p}$
Sample with $z \geq \mathbf{p}^s_z$

(a) Blocker search (b) Penumbra width estimation (c) Filtering

Figure 5.2 Percentage-closer soft shadows [Fernando, 2005] first search the shadow map for blockers, then, assuming a single planar occluder at their average depth z_{avg}, derive the penumbra width and determine a filter window size, and finally filter the shadow map results accordingly to get approximate soft shadows.

and the light's position, this area may be very large and the flood-fill is not well supported by current graphics hardware. It still remains relatively slow even when using the optimization of jump flooding [Rong and Tan, 2006].

All algorithms of this class suffer from several inherent limitations which often lead to clearly visible artifacts. For instance, since the search radius is usually bounded and only the occluders closest in shadow map space are found, relevant occluders might get ignored, limiting the penumbra's extent and causing transition artifacts. Moreover, the umbra regions are often significantly overestimated, especially with techniques only accounting for outer penumbrae. Also note that flood-fill-like methods operating on initial screen-space hard shadows ignore penumbrae of blockers whose hard shadows project outside the screen border.

5.2.2 Percentage-Closer Soft Shadows: Adaptive Blurring of Hard Shadow Map Results

Recall from Chapter 4 that blurring the shadow map result with PCF can yield results which appear like soft shadows. Actually some games and game engines implement just this and call it "soft shadows". However, irrespective of the physical incorrectness of such an approach, using a fixed-size filter window is unable to reproduce varying penumbra widths and hence visually prominent soft shadow behavior like hardening on contact.

To alleviate this major shortcoming, Fernando [2005] suggested to adaptively vary the filter window size across screen space according to a simple heuristic assuming a single planar occluder. His pretty popular technique, called *percentage-closer soft shadows* (PCSS), comprises two main steps: blocker search and PCF filtering.

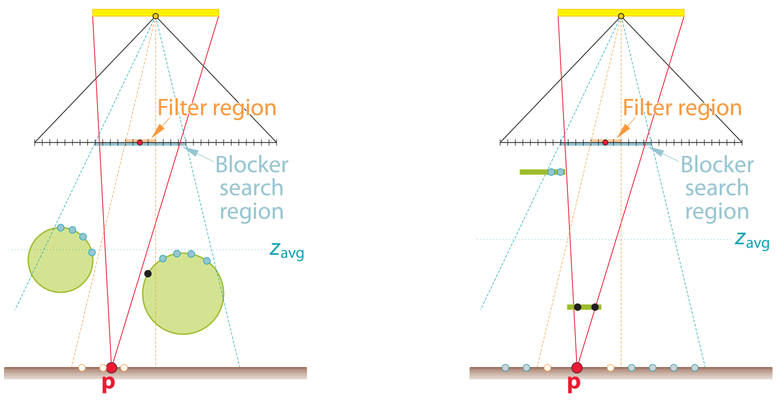


Figure 5.3 Percentage-closer soft shadows can easily lead to wrong results. In the left example, all samples considered as occluding (colored blue and black) during the blocker search are actually not blocking any light from the receiver point; analogously, the black sample in the resulting filter window is wrongly treated as occluding in the filtering step. Consequently, the algorithm erroneously yields a penumbra response (25% shadowing, one sample blocking, three not) for the entirely lit receiver point **p**. Similarly, in the setup on the right side, the receiver point is wrongly considered to be in penumbra (50% shadowing, two samples blocking, two not) instead of in umbra.

Blocker Search At first, the shadow map region $\mathcal{R}_s$ defined by the intersection of the light–receiver-point pyramid (with the area light as base and the receiver point as apex) with the shadow-map near plane is searched for blockers, i.e. for entries which are closer to the light source than the receiver point **p**, and the average depth z_{avg} of them is determined. Based on this information, the algorithm now assumes a single planar occluder at this average depth which is parallel to the light source. Moreover, the receiver point is assumed to belong to a planar receiver that is likewise parallel to the planar occluder (see Figure 5.2).

Filtering Utilizing the intercept theorem, the penumbra width is determined via

$$w_{\text{penumbra}} = \frac{\mathbf{p}_z^s - z_{\text{avg}}}{z_{\text{avg}}} w_{\text{light}},$$

where $\mathbf{p}_z^s$ is the light-space depth of receiver point **p** and w_{light} denotes the size of the light source. Subsequently, the PCF window width is derived from the penumbra width, for instance via

$$w_{\text{f}} = \frac{w_{\text{penumbra}}}{\mathbf{p}_z^s} z_{\text{near}},$$

with z_{near} being the shadow-map near plane distance. Finally, the shadow map is queried and filtered according to this window size.

5.2.2.1 Issues

While the results attained with PCSS are often visually pleasing, at least for simple settings, the approach suffers from several inherent shortcomings:

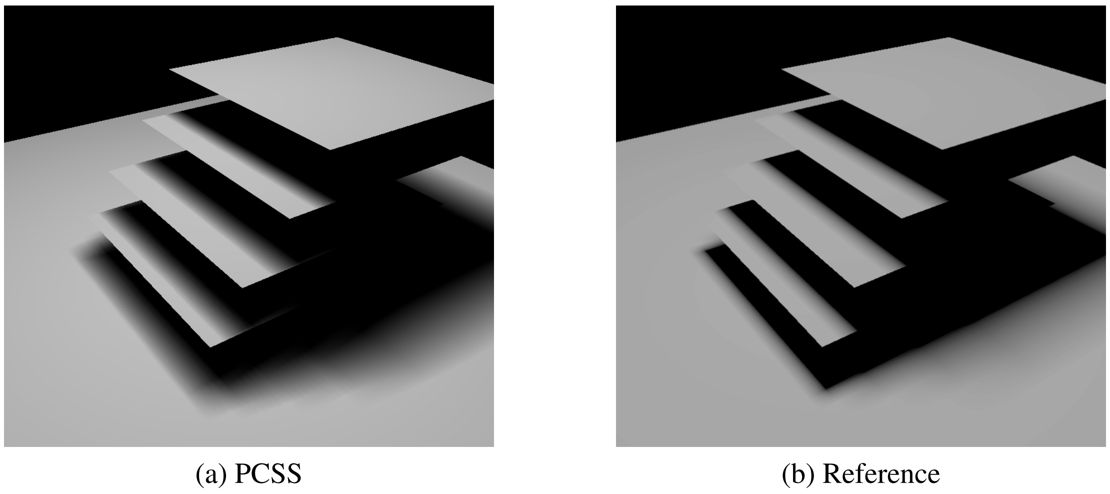

(a) PCSS (b) Reference

Figure 5.4 PCSS, classifying a shadow map texel as occluder solely based on its depth, thus ignoring whether it actually blocks light, can easily yield too large penumbrae due to incorrect excessive blurring.

- A single planar occluder parallel to the shadow map's near plane is assumed which is not always a good approximation.
- When processing shadow map samples, occluders are identified by just checking whether a recorded point is closer to the light than the receiver point $\mathbf{p}$. Consequently, shadow map entries which are outside the light–point pyramid and hence don't block the light at all from $\mathbf{p}$ may be wrongly considered as occluders. This can lead to incorrect estimates of the average depth of the relevant occluders, as well as to wrong visibility values (cf. Figures 5.3 and 5.4).
- A direct consequence of the previous shortcoming is that in case the assumption of the receiver being planar and parallel to the light source doesn't hold, incorrect self-shadowing can easily occur, necessitating large bias values.
- During filtering, the fraction of the light source that is occluded by a blocking sample is assumed to equal its PCF filter weight.

Another line of problems arise from the fact that both searching the shadow map for occluders as well as performing percentage-closer filtering requires many shadow map accesses, thus limiting the achievable performance. Consequently, in practice, the relevant shadow map region $\mathcal{R}_s$ is not searched exhaustively but blocker depth estimation takes just a limited number of samples into account. The resulting subsampling in case of large search windows $\mathcal{R}_s$ can cause neighboring pixels to choose largely varying average blocker depths, leading to different PCF window sizes and hence degrees of shadowing. Similar issues can arise if the number of samples employed for PCF is smaller than the number of texels in the filter window.

Fortunately, solutions to these latter, performance-related problems have been devised, ultimately rendering PCSS a constant-time algorithm. We cover them in the next two subsections.

5.2.2.2 Speeding Up Adaptive Filtering

Recalling from Section 4.2 that alternative shadow map representations which allow for prefiltering exist, it is obvious that the filtering cost in the PCSS algorithm can be reduced significantly by resorting to a technique like VSM or CSM. Note, however, that since the filter window size varies across screen to account for different penumbra sizes, prefiltering has to support variable window sizes. To this this end, typically one of the following two approaches are adopted: mipmapping and summed-area tables (see also the box on page 55).

Mipmapping The easiest and cheapest option is to create a mipmap chain of the shadow map representation, e.g. of the VSM, with each coarser level resulting from blurring the next finer one with a fixed-size filter kernel. Filtering with a certain window size is then simply approximated by appropriately sampling this mipmap with trilinear interpolation.

On the downside, mipmapping provides only prefiltering for several discrete filter sizes and requires filtering with intermediate sizes to be approximated by interpolation. Moreover, the spatial resolution of the prefiltering solution decreases with increasing filter sizes, which can easily cause artifacts. This particular shortcoming can be alleviated by employing an N-buffer-like representation [Décoret, 2005; Eisemann and Décoret, 2006b], which however consumes more memory and is more costly to build.

Summed-Area Table Superior results can be achieved by constructing a summed-area table (SAT) [Crow, 1984], which supports box filtering for arbitrary rectangular filter windows—utilizing just four texture fetches. However, SATs require more memory and are more expensive to create than mipmaps. An important issue is that summing up vast amounts of a certain quantity (e.g. the first moment in case of VSMs) can easily lead to a loss of numerical precision. Therefore, starting at a certain shadow map resolution, it becomes necessary to adopt a 32-bit integer/fixed-point (or if available 64-bit float) representation to avoid related artifacts. Lauritzen [2007] provides more detail and discusses using SATs together with VSMs.

5.2.2.3 Speeding Up Blocker Search

Utilizing the machinery of CSM, it is also possible to speed up the derivation of the average blocker depth z_{avg} and at the same time exhaustively search the relevant shadow map region $\mathcal{R}_{\text{s}}$. Annen et al. [2008a] observe that averaging the depths $z_{\mathbf{q}}$ of all entries $\mathbf{q}$ within $\mathcal{R}_{\text{s}}$ which are closer to the light and hence pass the complementary shadow test

$$\overline{S}(\mathbf{p}^{\text{s}}_z, z) = 1 - S(\mathbf{p}^{\text{s}}_z, z) = 1 - \text{step}(z - \mathbf{p}^{\text{s}}_z) = \begin{cases} 1, & z < \mathbf{p}^{\text{s}}_z \\ 0, & z \geq \mathbf{p}^{\text{s}}_z \end{cases}$$

can be expressed as a convolution:

$$z_{\text{avg}}(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\mathcal{R}_{\text{s}}} \omega_{\text{avg}}(\mathbf{p}^{\text{s}} - \mathbf{q}) \cdot \left(\overline{S}(\mathbf{p}^{\text{s}}_z, z_{\mathbf{q}}) \cdot z_{\mathbf{q}}\right)}{\sum_{\mathbf{q}\in\mathcal{R}_{\text{s}}} \omega_{\text{avg}}(\mathbf{p}^{\text{s}} - \mathbf{q}) \cdot \overline{S}(\mathbf{p}^{\text{s}}_z, z_{\mathbf{q}})},$$

where ω_{avg} is the averaging kernel and the denominator performs normalization. While the denominator can readily reuse the basis images from the regular CSM (since it essentially equals

$1 - \mathrm{pcf}(\mathbf{p})$ from Equation 4.5), the numerator requires computing additional basis images for the product $\overline{S}(\mathbf{p}_z^s, z_\mathbf{q}) \cdot z_\mathbf{q}$. As a consequence, blocker search becoming a constant-time operation like filtering comes at the price of necessitating many basis images which incur both additional storage and creation cost compared to the original PCSS approach of simply sampling the shadow map for blockers.

Note that more detailed derivations are provided in the cited papers. Furthermore, some insights concerning the practical implementation of PCSS and its sped up variants are given by Bavoil [2008].

5.2.3 Soft Shadow Mapping: Reconstructing and Backprojecting Occluders

While percentage-closer soft shadows employ some gross approximations and may easily fail to produce correct results, an important group of algorithms aims at producing physically plausible soft shadows by utilizing a more accurate approximation of the occluding geometry and treating the contribution of shadow map samples to light occlusion more precisely; in particular, samples of geometry not occluding the light source at all are correctly ignored. These techniques, which are often subsumed under the term *soft shadow mapping* or are alternatively referred to as *backprojection* methods,[1] employ a shadow map obtained from the light source's center and reconstruct potential occluders in world space from it. These are then backprojected from the currently considered receiver point **p** onto the light's plane to estimate the visible fraction of the light area. The various methods mainly differ in what kind of occluder approximation is employed, how the computations are organized, and how the occluded light area is derived. Among others, these choices directly influence performance, generality, robustness and visual quality.

The general approach was initially introduced by Atty et al. [2006], who adopt shadow map texels unprojected into world space as micro-occluders, termed *micropatches*, and, looping over all micropatches, scatter the occlusion caused by them to all affected pixels in a so-called soft shadow map, which finally gets projected onto the scene. As such a light-space computation of the scene shadowing has major shortcomings and limitations, e.g. requiring a separation of objects into shadow casters and shadow receivers, all other subsequently devised techniques choose a gathering strategy and operate in screen space. According approaches have been concurrently published by Guennebaud et al. [2006], Bavoil and Silva [2006] and Aszódi and Szirmay-Kalos [2006], with all following soft shadow mapping papers essentially building and improving on them.

In the following, we initially detail the basic approach, concentrating on the variant introduced by Guennebaud et al. [2006]. Serving as foundation for more advanced techniques, we then look closer at the largely orthogonal aspects of visibility determination, occluder approximations and acceleration strategies in the subsequent subsections.

[1]Note that both terms are rather fuzzy. Actually only one algorithm of this class [Atty et al., 2006] creates a *soft shadow map* but all algorithms utilize a shadow map to compute soft shadows—like most other image-based techniques discussed in this chapter. Concerning backprojection, again several other algorithms perform some kind of projection onto the light, particularly most geometry-based techniques covered later in this chapter.

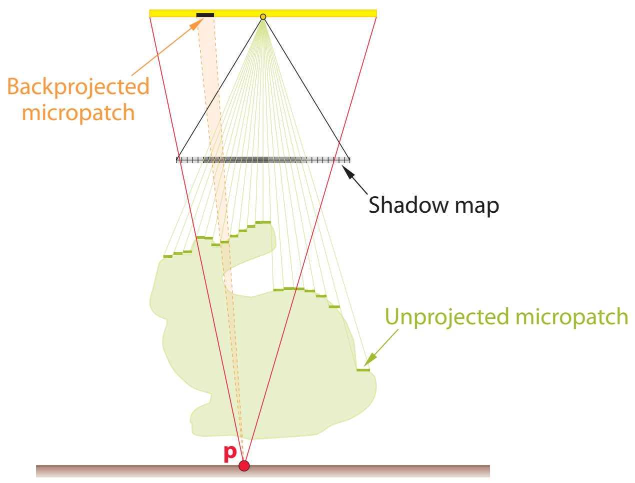


Figure 5.5 Basic soft shadow mapping derives occluder approximations by unprojecting shadow map texels into world space, and backprojects these micropatches onto the light source, accumulating the occluded areas, to determine its visibility from a point **p**.

5.2.3.1 Basic Approach

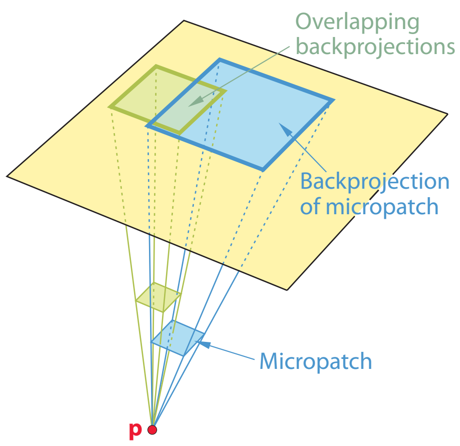
At first, a standard shadow map is generated from the center of the extended light source.[2] This depth map provides a point sampling of the scene geometry visible from the light center, and hence a representation of a subset of the occluders. An approximation of the captured geometry is generated by constructing a micro-occluder for each shadow map element. Typically, the whole texel is just unprojected into world space, resulting in a rectangular *micropatch* parallel to the shadow map's near plane, as illustrated in Figure 5.5. Alternative occluder approximations have been devised and are discussed below in Section 5.2.3.3.

The light visibility for a receiver point **p** is then computed in a pixel shader, where the shadow map is traversed and for each texel a micropatch is constructed on the fly. If a micropatch is closer to the light source than **p**, it potentially blocks some part of the light. In this case, the micropatch is backprojected from **p** onto the light plane and clipped against the light's extent to determine the occluded light area. The individual covered light areas of all backprojected micropatches are summed up to get an estimate of the overall occluded light area. Relating this to the total light area yields a visibility factor describing the percentage of light visible to **p**.

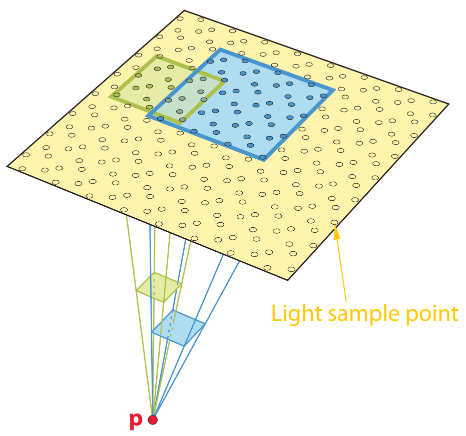
5.2.3.2 Visibility Determination With Occlusion Bitmasks

The simple approach of combining the occlusion of individual micropatches by accumulating the light areas covered by them is only correct if the projections of the micropatches onto the light source don't overlap. This, however, is typically not the case (cf. Fig. 5.6 a), although initially recording only occluders visible from the light's center surely helps obviating overlaps. The resulting incorrect occluder fusion leads to overestimating light occlusion and may cause clearly objectionable artifacts.

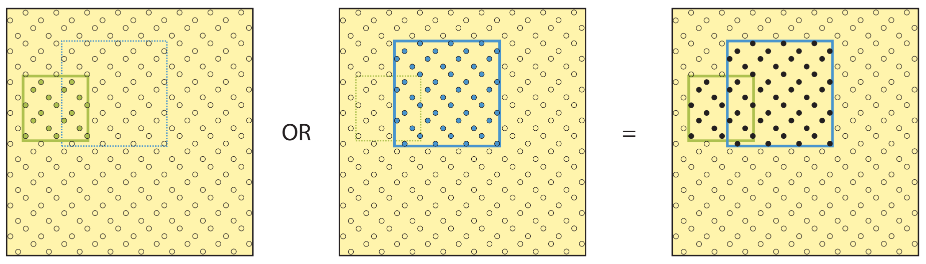
[2]In principle, any sample point on (or behind) the light source may be used instead of the center.



(a) Overlapping micropatches



(b) Light sample points



(c) Occlusion bitmask update

Figure 5.6 The backprojections of multiple micropatches often overlap (a). This can be correctly dealt with by considering the binary visibilities at many light sample points (b). These are efficiently encoded in a bit field; this occlusion bitmask is successively updated by ORing in the samples occluded by a certain micropatch (c).

Occlusion Bitmasks A robust solution for visibility determination that properly deals with arbitrary such overlaps was introduced by Schwarz and Stamminger [2007]. In their *bitmask soft shadows* (BMSS) algorithm, they approach the visibility determination as a sampling problem. Sample points are placed on the light source and a bit field is used to track which of them are occluded. The resulting *occlusion bitmask* provides a discrete representation of which light area parts are occluded. By counting the set bits, the light visibility percentage can easily be determined (cf. Equation 1.6).

When backprojecting a micro-occluder, rather then computing its area, a bitmask reflecting which light samples are occluded by it is determined, and incorporated into the existing occlusion bitmask with a bitwise OR. In case of overlapping micro-occluders the same bit gets set multiple times, i.e. occluder fusion is automatically dealt with correctly. Note that this technique is not restricted to soft shadow mapping but provides a general solution to the occluder fusion problem. The same fundamental approach is now also successfully used in accurate geometry-based approaches and we hence encounter it again in Section 5.3.3.

In principle, the sample points can be placed arbitrarily on the light source. However, for performance reasons, Schwarz and Stamminger [2007] suggest to restrict the positioning and number of sample points such that fast updates of the occlusion bitmask are possible using only arithmetic operations, exploiting the fact that a micropatch's backprojection is just an axis-aligned rectangle on a rectangular light source. Concretely, they advocate a sampling pattern with 256 regularly jittered sample points corresponding to an 8×8 tiling of the 2×2 rotated grid super sampling (RGSS) pattern commonly used for antialiasing [Akenine-Möller et al., 2008].

Note that arbitrary sampling patterns are possible, but require three texture look-ups per micropatch[3] (or one look-up per edge of the backprojection in case of other occluder approximations like microquads, detailed below) and hence lead to a significantly lower performance.

Discussion Since visibility is determined by point sampling, encountered light visibility factors can only take on a limited number of different discrete values, defined by the size of the bit field. Consequently, discretization artifacts may arise. It is hence pertinent to employ a large enough number of light samples, with 256 usually sufficing (if jittered). Note that in practice, discretization artifacts are often acceptable, especially because they are usually masked by the texture of the shadow-receiving surface and hence remain (largely) imperceptible.

Advanced Applications Since occlusion bitmasks can correctly handle arbitrarily overlapping micro-occluders and also provide explicit information about which parts of the light source are blocked instead of offering just a visibility factor, new applications beyond standard soft shadow mapping become possible.

Most notably, occluder information from multiple shadow maps, e.g. obtained via depth peeling, can easily be incorporated, thus basically enabling to capture all occluders instead of just the subset visible from the center of the light.[4] Note, however, that rendering cost increases essentially linearly with the number of considered shadow maps.

Leveraging the provided spatial information about the occluded parts of the light source, it further becomes possible to correctly deal with multi-colored light sources and even evaluate the soft shadow equation from Equation 1.3.

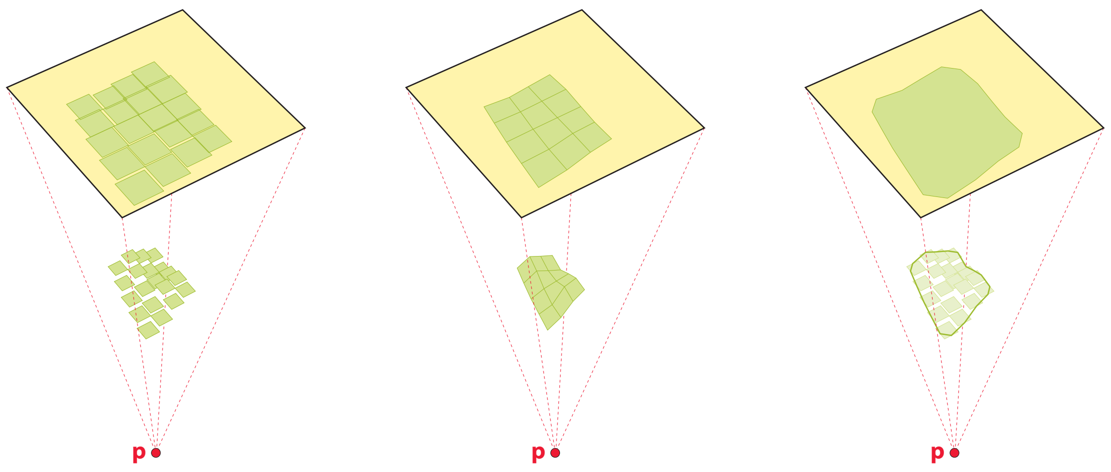
5.2.3.3 Occluder Approximations

Reconstructing an approximation of the captured occluders from the point sampling provided by the shadow map is central to soft shadow mapping approaches. Various options with different strengths and weaknesses have been proposed, which we briefly review in the following (see also Figure 5.7).

Micropatches Let us first look closer at the historically first option already introduced above, micropatches, before covering alternatives which try to improve on them. Recall that a micropatch is constructed by unprojecting a shadow map texel into world space. Together, micropatches provide a piecewise-constant approximation of the captured occluders. Since they are parallel to the light plane such that their backprojections are axis-aligned rectangles, many

[3]Since the backprojection is an axis-aligned rectangle, a strategy similar to summed-area tables can be used.

[4]Another option to capture more occluders and hence decrease artifacts due to ignored occluders is to split the light into multiple sub-lights and treat each sub-light separately [Assarsson et al., 2003; Yang et al., 2009] (see also Section 5.3.2.3). However, this comes at the cost of rendering a shadow map (and creating an according multiscale representation; see below) multiple times. Moreover, while this approach is orthogonal to the chosen visibility determination technique, overlapping-related artifacts still occur unless occlusion bitmasks are used.



(a) Micropatches (b) Microquads (c) Occluder contours

Figure 5.7 Various occluder approximations can be derived from a shadow map.

operations are rather simple and hence fast to execute. However, this simplicity also causes micropatches to suffer from several problems. For instance, occluders are frequently overestimated, potentially leading to noticeable enlargements of the penumbra's extent. On the other hand, potentially overestimating an occluder's extent helps capturing fine structures.

A major issue is that gaps can occur between neighboring micropatches. Usually, such gaps are not actual occluder-free regions but undesired holes in the reconstruction of surfaces, which lead to disturbing light leaks. Given the lack of information allowing a correct discrimination, it is hence reasonable to try to close gaps. To this end, Guennebaud et al. [2006] consider the left and bottom neighbors in the shadow map for each micropatch, dynamically extending it appropriately to the borders of these neighbors (cf. Figure 5.8 a). However, gaps towards the diagonal neighbor may still exist [Schwarz and Stamminger, 2007], which can be alleviated by explicitly accounting for this neighbor, too [Bavoil, 2008]. As a significantly more expensive alternative, Bavoil et al. [2008a] advocate using a multi-layer shadow map obtained via depth peeling and, for a given texel, using only the layer farthest away from the light that is still closer to the light than the receiver point for micropatch construction.

Treating gaps as reconstruction errors

Invariably closing gaps is a necessary approximation. It knowingly accepts overocclusion artifacts due to blockers that are wrongly introduced by assuming that two adjacent shadow map texels sample the same surface. Actually, correctly dealing with such gaps in absence of further information is a general problem also encountered in raytracing of depth images [Lischinski and Rappoport, 1998; Keating and Max, 1999]. While heuristics were developed, like performing gap filling between two adjacent micropatches only if their depth difference is below a user-specified threshold [Agrawala et al., 2000], they are far from robust.

Furthermore, micropatches easily lead to surface acne and require biasing to alleviate such self-shadowing artifacts. Using a mid-point shadow map (see Section 2.1.2) for depth comparisons [Bavoil et al., 2008a] helps, but incurs additional costs.

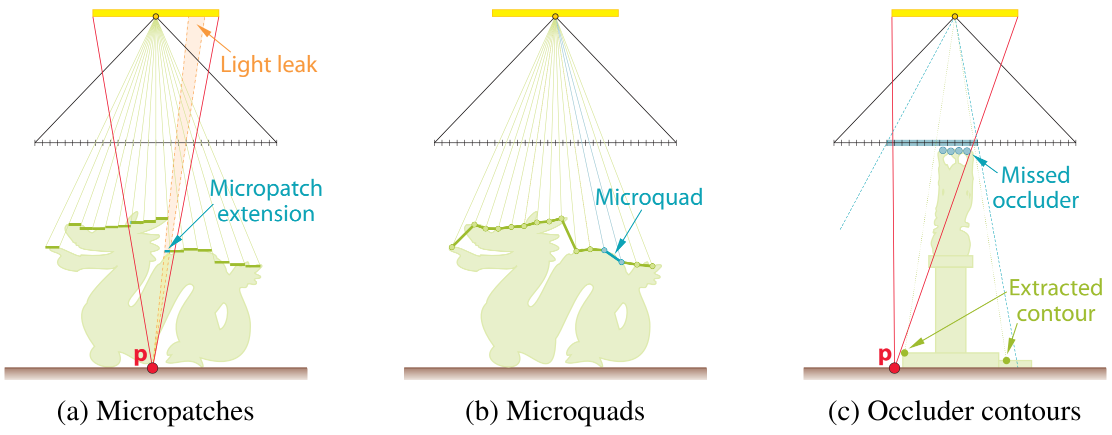


Figure 5.8 (a) Micropatches suffer from light leaks, which can be (approximately) closed by dynamically extending them. (b) By contrast, microquads implicitly avoid such gaps in the occluder reconstruction but can miss thin structures. (c) Occluder contours may ignore occluders captured by the shadow map.

Finally, note that Bavoil and Silva [2006] employ the bounding sphere of a micropatch as occluder, and compute the subtended solid angle to determine visibility of a spherical light source.

Microquads and Microtris Schwarz and Stamminger [2007] take the unprojected centers of the shadow map texels as vertices, which along with their texture-space adjacencies implicitly define a regular quad mesh. Each face serves as a micro-occluder, called *microquad*. It is created from four vertices corresponding to 2×2 neighboring texels, and gets only taken into account during visibility determination if all four vertices are closer to the light source than the point for which light visibility is computed.

Microquads provide a piecewise-(bi)linear interpolation and hence adapt better to the actual geometry than micropatches, and are thus less prone to cause surface acne. Another advantage is that since adjacent microquads share a common boundary no unwanted gaps occur in the first place (cf. Figure 5.8 b), and hence light leaks are avoided. Moreover, because two neighboring microquads are connected by an edge, their backprojections usually don't overlap, which is in strong contrast to the situation with micropatches as these are just isolated primitives. But overlaps can still occur and hence utilizing occlusion bitmasks is still advisable for high quality.

Since a quad's backprojection onto the light area does not yield an axis-aligned rectangle in general, correct clipping and area determination as well as occlusion bitmask updates are complicated. While accurate solutions have been devised [Schwarz and Stamminger, 2008a], they are rather expensive. Therefore, in practice, usually the simple approximate approach by Schwarz and Stamminger [2007] is employed which yields visually hardly distinguishable results but is roughly as fast as micropatch processing.

In contrast to micropatches, microquads have a tendency to underestimate occluders and miss thin geometry like twigs and branches covering only a single shadow map texel in width. Augmenting microquads with microtris [Schwarz and Stamminger, 2008a] helps reducing the underestimation and also improves smoothness at features lying diagonal in shadow map space. A *microtri* is simply a triangle constructed if a microquad gets ignored because only three of its four vertices pass the distance test, using these three vertices. However, this often merely slight visual quality improvement is faced by a considerable performance impact.

Occluder Contours Guennebaud et al. [2007] construct an *occluder contour* for each connected region of shadow map texels passing the depth test. To this end, they slide a window of 2×2 adjacent texels across the search area, and for each window position employ the corresponding binary depth test results to consult a lookup texture for deriving a set of oriented edges which together ultimately form the contours. Each edge is backprojected and the signed areas of the resulting radial segments with respect to the light center are accumulated to derive light visibility. Because only contour edges have to be backprojected, and their number is typically smaller than the equivalent micropatch count, some computations are saved. However, all shadow map texels within the search area still have to be accessed nevertheless.

Since a contour encompasses all neighboring shadow map samples passing the depth test, light leaks are implicitly avoided. However, since contours are extracted in 2D instead of 3D space, occluders recorded in the shadow map may be missed (cf. Figure 5.8 c). This can lead to noticeable popping artifacts as the depth values at the 2D contour may jump when the light moves relative to the occluder (even if the triggering occluder is captured in both the old and the new shadow map).

On the other hand, the way contours are constructed often causes neighboring receiver pixels to exclusively process the same occluder contours. This coherence is exploited by Yang et al. [2009] in their packet-based approach to speed up visibility determination. They group $N \times M$ receiver points to a packet and treat them at the same time (allocating a single thread per packet unlike packet-based ray tracing methods), sharing the dominating contour extraction computation among them. In case the receiver points of a packet don't have identical contours, the packet-based evaluation is aborted and all NM points are processed independently.

5.2.3.4 Acceleration With Multiscale Representations

For a reasonably high performance, it is essential to avoid useless computations like processing micro-occluders which don't affect the result. An effective tool for approaching this obvious goal are multiscale representations of the shadow map, which can yield significant acceleration.

Search Area Determination Instead of naively looping over all shadow map texels, ideally only those micro-occluders are processed for a certain receiver point which actually project onto the light source and hence block some light. In practice, a corresponding rectangular shadow map *search area* encompassing these relevant micro-occluders is determined and only the shadow map texels in the search area are considered.

A conservative first estimate of the search area is given by intersecting the near plane with the point–light pyramid defined by the point $\mathbf{p}$. It can be further tightened if the depth range $[z_{\min}, z_{\max}]$ of the samples within the search area is known [Guennebaud et al., 2006]: By intersecting the plane $z = z_{\min}$ with the pyramid and projecting the result onto the near plane, the search area may then be refined iteratively (see Figure 5.9).

Knowledge about the depth range of the search area further allows identifying fragments in umbra and completely lit regions where no micro-occluders need to be processed at all [Guennebaud et al., 2006]. More precisely, if $z_{\mathbf{p}} > z_{\max}$ holds, where $z_{\mathbf{p}}$ denotes the shadow-map depth

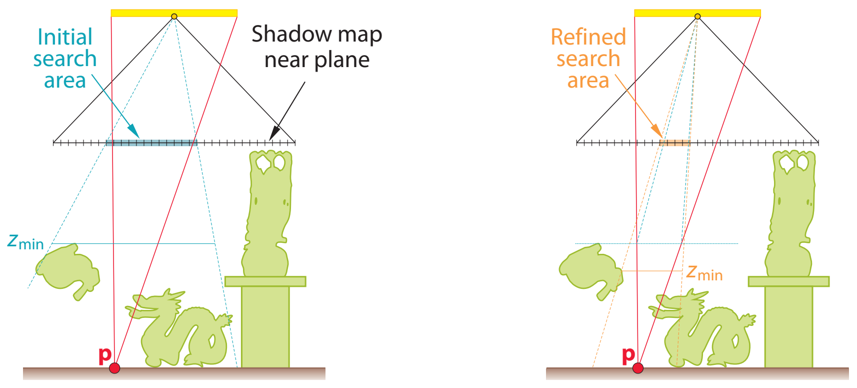


Figure 5.9 The intersection of the shadow map near plane with the point–light pyramid yields an initial search area for extracting occluder approximations. If the minimum depth $z_{\min}$ of this area is known, it becomes possible to refine the search area iteratively.

value for point **p**, it can safely be assumed that the light is totally blocked. Similarly, $z_{\mathbf{p}} \leq z_{\min}$ ensures that the whole light source is visible.

Hierarchical Shadow Map To quickly determine a conservative bound of the depth range of a shadow map region, a multiscale representation of the shadow map proves useful. The *hierarchical shadow map* (HSM) [Guennebaud et al., 2006], which essentially equals a hierarchical z-buffer [Greene et al., 1993], is a mipmap-like pyramid for the original shadow map that stores successively aggregated minimum and maximum depth values at each coarser level. To answer a depth range query, typically the finest level is chosen where up to 2×2 adjacent texels conservatively cover the area in question. While this keeps the number of required texture fetches constant, the actually considered shadow map area is usually larger than the specified area, resulting in looser depth bounds. As a consequence, the search area is often unnecessarily large and classifications as entirely shadowed or completely lit may be prevented.

Multi-Scale Shadow Map An alternative multi-scale representation which allows much more fine-grained area queries than the HSM, thus drastically reducing the amount of soft shadow computations, is the *multi-scale shadow map* (MSSM) [Schwarz and Stamminger, 2007]. Here, a texel at level i (with $i = 0$ denoting the finest level) stores the minimum and maximum depth values in a neighborhood region of size $2^i \times 2^i$ centered around the texel (similar to an N-buffer [Décoret, 2005]). Typically, an MSSM gets stored in a 2D array texture because this enables the dynamic selection of the sampled level within a shader.

Key to the improved results obtained with the MSSM is its ability to directly support queries for arbitrarily placed squares of power-of-two size. By contrast, the HSM, essentially being a quadtree constructed over the shadow map, is restricted to rectangles corresponding to quadtree tiles. Therefore, an arbitrary power-of-two-sized square typically has to be grown to the next coarser encompassing tile, quadrupling its size. Consequently, the depth range is looser, often precluding a classification as completely lit or in umbra (cf. Figure 5.10).

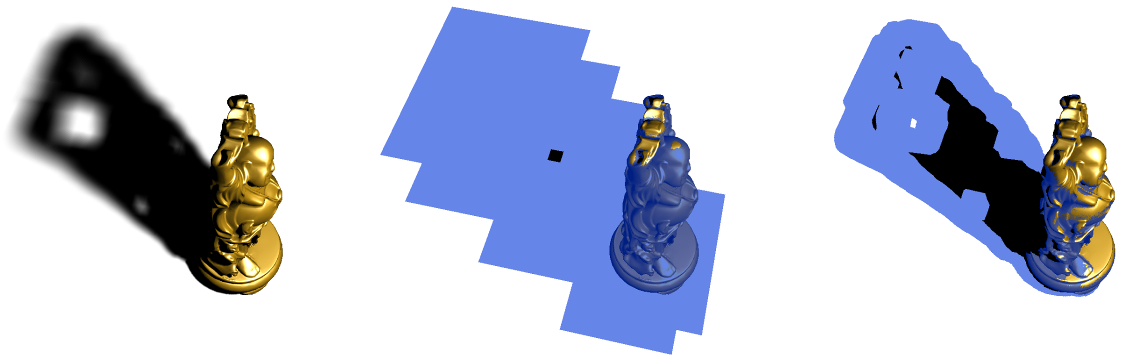

(a) Soft shadows (b) HSM (c) MSSM

Figure 5.10 While an HSM is cheaper to create than an MSSM, the latter offers significantly better classification results, considerably reducing the number of pixels where backprojection has to be actually performed to determine light visibility (colored bluish).

Hybrid Y Shadow Map Unfortunately, both its construction and its memory footprint render the MSSM too expensive for shadow map sizes greater than 1024^2 because the resolution is not reduced across levels. By contrast, the HSM entails significantly lower costs because of its pyramidal nature, which, however, is also responsible for the usually much more conservative results. To get the best of both approaches, a hybrid between the HSM and the MSSM, the *Y shadow map* (YSM) [Schwarz and Stamminger, 2008b], is a good choice in practice. It is constructed by combining the first n levels of the HSM with an MSSM built from level $n - 1$ of the HSM.

Hierarchical Occluder Approximation Extraction Since the HSM constitutes a quadtree constructed over the shadow map, it is also possible to hierarchically traverse this tree to identify and process relevant micropatches [Dmitriev, 2007] instead of determining a tight search area and looping over all texels within it. Similarly, the MSSM can be used to hierarchically extract occluder contours [Yang et al., 2009].

5.2.3.5 Acceleration by Adapting Accuracy

Another popular and effective possibility to speed up shadow computations is to adapt the accuracy, typically by reducing the sampling density.

Micro-Occluder Sub-Sampling Even if only relevant shadow map texels are considered, their number can easily reach and exceed several thousand for a single receiver point. Such large counts take a considerable amount of time to process, frequently preventing real-time performance. One simple possibility to deal with this situation is to restrict the number of micro-occluders that are actually processed, and perform an according subsampling of the search area. Proposed variants include sampling according to a Gaussian Poisson distribution [Bavoil and Silva, 2006] and regular subsampling [Bavoil et al., 2008a].

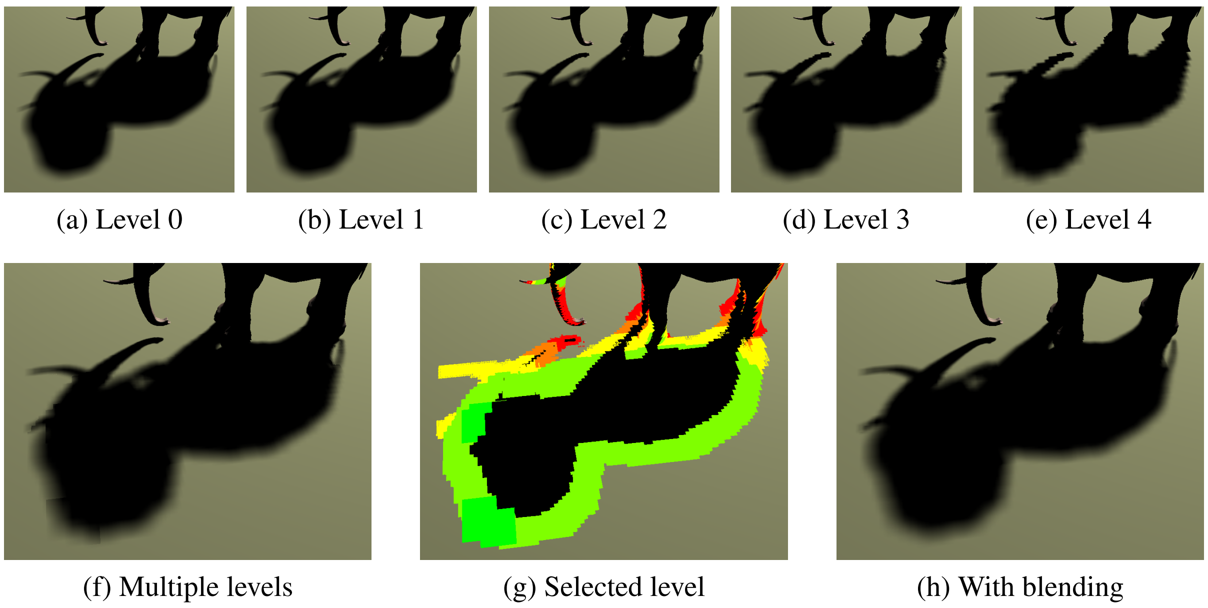


Figure 5.11 Coarser micropatches can be constructed by utilizing a coarser HSM level (a–e). In practice, each receiver pixel chooses the finest level which still satisfies an imposed upper bound on the number of micropatches (f, g). This, however, leads to noticeable transition artifacts (f). These can be alleviated by adopting a blending strategy (h), which incurs some overhead, though.

Coarser Occluder Approximations Another possibility to limit the number of processed shadow map entries is to resort to a coarser-resolution shadow map for constructing the occluder approximation. To avoid overhead and allow different effective shadow map resolutions per receiver point, in practice, simply the minimum channel of an appropriate level of the HSM is employed instead of actually rendering additional shadow maps. Given an imposed upper bound on the number of shadow map texels considered during visibility determination, typically the finest HSM level is selected where the search area comprises few enough texels to meet this budget (see Figure 5.11). Note that an MSSM can be used equally well as the information stored in the MSSM is essentially a superset of that in the HSM.

Picking the minimum depth value of 2×2 adjacent shadow map texel as their representative is simple and also conservative in that it ensures that if at least one of the original samples passes the depth test, then the coarser texel does so as well. Generally, this strategy preserves the tendency of micropatches and microquads to over- and underestimate occluders, respectively. In particular, fine structures are implicitly enlarged with micropatches at each coarser level, whereas microquads increasingly miss thin occluders. Occluder contours behave similar to micropatches in this respect, but they may be moved inwards to reduce the occluder approximation size and hence counter some overestimation; however, this may at the same time also introduce some underestimation.

In addition to that, approximation quality is negatively affected by the missing flexibility of the micro-occluders, entailed by their simplicity. Since, for instance, micropatches have a uniform size in texture space, with them it is often not possible to represent the occluder samples from the original shadow map well at coarser levels. Better results can achieved by generalizing micropatches, as demonstrated with the so-called *microrects* [Schwarz and Stamminger, 2008b], but this naturally incurs some additional overhead due to more expensive generation and representation.

Another source of problems, especially when using an occluder approximation which provides merely a piecewise-constant approximation is the depth bias determination for the coarser levels. In particular, simply using the same bias value for all levels can lead to visual artifacts like surface acne or missing contact shadows.[5]

Furthermore, selecting the level for constructing the occluder approximation individually per receiver point can lead to noticeable transition artifacts (cf. Figure 5.11 f). If different levels are used for two adjacent pixels, different occluder approximations get employed which generally yield an unequal amount of light blocking. To alleviate related artifacts, the employed level can be made a continuous quantity, with visibility being determined by considering the two closest integer levels and combining the results obtained for them via alpha blending [Guennebaud et al., 2007; Schwarz and Stamminger, 2008b] (cf. Figure 5.11 h). Note that this involves processing more shadow map texels, causing some overhead. Furthermore, a scheme has been suggested by Schwarz and Stamminger [2008b] where the shadow map texel budget can be specified locally in screen space, for instance lowering it in regions of high texture masking, which effectively allows to smoothly vary the soft shadow level of quality (LOQ) across screen.

Subsampling in Screen Space Apart from adapting the sampling of the occluders, performance further profits from reducing the sampling density in screen space. Guennebaud et al. [2007], for instance, suggested an according sparse sampling scheme for their soft shadow mapping algorithm, enabling high speed-ups. They derive an estimate of the penumbra's screen-space footprint and employ it to adjust the sampling density by appropriately skipping visibility computations for some pixels. The resulting subsampled visibility solution is then subjected to a pyramidal pull-push reconstruction to determine the final soft shadows. On the downside, with increasing sparseness objectionable patterns can appear with this approach. Moreover, because the underlying sparse sampling pattern is fixed in screen-space, these patterns can be expected to become particularly noticeable in animated scenes.

5.2.4 Approaches Utilizing Multiple Shadow Maps

While a single shadow map only captures the occluders closest to a dedicated light point $\mathbf{x}$, several approaches exist which take further samples along the corresponding rays emanating from $\mathbf{x}$ into account. To that end, they first generate shadow maps from several points on the light source and subsequently merge them into a single extended shadow map for the light center. Two basic options for the representation have been utilized: layered depth images [Shade et al., 1998] and deep shadow maps [Lokovic and Veach, 2000].

Layered Attenuation Maps Agrawala et al. [2000] chose layered depth images (LDIs) [Shade et al., 1998] as representation for their layered attenuation maps. An LDI stores at each pixel a list of depth samples, i.e. several depth layers. For each recorded depth sample, the depth value

[5]Note that this effectively precludes adding a bias during shadow map generation, which should be avoided anyway as it affects the backprojection's position and size, leading to inaccurate shadowing results.

as well as a counter reflecting the number of shadow maps and hence light points from which the sample is visible are maintained. In a preprocess, for all sample points on the light source, shadow maps are successively rendered and warped into the reference view from the light's center, with the warped samples being added to the LDI. If the LDI pixel already features an entry whose depth matches the warped sample's one within some small tolerance ϵ, the entry's counter is incremented. Otherwise a new layer is inserted, with its counter being initialized to one. Finally, an attenuation map is computed by dividing all counters by the number of considered light points. During rendering, this attenuation map is consulted to determine whether a depth sample has been recorded for the receiver point (again within tolerance ϵ), with the accompanying attenuation value indicating the fraction of the light (i.e. the percentage of light sample points) from which the receiver point is visible. If no entry is found in the attenuation map, the receiver point cannot be seen from any (considered sample) point on the light source and hence is in umbra.

Penumbra Deep Shadow Maps By contrast, St-Amour et al. [2005] adopt deep shadow maps [Lokovic and Veach, 2000], storing occlusion as a function of depth for each texel. For each light point, a shadow map is rendered. To incorporate its information, for each texel of the deep shadow map, a ray is cast against the shadow map, recording changes in visibility (events) along the ray. More precisely, the ray from the light center through the texel's center is considered and projected into the shadow map, with all covered shadow map texels being processed. Finally, attenuation as a function of depth is computed by integrating the visibility changes. The function is subsequently compressed by reducing the number of stored function samples such that a prescribed bound on the approximation error is respected. During rendering, the receiver point is projected into light space and the according attenuation factor is read from the penumbra deep shadow map.

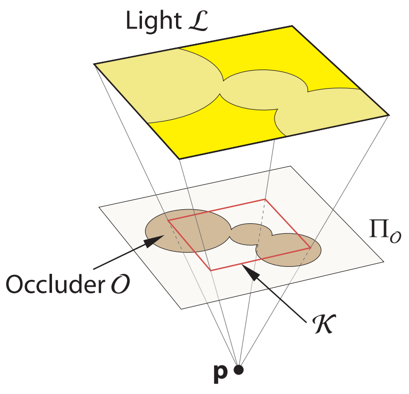
Although such methods allow for rendering rather accurate soft shadows in real-time once the extended shadow map has been created, the generation of this structure is typically only possible at interactive rates, at best. In particular, high quality usually requires considering many light sample points and thus acquiring and incorporating a large number of shadow maps. Consequently, these approaches are limited to static scenes in practice.

Ray tracing against multi-layered shadow maps

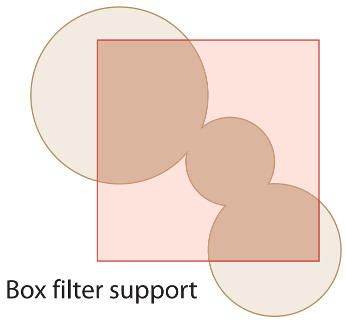
Extended shadow maps storing several depth values per texel provide an augmented image-based scene representation compared to ordinary shadow maps. This is exploited by several (off-line) methods which compute soft shadows by casting rays to sample points on the light source, utilizing such multi-layered shadow maps to determine intersections with the scene. Early examples include the works by Lischinski and Rappoport [1998] as well as Keating and Max [1999]. Agrawala et al. [2000] introduced several quality and performance improvements with their coherence-based raytracing technique. More recently, Xie et al. [2007] presented an algorithm that further supports semi-transparent objects.

5.2.5 Occlusion Textures

A cheaper algorithm that also uses multiple depth layers but runs in real-time are occlusion textures [Eisemann and Décoret, 2006b, 2008]. Unlike the approaches discussed in the previous



(a) Setup



(b) Filter

Figure 5.12 Occlusion textures build on the observation that for a rectangular light and a parallely aligned planar occluder, the light visibility from a point **p** in the scene corresponds to a box filter response.

section, the number of layers and their positions don't vary per shadow map pixel with this technique. Instead, the layers are established by slicing the scene parallel to the light source, and projecting the geometry inside a slice away from the light onto the slice's bottom plane, with the covered parts being recorded in a (binary) occlusion texture per slice. Shadows are then computed by adaptively filtering these textures, leveraging the result by Soler and Sillion [1998] who showed that for an aligned planar light source, planar occluder and planar receiver, the visibility integral from Equation 1.4 becomes a convolution with an appropriately scaled source.

Basic Idea The underlying approach for shadow computation is probably explained best by considering the simple setup from Figure 5.12 with a single planar occluder O parallel to a rectangular light source $\mathcal{L}$. The occluder is fully described by its supporting plane Π_O and the characteristic function

$$\delta_O : \Pi_O \mapsto \{0,1\} \qquad \text{with} \qquad \delta_O(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} \in O, \\ 0, & \text{otherwise.} \end{cases}$$

For a receiving point $\mathbf{p}$, the visible fraction of the light then equals

$$V = \frac{1}{|\mathcal{K}|} \int_{\mathbf{x}\in\mathcal{K}} (1 - \delta_O(\mathbf{x}))\, d\mathbf{x},$$

where $\mathcal{K}$ is the rectangular region in the plane Π_O resulting from intersecting this plane with the light–receiver-point pyramid. The size of $\mathcal{K}$ can easily be determined by scaling the light's size according to the ratio of distances of $\mathbf{p}$ to the occluder and the light, i.e.

$$\text{size}(\mathcal{K}) = \frac{\text{dist}(\mathbf{p}, \Pi_O)}{\text{dist}(\mathbf{p}, \mathcal{L})}\, \text{size}(\mathcal{L}).$$

Consequently, V can be computed by filtering $1 - \delta_O$ with a box filter of size size($\mathcal{K}$).

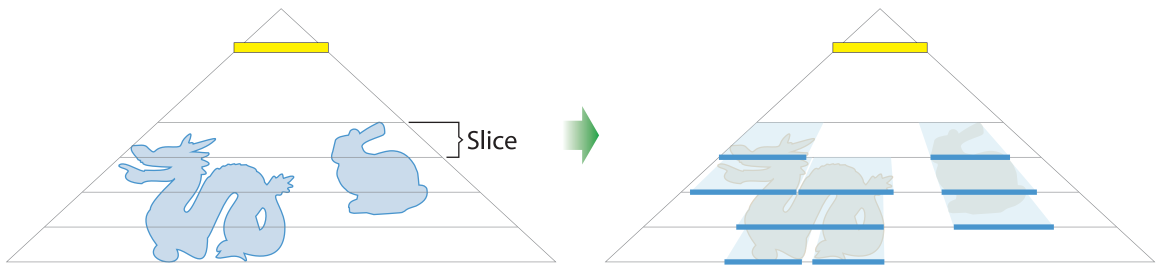


Figure 5.13 Occlusion textures are constructed by slicing the scene and projecting the slice's content to its bottom plane, recording covered parts.

Multiple Planar Occluders For a certain planar occluder, the binary function $1 - \delta_O$ is encoded in an occlusion texture. The shadow at receiver point $\mathbf{p}$ can then easily be determined by sampling an appropriately prefiltered version of this occlusion texture. To support multiple planar occluders, for each unique plane, an occlusion texture has to be determined. These are processed independently when determining the overall light visibility, filtering each of them with an appropriately sized box filter. The resulting visibility values for all planes are finally combined multiplicatively. It is important to note that hence occluder fusion is not treated correctly but handled heuristically, i.e. the resulting shadows are not necessarily correct.

General Occluders Arbitrary scenes are approximated by a set of planar occluders. To this end, the scene is sliced into multiple layers as mentioned above, where between 4 and 16 slices are chosen in practice (cf. Figure 5.13). Note that since each layer is treated as a planar occluder, intra-slice shadowing cannot be captured correctly.

Prefiltering To rapidly determine the box filter response for arbitrary filter sizes, the same methods as mentioned in Section 5.2.2 can be employed, namely mipmapping, N-buffers and summed area tables. While mipmapping potentially leads to noticeable artifacts, using the accurate SATs is costly. N-buffers are hence advocated by the authors as currently best option. Note that their implementation uses multiple textures instead of a texture array, and hence N-buffers can be assumed to perform actually faster than reported if implemented more efficiently, as is possible nowadays.

Properties Occlusion textures yield plausible shadows, but generally not accurate ones. Apart from not treating occluder fusion correctly, this is mainly due to the discretization of the scene into some small number of slices. This approximation can also lead to light leaks, especially in case of thin structures, although a heuristic to combat them was devised. On the other hand, the algorithm's performance is independent of the light's and the penumbra's sizes, routinely achieving real-time frame rates. Note that occlusion textures are one of the rare algorithms available with this property (PCSS with CSM-accelerated blocker search and filtering is another one, cf. Section 5.2.2). Altogether, this renders the technique attractive for compact indoor environments illuminated by a large area light.

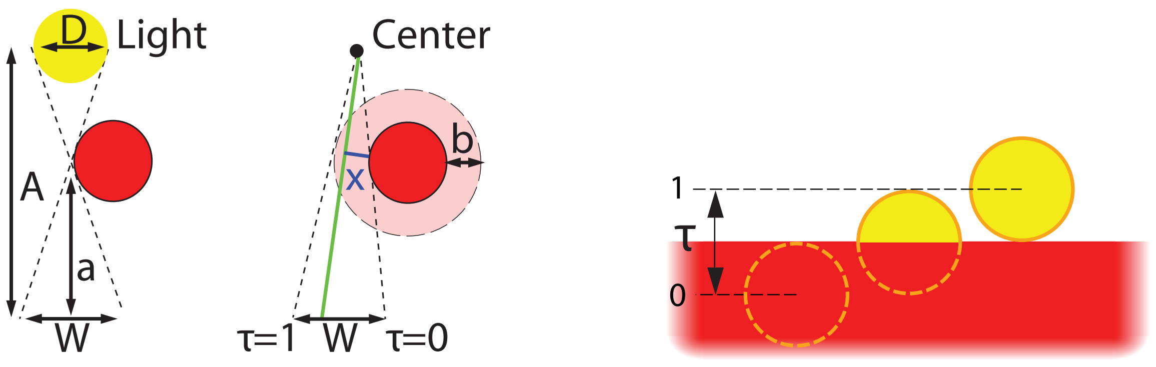


Figure 5.14 Single Sample Soft Shadows [Parker et al., 1998]

5.3 Geometry-Based Approaches

Geometry-based approaches do not share the aliasing and visibility problems of image-based methods. On the other hand, they are usually more computationally expensive.

5.3.1 Plausible Faking by Generating Outer Penumbra

We will start by describing a category of techniques that we classify as *Outer Penumbra* methods. The common denominator is that these methods starts by computing a hard shadow, which is then smoothed outwards to simulate the penumbra. The techniques described in this section can be considered obsolete, nowadays, but have historical interest. These techniques include Single Sample Soft Shadows [Parker et al., 1998], Soft Planar Shadows Using Plateaus [Haines, 2001], Penumbra Maps [Wyman and Hansen, 2003], and Smoothies [Chan and Durand, 2003].

Single-Sample Soft Shadows presented by Parker et al. [1998] provides a very simple technique to achieve a soft-shadow-like effect in the context of ray tracing. The main principle is to derive shadow intensity from a single sample ray based on how closely the ray passes to the original object. To find this distance, each object is surrounded by an englobing geometry that is tested for intersection. The text in [Parker et al., 1998] states that each triangle is enlarged per ray. Many people, however, assumed or proposed the less costly approach with bounding shapes that were chosen conservatively in advance.

The assumption is that the shadow casting surface is locally a plane hiding a part of a spherical source, as illustrated in Figure 5.14 (left).

Parker et al. derived the following formula that describes the amount of the source that is hidden with respect to the source's relative position:

$$s(\tau) = (1 + sin(\pi\tau - \pi/2))/2$$

They point out that a simple polynomial matching (of values and derivatives at the extremities) gives the simple expression: $s(\tau) = 3\tau^2 - 2\tau^3$. To further reduce computational costs, the value can be precomputed and stored in a texture [Haines, 2001; Chan and Durand, 2003].

Another contribution was the way to approximate the penumbra region. This is illustrated in Figure 5.14 (right). The goal is to provide a smooth variation of size equivalent to W on the left side. The source is replaced by a point light and each view-sample shoots a single ray toward it. Inside the hard shadow, the illumination is still assumed to be zero (only an outer penumbra is added). Otherwise, the shortest distance x of the ray to the object is computed. b should be aD/A because $W \approx aD/(A-a)$. τ can then be set to x/b to achieve the wanted behavior.

Many approaches took inspiration from their shadow computation, to name a few: Chan and Durand [2003]; Mo et al. [2007]; Arvo et al. [2004].

Soft Planar Shadows Using Plateaus Haines [2001] avoids ray tracing by attaching outer penumbrae to shadow boundaries on a planar ground. He uses the same approximations, and the algorithm also uses the z-Buffer to blend different shadow contributions. The darker the shadow, the higher the constructed element, hence the name *plateau*. The camera is placed orthogonally to the planar receiver. Each projected vertex and silhouette are transformed in a 3D shape that, projected, delivers the quad for an edge and a circular approximation for a vertex with correct shading. A big problem is that the construction of these volumes is not easy and they can contain hyperbolic surfaces if, for example, two vertices adjacent to an edge are not at the same distance from the light. Shading in graphics hardware makes the solution somewhat obsolete and the z-buffer method can be replaced by a blending operation.

Smoothies Chan and Durand [2003] propose a related solution for arbitrary receivers exploiting newer graphics hardware. They make use of additional primitives, the *smoothies*, produced per frame and attached at each silhouette and vertex. These small quads are of constant size in the light's view and orthogonal to the light direction. Each such *fin* encodes the distance from its edge/vertex with colors. Rendered from the source this results in a *smoothy mask*. To deal with superposition of smoothies, a *min blending* is used to keep the shortest distance per pixel. The resulting mask can then be queried via simple texture lookups. In a second rendering step, a depth buffer for the smoothies is derived. In the third and final rendering step, each view-sample is then classified as in shadow (standard shadow map) or in "penumbra" (smoothy depth). For the latter, a soft transition is computed based on the distance stored in the smoothy mask. Using this distance directly would lead to constant sized penumbrae even when the object is close to the receiver. Therefore, the authors use the same ratio formula as in [Parker et al., 1998]. One problem is that this method does rely on a silhouette extraction step that can be costly. Further, all smoothies are rasterized twice and recreated for each step and need to be sent to the card. The algorithm also inherits the aliasing artifacts of shadow maps where a blocker is close to the receiver. Here, smooth transitions are too small to hide the aliased hard shadow.

Penumra Maps Wyman and Hansen [2003] make the good observation that, like for smoothies, only an outer penumbra is added, which means that the entire penumbra region is visible from the light. This allows us to rasterize the soft shadows directly into a texture that can then be applied just like a shadow map. The algorithm starts by filling the depth buffer and computing a depth map. In a second pass, the penumbra map is created. Each vertex on the silhouette is extruded to a cone corresponding to the spherical light source projected through the vertex. Silhouette edges are transformed to sheets connecting these cones tangentially, which is a similar construct as used for the accurate penumbra region determination.

While rasterizing these sheets in the light's view, a shadow intensity is derived and stored in the *penumbra map*. To compute this value, they rely on an ad-hoc formula that combines the current height of the rasterized geometry, the receiver point from the underlying depth map, and the distance of the occluding element from light (this one is passed via texture coordinates). The approach is not very fast because of the silhouette extraction and geometry production. It delivers overestimated shadows and uses a discretized map despite the geometry extraction.

5.3.2 Soft Shadow Volumes

There is a class of algorithms that are based on using *penumbra wedges* [Akenine-Möller and Assarsson, 2002; Assarsson and Akenine-Möller, 2004] of which most are improvements and extensions of the *Soft Shadow Volume* algorithm [Assarsson and Akenine-Möller, 2003]. The initial penumbra wedge approach suggests encapsulating the penumbra regions by geometrically creating a wedge from each silhouette edge, where the wedge encloses the penumbra cast by the corresponding edge (Figure 5.15). Shadow receiving points that lies inside the wedge, receive a shadow value solely based on interpolation between the inner and outer wedge border. Wedges are not allowed to overlap, or artifacts appear, which leads to several complicated wedge situations (Figure 5.16). The soft shadow volumes technique lifts this restriction by proposing a method to split the shadow contribution unambiguously between all edges. The penumbra wedges then only need to be conservative. A nice feature is that soft shadow volumes creates shadows without any discretization artifacts or undesirable discontinuities, which often leads to a visually pleasing result. Later approaches improves on the speed [Assarsson et al., 2003; Lengyel; Johnson et al., 2009], ameliorates the approximations [Forest et al., 2006] and also show how to get correct sample based results [Laine et al., 2005; Lehtinen et al., 2006; Forest et al., 2008]. We will now treat the different approaches in more detail.

5.3.2.1 Penumbra Wedges

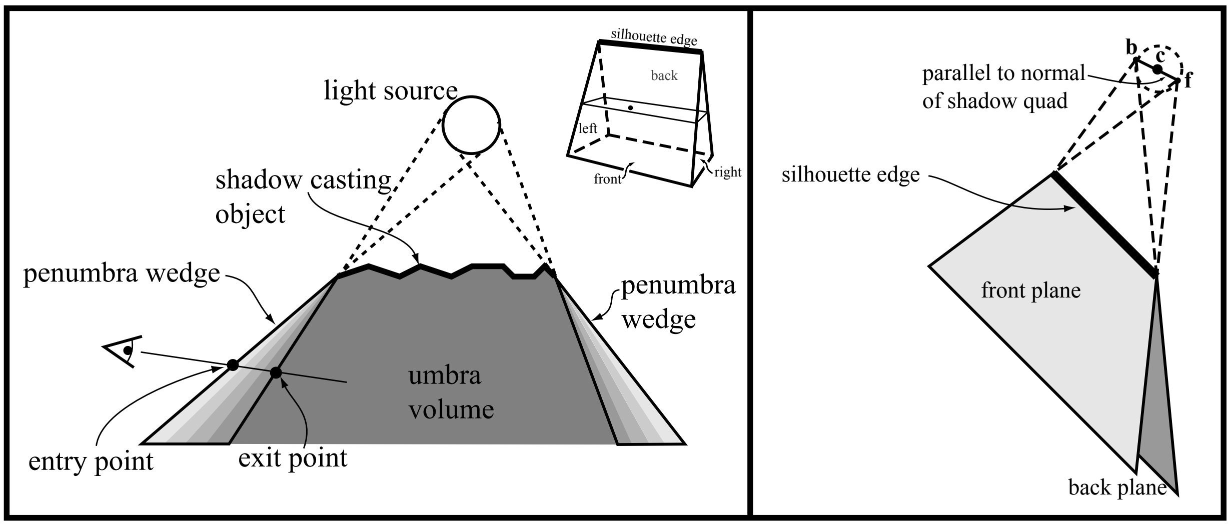


Figure 5.15 Penumbra Wedges are tangent to silhouette edges and the source.

Akenine-Möller and Assarsson [2002] launched geometry-based soft shadows into a new fundamental direction and presented a series of soft shadow papers that created a novel trend. The

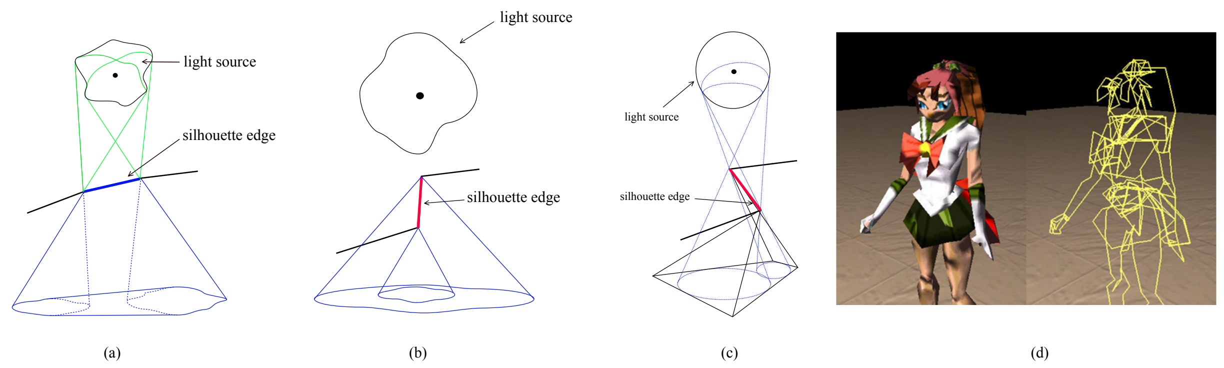


Figure 5.16 Problematic penumbra wedge situations. a) A standard non-problematic case. b) In b, the edge lies totally within the penumbra, cast by the edge itself, c) The edge is inside the penumbra cast by its adjacent (upper) neighboring edge, which makes a geometrical separation between the wedges very complicated. This illustrates the desire to separate the geometrical wedges from the penumbra computation and motivates the Soft Shadow Volume technique, described in Section 5.3.2.2. d) This illustrates that silhouettes for a real object can be complicated.

major idea was to adapt the shadow volume separation into umbra and lit regions for soft shadows. The observation was that light-silhouette edges should contain all the information that is needed to derive shadows. Therefore, they start by creating a hard shadow in the scene. Then, wherever a light silhouette is interacting with the shadow, the intial solution is overwritten. To compute the influence region of each light-silhouette edge (determined from the center of the source), the authors introduce *penumbra wedges* (figure 5.15). For each silhouette edge, two tangential points are found on the source that correspond to light-tangent planes containing the edge. The penumbra wedges are then defined by the shadow volumes created from the tangential points on the light. On the sides, two infinite triangles close these tangent faces to a volume. In [Akenine-Möller and Assarsson, 2002], a first algorithm exploiting this representation was introduced for cards with limited shader support. Consequently, the shadow computation was necessarily simple. The authors decided to linearly vary intensity inside of the penumbra wedges (not unlike [Wyman and Hansen, 2003]). To achieve a continuous behavior in this scenerio, care has to be taken for adjacent silhouette edges. The penumbra-wedge side triangles should be shared, and this results in a very involved construction that needs to distinguish many cases. Figure 5.16 depicts some of the problematic light source / edge configurations that could not easily be handled. The interested reader is referred to [Akenine-Möller and Assarsson, 2002] and its amelioration in [Assarsson and Akenine-Möller, 2004]. Because these historically important methods can be considered outdated due to their successors, we will not discuss them in more detail here.

5.3.2.2 Soft Shadow Volumes – Basic Approach

More important and still currently of relevance are [Assarsson and Akenine-Möller, 2003; Assarsson et al., 2003]. The principle is very similar. Each light silhouette gives rise to a penumbra wedge. The main difference is that they are now constructed independently for each light-

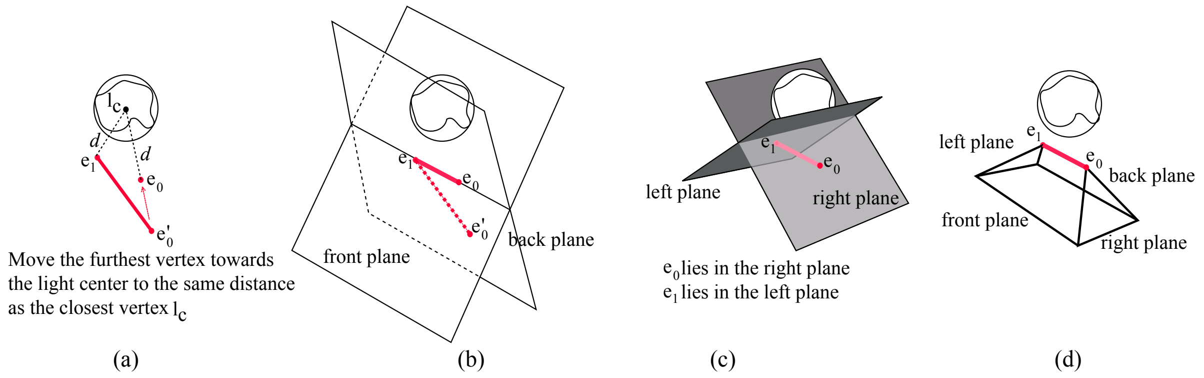

l_c
d
d
e_1
e_0
e'_0
Move the furthest vertex towards the light center to the same distance as the closest vertex l_c
(a)
e_1
e_0
e'_0
front plane
back plane
(b)
e_1
e_0
left plane
right plane
e_0 lies in the right plane
e_1 lies in the left plane
(c)
left plane
e_1
e_0
back plane
front plane
right plane
(d)

Figure 5.17 Wedge construction steps. a) Move the vertex furthest from the light center l_c towards l_c to the same distance as the other vertex. b) Create the front and back planes. c) Create the left and right planes. d) The final wedge is the volume inside the front, back, left, and right planes.

silhouette edge or adjacent edge (see Figure 5.17). For each penumbra wedge, shadows are no longer determined by interpolation; instead, a fragment program is executed on the contained view-samples. Each view-sample projects the silhouette onto the light source and computes its blocking contribution. This degradation of penumbra wedges to simple *markers* of the transitional shadow region allows us to simplify construction substantially because it can be very coarse, they are no longer used to directly interpolate shading. Many of the complex cases (e.g., (almost) alignment with the source—see Figure 5.16) can be circumvented by shifting the wedges' vertices.

Wedge construction The construction is as follows: both silhouette vertices are lifted to the same height (the nearest distance to the light), and the front and back quads are constructed as before. The sides, are based on tangential planes to the light, that contain the edge extremity and the cross product of the edge and the vector connecting the extremity with the light's center. The resulting volume is a conservative bound for the penumbra region generated by the light silhouette edge. Nevertheless, their determination is performed from the light's center, which is an approximation and can even result in temporal incoherence when an edge is suddenly becoming a silhouette (see Figure 5.19 left).

Computing Soft Shadow Value The next step is to compute the actual shadow intensity. This works almost like the approach for computing the area of a polygonal closed shape: choose an arbitrary reference point, e.g. the center of the light source, and sum up the *signed* areas of all triangles defined by edges and the reference point. The nice observation is that, as long as the edges have a consistent orientation, the final sum can be computed by treating the edges independently and accumulating the result. There are, nevertheless, some difficulties with this approach. We only want to compute the area inside the light source, and thus edges need to be clamped. This proves rather costly when using the area described by triangles as above. It becomes much simpler if, instead, one sums the opposites, meaning the sector (described by the two edge extremities and the center) minus the triangle formed by the edge itself and the

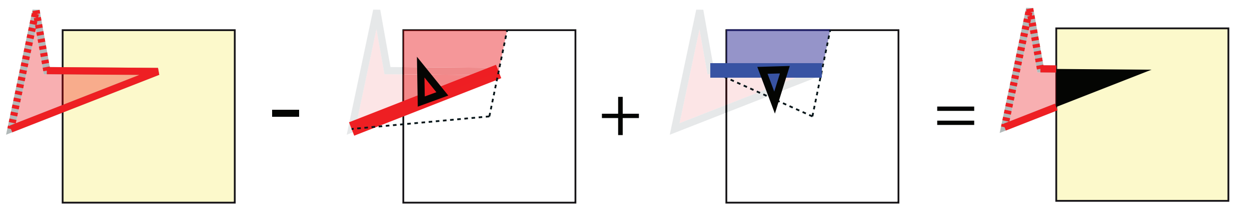


Figure 5.18 Computing the blocked area for an occluder at a receiver point. Each silhouette edge is treated separately. The blocking contribution of an edge is the intersection of the source with the sector described with the center of the source and excluding the triangle described by the edge and the center. Depending on the edge's orientation, the area is subtracted or added. This leads to an integration of the surface area once all source intersecting edges are processed. This is actually an example of using Green's formula in a plane to compute the area by integrating over a curve. The small triangles shows at which side of the edge the interior of the triangle is located, which determines the sign of the contribution. The integration constant is measured using shadow volumes.

reference point. (Figure 5.18 shows an example of the process). The solution works because the infinite area is clamped to the source resulting in a finite value. The edge orientation is chosen according to the view-sample and the light's center. For this, each penumbra wedge is virtually divided into an inner- our outer half-wedge by a hard shadow quad created from the light's center. This classification is equivalent to whether the center sample is in the "blocked" halfspace or not. The orientation of the edge is thus chosen accordingly to either include or exclude this sample. It is also necessary to add-in the umbra region not touched by the penumbra wedges. To compute the covered light area efficiently, a 4D texture can be derived in a precomputation; 4D because it is queried by a pair of 2D endpoints of the projected segments. Then the blocking contribution of each edge boils down to a single lookup. This is especially efficient for textured sources.

Problems Some shortcomings of this solution are that occluders are necessarily additively combined, otherwise an expensive method is needed where the shadow information for each occluder is derived separately [Assarsson et al., 2003] (see Figure 5.19 right). Unfortunately, this can lead to a strong umbra overestimation. On the other hand, for non-overlapping silhouettes the method derives an accurate solution. The main reason it has not yet been used in practice is the cost of the approach. It inherits the deficits of shadow volumes: costly silhouette determination with corresponding wedge creation and strong overdraw.

5.3.2.3 Improvements

Several improvements have been presented since the first soft shadow volume algorithm, both in terms of speed and quality.

Lengyel presents a way to use orientation and z-tests to optimize the penumbra-wedge rendering. He also passes the plane equations for the inner, outer, and hard shadow plane into the fragment shader to classify the samples based on their plane distances.

The quality of the shadows can easily be improved by splitting the light source into smaller patches and using the algorithm for each light patch [Assarsson et al., 2003]. This is somewhat similar to an approximation via several point lights, but here using small patches. By splitting

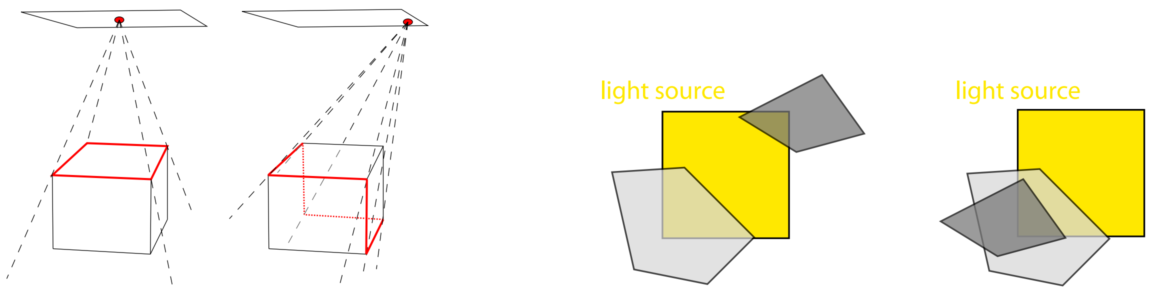


Figure 5.19 **Approximations** *Left*: When finding the silhouette edges, the silhouette is assumed as seen from the center of the light source, which is an approximation, since the true silhouette varies over the light source. *Right*: The original Soft Shadow Volume algorithm ignores correct treating of overlap between different occluders (i.e., incorrect *occluder fusion*), since each silhouette contributes with a constant amount, disregarding which part it occludes. Thus, the left and right situation will incorrectly produce the same amount of shadow. This is solved in more recent soft shadow volume algorithms.

the light source into 2×2 or 3×3 patches, both the artifacts from using only the silhouette edges as seen from the center of the light patch, and ignoring correct treating of overlapping occluders, are significantly improved. However, the execution time goes up rapidly.

Forest et al. [2006] do provide an amelioration over the several passes (for each region of the light) usually needed to improve the algorithm. They break the light virtually into four regions and compute the blocking contributions during a single pass for all four regions (the key is that the silhouettes are detected for the common corner of all four light regions). Further, they keep track of already-created blocker surfaces by maintaining a bounding box. Whenever a new silhouette is added, the overlap of the bounding boxes is tested and the shadow contribution is decreased by a relative amount, according to the detected intersection. Of course, this is a coarse heuristic, but results in better-looking, lighter shadows.

Depth Complexity Sampling Soft shadow volumes have also been used successfully in the context of offline rendering using ray tracing. Laine et al. [2005] use a map of counters instead of only an occlusion value (see Figure 5.20). After accumulation, the result indicates, up to some offset, how many surfaces are intersected by a ray from the view-sample to each light sample. As a result, only one final ray needs to be shot per view-sample to derive the correct values for all light samples. Mathematically, this corresponds to finding the integration constant, which is still ambiguous after just having done the covered region integration from the silhouette edges. To illustrate the necessity of this last ray, imagine that not a single penumbra wedge interacted with a view-sample. This does not necessarily mean that the sample cannot lie in shadow; for instance, it can lie deep in the umbra region (see Figure 5.20 f). To find silhouette edges related to view-samples, they use a hierarchical hemicube-like structure. Then each view-sample finds potential silhouette edges from this structure and then tests which ones of these are actual silhouettes that will be integrated in the process.

The work by Lehtinen et al. [2006] improved upon [Laine et al., 2005], taking advantage of the fact that locally clamping penumbra wedges based on the adjacent triangles can significantly reduce their size (see Figure 5.21). Futher, they use a three-dimensional kd-tree to recover the

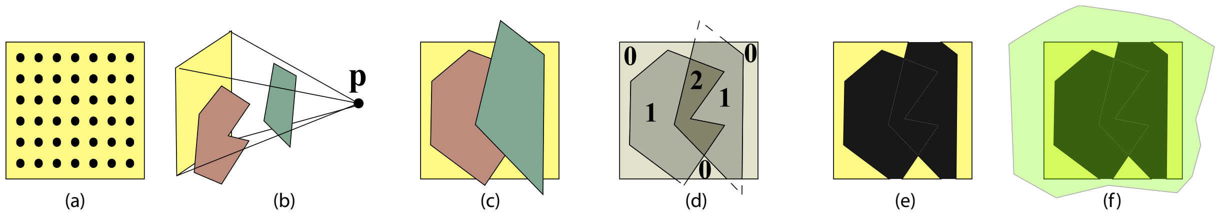


Figure 5.20 Illustration of the depth complexity function. (a) Each sample point on the light source (typically 256-1024 samples) is represented with a depth counter. (b) Two occluders are located between point p and the light source. (c) The occluders projected on the light source from p. (d) The depth complexity function tells the number of surfaces that overlap a point on the light source. During the integration, when adding the contribution from each silhouette edge, the affected depth counters are increased or decreased accordingly (e) The visibility function is reconstructed from the depth complexity function. (f) Finally, a ray is shot towards any unoccluded sample, to determine whether the light source happens to be fully covered by any object whose silhouette edges does not intersect the light source.

potential light-silhouette edges that are also silhouette edges for the view-sample in question. The point of using a 3D kd-tree is that, compared to the original 2D-grid used by Laine et al. [2005], when questioning the data structure for potential silhouette edges from a certain point in space, the returned set of edges is significantly less conservative, which both reduces memory usage and improves speed.

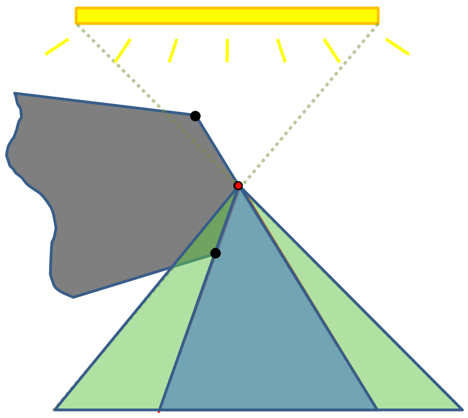


Figure 5.21 The green region is the wedge from standard creation [Laine et al., 2005] . The wedge can, however, be cropped by the planes of the adjacent triangle planes, since the silhouette (red dot) is only a silhouette for points in the blue region [Lehtinen et al., 2006].

Depth Complexity Sampling for Real-Time Rendering In Forest et al.'s [2008] most recent work, they present an algorithm that can be seen as an adaptation of [Laine et al., 2005] for the GPU. No hiearchical structures are used, instead penumbra wedges are directly involved to determine the view-samples with which the light-silhouette edges interact. The counters are packed into bits of (currently) floating point numbers, allowing to maintain several ones in a single variable. Nevertheless, care has to be taken to ensure that there is no overflow because it

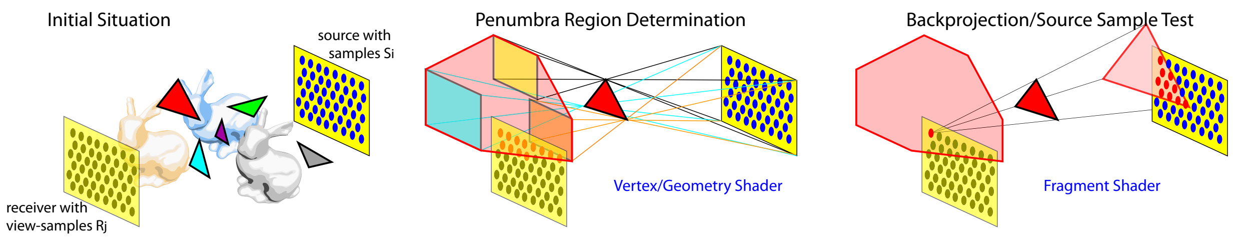

Initial Situation
source with samples Si
receiver with view-samples Rj
Penumbra Region Determination
Vertex/Geometry Shader
Backprojection/Source Sample Test
Fragment Shader

Figure 5.22 Eisemann and Décoret [2007] consider a sampled planar source patch, a sampled planar receiver patch and a set of triangular occluders placed between them (left). For each triangle separately, they produce a conservative penumbra approximation on the receiver plane (b). For each view-sample inside this estimated region, the triangle is backprojected onto the source plane and all source samples are tested against it (c). The occluded samples are then added to the blocked-sample set stored in the view-sample.

would pollute the neighboring counters. Lookup tables are used to evaluate the result, but imprecisions can lead to stronger artifacts in this case. The solution achieves interactive performance if the number of light samples is low (typically 16) because of the necessity to maintain counters. Instead of tracing a reference ray, in order to find the integration constant [Laine et al., 2005], Forrest et al. use shadow volumes, like the original Soft Shadow Volume algorithm [Assarsson and Akenine-Möller, 2003]. This is however not a fully robust approach, but more care could be achieved by using shadow volumes from more than just one sample position on the light source, and using voting for for finding the correct soft shadow value.

5.3.3 View-Sample Mapping

Rasterizing in Light Space One of the major problems with soft shadow volumes for reaching high frame rates, together with the fairly expensive fragment shader, is the fact that soft shadow volumes has the same tendency to produce a huge amount of fill due to large elongated polygons as for shadow volumes. One way to avoid this is to do the rendering of the wedges in light space instead of screen space. Thus, each wedge's influence region simply becomes a polygon corresponding to the bottom face of the wedge, which typically results in a significantly smaller rasterized area. Two algorithms that utilize this idea are the ones by Sintorn et al. [2008] and Johnson et al. [2009]. But before we describe these methods, we will explain the concept of view sample mapping, which is used by these two and more shadow algorithms. The basic concept is to store view samples in a light space map, and perform the shadow computations for these sample points with respect to one or many light samples. We have already briefly encountered this for hard shadows, in Section 3.2.5.

5.3.3.1 Theory

View-sample mapping methods compute accurate visibility relationships between a set of source samples $\mathbf{s}_i$, $1 \leq i \leq s$, placed on the light, and a set of view-samples $\mathbf{r}_j$, $1 \leq j \leq r$, (points for which we want to determine the shadow) in the presence of occluding triangles T_k, $1 \leq k \leq t$. The situation is illustrated in Figure 5.22 (left).

Such methods derive a discretized solution of the initial shadow definition in Equation 1.1; for each $\mathbf{r}_j$, the set of source samples not visible from view-sample $\mathbf{r}_j$ are computed:

$$\mathcal{B}_j := \{\mathbf{s}_i \mid \exists k\ [\mathbf{s}_i, \mathbf{r}_j] \cap T_k \neq \emptyset\}. \tag{5.1}$$

One thing to notice is that if the source is uniformly sampled, the ratio of the set's cardinality $|\mathcal{B}_j|$ and the total number of light samples s gives an approximation of the blocked light as seen from sample $\mathbf{r}_j$ and hence of the visibility integral from Equation 1.4. In fact, it should be pointed out that the set in Equation 5.1 also contains all information necessary to sample Equation 1.3 to produce truly physically-based shadows.

Despite this major precision benefit, computing this set is challenging. Fundamentally, computing $\mathcal{B}_j$ requires a triple loop with an inner intersection test, which can be expressed as:

$$\forall i \quad \forall j \quad \forall k \quad [\mathbf{s}_i, \mathbf{r}_j] \cap T_k \overset{?}{\neq} \emptyset \tag{5.2}$$

The commutativity of the "for each" operators results in six different ways of organizing the computations. In fact, due to the reciprocity of view- and source samples, only three major algorithmic choices arise. For $\mathbf{s}_i$ and $\mathbf{r}_j$ fixed, testing intersections against triangles is basically equivalent to ray-tracing. For $\mathbf{s}_i$ and T_k fixed, finding the $\mathbf{r}_j$s that pass the intersection test is very similar to the idea of creating a shadow map from $\mathbf{s}_i$ with all triangles T_k. Testing the shadow status of $\mathbf{r}_j$ then remounts to performing a shadow map test. The same computation structure was at the basis of the algorithm in [Heckbert and Herf, 1997]. An off-center perspective view of the occluders is rendered from each source sample, to obtain a black and white occlusion mask. These views are accumulated to obtain an image that represents exactly $|\mathcal{B}_j|$. In fact, by drawing a shaded occlusion mask, it is further possible to sample the physically-based integral in Equation 1.3.

Even though highly precise, this approach has some limitations. First, it only computes the cardinal numbers of the sets, not the sets themselves and is thus limited to visibility-only integrals like in Equation 1.4. Second, it requires as many renderings as there are source samples. For 1024 samples, this can drastically impact performance. Third, all view-samples $\mathbf{r}_j$ are located on a receiver plane.

In the following, we will show how view-sample mapping algorithms can evaluate visibility by performing a single rendering pass. The principle is to compute all blocking contributions of a single triangle while it is processed by graphics hardware. This corresponds to the third algorithmic permutation derivable from Equation 5.2.

5.3.3.2 Triangle-Based Approach

Basic principle To facilitate explanations of this method, we will first focus on the particular case of a *receiver plane*, that contains all samples $\mathbf{r}_j$ arranged on a regular grid (hence the name *View-Sample Mapping*) and a *source plane*, that contains all samples $\mathbf{s}_i$. We will explain later in this section, how this assumption was removed in [Sintorn et al., 2008] to allow a computation for a general placement of view-samples, hence arbitrary scenes.

Laine and Aila [2005] popularized the idea that it can be efficient to perform all computations involving a particular triangle while it is at hand instead of traversing the geometry several times. Their suggestion was to work on the Cartesian product between the set of light and view-samples, resulting in all possible combinations between elements of the two sets. Each entity in this set can then be associated to a segment running from a source- to a view-sample. In their approach, they successively cull elements from this set that correspond to segments that would intersect the triangle at hand. After having looped over all triangles, the remaining elements describe the visibility relationships.

Even though their algorithm did not aim at real-time and would be hard to realize on graphics hardware, it laid important groundwork for more adapted GPU computations [Eisemann and Décoret, 2007; Sintorn et al., 2008]. Eisemann and Décoret [2007] proposed the following algorithm:

1. Traverse all triangles.
2. Traverse all view-samples that are potentially affected by the current triangle.
3. Find source samples that are hidden by the current triangle from the current receiver sample.
4. Accumulate result with the one determined by previously treated triangles.

This approach requires a *single* rendering pass and can be implemented using vertex, geometry, and fragment shaders. An overview of the steps is illustrated in Figure 5.22.

The viewport of rendering is chosen to coincide with the planar receiver patch. Hence, the receiver samples correspond to pixels on the screen and the view-samples are *mapped* into a texture. Each of these pixels will be used to store the blocked sample list of the corresponding view-sample. The algorithm proceeds by treating each triangle separately. First, the triangle is projected in this view, such as to cover all view-samples whose visibility of the source is potentially affected by this triangle. For each of these view-samples the actual set of source samples that are blocked by this triangle is derived. This result is encoded as a bit-pattern in the form of a color, and the blending capacities of the hardware allow to update the current set of the blocked source samples. Proceeding in this way, once all triangles are treated, the correct blocked sample set is represented in form of a color at each view-sample's position.

Determining Potentially Affected View Samples For a single triangle, the region where occlusion can occur contains those view-samples for which the triangle is hiding a part of the source. This region can be equivalently defined as the union of all projections of the triangle from the source samples on the receiver plane or as the shadow produced by the source and the triangle. To compute the influence region of a triangle, one can use the penumbra wedges, that were introduced in Section 3. It is enough to compute the intersection of the three penumbra wedges with the receiver plane and then compute the convex hull of this intersection. An efficient CUDA implementation for this convex hull is described in [Sintorn et al., 2008]. Since blocked source samples will be determined in a second pass, it is enough to find a conservative penumbra estimate. Therefore, one simpler choice is to compute an axis-aligned bounding quadrilateral [Eisemann and Décoret, 2007]. In the particular case of a planar receiver, there is a special reason why a bounding quad is a good idea: It enables an efficient interpolation of values from the vertices of this quad and therefore avoids some of the costly computations in the second part of the algorithm, where the blocked source samples are computed. Such a solution would be more expensive if many corners were present.

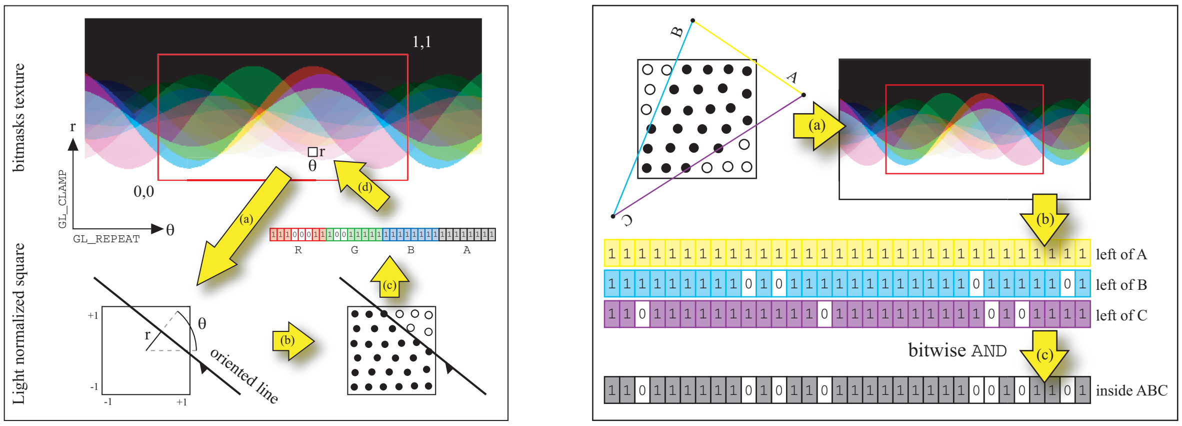

bitmasks texture
Light normalized square
1,1
0,0
r
θ
GL_CLAMP
GL_REPEAT
(a)
(b)
(c)
(d)
R
G
B
A
oriented line
+1
-1
A
B
C
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 left of A
1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 left of B
1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 left of C
bitwise AND
1 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 inside ABC

Figure 5.23 Overview of the source sample evaluation - Left: Pre-processing the left-sample map for a set of 32 samples. For every texel (θ, r), the line corresponding to this Hough space value is built (a) and samples located on the left of it are determined (b). These samples are then encoded as a bitmask (c) in form of the color of the texel (d). The sine wave in the resulting textures corresponds to the Hough dual of the sample points. Right: To find the light samples within the backprojection of a triangle (a), one can use the bitmask texture. Looking up bitmasks for each edge (b) and combining them using a bitwise AND (c) results in the contained samples.

Backprojection on a Planar Light Source For each potentially affected view-sample, one needs to find which source samples are hidden by the triangle. In a first step, the triangle is backprojected onto the source. As explained below, each source sample is subsequently tested for containment in the triangle's projection. Note that the backprojection matrix depends on the view-sample and can therefore not be passed as a uniform. Hence, in general, the projection matrix needs to be built in the fragment shader, but it is worth mentioning that in the particular case of a planar receiver a more efficient computation is possible because more values can be interpolated across view-samples [Eisemann and Décoret, 2007].

To enable a simpler computation of the blocked source samples in the next step, the backprojection matrix is defined such that the coordinates of the backprojected triangle are in a *normalized source frame* where the source is a square $x = \pm 0.5$, $y = \pm 0.5$, $z = 0$. This naturally allows to handle rectangular sources, by reverting their stretch in the projection matrices.

At this stage, fragments corresponding to view-samples inside the triangle's penumbra have computed the coordinates of the triangle's backprojection. The next step is to find the source samples inside the backprojected triangle.

Determining Blocked Source Samples The first idea is to store the set of blocked source samples as a color. To make this possible, each source sample is associated to one bit of the output color. Typically, for RGBA8 this means that 32 source samples can be treated (but on today's hardware using multiple render targets and integer textures, 1024 samples are possible). A source is blocked if the corresponding bit is one, otherwise it is visible for the current view-sample.

One solution to produce the blocked sample set is to loop over all source samples and perform a containment test against the backprojected triangle. Depending on the outcome, the bits are set accordingly. Though conceptually simple, it is highly inefficient for a general triangle. Instead a precomputation allows a parallel execution of this test for multiple samples.

The set of samples inside a 2D triangle is the intersection of the set of samples on the left of the supporting line of each (oriented) edge. This allows to devise a better method based on a precomputed texture.

An oriented line in the 2D plane of the normalized source frame can be represented by its Hough transform [Duda and Hart, 1972], that is an angle $\theta \in [-\pi, \pi]$ and a distance r to the origin. The distance can be negative because the lines are oriented. However, only those lines intersecting the normalized light square are needed. In consequence, it is possible to restrict the domain to $r \in [-\sqrt{2}/2, +\sqrt{2}/2]$. In a preprocess, the Hough space is then sampled in a 2D texture called the *left-sample map*. For every texel (θ, r), the samples located on the left of the corresponding line are found and encode as a color (Fig. 5.23, left).

At runtime, the fragment shader evaluates which samples lie within the backprojection as follows. First, one ensures that the backprojection is in counter-clockwise orientation, reversing its vertices if not. Then the lines supporting the three edges of the backprojection are transformed into their Hough coordinates, normalized so that $[-\pi, \pi] \times [-\sqrt{2}/2, +\sqrt{2}/2]$ maps to $[0, 1] \times [0, 1]$. These coordinates are used to look-up three bitmasks (one for each edge). Each bitmask encodes the samples that lie on the left of each edge. The Hough dual is periodic on θ. Thus the correct wrapping modes are `GL_REPEAT` for θ and `GL_CLAMP` for r. The latter correctly results in either none or all samples for lines not intersecting the normalized light square. These three bitmasks are AND-ed together. This logical operation is available on Direct3D-10-class graphics cards. The result is a single bitmask representing the samples inside the backprojection, which is then output as the fragment's color (Fig. 5.23, right).

Combining the Occlusion of All Triangles The set output by the fragment shader for a receiver sample $\mathbf{r}_j$ and a triangle T_k can be formally written as:

$$\mathcal{B}_{j,k} \equiv \{ i \mid [\mathbf{s}_i, \mathbf{r}_j] \cap T_k \neq \emptyset \} \tag{5.3}$$

It can be seen that $\mathcal{B}_j = \bigcup_k \mathcal{B}_{j,k}$. In other words, the blocked samples can be computed for each triangle separately and the resulting sets only need to be combined. Performing this union is possible using graphics hardware again. Because the blocked samples are stored as a bitmask, indicating with ones the source samples that were blocked, a union between two such sets can be realized by OR-ing the bitmasks. To perform this operation on the graphics card, one can rely on a *logical operation* blending mode. Note that this mode is currently only exposed in OpenGL but not in recent versions of Direct3D. Setting the blending to OR will ensure that the previously stored set of blocked samples is combined with the incoming set of blocked samples for the current triangle. Once all triangles were processed, the bitmask stored in the view-sample then reflects the samples blocked by the geometry of the entire scene.

Light Source One feature of precomputing the left-sample maps is that the source samples can be arranged in an arbitrary way without performance penalty. This offers an interesting option introduced in [Eisemann and Décoret, 2007]. One can choose from a set of precomputed source sample textures depending on the current view-sample. This increases quality if particular alignments are present in the scene or for very large sources. Also, the method allows to work with colored sources by clustering the samples in groups of constant colors that are passed as uniform values.

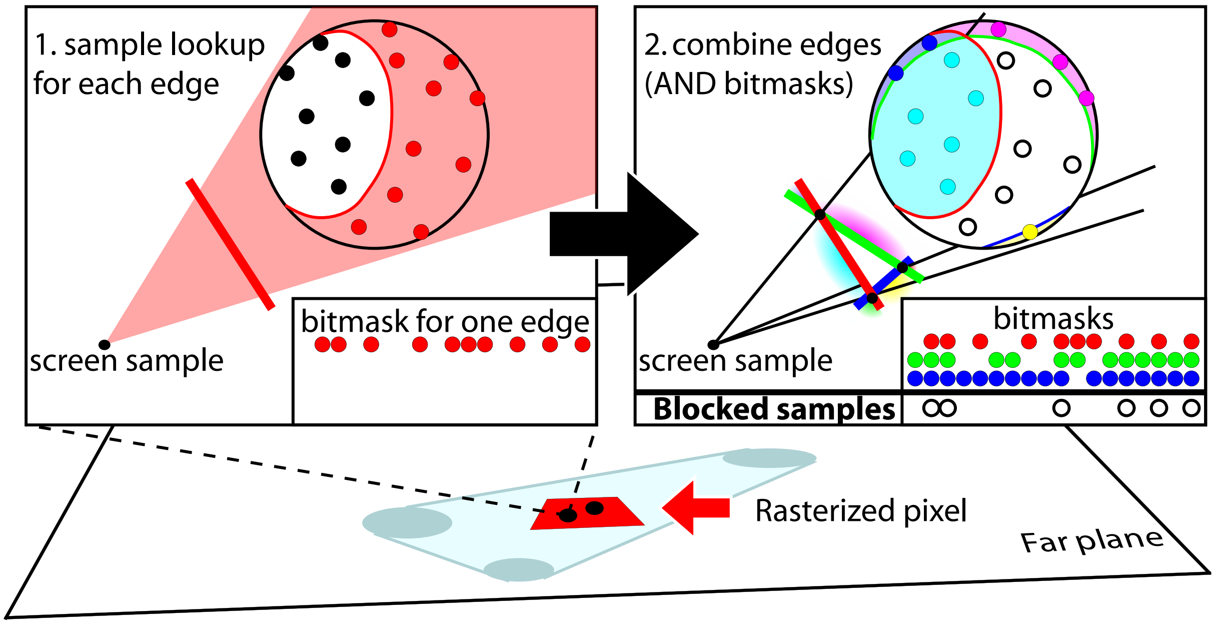


Figure 5.24 Sintorn et al. [2008] introduced a fast sample derivation technique for volumetric sources. They further deal with general scenes by projecting view-samples onto the far-plane of the light's frustum, where they are stored in per-pixel lists (in the figure, two view-samples (black) share the same pixel (red)). Details on these lists were given in Section 3

To work with volumetric sources, Sintorn et al. [2008] mentioned a different way of evaluating the shadows. Instead of a backprojection, they construct the frustum defined by the triangle's edges and the view-sample. The resulting plane equations can also be transformed into a 3D Hough Space (resulting in two spherical angles describing the normal of the plane, and its distance to the light source). The left-sample map then corresponds to a 3D texture encoding the samples lying on the same side of the plane as the triangle. This is illustrated in Figure 5.24. It also relates to [Kautz et al., 2004], where the light source is assumed to be an environment map and hence only the plane normal is used to lookup the blocked samples.

Receiver To lift the restriction of a planar receiver, Sintorn et al. [2008] proposed an efficient algorithm to store a lists of view-samples in each pixel. We have already reviewed this technique in the context of hard shadows in Section 3 and the same principle can be applied for soft shadow rendering too.

Optimizations In general scenes, shadows only need to be computed for light-facing view-samples, but further, if the caster is watertight, i.e., it encloses a volume, one can perform a front-face culling. In consequence, every triangle that is front-facing for all points on the light source can be ignored as a caster. Indeed, a ray blocked by such a triangle is necessarily blocked by a non-front-facing one (it must enter and exit the volume). Culling the front faces is more interesting because it yields smaller influence regions. In the geometry shader, triangles are tested by checking if it is front-facing for the four corners of the light source. If yes, it is safe to let the geometry shader discard it. This optimization eliminates roughly half the faces, and thus doubles the framerate.

Implementation on Direct3D-9-Class Cards

The algorithm for planar receivers (and even height field surfaces) can be implemented also on older hardware, lacking a geometry shader [Eisemann, 2008]. The solution is to create a *shadow mesh* from the original triangle mesh. For this structure, each triangle is replaced by a quad, storing in its texture coordinates the three vertex positions of the original triangle. The position-information of each vertex indicates what corner of the bounding quad it is supposed to represent (there are four entries in the position vector; setting the corresponding component to one allows to define what corner the vertex will correspond to). With this representation it is possible to determine the bounding quad of the triangle in the vertex shader using the texture coordinates, and to select the according corner. The other obstacle is the sample evaluation. Bitwise operations are not supported and one needs to resort to texture-based evaluations to encode the AND operation. Performing a bitwise logical AND between three values in the fragment shader requires integer arithmetic. However, if not available, it can be emulated [Eisemann and Décoret, 2006a] using a precomputed 256^3 texture such that:

$$\mathrm{opMap}[i, j, k] = i \text{ AND } j \text{ AND } k \tag{5.4}$$

Then, to perform the AND between three RGBA8 values obtained from the left-sample map, a 3D texture lookup is sufficient. This approach is general: any logical operation can be evaluated using the appropriate opMap texture. Further, 32-bit textures do not support blending, but this restriction can be overcome by using multiple render passes and computing 128 samples (4 color channels each with 8 bits and 4 MRTs) per pass.

5.3.3.3 Silhouette-Based Approach

Soft Irregular Shadow Mapping Johnson et al. [2009] present a combination of image-based and geometry based methods. They essentially use an irregular shadow map to compute the umbra contribution. Then, for each silhouette edge (as seen from the center of the light source), a penumbra wedge is created, which is projected, in light space, onto a representation of the screen space samples that we want to compute shadows for. This representation is a light-view spatial acceleration structure in the form of a 3D-grid shaped as the light frustum. The cells of the grid contains the screen-space samples. For each projected wedge, the affected screen-space samples are tested for penumbral occlusion against this silhouette edge, like previous real-time soft shadow volume algorithms. One difference, though, is that they note that doing an inverse cosine function call is faster than using a lookup table of inverse tangent values [Assarsson et al., 2003]. Johnson's method is highly adapted for the upcoming Larrabee platform and real-time performance is expected for game-scenes ($\sim$ 30 fps is reported).

On a high level, the algorithm by Johnson et al. [2009] and the one by Sintorn et al. [2008] have several similarities in that they both utilize light space and an irregular z-buffer for computing umbra. By moving this to light space, the algorithms comes closer to shadow maps, which normally has a considerable lower fill-rate demand. One major difference is that Sintorn et al. use the triangles to compute visibility per pixel whereas Johnson et al. use the silhouette edges.

5.3.4 Trade-offs for Geometry-Based Approaches

When selecting an appropriate geometry-based soft shadow algorithm, there are several choices to consider.

Triangles or Silhouette Edges Visibility can be determined from either all shadow casting triangles (see [Sintorn et al., 2008] and Section 5.3.3), or from all silhouette edges plus shooting a shadow ray or using shadow volumes to determine the integration constant (i.e., determining possible occlusion by an object whose silhouette edges does not intersect the light source and thus yet are unaccounted for, see Figure 5.20 f). Using silhouette edges has the advantage that they typically are much fewer in numbers than all scene-triangles. The average number of silhouette edges in a mesh of n triangles is around $n^{0.8}$ [McGuire, 2004]. The disadvantage is that probing for the integration constant is prone to robustness problems, whereas computing the visibility from the triangles is not. The shadow ray has to be shot towards a light sample that is some epsilon distance away from all edges that are projected onto the light source [Laine et al., 2005]. Numerical imprecision may cause an edge to end up on the wrong side of the sample after projection. This is particularly bad, since the outcome of the integration constant affects all sample values for a pixel which typically results in a totally black or white shadow value at the wrong place. Nevertheless, careful treatment of this problem results in a fully robust algorithm [Laine et al., 2005].

The same rule applies when using shadow volumes to sample the depth and selecting a suitable light sample. If the true silhouette edges are used for each shadow receiving point, then it is most likely hard to find a light sample that lies at least a distance epsilon from all silhouette edges for all shadow receiving points. Instead, shadow volumes could be created for e.g. three samples and voting could be used to select the integration constant. We believe this is enough to lower the error-rate to less than one faulty shadow value per frame, but it has yet not been tried, to the best of our knowledge.

Number of Light Samples Choosing only one light sample obviously provides the cheapest fragment shader, but also has the advantage of providing smooth penumbra without undesirable discontinuities. The problem lies in the occluder fusion where the shadow contribution from overlapping occluders are simply added. This can easily result in too dark penumbra regions.

Using four or more light samples is more expensive, but drastically improves the quality, and often becomes indistinguishable from the ground truth.

The techniques reviewed in Section 5.3.3 typically use a huge number (256-1024) of light samples, but only compute a binary visibility value per sample, in contrast to computing a smooth value in the range [0,1] per sample. Thus, the number of samples typically needs to be much higher than 256 in order to avoid visible discretization artifacts. Storing the intermediate result for all light samples per pixel during rendering comes with a higher memory cost. On the other hand, fully correct sample based visibility values are produced.

Light Space Rasterization of Wedges Rasterizing the wedges or triangles' shadow footprint in light space instead of eye-space reduces the number of rasterized fragments and significantly reduces the amount of generated fill. On the other hand, currently it comes with the disadvantage of more complicated rasterization algorithms.

5.4 Summary

Soft shadows are more pleasing to the eye than hard shadows. They increase realism in scenes but are much more difficult to compute. The main directions in the field of geometry-based approaches include soft shadow volumes, which create shadow-volume-like primitives bounding the penumbra region with wedges. This delivers very beautiful and alias-free shadows. On the other hand, a problem is that incorrect occluder fusion can result in disturbing artifacts. While later approaches deal with this problem, it requires more computations, which lowers the frame rate. The cost of creating shadow volume primitives makes the methods rather expensive and, even on the latest hardware, acceptable run-times are difficult to achieve even for average-sized scenes.

Image-based solutions, on the other hand, provide fast feedback. The varying interpretations of the shadow map allow us to eliminate many of the aliasing artifacts. The shadows look good for smaller sources, but can become less convincing for larger lights because in many cases only a single depth map is used. One such depth layer contains insufficient information about the scene. One possibility is depth peeling, but this implies that one pass is needed per layer, which can quickly outweigh the performance advantage of using image-based approaches.

View sample mapping computes visibility between a set of view samples and a set of light samples, and it is part of more recent soft shadow algorithms. Since the soft shadows are sample based, the number of light samples has to be high to avoid visible sampling artifacts. An advantage is that the method can compute visibility between one view sample and several light samples for a potentially blocking triangle very fast using lookup tables. Nevertheless, the overall costs of current algorithms are still high, and acceptable frame rates are only achieved for moderate scene-complexity.

6 Environmental Lighting

In this tutorial, we have seen many methods that aim at the computation of shadows for a given light source. With the increase of computational power, an increasing amount of light sources will be added to the scenes. Step by step, this will lead to the simulation of an entire environment that participates in the illumination computation. Ultimately, global illumination, where basically each element in the scene is considered a light source, will be simulated with all-frequency information. Until then, we still have a long way to go, but it is interesting to already take a short peek at scenarios in which light sources are no longer well-defined entities with limited spatial extent in the scene, especially as there are very approximate solutions that can be used to enrich the appearance of the final rendering. Figure 6.1 shows an example for the influence of the different types of illumination on the appearance.

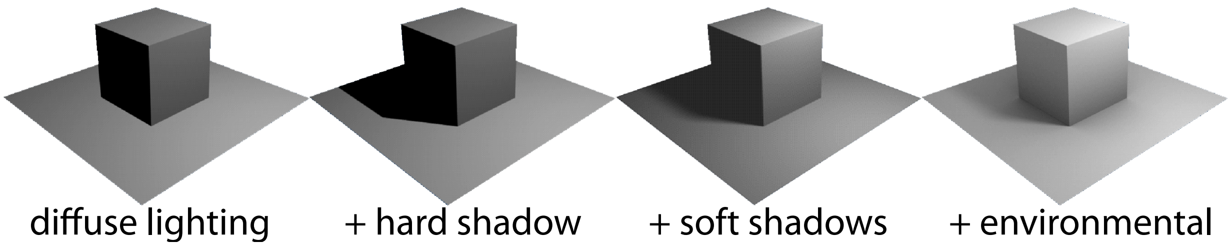


Figure 6.1 Diffuse illumination is the standard way of lighting a scene (left), hard shadows add spatial cues (middle left), but soft shadows make the scene look much more realistic (middle right). Adding environmental illumination further enriches the appearance (right).

6.1 Environment Map Shadows

A first step towards connecting soft shadow illumination with environmental lighting was presented in [Annen et al., 2008a] (Convolution Soft Shadow Maps CSSM). We already saw in Section 5.2.2.3 that this method was used to quickly compute soft shadows. The authors further show that the approach is suitable for environmental illumination by decomposing an environment map into a set of several area light sources. Just similarly to the idea of using several point sources to approximate an area light (as we discussed in Section 1.3), one can decompose an environment map on point or area lights. Such an environment map decomposition is not limited to this particular algorithm, in fact, any soft shadow algorithm could be extended to an environmental approach, just like any hard shadow algorithm could. To use area lights makes sense for larger bright features, whereas point lights are better for small light features. Nevertheless, a multi-light evaluation always implies that several passes have to be performed, one for each light.

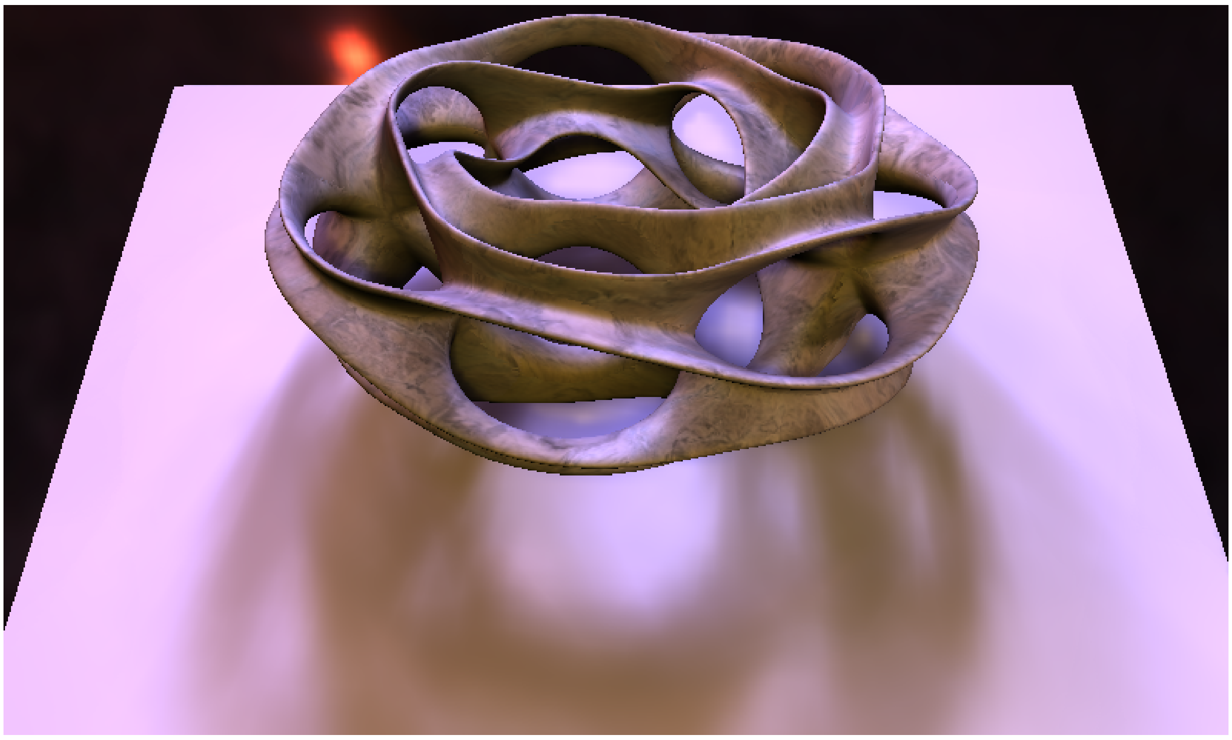


Figure 6.2 Convolution Soft Shadow Maps are relatively efficient to evaluate. This makes them well-suited when an environment map is to be approximated by several area lights.

6.1.1 Basis Functions for Low-Frequency Lighting

In order to avoid multiple soft shadow evaluations, one can borrow techniques from precomputed radiance transfer solutions (e.g., [Sloan et al., 2002]). The assumption for such approaches is that the scene is lit by large low-frequency light sources—such as a less detailed environment map. Here, the incident radiance as well as light visibility are expressed by weighted basis functions, similar to the Fourier expansion we have encountered in the context of convolution shadow maps [Annen et al., 2007] (Section 4.2.3). To represent directional functions, such as the incident radiance and the light visibility, spherical harmonics (SH) are a good choice, which are basically an extension of Fourier decompositions on a sphere. Of course, whenever a function is expressed in a finite set of such basis functions, certain approximations are necessary. In the case of SH, it is not possible to represent a high-frequency signal.

Unfortunately, visibility is a high-frequency signal due to its binary nature and a projection into the SH basis incurs an approximation error and causes smoothing. SH do allow a good approximation for low frequency illumination only. Fortunately, this is often the case for environmental light sources and makes such methods well adapted in this context.

Spherical Harmonics-based Approaches

Mainly targeting articulated characters, Ren et al. [2006] construct a hierarchical sphere set approximation for each scene object in a preprocess. During runtime, visibility is determined by accumulating the occlusion due to the spheres. Since each sphere covers a circular solid angle,

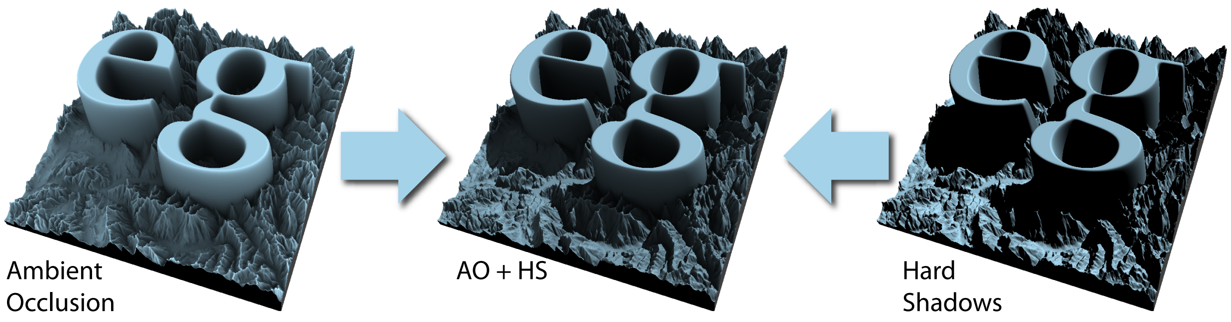


Figure 6.3 Ambient occlusion is a great way to complement direct illumination and shadows. For height fields the computation time for ambient occlusion is much faster than for general geometry.

a corresponding generic pretabulated SH projection can be utilized. By multiplying the resulting visibility functions (or adding them in log space) for the single spheres correct occluder fusion is performed, albeit the low-frequency nature of the SH representation prevents accurate results. Also, the sphere approximations are not always able to represent a character with sufficient precision.

Sloan et al. [2007] later improved the method and extended it to incorporate indirect lighting. While the sphere approximation and the choice of the SH basis enable real-time performance, they also cause unrealistic and missing shadows in locations where rather higher frequencies occur, e.g. where a box touches the ground. Contact shadows are often wrongly shaped and lose their contact, which is suboptimal since they constitute visually important features [Thompson et al., 1998]. Such inaccuracies become especially noticeable for smaller regions of high radiance, like the sun.

In contrast, such key lights are explicitly supported by Snyder and Nowrouzezahrai [2008], but they restrict themselves to the special case of dynamic height fields. Figure 6.3 illustrates the effect that can be achieved with ambient occlusion on height-field surfaces. This image well underlines the complementary nature of ambient occlusion and shadows. Direct illumination remains important for detail enhancement which is why environmental lighting (left) and shadow algorithms for key lights (right) should be combined (middle) to achieve the most appealing results. In their algorithm, Snyder and Nowrouzezahrai approximate the horizon blocking the environment light by visibility wedges and utilize pretabulated SH-projections for them to determine the overall visibility. In a follow-up [Nowrouzezahrai and Snyder, 2009] they accelerated this solution and extended it to indirect illumination.

6.2 Ambient Occlusion

A simpler scenario is Ambient Occlusion (AO). A survey of related techniques can be found in [Méndez-Feliu and Sbert, 2009]. It is an illumination method based on the assumption that light is locally impinging uniformly from all directions of the hemisphere onto each point of the scene. AO can be seen as an accessibility value [Miller, 1994]: The less a scene point is exposed to the rest of the scene, the darker it appears. The motivation behind it is that in the real world, we do not have a single point or area light source. Instead, light bounces off all

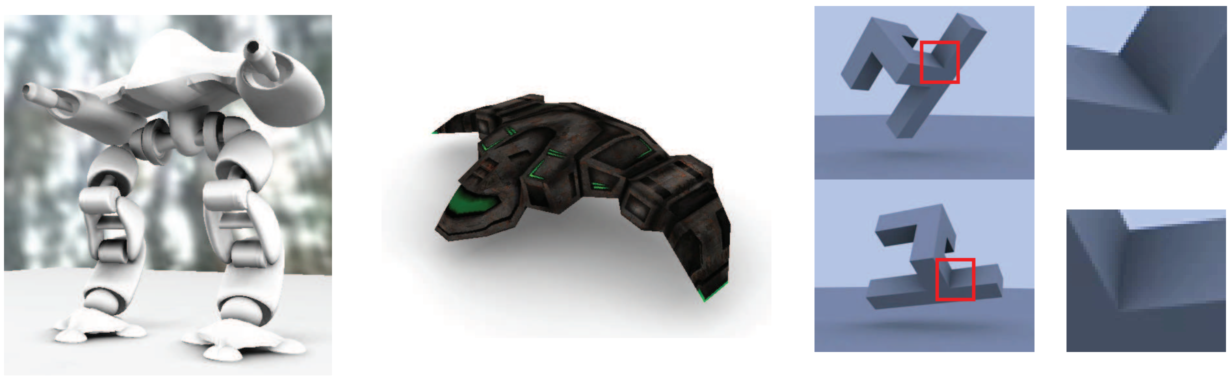


Figure 6.4 Ambient Occlusion often looks very convincing and is even used in the industry for feature-film productions. It delivers smooth and visually pleasing shadows (left, middle). One drawback is that, based on the "light from hemisphere" assumption, shading on the objects and "cast" shadows on the environment might look incoherent. In the lower image, the zoomed region would have received much more light if there had been a real caster that produced the shadow on the ground.

surfaces; it is coming from "everywhere", but is blocked by geometry nearby the impact point. An approximation such as ambient occlusion thus has a look of indirect illumination. This sounds like a crude approximation, but it leads to convincing and especially smooth shading.

Often AO is precomputed and stored in textures. This is usually done for static elements in games as well as movie productions, but this would not be reason enough to mention it in this tutorial where we focus on interactive shadow casting. The real reason is that such techniques can be used to compute a shadow-like effect for nearby elements of the scene. Nevertheless, this short chapter should be seen as an exploration of related topics.

6.2.1 Precomputed Ambient Occlusion

One way to achieve an approximation of ambient occlusion values and to propagate these into the space around an object is to *attach* occlusion information to objects. A preprocess is used to determine the occlusion values, but it also implies that the geometry of each object needs to remain static.

Kontkanen and Laine [2005] use a cube map surrounding the object where each texel entry stores occlusion information in form of a quadratic rational function of distance. It is evaluated for each pixel to derive its shading. They further combine several occluders based on a uniform repartition heuristic.

A faster and simpler solution was presented by Malmer et al. [2007]. They store ambient occlusion directly for points in space in a 3D texture surrounding the object. Theoretically, this results in an infinite support because some light should always be blocked by an object even when far away from the impact point. To obtain a finite support, they clamp the shadow values and ensure that outside of a fixed radius no more shading influence exists. Figure 6.4 shows an example of their results. The shadows are generally very blurry, which gives the images a nice feel.

Both solutions are extremely fast and another strength is that such methods are very temporally

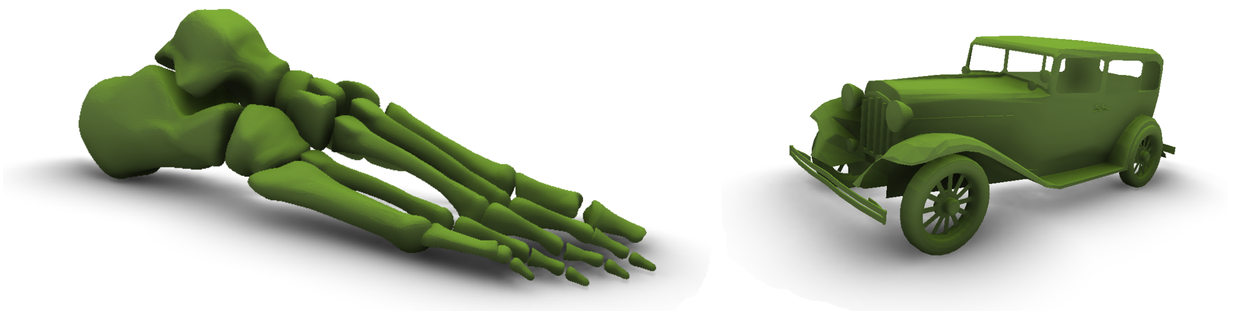


Figure 6.5 Screen-Space Ambient Occlusion - a, compared to the quality, relatively cheap procedure that estimates the incoming light at each pixel. From the point of view, the algorithm extracts a depth map, which is an approximation of the surface in form of a height field. The ambient occlusion value is derived by investigating a neighborhood in this depth map. Images were produced with [Bavoil et al., 2008b]

coherent. No high frequency defects appear in the image. Unfortunately, incoherences can always show and shadows might look incorrect and “attached” as illustrated in Figure 6.4.

6.2.2 Screen-Space Ambient Occlusion

For dynamic scenes, on-the-fly evaluations are necessary. Several approaches have been proposed and we refer the interested reader to, e.g., [Bunnell, 2006; Hoberock and Jia, 2007].

Nonetheless, one technique, Screen-Space Ambient Occlusion (SSAO), deserves special attention, as it is substantially used in games. The first appearance was in 2007 in the game Crysis by Crytek, described in more detail in [Akenine-Möller et al., 2008]. Recently it has even been added as a default option in modern graphics drivers. Figure 6.5 shows two result obtained with Screen-Space Ambient Occlusion (SSAO).

The most convincing element of this technique is its simplicity. The idea is to approximate ambient occlusion solely based on a depth map shot from the current point of view. For every pixel, the occlusion is estimated by sampling a neighborhood. The simplest way evaluates samples not unlike percentage-closer filtering (see Chapter 4). A more advanced approach is to evaluate the occlusion is to shoot rays in all 2D directions of the depth map. Along each ray, the steepest angle is measured. Hence, all directions combined define some cone opening. The larger the cone, the brighter the shading [Bavoil et al., 2008b]. This estimation was already very successful in the context of height-field illumination [Oat and Sander, 2007].

For high quality many such rays would be necessary. In practice, varying sets of rays can be used per pixel and an edge-aware filtering process combines the result of neighboring pixels in order to produce a smooth outcome. With such a method, 16 samples or less can be sufficient.

Extensions

The global illumination effect of light bleeding can be simulated by not only storing an angle, but also accumulating pixel colors. This results in colored shadows that further increase realism [Ritschel et al., 2009].

The drawback of screen space ambient occlusion is that it can lead to artifacts during animation if the depth map changes abruptly. Especially missing occluder information can lead to inconsistencies because, basically, only the first visible surfaces intervene. With this in mind, Reinbothe et al. [2009] applied a GPU voxelization technique [Eisemann and Décoret, 2006a] to obtain their *hybrid ambient occlusion*.

6.3 Conclusion

Image-based techniques have shown to be particularly efficient and even though SSAO might result in some artifacts due to the ambiguous surface definitions that arise from a depth-map interpretation. The result can be missing shadows and some temporal incoherence. Nevertheless, for nearby influences of objects on their surrounding, SSAO techniques are delivering visually pleasing results for a reasonable cost. In practice, even the standard SSAO approach works surprisingly well and will be used in many games to come. Nevertheless, SSAO should seen as a quality-improving supplement, it is not a replacement for soft-shadow computations. The missing directionality and it's screen-space nature do not provide the same visual cues that one can obtain via shadows. On the other hand, as an additional effect it is extremely useful and should be integrated to enhance the scene's details.

7 Conclusion

In practice, it is very frequent that people jump directly to conclusions. In a literal sense, so if you did, you might just as well continue reading because in this case, your choice of starting with the conclusion first, indicates what you are after: solutions to a particular problem, but not necessarily an exhaustive overview.

This tutorial hopefully provides a sufficient amount of details to understand and implement the major techniques. We described many in order to give a good overview of the state-of-the-art solutions. But why did we present so many techniques? Why not just the best solution? In fact, some readers might even feel a little overwhelmed by the content of these notes. The simple reason is that there is no ultimate shadow algorithm at the current moment in time. We have many solutions that all address certain situations and needs. Further, it can make perfect sense to combine several methods to yield a system that provides an even better behavior. We saw such a situation in Chapter 3 where reparametrization of shadow maps and partitioning lead to a better overall behavior. We feel it is therefore of large benefit to have a good overview of many techniques in order to find the most adequate solution for a particular situation.

In the following, we will investigate several application scenarios and point out which algorithms might be of high interest.

7.1 Hard Shadows

Algorithms for hard shadows are very interesting for real-time purposes and aim at the simulation of point light sources. The shadows look sharp, which might result in an artificial appearance. But it is easier to avoid artifacts than in the other shadow categories, making them still very interesting in the context of high-quality imagery.

Accurate If accurate hard shadows are needed, geometry cannot be ignored and needs to be considered. Usually, this implies that some culling should be applied to only consider the casters that are necessary and shadow volumes should be clamped if possible. There is some work in this direction [Lloyd et al., 2004], but it does not eliminate the problem of the actual shadow computation. Shadow volumes [Crow, 1977] in Section 2.2 are a good option to compute high-quality shadows. The overdraw and geometry processing can be prohibitive, but it found practical application in the industry [Stich et al., 2007].

Alternatively, accurate shadows can be computed with Irregular Z-Buffer [Johnson et al., 2005] strategies. Related CPU [Aila and Laine, 2004] and GPU [Sintorn et al., 2008] implementations exist, where the CPU solutions have more hierarchical adaptation, whereas the GPU solutions are currently clearly faster. The downside is that lights cannot be omnidirectional and the algorithms become costly if the shadow map has to cover a large scene. Adaptations in the direction of light space perspective shadow maps can improve the result significantly. Currently, it seems to be one of the more promising directions to go.

Approximately Accurate When lowering the quality to only approximately accurate hard shadows, very efficient solutions can be obtained. Approaches related to Fitted Virtual Shadow Maps [Giegl and Wimmer, 2007a] and [Lefohn et al., 2007] in Section 3.2.4.2 are most promising. These approaches make a good guess for a partitioning of the shadow map, in order to achieve a close-to-accurate result. The advantage of [Lefohn et al., 2007] is that it runs entirely on the GPU. But currently, it seems that putting some workload on the CPU, slightly accelerates the process. Depending on the workload of the application, one might favor one or the other.

Approximate When entering the realm of approximate hard shadows, the most important aspect is performance. The cheapest solutions are based on shadow map parameterizations. Light Perspective Shadow Maps [Wimmer et al., 2004] in Section 3.2.2.1 are a very good option. They basically come at almost no cost because the only GPU-side change is to switch the original light matrix to a modified one. This makes it particularly appealing for games.

Only slightly more expensive, but significantly more general, are Cascaded Shadow Maps [Engel, 2006] or Parallel Split Shadow Maps [Zhang et al., 2006a], where the view frustum is decomposed into several distances, mimicking the optimal, logarithmic, behavior. These techniques are a good compromise between quality and cost, and are probably the most practical and most widely used hard shadow algorithm today. They deliver even better quality when combined with the aforementioned parametrization techniques.

If computation time remains to treat geometry, Silhouette Shadow Maps (see box on page 47) are a valuable addition to hide some more of the pixel artifacts.

7.2 Filtered Hard Shadows

The artificial hard-shadow look can be hidden when applying a certain amount of blur to the shadow boundary. Currently, most real-time applications rely on shadows of this kind. In the case of small smooth shadow boundaries, the good tradeoff between cost and visual quality makes filtered shadows a good option for convincing, yet cheap solutions. Many methods aim at speeding up the computation of standard percentage-closer filtering [Reeves et al., 1987] (Section 4.1.2) and the main differences are related to the memory budget.

If memory is an issue, the best approach is probably still a well-tweaked implementation of Variance Shadow Maps [Donnelly and Lauritzen, 2006] (Section 4.2.1). Light leaks may appear, that can be combatted through Exponential Shadow Maps [Salvi, 2008; Annen et al., 2008b], or Exponential Warping [Lauritzen and McCool, 2008] (Section 4.2.4).

The other advantage is that filtered shadows address aliasing due to foreshortening. This is a very good property that allows nice looking shadows. Nonetheless, the result lacks the physical correctness in some scenarios, but in a game context, a designer is often able to tweak the scene in order to avoid such problems.

7.3 Soft Shadows

The highest physical plausibility is achieved by soft-shadow methods. These can either be used to achieve convincing shadows, but are also of interest when actual physical simulations are a must, such as for the previsualization applied in the context of architectural design.

Accurate Currently, the most efficient algorithm for animated scenes with reasonable light sources are View-Sample Mapping [Sintorn et al., 2008] solutions (Section 5.3.3). Depth-Complexity Sampling [Forest et al., 2008] approaches (Section 5.3.2.3) are generally slower and would only become interesting for larger sources, where View-Sample Mapping can become costly. Unfortunately, for such a scenario, it is very important to not only consider silhouette edges from the light's center. This results in more edges to be transformed into quad primitives which strongly increases the cost of the soft shadow volume approaches. The principle does help though with more advanced data structures to perform more efficient ray-tracing [Laine et al., 2005]. But currently, this leads us into an offline scenario and makes dynamic scene changes problematic. For highest accuracy, beam tracing [Overbeck et al., 2007] can be employed, but it can become very costly when primitives are complicated. The topic of accurate soft shadows will occupy us for some time to come.

Approximate This category of shadows is currently the most interesting for real-time applications. Relatively high quality is combined with a reasonable computational overhead. Percentage-Closer Soft Shadows (PCSS) [Fernando, 2005] in Section 5.2.2 are the state of the art for interactive applications such as games. It results in reasonable shadows for smaller light sources and achieves acceptable performance. One important element of PCSS is that they work in very large scenes and help to achieve realistic sunlight effects. If the scene is of a smaller extent, a practical solution are Occlusion Textures [Eisemann and Décoret, 2006b] (Section 5.2.5). Their performance is extremely high (on a Geforce 6 real-time performance is possible for 500,000 polygons). Their main advantage is the independence of the light's size. This allows soft shadows at very low cost and makes them a good solution for interior scenes with large sources. For large scenes, the method does not scale well.

For higher precision and more robustness, backprojection methods, introduced in [Atty et al., 2006; Guennebaud et al., 2006] (Section 5.2.3), could be used. The most accurate results are obtained when using occlusion bitmasks, ideally combined with microquads [Schwarz and Stamminger, 2007]. Occluder contours [Guennebaud et al., 2007] and simple area accumulation often yield good results, too; a rather fast implementation is possible by adopting hierarchical contour extraction and packet-based processing [Yang et al., 2009]. Often, achieving high performance may necessitate resorting to coarser occluder approximations, which however requires care to avoid artifacts like notable transitions; in particular, without adequate adjustment to the actual scene, results can easily lead to visually less pleasing shadows compared to the approximate solution obtained with simple PCSS.

7.4 Welcome to Tomorrow

As always, when giving advice, it is based on the current conditions. At the moment, we rely on programmable graphics hardware following the rasterization pipeline. Hardware modifications can thus void our previous suggestions. E.g., Doom 3 was a game that relied on shadow volumes [Crow, 1977], a technique that was previously considered too expensive because it does not scale well with geometry. But the new hardware made it possible to significantly reduce the polygon count by relying on new shading capabilities that simulate geometric detail via normal mapping, making shadow volumes again feasible. Today, shadow maps are again the first choice and it is likely that this will stay true in the near future.

Nevertheless, at the moment, we are at the point where tremendous hardware changes await. Larrabee is about to be introduced and we talked about some first algorithms for this platform. As it is no longer aiming preferably at rasterization techniques, hardware like this might open up the road to ray tracing in the long run. For the next years though, we believe that our assessment will be of help to guide you through the jungle of different shadow techniques.

We hope you enjoyed this tutorial and invite you to visit our webpage `http://www.mpi-inf.mpg.de/resources/ShadowCourse/`, where we will continue to provide you with more information in the future.

"The true work of art is but a shadow of the divine perfection."

Michelangelo (*1475 -1564*)

8 Acknowledgments

We would like to thank the many people that participated to this tutorial and the anonymous reviewers for their feedback.

Many thanks go to **Louis Bavoil** for his many insightful comments and the fruitful discussions. Also, we would like to particularly thank him for sharing many of his soft shadow implementations with us.

A big thank you also goes to **Zhao Dong** who helped us out with his implementation of the Convolution Soft Shadow Maps and produced some figures with his algorithm.

Thanks go to **Cyril Soler** and **Hedlena Bezerra** for some feedback and proofreading on a very early version of these notes. Thanks also to **Brandon Lloyd** for some figures and interesting discussion.

We are very grateful for the possibility to use the high quality models in this paper. They were provided by courtesy of Aim@Shape, DeEspona, INRIA, Stanford University and Utah University.

Bibliography

Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Proceedings of ACM SIGGRAPH 2000*, pages 375–384, July 2000.

Timo Aila and Tomas Akenine-Möller. A hierarchical shadow volume algorithm. In *Proceedings of Graphics Hardware 2004*, pages 15–23, August 2004.

Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 161–166, June 2004.

Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Proceedings of Eurographics Workshop on Rendering 2002*, pages 297–306, June 2002.

Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering*. A K Peters, 3rd edition, 2008.

Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 51–60, June 2007.

Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2008)*, 27(3):34:1–34:8, August 2008a.

Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proceedings of Graphics Interface 2008*, pages 155–161, May 2008b.

Jukka Arvo. Tiled shadow maps. In *Proceedings of Computer Graphics International 2004*, pages 240–246, June 2004.

Jukka Arvo, Mika Hirvikorpi, and Joonas Tyystjärvi. Approximate soft shadows with an image-space flood-fill algorithm. *Computer Graphics Forum (Proceedings of Eurographics 2004)*, 23(3):271–280, September 2004.

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

Ulf Assarsson and Tomas Akenine-Möller. Occlusion culling and z-fail for soft shadow volume algorithms. *The Visual Computer*, 20(8-9), 2004.

Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *Proceedings of Graphics Hardware 2003*, pages 33–40, July 2003.

Barnabás Aszódi and László Szirmay-Kalos. Real-time soft shadows with shadow accumulation. In *Eurographics 2006 Short Papers*, pages 53–56, September 2006.

Lionel Atty, Nicolas Holzschuch, Marc Lapierre, Jean-Marc Hasenfratz, Charles Hansen, and François X. Sillion. Soft shadow maps: Efficient sampling of light source visibility. *Computer Graphics Forum*, 25(4):725–741, 2006.

Louis Bavoil. Advanced soft shadow mapping techniques, 2008. Presentation, *Game Developers Conference 2008*. http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_SoftShadowMapping.pdf.

Louis Bavoil and Cláudio T. Silva. Real-time soft shadows with cone culling. In *ACM SIGGRAPH 2006 Sketches*, July 2006.

Louis Bavoil, Steven P. Callahan, and Cláudio T. Silva. Robust soft shadow mapping with backprojection and depth peeling. *Journal of Graphics Tools*, 13(1):19–30, 2008a.

Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, August 2008b.

P. Bergeron. A general version of crow's shadow volumes. *Computer Graphics and Applications*, 6(9):17–28, 1986.

William Bilodeau and Mike Songy. Real time shadows, May 1999. Creativity 1999, Creative Labs Inc. Sponsored game developer conferences, Los Angeles, California, and Surrey, England.

Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Proceedings of Graphics Interface 2002*, pages 219–228, May 2002.

Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Proceedings of Eurographics)*, 25(3), 2003.

Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Advances in Modelling, Animation and Rendering (Proceedings Computer Graphics International 2002)*, pages 397–408. Springer, 2002. ISBN 1-85233-654-4.

Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Practical shadow mapping. In *Graphics Tools: The jgt Editors' Choice*, page 343. A.K. Peters, 2005. ISBN 1568812469.

Michael Bunnell. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting. Addison-Wesley, 2006.

John Carmack. Z-fail shadow volumes. Internet Forum.

Patrick Cavanagh. The artist as neuroscientist. *Nature*, 434(7031):301–307, 2005.

Eric Chan and Frédo Durand. Rendering fake soft shadows with smoothies. In *Proceedings of Eurographics Symposium on Rendering 2003*, pages 208–218, June 2003.

Ying-Chieh Chen and Chun-Fa Chang. A prism-free method for silhouette rendering in inverse displacement mapping. *Computer Graphics Forum (Proceedings of Pacific Graphics 2008)*, 27(7):1929–1936, October 2008.

Norman Chin and Steven Feiner. Near real-time shadow generation using BSP trees. *Computer Graphics (Proceedings of SIGGRAPH)*, 24(4):99–106, 1990.

Hamilton Chong. Real-Time Perspective Optimal Shadow Maps. Master's thesis, Harvard University, 2003.

Yiorgos Chrysanthou and Mel Slater. Shadow volume BSP trees for computation of shadows in dynamic scenes. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics)*, pages 45–50, 1995.

Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (Proceedings of ACM SIGGRAPH 77)*, 11(2):242–248, 1977.

Franklin C. Crow. Summed-area tables for texture mapping. *Computer Graphics (Proceedings of ACM SIGGRAPH 84)*, 18(3):207–212, July 1984.

Carsten Dachsbacher, Marc Stamminger, George Drettakis, and Frédo Durand. Implicit visibility and antiravanagh:artistasnadiance for interactive global illumination. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3):61, 2007. ISSN 0730-0301.

Xavier Décoret. N-buffers for efficient depth map query. *Computer Graphics Forum (Proceedings of Eurographics 2005)*, 24(3):393–400, September 2005.

Christopher DeCoro, Forrester Cole, Adam Finkelstein, and Szymon Rusinkiewicz. Stylized shadows. In *Proceedings of NPAR (Symposium on Non-Photorealistic Animation and Rendering)*, pages 77–83. ACM, 2007. ISBN 978-1-59593-624-0.

Eugene d'Eon and David Luebke. Advanced techniques for realistic real-time skin rendering. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 14, pages 293–348. Addison Wesley Professional, August 2007. ISBN 0-321-51526-9.

Kirill Dmitriev. Soft shadows. White Paper WP-03016-001_v01, NVIDIA Corporation, February 2007.
http://developer.download.nvidia.com/whitepapers/2007/SDK10/SoftShadows.pdf.

William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2006*, pages 161–165, March 2006.

Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, January 1972.

Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3):257–266, July 2002.

Elmar Eisemann. *Optimized Representations for the Acceleration of Display- and Collision Queries*. PhD thesis, Grenoble Universities, 2008.

Elmar Eisemann and Xavier Décoret. Fast scene voxelization and applications. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)*, pages 71–78. ACM SIGGRAPH, 2006a.

Elmar Eisemann and Xavier Décoret. Plausible image based soft shadows using occlusion textures. In *Proceedings of SIBGRAPI 2006*, pages 155–162, October 2006b.

Elmar Eisemann and Xavier Décoret. Visibility sampling on GPU and applications. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):535–544, September 2007.

Elmar Eisemann and Xavier Décoret. Occlusion textures for plausible soft shadows. *Computer Graphics Forum*, 27(1):13–23, March 2008.

Wolfgang Engel. Cascaded shadow maps. In Wolfgang Engel, editor, *ShaderX^5: Advanced Rendering Techniques*, pages 197–206. Charles River Media, December 2006. ISBN 1-58450-499-4.

Cass Everitt and Mark J. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Published online at
http://developer.nvidia.com, 2002.

Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, July 2005.

flickrPrince. P9300851, `http://www.flickr.com/photos/prince_tigereye/1463788115/`, 2007. published under Creative Commons `http://creativecommons.org/licenses/by/2.0/deed.de`.

Vincent Forest, Loïc Barthe, and Mathias Paulin. Realistic soft shadows by penumbra-wedges blending. In *Proceedings of Graphics Hardware 2006*, pages 39–47, September 2006.

Vincent Forest, Loïc Barthe, and Mathias Paulin. Accurate shadows by depth complexity sampling. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2):663–674, April 2008.

Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier, and Bernard Peroche. Fast non-linear projections using graphics hardware. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)*, 2008.

Markus Giegl and Michael Wimmer. Fitted virtual shadow maps. In *Proceedings of Graphics Interface 2007*, pages 159–168, May 2007a.

Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)*, pages 65–72. ACM Press, 2007b. ISBN 978-1-59593-628-8.

Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *Proceedings of ACM SIGGRAPH 93*, pages 231–238, August 1993.

Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. Real-time soft shadow mapping by back-projection. In *Proceedings of Eurographics Symposium on Rendering 2006*, pages 227–234, June 2006.

Gaël Guennebaud, Loïc Barthe, and Mathias Paulin. High-quality adaptive soft shadow mapping. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):525–533, September 2007.

Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, 6(1):19–27, 2001. ISSN 1086-7651.

Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, December 2003.

Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997.

Tim Heidmann. Real shadows real time. *IRIS Universe*, 18:28–31, November 1991.

Wolfgang Heidrich. *High-quality shading and lighting for hardware-accelerated rendering*. PhD thesis, Universität Erlangen, 1999.

Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)*, pages 39–45, 1998.

Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, 33(Annual Conference Series):171–178, 1999. ISSN 0097-8930.

Wolfgang Heidrich, Katja Daubert, Jan Kautz, and Hans-Peter Seidel. Illuminating micro geometry based on precomputed visibility. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 455–464. ACM Press/Addison-Wesley Publishing Co., 2000. ISBN 1-58113-208-5.

Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast summed-area table generation and its applications. *Computer Graphics Forum (Proceedings of Eurographics)*, 24(3), 2005.

Jared Hoberock and Yuntao Jia. High-quality ambient occlusion. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 12, pages 257–274. Addison Wesley Professional, August 2007. ISBN 0-321-51526-9.

Samuel Hornus, Jared Hoberock, Sylvain Lefebvre, and John C. Hart. ZP+: correct z-pass stencil shadows. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)*. ACM, ACM Press, 2005.

J.-C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computer & Graphics*, pages 259–265, 1985.

Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Transactions on Graphics*, 24(4):1462–1482, 2005. ISSN 0730-0301.

Gregory S. Johnson, Warren A. Hunt, Allen Hux, William R. Mark, Christopher A. Burns, and Stephen Junkins. Soft irregular shadow mapping: Fast, high-quality, and robust soft shadows. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2009*, pages 57–66, February 2009.

James T. Kajiya. The rendering equation. *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, 20(4):143–150, August 1986.

Jan Kautz, Jaakko Lehtinen, and Timo Aila. Hemispherical rasterization for self-shadowing of dynamic objects. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, pages 179–184. Eurographics Association, 2004.

Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Proceedings of Eurographics Workshop on Rendering 1999*, pages 197–212, June 1999.

D. Kersten, D. C. Knill, P. Mamassian, and I. Bülthoff. Illusory motion from shadows. *Nature*, 379(31), 1996.

Florian Kirsch and Jürgen Döllner. Real-time soft shadows using a single light sample. *Journal of WSCG*, 11(2):255–262, February 2003.

Janne Kontkanen and Samuli Laine. Ambient occlusion fields. In *Proceedings of I3D (ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games)*, pages 41–48. ACM Press, 2005.

Janne Kontkanen, Emmanuel Turquin, Nicolas Holzschuch, and François Sillion. Wavelet radiance transport for interactive indirect lighting. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics, 2006.

Anders Wang Kristensen, Tomas Akenine-Möller, and Henrik Wann Jensen. Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3):1208–1215, 2005. ISSN 0730-0301.

Samuli Laine. Split-plane shadow volumes. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 23–32. ACM Press, 2005.

Samuli Laine and Timo Aila. Hierarchical penumbra casting. *Computer Graphics Forum (Proceedings of Eurographics 2005)*, 24(3):313–322, September 2005.

Samuli Laine, Timo Aila, Ulf Assarsson, Jaakko Lehtinen, and Tomas Akenine-Möller. Soft shadow volumes for ray tracing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2005)*, 24(3):1156–1165, July 2005.

Andrew Lauritzen. Summed-area variance shadow maps. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 8, pages 157–182. Addison Wesley Professional, August 2007. ISBN 0-321-51526-9.

Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *Proceedings of Graphics Interface 2008*, pages 139–146, May 2008.

Aaron Lefohn, Shubhabrata Sengupta, Joe M. Kniss, Robert Strzodka, and John D. Owens. Dynamic adaptive shadow maps on graphics hardware. In *ACM SIGGRAPH Conference Abstracts and Applications*, 2005.

Aaron Lefohn, Joe M. Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics*, 25(1): 60–99, 2006.

Aaron E. Lefohn, Shubhabrata Sengupta, and John D. Owens. Resolution matched shadow maps. *ACM Transactions on Graphics*, 26(4):20:1–20:17, 2007.

Jaakko Lehtinen, Samuli Laine, and Timo Aila. An improved physically-based soft shadow volume algorithm. *Computer Graphics Forum (Proceedings of Eurographics 2006)*, 25(3): 303–312, September 2006.

Jaakko Lehtinen, Matthias Zwicker, Janne Kontkanen, Emmanuel Turquin, François Sillion, and Timo Aila. Meshless finite elements for hierarchical global illumination. Technical Report TML-B7, Helsinki University of Technology, 2007.

Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. A meshless hierarchical representation for light transport. *ACM Trans. Graph.*, 27(3):1–9, 2008. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1360612.1360636.

E. Lengyel. Advanced stencil shadow and penumbral wedge rendering. Presentation at Game Developers Conference 2005.
http://www.terathon.com/gdc_lengyel.ppt.

Dani Lischinski and Ari Rappoport. Image-based rendering for non-diffuse synthetic scenes. In *Proceedings of Eurographics Workshop on Rendering 1998*, pages 301–314, June 1998.

Brandon Lloyd. *Logarithmic Perspective Shadow Maps*. PhD thesis, University of North Carolina, August 2007.

Brandon Lloyd, J. Wendt, Naga K. Govindaraju, and Dinesh Manocha. CC shadow volumes. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*. Eurographics Association, 2006.

D. Brandon Lloyd, Naga K. Govindaraju, Steven E. Molnar, and Dinesh Manocha. Practical logarithmic rasterization for low-error shadow maps. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)*, pages 17–24. Eurographics Association, 2007. ISBN 978-1-59593-625-7.

D. Brandon Lloyd, Naga K. Govindaraju, Cory Quammen, Steven E. Molnar, and Dinesh Manocha. Logarithmic perspective shadow maps. *ACM Transactions on Graphics*, 27(4): 106:1–106:32, October 2008.

Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of ACM SIGGRAPH 2000*, pages 385–392, July 2000.

Charles Loop and Jim Blinn. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2006)*, 25(3):664–670, July 2006.

Thomas Luft and Oliver Deussen. Real-time watercolor illustrations of plants using a blurred depth test. In *Proceedings of Symposium on Non-Photorealistic Animation and Rendering 2006*, pages 11–20, June 2006.

Mattias Malmer, Fredrik Malmer, Ulf Assarsson, and Nicolas Holzschuch. Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools*, 12(2):59–71, 2007.

Pascal Mamassian, David C. Knill, and Daniel Kersten. The perception of cast shadows. *Trends in Cognitive Sciences*, 2(8):288–295, August 1998.

Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, Springer Computer Science, pages 153–160. Eurographics, Eurographics Association, 2004.

Oliver Mattausch, Jiří Bittner, and Michael Wimmer. CHC++: Coherent hierarchical culling revisited. *Computer Graphics Forum (Proceedings of Eurographics 2008)*, 27(2):221–230, April 2008.

Morgan McGuire. Observations on silhouette sizes. *Journal of Graphics Tools*, 9(1):1–12, 2004.

Àlex Méndez-Feliu and Mateu Sbert. From obscurances to ambient occlusion: A survey. *The Visual Computer*, 25(2):181–196, February 2009.

Merriam-Webster. Merriam-webster online dictionary. http://www.merriam-webster.com/dictionary/, 2009.

Morten S. Mikkelsen. Separating-plane perspective shadow mapping. *Journal of Graphics Tools*, 12(3):43–54, 2007.

Gavin Miller. Efficient algorithms for local and global accessibility shading. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 319–326. ACM, 1994. ISBN 0-89791-667-0.

Qi Mo, Voicu Popescu, and Chris Wyman. The soft shadow occlusion camera. In *Proceedings of Pacific Graphics 2007*, pages 189–198, October 2007.

Rui Ni, Myron L. Braunstein, and George J. Andersen. Interaction of optical contact, shadows and motion in determining perceived scene layout. *Journal of Vision*, 4(8):615–615, 2004. ISSN 1534-7362.

Derek Nowrouzezahrai and John Snyder. Fast global illumination on dynamic height fields. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2009)*, 28 (4):1131–1139, June 2009.

Christopher Oat and Pedro V. Sander. Ambient aperture lighting. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2007*, pages 61–64, April 2007.

Manuel M. Oliveira and Fábio Policarpo. An efficient representation for surface details. Technical Report RP-351, Universidade Federal do Rio Grande do Sul, January 2005.

Ryan Overbeck, Ravi Ramamoorthi, and William R. Mark. A real-time beam tracer with application to exact soft shadows. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 85–98, June 2007.

Christian Azambuja Pagot, Joao Luiz Dihl Comba, and Manuel Menezes de Oliveira Neto. Multiple-depth shadow maps. In *Proceedings of SIBGRAPI (Brazilian Symposium on Computer Graphics and Image Processing)*, pages 308–315. IEEE Computer Society, 2004. ISBN 0-7695-2227-0.

Sylvain Paris and Frédo Durand. A fast approximation of the bilateral filter using a signal processing approach. In *Proceedings of European Conference on Computer Vision (ECCV) 2006*, 2006.

Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, University of Utah, October 1998.

Joseph P. Pickett et al., editors. *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, 4th edition, 2000.

Princeton University. WordNet 3.0.
http://wordnet.princeton.edu/, 2009.

William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):283–291, July 1987.

Christoph Reinbothe, Tamy Boubekeur, and Marc Alexa. Hybrid ambient occlusion. *EUROGRAPHICS 2009 Areas Papers*, 2009.

Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2006)*, 25(3):977–986, July 2006.

Tobias Ritschel, Thorsten Grosch, Jan Kautz, and Hans-Peter Seidel. Interactive global illumination based on coherent surface shadow maps. In *Proceedings of GI (Graphics Interface)*, 2008.

Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games 2009*, pages 75–82, February 2009.

Guodong Rong and Tiow-Seng Tan. Utilizing jump flooding in image-based soft shadows. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 2006*, pages 173–180, November 2006.

Marco Salvi. Rendering filtered shadows with exponential shadow maps. In Wolfgang Engel, editor, *ShaderX^6: Advanced Rendering Techniques*, pages 257–274. Charles River Media, February 2008. ISBN 1-58450-544-3.

Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 45–50, June 2007.

Christophe Schlick. A survey of shading and reflectance models. *Computer Graphics Forum*, 13 (2):121–131, 1994.

Peter Schröder and Pat Hanrahan. On the form factor between two polygons. In *SIGGRAPH: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 163–164. ACM, 1993. ISBN 0-89791-601-8.

Christian Schüler. Eliminate surface acne with gradient shadow mapping. In *ShaderX^4: Advanced Rendering Techniques*. Charles River Media, Inc., 2006.

Michael Schwarz and Marc Stamminger. Bitmask soft shadows. *Computer Graphics Forum (Proceedings of Eurographics 2007)*, 26(3):515–524, September 2007.

Michael Schwarz and Marc Stamminger. Microquad soft shadow mapping revisited. In *Eurographics 2008 Annex to the Conference Proceedings (Short Papers)*, pages 295–298, April 2008a.

Michael Schwarz and Marc Stamminger. Quality scalability of soft shadow mapping. In *Proceedings of Graphics Interface 2008*, pages 147–154, May 2008b.

Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of Graphics Hardware 2006*, pages 53–60, September 2006.

Pradeep Sen. Silhouette maps for improved texture magnification. In *Proceedings of Graphics Hardware (ACM SIGGRAPH/Eurographics Workshop on GH)*, pages 65–73. ACM, 2004. ISBN 3-905673-15-0.

Pradeep Sen, Mike Cammarano, and Pat Hanrahan. Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 22(3):521–526, 2003. ISSN 0730-0301.

Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of ACM SIGGRAPH 98*, pages 231–242, July 1998.

Ryan Shrout. Rendering games with raytracing will revolutionize graphics - pc perspective. http://www.pcper.com/article.php?aid=455&type=expert&pid=3, 2007.

Francois X. Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers Inc., 1994. ISBN 1558602771.

Erik Sintorn, Elmar Eisemann, and Ulf Assarsson. Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2008)*, 27(4):1285–1292, June 2008.

Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002)*, 21(3):527–536, July 2002.

Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of Pacific Graphics 2007*, pages 97–105, October 2007.

S. M. Smith and J. M. Brady. SUSAN – A new approach to low level image processing. Technical Report TR95SMS1c, Chertsey, Surrey, UK, 1995.

John Snyder and Derek Nowrouzezahrai. Fast soft self-shadowing on dynamic height fields. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2008)*, 27 (4):1275–1283, June 2008.

Cyril Soler. *Représentations hiérarchiques de la visibilité pour le contrôle de l'erreur en simulation de l'éclairage*. PhD thesis, Université Joseph Fourier (Grenoble), 1998.

Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Proceedings of ACM SIGGRAPH 98*, pages 321–332, July 1998.

Jean-François St-Amour, Eric Paquette, and Pierre Poulin. Soft shadows from extended light sources with penumbra deep shadow maps. In *Proceedings of Graphics Interface 2005*, pages 105–112, May 2005.

Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 557–562, 2002.

Martin Stich, Carsten Wächter, and Alexander Keller. Efficient and robust shadow volumes using hierarchical occlusion culling and geometry shaders. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 11, pages 239–256. Addison Wesley Professional, August 2007. ISBN 0-321-51526-9.

Carsten Stoll, Stefan Gumhold, and Hans-Peter Seidel. Incremental raycasting of piecewise quadratic surfaces on the GPU. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, pages 141–150, September 2006.

William B. Thompson, Brian Smits, Peter Shirley, Daniel J. Kersten, and Cindee Madison. Visual glue. Technical Report UUCS-98-007, University of Utah, March 1998.

Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 839–846, 1998.

Y. Uralsky. Efficient soft-edged shadows using pixel shader branching. In *GPU Gems 2*. Addison Wesley, 2005.

Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 485–493, 2006.

Yulan Wang and Steven Molnar. Second-depth shadow mapping. Technical Report TR 94-019, University of North Carolina at Chapel Hill, 1994.

D. Weiskopf and T. Ertl. Shadow Mapping Based on Dual Depth Layers. In *Short Paper Eurographics*, pages 53–60, 2003.

Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 78)*, 12(3):270–274, August 1978.

Lance Williams. Pyramidal parametrics. *Computer Graphics (Proceedings of ACM SIGGRAPH 83)*, 17(3):1–11, July 1983.

Michael Wimmer and Daniel Scherzer. Robust shadow mapping with light-space perspective shadow maps. In Wolfgang Engel, editor, *ShaderX^4: Advanced Rendering Techniques*, pages 313–330. Charles River Media, January 2006. ISBN 1-58450-425-0.

Michael Wimmer, D. Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, Springer Computer Science. Eurographics, Eurographics Association, 2004.

Andrew Woo. *Graphics Gems III*, chapter The shadow depth map revisited, pages 338–342. Academic Press Professional, Inc., 1992. ISBN 0-12-409671-9.

Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Proceedings of Eurographics Symposium on Rendering 2003*, pages 202–207, June 2003.

Feng Xie, Eric Tabellion, and Andrew Pearce. Soft shadows by ray tracing multilayer transparent shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 265–276, June 2007.

Baoguang Yang, Jieqing Feng, Gaël Guennebaud, and Xinguo Liu. Packet-based hierarchal soft shadow mapping. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering 2009)*, 28(4):1121–1130, June 2009.

Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of Virtual Reality Continuum and Its Applications 2006*, pages 311–318, June 2006a.

Fan Zhang, Leilei Xu, Chenjun Tao, and Hanqiu Sun. Generalized linear perspective shadow map reparameterization. In *Proceedings of VRCIA (International Conference on Virtual Reality Continuum and Its Applications)*, pages 339–342. ACM, 2006b. ISBN 1-59593-324-7.

Fan Zhang, Hanqiu Sun, and Oskari Nyman. Parallel-split shadow maps on programmable GPUs. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 10, pages 203–238. Addison Wesley Professional, August 2007. ISBN 0-321-51526-9.

Fan Zhang, Alexander Zaprjagaev, and Allan Bentham. Practical cascaded shadow maps. In Wolfgang Engel, editor, *ShaderX^7: Advanced Rendering Techniques*, pages 305–330. Charles River Media, March 2009. ISBN 1-58450-598-2.

Hansong Zhang. Forward shadow mapping. In *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, Springer Computer Science, pages 131–138. Eurographics, Eurographics Association, 1998.