Master's Thesis

Edge Distance Shadow Mapping

Michael Kenzel

Supervisor:

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg

Institute of Computer Graphics and Vision,
Graz University of Technology

Graz, May 2013

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Abstract

Shadow mapping is one of the most popular algorithms for rendering shadows in computer generated images. It offers distinctive advantages in terms of simplicity, generality and performance, while at the same time being a natural fit for the rasterization-based rendering systems commonly used today. But the technique is extremely vulnerable to artifacts caused by aliasing effects, which can greatly compromise the quality of the generated shadows. We propose a new algorithm that not only avoids the common aliasing artifacts and delivers pixel-perfect hard shadows at interactive frame rates, but can even take advantage of perspective aliasing for the generation of soft shadows.

Kurzfassung

Shadow Mapping ist einer der populärsten Algorithmen um computergenerierte Bilder mit Schatten zu versehen. Die überragenden Vorteile von Shadow Mapping liegen in seiner Einfachheit bei gleichzeitiger Generalität und hoher Performance. Zusätzlich eignet es sich hervorragend für die Implementierung in auf Rasterisierung basierenden Renderingsystemen. Allerdings leidet die Bildqualität der mit Shadow Mapping erzeugten Schatten unter Aliasing Effekten. Diese Arbeit befasst sich mit einem neuen Algorithmus, der in der Lage ist, die üblichen durch Aliasing erzeugten Artefakte zu umgehen und exakte Schlagschatten zu liefern.

it's not the end; it's the beginning...

Contents

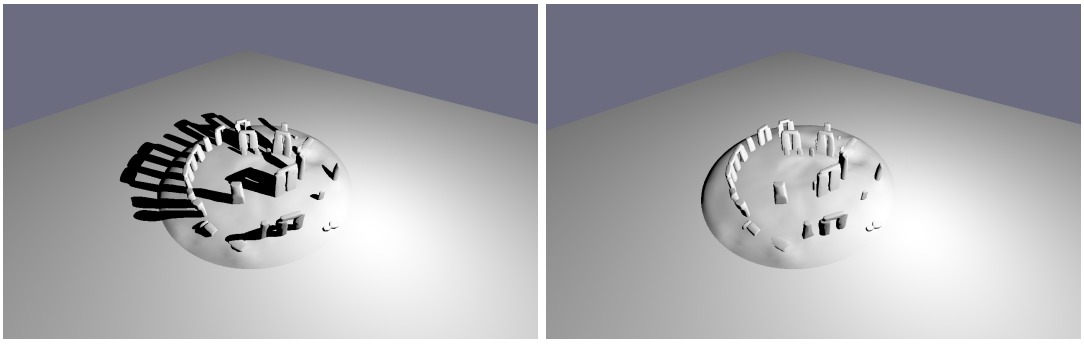
1 Introduction

Photons emitted from a light source travel through space until they interact with other particles. Such interactions may, among other possibilities, lead to absorption, emission and deflection of photons. These comparatively simple local interactions give rise to extremely complex phenomena, as photons influenced by one particle again interact with another particle. Information is transmitted across vast distances almost instantly. In general, any object in a natural scene has a potential influence on the visual appearance of any other object. The sheer number of particles that make up the physical objects in such a scene is enormous, as is the number of photons illuminating it.

Looking at the laws of physics that govern the world we see around us, simulating the propagation of light through an arbitrary scene as it reaches the eye of the observer becomes a daunting task. But it is exactly this problem one has to solve in digital image synthesis. Due to the almost incomprehensible scale of the problem, one has to resort to—at times very crude—approximations. Especially in interactive applications, global phenomena, *i.e.*, all indirect influences of objects onto one another, are generally ignored and only direct illumination is considered, greatly reducing the complexity of the problem. But some of these global phenomena have a defining influence on the visual appearance of a natural scene and cannot simply be ignored if visual realism is a concern. One such phenomenon of fundamental importance to the perceived realism of a scene are shadows (Figure 1.1).

Due to their nature as a global illumination phenomenon, fast and efficient generation of dynamic, accurate, and high-quality shadows is still a challenging problem in computer graphics, especially when interactive frame rates are to be accomplished. All the algorithms used in digital image synthesis today basically belong to one of two categories:

- *offline* algorithms rely on complete scene information being available during ren-



(a) (b)

Figure 1.1: Two images of the same scene, once with and once without shadow to illustrate the importance of shadows to the overall appearance of a scene.

dering, whereas

- *online* algorithms produce an output image while streaming scene data.

In many offline rendering algorithms, *e.g.*, most ray-tracing-based approaches, shadows arise naturally due to the global nature of the underlying method. But such algorithms are generally too computationally expensive to fit into the tight budget of interactive applications. Rendering in such applications, is usually based on online rasterization of primitives, which lends itself more easily to efficient hardware implementation. While rasterization enables high-performance graphics, shadows pose a challenging problem in online rendering systems.

Figure 1.2 illustrates the basic configuration considered in shadow rendering. A single *light source* illuminates a scene. An *occluder* blocks the direct path from the light source to the *receiver*, limiting the amount of light that can reach each point on the surface of the receiver, generating a shadow. Taking into account the physical extent of the light source, some points on the receiver are completely occluded from the light source, forming the *umbra*, while some points, the so-called *penumbra*, are only partly occluded and thus still reached by some amount of direct light. A commonly made idealization is to neglect the size of the light source, collapsing it to a point. The shadow from such a *point light* does not display a penumbra region, forming a so-called *hard shadow*, as opposed to the *soft shadow* cast by objects illuminated by a light source of non-zero size. As usual, the illumination from multiple light sources can be computed by superimposing the illumination of individual sources.

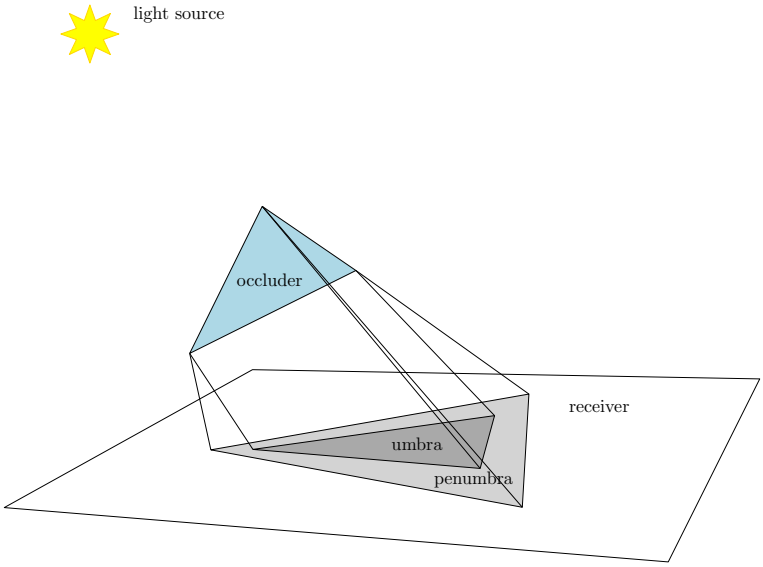


Figure 1.2: The basic problem encountered in shadow rendering.

1.1 Shadow Algorithms

Due to their visual importance, much work has gone into developing algorithms that enable the rendering of shadows in online rendering systems. A good overview of different shadow algorithms is given by Woo *et al.* [WPF90], in the work by Akenine-Möller *et al.* [AHH08] and, of course, in Eisemann *et al.* [Eis+11]. Two principal methods have proven successful for generating shadows in online rendering systems:

- *shadow volumes* [Cro77] and
- *shadow mapping* [Wil78].

Both of these techniques work in two stages. First, a representation of the spatial volume that is in shadow is generated. During rendering, this information is then used to identify parts of surfaces that are in shadow. The difference between both methods lies in the nature of the shadow-representation.

1.1.1 Shadow Volumes

In the shadow volumes approach, a geometric representation of the volume of all points in shadow is constructed. Silhouette edges are identified and then extruded away from the light source, forming a surface that encloses the shadow volume. During rendering, the number of times a viewing ray enters and leaves the shadow volume is counted before it hits a surface. Using this information, it can then be decided whether the surface point is in shadow or not [Cro77]. Stencil buffer techniques can be used to efficiently implement this counting scheme on rasterization hardware [Hei91; EK03] and create a mask for all shadowed regions in the rendered image. Using this mask, the respective regions can then be shaded accordingly, producing pixel-perfect hard shadows.

The most severe disadvantage of this approach is that a potentially very complex *boundary-representation* (BREP) of the shadow volume has to be constructed and rendered. Construction of the shadow volume traditionally had to be performed on the CPU. The introduction of the geometry shader stage made it feasible to create shadow volumes entirely on the GPU. Geometric complexity still is a problem, though, especially since shadow volumes are prone to consist of thin but very long triangles that are inefficient for hardware rasterization. Also, shadow volumes, by their nature, lead to high depth complexity and thus much overdraw, making the method very easily fillrate-bound. Apart from these efficiency issues, the fact that the shadow volume is constructed geometrically means that information on the geometry of all shadow-casting objects is needed. Elements such as billboards made from textured quads are commonly used today, and geometric information on the objects they represent is generally not readily accessible. Therefore, the shadow volumes algorithm is not capable of rendering shadows cast by such objects. Implementations based on the stencil buffer usually rely on all shadow-casting geometry being closed manifolds, posing further restrictions on what is feasible input data. While shadow volumes achieve perfect hard shadows, soft shadows are problematic. Any technique capable of rendering hard shadows can be used to simulate soft shadows by averaging the illumination from multiple images rendered with different light positions sampled from the surface of an area light source [Hei91]. But, in general, many rendering passes are needed for a convincing effect, making such methods too expensive for interactive applications. Other approaches more suitable for real-time use such as *penumbra wedges* [AA02] exist, but have not been widely adopted.

Yet, in its ability to generate perfect hard shadows directly without ray-tracing, the

shadow volumes algorithm is still somewhat unique and thus relevant today. It also enables one to query the exact location where a viewing ray enters the shadow volume and to estimate the distance traveled through the shadow volume. As demonstrated by [BN08], this fact can be exploited, *e.g.*, to generate effects such as crepuscular rays in participating media. Without shadow volumes, one would have to resort, for example, to comparatively expensive ray-marching techniques to achieve a similar effect.

1.1.2 Shadow Mapping

In its essence, shadow computation can be reduced to a visibility problem. Visibility is efficiently solved in rasterization-based graphics by means of *depth buffering* [Cat74]. Depth buffering works by storing for each pixel in addition to a color value also the depth of the associated sample. Whenever a new sample maps to the same pixel, its depth is compared to the depth currently in the depth buffer. Only if the new sample is closer to the camera, the pixel is overwritten and the new depth stored in the depth buffer. In this way, correct visibility of opaque primitives can be guaranteed in an online rendering algorithm.

A point is in shadow, if it is occluded from the light source. Or in other words: For a point to be in shadow means for that point to be invisible from the point of view of the light source. This way of looking at the problem leads to the basic idea of shadow mapping.

To compute shadows for a given light source, we first render the scene as seen from the point of view of this light source to fill a depth buffer. This depth buffer is usually referred to as a *shadow map*. When rendering the scene from the point of view of the camera, we can then decide for any given point whether it is in shadow or not by projecting the point onto the shadow map and comparing its depth to the depth stored at the corresponding location in the shadow map. If the point is farther away from the light source than the depth in the shadow map, then it is in shadow, otherwise, it is lit. Figure 1.3 illustrates this idea, Figure 1.4 shows an example scene with shadows generated by shadow mapping and the shadow map used to compute the shadows in the scene.

Among the significant advantages of shadow mapping is that it can make use of the existing rasterization pipeline in basically all of its aspects. Contrary to shadow volumes,

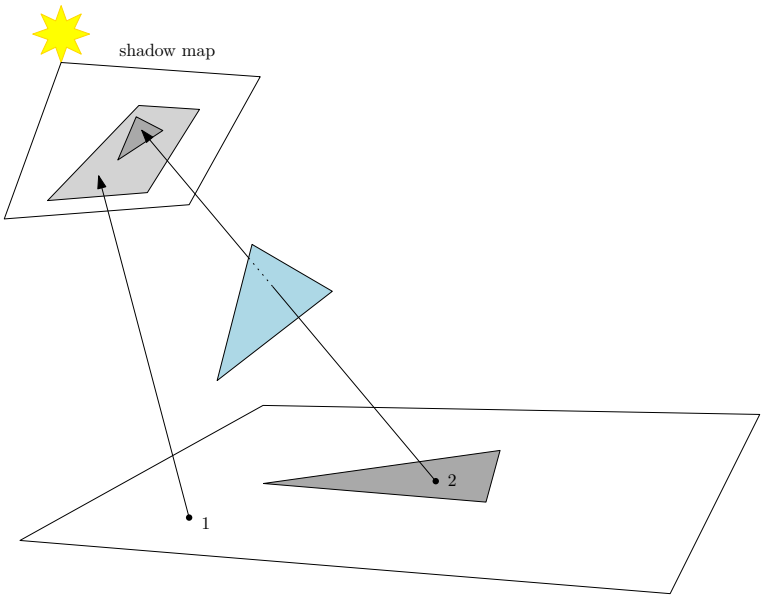


Figure 1.3: The basic shadow mapping algorithm: the scene is first rendered from the point of view of the light source, and scene depth is stored in a shadow map. When rendering the scene from the point of view of the camera, samples (*e.g.*, those marked as 1 and 2) are projected onto the shadow map. By comparing the depth in the shadow map to the depth of the sample as seen by the light source, we can decide whether the sample is in shadow (2) or not (1).

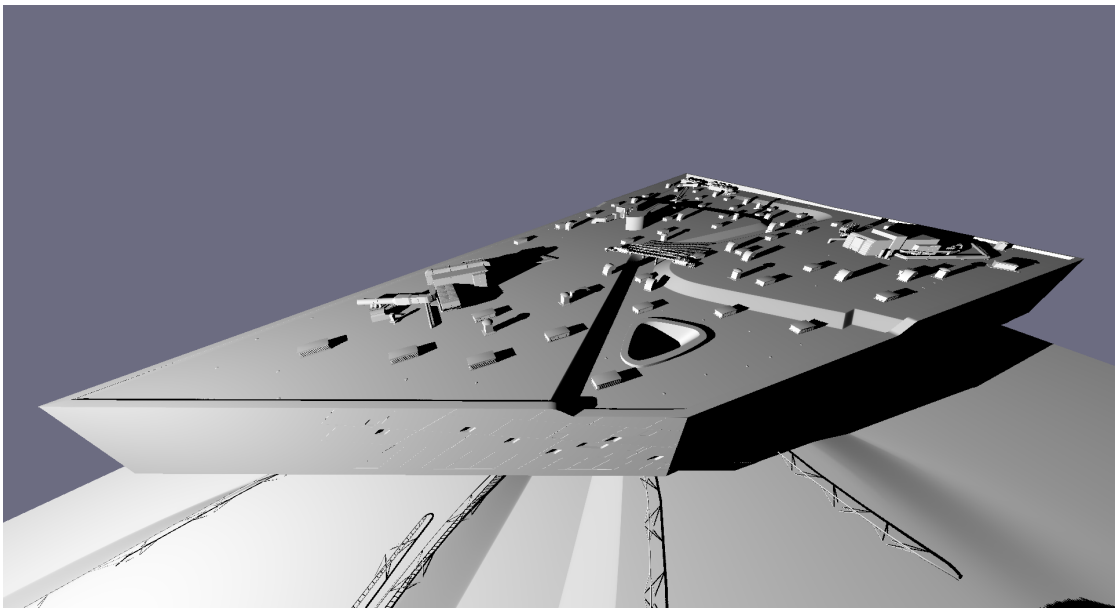
it offers quite predictable performance characteristics, as it is based on simply rendering the scene a second time instead of constructing and rendering additional geometry. Furthermore, there are no restrictions on the contents of the scene. Basically anything that can be rendered can automatically also cast shadows, particularly billboards. Overall, shadow mapping is simple to implement, very general and robust and offers reasonable quality at high performance, making it very attractive for real-time applications, most prominently video games.

Considering the fact that the scene has to be rendered from the point of view of the light source, a first problem becomes apparent: the projections supported by current graphics hardware only allow for a limited field of view. A single shadow map can thus hold information for a spot light or a directional light, but not, *e.g.*, an omnidirectional light. Nonlinear projections such as parabolic mapping can be employed to overcome this issue. While not strictly possible on standard hardware, under certain conditions, an approximation can be sufficient [BAS02]. Also, the views of multiple shadow maps can be combined. [Ger04] present an approach that uses cube textures to implement shadow maps holding a depth buffer for each of the six views along each of the six principal directions.

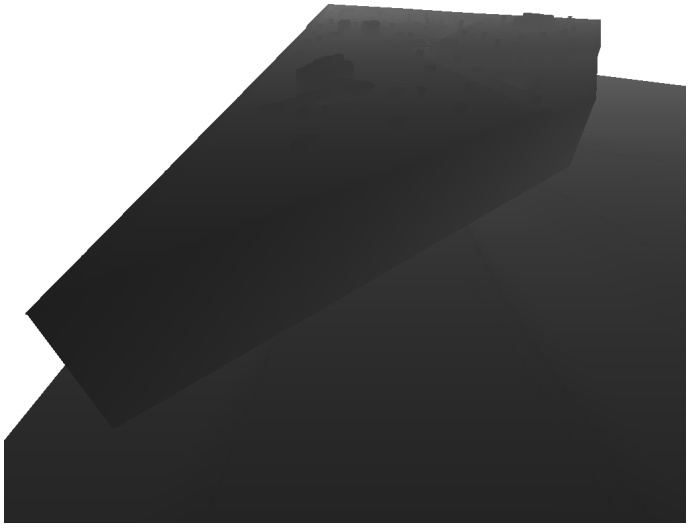
Artifacts

Despite its many advantages, shadow mapping suffers from a couple of problems. Precision issues can lead to erroneous self-shadowing, often simply called *shadow acne*. Figure 1.5a shows an example of the problem. There are a number of potential causes for this effect. When rendering the shadow map, the depth is computed by the rasterization hardware, while the depth used in the shadow test is computed in the shader. These two different arithmetic paths can lead to slightly different results due to quantization errors. Also, values in the depth buffer are generally stored at a different precision than they are computed. As a consequence of both of these issues, the result of the shadow test will fluctuate across the surface that marks as closest in the shadow map, resulting in the moiré patterns seen in Figure 1.5a.

The common solution to these issues is to apply a small *depth bias* either during rasterization of the shadow map or when comparing the depth values. If the bias is too small, shadow acne will continue to show up. A large bias is typically needed to sufficiently

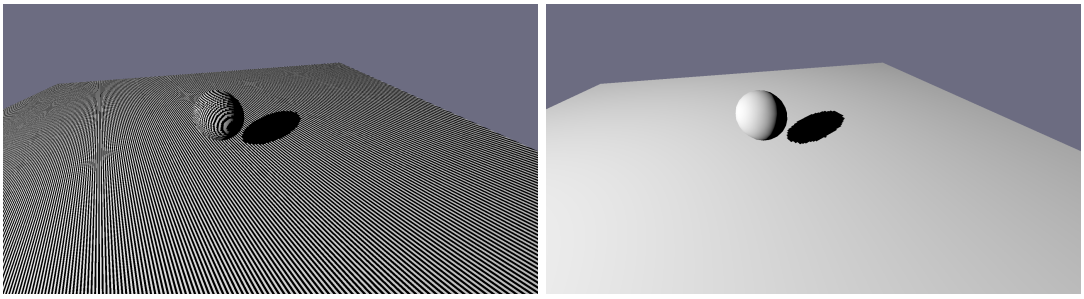


(a)



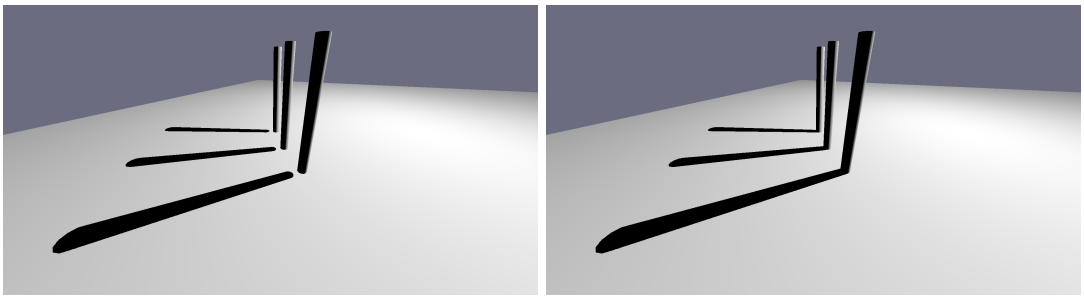
(b)

Figure 1.4: A rooftop scene with complex shadows.



(a) (b)

Figure 1.5: (a) Incorrect self shadowing due to precision issues. (b) By applying slope-scaled depth bias, the problem can be solved.



(a) (b)

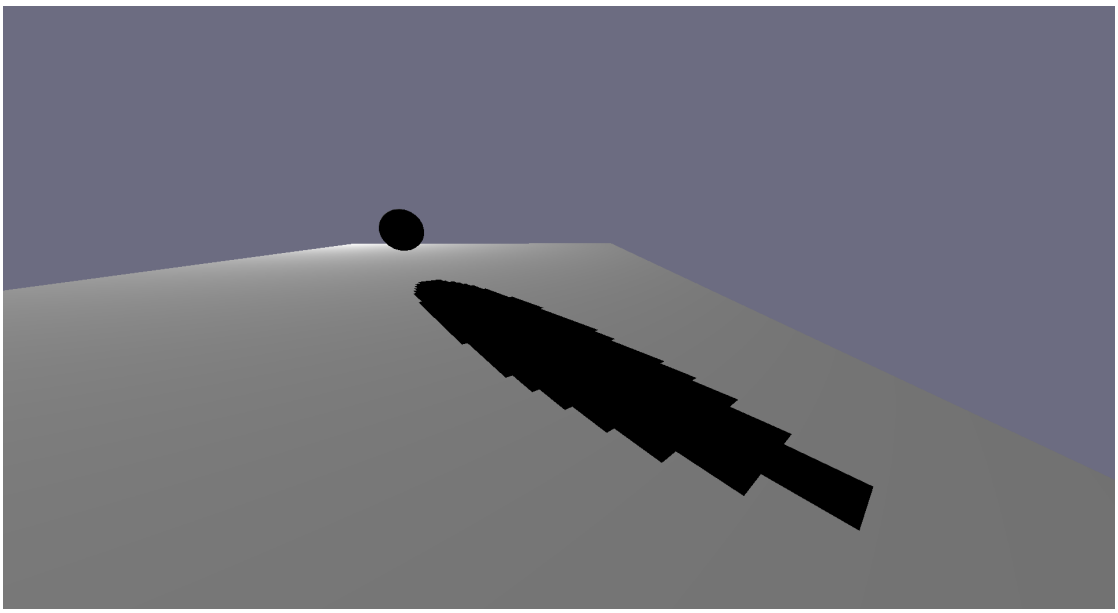
Figure 1.6: (a) Peter panning effect caused by a large, constant depth bias. (b) The same scene with slope scaled depth bias instead.

suppress shadow acne widely enough. But a large bias will result in another detrimental effect known as *peter panning* (Figure 1.6b) where objects appear to float as contact shadows do not meet up to the contact point anymore, loosing the "grounding" effect that we sought to achieve with shadows. Due to the hyperbolic nature of depth values, it is hard to impossible to find a single bias that would strike a right balance between these two effects. Yet, as can be seen in Figure 1.5b and Figure 1.6b, a *slope-scaled depth bias* can. Instead of using a constant, global bias, we vary the bias based on the slope of the polygon, applying a larger bias on steeper polygons. Since it is generally a very effective tool in preventing such precision artifacts, slope-scaled depth bias is supported directly by graphics hardware today.

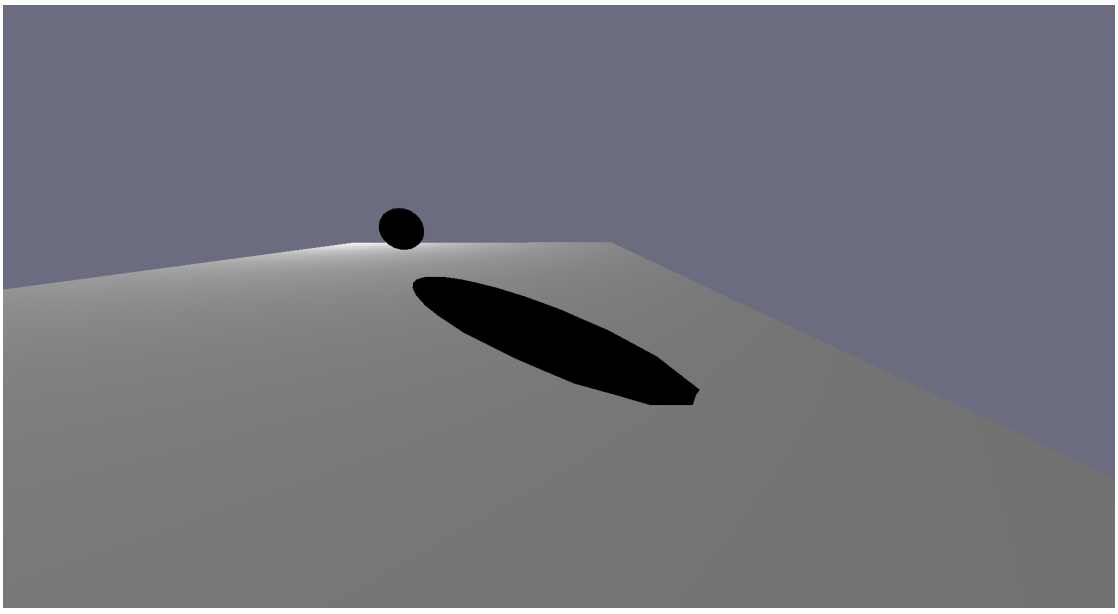
While the sampling-based nature of shadow mapping gives rise to its many advantages, at the same time, it also is the cause for its greatest weakness: *aliasing*. Aliasing is generally an issue in raster graphics, but shadow mapping offers all the ingredients to make the

problem especially severe. The way the scene is sampled when rendering the shadow map from the point of view of the light source and when rendering the frame from the point of view of the camera can differ greatly. Due to perspective and projection, objects taking up many samples in one view might take up very few or even no samples in the other view. Aliasing is caused either by *undersampling* or by *oversampling*. Oversampling happens where more than one shadow map sample correspond to a single sample taken from the viewport of the observer, *i.e.*, the sampling rate in the shadow map is higher than the image sampling rate. This issue is well known from texturing where it is solved, *e.g.*, by mip-mapping [Wil83]. It is usually less of a problem with shadow mapping. The big problem is undersampling. Undersampling occurs when many samples in the current camera view map to a single sample in the shadow map, *i.e.*, the sampling rate in the shadow map is lower than in the image being generated. Figure 1.8 illustrates this phenomenon. It is this form of aliasing that is responsible for the well known "jaggies" as seen in Figure 1.7. An in-depth analysis of aliasing in the context of shadow mapping can be found in [Llo+08].

The purpose of this work is to explore *edge distance shadow mapping* (EDSM), a new method to cope with shadow map aliasing. We extend the basic shadow mapping algorithm in a simple yet powerful way that allows us to combine both, the advantages of the sampling-based approach using fast, depth buffer based visibility queries with the advantages of using geometric features to create mathematically exact shadow borders, but without the massive overhead of actually constructing a full boundary representation for the entire shadow volume.



(a)



(b)

Figure 1.7: (a) Shadow mapping is riddled by aliasing artifacts. (b) Rendering the same scene with shadow volumes yields exact shadow boundaries.

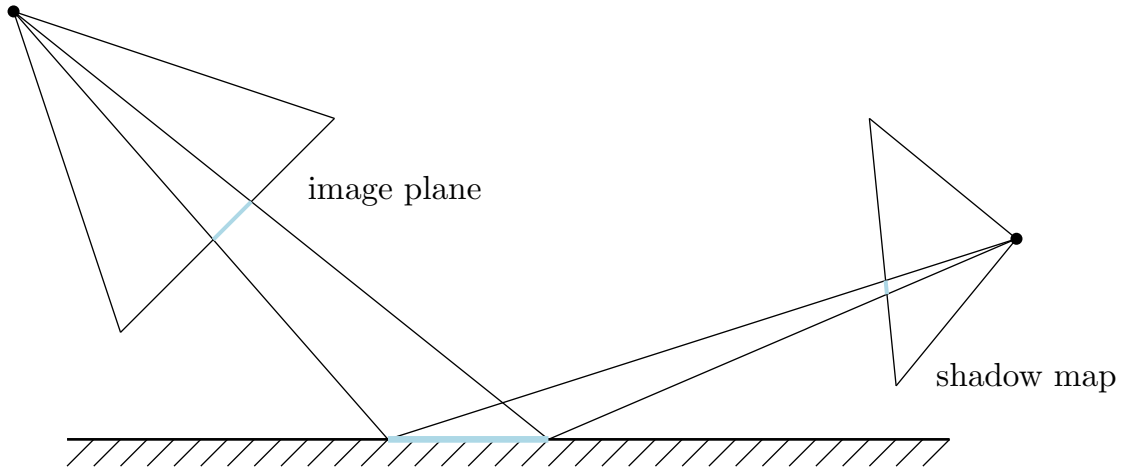


Figure 1.8: Aliasing explained: the blue part of the surface projects to a larger area in the image plane than it does in the shadow map.

2 Related Work

As this work is about shadow mapping, we will first give a short overview on the developments in the field. Shadow mapping was first suggested by Williams [Wil78]. The basic shadow mapping algorithm proposed in this early work is commonly referred to as *simple shadow mapping* (SSM).

2.1 Dealing with Shadow Map Aliasing

Much research has since focused on how to mitigate the effects of aliasing. There are basically two different angles to take on the problem of aliasing in the basic shadow mapping algorithm. One can either perform anti-aliasing on the results of the visibility test by some means of filtering, or optimize the sample distribution in the shadow map to better match spatial sampling frequencies and reduce aliasing in the first place. These two kinds of approaches work independently of each other and thus can be—and usually are—combined for the best results.

2.1.1 Anti-Aliasing

The perhaps oldest approach to provide anti-aliasing for shadow mapping is *percentage closer filtering* (PCF) [RSC87]. The key idea of PCF is to not just use the nearest sample in the shadow map, but to perform the depth comparison for a whole neighborhood and average the results. Due to its simplicity, PCF is a standard algorithm today. In the form of texture comparison sampling, modern graphics hardware even has native support for performing PCF built into its texture units.

The problem with PCF is that the operation gets more and more costly as the filter size is increased. Unfortunately, a large filter size is typically needed to achieve the desired visual quality across a broad range of scenes. More recent approaches have focused on reformulating the shadow test in ways that allow the shadow map itself to be filtered before shadow computation. *Variance shadow maps* (VSM) [DL06] do not store a single depth value per pixel, but rather store the mean and mean square of a distribution of depths. In this way, the variance of the depth distribution can efficiently be computed over any filter region. Using the variance, an upper bound on the fraction representing the occlusion of a shaded fragment is computed. This bound forms a good approximation for the true occlusion. Using simple Gaussian filtering on the variance shadow map, aliasing artifacts are strongly reduced and a reasonable shadow is generated. *Convolution shadow maps* (CSM) [Ann+07] enable the filtering of a shadow map by linearizing the shadow test. While—due to the binary shadow test—traditional shadow mapping is inherently non-linear with respect to the stored depth values, CSMs are not. To get to their formulation of a weighted summation of basis terms for filtering, they assume all receivers to be parallel to the shadow map. As basis function, the Fourier Expansion is used, leading to a multitude of Fourier basis textures representing the convolution shadow map. Instead of linearizing the shadow test using multiple basis functions, *exponential shadow maps* (ESM) [Ann+08] approximate the shadow test using an exponential function. Due to the limited spatial and numerical resolution and due to the applied filtering, the shadow test of ESM might lead to wrong results in some circumstances. To counteract this problem, a manually tuned offset is used. With a well tuned offset, ESM achieves visual quality equivalent to CSM, while being faster and performing better at contact shadows. Compared to VSM, ESM is more forgiving with respect to light leaking.

2.1.2 Optimizing the Sample Distribution

Many algorithms exist which try to warp the projection onto the shadow map such that the distribution of shadow map samples when projected onto the current viewport becomes more uniform. *Perspective shadow maps* (PSM) [SD02] are generated in normalized device coordinate space, *i.e.*, after perspective transformation, and thus distribute samples similar to the main view. *Light space perspective shadow maps* (LiSPSM) solve many of the problems of PSM, like singularities in post-projective space and missed shadow casters, by applying a transformation that allows all lights to be treated as directional lights.

Trapezoidal shadow maps [MT04] address the resolution problem by approximating the eye's frustum as seen from the light with a trapezoid to warp it onto a shadow map. *Logarithmic perspective shadow maps* [Llo+08] combine a perspective projection with a logarithmic transformation to reduce the aliasing error of perspective shadow mapping. *Rectilinear texture warping* [Ros12] approaches the problem of a uniform sample distribution by two 1D warping functions, increasing the resolution of the shadow map based on an analysis in both, the light space and the eye space.

Plural sunlight buffers [Tad+99], *parallel split shadow maps* (PSSM) [Zha+06] and *cascaded shadow maps* (CSM) [Eng06] recognize the fact that, due to perspective foreshortening, shadows closer to the camera will need a higher resolution than shadows farther away. Multiple shadow maps are used for areas of different distance to the viewer. A more recent refinement of these techniques is presented in [Lia+11].

Instead of altering the projection to adjust the sample distribution in a shadow map, placing samples in an irregular manner could result in perfect hard shadows. By transforming the visible pixels from screen space to the image plane of the light source and using these positions as sampling points, one sample is generated for every pixel in the eye's image [AL04]. This strategy can be implemented using an irregular z-buffer [Joh+05]. Using multiple regularly sampled layers, an irregular sampling scheme can be set on graphics hardware [Arv07].

2.1.3 Hybrid Approaches

A third way to cope with aliasing is to apply basic shadow mapping only in areas the algorithm can handle without problems and fall back to a different technique for regions that shadow mapping cannot decide correctly. [CD04] use shadow mapping to quickly determine areas that are completely in shadow or completely lit. In regions where the depth test on the samples surrounding the projected location yields differing results, *i.e.*, areas where a silhouette edge crosses the shadow map, they apply shadow volumes. While such a combination of the two techniques inherits most of the disadvantages of shadow volumes, efficiency problems due to high depth complexity and rasterization loads can somewhat be contained by restricting rasterization of shadow volumes to only small areas of the screen. As such, this method can be viewed as an optimization for shadow volumes.

Similar to our method is *shadow silhouette shadow mapping* (SSSM) [SCH03]. In addition to the depth buffer, SSSM rasterizes silhouette edges into a second buffer, storing the coordinates of a representative point on the silhouette closest to each pixel. During shadow rendering, using the representative points from a neighborhood around each sample location, a piecewise-linear approximation of the true shadow silhouette can be constructed in a shader. To do this, they rely on a scheme similar to dual contouring [Ju+02], exploiting the fact that there is only a limited number of possible configurations in which a contour can pass through a pixel. As long as a sufficient number of samples are present around silhouettes, very high quality shadow boundaries can be reconstructed in this way.

The method we are about to present could also be classified as a hybrid. We also use conventional shadow mapping for fast shading of larger areas completely in shadow or light while at the same time identifying areas where a silhouette edge crosses the image. And we also rely on rasterizing silhouette edges, storing information that allows the each edge to be reconstructed during shadow rendering. But we use a different way of encoding this information. Like Green [Gre07], we employ a signed distance field as our edge representation. In his work, he presents a simple and efficient method for improved rendering of textures containing glyphs. From a high resolution image, a distance field is generated and then stored into a channel of a lower-resolution texture. Using bilinear texture filtering, the distance field is evaluated and improved glyph outlines are drawn. He showed that his approach can vastly improve the quality of textures with vector based content such as text. Similar work has been done by Qin *et al.* [QMK06]. To achieve even higher quality for glyph rendering, they overlay multiple distance fields, one for each line segment of the glyph. While computationally more expensive, this approach does not suffer from artifacts at sharp corners as Greene's does. As our goal is to render shadows for dynamic scenes, precomputation is not an option. We will show how the necessary distance information can be generated in real-time for triangle meshes.

For a more detailed discussion on the state of the art in shadow mapping, the reader is referred to the recent survey on hard shadow mapping algorithms by Scherzer *et al.* [SWP11] or the book *Real-Time Shadows* [Eis+11].

3 Edge Distance Shadow Mapping

Inspired by [Gre07], we propose *edge distance shadow mapping* (EDSM). In addition to the conventional depth buffer, we also encode a mathematical representation of silhouette edges passing through shadow map texels in the shadow map. Based on this encoding, while rendering the shadow, whenever a sample falls onto a location where the simple depth-based visibility test would yield differing results for the surrounding shadow map texels, *i.e.*, it falls in an area where a silhouette edge crosses the image, we can reconstruct the silhouette edge and perform an exact test whether the sample falls inside or outside the silhouette edge. With this method, we can actually get pixel-perfect hard shadows at a very low cost and high performance.

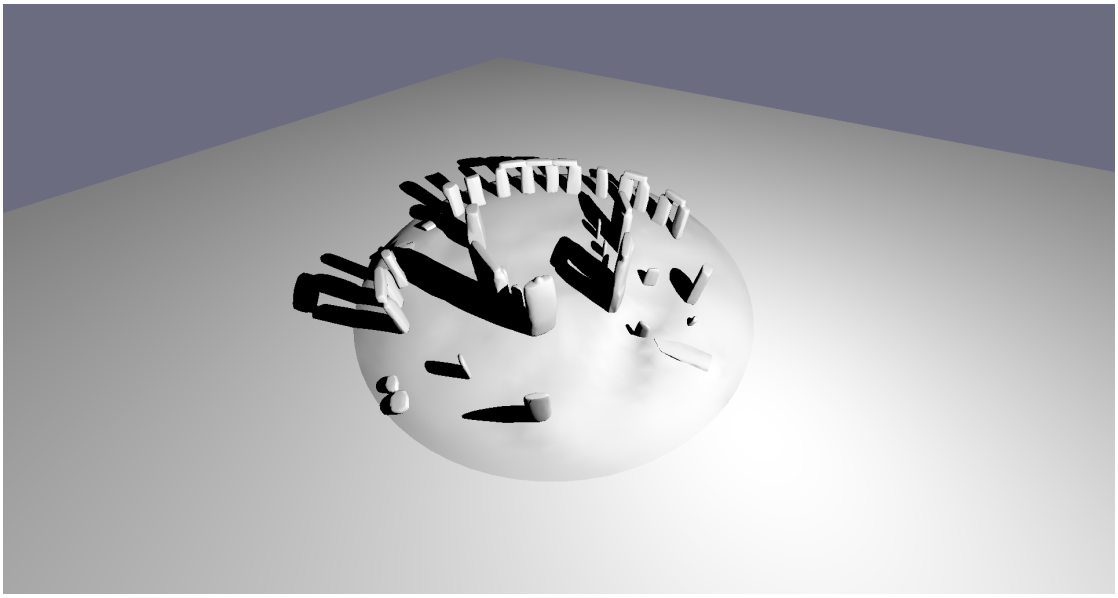
Silhouette information is stored as a distance field that holds in every texel the signed distance to the closest silhouette edge. As we will show, the distance of a point to a straight line can accurately be reconstructed from the sampled distance field by bilinear texture filtering. Given the signed distance of the point, it is trivial to decide whether the point is inside or outside of the silhouette. Figure 3.1 illustrates this idea by giving an example of a scene, the shadow map and the corresponding edge distance map.

3.1 Theoretical foundations

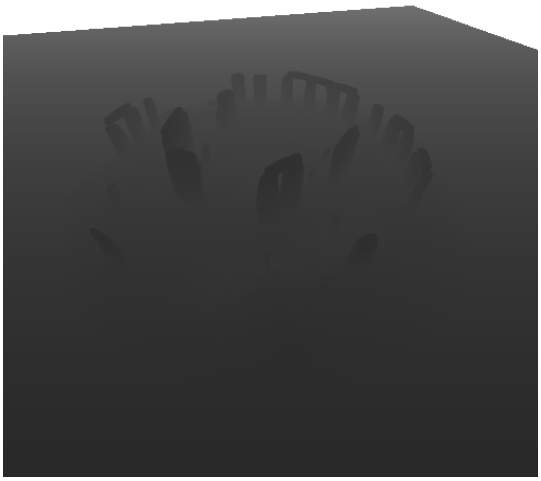
In the two-dimensional affine plane, a line can be defined as the set of all points that satisfy the equation

$$a \cdot x + b \cdot y + c = 0 \tag{3.1}$$

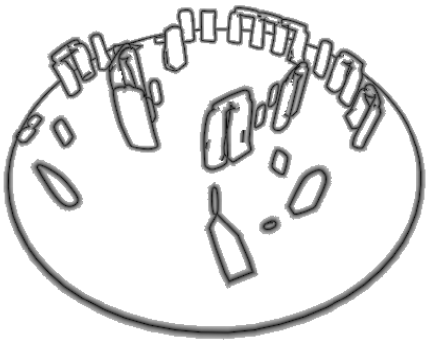
where $(x, y) \in \mathbb{R}^2$ are the coordinates of the point, and $(a, b) \in \mathbb{R}^2 \setminus \{(0, 0)\}$ and $c \in \mathbb{R}$ are the coefficients of the line. Given any point on the line, a second point at offset



(a)



(b)



(c)

Figure 3.1: EDSM.

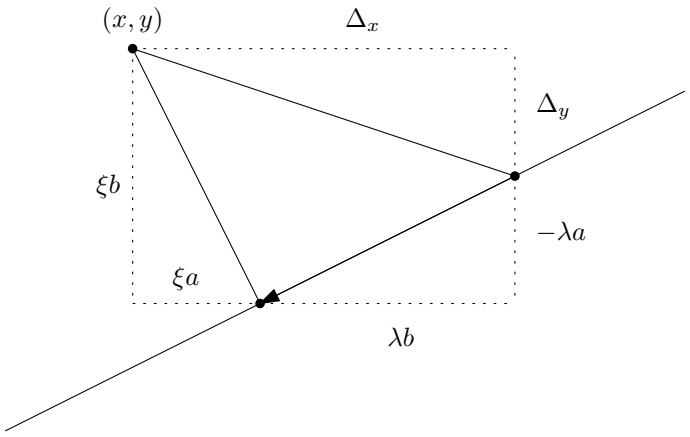


Figure 3.2: The construction leading to our finding that the minimal distance of a point to a line is always measured along the direction orthogonal to the line.

$(\Delta_x, \Delta_y) \in \mathbb{R}^2$ must satisfy

$$\begin{aligned} a \cdot (x + \Delta_x) + b \cdot (y + \Delta_y) + c &= 0 \\ \underbrace{(a \cdot x + b \cdot y + c)}_{=0} + (a \cdot \Delta_x + b \cdot \Delta_y) &= 0 \\ \Leftrightarrow \quad a \cdot \Delta_x &= -b \cdot \Delta_y \end{aligned}$$

to also fall on the line. This underdetermined relation is fulfilled iff $\Delta_x = \lambda \cdot b$ and $\Delta_y = -\lambda \cdot a$ for any $\lambda \in \mathbb{R}$. The conclusion we draw from this finding is twofold. First, given any point on the line, another point on the line can be found along the direction given by $(b, -a)$. Second, for any $\delta > 0$, for any point (x, y) on the line, there are exactly two points at distance δ also on the line, since

$$\delta = \Delta_x^2 + \Delta_y^2 = \lambda^2 (b^2 + a^2)$$

always has exactly two solutions for λ (due to the nontriviality of (a, b)). Thus, not just can other points of the line be found at any distance along direction $(a, -b)$, but only along this direction; a line is a one-dimensional entity of infinite extent. Based on this result, we arrive at the following, parameterized description of a line as the set of all points $\mathbf{x}(\lambda)$ given by

$$\mathbf{x}(\lambda) = \mathbf{p} + \lambda \mathbf{d} \tag{3.2}$$

where $\mathbf{p}$ is a reference point on the line, $\mathbf{d}$ specifies the direction of the line and $\lambda \in \mathbb{R}$ is the line parameter.

Any point $(x, y) \in \mathbb{R}^2$ can be expressed by an offset $(\Delta_x, \Delta_y) \in \mathbb{R}^2$ relative to a point $(x + \Delta_x, y + \Delta_y) \in \mathbb{R}^2$ on the line. As we have just seen, starting at any point on the line with coefficients (a, b, c), another point on the line will be found at an offset $(\lambda b, -\lambda a)$ for any given $\lambda \in \mathbb{R}$. As illustrated in Figure 3.2, the offset of (x, y) from such a new point is given by $(\Delta_x + \lambda b, \Delta_y - \lambda a) \in \mathbb{R}^2$. We can then consider the distance $d(\lambda)$ of the point on the line corresponding to the parameter λ from (x, y):

$$d(\lambda) = (\Delta_x + \lambda b)^2 + (\Delta_y - \lambda a)^2$$

As this is a quadratic function, it has a single global minimum at the point where the first derivative vanishes. The parameter λ that corresponds to the point on the line with minimum distance to (x, y) can thus be found to be:

$$\begin{aligned} \frac{\partial}{\partial \lambda} d(\lambda) = 2b\Delta_x + 2\lambda b^2 - 2a\Delta_y + 2\lambda a^2 &\stackrel{!}{=} 0 \\ \lambda(a^2 + b^2) &= a\Delta_y - b\Delta_x \\ \lambda &= \frac{a\Delta_y - b\Delta_x}{a^2 + b^2} \end{aligned}$$

We notice that for any offset $\Delta_x = \xi \cdot a$ and $\Delta_y = \xi \cdot b$ with $\xi \in \mathbb{R}$, the optimal parameter λ becomes zero. This means that, starting at a point on a line, the distance to any point along the direction (a, b) cannot be decreased by moving along the line in any way. Intuitively, we have now found a geometric interpretation for the coefficients a and b: they correspond to the direction orthogonal to the line, *i.e.*, the coordinates of the *normal vector* of the line.

Finally, if we plug the coordinates of the point at offset $(\xi \cdot a, \xi \cdot b)$ from a point (x, y) on the line back into (3.1), we get:

$$\begin{aligned} a(x + \xi \cdot a) + b(y + \xi \cdot b) + c &= \underbrace{a \cdot x + b \cdot y + c}_{=0} + \xi(a^2 + b^2) \\ &= \xi(a^2 + b^2) \end{aligned}$$

Therefore, plugging the coordinates of any point $(x, y) \in \mathbb{R}^2$ into the function

$$d(x, y) = \frac{a \cdot x + b \cdot y + c}{c^2 + b^2} \tag{3.3}$$

will produce the *signed distance* of that point to the line, *i.e.*, a value whose absolute gives the closest distance of that point to the line and which is positive on one side of the

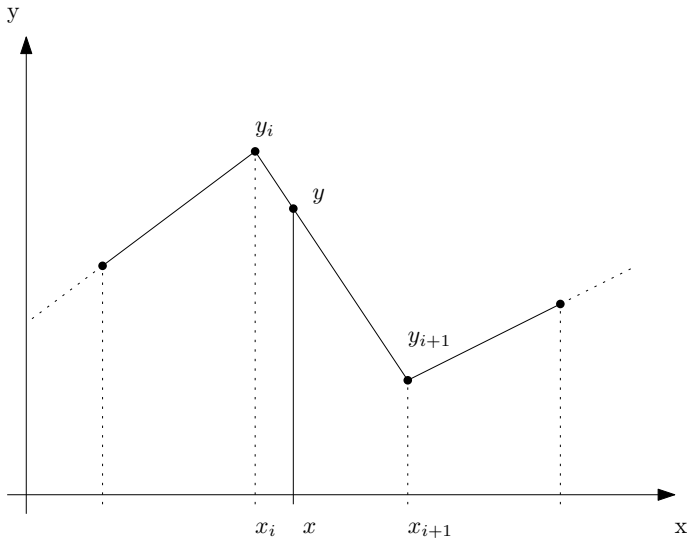


Figure 3.3: Piecewise linear approximation of a continuous signal from its sampled values y_i.

line and negative on the other. Based on the signed distance, the notion of a positive and a negative *half space* can be defined as the set of all points for which the signed distance is positive or negative respectively.

3.1.1 Bilinear Interpolation

A simple and common way to reconstruct a continuous signal from a sampled representation is by *linear interpolation*. The course of the signal between two samples is approximated by a line segment, leading to a *piecewise linear approximation* as illustrated in Figure 3.3.

Given two sample values y_i and y_{i+1} at sample locations x_i and x_{i+1}, we can compute the interpolated value y at a location x between x_i and x_{i+1} as

$$y = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i). \tag{3.4}$$

The idea of linear interpolation can be generalized to two dimensions. We perform linear interpolation first along one dimension and then again interpolate the already

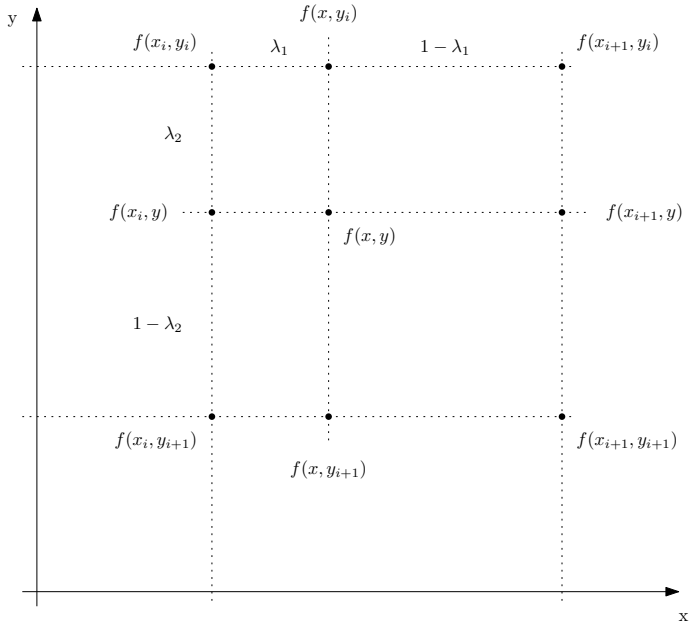


Figure 3.4: Bilinear Interpolation

interpolated values along the other dimension, leading to *bilinear interpolation* (Figure 3.4). By introducing the *interpolation factors* $\lambda_1 = \frac{x - x_i}{x_{i+1} - x_i}$ and $\lambda_2 = \frac{y - y_i}{y_{i+1} - y_i}$, we can write

$$\begin{aligned} f(x,y) &= (1-\lambda_2)((1-\lambda_)f(x_i,y_i) + \lambda_1 f(x_{i+1},y_i)) \\ &\quad + \lambda_2((1-\lambda_1)f(x_i,y_{i+1}) + \lambda_1 f(x_{i+1},y_{i+1})) \\ &= (1-\lambda_1)((1-\lambda_2)f(x_i,y_i) + \lambda_2 f(x_i,y_{i+1})) \\ &\quad + \lambda_1((1-\lambda_2)f(x_{i+1},y_i) + \lambda_2 f(x_{i+1},y_{i+1})) \end{aligned}$$

We can show that for the signed distance function (3.3), the following property holds:

$$\begin{aligned} d((1-\lambda)x_1 + \lambda x_2, y) &= a((1-\lambda)x_1 + \lambda x_2) + by + c \\ &= (1-\lambda)ax_1 + \lambda a x_2 + (1-\lambda+\lambda)by + (1-\lambda+\lambda)c \\ &= (1-\lambda)(ax_1 + by + c) + \lambda(ax_2 + by + c) \\ &= (1-\lambda)d(x_1, y) + \lambda d(x_2, y) \end{aligned}$$

and analogously for y. It follows that, by performing bilinear interpolation, the signed distance to an arbitrary edge can correctly be reconstructed for arbitrary coordinates

correctly be reconstructed from a sampled representation of the distance field.

3.2 Distance map generation

At the core of our algorithm lies the computation of the distance map. A complete distance field for the whole image is not needed because distance information is only used where a silhouette edge crosses the image. Therefore, an edge distance shadow map can be produced by simply rasterizing the edge distance along silhouette edges. To do so, we first identify silhouette edges and then construct a screen aligned quad along each edge that we can use to rasterize the edge distance into our distance map.

3.2.1 Silhouette Detection

Given a manifold triangle mesh, we can determine if an edge is a silhouette edge as seen from a given viewpoint by looking at the projections of the two triangles adjacent to the edge. Let $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \mathbb{R}^3$ be the coordinate vectors of the three vertices of a triangle in the projective plane $\mathbb{P}^2$. The determinant of the matrix

$$\mathbf{M} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix}$$

will yield the signed volume of the tetrahedron formed by the three vertices and the origin. The *winding order* (clockwise or counterclockwise) of the projected triangle vertices is determined by the sign of $\det \mathbf{M}$ [OG97]. If the triangles in the mesh are consistently specified in a way that retains a certain winding order, an edge will be a silhouette edge iff the projections of the vertices of the two triangles sharing the edge appear in different winding orders. Thus, all we have to do to find out if an edge of such a mesh is a silhouette edge is check if the sign of the determinant $\det \mathbf{M}$ for the two triangles joined by that edge is different.

3.2.2 Edge Quad Construction

After a silhouette edge has been identified, we need to construct a screen-aligned quadrangle. To avoid clipping errors, we have to do so in projective space. If we consider our affine plane embedded into a real projective 2-space $\mathbb{P}^2$, according to (3.1), a line in this plane is described by all points $(x, y, w) \in \mathbb{R}^3$ satisfying

$$\begin{aligned} & a \cdot \frac{x}{w} + b \cdot \frac{y}{w} + c = 0 \\ \Leftrightarrow \quad & a \cdot x + b \cdot y + c \cdot w = 0, \end{aligned} \tag{3.5}$$

or in vector notation

$$\mathbf{p} \cdot \mathbf{c} = 0, \tag{3.6}$$

where $\mathbf{p} = \begin{bmatrix} x & y & w \end{bmatrix}^T \in \mathbb{R}^3$ denotes the coordinate vector of the point and $\mathbf{c} = \begin{bmatrix} a & b & c \end{bmatrix}^T \in \mathbb{R}^3$ the vector of line coefficients. We note that we can scale the coefficient vector by any scalar multiple without changing the equation. By solving the system of equations we get from plugging the coordinates $\mathbf{p}_1 = \begin{bmatrix} x_1 & y_1 & w_1 \end{bmatrix}^T, \mathbf{p}_2 = \begin{bmatrix} x_2 & y_2 & w_2 \end{bmatrix}^T \in \mathbb{R}^3$ of two points on the line into (3.6), we can derive the line coefficients for the line passing through these two points:

$$\begin{aligned} & a \cdot x_1 + b \cdot y_1 + c \cdot w_1 = 0 \\ & a \cdot x_2 + b \cdot y_2 + c \cdot w_2 = 0 \\ \Rightarrow \quad & (a \cdot x_1 + b \cdot y_1 + c \cdot w_1) w_2 = (a \cdot x_2 + b \cdot y_2 + c \cdot w_2) w_1 \\ \Leftrightarrow \quad & b(y_1 w_2 - y_2 w_1) = a(x_2 w_1 - x_1 w_2) \end{aligned}$$

Again, this underdetermined relationship is fulfilled iff $a = \lambda(y_1 w_2 - y_2 w_1)$ and $b = \lambda(x_2 w_1 - x_1 w_2)$ for any $\lambda \in \mathbb{R}$. Plugging this result back into one of the initial equations yields the last coefficient:

$$\begin{aligned} \lambda(y_1 w_2 - y_2 w_1) \cdot x_1 + \lambda(x_2 w_1 - x_1 w_2) \cdot y_1 + c \cdot w_1 &= 0 \\ w_1(\lambda x_2 y_1 - \lambda y_2 x_1 + c) + w_2(\lambda y_1 x_1 - \lambda x_1 y_1) &= 0 \\ w_1(\lambda x_2 y_1 - \lambda y_2 x_1 + c) &= 0 \end{aligned}$$

Thus, we find the line coefficients for the line through $\mathbf{p}_1$ and $\mathbf{p}_2$ to be:

$$\mathbf{c} = \lambda \begin{bmatrix} y_1 w_2 - y_2 w_1 \\ x_2 w_1 - x_1 w_2 \\ y_2 x_1 - x_2 y_1 \end{bmatrix}.$$

We notice that the vector part of this expression coincides with the cross product of $\mathbf{p}_1$ and $\mathbf{p}_2$. As noted earlier, we can drop the factor λ, because any scalar multiple of the coefficients defines the same line, leading us to:

$$\mathbf{c} = \mathbf{p}_1 \times \mathbf{p}_2 = \begin{bmatrix} y_1 w_2 - w_1 y_2 \\ w_1 x_2 - x_1 w_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix} \tag{3.7}$$

As we already know, the coefficients of the line correspond to the direction orthogonal to the line. Therefore, after computing the line coefficients as described above, we are already equipped with everything we need to know to calculate the corner points of a quadrilateral enclosing our line segment.

3.2.3 Sorting of Distance Samples

We cannot allow for gaps in the silhouette as they would cause objectionable artifacts. While it would be possible to use a more sophisticated—for example *half-edge*-based—data structure than a simple indexed triangle list to provide the local topology information necessary to generate continuous silhouette geometry, traversing such a complicated data structure for every silhouette vertex in every frame would not be very efficient.

To always ensure connectedness of the silhouette, we construct our edge quads in such a way that quadrangles belonging to edges that are joined at a common vertex overlap at the corner where they meet (see Figure 3.5).

But, while we now have our silhouette edges always surrounded by quads, we need to ensure proper sorting of the distance samples rasterized from these quads. Simply rendering these quads in the order of the primitive stream would result in a distance map such as the one depicted in Figure 3.6. Sorting the primitives according to a global

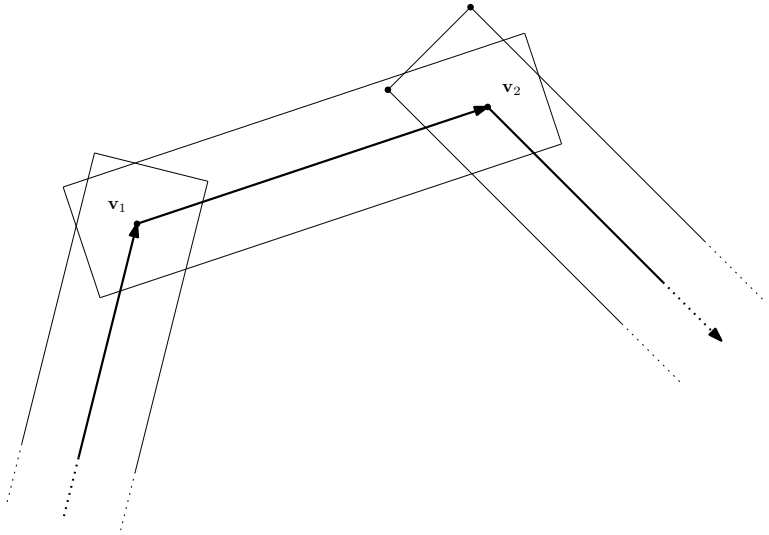


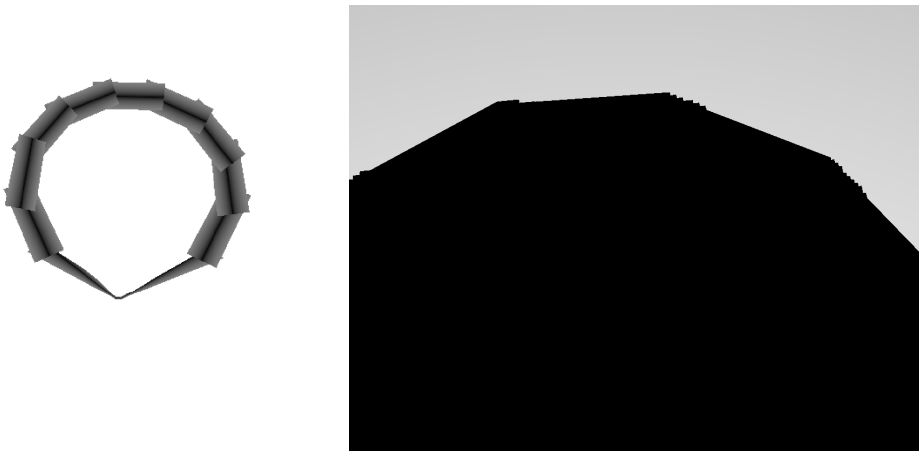
Figure 3.5: Overlapping quads generated along a strip of silhouette edges.

$\min(\cdot)$, while already better, is not successfull either as can be seen in Figure 3.7. Apart from solving the sample order at corners, we will also have to account for unrelated silhouette edges crossing each other in the current viewport.

To still achieve correct ordering of edge distance samples given all these difficulties, we rely on a *corner priority function* f_p:

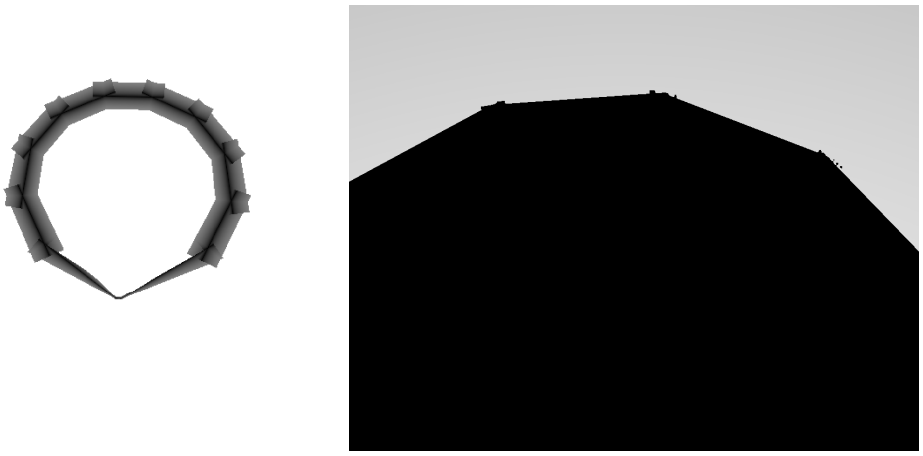
$$f_p(\mathbf{p}) = \begin{cases} |\mathbf{p} \cdot \mathbf{e}| & \text{if } (\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{e} > 0 \\ \max\left(\frac{\mathbf{p}-\mathbf{p}_0}{\|\mathbf{p}-\mathbf{p}_0\|} \cdot \mathbf{e}, |\mathbf{p} \cdot \mathbf{e}|\right) & \text{otherwise} \end{cases} \tag{3.8}$$

where $\mathbf{p}$ is the sample location $\mathbf{p}_0$ the coordinates of the corner vertex, $\mathbf{e}$ the vector of line coefficients of the edge and the product $\mathbf{p} \cdot \mathbf{e}$ denotes evaluation of the signed edge distance. Figure 3.8a shows a surface plot of this function for a certain edge direction. We see that the function displays a more spherical shape in close proximity to a silhouette vertex, while retaining the overall V-shape of the edge distance along the silhouette border. Sorting fragments by smallest value according to this priority function will yield the desired edge distance map, enabling exact reconstruction of silhouette edges during shadow rendering as seen in Figure 3.9. See also Figure 3.8b for an illustration on how the sorting process works.



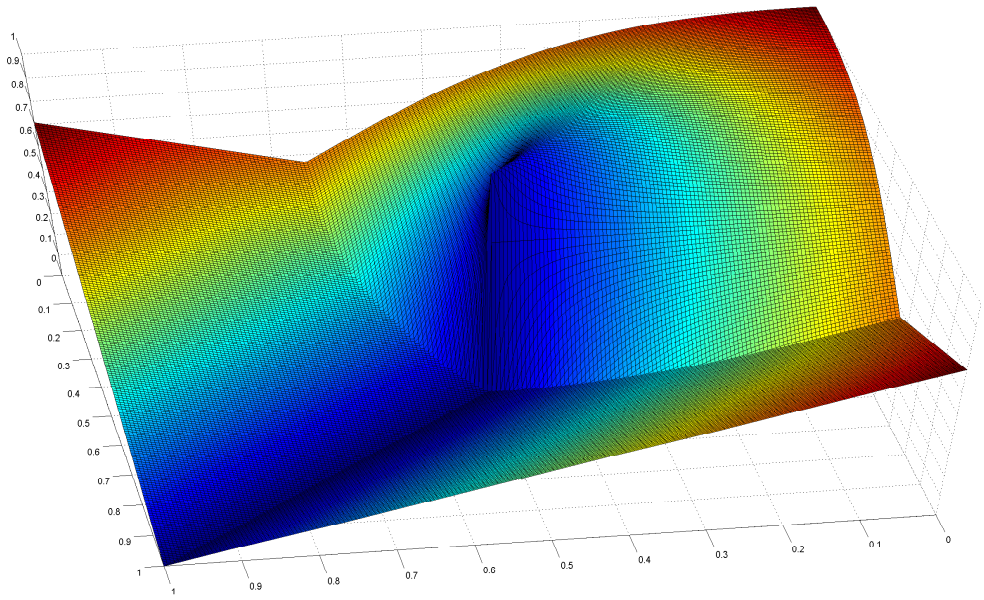
(a) (b)

Figure 3.6: Rendering edge quads in primitive stream order results in an incorrect distance map (a), causing artifacts in the shadows generated based on such a distance map (b).

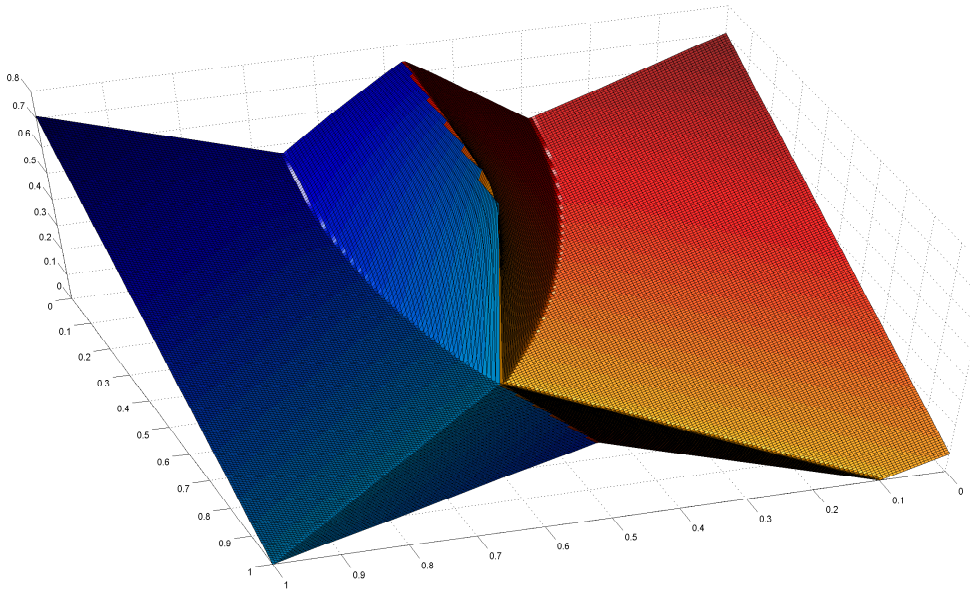


(a) (b)

Figure 3.7: Sorting based on a $\min(\cdot)$ operation yields a much better, but still not the correct distance map (a) and also causes very noticeable artifacts in the shadows (b).

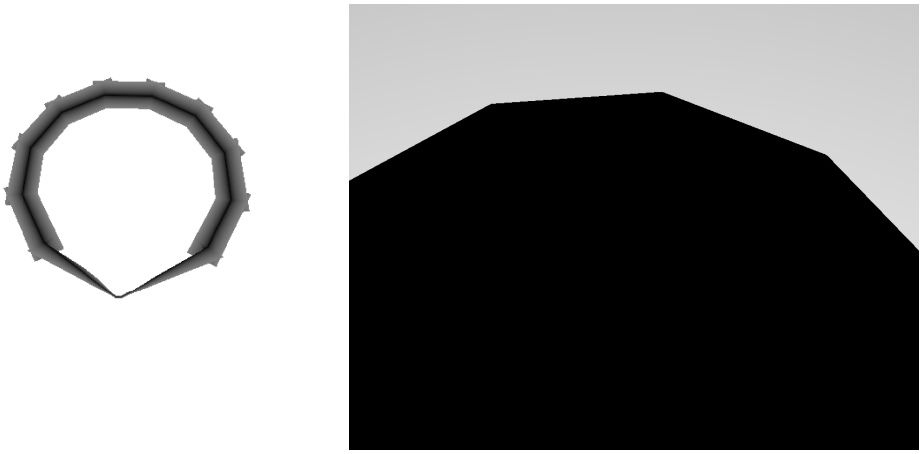


(a)



(b)

Figure 3.8: (a) Corner priority function for a single edge (height marks the function value while the color visualizes the corresponding edge distance). (b) The ordering resulting from performing a min(·) operation on the priority function of two joined edges (one blue, the other one orange).



(a) (b)

Figure 3.9: Sorting according to the lowest value of the priority function finally leads to the desired result.

4 Implementation

4.1 Framework

As part of this thesis, an environment for the development, testing and evaluation of shadow algorithms has been created. It features a build system, a core application, and some utility libraries and tools that make for a very smooth and convenient developer experience and will serve as an excellent base for future work.

Build System The build system utilizes MSBuild and some custom extensions to enable offline compilation of HLSL shader sources as part of the build process and also integrates seamlessly into Visual Studio. Shader binaries created in this build process are linked to and distributed as part of the produced executable images.

Core Application Written for Microsoft Windows in C++ using the Direct3D 11 API, the core application provides basic services such as the main rendering pipeline, scene management and navigation, and a simple graphical user interface. Individual shadow algorithms are implemented in the form of plug-ins that are dynamically loaded by the core application. To facilitate rapid iteration cycles during development, the system supports hot swapping, *i.e.*, individual plug-ins can be modified, recompiled and reloaded all while the application is running.

Utilities The utility libraries provide programming support for interfacing with system APIs as well as basic 2D drawing and font rendering, reading and writing of 2D image file formats, and graphical user interface components. Since, for performance reasons, the core application uses a custom binary scene format, the additional tools are mainly concerned

with reading common 3D file formats and performing the necessary preprocessing to convert data into the custom format.

Rendering Pipeline The rendering pipeline in our application is structured into two stages. First, the current shadow algorithm is run. Shadow algorithms get a triangle mesh as input and render into a separate buffer the amount of light that reaches every pixel visible in the current view. This buffer is then used in the second stage to produce the final, shaded image. While for algorithms such as shadow mapping that could partially be incorporated directly into the shader used to render the shadow receiver, this approach introduces an additional rendering pass that could, in many circumstances, be avoided, it has a some advantages in our application. First and foremost, the shadow computation is thus decoupled from the rest of the application. Since we want to be able to plug any shadow algorithm into the application, we need a system that enables the shadow algorithm to vary freely and independently of the rest of the application—a feat that would be difficult to achieve with a less general approach. Second, it enables us to measure how long it takes for a particular algorithm to just generate shadows. Triangle meshes serve as input for the shadow algorithms The meshes we use are all manifold, since some algorithms, *e.g.*, shadow volumes, only work for such topologies. Data is presented in the form of indexed triangle lists with adjacency information; the latter is needed by algorithms such as EDSM or shadow volumes that have to perform silhouette detection.

4.2 EDSM Implementation

For this work, we implemented the EDSM algorithm on triangle meshes. The edge distance shadow map is rendered in two passes. First, a conventional depth buffer is rendered. To prevent incorrect self shadowing, we apply a slope scaled depth bias.

4.2.1 Distance Map Generation

To generate the distance map, we use a geometry shader to spawn screen-aligned quads along triangle edges. These quads are then rendered using a pixel shader that calculates

the distance of the interpolated fragment position to the corresponding silhouette edge and writes it into the render target.

While we could simply rasterize the edge distance for all edges, to keep the number of quads as low as possible, we generate quads only for silhouette edges. Silhouette detection is performed in projective space as described in section 3.2.1. As the geometry shader will be called for all primitives, we also perform backface culling to ensure that only one quad is generated per silhouette edge. For increased parallelism, we also make use of geometry shader instancing to launch one instance of the geometry shader for each triangle edge instead of handling the individual edges sequentially. We found that this can slightly increase performance in geometry-heavy scenes.

Our initial idea of simply exploiting the blending functionality of the output merger stage to apply a global $\min(\cdot)$ operation on all fragments as a cheap and efficient way of determining the distance to the silhouette edge closest at each pixel turned out not to be fruitful. A big part of the problem owes to the fact that we always have to maintain connected silhouettes. The edge distance of the quads necessarily overlapping each other at corners cannot be decided by a simple $\min(\cdot)$ operation. Thus, we resort to the strategy described in section 3.2.3. We implement the fragment sorting based on our priority function using depth buffering. The priority function is computed at the fragment level and the value written to the fragment depth. Depth buffering will then ensure that the fragment with the minimum priority value ends up in the distance map. Figure **??** illustrates the importance of the issue. Since we already rely on depth buffering to solve the ordering of fragments at corners, we cannot also use it to ensure correct ordering of overlapping edges. Therefore, we sample the depth buffer that has already been created in the first pass and issue a fragment kill if the edge would fail the depth test to take care of overlapping edges and remove occluded edges.

4.2.2 Shadow Computation

When rendering the scene from the current point of view, we now use the depth map and distance map generated in the previous pass. We project points onto the shadow map as usual. To decide whether the projected position is completely in shadow, completely outside of the shadow or falls on a silhouette edge, we exploit texture comparison filtering. Texture comparison filtering will return a value of 1 in areas where all neighboring

samples pass the depth test, 0 where all of then fail the test and a value between 0 and 1 if the sample falls on a silhouette edge, thus providing a quick, hardware accelerated neighborhood check. In the case that the sample falls on a silhouette edge, we further sample the distance map using bilinear texture filtering to reconstruct the signed edge distance at the current position.

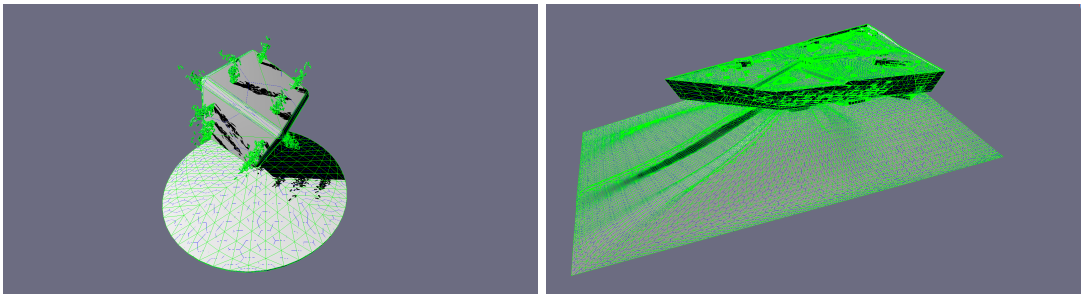
5 Results

For comparison with our method, we also implemented simple shadow mapping (SSM), shadow mapping with percentage closer filtering (PCF), shadow silhouette maps (SSSM), and shadow volumes (SV). Just like in our EDSM implementation, we use the geometry shader stage to construct the edge quadrilaterals for the SSSM algorithm. Texture gather operations allow us to further optimize some of the more expensive texture lookups found in SSSM on modern hardware. We also use geometry shaders to construct the shadow volume in the SV algorithm directly on the GPU. Our shadow volumes implementation is based on the *z-fail method* (also known as "Carmack's reverse"), and we make use of homogeneous coordinates to project and rasterize the necessary *back-cap* at infinity as described in [EK03].

Experiments were conducted on a NVIDIA GeForce GTX 560 Ti graphics card in a host PC driven by an Intel Core i7-2600K CPU running at $3.4GHz$ with $8GB$ RAM. Each measurement represents a sample after a burn-in period of 100 frames; data was collected using Direct3D 11 queries.

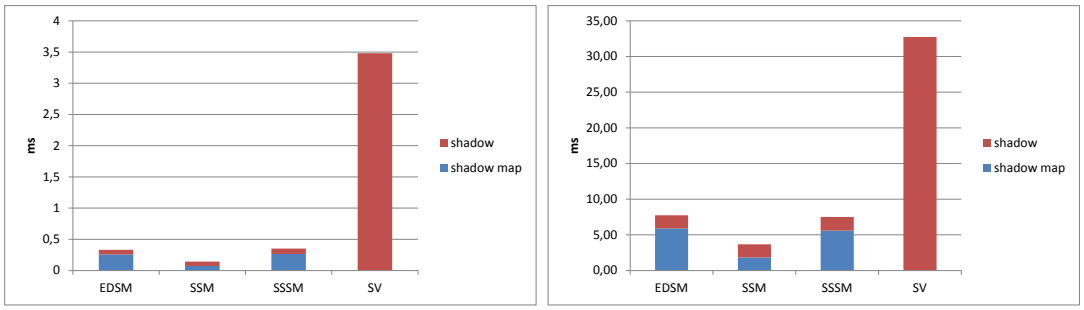
5.1 Performance

Two different scenes were used to evaluate algorithm performance. Both of them are depicted in Figure 5.1. The *jenga* scene consists of 17848 triangles while the *phaeno* scene consists of 505270 triangles. These two different setups were chosen to investigate the influence of geometric complexity on the overall result. As we can see in Figure 5.2, shadow volumes are by far the slowest method in both scenes. EDSM and SSSM generally show the same performance. As we would expect, SSM is the fastest method, but EDSM and SSSM are still comparatively cheap when we consider shadow volumes. When comparing the number of primitives processed during the shadow rendering (Figure



(a) Jenga (b) Phaeno

Figure 5.1: The test scenes used in our evaluation. The jenga scene (a) consists of 17848 triangles, the phaeno scene (b) of 505270 triangles.



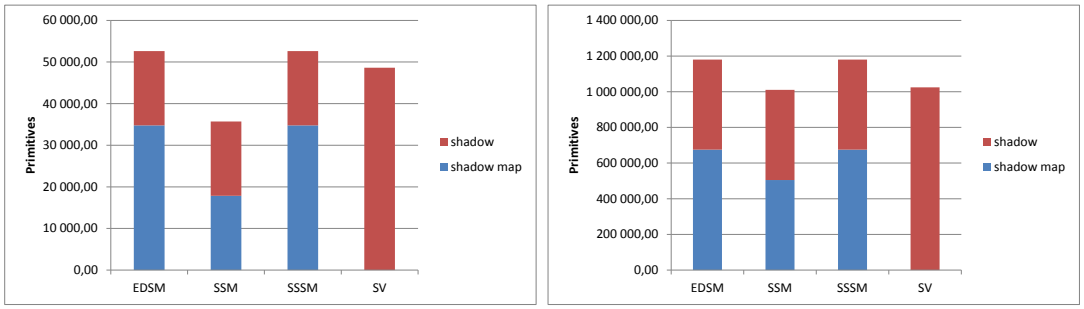
(a) jenga (b) phaeno

Figure 5.2: Time needed by different algorithms to update the shadow map and render a shadow.

5.3), the extreme additional geometric complexity due to the shadow volumes becomes apparent. Keep in mind that EDSM and SSSM both need two passes to generate their shadow map. The number of pixel shader invocations (Figure 5.4) can serve as a rough estimate for fillrate demands, but one has to be careful as, due to optimizations such as *early depth culling*, the real overhead of high overdraw is not correctly represented. Also, SSM does not need a pixel shader to be run during shadow map generation, as it can simply use the depth output by the rasterizer directly.

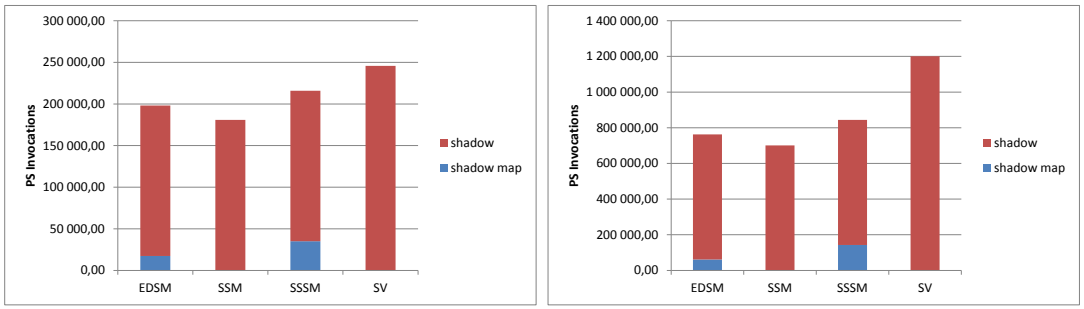
5.2 Visual Quality

Figure 5.5 compares the visual results of our method with SSM and hardware PCF in a situation of extreme aliasing. While the other techniques display badly jagged edges,



(a) jenga (b) phaeno

Figure 5.3: Number of primitives that had to be processed during different stages of the algorithms investigated in our evaluation.



(a) jenga (b) phaeno

Figure 5.4: Number of pixel shader invocations as a rough estimate of rasterization pressure.

EDSM is still able to reconstruct an exact hard shadow. SSSM yields almost identical results and suffers from the same kinds of aliasing artifacts as EDSM.

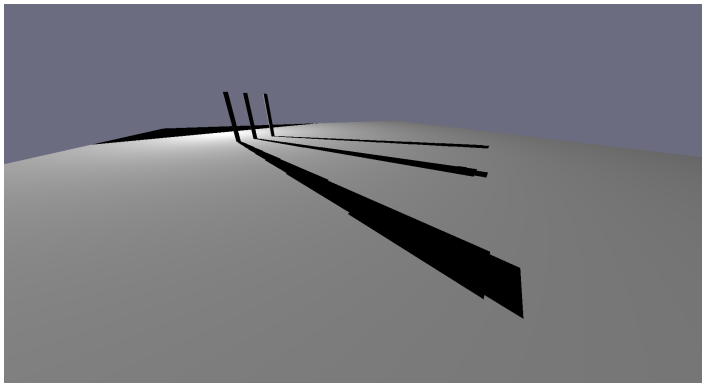
5.2.1 Artifacts

Since we rely on sampling, our algorithm is also affected by aliasing. But the effects of aliasing manifest differently and, potentially, in visually less severe ways compared to conventional shadow mapping.

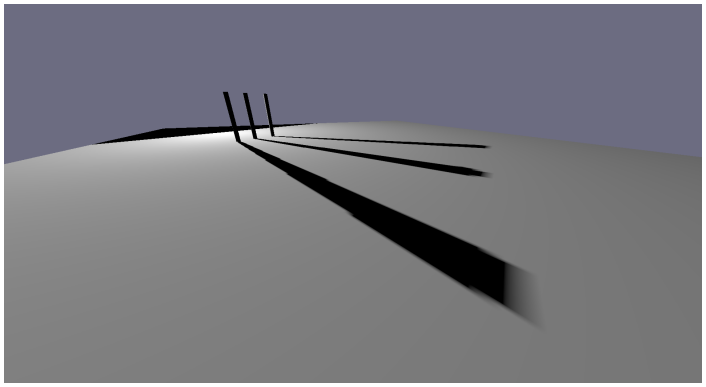
Due to the way we construct our edge quads, they undergo perspective foreshortening not only in longitudinal, but also in lateral direction. Perspective aliasing causes less and less edge distance samples to be available in the neighborhood of distant silhouette edges until edge reconstruction breaks down due to insufficient information (Figure 5.6). Constructing the quads in post-projective space would help to mitigate this problem, but give rise to other problems like handling primitive clipping correctly. Foreshortening in longitudinal direction leads to the same aliasing problems and naturally cannot be avoided. Level of detail methods would, for example, be needed to cope with this kind of aliasing by adapting the scene geometry to the viewport of the light.

Apart from aliasing, the algorithm produces artifacts at sharp corners as depicted in Figure 5.7. Since $\min(\cdot)$ is a nonlinear operator, our assumptions that allowed for edge distance to be reconstructed by bilinear filtering do not hold around areas where the distance fields of multiple edges meet. To avoid this problem, instead of computing a combined minimum distance field, one would have to store the distance fields of all the individual edges, interpolate and perform the inside/outside check separately, and then combine the results, *e.g.*, by a logical AND.

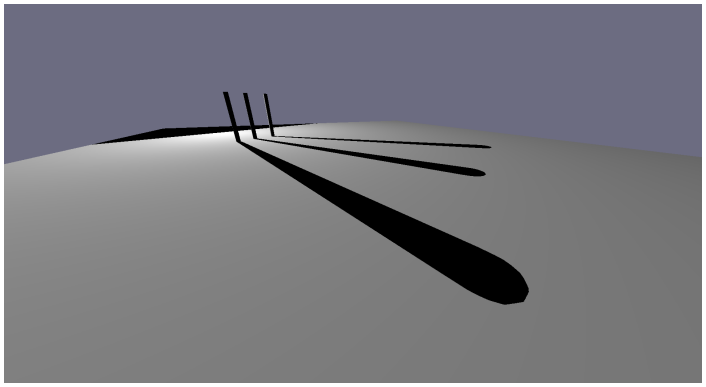
We find it interesting to note that perspective aliasing can lead to such nonlinearities too. Perspective causes edges to move closer to one another as the move away from the viewer. Once they get so close that there are less than the necessary two pixels around each edge, the reconstruction by bilinear filtering breaks down again (Figure 5.8).



(a) SSM

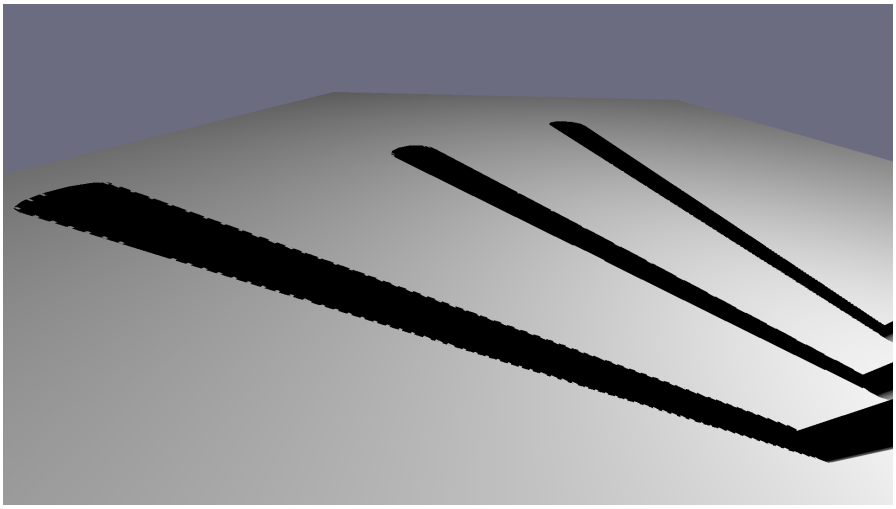


(b) PCF

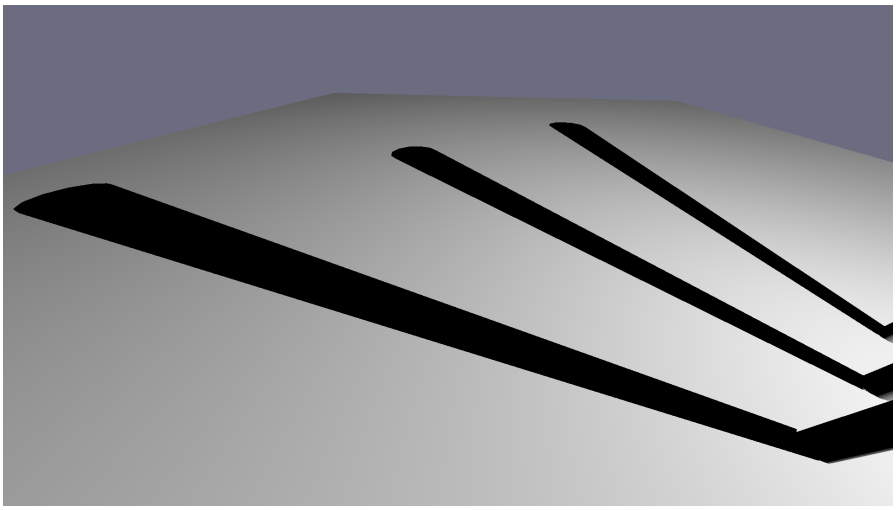


(c) EDSM

Figure 5.5: Quality Comparison

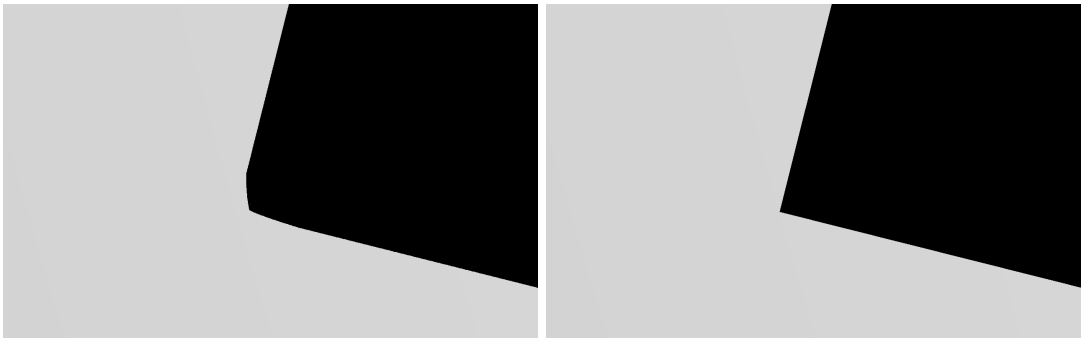


(a)



(b)

Figure 5.6: The effects of undersampling in EDSM.



(a) EDSM (b) ground truth

Figure 5.7: Corner nonlinearity.

(a) EDSM (b) ground truth

Figure 5.8: Overlapping edge nonlinearity.

6 Conclusion

The purpose of this work was to serve as a proof of concept to see if the EDSM algorithm could work and future research be justified. We were able to show that our algorithm can compete with other current methods to generate hard shadows in rasterization-based rendering systems. It is able to provide pixel perfect hard shadows at a low additional cost compared to standard shadow mapping and runs significantly faster than shadow volumes.

Our algorithm depends on an edge distance map from the point of view of the light source, which can, as demonstrated in this work, efficiently be produced on current graphics hardware. The method we have shown here needs adjacency information on the input geometry so that silhouette detection can be performed. Shadow volumes generally require all input geometry to be closed manifolds, while standard shadow mapping imposes, no such restrictions on the input geometry at all. It is worth noting that silhouette detection in our implementation purely serves as an optimization step and is not strictly necessary. Also, contrary to shadow volumes, our approach can work for object representations of non-geometric nature, as long as a concept of edge distance can somehow be defined and an edge distance map computed.

While our algorithm performed similar compared to shadow silhouette maps in the test scenarios we used, we would like to point out a number of advantages of our approach. First and foremost, we exploit texture hardware to perform edge reconstruction, whereas SSSM relies on a more complicated dual contouring approach. Second, the distance to the closest edge is computed as a side effect of the edge representation we use. This can be used to generate plausible soft shadows or at least smoothen shadow borders akin to PCF, but by using a kernel function and without performing any additional filtering. Again, our algorithm does not need an explicit edge, only a distance field is needed. Such a distance field could, *e.g.*, be precomputed for a billboard as done in [Gre07] and [QMK06], and as a result, shadows be cast by such scene elements.

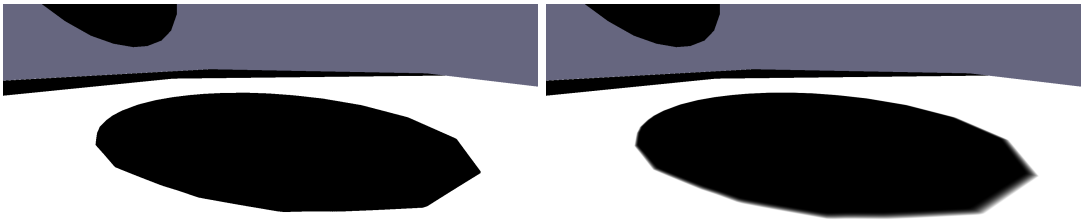


Figure 6.1: Edge distance can be used in EDSM to generate soft shadow boundaries without performing any expensive filtering save for the single, bilinear interpolation used to reconstruct edge distance.

Using a kernel function that models the shape of the light source, mapping edge distance and location to illumination strength, very cheap yet plausible soft shadows similar to smoothies [CD03] become a possibility (Figure 6.1).

Another important aspect of EDSM is its potential for simple and efficient hardware implementation. Actually, due to the way rasterization is usually implemented on current graphics hardware, edge distance already has to be computed during rendering anyway. The information exists in the GPU, it is simply not exposed to the user. We think that having access to the signed edge distance at the fragment level would prove to be of great value not only to us. Great interest seems to have sparked around custom screen-space anti-aliasing techniques lately and having access to this kind of information, especially as it would come at almost no cost, would appear to be very useful in general. Together with an option for *conservative rasterization*—another feature that is often called for, see for example [HAO05]—EDSM could already be supported almost entirely by graphics hardware. The final piece in the puzzle would be some support for ensuring correct ordering of distance samples. Fully programmable blending is nowadays available at least on some hardware, but even, *e.g.*, a pre-blend modifier that would enable the use of the absolute value of its argument in blending operations might already be sufficient.

6.1 Future Work

While the algorithm looks promising, much work is still to be done. More clever ways of sorting fragments during distance map generation are needed to cope with artifacts caused

by incorrect ordering. A ground-truth method of the algorithm could be implemented based on fragment-level linked-lists as presented in [Yan+10]. Using more than one channel to store the distance to more than one edge could yield a sufficient and still very fast approximation of this ground truth algorithm.

To improve performance, it would be desirable to find a technique that allows for rendering the depth and distance maps in a single pass. The render target array feature of modern graphics might be able to give us a way to achieve this goal.

It seems possible to devise a scheme that allows for hardware tessellation to be used instead of the geometry shader, which might perform better under higher geometry loads.

The use of the hardware rasterizer might actually not be the most efficient way to create the distance map. Performing silhouette detection and especially rasterizing depth using quads along silhouette edges is a fast way to map our algorithm to current graphics hardware. Alternatively, the general purpose computation facilities of the GPU could be utilized either through compute shaders or the use of more powerful APIs such as OpenCL or CUDA. Distance map rasterization could thus be implemented in software on the GPU. A specialized software approach might even prove superior. Small, elongated triangles do not pose the most efficient workload for modern rasterization hardware. Also, the hardware pipeline has to obey primitive ordering, which is not strictly necessary in our use case, because we have to enforce correct ordering at the fragment level anyway. Further, determination of silhouette edges could likely be skipped, as it is just an optimization to keep the number of generated quads as low as possible in our current implementation. This would also lift any remaining restrictions on the input geometry.

Apart from these approaches, hardware multisampling could maybe also be exploited to produce a fast approximation.

A completely different approach might be to use GPU implementations of image-based algorithms that generate distance fields; but since they would have to run on already rasterized output, such ventures will likely prove fruitless, as essential information is already lost at that stage.

Last, but not least, it would be interesting to explore the capabilities to generate plausible

soft shadows based on applying kernel functions to the edge distance in more detail.

List of Figures

Bibliography

[AL04] Timo Aila and Samuli Laine. "Alias-Free Shadow Maps". In: The Eurographics Association, 2004, pp. 161–166. ISBN: 3-905673-12-6. URL: http://diglib.eg.org/EG/DL/WS/EGWR/EGSR04/161-166.pdf.abstract.pdf;internal&action=action.digitallibrary.ShowPaperAbstract.

[AA02] Tomas Akenine-Möller and Ulf Assarsson. "Approximate soft shadows on arbitrary surfaces using penumbra wedges". In: *Proceedings of the 13th Eurographics workshop on Rendering*. EGRW '02. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2002, 297—306. ISBN: 1-58113-534-3. URL: http://dl.acm.org/citation.cfm?id=581896.581935.

[AHH08] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Third Edition*. 3rd ed. AK Peters, July 2008. ISBN: 1568814240.

[Ann+07] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. "Convolution Shadow Maps". In: *Rendering Techniques 2007: Eurographics Symposium on Rendering*. Grenoble, France: Eurographics, 2007, 51—60. ISBN: 978-3-905673-52-4.

[Ann+08] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. "Exponential Shadow Maps". In: *GI '08: Proceedings of graphics interface 2008*. Windsor, Ontario, Canada: Canadian Information Processing Society, 2008, 155—161. ISBN: 978-1-56881-423-0.

[Arv07] Jukka Arvo. "Alias-Free Shadow Maps using Graphics Hardware". In: *Journal of Graphics, GPU, and Game Tools* 12.1 (2007), pp. 47–59. ISSN: 2151-237X. DOI: 10.1080/2151237X.2007.10129231. URL: http://www.tandfonline.com/doi/abs/10.1080/2151237X.2007.10129231.

[BAS02] 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). Ed. by John Vince and Rae Earnshaw. Bradford, UK: Springer, 2002, pp. 397–408. ISBN: 1-85233-654-4.

[BN08] Eric Bruneton and Fabrice Neyret. "Precomputed atmospheric scattering". In: *Proceedings of the Nineteenth Eurographics conference on Rendering*. EGSR'08. Sarajevo, Bosnia and Herzegovina: Eurographics Association, 2008, pp. 1079–1086. DOI: 10.1111/j.1467-8659.2008.01245.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01245.x.

[Cat74] Edwin Earl Catmull. "A subdivision algorithm for computer display of curved surfaces." PhD thesis. The University of Utah, 1974.

[CD03] Eric Chan and Frédo Durand. "Rendering fake soft shadows with smoothies". In: *Proceedings of the 14th Eurographics workshop on Rendering*. EGRW '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, 208—218. ISBN: 3-905673-03-7. URL: http://dl.acm.org/citation.cfm?id=882404.882435.

[CD04] Eric Chan and Frédo Durand. "An efficient hybrid shadow rendering algorithm". In: *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques*. EGSR'04. Norrköping, Sweden: Eurographics Association, 2004, pp. 185–195. ISBN: 3-905673-12-6. DOI: 10.2312/EGWR/EGSR04/185-195. URL: http://dx.doi.org/10.2312/EGWR/EGSR04/185-195.

[Cro77] Franklin C. Crow. "Shadow algorithms for computer graphics". In: SIGGRAPH '77. New York, NY, USA: ACM, 1977, 242—248. DOI: 10.1145/563858.563901. URL: http://doi.acm.org/10.1145/563858.563901.

[DL06] William Donnelly and Andrew Lauritzen. "Variance shadow maps". In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. I3D '06. Redwood City, California: ACM, 2006, pp. 161–165. ISBN: 1-59593-295-X. DOI: 10.1145/1111411.1111440. URL: http://doi.acm.org/10.1145/1111411.1111440.

[Eis+11] Elmar Eisemann, Michael Schwarz, Ulf Assarsson, and Michael Wimmer. *Real-Time Shadows*. 1st ed. A K Peters/CRC Press, July 2011. ISBN: 1568814380.

[Eng06] Wolfgang Engel. "Cascaded Shadow Maps". In: *Shader X5: Advanced Rendering Techniques*. Ed. by Wolfgang Engel. Rockland, MA, USA: Charles River Media, Inc., 2006. ISBN: 1584504994.

[EK03] C. Everitt and M. J. Kilgard. “Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering”. In: *ArXiv Computer Science e-prints* (Jan. 2003). eprint: arXiv:cs/0301002.

[Ger04] Philipp S. Gerasimov. “GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics”. In: ed. by Randima Fernando. Pearson Higher Education, 2004. ISBN: 0321228324.

[Gre07] Chris Green. “Improved alpha-tested magnification for vector textures and special effects”. In: *ACM SIGGRAPH 2007 courses*. SIGGRAPH ’07. San Diego, California: ACM, 2007, 9—18. DOI: 10.1145/1281500.1281665. URL: http://doi.acm.org/10.1145/1281500.1281665.

[HAO05] Jon Hasselgren, Tomas Akenine-Möller, and Lennart Ohlsson. “Conservative Rasterization”. In: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Ed. by Matt Pharr and Randima Fernando. Addison-Wesley Professional, 2005. ISBN: 0321335597.

[Hei91] Tim Heidmann. “Real shadows, real time”. In: *Iris Universe* 18 (1991), pp. 28–31.

[Joh+05] Gregory S. Johnson, Juhyun Lee, Christopher A. Burns, and William R. Mark. “The irregular Z-buffer: Hardware acceleration for irregular data structures”. In: *ACM Trans. Graph.* 24.4 (Oct. 2005), 1462—1482. ISSN: 0730-0301. DOI: 10.1145/1095878.1095889. URL: http://doi.acm.org/10.1145/1095878.1095889.

[Ju+02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. “Dual contouring of hermite data”. In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. SIGGRAPH ’02. New York, NY, USA: ACM, 2002, 339—346. ISBN: 1-58113-521-1. DOI: 10.1145/566570.566586. URL: http://doi.acm.org/10.1145/566570.566586.

[Lia+11] Xiao-Hui Liang, Shang Ma, Li-Xia Cen, and Zhuo Yu. “Light Space Cascaded Shadow Maps Algorithm for Real Time Rendering”. English. In: *Journal of Computer Science and Technology* 26.1 (2011), pp. 176–186. ISSN: 1000-9000. DOI: 10.1007/s11390-011-9424-7. URL: http://dx.doi.org/10.1007/s11390-011-9424-7.

[Llo+08] D. Brandon Lloyd, Naga K. Govindaraju, Cory Quammen, Steven E. Molnar, and Dinesh Manocha. “Logarithmic perspective shadow maps”. In: *ACM Trans. Graph.* 27.4 (Nov. 2008), 106:1–106:32. ISSN: 0730-0301. DOI: 10.1145/1409625.1409628. URL: http://doi.acm.org/10.1145/1409625.1409628.

[MT04] Tobias Martin and Tiow-Seng Tan. “Anti-aliasing and Continuity with Trapezoidal Shadow Maps”. In: The Eurographics Association, 2004, pp. 153–160. ISBN: 3-905673-12-6. URL: http://diglib.eg.org/EG/DL/WS/EGWR/EGSR04/153-160.pdf.abstract.pdf;internal&action=action.digitallibrary.ShowPaperAbstract.

[OG97] Marc Olano and Trey Greer. “Triangle scan conversion using 2D homogeneous coordinates”. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. HWWS ’97. Los Angeles, California, USA: ACM, 1997, pp. 89–95. ISBN: 0-89791-961-0. DOI: 10.1145/258694.258723. URL: http://doi.acm.org/10.1145/258694.258723.

[QMK06] Zheng Qin, Michael D. McCool, and Craig S. Kaplan. “Real-time texture-mapped vector glyphs”. In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. I3D ’06. Redwood City, California: ACM, 2006, pp. 125–132. ISBN: 1-59593-295-X. DOI: 10.1145/1111411.1111433. URL: http://doi.acm.org/10.1145/1111411.1111433.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. “Rendering antialiased shadows with depth maps”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), 283—291. ISSN: 0097-8930. DOI: 10.1145/37402.37435. URL: http://doi.acm.org/10.1145/37402.37435.

[Ros12] Paul Rosen. “Rectilinear texture warping for fast adaptive shadow mapping”. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’12. New York, NY, USA: ACM, 2012, 151—158. ISBN: 978-1-4503-1194-6. DOI: 10.1145/2159616.2159641. URL: http://doi.acm.org/10.1145/2159616.2159641.

[SWP11] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. “A Survey of Real-Time Hard Shadow Mapping Methods”. In: *Computer Graphics Forum* 30.1 (2011), pp. 169–186. ISSN: 1467-8659. DOI: 10.1111/j.1467-8659.2010.01841.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2010.01841.x.

[SCH03] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. “Shadow silhouette maps”. In: *ACM Trans. Graph.* 22.3 (July 2003), 521—526. ISSN: 0730-0301.

DOI: 10.1145/882262.882301. URL: http://doi.acm.org/10.1145/882262.882301.

[SD02] Marc Stamminger and George Drettakis. "Perspective shadow maps". In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '02. New York, NY, USA: ACM, 2002, 557—562. ISBN: 1-58113-521-1. DOI: 10.1145/566570.566616. URL: http://doi.acm.org/10.1145/566570.566616.

[Tad+99] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. "Rendering Optimal Solar Shadows Using Plural Sunlight Depth Buffers". In: *Proceedings of the International Conference on Computer Graphics*. CGI '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 166–. ISBN: 0-7695-0185-0. URL: http://dl.acm.org/citation.cfm?id=792758.793029.

[Wil78] Lance Williams. "Casting curved shadows on curved surfaces". In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '78. New York, NY, USA: ACM, 1978, 270—274. DOI: 10.1145/800248.807402. URL: http://doi.acm.org/10.1145/800248.807402.

[Wil83] Lance Williams. "Pyramidal parametrics". In: *SIGGRAPH Comput. Graph.* 17.3 (July 1983), pp. 1–11. ISSN: 0097-8930. DOI: 10.1145/964967.801126. URL: http://doi.acm.org/10.1145/964967.801126.

[WPF90] A. Woo, P. Poulin, and A. Fournier. "A survey of shadow algorithms". In: *IEEE Computer Graphics and Applications* 10.6 (Nov. 1990), pp. 13–32. ISSN: 0272-1716. DOI: 10.1109/38.62693.

[Yan+10] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. "Real-time concurrent linked list construction on the GPU". In: *Proceedings of the 21st Eurographics conference on Rendering*. EGSR'10. Saarbrücken, Germany: Eurographics Association, 2010, pp. 1297–1304. DOI: 10.1111/j.1467-8659.2010.01725.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2010.01725.x.

[Zha+06] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. "Parallel-split shadow maps for large-scale virtual environments". In: *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*. VRCIA '06. New York, NY, USA: ACM, 2006, 311—318. ISBN: 1-59593-324-7. DOI: 10.1145/1128923.1128975. URL: http://doi.acm.org/10.1145/1128923.1128975.