TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

D I P L O M A R B E I T

Shadow Mapping of Large Environments

ausgeführt am Institut für

Computergraphik und Algorithmen

der Technischen Universität Wien

unter Anleitung von

o. Univ.Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer,
Univ.Ass. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

durch

Daniel Scherzer

Glashüttenstraße, 10
6330 Kufstein

February 16, 2006
Datum

Unterschrift

Abstract

This thesis is about shadow generation in real-time and its problems. Its focus is on *shadow mapping*, a real-time technique to render high quality shadows. A lot of literature is available concerning *shadow mapping* and the problems associated with it. But most of this literature concerns itself only with certain problematic parts of *shadow mapping*, but not with all the problems of *shadow mapping* together. In this thesis we will give a minute report of all these problems and why and how they occur. We will discuss the major methods to cure them and identify and adopt the methods so that they can be used together for demanding real-time applications to avoid all of the *shadow mapping* problems.

All things considered this thesis should provide an insight into the current state of research in the field of real-time shadow generation with *shadow maps*, and should expand this overview by giving the reader all the methods at hand to cure the problems of *shadow mapping* and use it in a complex real-time scenario without visible artefacts.

Kurzfassung

Diese Diplomarbeit ist über die Erzeugung von Schatten in Echtzeit und den dabei auftretenden Problemen verfasst. Das Hauptaugenmerk liegt dabei beim *Shadow Mapping*, einer Echtzeittechnik für die Erzeugung von hochqualitativen Schatten. Eine Menge Literatur, die sich mit *Shadow Mapping* und den dabei auftretenden Problemen beschäftigt, ist vorhanden. Aber der Großteil dieser Literatur beschäftigt sich nur mit einzelnen problematischen Teilen des Verfahrens und schafft deshalb keinen Überblick über alle vorhanden Probleme dieses Ansatzes. In dieser Diplomarbeit berichten wir genau über die Ursache und Wirkung all dieser Probleme. Wir besprechen die wichtigsten Methoden und wir kombinieren diese, damit sie gemeinsam genutzt werden können, um den Anforderungen von Echtzeit-Anwendungen zu entsprechen.

Insgesamt sollte diese Diplomarbeit einen Einblick in den Status quo der Forschung im Bereich der Generierung von Schatten in Echtzeit mit *Shadow Maps* liefern. Zusätzlich sollte diese Arbeit dem Leser die Werkzeuge zur Verfügung stellen, mit denen er die Probleme von *Shadow Mapping* unter komplexen und praktischen Bedingungen, ohne sichtbare Artefakte lösen kann.

Contents

Chapter 1

Overview

- Chapter 2 of this thesis gives an introduction into the field of shadow generation and discusses why shadows are so important for computer generated graphics.

- Chapter 3 will describe the three problems of shadow mapping: Projection aliasing, perspective aliasing and incorrect self-(un) shadowing and illustrates how and why these errors arise.

- Chapter 4 introduces a short overview of methods that cure all three problems at once. Regrettably these methods are not real-time methods. We will also discuss the intricacies of focusing the shadow map on the right volume in space that contains all the necessary shadow casters and receivers. Additionally filtering, as a tool that can help to alleviate all three errors, is explained in this chapter.

- Chapter 5 will elaborate algorithms that can handle incorrect self-(un) shadowing, like biasing.

- Chapter 6 is aimed to resolve the problems with projection aliasing.

- Chapter 7 discusses the minimization of perspective aliasing and some of the hardware accelerated methods derived from the initial paper *Perspective shadow maps* (PSM) by Stamminger and Drettakis [SD02].

- Chapter 8 provides a minute explanation of *Light space perspective shadow maps*, a shadow mapping technique developed by Michael Wimmer and the author [WSP04]. This technique is used to minimize the effects of perspective aliasing.

- Chapter 9 is devoted to the involved implementation of applying shadow maps to large-scale, polygon-rich and dynamic environments, a very demanding setup with many real-world properties found in todays applica-

tions. This implementation uses a combination of solutions to all the described problems. We also state the results achieved by this implementation.

- Chapter 10 summarizes all our efforts and concludes this thesis.

Chapter 2

Introduction

In this chapter we will give an introduction to the field of shadowing with an emphasise on real-time applications. In Section 2.1 we will relate the reasons for using shadows in computer graphics and the role shadows play in computer-generated images (the why). In Section 2.2 we will introduce the basic definitions used in this thesis and explain what shadows are and how they can be generated algorithmically in computer graphics. Finally we provide a brief discussion of the various shadow generation algorithms for real-time purposes in Section 2.3, namely the two most robust, shadow maps and shadow volumes.

2.1 Why shadows?

Shadows give important visual cues for perceiving the geometric relationship between objects. This means a shadow can help to clarify position, size and geometry of the shadow casters and the geometry of the shadow receiver as well as the distance to the light source (see Figure 2.1).

Consider a tree: Without a shadow, the tree just floats above the ground, missing the firm connection to the terrain in reality provided by his shadow. In general the gap between the object and its shadow on the ground plane gives the observer a hint how distant the object is to the shadowed surface. In the case of the ground, this answers the question how high the object is flying above ground or if it is standing on the ground. Additional shadows help to create a certain atmosphere and add immensely to the realism of a scene.

Today the most commonly used form of shadows are hard shadows produced from a directional or point light. This simple model produces so-called hard shadows. In the case of a hard shadow, a point is either in shadow or not. This binary decision produces very noticeable shadow edges, which can make the shadow be falsely perceived as a separate object. In addition the high contrast edges are very noticeable to the human visual apparatus, too. Artefacts are therefore easily noticed.

In contrast, most of the shadows present in nature are very soft and blend

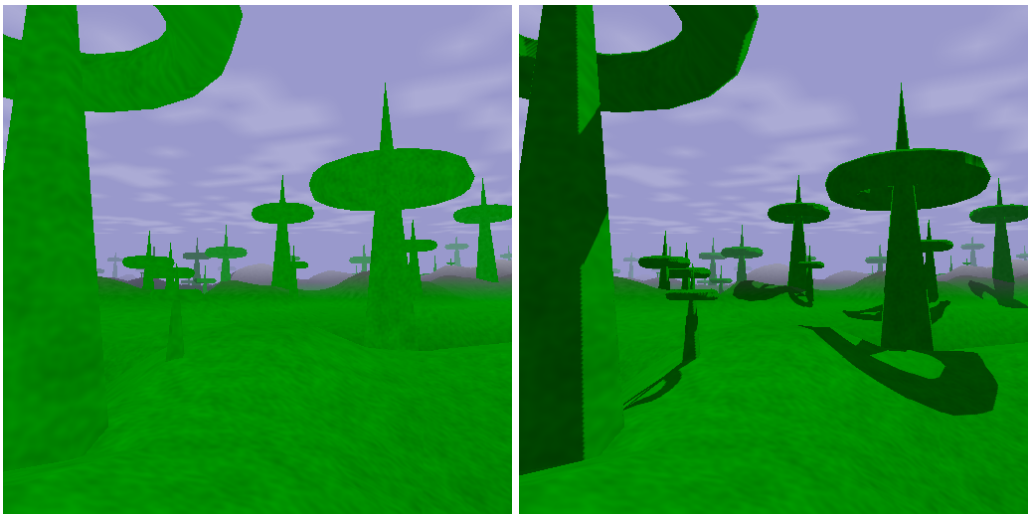

Fig. 2.1: The same image is shown once without and once with a shadow. With the shadow attached the objects interrelationship is far easier to understand.

seamlessly into their environment. These soft shadows are inconspicuous, because they gradually shift their color from a bright one in the lit regions, to the darkened shadowed color. But nevertheless, they influence the realism of the generated image immensely. Definitely more complex to compute than hard shadows, soft shadows are therefore a valuable technique to investigate, and the recent effort to do so can be measured by the large number of published papers on this topic. The used light model for these kinds of shadows are volume or area lights.

An excellent overview of shadow algorithms as well as real-time computer graphics in general can be found in Möller and Haines' Real-Time Rendering book [MH02]. A bulk of information and references can be found in [HLHS03] and on the accompanying web-page.

2.2 The basics

In this section, we introduce the basic definitions we use in this thesis. We introduce the term shadow and the basics of shadow generation.

2.2.1 Real-time vs. interactive frame rates

In future sections we will repeatedly talk about real-time applications, so at first we need to define what real-time means: We define real-time applications as applications that demand at least 60 frames per second as an absolute minimum. Interactive applications on the other side need a frame rate typically situated in

the range of 2-12 frames per second. A frame in this respect is a new rendering of the world, which incorporates a change of the camera, change of the environment and/or change of the lighting situation.

The main focus in this thesis lies on techniques that can be used in real-time applications. Methods such as ray-tracing, photon mapping, radiosity and Monte-Carlo ray-tracing can produce realistic shadows, but none of them is currently capable of delivering these at real-time or even at interactive frame-rates. As a consequence, thess approaches are not handled here in detail.

2.2.2 What are shadows?

A point is considered to be in shadow when this point cannot be seen from the viewpoint of the light source. The object that is the cause for not seeing this point is called the shadow caster, occluder or blocker, which blocks the light rays from reaching the point. The object on which the point in shadow lies is called the shadow receiver.

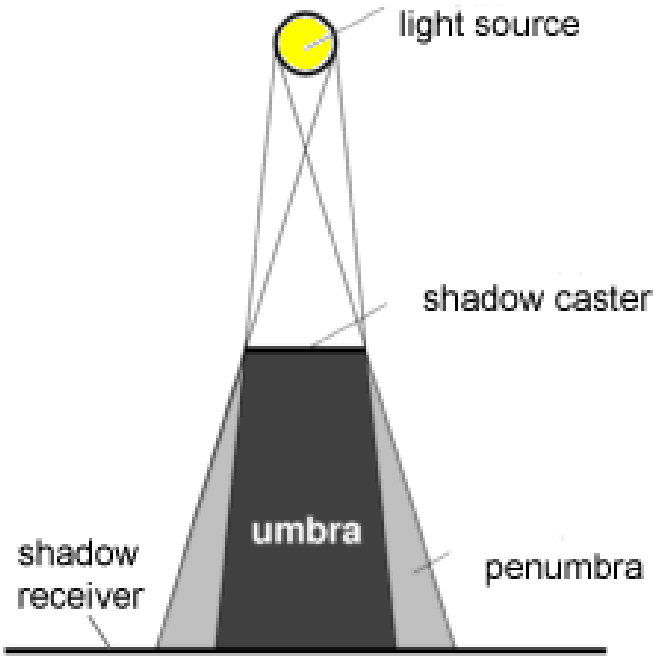

Fig. 2.2: A shadow caster casts an umbra and penumbra region when illuminated by an area light source.

This simple definition holds for point lights. When we consider an area or volume light source it is possible that an object is just partly hidden from the light source. A point in such a partially hidden region lies in the penumbra region of the

shadow. The points which are totally hidden from the light source lie in the umbra of the light source. An example of such a configuration can be found in Figure 2.2. The umbra and penumbra together are called the shadow. The common type of real-time shadows, the hard shadow, has no penumbra region because the light source that generates the shadow has no physical extent. It simply consists of the umbra.

Note that the penumbra region is not just a blurred version of the shadow receiver outline. A soft shadow gets blurrier the farther the shadow caster is away from the shadow receiver, as can be seen in Figure 2.3. The larger the light source gets, the smaller the umbra region becomes, therefore the soft shadow umbra is generally not equivalent to the hard shadow generated by a point light at the center of the light source. The exact determination of the umbra and penumbra region is a 3D visibility problem, which is notoriously hard to solve.

Fig. 2.3: The blurriness of a soft shadow increases the farther the shadow caster and receiver are apart.

A special case occurs when the receiver and the caster are the same object. We use the therm *self-shadowing* for such a constellation. The toes of the left foot in Figure 2.3 are shadowed by the shadow of the right foot. If both feet are considered to be part of one object, this is a case of self-shadowing.

2.2.3 Rendering methods

In this branch of computer graphics a synthetic image of the world is generated through a mathematical description of a scene. The objects inside the scene, the camera, the lighting conditions and the surface properties are described through numbers, and some formulas for animation of these numbers may be given. To produce the final image, consisting of numbers that represent colors, various methods exists, called rendering methods. A rendering method is therefore a mechanism to convert the mathematical description of the scene into color information.

To understand the problems of the different shadow algorithms, we must understand the different rendering methods. Each method introduces its own problems and difficulties for shadows generation. For the sake of simplicity we distinguish two methods of rendering: ray tracing and polygon rasterization.

Ray tracing

Ray tracing has its basics in a simplified model of the physical propagation light. It models laser like light rays passing through the scene, after emanating from the eye. At each intersection of a ray and an object the object properties, like reflectance and transparancy, are considered and the previous ray is ended, but maybe new rays are generated according to the properties of the object.

For ray tracing shadows, are a natural extension of the algorithm. For each intersection of a ray and an object, simply send shadow feelers to each light source in the scene. If there is no intersection on the route to the light source, this light source is visible to this point and therefore the illumination of the point has changed by the influence of this light source.

Ray tracing is a global rendering method. This means that at each point, we need to access all the information of the scene, like visibility, through casting rays. Ray tracing has to be done for each pixel (picture elements) on the cameras image plane. For a viewport of, for example, 800x600 pixels this means 480.000 ray tracing steps, as described above, have to be calculated. This technique is therefore slow and not even suitable for interactive image generation with today's speed of hardware. Nevertheless the generated images are accurate for scenarios with a dominant specular lighting. For diffuse lighting conditions the laser like rays, used in ray tracing, are not valid anymore. In ideally diffuse lighting conditions the incoming radiation is, upon reflection, reemanated equally into all directions. Global illumination methods, like radiosity, provide solutions for such scenarios.

Polygon rasterization

A way to generate the pixels faster is polygon rasterization. For each polygon, the screen space coordinates of its vertices are calculated. The pixels that are

overlapped by the polygon can then be easily determined and are set to a certain color. This color can be calculated through the application of formulas to incorporate the lighting contribution and the various surface properties. However this method doesn't solve the visibility problem. Here a z-buffer can help. For each screen-pixel the depth value of the nearest fragment is stored in this z-buffer. We use the term fragment for a pixel with additional information like depth, normal and possible other data. On rasterizing each fragment, the z-buffer is checked if there is no nearer fragment already at this position. If this is the case, the current fragment is discarded and the fragment already calculated stays put.

Shadow determination would be the responsibility of the formulas used to calculate the lighting contribution. How this is done exactly, without using shadow feelers for each fragment, is the basic of real-time shadow generation, which is desribed in the next section.

2.3 Shadowing in real time

Shadowing is a visibility problem because it is concerned with the question: Is a point visible from the point-of-view of the light? As a visibility problem the range of applicable methods is very wide. We can use 3d, 2.5d as well as 2d methods. We can use rather unintuitive spaces, like line-space or the post-perspective space. We can state the visibility problem in object or in image space. All this is done to make it easier (faster) to solve the problem of visibility because it is extremely difficult to solve fast and accurately.

The fastest methods for shadowing are specialized visibility algorithms. The two methods described in the following are both rather intuitive 3d methods. The first, shadow volumes by Crow [Cro77] is an object-space method, which means that it works with geometrical representations of the objects in the scene to calculate the shadows and therefore gives accurate results. The second method, shadow mapping by Williams [Wil78], on the other hand is an image-space method. This means it works with a two-dimensional image with a finite resolution, to store some values needed for shadow determination. Both can handle scenes with general sets of shadow casters and receivers, including self-shadowing. We will describe these two methods in the next two subsections.

2.3.1 The shadow map algorithm

The use of shadow mapping has gone a long way since its introduction by Williams [Wil78] and its fully hardware-accelerated use today. His basic idea was to render the scene from the point of view of the light into a z-buffer and save the so gathered depth values in the so called *shadow map*. For a point light, these depth

values give the distance each point visible to the light has to the light source. Upon rendering the scene from the eye-point, each pixel is transformed into light space and its depth compared to the depth stored in the shadow map. If the stored depth is nearer to the light, the pixel must be in shadow and the lighting of the pixel can be altered to reflect this (see Figure 2.4).

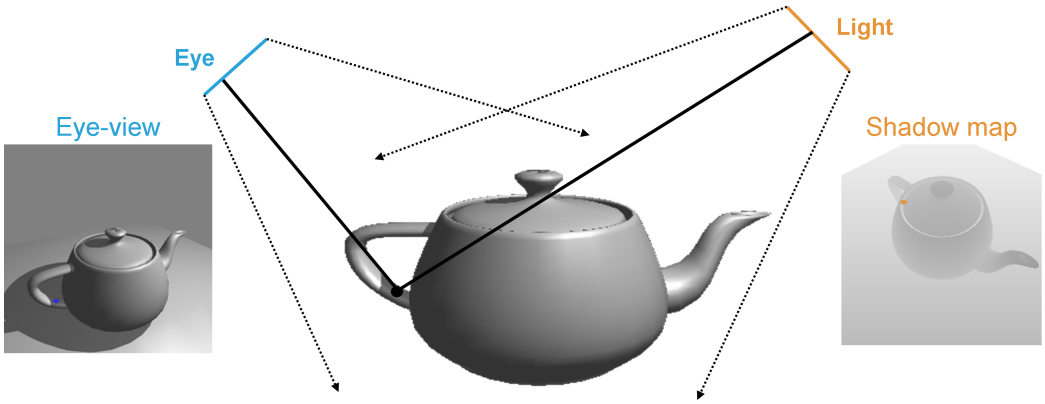

Fig. 2.4: On the *right* side we see the shadow map, generated from the point-of-view of the light. In the *middle* the transformation of a sample fragment from eye-space (blue sample) to light-space (orange sample) is shown. On the *right* side we see the final rendering with the applied shadow-map.

With the use of projective texture mapping as described by Segal [SKvW+92], shadow mapping can be easily matched to hardware, with one additional render pass to create the light view depth image, the *shadow map*.

An important benefit of shadow maps is that they can handle everything that can be drawn. This means we are not limited to polygonal data as geometry input. Another important benefit of shadow mapping is that it is independent of the scene geometry, which makes it suitable for highly tesselated scenes. This independence comes from the fact that shadow mapping is an image space technique. This means it uses a two-dimensional finite image, the shadow map, to store the gathered depth samples in a regular grid. This on the other hand introduces some aliasing problems (see Chapter 3) due to the nature of regular sampling through the shadow map. Another disadvantage of shadow maps is the difficulty of calculating the shadow of omnidirectional point lights. The reason why a point light is difficult to do with shadow maps lies in its spherical view. No single frustum can reflect this, and so a number of shadow maps and associated frusta have to be built to divide this spherical view into manageable bits.

2.3.2 The shadow-volume algorithm

A completely different approach to shadow creation was first described by Crow [Cro77]. This object-space method builds the actual shadow volumes created by the silhouette edges of the shadow casters as seen from the light source. Each silhouette edge is extruded in light direction to infinity to form a quad and together with the other edges creates a shadow volume.

A means of determination whether a pixel is in shadow is then provided with an inside-outside-test. For each pixel, we count the shadow volume faces we cross between the view point and the pixel. For front-facing faces, we increase our counter because we enter another shadow-volume, and for back-facing faces, we decrement our counter because we leave a shadow-volume (see Figure 2.5). If the total count is greater than zero, we are in a shadow-volume, hence the pixel is shadowed. This test can be implemented with the aid of a stencil-buffer in hardware.

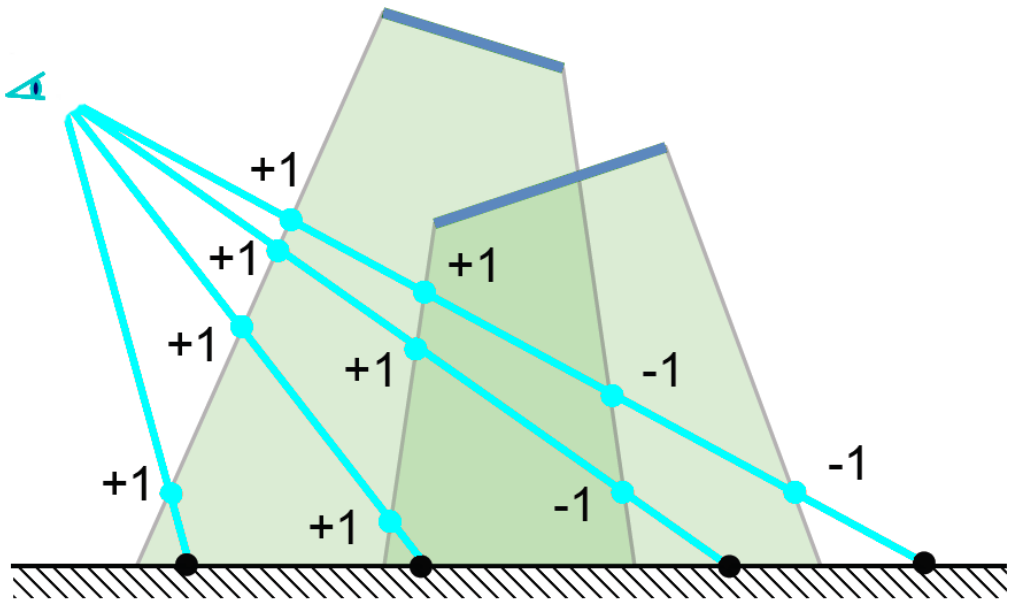

Fig. 2.5: A pixel is in shadow if the count is positive.

The obvious advantage over shadow maps is that no aliasing can appear, because the shadow boundaries are determined geometrically. Omnidirectional point light shadows can be calculated too, which are complex to calculate with shadow maps.

A disadvantage of this technique is the vast amount of fill-rate needed to render all the shadow-volumes. Additionally a silhouette detection has to be made, which

for polygon-rich scenes means another performance penalty. Finally, only polygonal data can be processed, because we need a simple way to detect the edges and extrude them. Under the aspect of soft shadows, the inherent hard shadow edges of this method are another drawback.

2.4 Summary

In this chapter we introduced shadowing as an important technique to enhance the realism of a computer-generated scene. We described what shadowing means and how it is done in theory, as well as in real-time with the two most important shadowing techniques, shadow volumes and shadow maps.

For further information, regarding shadow algorithms, we recommend Möller and Haines' Real-Time Rendering book [MH02].

Chapter 3

Problems of shadow mapping

In this chapter we give an extensive presentation of the problems of the shadow map algorithm and introduce the practical implications that are the result of these problems.

As discussed in Section 2.3.1, shadow mapping is an image space technique normally used with a z-buffer (the shadow map) to store the depth information from the point-of-view of the light source (see Figure 3.1 for an example depth buffer image).

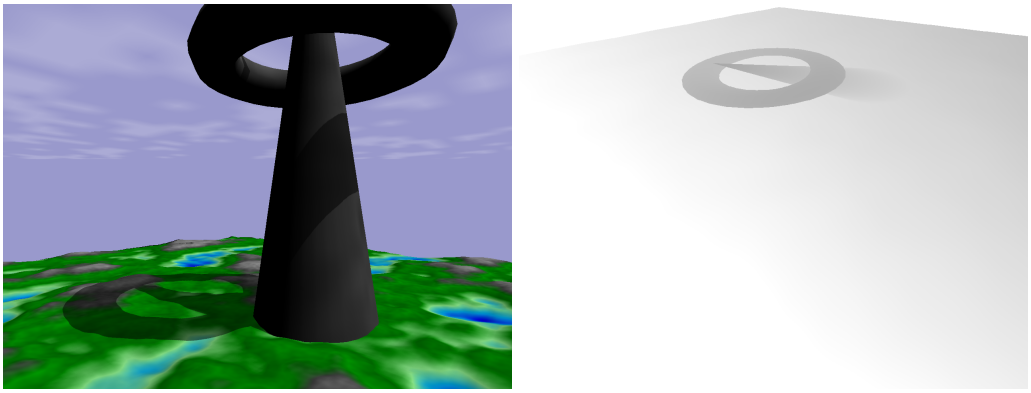

Fig. 3.1: The shadowed eye view (*left*) is created with the depth informations stored in the shadow map (*right*), the depth view of the light.

The z-buffer is a finite and regular grid, which stores a distance at each of its grid cells. On sample of this grid can map to several samples, leading to the 'blocky' appearance typical of image-space algorithms. In general we need an individual shadow map depth information for each transformed fragment to avoid such errors. This would mean that the resolution of the shadow map should vary and should give us an individual depth value for each transformed eye-space fragment. This effect is called undersampling and it causes these incorrect shadowing results, which are a form of aliasing artefacts present in all image space methods. This basic (meaning not "augmented") form of the technique will be called

standard shadow mapping (SSM) in the rest of the thesis.

In the following sections we will describe the three main problems of shadow mapping:

- Perspective aliasing (see Section 3.1)
- Projection aliasing (see Section 3.2)
- Incorrect self-(un)shadowing (see Section 3.3)

3.1 Perspective aliasing

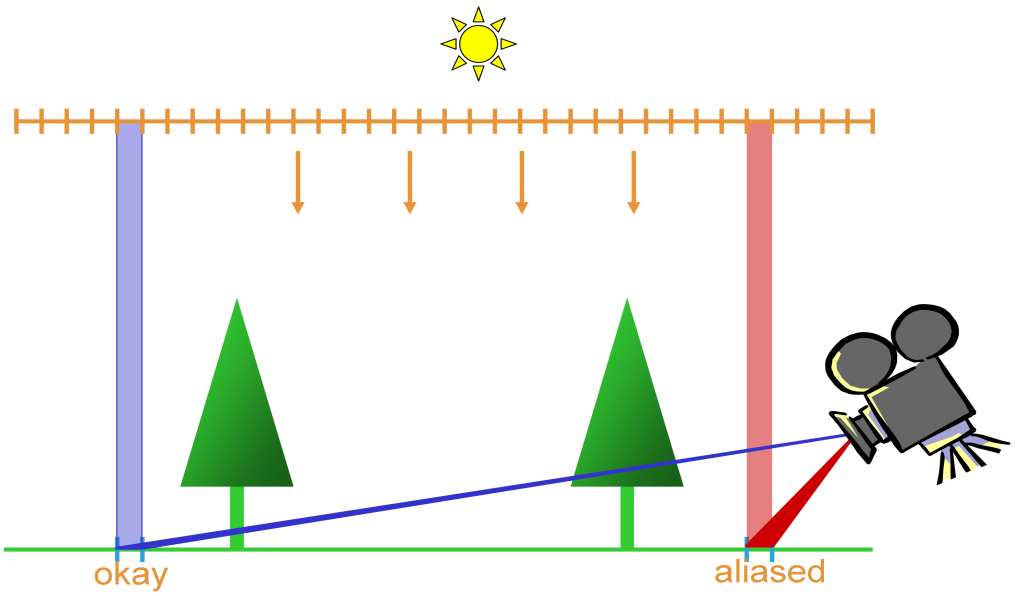

Fig. 3.2: The far away shadows have enough resolution, but the shadows near the view point have insufficient resolution caused by *perspective aliasing*.

Perspective aliasing is common with standard shadow maps for a perspective eye view. A perspective view shows nearby objects larger than distant objects. The light space in which the standard shadow map is calculated does not incorporate this information. So an object is stored with a fixed resolution in the shadow map, regardless of the distance to the eye. The outcome is a shadow resolution that is too low for nearby objects and too high for distant objects in eye space. In Figure 3.2 the light source is symbolized by the small sun (yellow). The shadow map is represented by a one-dimensional grid (orange) and its texels are symbolized through the space between the small vertical strips. The light direction is

straight downward (orange arrows). The scene objects are two trees on the ground (green). The camera is observing this scene from the right side. The equally spaced shadow resolution (in light-space) is unequally distributed in eye-space. This leads to enough resolution for far away shadows (blue), but insufficient resolution for the shadows near the view point (red).

This effect is biggest if the light direction and the view direction of the camera are perpendicular to each other. The smallest effect of perspective aliasing occurs if the light direction and the camera are facing in the same direction (are parallel to each other).

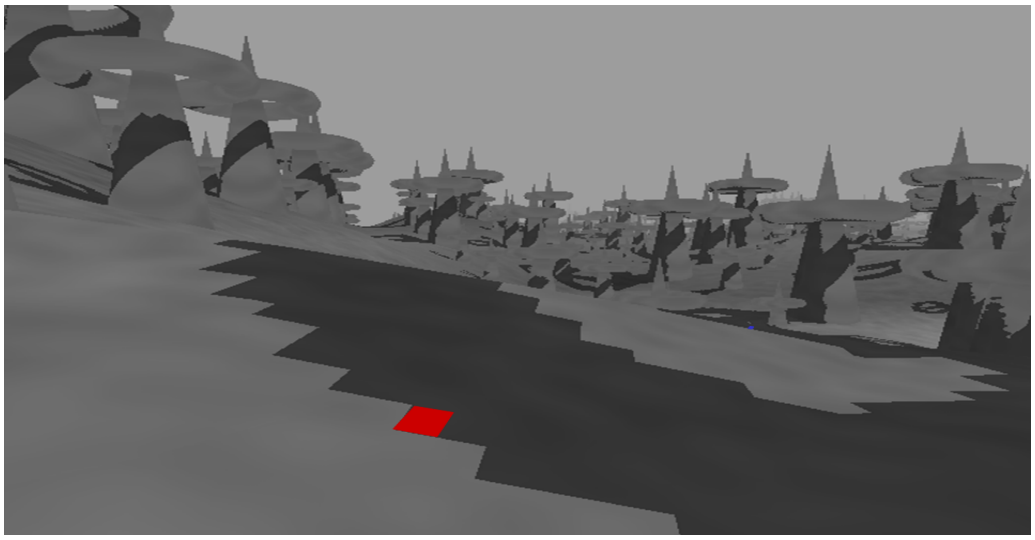

Fig. 3.3: The result of perspective aliasing is insufficient shadow map resolution for shadows near the view point.

In Figure 3.3 the actual effect on shadows can be seen. The marked (red) square in the front is one texel of the shadow map, magnified in eye-space through projection aliasing. Also it is clearly visible that perspective aliasing is only a problem near the point-of-view of the camera because the shadow map texels in the distance are too small to be distinguished from each other and therefore provide sufficient shadow resolution for far away objects.

3.2 Projection aliasing

Surfaces that are roughly parallel to the light direction are sampled sparsely in the shadow map like in Figure 3.4. In this figure the crown of the tree is roughly parallel to the view direction of the light. The marked (red) part of the tree is stored in just one shadow map texel. The camera sees the same texel enlarged and with

insufficient resolution. A single shadow texel projects to many pixels on screen, leading to the *projection aliasing* error. The outcome are incorrect shadows. But the worst thing about projection aliasing is its large temporal incontinuity on camera movement. This can lead to severe shadow flickering when moving around in a scene and greatly disturbs the intended effect of the shadow.

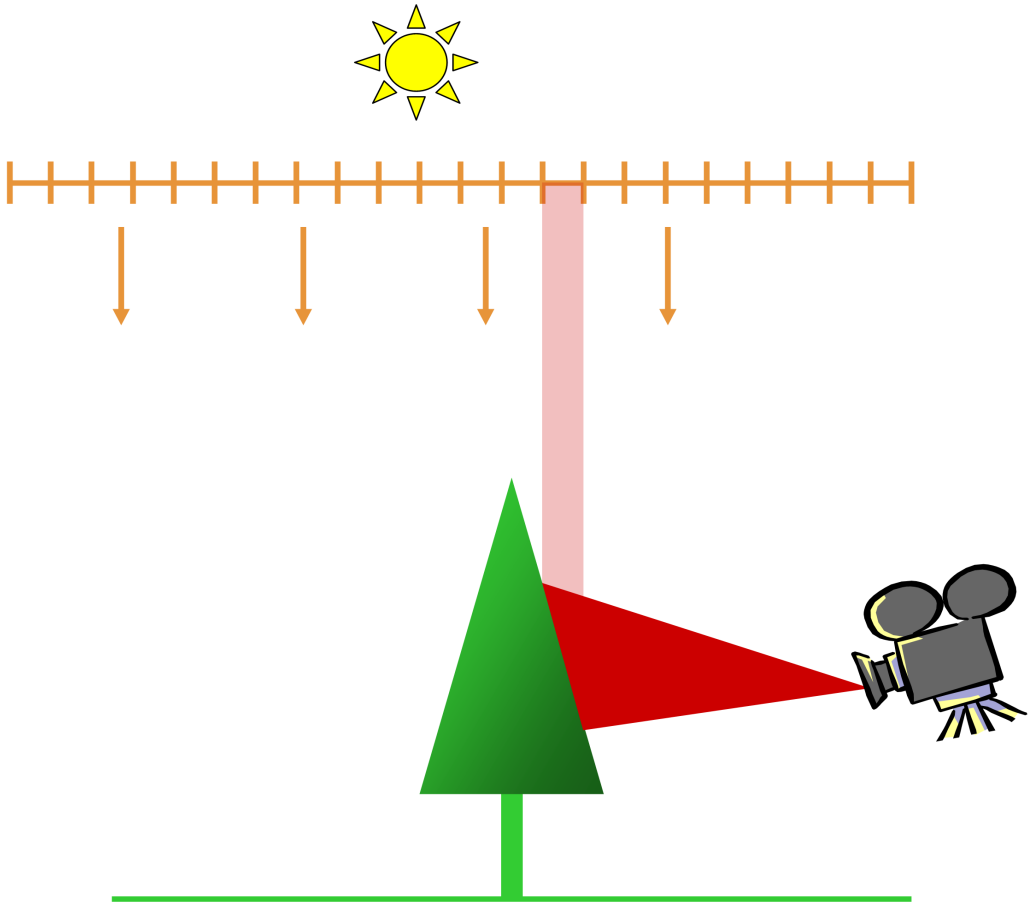

Fig. 3.4: The result of perspective aliasing is insufficient shadow map resolution for shadows near the view point.

In contrast to perspective aliasing, projection aliasing is independent of the angle between the view direction of the camera and the light direction. It is only dependent of the the light direction and the surface normal. It is largest if the surface normal is perpendicular to the light direction. In this case the surface projected into the shadow map has no area and so no depth information is stored. This leads to arbitrary shadowing results and an infinite projection aliasing error. Projection aliasing is smallest if the light direction and the surface normal are parallel to each other. In this case the area of the projected surface in the shadow

map is maximized and therefore a maximum of depth information is stored in the shadow map. The result is no projection aliasing in this case.

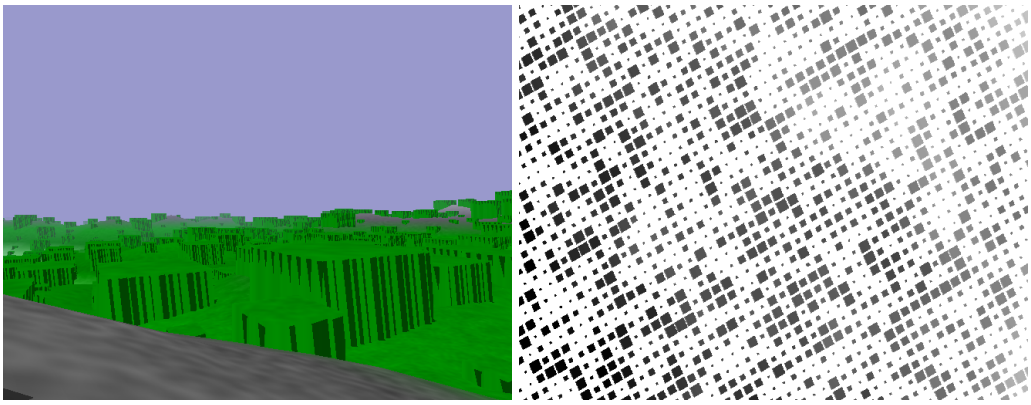

Fig. 3.5: The projection aliasing artefacts, the black stripes, in the eye view (*left*) are caused by too few samples of the cubes sides as seen from the light view (*right*).

Figure 3.5 shows an extreme case of projection aliasing. The light shines straight down on thousands of green cubes with their sides facing side-ways, standing on green hills. All things considered this is a case with large projection aliasing. The light view shows hardly any depth information for the sides of the cubes and the result are the stripes in the eye view. These stripes are the rest of the shadows that should shadow all of the side-surfaces of the cubes, so the shadows are badly out of shape. And worst of all these stripes are going to jump randomly when moving around, causing irritating flickering effects.

3.3 Incorrect self-shadowing (“shadow acne”)

In shadow mapping, sampling takes place at two times in the process: Once when the shadow map is created and the depth information of the scene from the point-of-view of the light source is stored in a regular and finite grid. And once when the unshadowed fragments from the point-of-view of the eye are calculated and have to fit the output frame-buffer resolution. Most of the time these two sampling processes will lead to two different samplings of the scene data. Therefore on transformation of the eye-space fragments to light-space, resampling takes place.

In Figure 3.6, the sampling is represented through the lines emanating in the mid-points of the corresponding pixels of the shadow map (orange) at the top and of the observer frame-buffer (blue) from the left side. The line (green) represents a polygon in the scene. Obviously this polygon should not shadow itself. In the marked case (red circle) the observer’s distance should be smaller than the depth stored in the shadow map to get the correct shadowing results. But the different

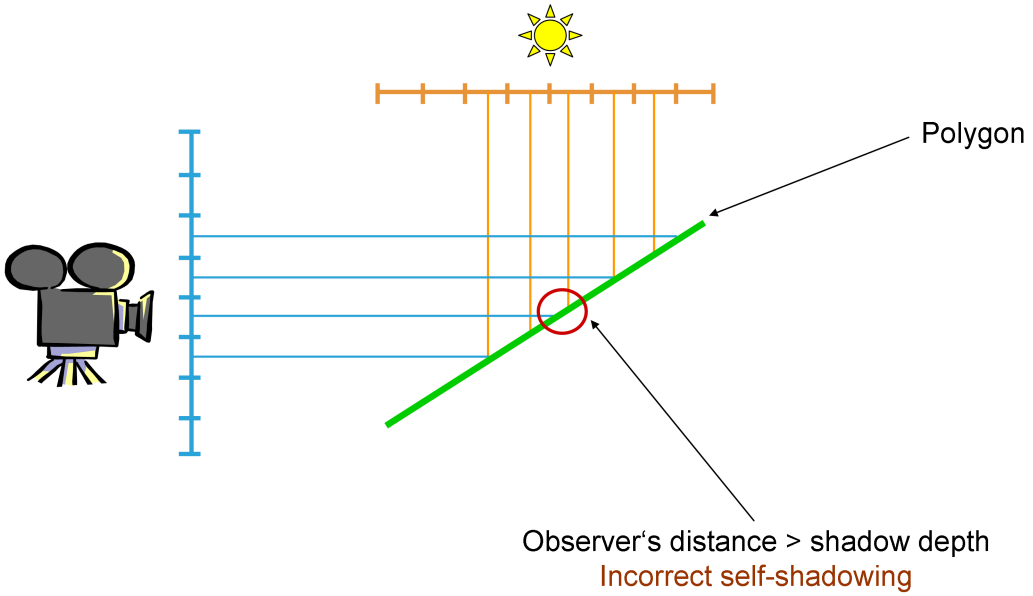

Fig. 3.6: Different sampling, once in light-space and once in eye-space can lead to incorrect self(un)-shadowing

samplings lead to the reverse case: The observer’s distance is greater than the distance stored in the shadow map, leading to *incorrect self-shadowing*.

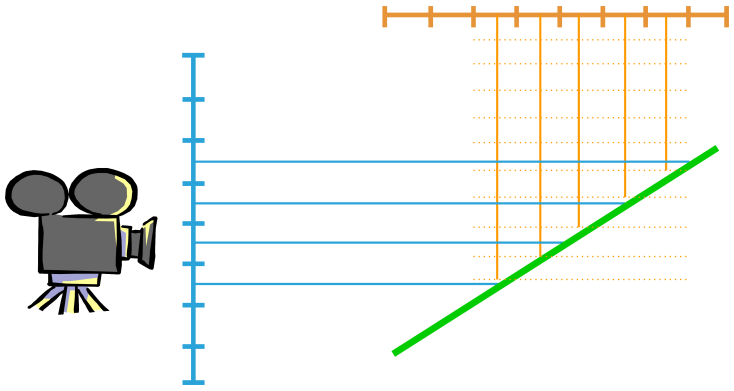

Fig. 3.7: Depth quantization is another cause for incorrect self-shadowing.

An additional problem, leading to incorrect self-(un)shadowing, is *depth quantization*. The depth information that is stored in the shadow map is not exact, but uses an integer or floating point representation of finite precision. Most common are shadow maps with 24 bit of integer information for a stored texel. This means not every depth value can be represented exactly in the shadow map, but the shadow values are rounded to the nearest depth value representable through

the given shadow map precision. This means the orange lines emanating from the light source cannot assume any length, but just certain lengths limited through the precision of the shadow map (see Figure 3.7). The stippled lines represent the depth values that can be stored in the shadow map. The depth that is used for the depth comparison for shadow mapping is therefore not the depth at the real intersection of sample and geometry.

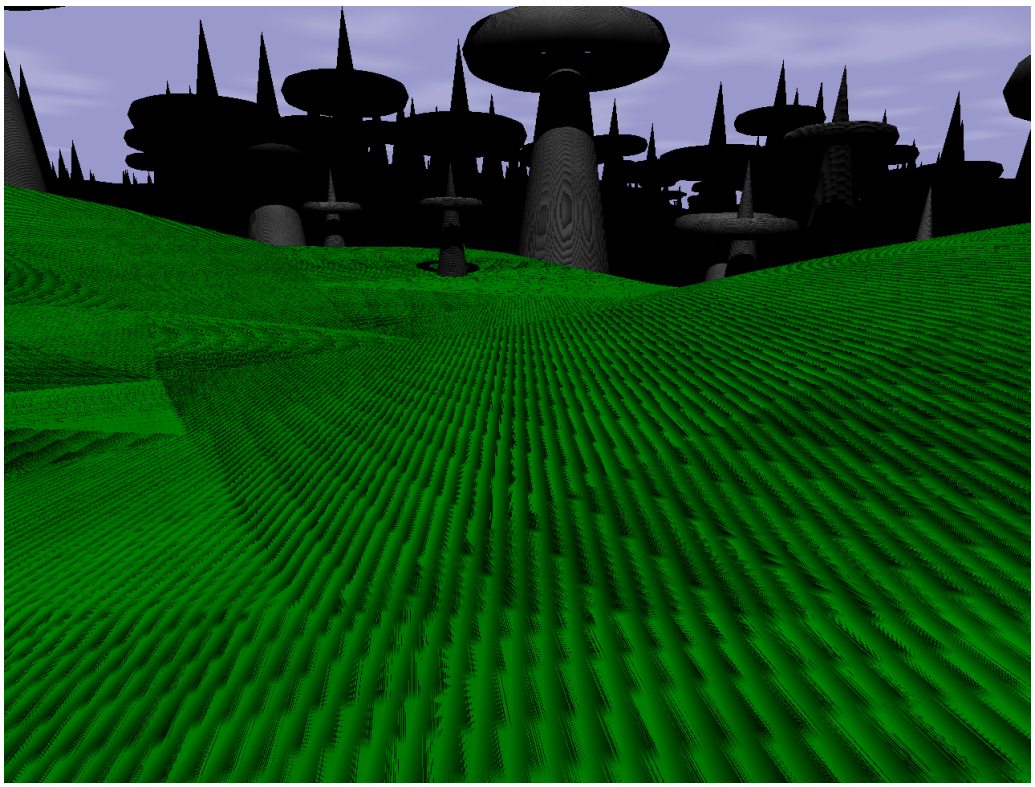

Fig. 3.8: The moiré patterns (incorrect self-shadowing) are caused by resampling errors

Figure 3.8 shows the results of incorrect self-shadowing: visible patterns of shadowed and unshadowed regions. Obviously the ground should be unshadowed, except where the trees are casting shadows.

3.4 Summary

In this chapter we discussed the various problems of shadow mapping. We identified three main problems: perspective aliasing, projection aliasing and incorrect self-shadowing. The causes for thess problems are found in undersampling, resampling and depth quantization errors, which are common problems for any image space algorithm. Table 3.1 summaries the causes for each of the noted errors.

Error	**Cause**	**Result**
Perspective aliasing	Undersampling	Insufficient resolution near the observer
Projection aliasing	Undersampling	Insufficient resolution on polygons almost parallel to the light direction
Incorrect Self-(un) shadowing	Resampling; depth precision	Moiré-patterns

Tab. 3.1: The different errors of shadow mapping.

Chapter 4

General solutions to the problems of shadow mapping

In this section we take a look at techniques that influence all aspects of shadow mapping (see Chapter 3). Section 4.1 shows techniques to solve all of the problems of shadow mapping at once. Regrettably, this comes at the cost of speed. Section 4.2 describes what volume of space is needed for the generation of the shadow map and how this volume can be determined. In Section 4.3 we elaborate on the various filtering techniques usable for shadow mapping.

4.1 Solutions for all problems of shadow mapping

The image-space nature of shadow mapping is already discussed in Section 2.3.1 and Chapter 3. The main problem of the image-space approach is undersampling and the resulting aliasing artefacts.

Numerous solutions to this undersampling problem exists. Johnson [JMB04] proposes the use of an irregular z-buffer that can eliminate thess problems, by providing different sampling densities, based on the projected size of the eye space pixels in the light space. The down side of this method is that it is currently not suitable for hardware, and a software implementation is not competitive to today's hardware methods in terms of the attainable speed. A similar approach was chosen by Aila and Laine [AL04].

Another solution introduced by Fernando et al. [FFBG01] is to use a hierarchical grid as a shadow map structure. This hierarchy is refined only when required, and the appropriate resolution is chosen according to the impact on image quality a given pixel has with a cost-benefit function, hence the name adaptive shadow maps. This approach doesn't map well to hardware implementations and therefore the speed of this method is only interactive.

4.2 Focussing the light space

In standard shadow maps, the view frustum often covers only a small area while, the remainder of the shadow map is wasted (no visible information is stored there). To increase the amount of useful information that is stored in the shadow map, we only want to consider those parts of the light space for the shadow map that can cast a visible shadow into the view frustum. Brabec [BAS02] showed the importance of focusing the shadow map to the visible parts of the scene. This step seems obvious, but some intricacies are involved. A geometrical recipe was given by Stamminger and Dretakis [SD02]. The idea is to calculate the convex hull of the view frustum V and the light position l (for directional lights this position is at infinity) and afterwards clip this body with the scene bounding volume S and the light frustum L. We call the resulting intersection body B and write the calculation as

$$B = (V + l) \cap S \cap L \tag{4.1}$$

where $+$ denotes the convex hull operation and $\cap$ denotes the clipping operation. Clipping to the scene bounding volume S is necessary because today very large view frusta are common and they frequently extend outside the scene borders. But this method can still be improved. The idea is to work with the clipped view frustum. This means before we do the convex hull determination and the clipping with scene and light frustum, we clip the view frustum with the scene bounding box S. This leads to

$$B = ((V \cap S) + l) \cap S \cap L \tag{4.2}$$

This removes the part of the volume that is inside the view frustum, but outside the scene bounding volume. Figure 4.1 shows this process. Our new method gains smaller sized volumes, but nevertheless includes all relevant casters and ocluders. In Figure 4.2 we can see how this process is done in 3D with a real scene.

For the implementation of such a simple method, special purpose geometrical functions are often the best choice, because we only use convex volumes with a small number of corner points. We will discuss such an implementation in more detail in Chapter 9.

If we have more CPU time to spare, we can also use a more involved method, for example unit cube clipping as described by Kozlov [Koz04]. If we use a visibility algorithm, O'Rorke's article [O'R04] gives various practical hints for using this visibility information to decrease the volume we have to consider for the shadow map generation.

All in all, a trade-off between exact volume determination for a given scene with higher resulting shadow quality and higher computational costs, and a simpler method with a coarser result but with a faster execution has to be made. For densely filled scenes, a coarser approach is often the wiser choice because the

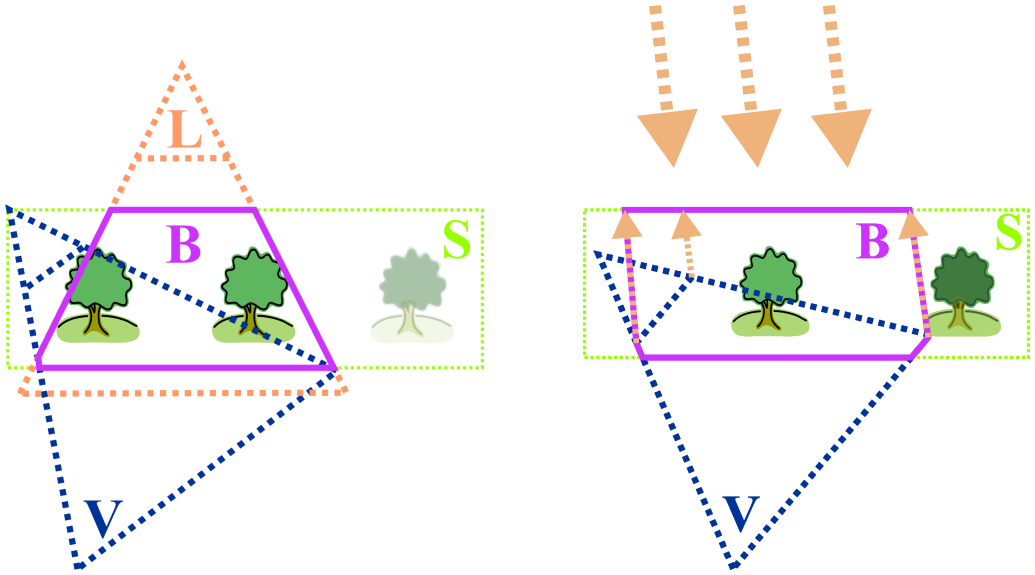

Fig. 4.1: For calculating the focus region B (violet) we use the view frustum V (blue), the light frustum L (orange) and the scene bounding volume S (green). *Left:* B of a spot light is calculated by extruding the clipped view frustum $V \cap S$ and clipping it with the spot light frustum L and the scene bounding volume S. *Right:* B of a directional light is calculated by extruding the clipped view frustum and clipping it with S.

computation cost for huge numbers of objects are high, and often the gain for such a scene is minimal. On the other side, for sparely crowded scenes a more involved approach can gain a lot over a simple scene approach.

4.3 Filtering

Filtering of shadow maps is an important technique that can improve the quality of the resulting shadows greatly [Hec86]. We will discuss the basics of fast filtering of shadow maps and the ideas of percentage closer filtering to produce softer, less blocky shadows.

4.3.1 Bilinear filtering

Normally textures are filtered bilinearly. Bilinear filtering takes four samples to interpolate one resulting sample. The four used samples are the nearest pixels in the texture to the exact texture coordinate. First we interpolate twice in one coordinate direction between samples 0 and 1 and between samples 2 and 3 to get the two interpolated samples A and B. Finally we once interpolate in the other coordinate direction between samples A and B to get the final sample S

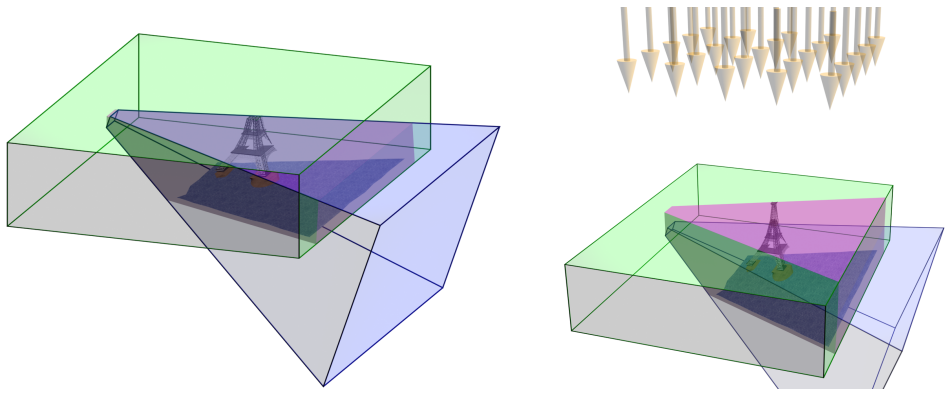

Fig. 4.2: *Left:* The clipping of the view frustum (blue) with the scene bounding box (green) decreases its size considerably ($V \cap S$). *Right:* The final intersection body B (violet) with a light direction from above (orange arrows)

(see Figure 4.3 and the following formulas).

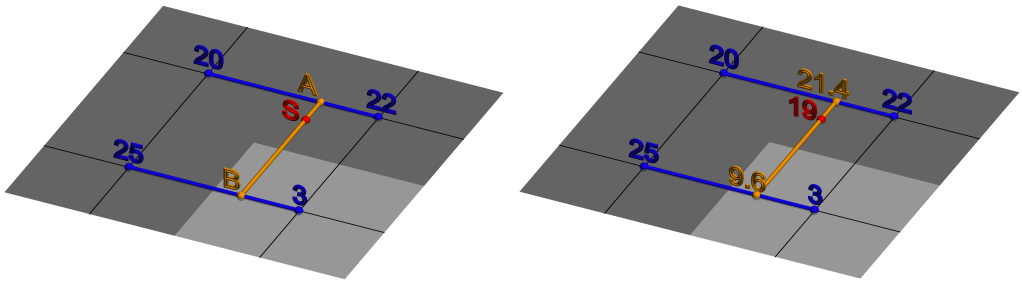

Fig. 4.3: *Left:* The two blue sample pairs are each linearly interpolated to get A and B. Finally these two are linearly interpolated to get the final sample S. This process is called bilinear interpolation. *Right:* The interpolation of the first two samples leads to 21.4 and of the second pair to 9.6. After the final interpolation the resulting value is 19.

$$sample_A = sample_0 + u(sample_1 - sample_0) \tag{4.3}$$
$$sample_B = sample_2 + u(sample_3 - sample_2) \tag{4.4}$$
$$sample_S = sample_A + v(sample_B - sample_A) \tag{4.5}$$

$sample_{0..3}$ denote the four samples $0..3$. $sample_{A,B,S}$ denote the interpolated samples A, B, S. u, v denote the u respective the v coordinate direction.

For example if we use a texture of 512x512 pixels and we also use texture coordinates that are in the range $[0, 511]$ for each dimension, we can address each pixel of the texture with an integer pair of texture coordinates. $(12, 40)$ would for instance reference to the pixel in the 13th column and in the 41th row of our texture.

Real texture coordinates are fractional. If we had gathered the texture coordinate $(4.7, 320.2)$, we would use the pixels at coordinates $(4, 320)$,$(5, 320)$,$(4, 321)$ and $(5, 321)$ for interpolation.

As can be seen in the Formulas 4.3- 4.5, the fractional parts of the texture coordinates are used as weights to mix the four samples together. If the four sample values were $20, 22, 25, 3$ than the bilinear interpolation would lead to:

$$\begin{aligned} sample_A &= 20 + 0.7(22 - 20) = 21.4 \\ sample_B &= 25 + 0.7(3 - 25) = 9.6 \\ sample_S &= sample_A + 0.2(sample_B - sample_A) = \\ &= 21.4 + 0.2(9.6 - 21.4) = \\ &= 19 \end{aligned}$$

4.3.2 Percentage closer filtering

For depth maps, ordinary bilinear filtering gives unsatisfactory results. The first problem is that after bilinearly interpolating a new depth value, this depth value is subjected to the depth comparison to determine if this pixel lies in shadow. This test can only give a binary decision which leads to hard shadow edges and makes soft shadow edges impossible. The second and more serious problem is the fact that the bilinear interpolation makes no sense for the depth values along the edges of objects.

For example take a look at Figure 4.3, right. A surface has a depth value of for example 19.5 and lies on the red spot in the shadow map. Bilinear filtering gives a shadow map depth on this position of 19. $19 < 19.5 \rightarrow 1$ which means the red point in the shadow map is nearer and the example point is farther away. The binary result prohibits smooth shadow borders and determines the point to be 100% inside the shadow.

Percentage closer filtering (PCF) [RSC87] solves these problems by reversing the order in which the filtering and the comparison steps are applied. In its simple hardware implemented form, PCF uses a bilinear filter after the comparison has taken place. Nevertheless the filtering is not applied to colors, but to binary texture values, the results of the depth comparison. This means that after the depth comparison a binary map results and on this binary map the bilinear filtering takes place.

For our example this leads to four comparisons:

$$\begin{aligned} 20 < 19.5 &\rightarrow 0 \\ 22 < 19.5 &\rightarrow 0 \\ 25 < 19.5 &\rightarrow 0 \end{aligned}$$

$$3 < 19.5 \rightarrow 1$$

Three times we get the result that our sample is in front of the shadow map sample, so it is unshadowed, and one time we get the result that our sample is behind the shadow map sample and is shadowed. These results are bilinearly interpolated

$$\begin{aligned} sample_A &= 0 + 0.7(0 - 0) = 0 \\ sample_B &= 0 + 0.7(1 - 0) = 0.7 \\ sample_S &= sample_A + 0.2(sample_B - sample_A) = \\ &= 21.4 + 0.2(0.7 - 0) = \\ &= 0.14 \end{aligned}$$

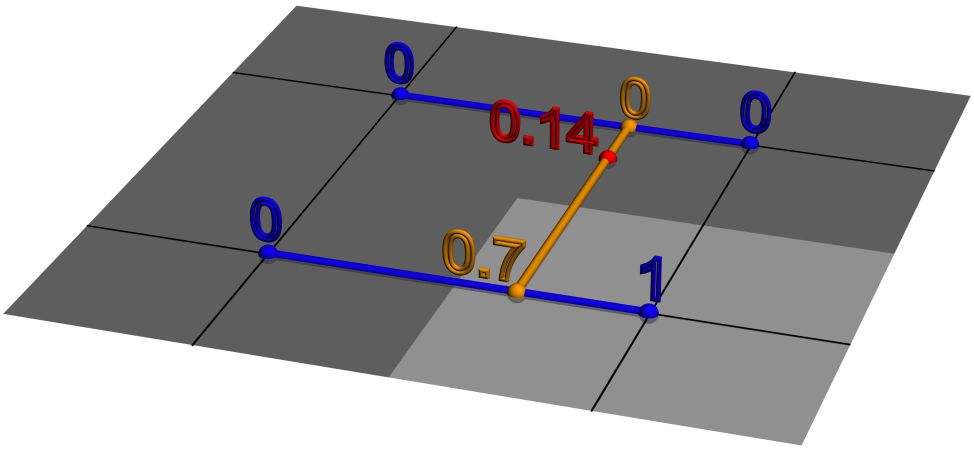

Fig. 4.4: *Percentage closer filtering* first conducts the depth tests and afterwards bilinearly interpolates the results to acquire a smooth shadow value.

and result in 0.14. The sample is 14% shadowed (see Figure 4.4). We used a simple 2x2 box filter on the shadow map. PCF results in smooth shadow boarders, removing some of the "blockyness" of a unfiltered shadow map. Improvements of this method can include a larger box filter (3x3 or 4x4 are feasible values, but lead to slower results because 9 respective 16 texture accesses are needed, opposite to only 4 texture accesses with the 2x2 box filter).

The original percentage closer filtering is much more involved. It requires that the region for which we like to determine the shadowing status is mapped into the shadow map space and is then stochastically sampled. This means the size and shape of the filter kernel would change for each region. Off-line rendering systems implement this more advanced form of PCF and yield superior results.

NVidia GeForce 3+ hardware implements a version of the 2x2 box filtered percentage closer filtering in hardware. ATI only supports PCF in the form of emulation through shaders, which results in slower rendering. Figure 4.5 shows the results of the different filtering methods.

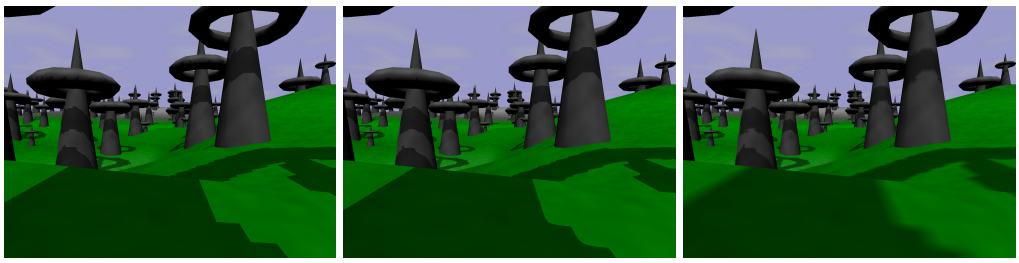

Fig. 4.5: The results of different filtering schemes. *Left:* Unfiltered (the nearest shadow map texel is used) *Middle:* Bilinear filtering *Right:* Percentage closer filtering.

4.4 Summary

In this chapter we gave a brief overview of techniques that can alleviate all the problems of shadow mapping. We introduced techniques, like adaptive shadow maps, that can solve all aliasing problems of shadow maps, but are too slow for real-time purposes. We discussed the intricacies involved in focusing the shadow map and provided several recipes for managing this step. Finally we discussed in detail the most important filtering method for shadow maps, percentage closer filtering. In the following chapters we will venture into the different errors of shadow maps one at a time and search out more specialized methods to deal with them.

Chapter 5

Solutions to incorrect self-shadowing

As already mentioned in Section 3.3 incorrect self shadowing is a problem that is almost surely present in most shadow mapping cases. Therefore fast and robust solutions to this problem are of utmost importance. In the next sections we will describe most of the import methods for real-time computer graphics. In Section 5.1, we will talk about the most widely used method, biasing, and give an in-depth discussion of the different methods for bias calculation. Afterwards we take a look at more elaborate schemes from the literature in Section 5.2. Section 5.3 gives a valuable comparison of practical biasing approaches that work robustly and fast. Finally Section 5.4 summarizes our findings.

5.1 Biasing

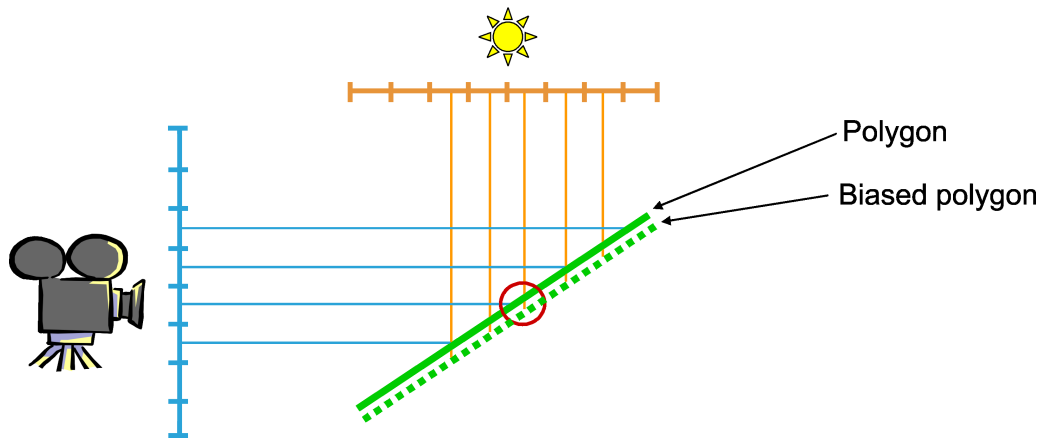

Fig. 5.1: A biased polygon can avoid incorrect self-shadowing.

The standard solution to incorrect self shadowing is, as is so often the case in computer graphics, just a workaround for the problem. This solution is a manually defined depth bias. A depth bias is a small increment added to the shadow map

depth values to move them further away in order to avoid an incorrect shadowing of the corresponding shadow casters. This normally needs user intervention and provides in general no automatic solution for an arbitrary scene. The main benefit of this method is its easy support through hardware and the speed penalty (that is, however, not worth mentioning) involved in using this method.

Figure 5.1 shows that biasing actually moves the geometry away from the point of view of the light source. The unbiased polygon (green line) becomes the biased polygon (stippled green line) and the beforehand wrong shadow map depth becomes correct because in the marked case (red circle) the observer's distance to the light is now smaller than the "bias augmented" distance stored in the shadow map. As a consequence, incorrect self-shadowing disappears (see Figure 5.2).

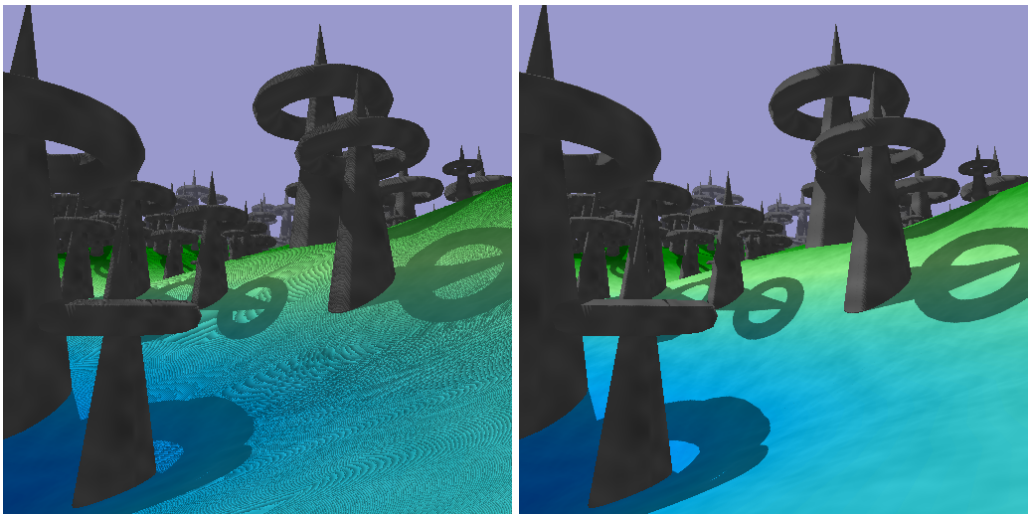

Fig. 5.2: The moiré patterns are caused by incorrect self-shadowing (*left*). When a bias is applied the artefacts disappear (*right*).

A problem of the bias is that the shadow is moved in light-space z-direction. In Figure 5.1, we can see that biasing is equivalent to moving the geometry away from the light source for the depth test. This can lead to noticeably misplaced shadows (see Figure 5.3).

If we use a simple constant bias, all depth values are moved the same amount. This leads to varying results for polygons with different depth slopes. For a depth slope near zero, hardly any biasing is needed, while for a polygon that is almost parallel to the light direction (large depth slope) a big bias is appropriate. Figure 5.4 shows this effect and introduces the solution: slope-scale biasing. Here the bias is altered dependent on the depth slope of the polygon. Because a rasterizer has to calculate the depth slope anyway, this method is extremely hardware-friendly.

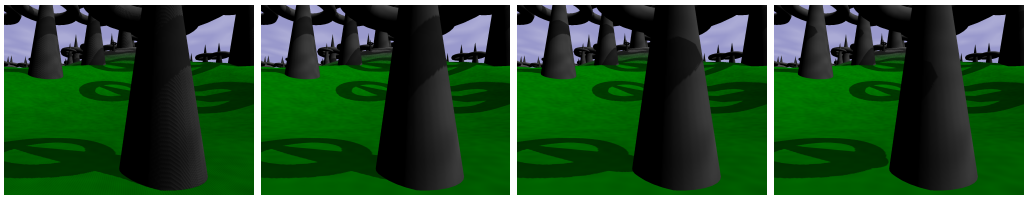

Fig. 5.3: A larger biasing value leads to a larger misplacement of the resulting shadows, but also leads to a more reliable avoidance of incorrect self-shadowing.

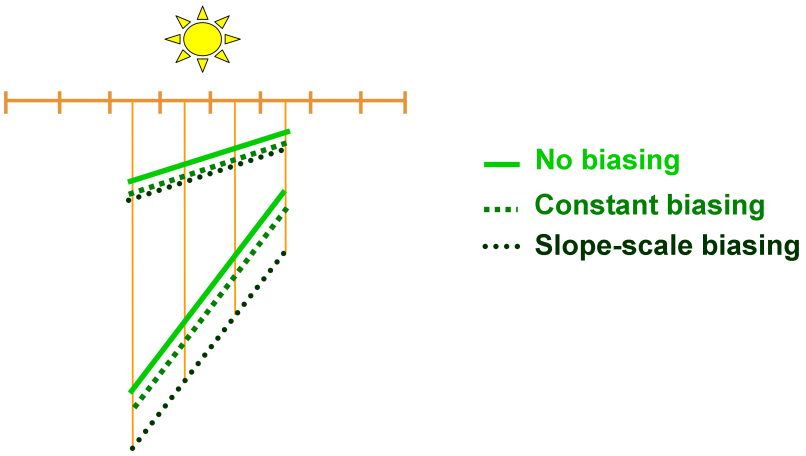

Fig. 5.4: Slope-scale biasing moves a polygon dependent on it's depth slope

However, slope-scale biasing has problems with the non-linear distribution of z-depth values present at point (spot) lights, PSM, TSM, LispSM and similar methods that will be discussed later in Chapter 7. This non-linear distribution of depth values is generated by the perspective transformation that involves an $1/w$ term, generating a hyperbolic depth value distribution. Therefore the false self-shadowing problem is increased for these algorithms. For Trapezoidal shadow maps (TSM [MT04]), the biasing problem is so great that the authors of the paper recommend omitting the z-coordinate from the perspective transformation, actually generating linearly distributed depth values. Kozlov [Koz04] proposes to use slope-scale biasing in world-space for PSM and transforms the results into post-projective space. LispSM has less problems with self-shadowing artefacts and can reuse slope-scale biasing.

Another method for avoiding incorrect self-shadowing is backside rendering. As can be seen in Figure 5.5, backside-rendering uses the backside of the geometry for generating the shadow map. This can be achieved by rendering with reversed backside-culling. The shadow map depth comparison is therefore shifted to the back side of the geometry, removing any incorrect self-shadowing, but in-

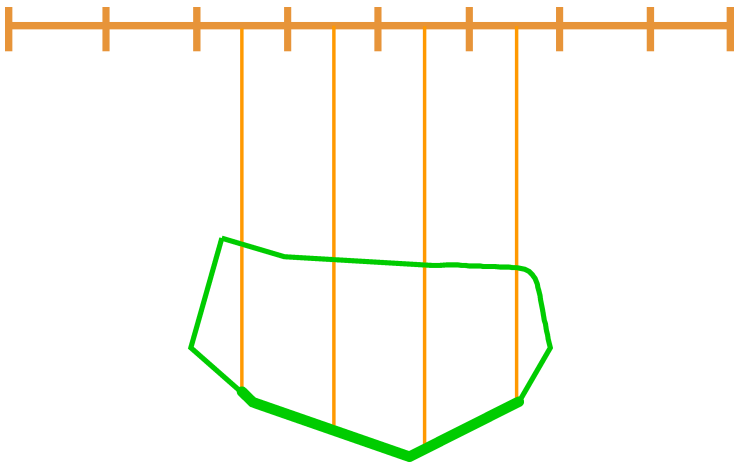

Fig. 5.5: Backside rendering uses the backside of the geometry for generating the shadow map, leading to a more robust shadow map test for closed geometry.

troducing incorrect self-unshadowing. Incorrect self-unshadowing is the pendant to incorrect self-shadowing that occurs on the front side of geometry as seen from the point of view of the light. Incorrect self-unshadowing only occurs on the back side of geometry, as seen from the point of view of the light source (see Figure 5.6, bottom, left). It has the same reasons, the resampling from eye to light space as incorrect self-shadowing. What we have gained is that self-unshadowing only takes place on polygons that are facing away from the light and should be darkened by the normal shading equations anyway. Alternatively we could also use slope-scale biasing for the backsides with an inversed bias (see Wang and Molnar [WM94]).

5.2 Two shadow maps

The following techniques are more theoretical than as practical methods because of their speed penalty.

Second-depth shadow mapping, as proposed by Wang and Molnar [WM94], is a solution that can handle most incorrect self-shadowing configurations well. The idea is to assume solid shadow casters. With this assumption, the depth test can be transfered from the nearest surface as seen from the light source to the second nearest surface. This gives different results as with backside rendering. Here a technique like depth peeling is needed to correctly determine the second nearest surface.

Later Weiskopf and Ertl influenced by the work of Wang and Molnar and Woo [Woo92], the midpoint shadow maps, proposed dual shadow maps [WE03]. The first shadow map contains the depth of the first surface visible from the light

source, as any normal shadow map does. The second shadow map contains the depth of the second surface, the surface you would encounter if you had stripped away the surfaces stored in the first depth map. The two used shadow maps contain the first respectively the second depth from the light source. These two shadow maps are combined in the function

$$z_{bias} = min(\frac{z_2 - z_1}{2}, z_{offset}) \tag{5.1}$$

where z_1 denotes the depth value from the first shadow map, z_2 the depth value from the second shadow map and z_{offset} an offset value, that is typically in the magnitude of the size of the objects in the scene. The first part $\frac{z_2-z_1}{2}$ is responsible for shifting the z-value used for the later depth test for shadow mapping between the first and second depth and for this reason makes the comparison more robust, for closed occluders and receivers. z_{offset} removes unshadowing artefacts of far-away receiver and occluder constellations. This constant is the maximum bias and therefore can be chosen quite large because the closed configurations are already handled by $\frac{z_2-z_1}{2}$. The disadvantage of this method is the performance penalty introduced by the additional shadow map generation pass and for depth peeling.

For arbitrary scenes with large quantities of objects, an automatic method seems indispensable. One offset value rarely suffices to solve the biasing problem of the whole scene exactly, and manually choosing offset values can be too time consuming. Dual shadow maps are problematic to use because the generation of the shadow map per se is already expensive. A lot of objects are visible for the common directional lights and have to be rasterized. The second shadow map we would need to generate would nearly double the costs of this step. Also depth peeling introduces further costs, which make these approaches slow.

5.3 In practice

The methods [WM94] [Woo92] [WE03] mentioned in Section 5.2 are unsuited for real-time applications because of the performance penalty introduced by the generation of the second depth image needed by all of them. In the following discussion we therefore restrict ourselves to the fast methods that work with small or no speed penalty.

We present extensive experiments with different biasing methods for a non linear shadow mapping approach, LispSM, that is suitable for practice. We found that a simple slope-scale biasing with the hyperbolic z-distribution of LispSM, as for example provided by the *polygon offset* interface of Open GL, gives satisfying results for common configurations. The needed bias can be much smaller than the value needed for a constant bias. The resulting shadows are therefore less

shifted respectively more correctly positioned than with the bigger constant bias. For example in our test scene, we could avoid most self-shadowing artefacts with a relatively small slope-scale bias of 2.0/4.0, where 2.0 is the value of the $factor$ parameter and 4.0 is the value of the $units$ parameter. The formula that calculates the resulting bias is

$$bias = factor\,\Delta z + r\,units \tag{5.2}$$

where Δz is the depth slope of the polygon relative to the screen are of the polygon and r is the smallest value that is guaranteed to produce a resolvable offset for a given implementation. This means that $factor$ sets the influence of the depth slope and $units$ sets the minimal bias value.

The use of a linear z-distribution as proposed in the TSM paper is also easily incorporated into LispSM with the aid of vertex shaders. The results with linear z biasing are slightly better as with the hyperbolic z-distribution, but come at the cost of additional hardware requirements, namely vertex shaders. Backside rendering was implemented too. This method removes all self-shadowing artefacts on the ground, but in all other cases the quality is similar to normal slope-scale biasing.

The matrix of images in Figure 5.6 shows the results with the various combinations of the afore-mentioned methods. The performance of the different methods is equal, because of the unnoticeable penalty in rendering time these methods introduce. On all our test platforms, no speed difference at all is perceivable. As can be seen, the version with a linear z-distribution together with slope-scale biasing, and the back-side rendering method give the best results in this setup.

5.4 Summary

In this chapter we gave an explanation of the different approaches to cure the incorrect self-(un)shadowing problems of shadow maps and stated their individual advantages and drawbacks. We divided our analysis into the fast biasing methods and the more complex, more time-consuming methods that use two shadow maps for a correcter bias estimation. However, in practice the bias methods have the definitive speed advantage and can provide usable results for common situations. On comparing the various fast methods, we identified backside rendering as the most robust method for scenarios with closed geometry and common lighting calculations.

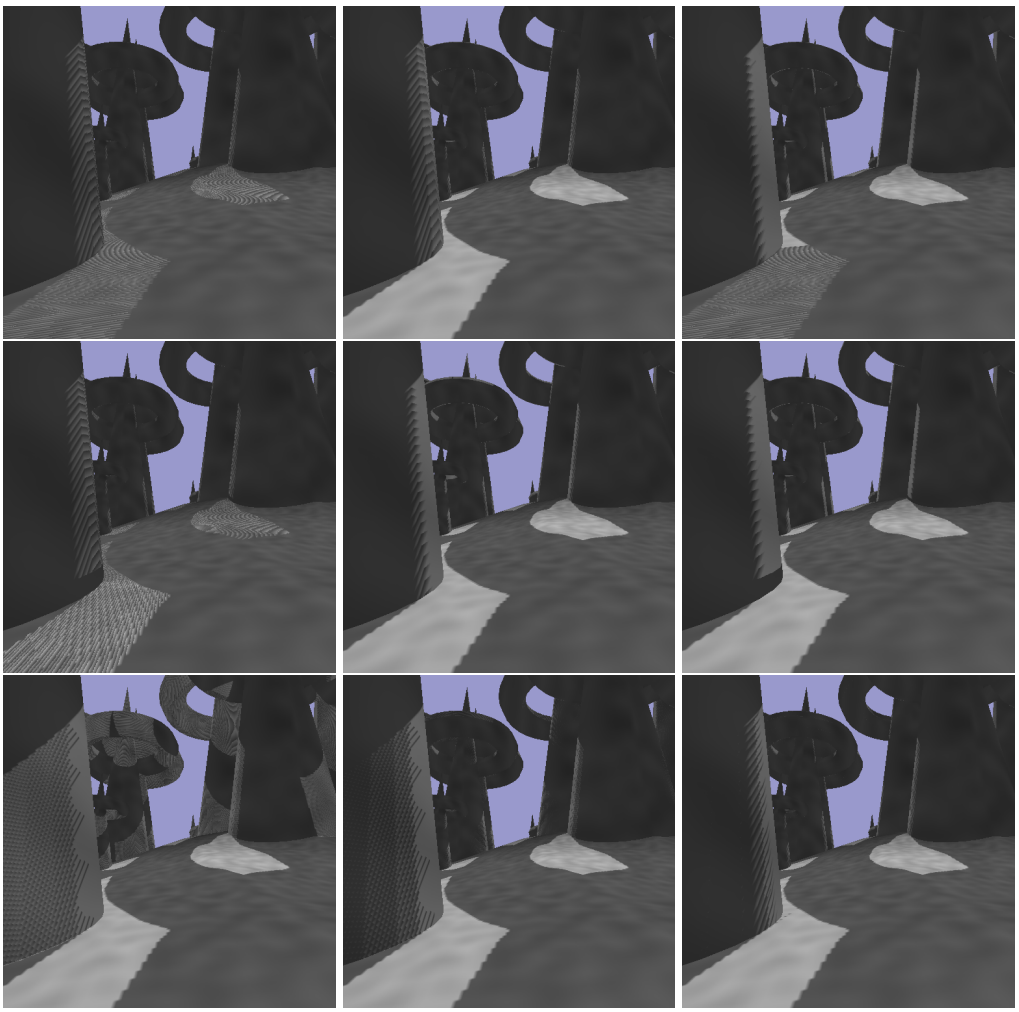

Fig. 5.6: *Left:* no biasing; *Middle:* constant biasing; *Right:* slope-scale biasing; *Top:* hyperbolic z-distribution; *Center:* linear z-distribution; *Bottom:* back-side rendering

Chapter 6

Solutions to projection aliasing

In this chapter we will discuss various methods to alleviate the problem of projection aliasing. After a general discussion in Section 6.1, we will talk in Section 6.2 about how the normal scene lighting can help to hide projection aliasing. Finally Section 6.3 investigates blurring to hide projection aliasing and increase the realism of the resulting shadows through softer shadow edges.

6.1 Exact solutions

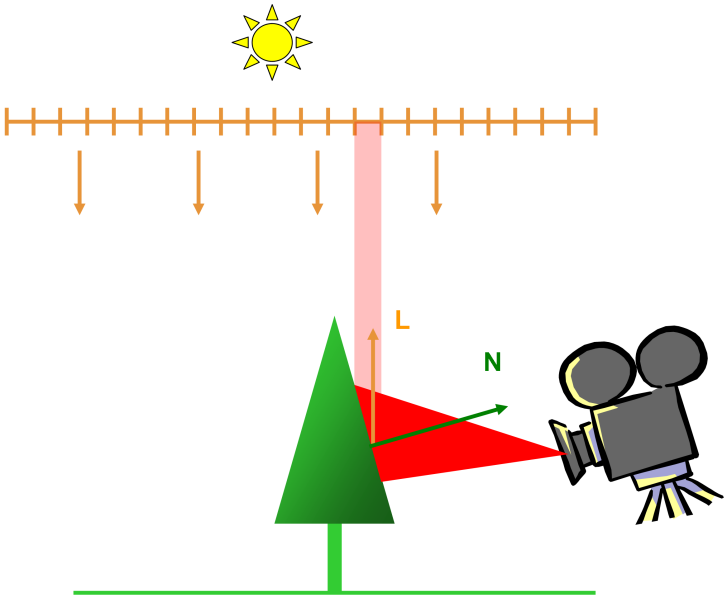

Fig. 6.1: The angle between the light vector L (orange) and the surface normal N (green) determines the amount of projection aliasing.

For a fixed eye position, the amount of projection aliasing depends on the angle between the light vector L and the surface normal N (see Figure 6.1). If

this angle is small, a small amount of projection aliasing occurs. If this angle is big, approaching $90°$, the maximum of projection aliasing for this constellation is reached.

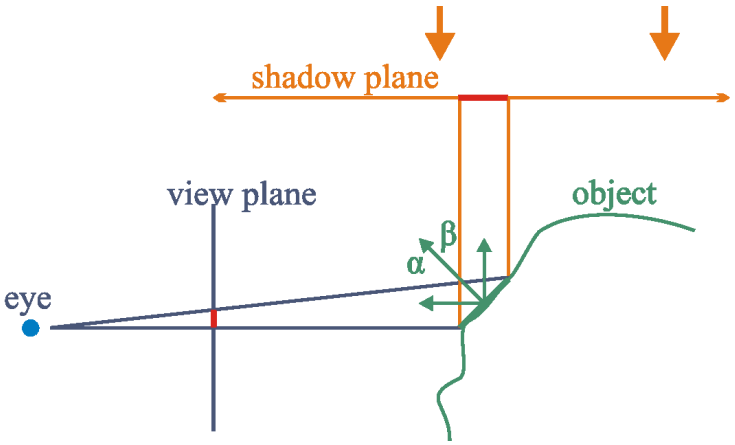

Fig. 6.2: The ratio between the angle β, the angle between the light vector and the surface normal, and the angle α, the angle between the view vector and the surface normal, determines the amount of projection aliasing.

For a variable eye position the amount of projection aliasing (see Figure 6.2) is determined by

$$\frac{cos(\alpha)}{cos(\beta)}. \tag{6.1}$$

If this ratio is ≤ 1 no projection aliasing is present. If it is > 1 projection aliasing is present.

It is difficult to counteract this shadow mapping problem because it is dependent on the scene geometry. Therefore projection aliasing cannot be solved by a simple global method that operates on the whole scene, but requires a detailed analysis of the scene geometry. A correct solution to projective aliasing remains an open problem for real-time shadow mapping approaches.

Solutions exists that work at interactive speed. For example hierarchical shadow maps [FFBG01], or the irregular z-buffer [JMB04] [AL04]. These approaches are already discussed in Chapter 4 in more detail. Also more advanced shadowing methods, like soft shadowing, can reduce projection aliasing artefacts, through the additional visibility information used in these approaches or through the softer nature of the resulting shadows, which hides a number of artefacts.

6.2 Lighting

Luckily we don't need to solve projection aliasing if we can hide it. Scene lighting is normally not entirely determined by a shadow map, but it is common to use other lighting approximations too. For example, Lambert's law,

$$I = I_L * cos(\alpha) \tag{6.2}$$

where I is the intensity at a point, I_L is the intensity of the light and α is the angle between the light vector L and the surface normal N (see Figure 6.1). Equation 6.2 is a good model for the lighting of a perfectly diffuse material. This equation gives a high intensity for points on a surface roughly perpendicular to the light direction and a low intensity, little illumination, for points on a surface roughly parallel to the light direction. If we compare this to the cases with a lot of projection aliasing and the cases with little projection aliasing, we deduce that in the former case, the large amount of projection aliasing is little illuminated and therefore hidden from the eye of the observer.

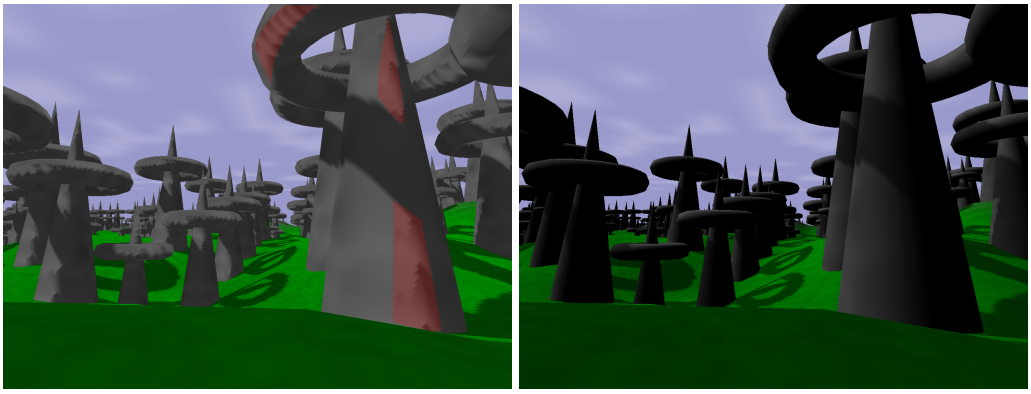

Fig. 6.3: The image quality of a scene with severe projection aliasing (*left*) is greatly improved by applying diffuse lighting (*right*).

In Figure 6.3 we can see on the left side an unlit scene with a shadow map applied and noticeable projection aliasing artefacts. We have marked three areas (red transparent areas) of severe projection aliasing as samples, but note that many other areas of projection aliasing are present in this image. On the right side diffuse lighting is applied. Very noticeably the projection aliasing artefacts are hidden in the dark areas created through the lighting.

But in real-time graphics lighting is often simulated in partitioning it in three parts (Phong illumination model):

- ambient part: $I_{amb} = I_L(amb)$

- diffuse part: $I_{diff} = I_L(diff) * cos(\alpha)$
- specular part: $I_{spec} = cos(\rho)^{shi}$

and the properties of the light are partitioned into two parts: the ambient factor $I_L(amb)$ and the diffuse factor $I_L(diff)$. The ambient part should simulate light that is so scattered around that no inherent direction is noticeable. It is only dependent on the ambient factor of the light. For hiding projection aliasing this part is unsuited because it can only raise the lighting, created by the diffuse term. Also the ambient part should never get shadowed because it has no incident light direction.

The diffuse part of the lighting we have already discussed. This part represents the light intensity of a given point from a perfectly diffuse object. It only depends on the incident angle α of the light-rays and the diffuse factor of the light. The diffuse part is our great hope for hiding projection aliasing. It is well suited for concealing the artefacts of projection aliasing.

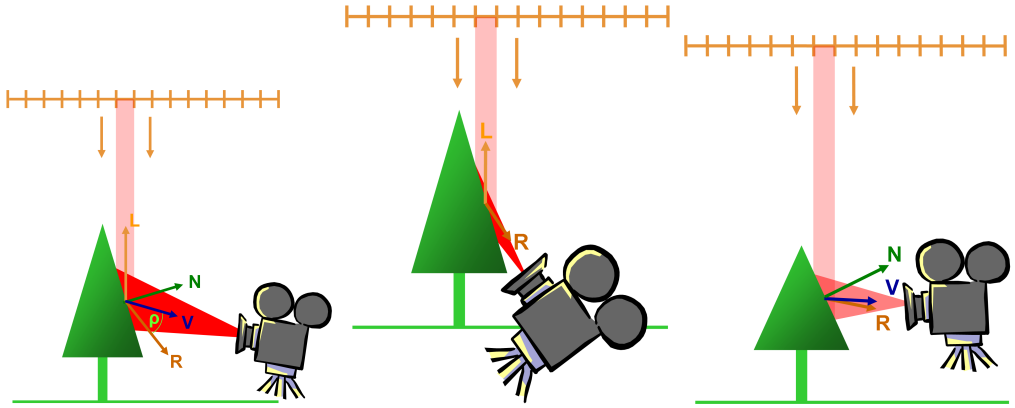

Fig. 6.4: The angle ρ (green) between view vector V (blue) and the reflexion vector R (orange) is used for the specular lighting calculation (*left*). Two border-cases for high specularity: The view vector and the reflection vector are closely aligned (*middle*, *right*).

The specular part should simulate perfectly specular (reflective) objects, like mirrors, through the generation of a highlight, when gazing into the reflection vector of the light. This part of the lighting simulation is dependent on the angle ρ and the shininess shi of the surface the point is on. The angle ρ is the angle between view vector V and the reflection vector R. The reflection vector R is calculated by reflecting the light vector L around the surface normal N (see Figure 6.4 left). This dependence of ρ on the view vector makes the specular part view dependent. The shininess is a material property and can be chosen when specifying the material. A greater shininess leads to a more concentrated highlight, while a smaller shininess leads to a dumper surface and a more diffuse highlight. The specular part is

biggest when the view vector is aligned with the reflection vector (see Figure 6.4 middle and right). The specular part is at its maximum, when the angle between the incident light and the emitted light are equal.

For projection aliasing exactly the reverse is true. This means that perspective aliasing is not present, when the specular lighting is at its maximum. All in all the specular part of the lighting therefore poses no problem in respect to projection aliasing.

On analysing the three different parts of the lighting simulation in real-time graphics for our goal, hiding projection aliasing, we can conclude with three rules:

- ambient part: choose as small as possible if per light; a global ambient is independent and therefore poses no problem;
- diffuse part: is good for hiding artefacts;
- specular part: no specular part inside shadows;

6.3 Blur

Another method to hide projection aliasing is blurring of the applied shadow map. Direct blurring of a shadow map does not yield a correct result because a shadow map contains depth values. Blurring is a weighted average calculation of neighboring pixels. If we blur depth values we would calculate a new depth at the processed pixel, lying somewhere in between the pixel and its neighbor's original depth. This makes no sense in respect to the following depth compare with the transformed fragments from eye-space. The depth profile stored in the shadow map would change and the shadow map test would give wrong results, especially in parts where the shadow map has discontinuities.

Fig. 6.5: The shadow map is applied to the unlit and unicoloured geometry.

Therefore blurring must be applied after the shadow map test is performed. This leads to the idea of an eye-space blur. The main idea is to apply shadow mapping onto unlit, unicoloured geometry, generating a monochrome or grey-scale (grey values originate from percentage-closer-filtering) image of the mapped shadows (see Figure 6.5).

This image can be blurred to generate a 2D-texture that can be applied as an intensity lookup texture for the lighting calculation in the final rendering. As can be seen in Figure 6.6, projection aliasing can be removed with a high enough blur at the cost of shadow details.

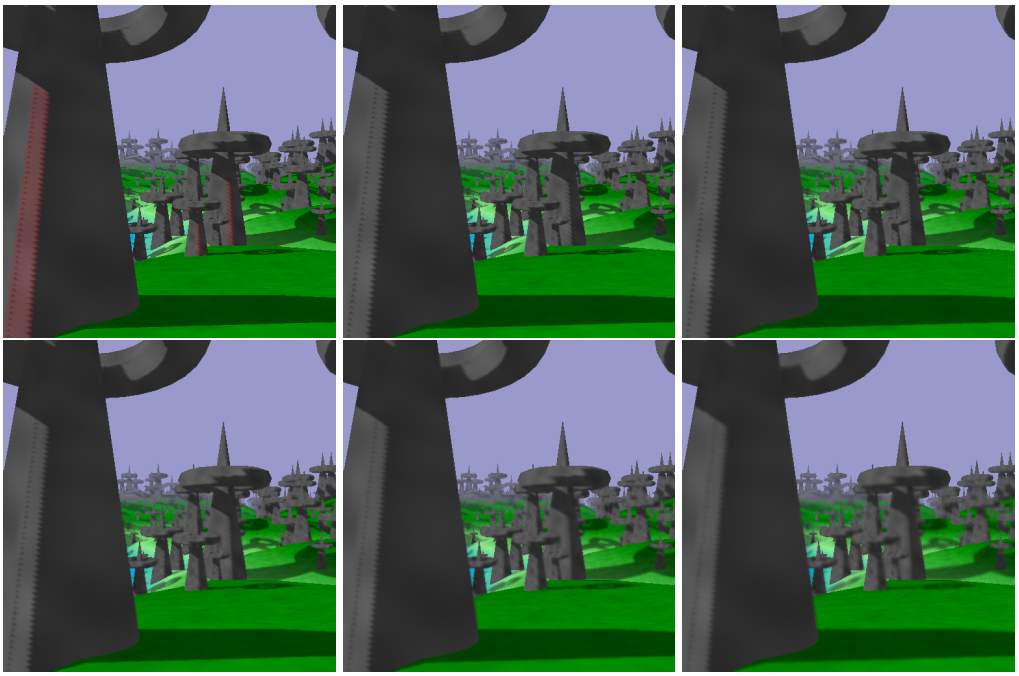

Fig. 6.6: The same scene starting with no shadow map blur and with 1x, 2x, 4x, 8x, and 16x blur. Notice the regions marked in red with projection aliasing artefacts and how the artefacts disappear with the increasing blur.

When we look closer at the blurred shadow maps in Figure 6.6, we can soon identify the drawback of this solution: While shadows in the front profit highly from a large blur, shadows in the back are overblurred and lack any detail. The solution to the loss of shadow detail in the distance is to use a depth-dependent blur. Near the viewer, the shadow map is blurred more, and with increasing depth the shadows are less and less blurred, to preserve shadow details in the distance. This is successful because the projection aliasing artefacts in the distance are generally much smaller (in terms of the pixel area) than the ones near the view point.

6.4 Summary

An open problem for future work remains projection aliasing. As a problem that depends on the actual geometry in the scene it is difficult to find a fast method that can deal with it. The use of a common lighting scheme, including a diffuse lighting term, can hide most of the projection aliasing. Another method is the use of blurring. But our results show that some shadow details have to be sacrificed to remove projection aliasing with this method. In combining the two methods, we can produce images with little visible projection aliasing and less blur (see Figure 6.7).

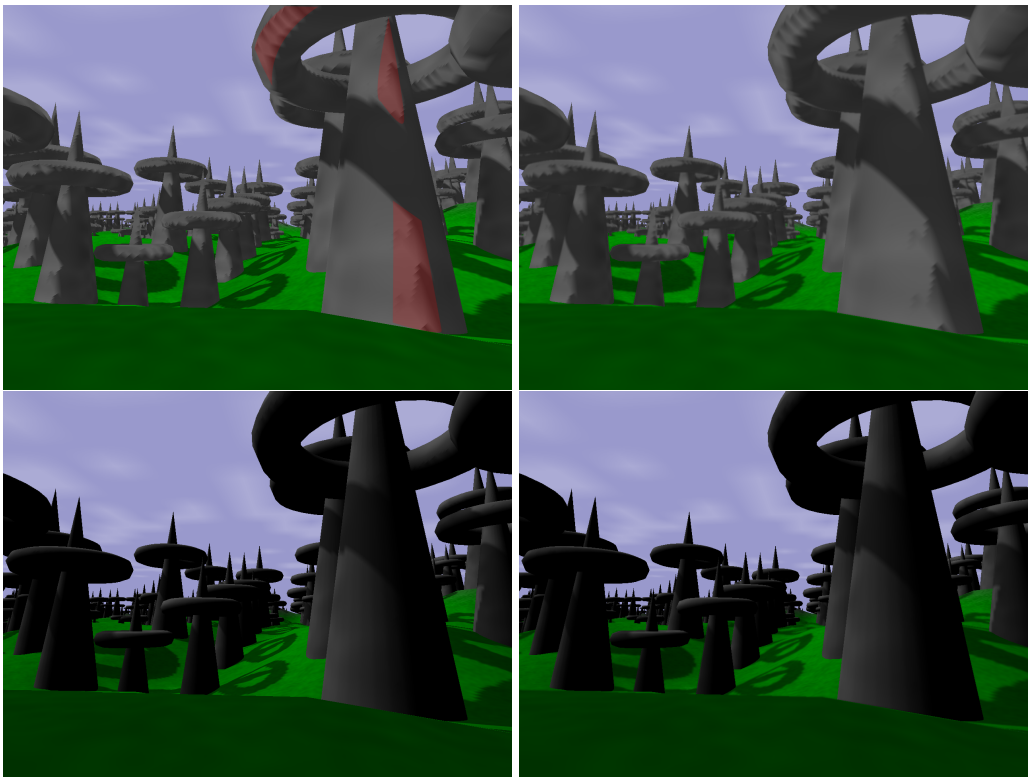

Fig. 6.7: Noticeable projection aliasing (*top-left*) can be reduced through blurring (*top-right*) and lighting (*bottom-left*). When blurring and lighting are combined, they can hide projection aliasing quite efficiently (*bottom-right*).

Chapter 7

Solutions to the perspective aliasing problem

In this chapter we will take a look at solutions that avoid the perspective aliasing error present in the shadow mapping process. We recommend Appendix A for a short introduction to perspective space.

In Section 7.1 we will explain the basic idea to solve projection aliasing. Section 7.2 describes the first fully hardware-accelerated algorithm, *perspective shadow mapping*, and its problems. Section 7.3 states improvements to the perspective shadow mapping algorithm. Finally Section 7.4 summarizes the findings of this chapter.

7.1 The idea

Perspective aliasing describes the problem that for a perspective view often the near shadows (with respect to the point of view of the eye) have too little resolution in the shadow map, while the farther away objects display sufficient resolution (see Figure 7.1).

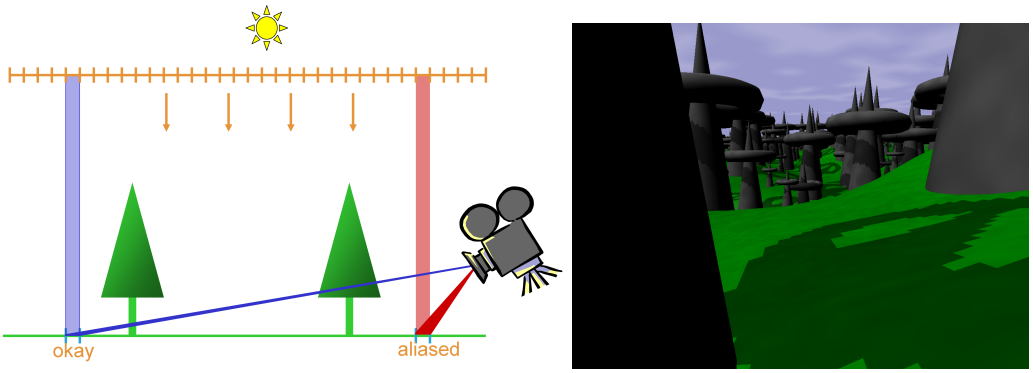

Fig. 7.1: In a standard shadow map (SSM) perspective aliasing leads to insufficient shadow resolution near the observer.

As discussed in section 3.1 this is due to the fact that in standard shadow maps (SSM) the resolution with which objects are stored wholly depends on the point of view of the light source. For point (spot) lights, this means objects near the light source get more resolution than those farther away, and for directional lights this means all objects get the same resolution because the light source is indefinitely far away.

A simple solution is to use multiple shadow maps. For example one for the near objects and one for the far objects. This means a partitioning of the view frustum into subsets and each subset is rendered into its own shadow map. The problem of this approach is the generation of disjunct sets of shadow casters and receivers. Each of these sets is rendered into its own shadow map. If no disjunct sets can be found, geometry has to rendered multiple times generating overlapping sets and degenerating performance, due to the additional rendering costs involved. On applying the shadow maps, a decision must be made which shadow map is used for an object present in different shadow maps.

Another idea is to redistribute the resolution of the shadow map in a better way. To minimize perspective aliasing, we need to introduce a view-dependent redistribution of the shadow map. Objects that are near to the eye should have a higher resolution, i.e., more space in the shadow map (see Figure 7.2).

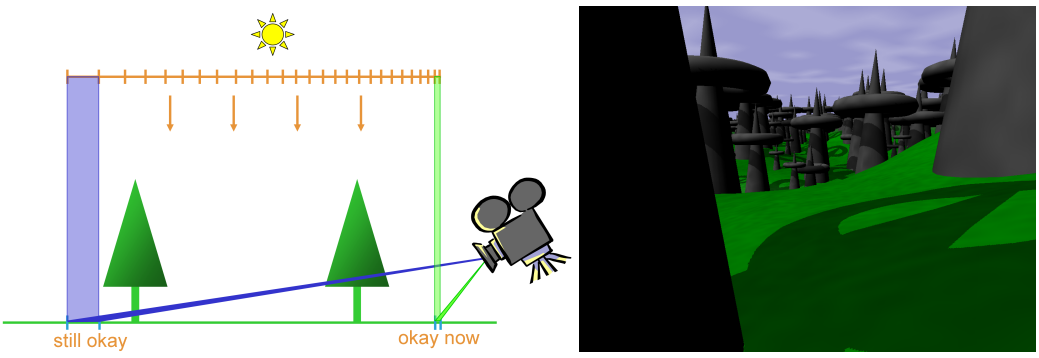

Fig. 7.2: A view-dependent shadow map, which redistributes the available resolution of the shadow map, can decrease perspective aliasing.

Complex data structures to encompass this functionality are for example hierarchical shadow maps [FFBG01], which use a hierarchical shadow map that is updated as needed. Or the irregular z-buffer [JMB04] [AL04] that uses a z-buffer with irregular resolution to minimize aliasing artifacts. These approaches are discussed in more detail in Chapter 4. But the main problem of these approaches is that they are not hardware accelerated and therefore the speed is currently just interactive.

The problem is to do a redistribution of the shadow map with the means of

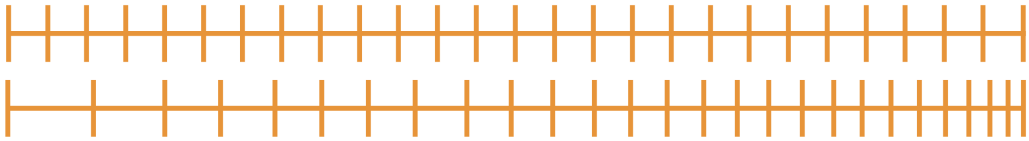

Fig. 7.3: Redistribution of the resolution of the shadow map is the key to minimize perspective aliasing. A standard shadow map has a uniform resolution distribution (*top*). A shadow map, like in Figure 7.2, has a nonuniform distribution (*bottom*).

current hardware. Figure 7.3 shows how such a redistribution can look like in a 1-dimensional shadow map. The main idea of the faster methods is the use of a perspective transformation to generate the desired redistribution. This simple idea is influenced by the fact that the perspective aliasing is introduced by the perspective transformation of the eye. In an orthogonal view, no perspective aliasing is present.

This trick makes shadow maps view dependent because the distance relationship from the eye to each object changes at each view change that includes a translation. Additionally the set of shadow casters to consider may change with every viewpoint transformation. This implicates a regeneration of the shadow map every single frame. But this means no additional cost for today's dynamic environments with dynamic lighting conditions because we already have to recalculate the shadow map each frame anyway. Finally, the regeneration each frame demands as small as possible calculation overhead for the used algorithms because the execution costs have to be paid each frame.

7.2 Perspective shadow mapping

The idea to use an additional perspective transformation on creating the shadow map to decrease perspective aliasing was first published in the paper *Perspective shadow maps* (PSM) by Stamminger and Drettakis [SD02]. The eye's perspective transformation is reused for this purpose, effectively moving the shadow mapping calculations into the post perspective space of the eye. This means that first we transform the scene into the post-perspective space of the eye, as we would for normal rendering, and afterward we construct in this space the light space used for shadow mapping. In Figure 7.4 we can see the effect of such a transformation. The view frustum (blue) is transformed into a rectangle. This should create an even distribution of the depth map samples in eye space.

Unfortunately in real-world scenarios, serious robustness and quality issues emerge [WSP04] and various special cases have to be considered, which complicates implementation.

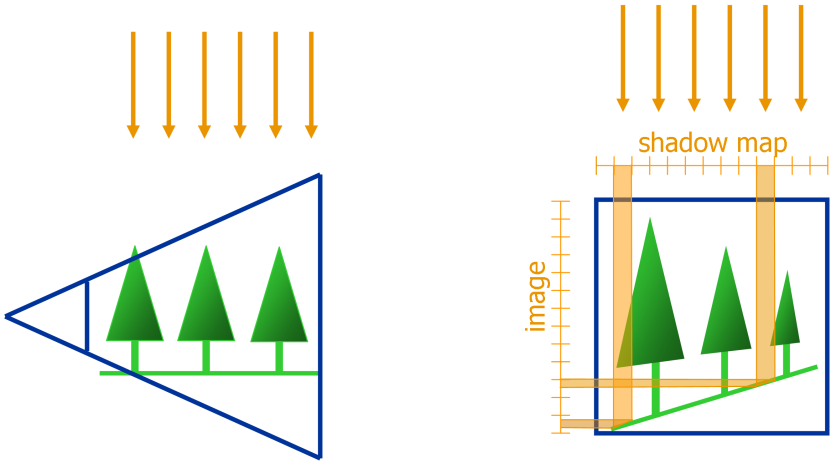

Fig. 7.4: In theory perspective shadow maps should provide an ideal redistribution.

7.2.1 Uneven z-distribution

The most severe problem of perspective shadow mapping, in terms of strength and frequency of occurrence, is the uneven z-distribution. The result of this is a very high shadow quality near the eye, but very low shadow quality farther away. The cause for this is the post perspective space of the eye, which introduces too much warp for scenes with a small near plane distance (see Figure 7.5).

To understand why this is the case we have to take a look at post-perspective z. With a common perspective transformation in column major style

$$\begin{pmatrix} c & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

with n denoting the near plane distance and f indicating the far plane distance, we get the post perspective z_{pp} by

$$z_{pp} = \frac{f}{f-n}\left[\frac{f+n}{f} + 2\frac{n}{z}\right]. \tag{7.1}$$

For common scenes with f much larger than n, $\frac{f}{f-n}$ and $\frac{f+n}{f}$ are both very close to 1, which leads to

$$z_{pp} \approx 1 + 2\frac{n}{z} \qquad \lim_{f\to\infty} z_{pp} = 1 + 2\frac{n}{z}. \tag{7.2}$$

As we can see the distribution of z_{pp} depends entirely on n, the near plane

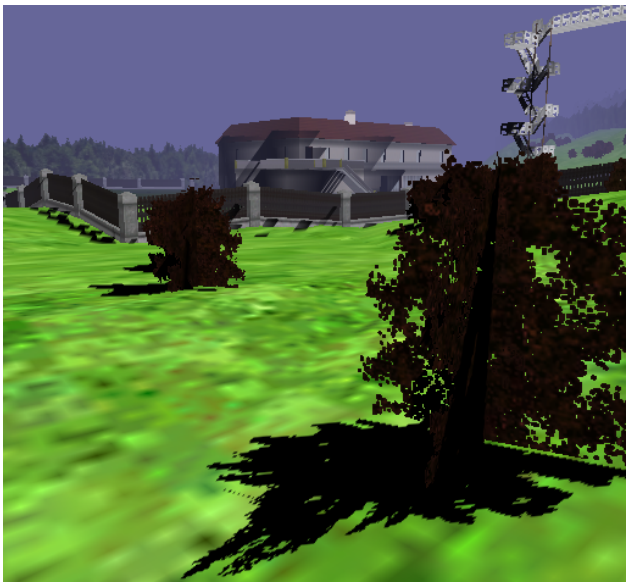

Fig. 7.5: The uneven z-distribution of perspective shadow maps leads to very high quality near shadows, but low quality far of shadows. This scene has a near plane distance of 0.1 and a far plane distance of 70.

distance. This means that the near plane distance highly influences the post perspective space.

z	-1	-1.25	-1.5	-2	-4	-16	-100
z_{pp}	-1	-0.6	$0.\dot{3}$	0	0.5	0.875	0.98

Tab. 7.1: The input z-values are in the range $[-n, -f]$ (here $[-1, -\infty]$) and are mapped to the output range $[-1, 1]$.

Table 7.1, for example shows for $n = 1$ and $f = \infty$ the z_pp values for input z values. We can see that half of the resolution of z_{pp} (range $[-1, 0[$) is used up for the first unit of z (range $[-1, -2]$). The rest of the available resolution (range $]0, 1]$) is used for the entire scene in the range from $[-2, \infty]$, which obviously is the much greater part of the scene. This means half of the space of the shadow map is used up for the first unit of z. Obviously shadows that are farther away have very little shadow map space left and are therefore of less quality. This problem was already detected by the authors of the paper and they recommended a read-back of the z-buffer to push away the near plane as far as possible. Later geometrical methods where used to give faster, but coarser results.

Therefore the shadow quality is hard to predict and usually bad for scenes with very near and far off objects, which are exactly the kind of scenes we would need perspective shadow maps for. The shadow quality can change rapidly from frame

to frame on viewpoint changes because of the possible changes in the near plane distance. This results in flickering shadows when animating the scene.

7.2.2 Shadows from behind

The next intricacy of perspective shadow maps are shadows from behind the point of view of the eye. In this case geometry behind the point of view of the eye can cast a shadow in front of the eye. The post perspective space of the eye has a singularity in the eye position. Upon perspective transformation, points on lines that intersect the eye (camera) plane can change their order. And for shadows from behind, such lines occur when geometry behind the point of view of the eye cast a shadow in front of the eye.

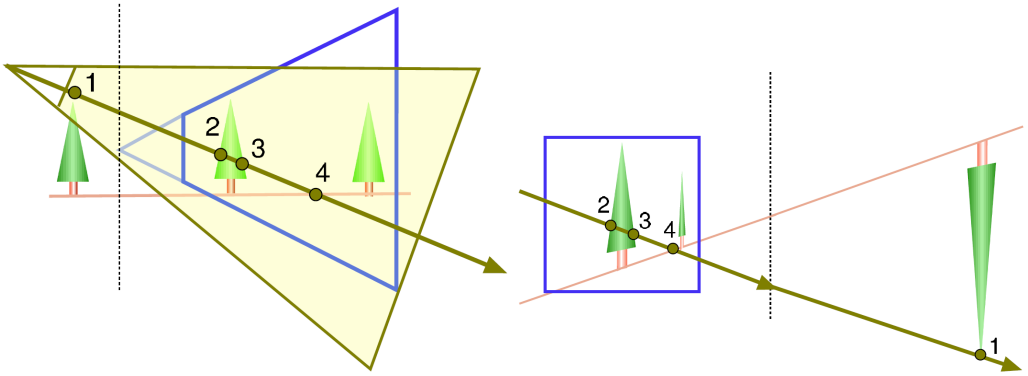

Fig. 7.6: In post perspective space lines are mapped to lines, but points on lines may change their order. On the *left* side a line through the camera plane with four sample points is shown. On the *right* side this line is transformed into post perspective space. The point (1) on the line behind the eye is mapped to the other side of the infinity plane.

In Figure 7.6 the point (1) is mapped on the other side of the infinity plane, moving it from the first position to the fourth position.

To avoid this situation all points we are interested in for shadow mapping must lie in front of the camera plane. Usually this amounts to the points inside the intersection body B (see Chapter 4 for details regarding B). One solution to this problem is to virtually move back the camera for the generation of the shadow map to get all points of B lying in front of the camera plane. In Figure 7.7 we can see the effect this has on the perspective transformation. On the *left* side the virtual camera is moved backwards until all points of interest are in front of the camera plane. Note that the far plane stays at the same distance to the point of view of the eye, but the near plane is adapted to encompass all of B. After the perspective transformation into the post perspective space of the virtual camera, the ordering of the points stays the same as in normal space.

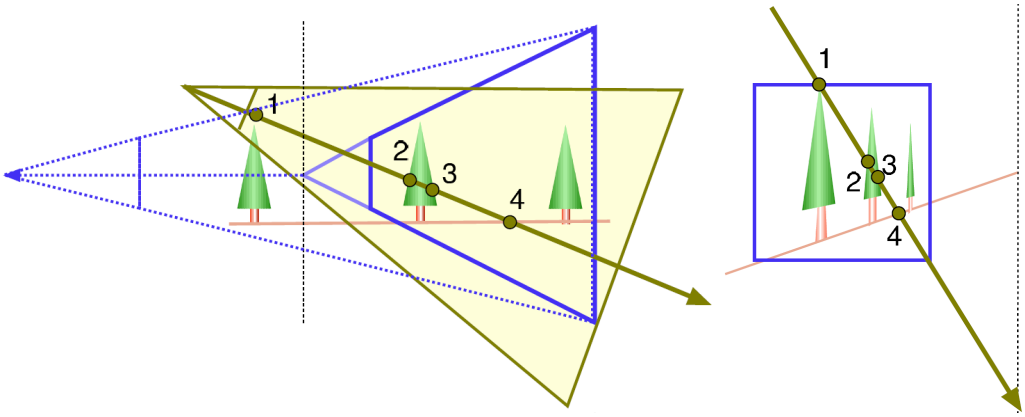

Fig. 7.7: A virtual camera move-back eliminates the problem of a changed point order in post perspective space.

This move-back alters the post-perspective space. And the result is decreased perspective foreshortening. As we move the virtual camera more backwards, the perspective foreshortening decreases further. If we reach an infinite move-back, no perspective foreshortening is present, resulting in an uniform shadow map and nullifying the effect of perspective shadow maps. This is the case for example for a light shining directly from behind into the view direction. A problem of this approach is discontinuity. We try to find the smallest move-back that encompasses all shadow casters that contribute to the shadows in the view frustum because as the move-back increases, the quality of the shadows decreases. This leads to advanced visibility methods that use hierarchical scene representations or similar methods to determine the needed shadow casters. With these methods, the move-back distance can change abruptly when a shadow caster suddenly has no influence on the visible shadows. This introduces the before mentioned discontinuity in shadow quality, which leads to fidgety shadows in animated scenes.

7.2.3 Light sources

Working in post-perspective space is certainly less intuitive than working in Euclidean space. For instance the type of the light source we are working with changes when entering the perspective space. The cause for this is the perspective transformation that can move the finite positions of point lights to infinity, changing them to directional lights, or moving directional lights at infinity to a finite position, converting them to point lights. Even an inverted point light is possible. This means this light source does not emanate light rays, but the light rays are converging on a single point. This is the case for light sources from behind the viewer, because these are mapped beyond the infinite plane through the

perspective transformation.

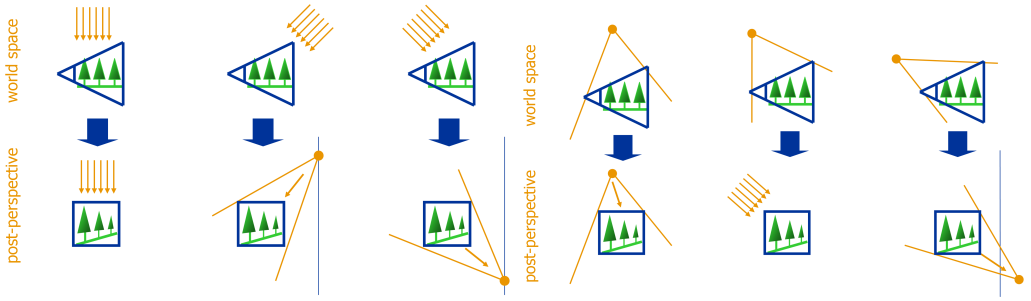

Fig. 7.8: The transformation into post perspective space changes the type of light source.

In Figure 7.8 we can see the six different cases. On the *left* side we see the three transformations of directional lights, and on the *right* side we see the three transformations of point lights. We can see that every directional light that is not parallel to the near plane of the view frustum is transformed into a point light in post perspective space. So only directional lights that are parallel to the near plane, for instance a light that shines directly from above the viewer, stays a directional light in post perspective space (see Figure 7.8, *left*, first case). For point lights the situation is similar. The only point lights that result in directional lights in post perspective space are the ones on the eye (camera) plane (see Figure 7.8, *right*, second case). The eye (camera) plane is here defined as the plane through the eye position with the view direction as normal vector.

The ideal case occurs when a directional light source is resulting in post perspective space. A directional light source does not introduce any additional perspective distortion, as would be introduced by a post perspective point light. The worst cases are point lights in post perspective space that are near the unit cube of the post-perspective intersection body B. These are bad cases because these lights need to have a very large field of view to encompass the whole unit cube. Regrettably post-perspective point lights are the majority case and point lights near the unit cube are frequent for scenes with an f much larger than n.

7.3 Improvements to perspective shadow mapping

Various papers concerning the problems of perspective shadow maps have been published. Chong's master thesis [Cho03], for instance, presented a reparameterization of PSM into a more general frame-work. In this frame-work, perspective shadow maps were analyzed in 2D. Based on this, Chong and Gortler [CG04] presented a shadow algorithm that is capable of calculating a perspective shadow map that gives optimal results for a chosen plane of interest.

Besides these more theoretical approaches, Kozlov [Koz04] investigated practical advancements of PSM to solve their problems. He introduced an inverse projection matrix that can eliminate the need for virtual cameras and the associated move-back. The idea is to construct a perspective transformation that allows a negative near plane and a positive far plane. This makes it possible to look beyond the infinity plane and restore the original order of points on lines intersecting the camera plane (see Section 7.2.2 for details to the problem). To bring this matrix to good use, the graphics hardware needs high precision shadow map depth-values. The author states that 24-bit fixed-point depth values are enough for reasonable cases. However, the introduction of floating-point 32-bit precision depth-values in new hardware mitigates this problem.

Kozlov further proposes to use a more elaborate determination of the intersection body B with *Unit cube clipping* limited to the bounding volumes of the shadow receivers. He argues that this method decreases the dependency of the shadow quality on the light position (see Section 7.2.3 for details on lights and PSM). A second method Kozlov uses to decrease the dependency of light position and resulting shadow quality is a cube shadow map. He uses the backfacing unit cube faces with respect to the light as cube-map faces. This means that up to six smaller shadow maps need to be calculated.

Trapezoidal shadow maps (TSM) [MT04] build on top of the idea of perspective shadow maps. The idea is to approximate the eye frustum as seen from the light with a trapezoidal. For this a new space, the trapezoidal space is introduced. The size of this trapezoid is maximized in the shadow map. Then a 2D perspective transformation is used to redistribute the shadow map resolution. This is used together with an iterative process that repeatedly determines the perspective transformation and so tries to minimize perspective aliasing.

Light space perspective shadow maps (LispSM) [WSP04] is a concurrent approach to the TSMs of the last paragraph. The shadow quality of both approaches is similar. This approach uses an additional perspective transformation that is applied after the light space is determined. The free parameter of this transformation determines the strength of the perspective warp. This parameter is calculated with a formula, derived from a perspective aliasing error analysis. This approach is handled in detail in Chapter 8.

Both approaches can handle most cases where PSM has problems well, but degenerate to the standard shadow map in the dueling frusta case. Here the light direction is roughly parallel to the view direction of the eye. Generally, these reparameterisations cannot increase the shadow quality for parallel view and light directions. One example of this instance is the miner's headlamp situation. In this situation the standard shadow map causes nearly no perspective aliasing. This means that the degradation to standard shadow maps is the logical and correct proceeding. Other instances of the dueling frusta case are less optimal, but are not

simply solved by a reparameterisation.

Recently a recipe for handling the duelling frust case with an extension to TSM was published on the internet [Ald04]. This method uses a single shadow map and divides it into four viewports that are used to render into adaptively. An update of up to four light views is, however, costly for polygon-rich scenes and additionally makes the implementation and the involved data structure more complicated.

7.4 Summary

This chapter has elaborated solutions to the perspective aliasing problems inherent in shadow maps. We discussed the most promising approach, Perspective shadow maps, and its problems. We showed the importance of this initial idea for more recent algorithms that overcome the limitations of PSM and provide robust and versatile shadow maps with a much lesser perspective aliasing problem. The following chapter 8 describes one of these methods in more detail: *Light space perspective shadow maps*, a technique developed by the author.

Chapter 8

Light space perspective shadow maps

In this chapter we describe the algorithm called *Light space perspective shadow maps* (LiSPSM) [WSP04]. We start from *Perspective shadow maps* (PSM) [SD02] in Section 8.1, where we state what we want from a shadow mapping algorithm and in which way we have to apply the ideas of PSM to meet our requirements. Section 8.2 relates methods to choose the free parameter of LiSPSM, n, which controls the strength of the perspective foreshortening. Section 8.3 gives some impressions how a reparameterization can improve, but also how a reparameterization can be in vain in certain cases. Finally Section 8.4 summarizes the findings of this chapter.

8.1 The setup

Perspective shadow maps sound good in theory (see Chapter 7 for details), but when it comes down to real world applications, a lot of problems pop up. As we have already described in Sections 7.2.1, 7.2.2 and 7.2.3, the main problems of PSM are:

- An uneven z-distribution
- Quality fluctuations with shadows from behind
- Quality dependence on the light source position
- Unintuitive post-perspective space

Our approach is to use the initial idea of using a perspective transformation to redistribute the shadow map resolution (further on called LiSPSM transformation), but to avoid the known pitfalls of perspective shadow maps. Our first observation is that the LiSPSM transformation does not have to be the perspective transformation used for the eye. The only things it has to satisfy, to be useful in our context, is to provide more resolution near the eye. Realizing this, we searched for

a way we could specify a transformation that satisfies all the properties we have not found in PSM:

- Better z-distribution (tunable)
- Shadows from behind should be no special case
- Less quality dependence on the light source position
- Remove unintuitive post-perspective space

The first step to our solution is to specify the LiSPSM transformation in light space. The light space is the local coordinate frame that is gained after transforming the geometry by the transformations of the light source. This includes a rotation and a translation for a directional light and additionally includes a perspective transformation for point (spot) light. This has the effect that a spot light is transformed into a directional light and can be handled the same way as a directional light.

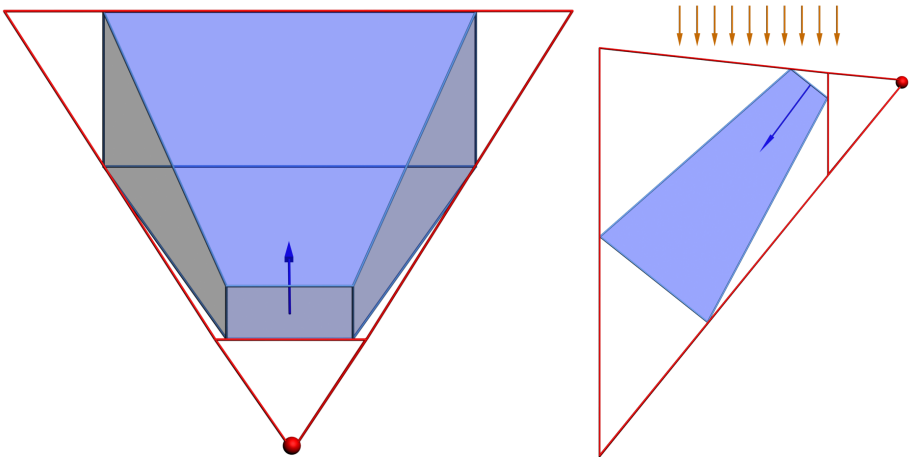

Fig. 8.1: *Light space perspective shadow maps* use a perspective transformation that has near and far planes aligned with the light direction. The view frustum is blue. The frustum of the LiSPSM transformation P is red. The light rays are orange. *Left:* The configuration as seen from the point of view of the light. *Right:* A side view of the same setting with the light-rays coming top-down.

In this space we specify the LiSPSM transformation so that the near and far plane of this transformation (of the frustum P) are parallel to the direction of the light (see Figure 8.1). This guarantees that light rays are mapped to light rays, and we avoid the light source conversion with many different cases, as required with PSM.

To achieve the effect of getting a better shadow resolution near the eye, we should position the near plane of P near the eye. Additionally P has to contain all relevant shadow casters and receivers. This means that P has to be focused to contain the intersection body B (see Chapter 4 for details regarding B). In Figure 8.2, we can see the effect such a transformation has. The scene (green) is warped. The parts of the scene near the eye are enlarged and the parts further away are shrunk. When this scene is afterwards written into the shadow map more resolution is granted to the larger parts.

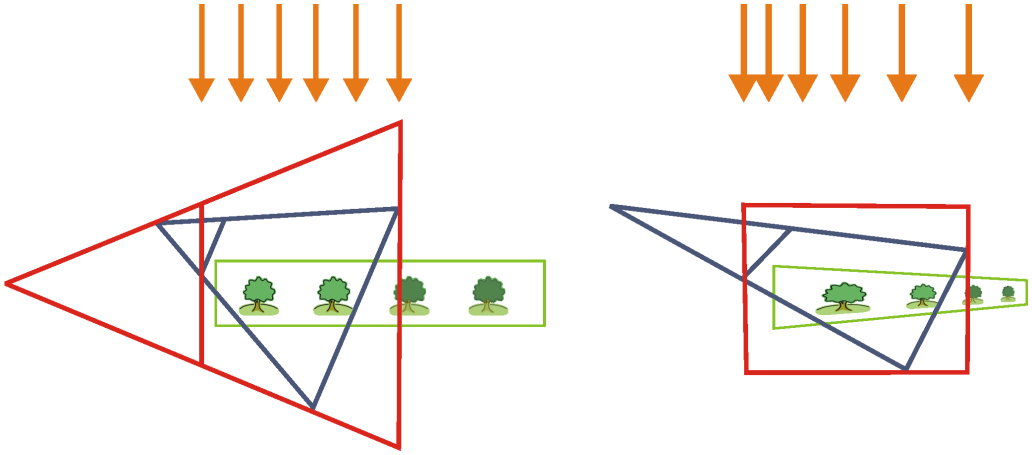

Fig. 8.2: *Light space perspective shadow maps* use a light space aligned perspective transformation for the redistribution of the shadow map resolution. We see the scene before (*left*) and after the application (*right*) of the LiSPSM transformation.

What we still need to specify to completely determine the frustum P is the projection center C (the red sphere in Figure 8.1). The distance of C to the near plane of P will determine the strength of the perspective warp. This means the greater this distance is, the more LiSPSM will resemble uniform shadow maps, and the smaller this distance is, the more space on the shadow map is used for objects near the near plane of P and less space for distant objects. We will later show how to choose this distance n in a way that the projection error for objects at the near plane and objects at the far plane is equalized. The other two remaining coordinates of C can be chosen to resemble the eye coordinates in light space. This provides a similar perspective distribution as the perspective transformation of the eye.

A fact that was already stated before is that shadow maps with a shadow map resolution redistribution tailored to the eye have to be calculated every frame or at least at every major view change, because otherwise the redistribution of the shadow map is not valid anymore. B also becomes invalid and some newly visible shadow caster are not sampled into the shadow map. This is true for focused

uniform shadow maps, for PSM, for TSM and for LiSPSM, as well as others, too. To put this problem into perspective, we only want to state that in dynamic environments, a regeneration of shadow maps every frame is mandatory anyway.

8.2 The free parameter n

The parameter n denotes the near plane distance of the projection center C and thus controls the strength of the perspective foreshortening (see Figure 8.3 and Figure 8.4). To find a "good" n means to find a level of perspective foreshortening that minimizes projection aliasing errors in the resulting shadows.

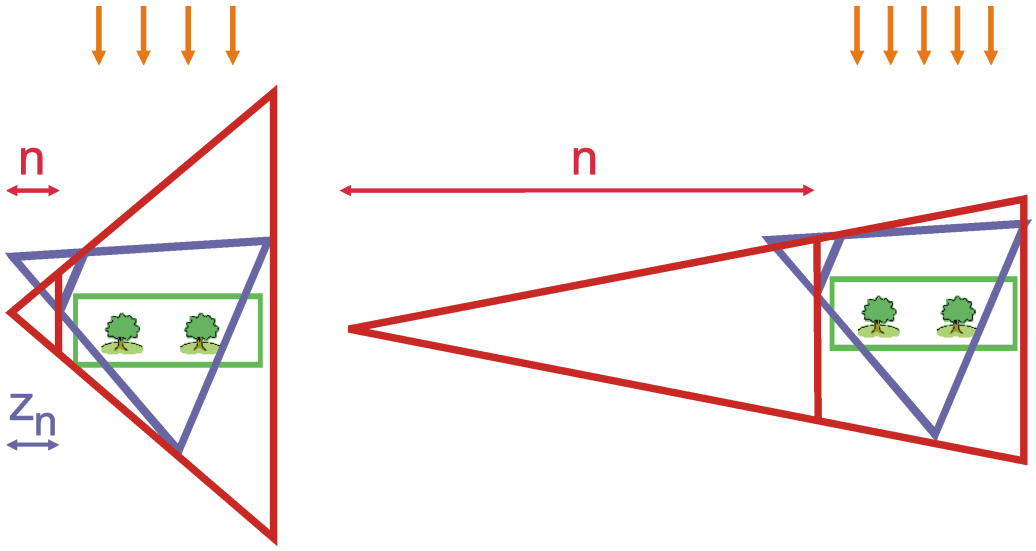

Fig. 8.3: Changing n leads to different warps of the scene. *Left:* n is small (in the size of z_n leading to similar warps as with PSM. *Right:* n is big leading to hardly any warp.

Various methods of choosing n are possible. For instance empirically gathered tables or formulas could be used. A very simple method would be to choose a constant n. To understand why a constant n is not advisable and why we adopted a more mathematical approach, we must first understand on what parameters projection aliasing is dependent.

8.2.1 The aliasing formula

Aliasing in shadow mapping is caused by undersampling. The process is depicted in Figure 8.5 for a small edge. Note that this figure shows a directional light because we can make the same observations as in the complexer point light case.

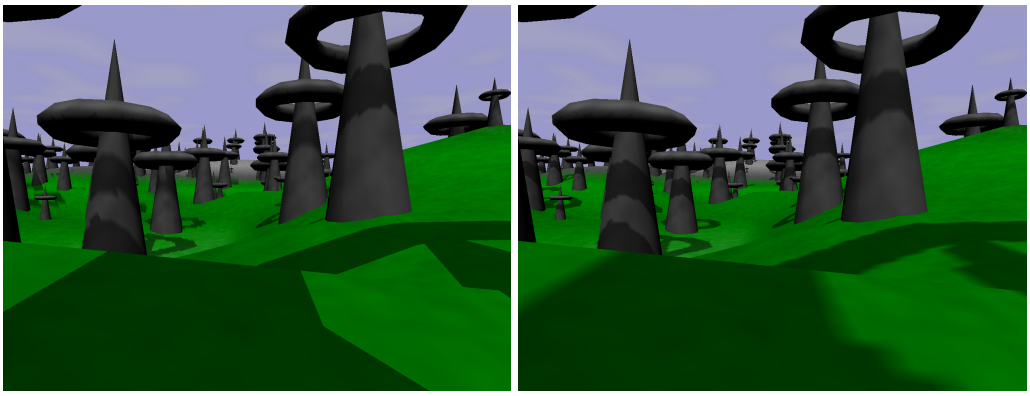

Fig. 8.4: *Left:* n is chosen too small and too much shadow map resolution is used for near shadows. *Right:* n is chosen too big and too little shadow map resolution is used for near shadows.

We assume that the shadow map has texels in the size of $ds \times ds$ in the local parameterisation of the shadow map, representing the shaft of rays passing through it. From the light source a shaft of width ds hits the ground at length dz. This edge has length $dz/\cos\beta$. When seen from the point of view of the eye (before perspective projection) this edge assumes the length $dy = dz\frac{\cos(\alpha)}{\cos(\beta)}$. After the perspective projection of the eye the edge becomes $dp = \frac{dy}{z}$ wide on the screen. The shadow map aliasing error is the ratio of dp in respect to ds, so we have to construct a term for $\frac{dp}{ds}$. If $\frac{dp}{ds} = 1$ no undersampling and aliasing is present. If $\frac{dp}{ds} > 1$ undersampling occurs and aliasing is present. Putting our observations together this leads to

$$\frac{dp}{ds} = \frac{1}{z}\frac{dz}{ds}\frac{\cos\alpha}{\cos\beta}. \tag{8.1}$$

This formula can be divided into three parts:

- Projection: $\frac{\cos\alpha}{\cos\beta}$
- Perspective: $\frac{1}{z}$
- Parameterization: $\frac{dz}{ds}$

Projection is responsible for projection aliasing, which we cannot change because this part depends on the scene geometry (see Figure 8.5 green parts) itself. What we want to do is to reduce perspective aliasing $\frac{1}{z}$ through a reparameterization of the shadow map $\frac{dz}{ds}$. Perspective aliasing is caused by the perspective projection of the viewer. If a perspective foreshortening effect occurs along the

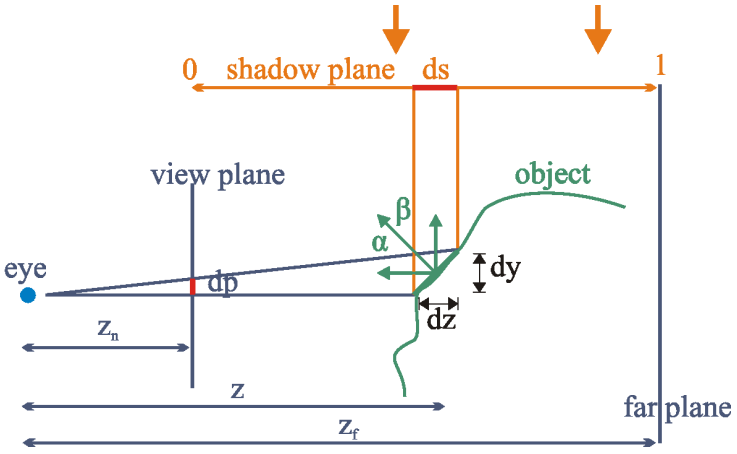

Fig. 8.5: Aliasing in shadow mapping.

shadow map plane (on a axis roughly parallel to the shadow map plane) we can counteract the induced perspective aliasing by a shadow map reparameterization.

This means that we have the greatest influence on perspective aliasing when the view direction is parallel to the shadow map plane (see Figure 8.6). We will therefore look at the perspective aliasing errors of different reparameterizations of shadow maps $s = s(z)$ for this special case.

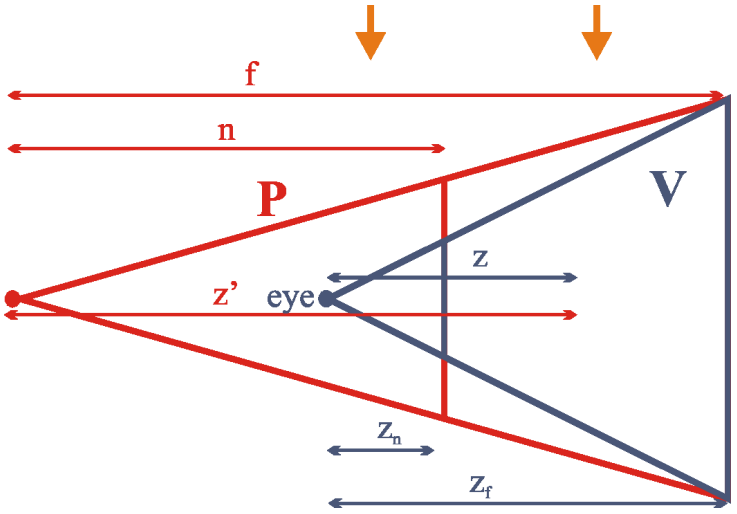

Fig. 8.6: The optimal case for a shadow map reparamterization, the view direction is parallel to the shadow map plane.

8.2.2 Reparameterizations of shadow maps

To remind us of the criteria for an optimal parameterization, we state that such a reparameterization would make $\frac{dp}{ds} = 1$ (constant) over the whole depth range.

We will start by the most common reparameterization, the uniform "reparameterization". Here

$$s \sim z \Rightarrow \frac{ds}{dz} \sim 1 \Rightarrow \frac{dp}{ds} \sim \frac{1}{z} \tag{8.2}$$

This means that $\frac{dp}{ds}$ is large when $\frac{1}{z}$ is large, so the error is greatest on the near plane and diminishes at the distance.

Next we take a look at perspective shadow maps. Here

$$s \sim \frac{1}{z} \Rightarrow \frac{ds}{dz} \sim \frac{1}{z^2} \Rightarrow \frac{dp}{ds} \sim z \tag{8.3}$$

So we have a small error at the near plane and a *linear* increase in error as we move away from the near plane.

The ideal reparameterization can be constructed by setting (assuming no projection aliasing is present)

$$\frac{dp}{ds} = \frac{dz}{ds}\frac{1}{z} = 1 \Rightarrow ds = \frac{dz}{z} \qquad s = \int ds = \int_{z_n}^{z} \frac{dz}{z} = \ln \frac{z}{z_n} \Rightarrow$$
$$s \sim \ln z \Rightarrow \frac{ds}{dz} \sim \frac{1}{z} \Rightarrow \frac{dp}{ds} \sim 1 \tag{8.4}$$

So the ideal reparameterization is logarithmic.

For LiSPSM we have to find the reparameterisation that is created through P. In this case $\frac{dp}{ds}$ depends on n. Figure 8.6 shows the needed parameters for this analysis. z_n and z_f denote the near respective the far distance of the view frustum of the eye. n and f are the same parameters for the frustum P. z and z' describe the distance to an arbitrary point in the view frustum respective in the frustum P. It is simpler to describe the effect of P on s in terms of the parameters of P. So we write our equations in terms of z'. z' has the following relationship to s

$$s = \frac{1}{2} + \frac{f+n}{2(f-n)} + \frac{nf}{z'(f-n)} \tag{8.5}$$

From Figure 8.6 we can easily deduce that $z' = z - z_n + n$ and substituting it into the formula

$$s = \frac{1}{2} + \frac{f+n}{2(f-n)} + \frac{nf}{(z-z_n+n)(f-n)} \tag{8.6}$$

We can differentiate $\frac{ds}{dz}$

$$\frac{ds}{dz} = \frac{nf}{(z-z_n+n)^2(f-n)}. \tag{8.7}$$

From Figure 8.6 we can easily deduce that $f = n + z_f - z_n$ and substituting it into the formula

$$\frac{ds}{dz} = \frac{n(n + z_f - z_n)}{(z - z_n + n)^2(z_f - z_n)} \tag{8.8}$$

and finally inserting this result into Equation 8.1 (with the assumption of projection aliasing $\frac{\cos\alpha}{\cos\beta} = 1$), we get

$$\frac{dp}{ds} = \frac{1}{z}\frac{(z - z_n + n)^2(z_f - z_n)}{n(n + z_f - z_n)}. \tag{8.9}$$

This equation has one minimum at $n - z_n$. The two maxima are therefore at the borders of the relevant range $[z_n, z_f]$. We opted for a solution of this equation that equalizes the error at these maxima (for details see [WSP04]), leading to

$$n_{opt} = z_n + \sqrt{z_n z_f} \tag{8.10}$$

Figure 8.7 compares the errors of the described shadow map reparameterizations.

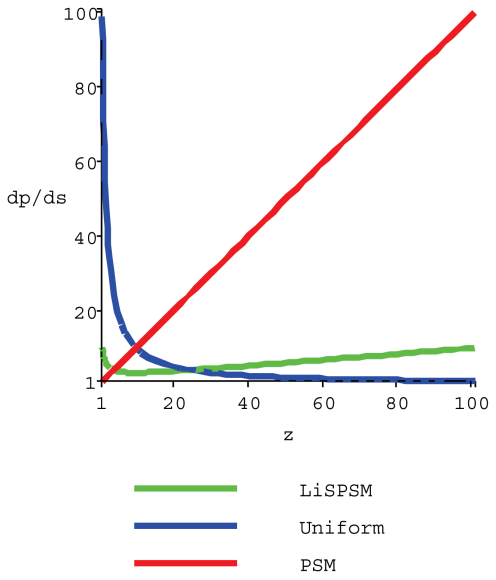

Fig. 8.7: Perspective aliasing errors plotted against z-coordinate for different shadow mapping techniques.

8.2.3 The general case

It is important to note that a reasonable reparameterization should consider the angle between the view direction and the shadow map plane. As already noted the influence of a reparameterization is greatest when the view direction is parallel to the shadow map plane. This influence decreases till the view direction is perpendicular to the shadow map plane (parallel to the light direction). In this case a reparameterization has no positive effect and therefore should converge to the uniform parameterization of the shadow map.

We can incorporate this by including an angle γ into our formula that accounts for the tilt of the view vector from the light direction (see Figure 8.8).

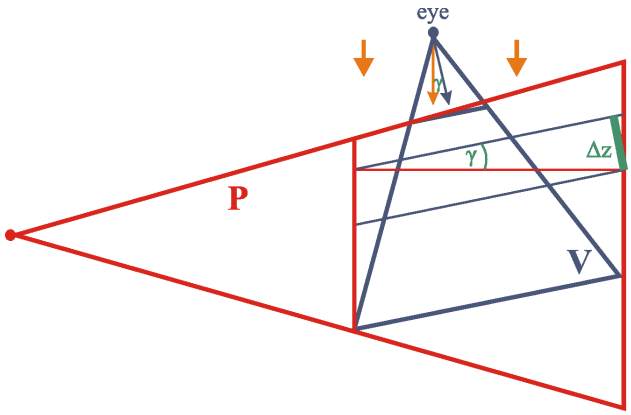

Fig. 8.8: The z-range affected by the reparameterization is small when the view direction gets near the light direction.

$$n_{opt} = \frac{z_n + \sqrt{z_n(z_n + d\sin\gamma)}}{\sin\gamma} \tag{8.11}$$

In this formula, z_n is the near plane distance of the camera, γ is the absolute value of the angle between view direction and light direction, and d is the extent of the frustum of P in the light space z direction. When looking at the optimal case formula, we can identify that we removed z_f and substituted it by $\Delta z = dsin\gamma$, with $d = z_f - z_n$.

On checking our new formula, we get that for $\gamma = 0$ (parallel view and light direction) n_{opt} goes to infinity and therefore a uniform shadow map results, and for $\gamma = \frac{\pi}{2}$ (parallel view and shadow plane) the original, optimal case formula, results.

$$n_{opt} = \frac{z_n + \sqrt{z_n(z_n + (z_f - z_n)\sin(\frac{\pi}{2}))}}{\sin(\frac{\pi}{2})} =$$

$$= \frac{z_n + \sqrt{z_n(z_n + (z_f - z_n)1)}}{1} = \\ = z_n + \sqrt{z_n z_f} \tag{8.12}$$

8.2.4 A new formula

This formula works fine for directional lights. For point lights, however, we need to generalize the formula to take into account the change in scale induced by the perspective projection of the point light. We also need to take into account that the view frustum may be clipped by the scene bounding box or the light frustum. We have therefore derived a new formula which is applicable to all cases:

$$n_{opt} = \frac{d}{\sqrt{\frac{z_1}{z_0}} - 1} \tag{8.13}$$

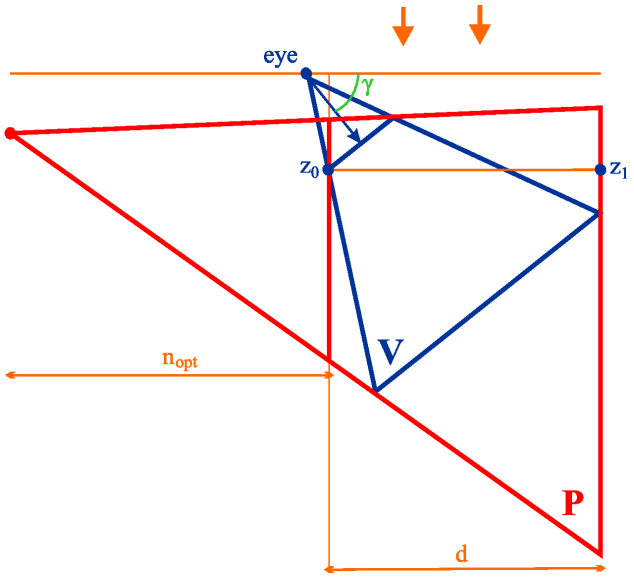

Fig. 8.9: Construction of z_0 and z_1 (for simplicity, V is shown instead of B.

The values z_0 and z_1 (signed!) represent the range of eye-space z-coordinates for which the perspective error should be equalized (see Figure 8.9). Note that when setting $z_0 = -z_n$ and $z_1 = -(z_n + d\sin(\gamma))$ (the minus sign arises because of the $-z$-axis), this formula is identical to the one given above. It is valid for directional lights where the view frustum is contained in the scene bounding box.

8.3 Good and bad cases for reparameterizations

The use of a perspective transformation gives the best results for perpendicular view and light directions because in this case the perspective transform can influence the whole depth range of the view frustum (see Figure 8.10). This is also the case where the most perspective aliasing is present. In the case of parallel light and view vectors, no perspective aliasing is present and the perspective transformation can only worsen the quality of the shadow map. This is called the duelling frusta case (see Figure 8.11).

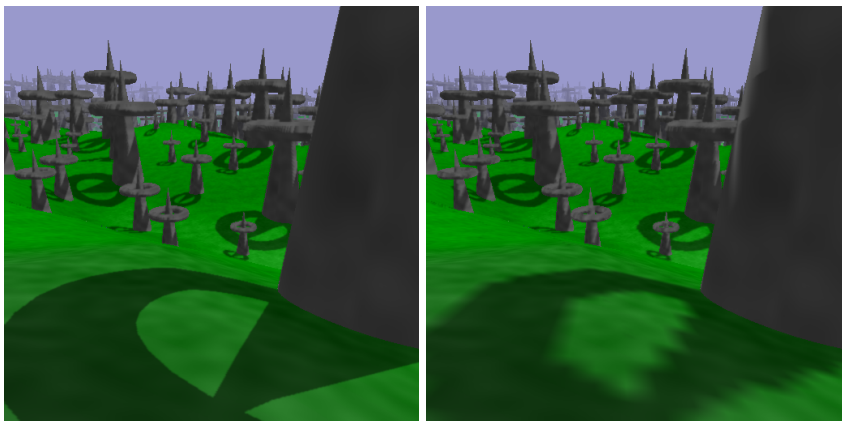

Fig. 8.10: For near perpendicular view and light directions LispSM (*left*) gives the best results. Uniform shadow mapping (*right*) has much more perspective aliasing.

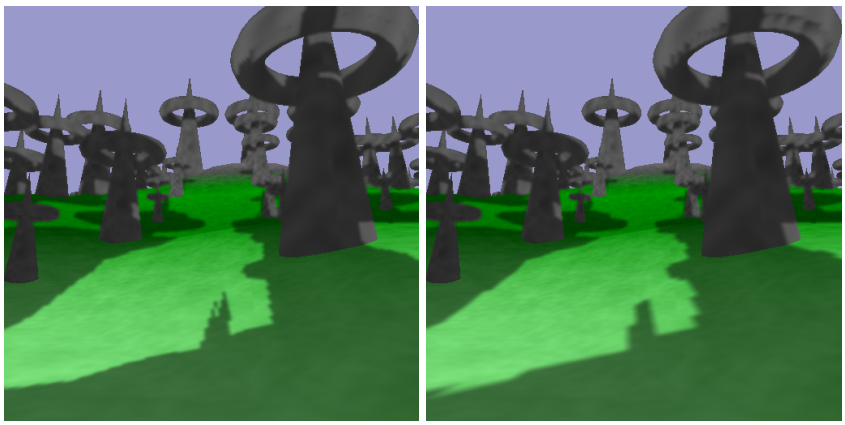

Fig. 8.11: For near parallel view and light directions LispSM (*left*) converges to uniform shadow mapping (*right*), making the shadow borders blocky.

8.4 Summary

This chapter gave an in-depth analysis of Light space perspective shadow maps. We explained the shortcomings of previous approaches and what we gained by using our new method, located in light-space. Additionally we gave considerable inside information about the choice of n and how an "optimal" parameter can be found for directional and point lights. This theoretical background should enable the reader to use his own analogous proceedings, if the presented formulas are not sufficient for his special needs. The next chapter will discuss how the theory developed in this chapter can be used to implement a robust and versatile shadow mapping algorithm that can be used in real-world scenarios (see Figure 8.12).

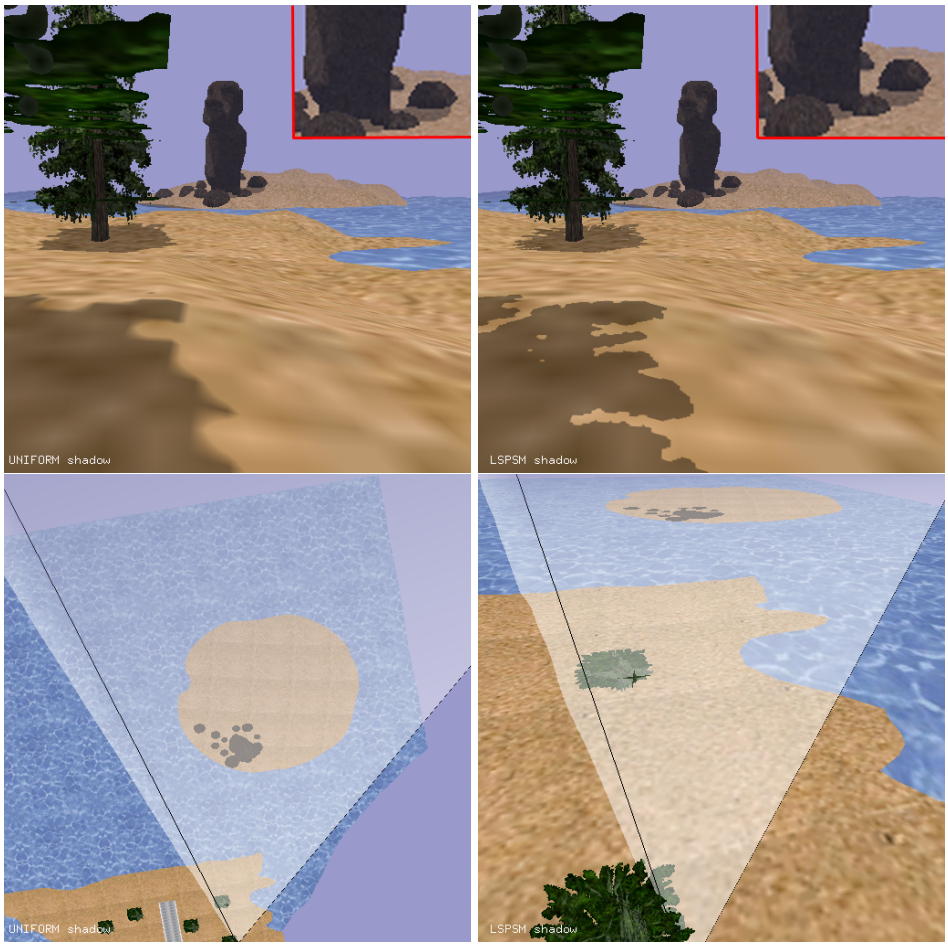

Fig. 8.12: Illustrates the effect of the LiSPSM perspective warp (*right*) in comparison to standard shadow mapping (*left*). The lower images show the corresponding light views

Chapter 9

Implementing shadow maps in real-world scenarios

In this chapter we tackle the intricacies of applying shadow mapping to a real-world scenario. We chose a demanding challenge for any real-time shadow algorithm: a large-scale, polygon-rich and dynamic environment. We use the techniques presented in various chapters of this thesis to solve the problems encountered. This will result in a combination of the adapted LiSPSM with backside biasing and blurring to counteract perspective aliasing, false self-shadowing and projection aliasing.

The remainder of this chapter is structured as follows: Section 9.1 states the general problems our setup implicates. In Section 9.2 we describe the actual implementation of the algorithms. Finally, Section 9.5 concludes this chapter with a summary of the properties of our implementation.

9.1 The Problem

What are the problems of shadowing a large-scale, polygon-rich and dynamic environment? Let's take a look at each of the three properties of the environment and the demands that these properties place on any shadow algorithm:

A *dynamic* environment is a scene that may vary from frame to frame. For example some objects move or the position or direction of the light changes. This makes it necessary to regenerate the shadow each frame because of the possible changes in shadow casters or lighting conditions that affect the resulting shadow. This demands an algorithm that can calculate a new shadow each frame in a matter of milliseconds.

A *polygon-rich* environment, in our context and at this point of hardware speed, is a scene that contains roughly 100,000 or more visible triangles. This makes the use of classical geometry-based shadow algorithms, for example the classical shadow volume algorithm [Cro77], difficult. The shadow-volume algorithm needs a silhouette search and consumes fill-rate proportional to the number

of silhouette edges. Approaches to use simpler geometry for the shadow generation are possible, but may result in incorrect shadows. Additionally the generation of the simpler geometry is a non-trivial task. There simply is no robust and universal algorithm currently known that handles all configurations correctly.

A *large-scale* environment is a scene that contains near as well as far off objects, in arbitrary positions and sizes. In these environments, point, spot and directional lights are common. Especially directional lights, used to simulate the sun for instance, often cover large parts of the terrain. This setup means no additional problems for the shadow-volume approach because the geometrical nature of the algorithm makes it independent of the scales that occur in the scene. However we had to rule this approach out because of its performance deficit in polygon-rich scenes. The image space approaches, namely shadow mapping, have problems with such constellations. Solutions to these problems exist. We will use light space perspective shadow maps [WSP04] to overcome these problems.

9.2 Implementation of LiSPSM

In this section, we will give a recipe for the LiSPSM algorithm. The main idea of LiSPSM is to warp the shadow map by introducing a perspective transform into the shadow mapping process (see Chapter 8). In the transformed shadow map, near objects will have higher resolution, so that shadow quality is equally good for near and distant objects.

9.2.1 Algorithm outline

The main steps of the algorithm are to focus the shadow map on the relevant parts of the scene, and to find the perspective matrix and its parameters for LiSPSM (note that the focusing step is a good idea even for standard shadow maps) (see Chapter 4 for details):

1. Focus shadow map on the focus region
2. Find light space matrix L (which is a conventional shadow mapping matrix)
   (a) Set L to be the frame L_v^{-1} created from the view direction v and the light direction l, rooted at the eye point (directional lights) or the light position (point lights)
   (b) For point lights, append the point light projection L_p, so that $L = L_p L_v^{-1}$
   (c) Rotate L so that the view frustum points "upward" (in the shadow map), making $L = L_r L_p L_v^{-1}$

3. Find LiSPSM projection matrix P
4. Calculate n, the near plane distance of the LiSPSM projection
5. Find projection center C for P and generate its translation P_t
6. Find frustum planes for P, giving the LiSPSM matrix as $P = P_p P_t$
7. Apply shadow mapping using the joint matrix PL instead of just L

9.2.2 Focusing the shadow map

As already discussed at length in Section 4.2 focusing is an important step for any shadow mapping algorithm, to waste as little as possible of shadow map space to invisible parts of the scene. Since all involved bodies are simple convex polyhedra, it is easier to implement the required operations directly instead of using general purpose convex hull and polyhedra intersection algorithms. Equation 4.2 indicates the following focusing steps, here given in pseudo-code:

```
Object temp; // temporary intersection object

calcFrustum(temp, invEyeProjView);
clip(temp, sceneBV);

if (pointLight) {
  convexHullWithPoint(temp, lightPos);
}
else {
  convexHullWithDirection(temp, -lightDir, sceneBV);
}

clip(temp, sceneBV);

if (pointLight) {
  calcFrustum(lightFrustum, invLightProjView);
  clip(temp, lightFrustum);
}

return temp;
```

Some notable operations are:

- `calcFrustum()` creates a solid view frustum representation (e.g., a b-rep) in world space by feeding the 8 points of a centered 2-unit cube into the inverse of the combined view and projection matrix (PV) of the desired frustum.

- `clip()`: intersection of two convex objects. The first object is simply clipped by each plane of the second object and the created holes are filled with new polygons.

- The convex hull operation depends on whether the light is a point or directional light:

  - `convexHullWithPoint()` calculates the 3D convex hull of a convex body and a point. It removes all polygons of the body that are front-facing with respect to the point, leaving a hole in the form of a closed edge loop. The hole is closed by connecting the line loop to the point with a triangle fan.
  - `convexHullWithDirection()`: The directional case is best handled separately: We extrude the given convex body `temp` along the inverted light direction $-l$, knowing that we will clip to the scene bounding volume afterwards. This is done by moving the light-facing polygons of `temp` along $-l$ up to the given scene bounding volume. The resulting holes are then filled with quads.

9.2.3 Finding the light space matrix L

The light space matrix L transforms from world space into light space, the local coordinate frame of the light. This is a space spanned by the plane in which the conventional shadow map is defined, and its normal. For point lights, light space also includes the point light pro-jection. The LiSPSM projection P will be defined in light space. Therefore, light space needs to be rotated so that the view vector coincides with the projection direction of P.

Initial light space definition

The coordinate frame L_v is basically the transformation matrix used in conventional shadow mapping, but with the axes exchanged so that the LiSPSM projection is easier to define (more specifically, the z-axis of light space will also be the z-axis of the LiSPSM projection, which makes the mathematical formalism to find the optimal LiSPSM parameters more natural). It is created from the inverted light direction $-\mathbf{l}$ and the view direction $\mathbf{v}$ in the same way as a look-at matrix

(but with y and z exchanged), using $-\mathbf{l}$ as view direction and $\mathbf{v}$ as up-vector (see Figure 9.1):

$$\begin{aligned} L_{v_y} &= -\mathbf{l} \\ L_{v_x} &= \mathbf{v} \times L_{v_y} \\ L_{v_z} &= L_{v_x} \times L_{v_y} \end{aligned}$$

This choice makes the shadow projection plane parallel to the x/z-plane (as opposed to the x/y-plane as in conventional shadow mapping), so the light is actually looking down the y-axis. For the frame's origin, we use the eye position for directional lights and the light posi-tion for point lights. The required matrix is the inverse of the matrix L_v, which is constructed from the normalized versions of the above vectors.

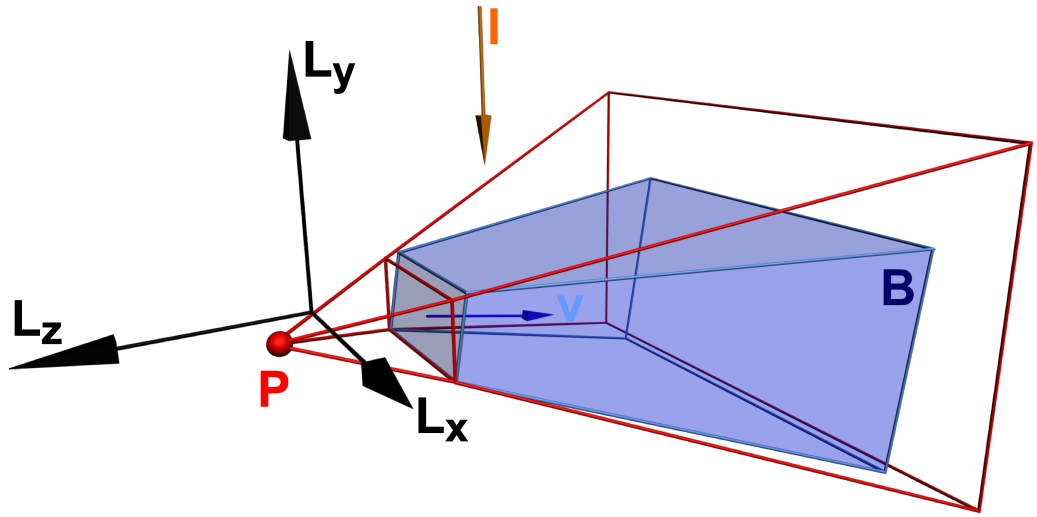

Fig. 9.1: Construction of the perspective frustum P in 3D

For directional lights, we set the light space matrix $L = L_v^{-1}$. For point lights, we set $L = L_p L_v^{-1}$, where L_p is the point light projection matrix associated with the light (a stan-dard projection matrix as defined for example by an OpenGL Frustum-call).

Light space rotation

One degree of freedom in determining light space is the rotation on the shadow map plane. The initial frame defined above rotates the shadow map so that the view vector, projected into the shadow map, points "upwards" in the shadow map before perspective projection. For point lights, however, the projection can cause

the frustum orientation to flip for views that are tilted with respect to the shadow map.

The aim is to rotate the shadow map (after projection!) so that the redistribution of shadow map samples occurs in the correct direction. We assume that the transformed view vector $\mathbf{v}_{\mathbf{ls}}$ (after the point light perspective projection L_p) indicates this "correct" direction. However, we can not simply transform the vector $\mathbf{v}$ by L_p, since parallel directions are not maintained by perspective projection. Instead we transform a ray corresponding to $\mathbf{v}$. A natural choice for the starting point of this ray would be the eye position. However, there is no guarantee that the eye position is in front of the shadow plane, or even anywhere near the body B. We therefore employ the following simple algorithm to find a "safe" starting point for the orientation ray:

1. Create the body $LVS = L \cap V \cap S$ (using the same clipping operations as before).

2. Transform LVS into eye space.

3. Find a vertex $\mathbf{e}_{\mathbf{eye}}$ of LVS nearest to the eye, i.e., with the maximum z-coordinate (note that LVS is by construction always in front of the eye). In order to find a more robust solution, we take the average of all vertices that are within a small distance to the vertex with the maximum z-coordinate.

4. Transform $\mathbf{e}_{\mathbf{eye}}$ back into world space $\mathbf{e}_{\mathbf{world}}$.

In most cases, $\mathbf{e}_{\mathbf{world}}$ will be a point on the near clipping plane of the view frustum. To define the orientation ray, we also find another point $\mathbf{b}_{\mathbf{world}}$ which is sufficiently far along $\mathbf{v}$ from $\mathbf{e}_{\mathbf{world}}$. The ray is then transformed to light space by transforming $\mathbf{e}_{\mathbf{world}}$ and $\mathbf{b}_{\mathbf{world}}$ using L, and homogenizing the points, giving $\mathbf{e}_{\mathbf{ls}}$ and $\mathbf{b}_{\mathbf{ls}}$.

The rotation L_r is now created with the light space vectors $(0, 1, 0)$ and $\mathbf{b}_{\mathbf{ls}} - \mathbf{e}_{\mathbf{ls}}$ as the up-vector in exactly the same way as in the previous section (look-at). Finally, we include this rotation into our light space matrix by setting $L = L_r L$.

9.2.4 Finding the LiSPSM projection matrix P

The LiSPSM projection matrix P creates a perspective transformation in light space such that the projection plane of P is perpendicular to the shadow map plane. As discussed in Chapter 8, this ensures that light rays are mapped to light rays so that light direction and type remains unchanged. In order to completely define P, we need to find a projection center C and the frustum planes.

Finding the LiSPSM projection center C

We transform the focus body B into light space using L and calculate its bounding box in light space. The near and far planes of the LiSPSM projection are then defined to be at $z = B_{z_{near}}$ and $z = B_{z_{far}}$ (we work along the $-z$-axis, so $B_{z_{near}} = B_{z_{max}}$ and $B_{z_{far}} = B_{z_{min}}$) respectively, for z-coordinates in light space. To find C, we proceed in two steps. First select an appropriate point C_{start} on the near plane of the LiSPSM projection, then move distance n back from the near plane. The distance n of C to the near plane is the main parameter of the LiSPSM method, and will be discussed in a following section; here, we assume it is given. C_{start} is defined as $(\mathbf{e_{ls_x}}, \mathbf{e_{ls_y}}, B_{z_{near}})^T$, i.e., the projection of $\mathbf{e_{ls}}$ (see Section 9.2.3) onto the near plane. $\mathbf{e_{ls}}$ is a good starting point, as it is the best approximation to the original viewpoint we have available in light space (if C_{start} is close to the original viewpoint, this will help self shadowing problems later on) (see Figure 9.2)

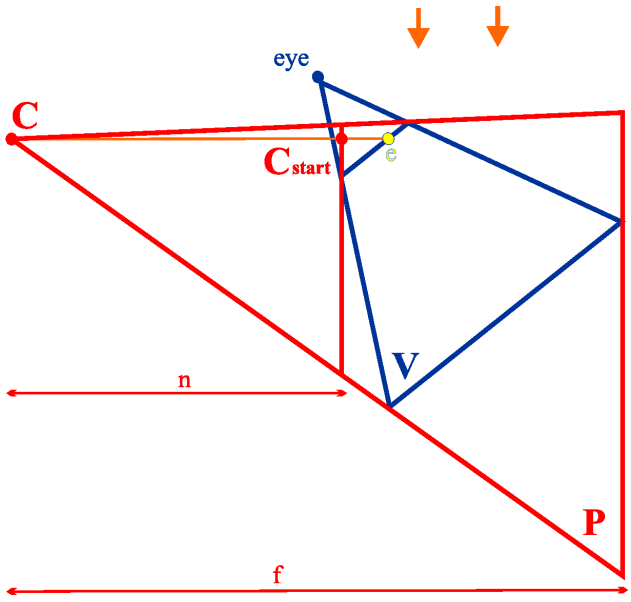

Fig. 9.2: Construction of C_{start} from a point $\mathbf{e_{ls}}$ on the camera near plane in light space (for simplicity, V is shown instead of B.)

Knowing C, P can be initialized with a translation P_t that centers light space at C.

LiSPSM frustum planes

To find the frustum planes of P, we project B onto the LiSPSM projection near plane and find its extents. This can be done using a temporary projection matrix P_{temp} with the frustum parameters $[l, r, b, t] = [-1, +1, -1, +1]$, while from the

previous section it already follows that the near plane distance should be set to n, and the far plane distance to $n + |B_{z_{far}} - B_{z_{near}}|$. The bounding rectangle of the projected vertices of B gives the final frustum parameters l, r, b, t, so that P can be set up as $P_p P_t$. In a last step, P is also multiplied with the viewport matrix that is usually used to fit the result of the projection, which lies between -1 and 1 in all coordinates, into the range of $[0, 1]$ for texture mapping.

9.2.5 Setting the parameter n

The parameter n encodes the near plane distance of the LiSPSM projection and thus controls the strength of the perspective warping effect. As discussed in Section 8.2, a smaller n gives a stronger warping effect, causing near shadows to gain quality and far shadows to lose quality. In Section 8.2.3, we introduced Equation 8.11. In this equation, we could use $d = |B_{z_{far}} - B_{z_{near}}|$ for our implementation, because the focus body B z-extents in light-space are a more exact approximation of the affected z-range. However, with the the more general Equation 8.13 we can handle point lights too.

The relevant z-values

The values z_0 and z_1 can be said to represent the eye-space z-range for which the LiSPSM projection is optimized. Mathematically, Equation 8.13 is derived by choosing two points at the near and far plane of the LiSPSM frustum P and choosing n such that the perspective error is identical for these two points. We choose the points on a line perpendicular to the near plane of P (and therefore to the light direction). As shown in Figure 8.9, the idea is to pick z_0 at the intersection of the camera and the LiSPSM frustum near planes. If the camera near plane does not intersect B, we use the next possible parallel plane that does (as shown in a previous section, this plane goes through $\mathbf{e_{world}}$).

In order to find the intersection point $\mathbf{z_{0_{ls}}}$ in light space, we transform the plane equation of the chosen plane into light space by multiplying it with the inverse transpose of L. Then we intersect the transformed plane with the planes $z = B_{z_{near}}$ and $x = e_{ls_x}$ to give $\mathbf{z_{0_{ls}}}$. The second point, at the LiSPSM far plane, is $\mathbf{z_{1_{ls}}} = [z_{0_{ls_x}}, z_{0_{ls_y}}, B_{z_{near}}]$. The values z_0 and z_1 in eye space are calculated by transforming $\mathbf{z_{0_{ls}}}$ and $\mathbf{z_{1_{ls}}}$ using L^{-1} and the camera view matrix.

Note that due to construction, z_0 will usually be fixed at z_n if the camera near plane is at least partly visible in the light frustum. For directional lights, z_1 will give the same result (but signed) as in the original formula. For point lights, z_1 varies due to the perspective projection of the light. Also, both values can vary when B differs significantly from V.

Considerations for z_n and z_f

For many situations, z_0 and z_1 will – through the construction given in the previous section – largely depend on the near and far camera frustum planes z_n and z_f, so we take a look at how these values influence shadow quality. Note first that the perspective aliasing error has two maxima at the (LiSPSM) near and far plane, and a minimum near the viewer (see Figure 9.3 for $z_f = 100$). When using the formula given above, the maxima at the near and far plane are equalized. The value of these maxima can be shown to depend on the ratio z_1/z_0. This means that larger frusta (smaller z_n and/or larger z_f) lead to larger errors. However, this is not necessarily visible, since the location of the minimum can accidentally shift to a location where shadow quality is more apparent.

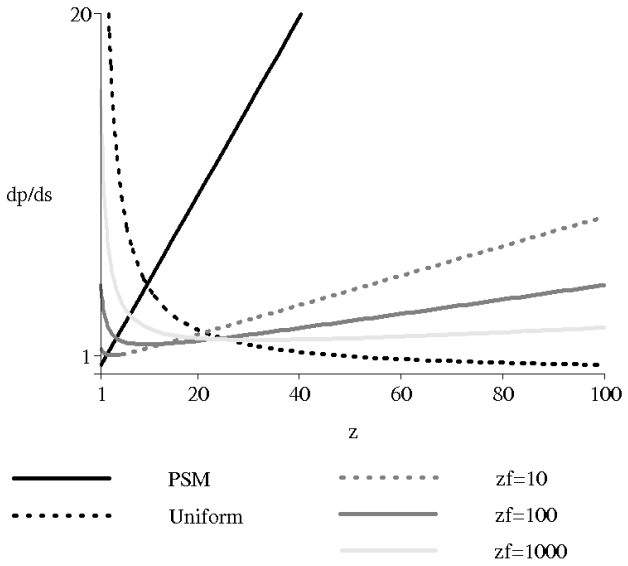

Fig. 9.3: Error distribution from near plane to $z = 100$ for LiSPSM (different values of z_f), uniform shadow maps ($z_f = 100$) and PSM ($z_f = 100$).

Let us then analyze the simple case that the view direction is perpendicular to the light direction, with a directional light source and a view frustum contained in the scene bounding box. The parameter n then evaluates to

$$n_{opt} = z_n + \sqrt{z_n z_f} \tag{9.1}$$

Enlarging the frustum (bigger z_f) will lead to a smaller warping effect (larger n), because more accuracy needs to go into the distant regions, which can only be achieved by making the mapping more like a uniform shadow map. This can be undesirable for two reasons:

- The total maximum error increases for larger frusta.
- The location of minimum error will move away from the viewer, so that the apparent shadow quality can be even lower even though the maximum error is not so strong (see Figure 9.3 for error plots with different z_f).

It is therefore advisable to choose z_f as small as possible. Sometimes, however, the far plane is determined through other application concerns. An interesting option is therefore to construct a "virtual" view frustum V_v for the calculation of n. In this virtual frustum, z_f can be used as a tuning parameter by which the user can determine up to which point in the actual view frustum shadows should have acceptable quality. V_v is applied by basing the calculations in section 9.2.5 (i.e., the values d, z_0 and z_1) not on the intersection body B, but on an intersection body calculated using V_v.

It is important to note that this new far plane distance is not usable for the intersection body calculation used for focusing because it is possible that invisible objects inside the view frustum cast a shadow inside the visible part of the view frustum (see [O'R04] for details). The shadows of these objects would not be generated if we used this new far plane distance for all calculations. We only propose to use the new far plane distance for the calculation of n_{opt}. With this optimization, we give these invisible shadow caster objects with visible shadows less space/resolution in the shadow map.

If occlusion culling is used before the main render pass (for example in a preprocess or in a depth-only pass [BWPP04]), this can be used to adjust z_f automatically: just set z_f to the farthest vertex of the farthest encountered object. Note, however, that this may introduce shadow continuity problems if objects become visible or disappear, because the area of minimum error suddenly moves.

Another effect is that sometimes a smaller value z_n can lead to better apparent shadow quality. A smaller z_n causes a stronger warping effect (smaller n), because more accuracy is needed at the near plane, which in turn moves the area of minimum error nearer to the viewer. The main reason for the apparent better quality is that this area of minimum error is more conspicuous to the viewer than distant shadows, which will deteriorate in the process. If the reduction of shadow quality in distant regions is not visible, then z_f was chosen to be too large in the first place.

9.2.6 A side-note for directional lights: the body vector

Until now we assumed that the view direction is the most logical choice for the up-direction of our light-space for directional lights. This choice influences the validity of our choice of z_0 and z_1, and with this the quality of the shadows generated by our formula.

The problem is that we want to focus on the intersection body B, and this body can greatly differ from the original view frustum. We propose to use the direction from the eye position to the other end of the intersection body B that is in the middle of the volume of the body. We call this vector body vector. It is clear that in certain cases this vector differs from the view direction we used till now.

A very fast method of calculating the body vector is to treat B as a point cloud and sum up all vectors emanating from the eye position to each point. This method automatically gives more weight to points that are far away from the eye position (intersection points with the far plane), which is good because generally these points influence the volume of the body much more than the points near the view frustum near plane. Wrong results are possible if the intersection body B had had parts near the far plane intersection results with much more intersections than on other parts of the far plane intersection results. In this case, simple adding up gives wrong results. In practice we found no robustness problems with this approach that would call for the usage of a more complex method.

9.3 Correct biasing

We have noticed that with LiSPSM, simple slope-scale biasing works fairly well. This is probably due to the choice of the projection center C, which is as close as possible to the original eye direction, and due to the fact that the required perspective warp is often not very strong. The best results, however, were obtained using a technique called backside mapping (as discussed in Section 5.1). This simple trick reverses backface culling for the shadow map rendering pass, thus only rendering the back sides of objects into the shadow map [WM94]. This can only be used for closed objects, of course.

9.4 Projection aliasing

9.4.1 Lighting equation effects

As already discussed in Section 6.2, the Phong lighting model actually helps us to hide some of the projection aliasing artefacts. The only thing to avoid is a strong per-light ambient term, which can lead to strong illumination regardless of the incoming angle. Since the ambient term is meant to simulate global interreflections, a hard shadow due to an ambient term would be unrealistic anyway.

9.4.2 Blurring

Should projection aliasing still be objectionable, the effect can be hidden using a simple image-space blur effect (see Section 6.3). Instead of rendering the shadow in the main rendering pass, we start with an initial pass where, in addition to the depth buffer, we write the shadow attenuation term (subject to percentage-closer filtering) to the color channel. The shadow term is then copied into a texture using a shader that executes a simple blur. It is advisable to decrease the blur filter kernel with increasing depth value, so that small shadows in the distance don't get overly blurred. For multiple light sources, this has to be carried out for each light separately, either using multiple passes or the multiple render target functionality in newer cards. During the main rendering pass, the blurred shadow texture(s) is used to attenuate the diffuse color. No shadow calculations are necessary, and depth writes can be disabled.

9.5 Summary

Shadow mapping is a very popular shadow algorithm especially for computer games. While traditional shadow maps have been prone to aliasing errors, the techniques used in this chapter helps to improve the quality of shadow mapping in most common situations without increasing the cost of the basic algorithm. Light Space Perspective Shadow Maps redistribute shadow quality so that near objects receive more resolution, but also maintain sufficient resolution for far objects. We have shown how to deal robustly with perspective aliasing and biasing, and proposed a method to deal with the problem of projective aliasing. LiSPSM works for both directional lights and point lights, but has a greater effect for lights that are directional or close to directional. This makes LiSPSM an ideal choice for outdoor lighting of huge scenes.

Chapter 10

Summary

This thesis has presented an introduction to the field of shadow mapping. We gave a thorough description of the underlying theory, as well as a detailed report of the problems involved in shadow mapping. The most important ones are perspective aliasing and projection aliasing, which are caused by undersampling, and incorrect self-shadowing, caused by resampling and depth quantization. Later on, we attacked each of these problems and provided theoretically sound as well as practically usable solutions. For incorrect self-shadowing, we showed that biasing is a valuable tool to correct this artifact. For projection aliasing, we gave hints how to hide it with the help of the Phong lighting model and blurring. But the main part of this thesis was devoted to the removal of perspective aliasing artifacts. We listed the main methods for getting rid of this problem and showed their shortcomings. Thence we embarked on a minute explanations of the reasons why and how we can overcome most of these problems with the aid of Light space perspective shadow maps.

Furthermore we provided the recipe of a real-world implementation that actually shows the usability of our theoretical findings in practice.

All in all we hope to give you with this thesis an immediate starter kit to shadow maps and a theoretical basis too, for the ones who want to improve on the shown techniques.

These final pictures (see Figure 10.1) where taken from the implementation described in the last chapter. In this demo we have implemented the described methods and improvements based on the LiSPSM algorithm, and used it for a scene lit by one directional light source. The scene is an outdoor environment that contains 5.000 tree-like objects and uses Coherent Hierarchical Culling (CHC) [BWPP04] for visibility determination.

We implemented 2x2 percentage closer filtering with the OpenGL Shading Language because not all hardware vendors supply us with automatically applied PCF for shadow maps. Fogging was disabled. The platform was a Pentium4 2.4GHz with 1GB RAM and an ATI Radeon 9600 with 265MB RAM. The pictures shown in Figure 10.1 where captured using a 512x512 pixel viewport res-

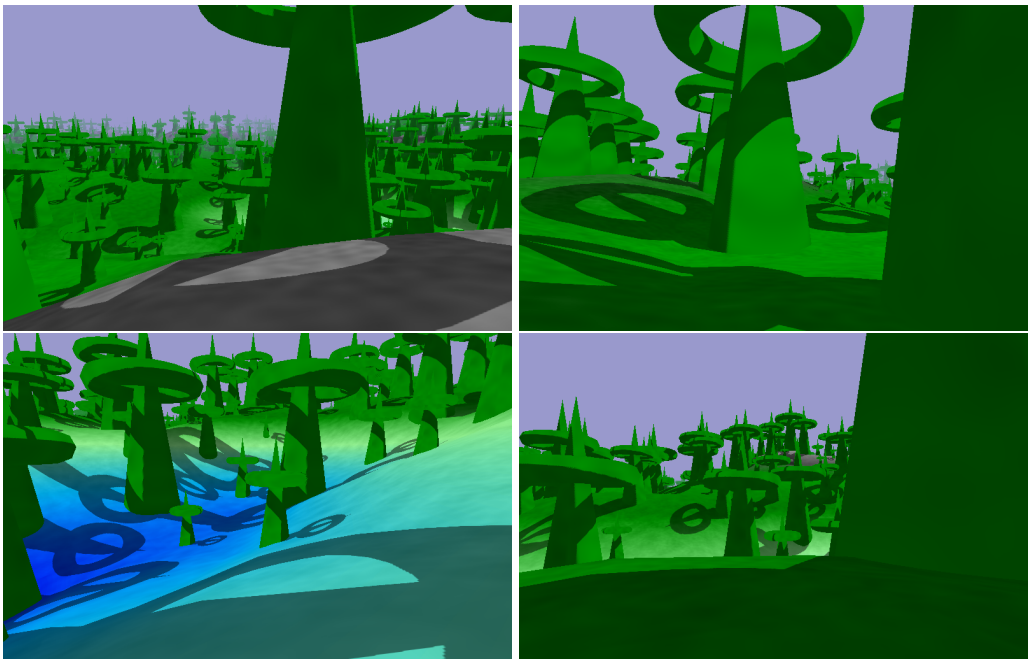

Fig. 10.1: Various views of our test scene viewport 800x512 pixel, shadow map 2048x2048 pixel, 2x2 PCF enabled

olution and a 2048x2048 pixel shadow map resolution. The field of view of the view frustum was $60°$, near plane distance was 0.1 and the far plane distance was 70.

Appendix A

The perspective space

In this appendix we will introduce the basics of perspective projections in Section A.1 and of perspective transformations in Section A.2. After this introduction we show how a correct perspective transformation is constructed and explain the OpenGL perspective transformation matrix in its homogeneous form in Section A.3.

A.1 The perspective projection

A projection is an operation that "loses" one dimension. So for instance if we want to display a three-dimensional scene on a dwo-dimensional screen, we do a projection. Two types of projection are very common:

- Orthogonal projections
- Perspective projections

An orthogonal projection is done by running a line through each point we want to project. All of this lines are parallel to each other, and we intersect these lines with the viewing plane. The resulting intersection points are the two-dimensional orthogonal projections of the original three-dimensional points.

A perspective projection is constructed the following way: First we identify all points with a line through a "center of projection". Then we intersect these lines with the viewing plane. The intersection points are the two-dimensional perspective projections of the original three-dimensional points.

A.2 Towards the perspective transformation

We have already underlined that projections remove one dimension. For computer graphics, where we usually have a two-dimensional output medium and three-dimensional input data, all seems to work fine at first glance. On the second glance, problems appear. We want to determine visibility. For instance, we

want to know if a polygon is in front of another polygon. We further want that three-dimensional lines map to three-dimensional lines, which is useful for hidden surface removal.

Therefore we need a transformation that keeps the third coordinate, the perspective transformation. The difference between a perspective projection and a perspective transformation in our context is that the perspective projection maps from 3D to 2D and the perspective transformation from 3D to 3D. This means that a perspective transformation transforms a 3D space into another 3D space, called post-perspective space. With this property in mind, it is clearly no problem to repeatedly use perspective transformations to construct new spaces. But first we must find such a perspective transformation. The simple mapping

$$(x, y, z) \rightarrow (\frac{xn}{z}, \frac{yn}{z}, n)$$

maps 3D lines to 3D lines, but all depth information is lost.

$$(x, y, z) \rightarrow (\frac{xn}{z}, \frac{yn}{z}, z)$$

maps endpoints of a line in a way that the same relative depths are conserved after mapping, but it fails to map lines to lines. In Section A.3, we show a representation of perspective transformations in standard 4 by 4 matrices to use the hardware acceleration provided by common consumer hardware with a fixed rendering pipeline.

A.3 The perspective transformation and its matrix

We start with an instructive version of a perspective projection on the plane $z = -d$. Our projection center is the origin. If we want to project a point p onto this plane at the point q, we first take a look at how we can project its x-coordinate p_x. The idea to the projection is found by applying similar triangle math.

In Figure A.1, we see that p_x and p_z are similar to q_x and $-d$, which gives us the resulting q_x as

$$\frac{p_x}{p_z} = \frac{q_x}{-d} \Leftrightarrow q_x = -d\frac{p_x}{p_z}. \tag{A.1}$$

The y-coordinate of q is found in the same way as $q_y = -dp_y/p_z$ and the z-coordinate is already given by $q_z = -d$.

To find the perspective *transformation* we use a similar method. For a perspective transformation it is suitable to specify a near and a far plane between which we want our input z-values to lie. These planes are given by their signed distance to the origin, n for the near plane and f for the far plane. The mapping of x and

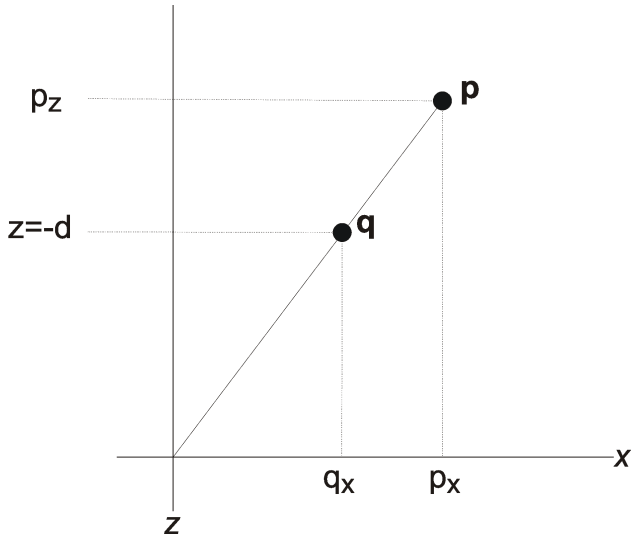

Fig. A.1: Projection works with similar triangle math.

y coordinates should be the same mapping the perspective projection provides. With Equation A.1 this leads to

$$q_x = n\frac{p_x}{p_z} \qquad q_y = n\frac{p_y}{p_z}.$$

This input z-range should be mapped by the transformation into the range $[-1, 1]$.

$$[n, f] \rightarrow [-1, 1] \tag{A.2}$$

Since we also want to map the z-coordinates perspectively, we use the same formula as devised for the mapping of x and y coordinates, but in a general form. Where we used just np_x resp. np_y, we now use $ap_z + b$, resulting in

$$q_z = \frac{ap_z + b}{p_z} = a + \frac{b}{p_z}.$$

To solve this equation we recall Equation A.2 and we can therefore state

$$-1 = a + \frac{b}{n} \qquad \wedge \qquad 1 = a + \frac{b}{f}.$$

After a derivation we get

$$b = \frac{2nf}{n - f} \qquad \wedge \qquad a = \frac{n + f}{f - n}.$$

To summarize, q is

$$q = \begin{pmatrix} n\frac{p_x}{p_z} \\ n\frac{p_y}{p_z}. \\ \frac{n+f}{f-n} + \frac{2nf}{(n-f)p_z} \end{pmatrix}$$

The 4x4 matrix in homogenous form looks like this

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & \frac{2nf}{n-f} \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Common graphics libraries, like OpenGL for instance, use a changed coordinate frame after perspective transformation. Before the perspective transformation, OpenGL uses a right handed coordinate system with the z-axis pointing towards the viewer, and afterwards OpenlGL uses a left handed coordinate system with the z-axis pointing away from the viewer. This can be introduced into our matrix by a scale matrix that inverts the z-coordinate

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & \frac{2nf}{n-f} \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

To imitate a real camera, OpenGL adds a field of view in the y-direction fov_y and an aspect ratio $aspect$ for the field of view of the x-direction. This leads to the perspective transformation matrix generated by the $gluPerspective$ OpenGL call

$$\begin{pmatrix} \frac{cotan(\frac{fov_y}{2})}{aspect} & 0 & 0 & 0 \\ 0 & cotan(\frac{fov_y}{2}) & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

where $cotan$ denotes the cotangent.

List of Figures

List of Tables

Bibliography

[AL04] Timo Aila and Samuli Laine. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 161–166. Eurographics Association, 2004.

[Ald04] Graham Aldridge. Generalized trapezoidal shadow mapping for infinite directional lighting. *http://legion.gibbering.net/projectx/paper/ shadow%20mapping/*, October 2004.

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

[BWPP04] Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *Computer Graphics Forum*, 23(3):615–624, sep 2004. Proceedings EUROGRAPHICS 2004.

[CG04] H. Chong and S. J. Gortler. A lixel for every pixel. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.

[Cho03] Hamilton Chong. Real-time perspective optimal shadow maps. Senior thesis, Harvard College, Cambridge, Massachusetts, April 2003.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In James George, editor, *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, volume 11, pages 242–248. ACM Press, July 1977.

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In Eugene Fiume, editor, *SIGGRAPH 2001 Conference Proceedings*, Annual Conference Series, pages 387–390. ACM SIGGRAPH, Addison Wesley, August 2001.

[Hec86] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*. Eurographics, Eurographics, 2003. State-of-the-Art Report.

[JMB04] Gregory S. Johnson, William R. Mark, and Christopher A. Burns. The irregular z-buffer and its application to shadow mapping. Research paper, The University of Texas at Austin, 2004.

[Koz04] S. Kozlov. Perspective shadow maps - care and feeding. *GPU Gems*, pages 217–244, 2004.

[MH02] Tomas Möller and Eric Haines. *Real-Time Rendering, Second Edition*. A. K. Peters Limited, 2002.

[MT04] Tobias Martin and Tiow-Seng Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, pages 153–160, 2004.

[O'R04] J. O'Rorke. Managing visibility for per-pixel lighting. *GPU Gems*, pages 245–257, 2004.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):283–291, July 1987.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Siggraph 2002 Conference Proceedings*, volume 21, 3, pages 557–562, July 2002.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.

[WE03] D. Weiskopf and T. Ertl. Shadow Mapping Based on Dual Depth Layers. In *Procceedings of Eurographics '03 Short Papers*, pages 53–60, 2003.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):270–274, Aug. 1978.

[WM94] Yulan Wang and Steven Molnar. Second-depth shadow mapping. Technical report, University of North Carolina at Chapel Hill, 1994.

[Woo92] Andrew Woo. The shadow depth map revisited. *Graphics Gems III*, pages 338–342, 1992.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In *Proceedings of Eurographics Symposium on Rendering 2004*, 2004.