FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Robust Hard Shadows

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik/Digitale Bildverarbeitung

eingereicht von

Martin Stingl
Matrikelnummer 0226290

an der
Fakultät für Informatik der Technischen Universität Wien

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

Wien, 19.09.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Martin Stingl
Neudorf 15, 3335 Weyer

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

(Ort, Datum)	(Unterschrift Verfasser)

Abstract

The generation of shadows in large virtual environments for real-time rendering applications like e.g. video games is still a great challenge for computer graphics. In the past few years shadow mapping and its variants have become widely accepted as appropriate methods for shadow creation, which resulted in large number of advanced shadow mapping techniques that have been introduced recently. For this reason, it sometimes can be difficult for developers to choose a suitable method based on the certain given scenarios.

This thesis focuses on an analysis of some common fully hardware-accelerated shadow mapping techniques and their capabilities to combine them with each other. We present an interactive framework that allows the user to experiment with the chosen methods and to visualize almost every interesting aspect of the shadow creation process for arbitrary scenes. It offers the opportunity to generate the shadows by using multiple shadow maps, sample redistribution or both and provides most possible flexibility in terms of various adjustable parameters. Furthermore, it includes a feature to analyze the final results from different points of view, which should help developers to find the best suited algorithm for the given scene.

Kurzfassung

Die Generierung von Schatten in großen virtuellen Umgebungen für Echtzeitrendering Anwendungen wie z.B. Videospielen ist nach wie eine große Herausforderung für die Computergraphik. In den letzten Jahren haben sich Shadowmapping und verschiedene Varianten davon in großem Umfang als geeignete Methode zur Schattenerzeugung durchgesetzt, was in letzter Zeit zur Einführung sehr vieler erweiterter Shadowmapping Techniken geführt hat. Aus diesem Grund ist es manchmal schwierig für Entwickler die passende Methode für bestimmte gegebene Szenarios auszuwählen.

Diese Diplomarbeit konzentriert sich auf die Analyse von einigen gebräuchlichen komplett hardware-beschleunigten Shadowmapping Techniken und die Möglichkeit sie zu kombinieren. Wir präsentieren ein interaktives Framework, dass es dem Anwender erlaubt mit den gewählten Methoden zu experimentieren und fast alle interessanten Aspekte des Schattenerzeugungsprozesses für beliebige Szenen zu visualisieren. Es unterstützt die Möglichkeit der Schattengenerierung durch die Verwendung von mehreren Shadowmaps, durch Umverteilung der Samples oder beides gemeinsam, und bietet größt mögliche Flexibilität durch zahlreiche veränderbare Parameter. Außerdem enthält es die Möglichkeit die finalen Resultate aus unterschiedlichen Blickpunkten zu analysieren, was es Entwicklern ermöglichen sollte für die gegebenen Szene den am besten geeigneten Algorithmus zu finden.

Contents

1 Introduction **1**
- 1.1 Goal of this thesis . . . 1
- 1.2 Contributions . . . 2
- 1.3 Structure of this thesis . . . 3

2 Related work **5**
- 2.1 Basic shadow mapping algorithm . . . 5
- 2.2 Shadow mapping errors . . . 6
  - 2.2.1 Self-shadowing artifacts . . . 6
  - 2.2.2 Projection aliasing . . . 7
  - 2.2.3 Perspective aliasing . . . 9
- 2.3 Error analysis . . . 10
  - 2.3.1 Simple error analysis . . . 10
  - 2.3.2 Accurate error analysis . . . 11
  - 2.3.3 Error analysis for both shadow map directions . . . 13
- 2.4 Soft Shadows . . . 14

3 Advanced shadow mapping **17**
- 3.1 Focused shadow mapping . . . 18
  - 3.1.1 Calculation of convex intersection body . . . 18
  - 3.1.2 Focusing the shadow map . . . 20
  - 3.1.3 Error analysis . . . 21
- 3.2 Warping . . . 22
  - 3.2.1 Light Space Perspective Shadow Maps . . . 22
  - 3.2.2 Re-parametrized Light Space Perspective Shadow Maps . . . 25
  - 3.2.3 Error analysis . . . 26
- 3.3 Z-partitioning . . . 26
  - 3.3.1 Adjusting the view frustum . . . 28
  - 3.3.2 Split scheme . . . 29
  - 3.3.3 Split selection . . . 31
  - 3.3.4 Error analysis . . . 32
  - 3.3.5 Storage strategy . . . 32
  - 3.3.6 Minimizing the number of rendering passes . . . 35

3.4 Combinations . . . 36
3.4.1 Error analysis . . . 37
3.5 Reducing shadow flickering . . . 38
3.6 Filtering . . . 41
3.6.1 Kernel size . . . 42
3.7 Summary . . . 44

4 Optimizations **45**
4.1 Projection center for warping . . . 45
4.2 Pseudo-near plane and pseudo-far plane . . . 46
4.3 Maximizing the near plane distance . . . 47

5 Framework **51**
5.1 Concepts . . . 51
5.2 Visualization view . . . 53
5.3 Intersection body . . . 54
5.4 Texel borders . . . 54
5.5 Split selection . . . 55
5.6 Light views . . . 57
5.7 Sampling frequency . . . 58
5.8 Perspective aliasing error . . . 59

6 Implementation **61**
6.1 Implementation overview . . . 61
6.2 Code structure and reuse . . . 62

7 Results **65**
7.1 Sights of Paris . . . 65
7.2 Winter scenery . . . 65
7.3 Benchmarking results . . . 66
7.4 Comparison of pseudo-near plane and adjusted near plane . . . 68
7.5 Visualizing the warping effect and the shadow map area . . . 69

8 Conclusion **75**
8.1 Summary . . . 75
8.2 Future work . . . 76
8.2.1 Future work to shadow mapping . . . 76
8.2.2 Future work to the presented framework . . . 77

A Implementation details **79**
A.1 Geometry cloning . . . 79
A.2 Mip map chain to retrieve minimal depth . . . 80
A.3 Sampling rate . . . 81
A.4 Texel borders . . . 82

B Framework details **85**
B.1 Main configuration and scene loading 85
B.2 Main structure 86

List of Figures **88**

Bibliography **91**

CHAPTER 1

Introduction

Shadows represent a great enhancement of the visual perception in virtual environments. They provide important information about the geometric relations between the objects in the scene like the position, size or shape of the shadow casters and the shadow receivers. Especially for real-time applications like for example video games, the generation of highly detailed shadows in polygon-rich dynamic large-scaled virtual environments at appropriate frame rates is still one of the most demanding challenges for computer graphics.

In the past few years *shadow mapping*, introduced by Williams [Wil78], has become the number one choice of developers to generate shadows in their applications. Moreover, with Segal's et al. [SKvW+92] hardware adaptation of shadow mapping and the recent improvements of the graphics hardware, many enhancements and variants of the basic shadow mapping algorithm have been introduced in the last few years.

Since different shadow mapping methods are designed for different scenarios, in most cases it is very difficult for developers to choose an appropriate technique, which is why modern game engines like CryENGINE®3 (see Figure 1.1) or Frostbite™2 usually offer several different shadow mapping techniques that are chosen and applied based upon the current used scenario.

In the following sections of this chapter we will declare the goals of this thesis, give a short overview of our contributions and provide an outline of the structure of the following chapters.

1.1 Goal of this thesis

Since Williams [Wil78] introduced the shadow map algorithm, its use has gone a long way. Many methods and ways have been introduced to improve its performance and quality, respectively minimize the occurring errors. Because of this large number of methods, as mentioned before, in most cases it is very difficult to choose the right one, based on the present form of

Figure 1.1: High quality real-time shadows generated with CryENGINE®3. Image courtesy of Crytek GmbH.

applications and scenarios.

The goal of this thesis is to analyze some commonly used advanced shadow mapping techniques and compare them with each other considering the performance and quality improvements. Additionally we will combine specific methods and determine the enhancement respectively degradation of their characteristics compared to an individual application or to other combinations. We intend to investigate the results for various configurations and different scenarios, using several visualization tools, to offer the opportunity to find an appropriate robust shadow mapping technique for each possible example scene.

The focus of the chosen techniques, which we plan to analyze and use for our experiments, lies on fully hardware-accelerated real-time capability, which means an optimized application should at least generate 60 frames per second.

1.2 Contributions

During the work on this thesis we developed a shadow mapping framework that allows to experiment with the chosen shadow mapping algorithms and apply them to arbitrary scenes. The major contribution of this framework compared to other existing shadow mapping systems is given by the provided flexibility of our application and the possibilities to analyze and visualize

various aspects of the shadow mapping process.

In addition to the feature to choose between several state of the art shadow mapping techniques, our framework allows the user to adjust almost any of the important parameters and experiment with different reasonable combinations of the supported shadow mapping methods. Besides the changes of the shadow quality itself, the immediate results of these experiments can be examined interactively at any time and from different points of view through the various offered visualization opportunities.

Furthermore, we offer the possibility to compare the final shadow mapping results based on an analytical evaluation of the occurring errors, which are computed for every frame during the run-time.

During our experiments and analysis, we found some problems related to some certain techniques that partly involved robustness issues and sometimes caused a degradation of the shadow quality. In Chapter 4 we describe our found optimizations for the mentioned techniques and present our solutions to the problems. Furthermore, we experiment with some new approaches to further improve the shadow quality and robustness of some specific methods.

1.3 Structure of this thesis

This thesis is structured into different chapters as follows:

1. Chapter 2 reviews the basic shadow mapping algorithm, gives an overview of the main problems of shadow mapping and the existing techniques to resolve them, and discusses some common methods to analyze the errors.

2. Chapter 3 discusses various well known advanced techniques to improve the shadow quality and reduce the shadow mapping errors.

3. Chapter 4 introduces some new approaches to improve the quality and robustness of several techniques.

4. Chapter 5 explains the features of our implemented framework and discusses the different implemented visualization possibilities.

5. Chapter 6 deals with some implementation details of our framework and describes the overall structure of our system.

6. Chapter 7 presents several results of our experiments and shows the application of our visualization tools.

7. Chapter 8 concludes this thesis and discusses some possible additional and future improvements.

CHAPTER 2

Related work

In this chapter we will first recall the basics of the shadow map algorithm and give a brief review on how it is done on today's hardware. Then we will review the major problems of shadow mapping and the resulting errors, and give a brief overview of most of the techniques that have been introduced so far to improve the shadow quality and to reduce the shadow mapping errors.

2.1 Basic shadow mapping algorithm

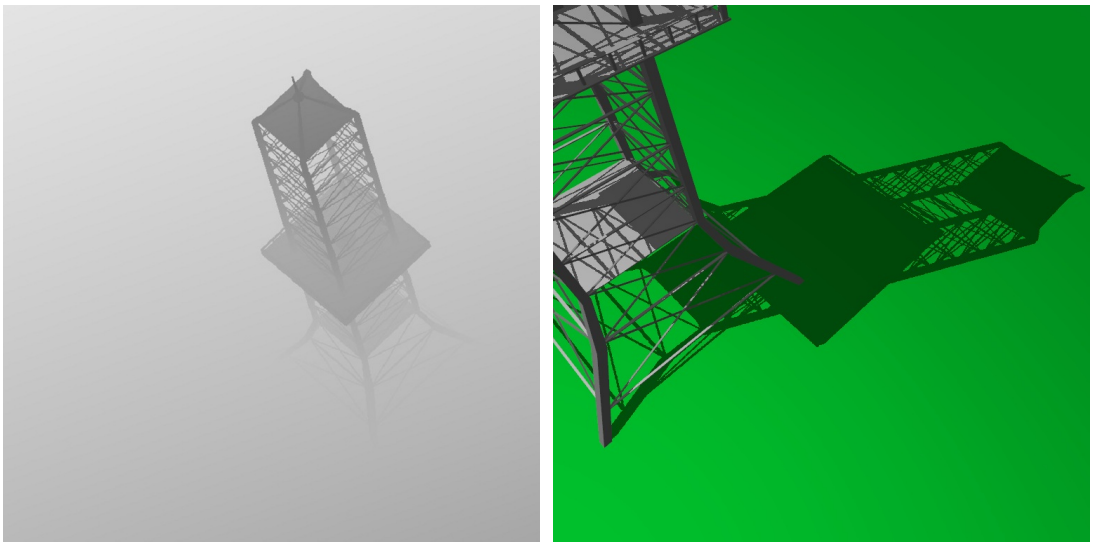
To generate shadows in virtual environments Williams introduced in [Wil78] *shadow mapping*. He was the first who formulated the basic well-known shadow mapping algorithm with the following two steps:

1. render the scene from the light's point of view only computing the depth values using a z-buffer and storing them into a depth map (*shadow map*),

2. while rendering the scene from the eye's point of view, perform a linear transformation that maps each point into the light source space and applies a visibility test against the stored values in the shadow map (*shadow test*) to determine shadowed and unshadowed points.

By using projective texture mapping and three rendering passes Segal et al. [SKvW$^+$92] showed how the generation of the shadow map and computation of the shadows in the final rendering can be easily matched to hardware.

On modern programmable GPUs, shadow mapping, using shadow maps up to 8,192 x 8,192 pixels, can be easily performed fully hardware accelerated in two rendering passes by using William's original formulation of the algorithm and the projective texture mapping approach from Segal et al. An example configuration of a shadowed scene with a visualization of the corresponding shadow map can be seen in Figure 2.1.



(a) Depth map from the light's point of view (b) Shadowed image from the eye's point of view

Figure 2.1: Example visualization of the light's view depth map (shadow map) (a). Corresponding shadowed eye view (b).

2.2 Shadow mapping errors

As Scherzer discussed in [Sch05], since shadow mapping is an image space technique using finite resolution shadow maps, it suffers from three major problems that are caused by sampling and resampling issues into the z-buffer.

2.2.1 Self-shadowing artifacts

Referring to Section 2.1 we need to transform the regularly spaced pixels from eye-space to light-space to perform the shadow test. After this transformation, the transformed pixels may fall between the regular sample values stored in the shadow map. Combined with the finite precision of the depth values the result of the shadow test for a fully illuminated pixel can be wrong. These resampling inaccuracies can lead to moiré patterns (see Figure 2.2).

According to Scherzer [Sch05] there are numerous workarounds to minimize the occurrence of these patterns. A very simple one is to add a small user defined constant bias to the shadow map's depth values. Unfortunately this method can lead to incorrect shadows because of their movement in light-space z-direction. It also leads to problems for polygons with different depth slopes, because the more the depth slope of the polygons is increasing the more bias is needed. For this reason, instead of using a constant bias, a slope-scaled bias is a better way to remove the moiré patterns. Although this slope-scaled biasing method works fine in most configurations to remove self-shadowing artifacts, there might remain problems with shadow mapping techniques that involve a non-linear distribution of z-values.

Another solution to remove the moiré patterns is to only render the back sides of the scene geometry into the shadow map, which is actually another form of biasing. Though this method only works with closed objects, which can be problematic at rendering of foliage, it is more robust than common biasing methods and removes almost any self-shadowing artifacts.

To summarize, according to the extensive tests of Scherzer [Sch05] the best results to resolve the problem of the self-shadowing artifacts in most configurations are provided by a combination of slope-scaled biasing with back-side rendering.

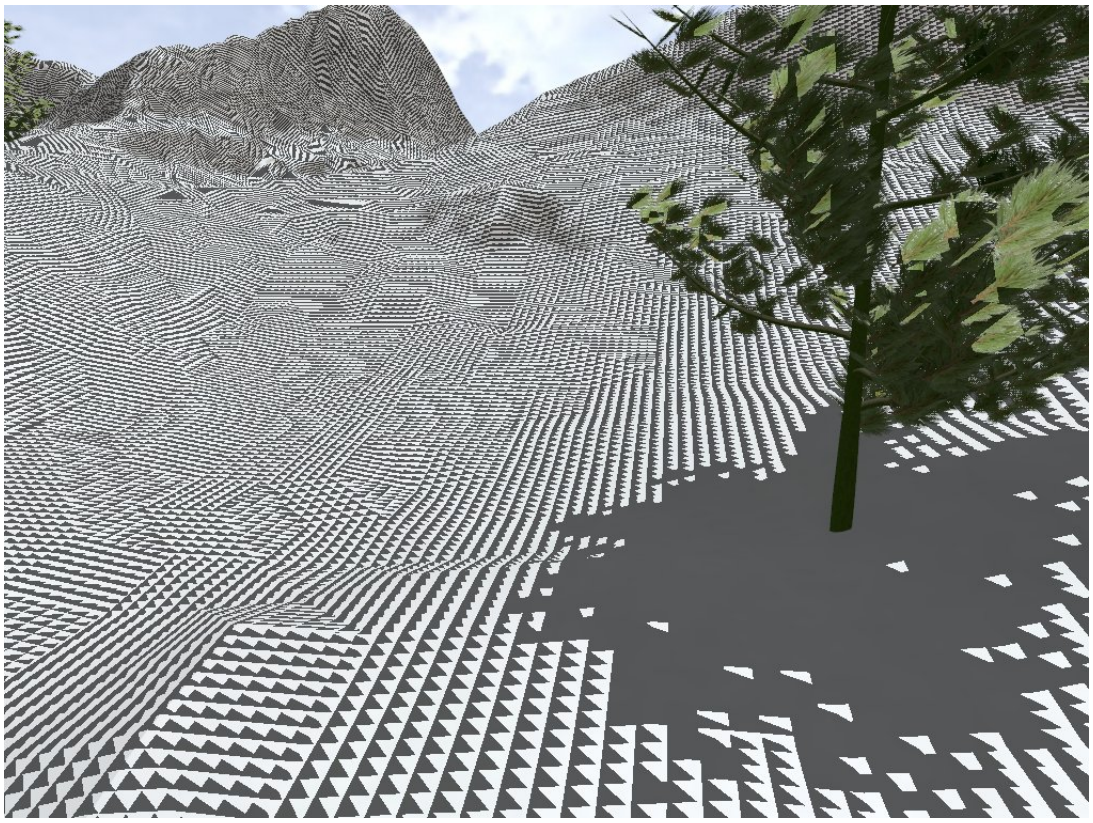


Figure 2.2: Self shadowing artifacts caused by resampling inaccuracies.

2.2.2 Projection aliasing

Due to the limitation of the shadow map resolution for almost every shadow mapping algorithm, aliasing errors mainly caused by undersampling need to be kept in mind. According to Stamminger and Drettakis [SD02] these aliasing artifacts can be divided into projection aliasing and perspective aliasing. Projection aliasing mainly appears on surfaces that are almost parallel to the light direction (see Figure 2.3) because such surfaces are hardly visible from the light's point of view and therefore very sparsely sampled.

Since this error appears locally at certain objects, there is no global solution to this problem which can be applied to the whole scene. For direct lighting a good workaround is to minimize the ambient term since the diffuse term $dot(L, N)$ is already very small and hides the artifacts. A possible solution are Adaptive Shadow Maps [FFBG01], where an ordinary shadow map is hierarchically subdivided to provide a better resolution for visually important regions. Scherzer [Sch05] proposed an approach by blurring the shadow map in eye space and use it as intensity lookup for the final rendering to hide the projection aliasing artifacts. Both previously mentioned methods require some additional extensive computations or rendering passes and lessen the performance regarding the real-time capabilities.

Summarizing, without an expensive detailed analysis of the scene geometry, the problem of projective aliasing cannot be fully resolved and remains for any real-time shadow mapping approach.

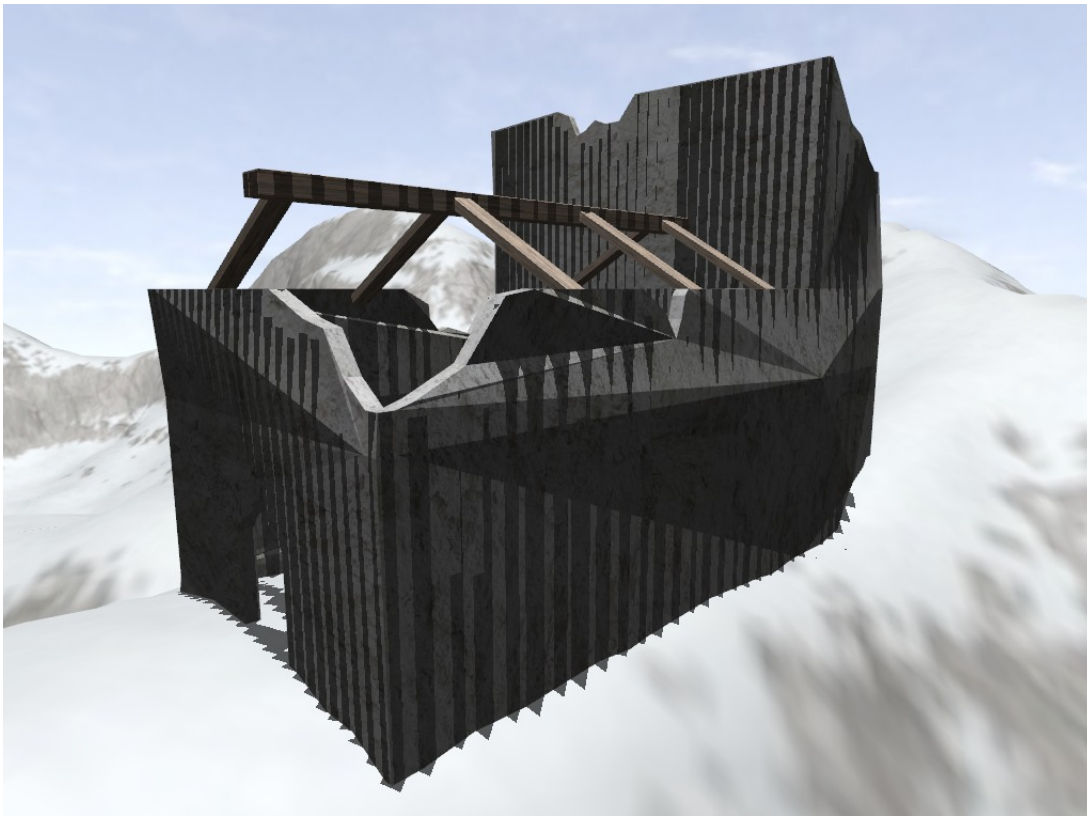


Figure 2.3: Projection aliasing artifacts caused by very sparsely sampled surfaces.

2.2.3 Perspective aliasing

As mentioned in the previous sections, common shadow mapping stores the scene objects with a fixed resolution in the shadow map using the light-space. Therefore there isn't any information about the eye view and its origin included. Because of the perspective eye view, nearby objects are shown larger than distant objects. So is the stored information in the shadow map after the projection into the scene. This normally results in a shadow resolution that is too low for nearby objects (undersampling) and a too high for distant objects (oversampling). This so called perspective aliasing gets visible through the typical jagged shadow boundaries nearby the viewing position (see Figure 2.4).

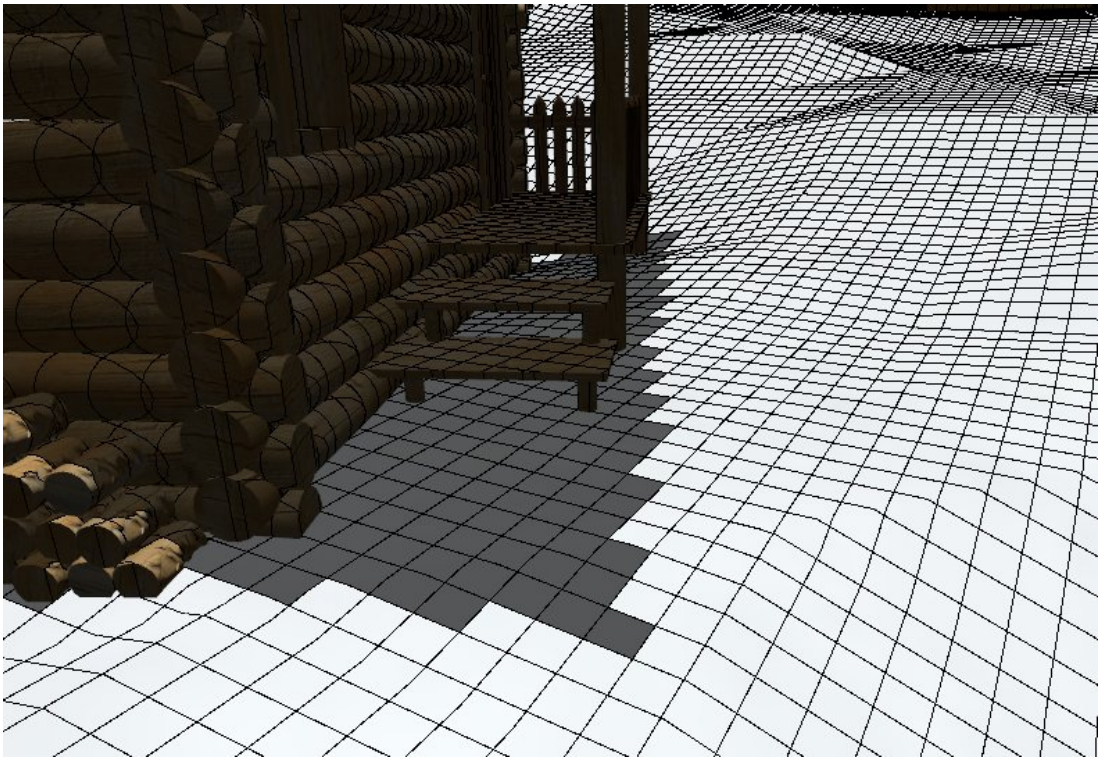


Figure 2.4: The typical jagged shadow boundaries, including corresponding texel borders, caused by perspective aliasing through undersampling nearby the camera.

Brabec et al. [BAS02] showed how important it is to focus the shadow map on the only visible area of the scene respectively the current camera frustum to reduce perspective aliasing and improve the shadow quality. We will discuss the focusing of the shadow map in detail later in this thesis.

Further solutions to the problem of perspective aliasing are the so called *warping algorithms* like for example Perspective Shadow Maps (PSM) [SD02] or Light Space Perspective Shadow

Maps (LiSPSM) [WSP04] which redistribute the shadow map samples.

Another approach to minimize the perspective aliasing error is to split the view frustum into several partitions and use multiple shadow maps. A common approach, called *z-partitioning*, which splits the view frustum along its length, was proposed by Tadamura et al. [TQJN99]. Since Tadamura's method cannot be applied on current graphics hardware, in 2006 Engel and Zhang introduced almost simultaneously two fully hardware accelerated solutions of this technique called Cascaded Shadow Maps (CSM) [Eng06] by Engel and Parallel Split Shadow Maps (PSSM) [ZSXL06] by Zhang et al.

A further approach that combines warping and partitioning by splitting the view frustum into its side faces and using for each face a single shadow map was introduced by Kozlov [Koz04].

Not only are the previous mentioned methods good ways to minimize the perspective aliasing error, they also can be easily combined with each other for further shadow quality improvements.

In Chapter 3 we will discuss several redistribution- and partitioning techniques and the opportunities of combining them. We will also visualize the perspective aliasing error and analyze its behavior for the discussed shadow mapping methods.

2.3 Error analysis

As previously mentioned, since shadow mapping is an image-space technique using shadow maps with finite resolutions, we have to deal with aliasing errors. Since almost every introduced shadow mapping technique, trying to reduce aliasing errors, is based on a prior error analysis, we will give a short overview of the most common formulations of shadow map aliasing in the following sections.

2.3.1 Simple error analysis

With the introduction of PSM Stamminger and Drettakis [SD02] were the first who formulated the distribution of the aliasing errors in a scene along the light-space z-axis in a simplified way. By assuming an overhead directional light source, they describe the aliasing error for a surface element located somewhere on the z-axis of the view frustum (see Figure 2.5) and decompose shadow map aliasing into scene-independent *perspective aliasing* which only depends on the relative position of the light source to the camera, and *projection aliasing* which depends on the orientation of the surfaces in the scene in relation to the light direction.

Based on Stamminger's and Drettakis' error analysis Wimmer et al. [WSP04] introduce another simplified formulation of shadow map aliasing. According to Figure 2.5 they assume a local parameterization of the shadow map from 0 to 1 between the near plane and the far plane of the view frustum. This means the shadow map is already properly focused to the visible parts of the scene (see Section 3.1) and allows them to analyze different parameterizations. With the

additional assumption that a small edge can be translated along the z-axis of the view frustum their formulation of shadow map aliasing in light-space z-direction results in

$$\frac{dp}{ds} = \frac{1}{z}\frac{dz}{ds}\frac{\cos\alpha}{\cos\beta} \tag{2.1}$$

where the term $\frac{1}{z}\frac{dz}{ds}$ describes perspective aliasing and $\frac{\cos\alpha}{\cos\beta}$ corresponds to projection aliasing.

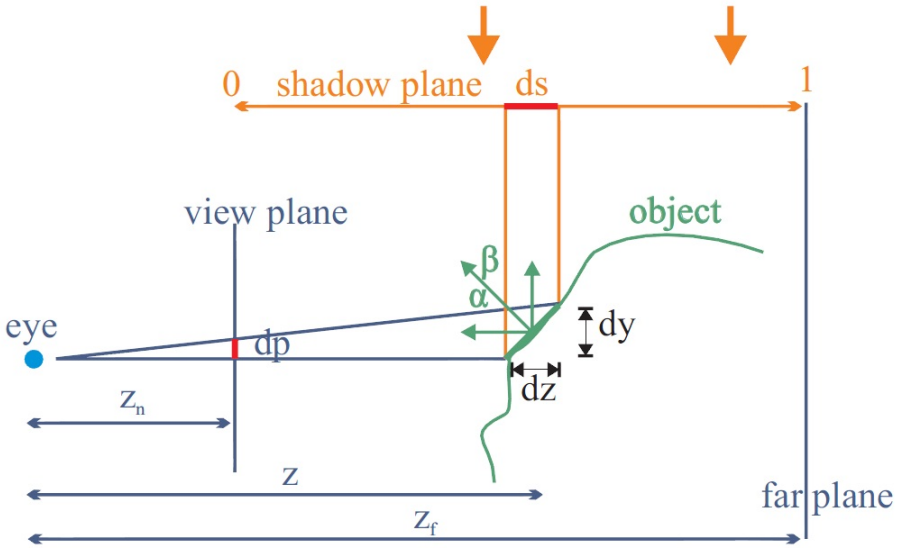


Figure 2.5: Illustration of shadow map aliasing for an overhead directional light source. Image courtesy of Wimmer et al. [WSP04].

2.3.2 Accurate error analysis

Lloyd presents in his thesis [Llo07] respectively in Lloyd et al. [LGQ+08] an accurate analysis of shadow map aliasing. Compared to the previous discussed simplified formulations he no longer assumes an overhead directional light source and takes the actual location of the investigated surface element into account (see Figure 2.6).

According to Lloyd [Llo07] and Lloyd et al. [LGQ+08] an accurate formulation of the aliasing error in light-space z-direction is given by

$$m = \frac{r_j}{r_t}\frac{dG}{dt}\frac{W_l}{W_e}\frac{n_e}{n_l}\frac{d_l}{d_e}\frac{\cos\phi_l}{\cos\phi_e}\frac{\cos\psi_e}{\cos\psi_l}. \tag{2.2}$$

Considering Figure 2.6, in this formulation

- r_j, r_t are the image height and the shadow map resolution in t-direction

- $\frac{dG}{dt}$ is the derivative of the shadow map parameterization (corresponds to $\frac{dz}{ds}$ at the simplified analysis)
- W_l, W_e are the widths of the light and image planes
- n_l, n_e are the near plane distances of the light frustum and the eye frustum
- d_l, d_e are the distances from the light respectively eye to the investigated surface element (d_e corresponds to z in the simplified formulation
- ϕ_l, ϕ_e are the angles between light/eye beam and image/shadow map plane normal
- ψ_l, ψ_l are the angles between light/eye beam and the surface normal of the investigated element (correspond to α and β in the simplified formulation).

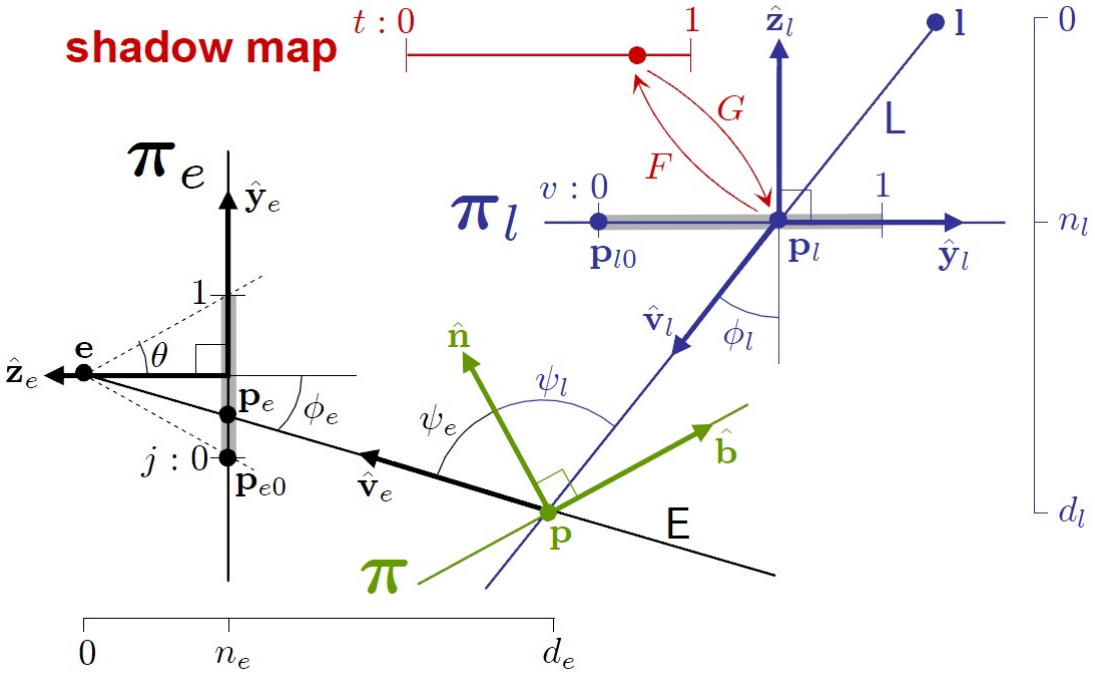


Figure 2.6: Generalized illustration of shadow map aliasing for the accurate computation of the shadow map aliasing error. Image courtesy of Lloyd [Llo07].

For a detailed description of this formulation we refer to Lloyd et al. [LGQ+08].

2.3.3 Error analysis for both shadow map directions

According to Lloyd et al. [LTYM06] the aliasing errors change with different parameterizations in both shadow map directions x and z (see Figure 2.7). To achieve their goal to find the best possible combination of warping and partitioning regarding perspective aliasing errors and real-time capabilities they extend Wimmer's simplified error analysis (Wimmer et al. [WSP04], Wimmer and Scherzer [WS06]) considering both shadow map dimensions.

Similar to Wimmer et al. they assume an overhead directional light source respectively a point light with a fairly narrow field of view. Based on this assumption Lloyd et al. [LTYM06] introduce a new error metric for determining the distribution of the perspective aliasing error over the length of the view frustum along the eye's view vector for warped and partitioned shadow algorithms. For an image resolution res_x x res_y, a shadow map resolution res_s x res_t and the warping parameter n' - controls the warping strength - their formulation of the aliasing errors m_x in x-direction and m_z in z-direction results in:

$$m_x(z, n') = \frac{res_{ix}}{res_s} \; f \; \left(\frac{(n' + z - n)}{z\,(n' + f - n)} \right), \tag{2.3}$$

$$m_z(z, n') = \frac{res_{iy}}{res_t} \; \frac{(f - n)}{2 \tan \Theta} \; \left(\frac{(n' + z - n)^2}{z n'\,(n' + f - n)} \right). \tag{2.4}$$

The above equations describe the perspective aliasing error distribution for a common warped shadow mapping technique like for example LiSPSM. Θ corresponds to the half the field of view (FOV), n and f denote the near-plane and far-plane of the eye's view-frustum and z stands for the depth of the current point.
By setting $n' = n$ we get the perspective errors for PSM:

$$m_x(z, n') = \frac{res_{ix}}{res_s}, \tag{2.5}$$

$$m_z(z, n') = \frac{res_{iy}}{res_t} \; \frac{(f - n)}{2 \tan \Theta} \; \frac{z}{n\; f}. \tag{2.6}$$

For $n \to \infty$ and application of l'Hôpital's rule the error formulation for uniform shadow mapping results in

$$m_x(z, n') = \frac{res_{ix}}{res_s} \; \frac{f}{z}, \tag{2.7}$$

$$m_z(z, n') = \frac{res_{iy}}{res_t} \; \frac{(f - n)}{2 \tan \Theta} \; \frac{1}{z}. \tag{2.8}$$

Figure 2.8 shows two example plots of $m_x(z, n')$ and $m_z(z, n')$ comparing the error distributions of uniform shadow maps, LiSPSM and PSM.

Since the error metric discussed in this section can be easily implemented to perform an interactive online analysis, and treats both shadow map dimensions we have decided to use this

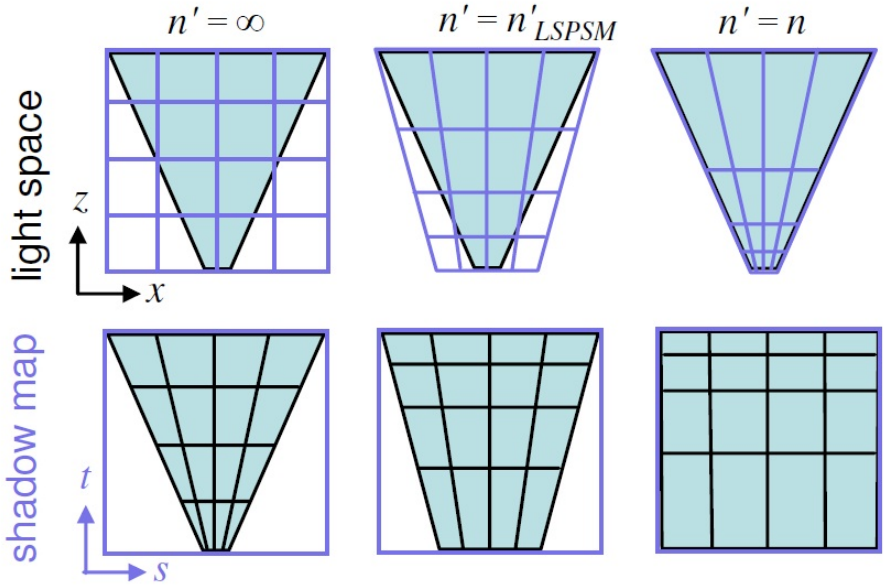


Figure 2.7: Illustration of error changes in both directions x and z for different parameterizations (Left: Uniform shadow mapping, Center: LiSPSM, Right: PSM). Top: the compression of the projected grid in light space to match the view frustum. Bottom: the expansion of the grid to match the shadow map dimensions. Image courtesy of Lloyd et al. [LTYM06].

metric for our further analysis and visualizations of the perspective aliasing errors in the following chapters of this work.

2.4 Soft Shadows

Generating physically correct soft shadows is still one of the most challenging research fields in real-time rendering. Most soft shadow algorithms or filtering techniques like percentage closer filtering (PCF) [RSC87] or Variance Shadow Maps (VSM) [DL06] assume a fixed size penumbra just to hide the aliasing artifacts caused by undersampling.

Due to the increasing performance capabilities of modern graphics hardware, in the past few years various approaches for generation of realistic soft shadows with a variable penumbra like for example Percentage Closer Soft Shadows (PCSS) [Fer05] have been introduced.

Since the focus of this thesis lies on analyzing and minimizing the perspective aliasing error, we limited the amount of our soft shadow mapping experiments on PCF-based techniques.

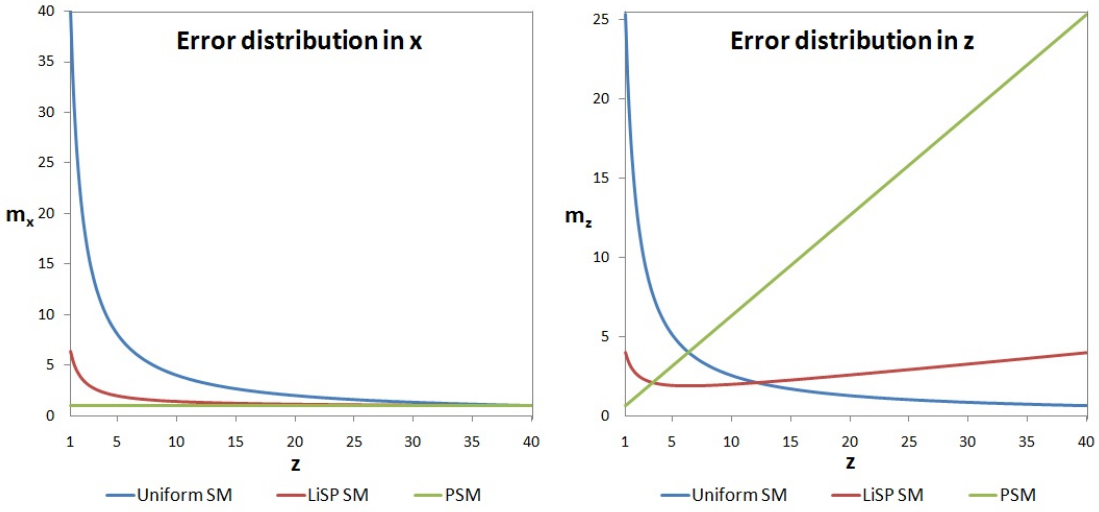


Figure 2.8: Perspective aliasing error distributions along the length of the view frustum for uniform shadow mapping ($n' = \infty$), LiSPSM and PSM ($n' = n$) with $n = 1$, $f = 40$.

CHAPTER 3

Advanced shadow mapping

So far we have discussed the common shadow mapping algorithm and how it can be implemented on today's graphics hardware. We have seen that due to the limitations of an image space technique it suffers from several problems like for example perspective aliasing caused by undersampling nearby the camera and oversampling of the objects distant from the camera.

In Chapter 2 we mentioned various advanced shadow mapping algorithms addressing the problem of perspective aliasing. In this chapter we will review the most common of these algorithms in detail and explain how they can be implemented using the capabilities of modern graphics hardware.

All techniques, which will be discussed in the following sections, are implemented in our introduced shadow mapping framework to be able to experiment with different configurations and combinations of them.

In Section 3.1 we will discuss the focusing of the shadow map on the visible objects of the scene and show how tremendous the improvements of the shadow quality and reduction of perspective aliasing can turn out. However especially for very large scenes aliasing artifacts still can be visible for certain camera positions or rotations and the shadow quality can change very fast within a few frames.

For this reason we will describe in the subsequent sections some advanced techniques that have been introduced to further reduce the perspective aliasing errors and improve the robustness and shadow quality of focused shadow mapping. The discussed methods are mainly based on redistribution of the shadow map samples, on using multiple shadow maps by partitioning the camera's view frustum and on combinations of the redistribution and partitioning techniques.

At the end of this chapter we will review the possibility of how perspective aliasing can be

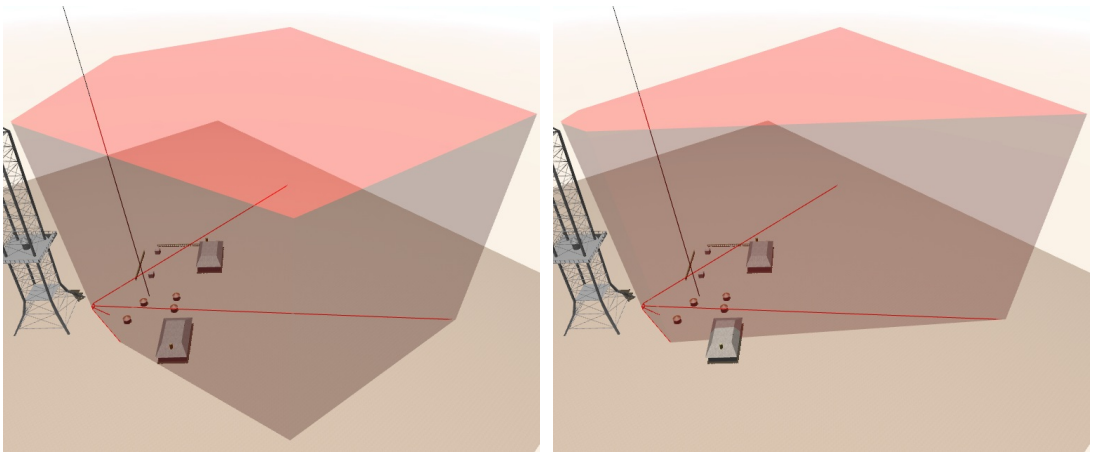
hidden by using common filtering techniques and how filtering can be combined with multiple used shadow maps.

3.1 Focused shadow mapping

Stamminger and Drettakis [SD02] showed a simple way to adjust the light's frustum to the current camera frustum. These adjustments allow that the available shadow map resolution and the depth buffer precision can be used in a nearly optimal way, and shadow map aliasing can be greatly reduced. The basic idea is to find a convex intersection body containing all shadow casters visible from the current eye's point of view and adapt the light source's view frustum to it.

3.1.1 Calculation of convex intersection body

Considering the bounding box of all scene objects S, V the current eye viewing frustum and L the light source viewing frustum. For point lights the light source is at position l and for directional lights L equals R^3 and l is at infinity for a given light direction. According to Stamminger and Drettakis [SD02] for point lights we need to compute the convex hull M of V and l so that M contains all rays from l to points in V. In most cases V is just partly inside of S. So if we compute M using V we would take unnecessary points outside of S into account which mostly adds points to M that are actually outside of V (see Figure 3.1). Based on this observation Wim-



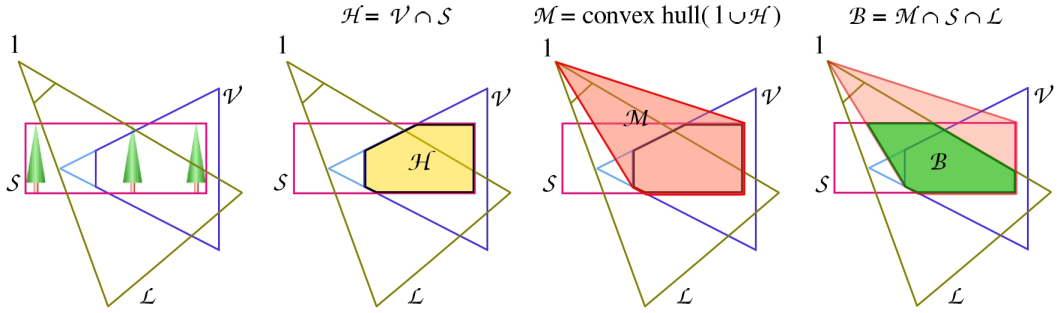
(a) (b)

Figure 3.1: Comparison of intersection body (red semi-transparent solid object) computation: (a) intersection body according to Stamminger's and Drettakis' [SD02] method. (b) intersection body computed with our method proposed by Wimmer and Scherzer [WS06] - note the not included points lying outside of the view frustum (red wire-frame object). Light direction: black line.

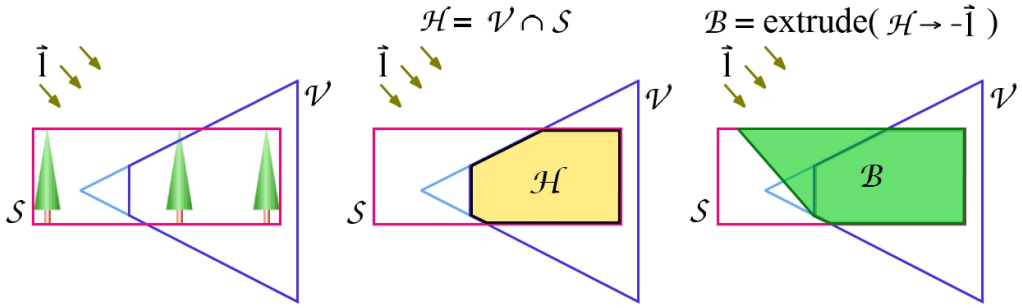
mer and Scherzer [WS06] suggest to build the convex hull M using $H = V \cap S$ (intersection of

the view frustum and the scene's bounding box) and l. Finally, to get the final convex body B, we need to remove all points outside the scene's bounding box S and the light frustum L from M: $B = M \cap S \cap L$. Since the light position l for directional lights is at infinity, we cannot compute the convex hull M. In this case we compute M by extruding each point of H along the light direction until we are outside of the scene's bounding box. Figure 3.2 shows a schematic illustration of the computation of B for point lights (Figure 3.2a) respectively directional lights (Figure 3.2b).

Our experiments have shown that our used method to calculate the convex body in the discussed way and adapt the light projection to it, as explained in the next section, works very fast and robust and can be easily applied to many other shadow mapping algorithms. An example for the final computed convex intersection body is shown in Figure 3.3.

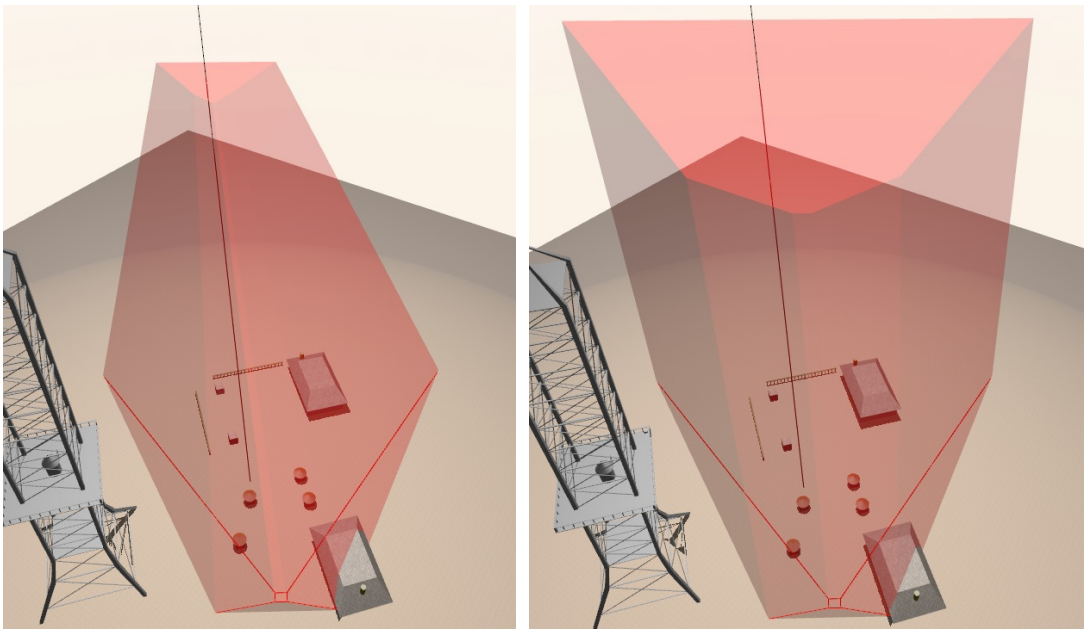


(a) Convex body B is obtained by extending the intersection H of V and S towards the light position l and intersect the resulting convex hull M with S and L.



(b) Convex body B results from extruding the intersection H of V and S towards the negative light direction $\vec{l}$ until a boundary surface of S is reached.

Figure 3.2: Computation of convex intersection body B for a point light (a), and for a directional light (b). S denotes the scene bounding box, V the current view frustum, L the light frustum, l the light position of the point light and $\vec{l}$ the light direction of the directional light.



(a) Convex body B for a point light source.

(b) Convex body B for a directional light source.

Figure 3.3: Example visualization of the final convex intersection body B for a point light source (a) and a directional light source (b) in 3D. The black line corresponds to the light direction and the red line to the edges of the view frustum.

3.1.2 Focusing the shadow map

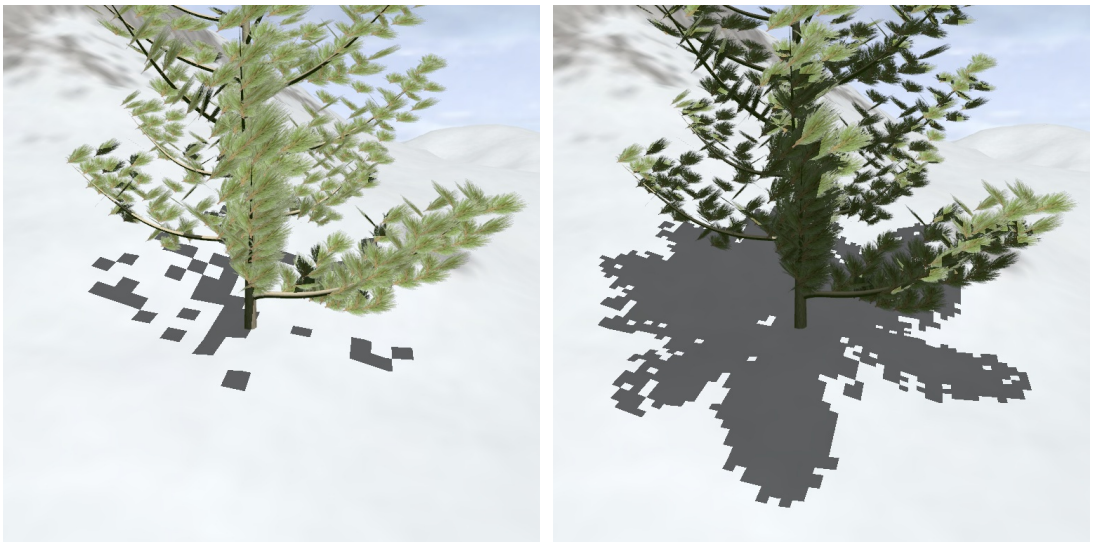
Once we have calculated the convex body B we need to adjust the light projection according to the points of B. The first step of this adjustment is to find the axis aligned bounding box of B in the post perspective light-space. Based on the computed bounding box we create an orthographic projection matrix in the following way: Considering the light-space bounding box extents min and max the projection matrix can be formulated for a right handed coordinate system as

$$A_{fit} = \begin{pmatrix} \frac{2}{max_x - min_x} & 0 & 0 & -\frac{max_x + min_x}{max_x - min_x} \\ 0 & \frac{2}{max_y - min_y} & 0 & -\frac{max_y + min_y}{max_y - min_y} \\ 0 & 0 & \frac{1}{max_z - min_z} & -\frac{min_z}{max_z - min_z} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.1}$$

Transformation of the current light projection by this projection causes the final adaptation of the light frustum onto the region covered by the convex body B.

As mentioned before, additionally to the great improvements of shadow quality for most scenes

(see Figure 3.4), focused shadow mapping is as fast as common shadow mapping and it can be



(a) Standard shadow mapping. ($n = 1$, $f = 1500$, $\Theta = 30°$)

(b) Focused shadow mapping. ($n = 1$, $f = 500$, $\Theta = 30°$)

Figure 3.4: Comparison between standard shadow mapping (a) and focused shadow mapping (b).

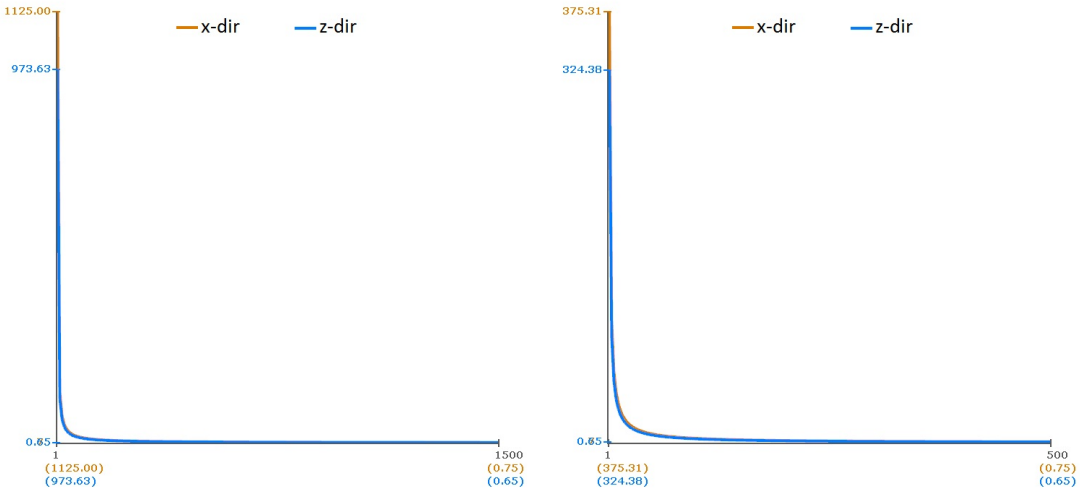
easily combined with other shadow mapping algorithms. However for dynamic scenes and camera movements it can lead to 'flickering' or 'swimming' shadow boundaries, which is actually a problem of almost every advanced shadow mapping method. We will discuss a possible solution to these swimming artifacts later in this thesis.

In addition to the previous calculations Brabec [BAS02] proposes a way to linearize the depth values of the z-buffer to increase the depth precision for distant objects from the light's point of view. He also introduces a technique to increase the near plane distance of the light frustum of point lights by excluding objects outside of the viewing frustum and using the clamping function of the graphics hardware. Considering uniform shadow mapping, these additional methods work fine, however according to our experiments using them with shadow mapping algorithms where the samples get redistributed like for example LiSPSM, they can lead to problems.

3.1.3 Error analysis

Compared to standard uniform shadow maps the characteristics of the perspective aliasing error along the view frustum does not change for focused uniform shadow maps. For an overhead light the maximum perspective aliasing occurs at the near plane distance and the minimum at the far plane distance. Figure 3.5 shows a comparison of the perspective error between standard uniform shadow mapping and focused uniform shadow mapping. It can be seen that the absolute

maximum of the errors usually can be reduced by a great amount that depends mainly on the final extent of the convex intersection body B (see also Equation 2.7 and Equation 2.8).



(a) Standard uniform shadow mapping

(b) Focused uniform shadow mapping

Figure 3.5: Comparison of the perspective aliasing error distribution between standard uniform shadow mapping (a) and focused uniform shadow mapping (b). Note that the overall characteristic of both curves is still the same but the maximum error values of (b) are almost one third compared to (a).

3.2 Warping

In this context warping is a commonly used term for shadow mapping algorithms that handle perspective aliasing by redistribution of the shadow map samples since the light-space is warped in a certain way. The main idea of almost any of these methods is to use a perspective transform based on the post perspective eye-space to reconstruct the light-space. This should lead to a uniform distribution of the shadow map samples in the eye space.

The first completely hardware accelerated algorithm using this approach was PSM [SD02]. Unfortunately, according to Wimmer et al. [WSP04], this technique suffers from serious robustness and quality issues in real world scenarios. Therefore we lay our focus on alternative respectively improved warping-techniques in the following sections.

3.2.1 Light Space Perspective Shadow Maps

Following the basic idea of PSM to minimize the perspective aliasing error via perspective redistribution of shadow map samples, Wimmer et al. introduced in [WSP04] LiSPSM by using

the main advantages and avoid the weaknesses of PSM. LiSPSM were motivated by the observations that, contrary to PSM, the applied perspective transform to warp the shadow map does not need to be tied to the view frustum, and using a warp that only affects the shadow map plane and not the axis perpendicular to the shadow map is sufficient.

According to Wimmer et al. [WSP04] LiSPSM is applied generally in four steps. The first step includes the focusing on the convex body B which encloses all relevant light rays for the shadow computation. The calculation of B follows exactly the descriptions in Section 3.1.

In a second step B is enclosed with an appropriate perspective frustum P. According to Wimmer et al. [WSP04] the parameters for this perspective frustum are found in light-space which is constructed with respect to the light direction, the camera position and the camera's view direction. Contrary to the original LiSPSM paper we construct the light-space in right handed order based upon the original light viewing transformation in order to save CPU load and since we need to apply the final LiSPSM projection matrix P independently from the light projection for a certain partitioning approach which will be discussed later in this thesis. Considering this circumstance we construct the light-space matrix L in the same way as explained in Wimmer's and Scherzer's revised version of the original LiSPSM paper [WS06] except we do not exchange y and z. For a schematic illustration of our used light-space coordinate system and the related perspective frustum see Figure 3.6.

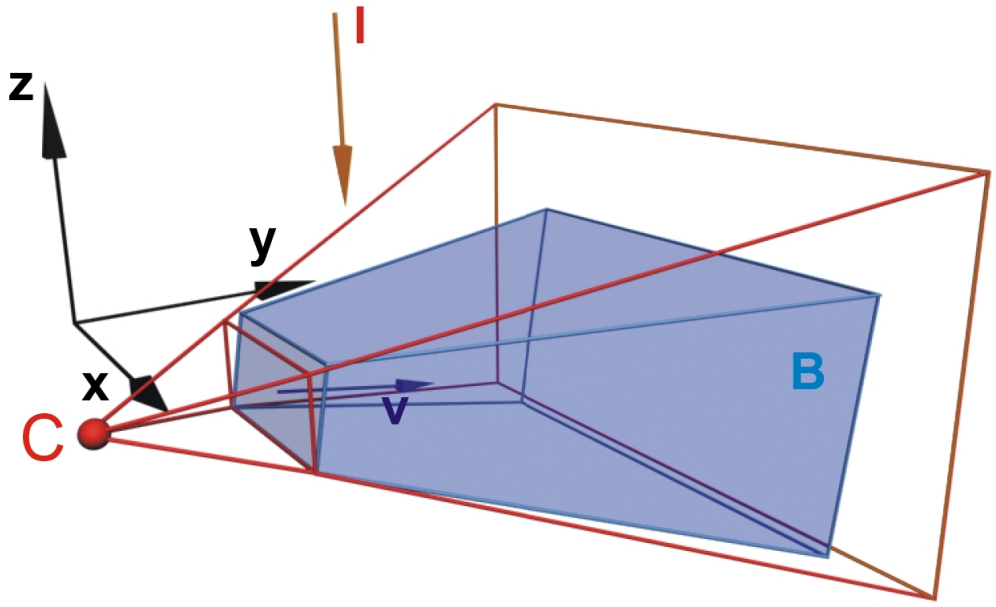


Figure 3.6: Example configuration of the perspective frustum P including the corresponding light space coordinate system. Image courtesy of Wimmer et al. [WSP04]

Before the final rendering step the free parameter n of the perspective frustum P needs to be defined. This parameter represents the distance between the projection center C of P and its near plane and controls the actual strength of the perspective warp in the shadow map. For smaller n the warping effect will get stronger until it reaches PSM when n equals the near plane distance of the view frustum. Increasing n lessens the perspective warp approaching uniform shadow maps for $n = \infty$.

Based on their analysis of the perspective aliasing error Wimmer et al. [WSP04] found an analytical optimal solution for the free parameter and call the parameter therefore n_{opt}. Choosing n_{opt} according to theirs approach, the perspective aliasing error has two identical maxima at the near plane and the far plane of the (focused) view frustum and a minimum nearby the viewer if the view direction is perpendicular to the light direction. Both maxima result in a much lower value than the maximum of uniform shadow maps at the near plane distance. For parallel light direction and view direction n_{opt} approaches infinity and LiSPSM falls back to uniform shadow mapping. Wimmer et al. [WSP04] call this special case the *duelling frusta* problem and show that for this configuration uniform shadow mapping is the optimal solution.

According to Wimmer et al. [WSP04] their formulation of n_{opt} works fine for directional lights but not for point lights. Based on this observation Wimmer and Scherzer [WS06] derived a generalized formula which works for directional and point lights (see Equation 3.2). This new formulation takes all possible clipping cases - view frustum clipping by the scene bounding box or a perspective light frustum - into account and considers the change in scale involved with a perspective light projection:

$$n_{opt} = \frac{d}{\sqrt{z_1/z_0} - 1} \tag{3.2}$$

In this formulation d denotes the extent of the warping frustum P in the light-space z-direction. The values z_0 and z_1 represent the extent of the perspective warping frustum and correspond to the eye-space z-coordinates where the perspective aliasing error reaches its maxima.

Once the light-space matrix L, the free parameter n_{opt} and the extent of the warping frustum have been computed we need to determine the projection center C. For this purpose Wimmer and Scherzer [WS06] propose to use an appropriate point C_{start} on the near plane of the warping frustum and translate the distance n_{opt} back from the near plane to obtain C. For the definition of C_{start} they compute the nearest vertex e to the eye on the intersection body LVS which denotes the intersection of the light frustum (in case of a point light), the view frustum and the scene bounding box. This technique works fine in most cases, however their definition of C_{start} results in some minor robustness issues and cannot be applied in combination with our chosen partitioning approach. While the camera moves or the light position gets changed this vertex does not change its position continuously according to the continuous movement of LVS. These small erratic changes do not cause any problems for the actual shadow projection but it imposes some robustness issues on the illustration of the light view or the texel borders. The arising problems with this approach to define C_{start} in association with the combination of LiSPSM and partitioning will be discussed later in this thesis.

In Chapter 4 we will propose an alternative approach to compute C that solves the robustness problems and avoids the errors involved with warped partitioning in most cases.

Once C has been computed correctly, the frustum planes of P can be initialized in the same way as described in [WS06].
The final rendering passes for shadow map generation and shadow projection work analog to standard shadow mapping except for the perspective frustum P that is applied by the graphics hardware during both of them. For this reason, there are no additional costs involved compared to standard shadow mapping.

3.2.2 Re-parametrized Light Space Perspective Shadow Maps

So far Stamminger and Drettakis [SD02] and Wimmer et al. [WSP04] respectively Wimmer and Scherzer [WS06] mainly focused on perspective error analysis along the z-direction of light-space (in our case the y-direction). As discussed in Chapter 2.3, Lloyd et al. [LTYM06] extend their error analysis and consider the perspective error for both directions, x- and z-direction, parallel to the shadow map plane. In [Llo07] Lloyd shows that the error distribution of LiSPSM [WS06] increases especially in x-direction for $\gamma < \Theta$, where γ corresponds to the angle between light-direction and view-direction and Θ stands for half the field of view (FOV) of the camera's view frustum.

Therefore he proposes a new parameterization of the warping parameter by subdividing the range of γ into four intervals:

$$\eta = \begin{cases} -1 & \gamma \leq \gamma_a \\ -1 + (\eta_b + 1)\frac{\gamma - \gamma_a}{\gamma_b - \gamma_a} & \gamma_a < \gamma \leq \gamma_b \\ \eta_b + (\eta_c - \eta_b)\sin(90^\circ \frac{\gamma - \gamma_b}{\gamma_c - \gamma_b}) & \gamma_b < \gamma \leq \gamma_c \\ \eta_c & \gamma_c < \gamma \end{cases} \tag{3.3}$$

$$\gamma_a = \tfrac{\Theta}{3} \quad \gamma_b = \Theta \quad \gamma_c = \Theta + 0.3(90^\circ - \Theta)$$

$$\eta_b = -0.2 \quad \eta_c = 0$$

Using this new parameterization via η, the free parameter of the perspective warping frustum, which we call in this case n_{repar}, results according to Lloyd [LGQ+08] in:

$$\begin{aligned} n_{repar}(\eta) &= \frac{W_{lz}}{\alpha - 1} \begin{cases} \frac{1 + \sqrt{\alpha} - \eta(\alpha - 1)}{\eta + 1} & \eta < 0 \\ \frac{1 + \sqrt{\alpha}}{\eta\sqrt{\alpha} + 1} & \eta \geq 0 \end{cases} \\ \alpha &= \frac{f_e}{n_e} \end{aligned} \tag{3.4}$$

where n_e and f_e represent the eye-space near plane and far plane distances of the view frustum and can be seen as z_0 and z_1 in Equation 3.2. We assume a directional light or a point light with a fairly narrow field of view, so that it can be treated as a directional light source. Under these

assumptions according to Lloyd [LGQ+08] W_{lz} results for an overhead light in

$$W_{lz} = W_n^{'} + W_{s2}^{'} + \begin{cases} W_{s1}^{'} & 0 \leq \gamma \leq \Theta \\ 0 & \Theta < \gamma \leq 90^{\circ} \end{cases}$$
$$W_n^{'} = W_n \cos\gamma \quad W_{s1}^{'} = W_s(1 - \cos(\Theta - \gamma) \quad W_{s2}^{'} = W_s \sin(\Theta + \gamma) \tag{3.5}$$
$$W_n = W_e \quad W_s = \frac{f_e - n_e}{\cos\Theta}$$

$W_n^{'}$, $W_{s1}^{'}$ and $W_{s2}^{'}$ are the exit faces of the view frustum projected on the shadow map plane which are computed by using simple geometry (see Figure 3.7).

The focusing step, the definition of the light-space matrix L, the computation of the projection center C and the final rendering step follow exactly the descriptions of common LiSPSM with n_{opt}.

For $\gamma > \Theta$ the results of LiSPSM with n_{repar} are almost the same as the results of LiSPSM with n_{opt}. According to Lloyd [Llo07], if $\gamma < \Theta$, approaching the duelling frusta case, LiSPSM with n_{repar} keeps the error, especially in x-direction, low compared to n_{opt} LiSPSM. A comparison of both LiSPSM techniques at a configuration where $\gamma < \Theta$ is shown in Figure 3.8. It includes a visualization of the texel borders and the sampling density via color overlay. For a detailed comparison of both LiSPSM techniques we refer to Lloyd [Llo07].

3.2.3 Error analysis

As discussed in Section 3.2.1 Wimmer et al. [WSP04], [WS06] choose the free parameter n_{opt} of the perspective warping frustum so that the error distribution in z-direction contains two maxima at the near plane distance and the far plane distance of the current view frustum. Using Equation 2.3 and Equation 2.4 to evaluate the errors and considering only these two points at the near plane and the far plane, it can be said that the error is in eye-space uniformly distributed in an optimal way resulting in a much lower maximum than for uniform shadow mapping. Since the samples are redistributed in both shadow map dimensions the maximum error in x-direction is reduced as well. But the redistribution in x-direction is not applied in an optimal way, so that the overall error distribution in x-direction still contains a maximum at the near plane distance and a minimum at the far plane distance just like uniform shadow maps (see Figure 3.9).

3.3 Z-partitioning

As mentioned in Chapter 2 the approach of using multiple shadow maps to decrease perspective aliasing and improve the shadow quality was proposed by Tadamura et al. [TQJN99] in the first place. Their proposition subdivides the view frustum into several *split frusta* and uses a dynamic array of shadow maps with varying resolutions to approximate the continuous resolution changes along the length of the view frustum. This method of subdividing the view frustum into several smaller perspective frusta along its length respectively along the eye-space z-axis to use it for

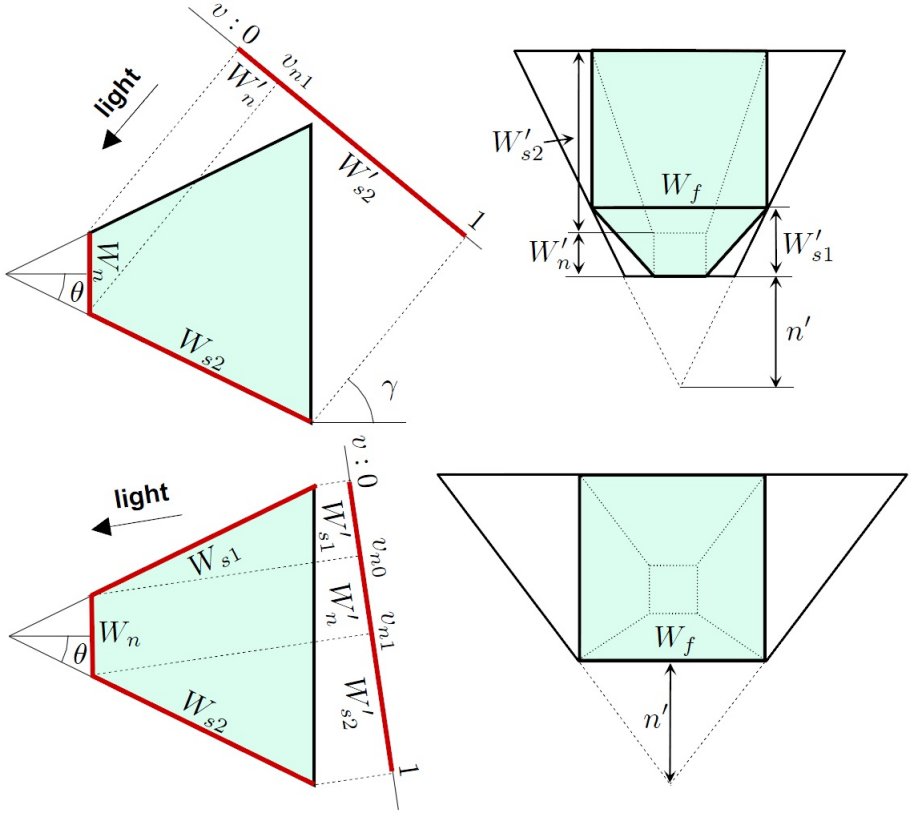
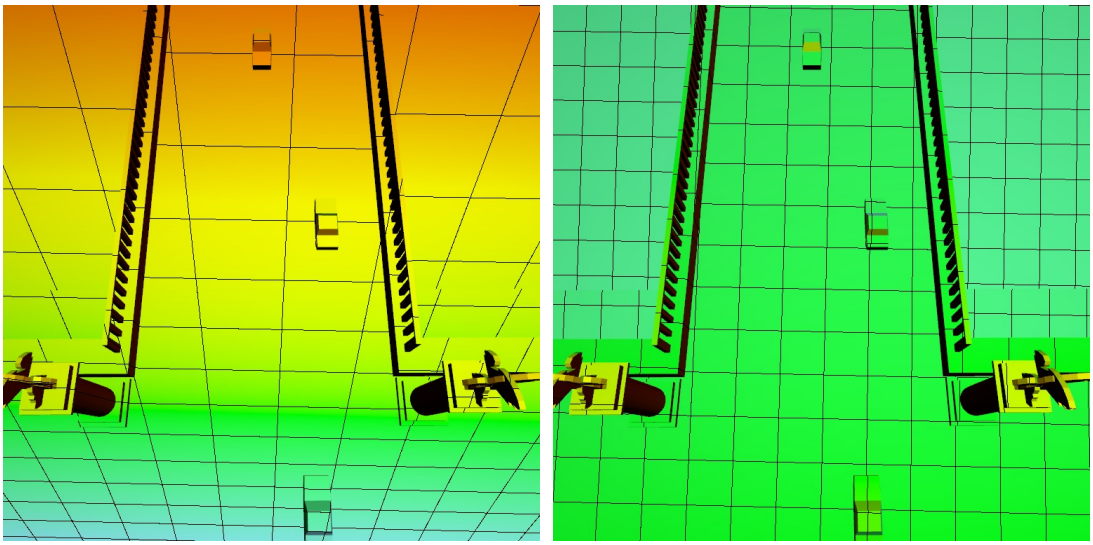


Figure 3.7: Parameterizing of the view frustum for a directional light. Left: side views of the view frustum with projected exit faces onto the shadow map plane at different light directions. Right: the corresponding light views with the surrounding perspective warping frusta shown in black. Image courtesy of Lloyd et al. [LGQ+08].

multiple shadow maps is called z-partitioning (see Figure 3.10). Since Tadamura's approach requires many complicated and time-consuming computations to find the optimal length of the split parts and needs support of varying resolutions for the shadow maps it is not applicable to hardware-accelerated real-time applications. Though this technique works very robustly and does not involve any special cases like for example the duelling frusta problem for LiSPSM.

Therefore in 2006 with CSM [Eng06] and PSSM [ZSXL06] two similar techniques were introduced that are based upon Tadamura's [TQJN99] approach and adapted to modern graphics hardware. PSSM introduced by Zhang et al. uses z-partitioning in combination with an efficient and robust split scheme based on an analysis of perspective parameterization. Considering the limitations of current graphics hardware, instead of varying resolution shadow maps, Zhang et al. apply for each split frustum a fixed resolution shadow map. As an additional improvement respectively simplification to Tadamura's approach they propose a faster way of rendering the shadows by using the capabilities of programmable GPUs.



(a) n_{opt} LiSPSM (b) n_{repar} LiSPSM

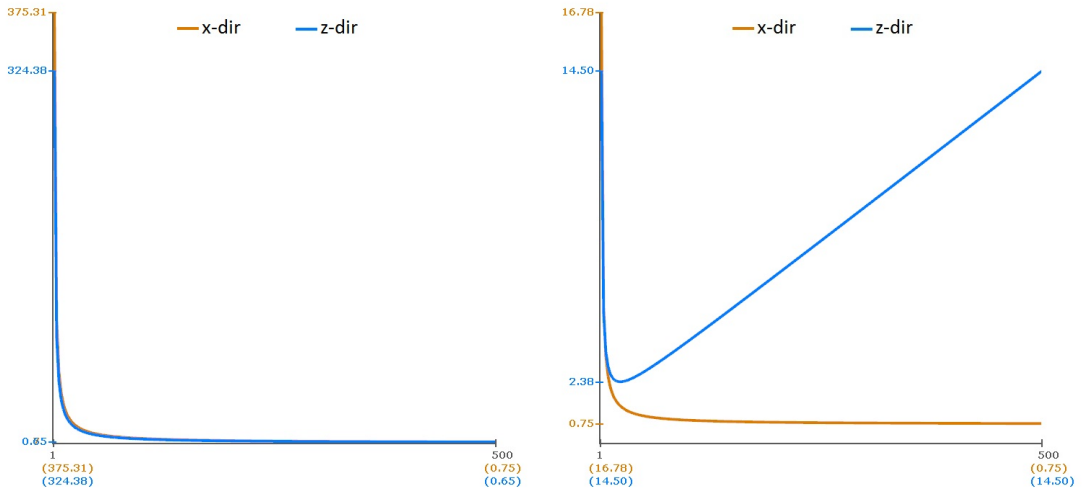
Figure 3.8: Error comparison of LiSPSM techniques. (a) shows LiSPSM with n_{opt} introduced by Wimmer and Scherzer [WS06]. (b) shows re-parametrized LiSPSM using n_{repar} introduced by Lloyd et al. [LTYM06]. Note that the re-parametrized method almost correspond to uniform shadow mapping while the results of n_{opt} LiSPSM are almost worse than uniform shadow maps. Parameters: $\Theta = 30°$, $\gamma = 20°$. For the corresponding color mapping see Figure 5.6.

Compared to common uniform or warped shadow mapping the main drawback of z-partitioning is that for each split frustum a single rendering pass is needed to generate the shadow map. This usually resulting performance degradation can be easily minimized through the following actions. The focusing technique discussed in Section 3.1 can be easily combined with z-partitioning by using the current split frustum instead of the entire view frustum. By applying this focusing to each split frustum, only the scene objects are rendered that lie in the current split part. Further improvements can be achieved by using the *multiple render targets*-functionality of current graphics hardware which can render different split parts simultaneously. We will discuss some additional opportunities to reduce the performance degradation later in this chapter.

Besides the reduction of the perspective error and the improvement of the shadow quality, z-partitioning can be seamlessly combined with other methods like for example LiSPSM, and in most applications we can even save texture memory compared to single-map methods.

3.3.1 Adjusting the view frustum

In most real world outdoor scenarios a distant sky box or a sky dome surrounds the actual scene. Therefore the camera's far plane has to be chosen sufficiently large to avoid unwanted far plane



(a) Focused uniform shadow mapping

(b) Light space perspective shadow mapping

Figure 3.9: Comparison of the perspective aliasing error distribution between focused uniform shadow mapping (a) and light space perspective shadow mapping (b) for an overhead directional light source. Note the two equal maxima at the near plane and the far plane of the (adjusted) view frustum. Parameters: $\Theta = 30°$, $\gamma = 90°$.

clipping of the sky box. To minimize the amount of empty space in the view frustum and optimize the available shadow map resolution Zhang et al. [ZSN07] suggest to adjust the camera's near and far planes before we split the view frustum. This guarantees that the view frustum contains all visible objects, relevant for shadow mapping, as tight as possible.

According to Zhang et al. [ZSN07] recommendations we compute the intersection of view frustum and scene bounding box $H = V \cap S$ and calculate for each point of the resulting intersection body the distance to the camera position along the current view vector. The new near and far planes of the adjusted view frustum are given by the minimum and the maximum of the computed distances. Figure 3.11 shows a comparison between z-partitioning with a not adjusted and an adjusted view frustum.

3.3.2 Split scheme

Based on Wimmer's et al. [WSP04] perspective aliasing analysis Zhang et al. [ZSXL06] discussed different well known split schemes. The simplest approach is to position the split planes uniformly along the camera's z-axis. Thus we get according to Zhang et al. [ZSXL06] for the i-th split position, with partitioning of the view frustum into m parts, with eye-space near plane distance n and eye-space far plane distance f a *uniform split scheme*

$$C_i^{uniform} = n + (f - n)i/m. \tag{3.6}$$

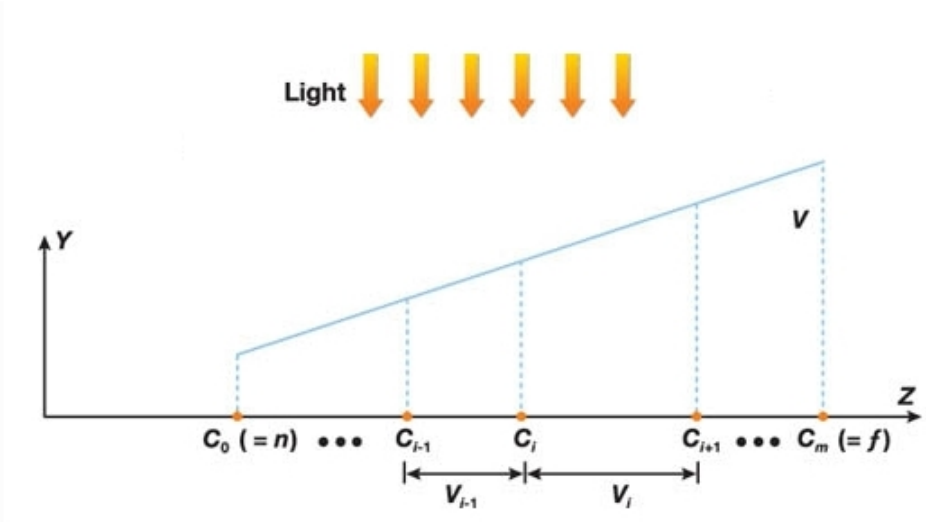


Figure 3.10: Schematic illustration of z-partitioning with an overhead light. The view frustum V is split along its length into m split frusta V_i. Image courtesy of Zhang et al. [ZSN07].

Zhang et al. shows that the distribution of the perspective aliasing error results for the uniform split scheme in the same as for common uniform shadow maps. This means we get undersampling nearby the camera and oversampling further away from the camera (see Figure 3.12(a)).

According to Wimmer et al. [WSP04] the optimal parameterization for an overhead light is logarithmic. Therefore Lloyd et al. [LTYM06] and Zhang et al. [ZSXL06] experimented with a *logarithmic split scheme* which approximates this optimal parameterization to approach the theoretical ideal even distribution of perspective aliasing errors:

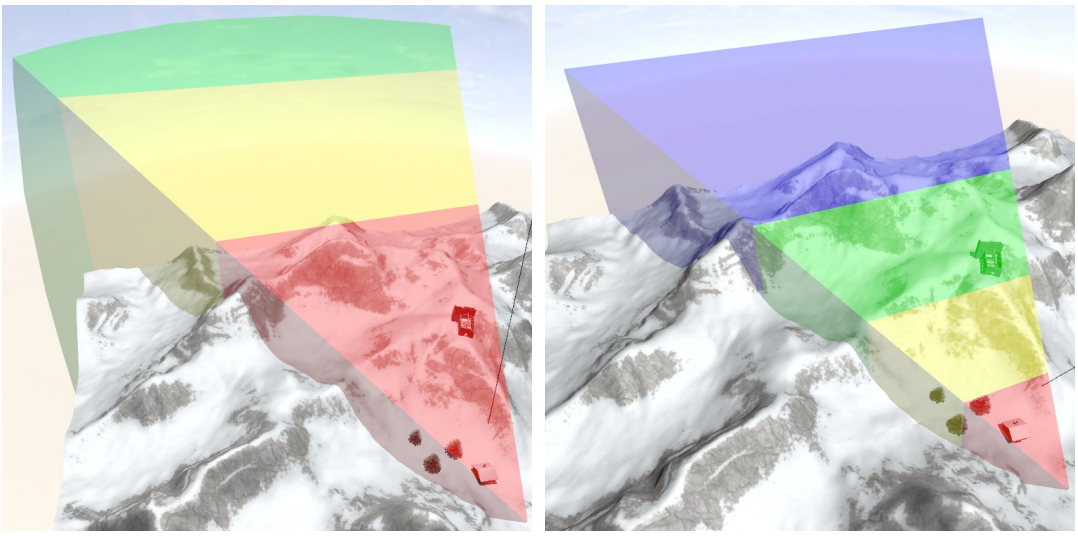
$$C_i^{log} = n(f/n)^{i/m}. \tag{3.7}$$

Zhang et al. [ZSXL06] showed that this logarithmic split scheme can be hardly applied in practice, since it results in oversampling near the view point and undersampling at distant points from the viewer (see Figure 3.12(b)).

Based on the previous observations, Zhang et al. [ZSXL06] proposes a new *practical split scheme* which represents a combination of the uniform and logarithmic split scheme:

$$C_i = (1-\lambda)\left(n + (f-n)\,\frac{i}{m}\right) + \lambda\left(n\,(f/n)^{i/m}\right) \quad \lambda \in [0,1] \tag{3.8}$$

In their original paper Zhang et al. chose $\lambda = 0.5$ since it produces appropriate sampling rates for the entire view frustum in most cases (see Figure 3.12(c)).



(a) View frustum splitting without near and far planes adjusting

(b) View frustum splitting with near and far planes adjusting

Figure 3.11: View frustum splitting with and without near and far planes adjusting. (a) the third (green) and the fourth (not visible) split frustum are completely outside of the relevant scene and therefore empty. (b) the far plane is adjusted to the visible scene objects so that the amount of empty space is minimized and the available shadow map resolution is used in an optimized way.

3.3.3 Split selection

On modern programmable GPUs the final rasterization and shadow rendering step can be easily done in one rendering pass. Thus for every pixel being rasterized we need to select the corresponding shadow map. In the original PSSM paper Zhang et al. [ZSXL06] choose the shadow map based on the eye-space depth of the current fragment. This is a very simple and robust solution to this problem. However, if the light direction is almost parallel to the view direction, the selected shadow map might not be optimal for the current point. A schematic illustration of such a case is shown in Figure 3.13: The point P is covered by all three shadow maps. The optimal choice for this point would be the first shadow map (Split 0) since it provides the highest resolution. However, based on the eye-space depth we would select the third shadow map (Split 2) with the lowest resolution.

To avoid this problem Zhang et al. [ZZB09] proposes a method which always chooses the shadow map with the best resolution for the current point. Although this method requires a few more shader instructions, it can lead to much better results especially for light directions from behind. A comparison of both methods can be seen in Figure 3.14.

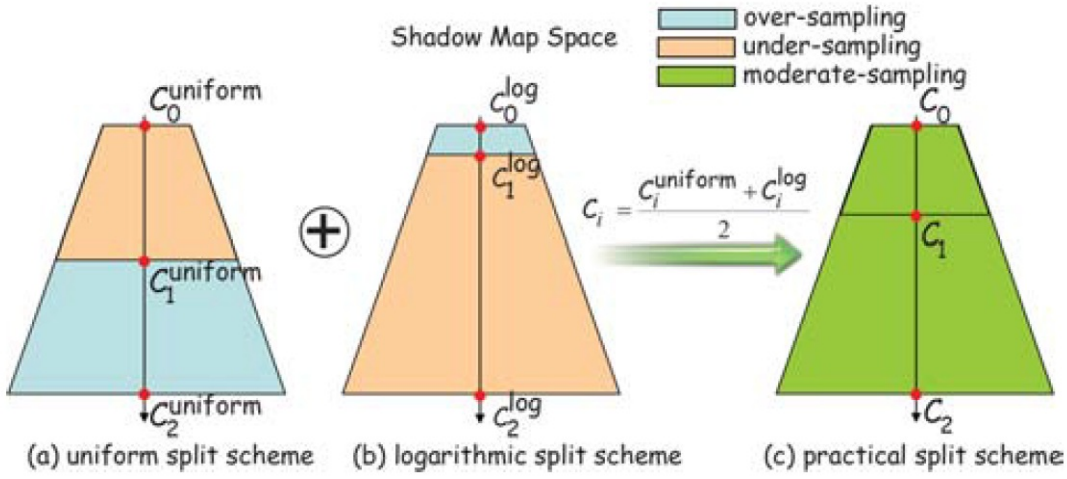


Figure 3.12: Comparison of split schemes for z-partitioning with two split parts. Image courtesy of Zhang et al. [ZSXL06]

3.3.4 Error analysis

The error analysis of z-partitioning can be done in the same way as for the single split techniques. In case of partitioned uniform shadow maps we evaluate the equations 2.7 and 2.8 analog to single split uniform shadow maps for every sample along the length of the view frustum. Figure 3.15 shows the error distribution of PSSM [ZSXL06] ($\lambda = 0.5$) using 2, 4, 8 and 16 split parts. It can be seen that the overall maximum error still occurs at the near plane distance and decreases about half the amount when the number of splits gets doubled. We can also see that the characteristic of each split region corresponds to uniform shadow mapping based on the far and near plane ratio of the current split frustum.

3.3.5 Storage strategy

In the original PSSM [ZSXL06] respectively CSM [Eng06] implementation, there are no restrictions regarding the storage strategy of the split shadow maps. There exist basically three possibilities. The simplest approach is to store the shadow maps separately [ZSXL06] [Eng06]. According to Zhang et al. [ZZB09] this method is not very suitable to the implementation of filtering techniques and can also cause problems in applications that use a large number of textures simultaneously.

On modern GPUs the usage of hardware supported texture arrays and cube maps has become more and more common practice to store the split shadow maps [ZSN07]. Similar to separately stored shadow maps one great benefit of these methods is that the size of each shadow map is only limited by the maximum supported texture size of the graphics hardware. Though hardware

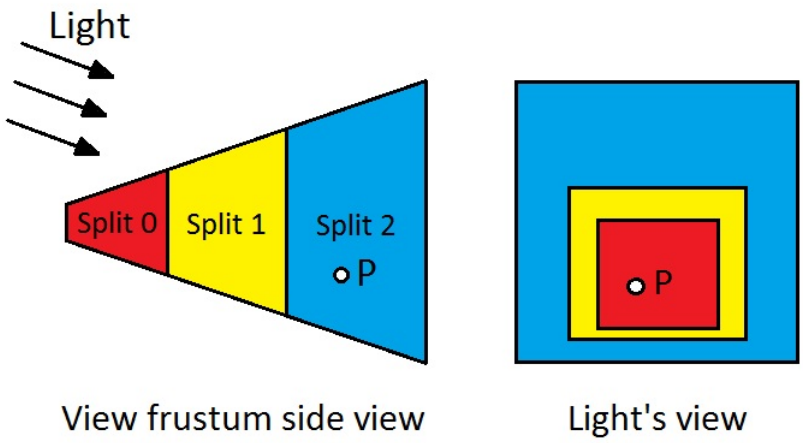
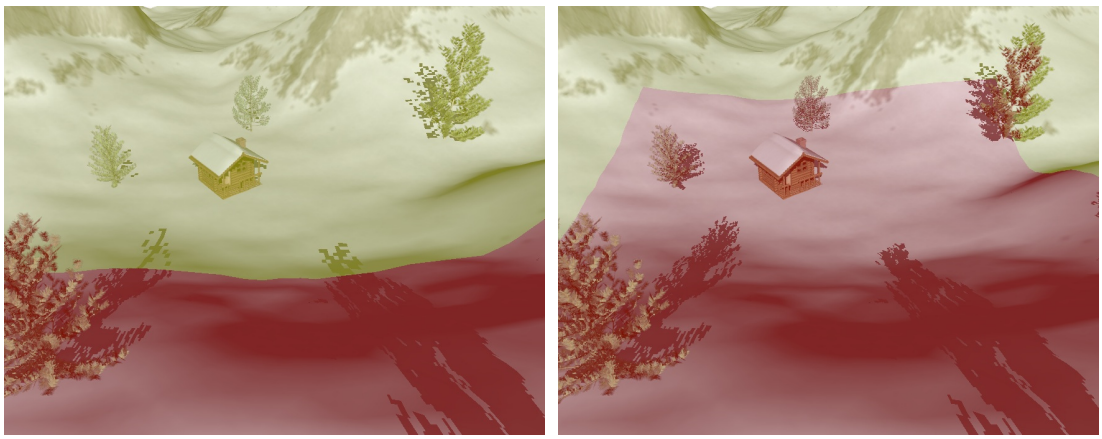


Figure 3.13: Split selection: point P is covered by all shadow maps. Instead of selecting split 2 based on the eye-space depth, the best choice is split 0, since its shadow map provides the best resolution.



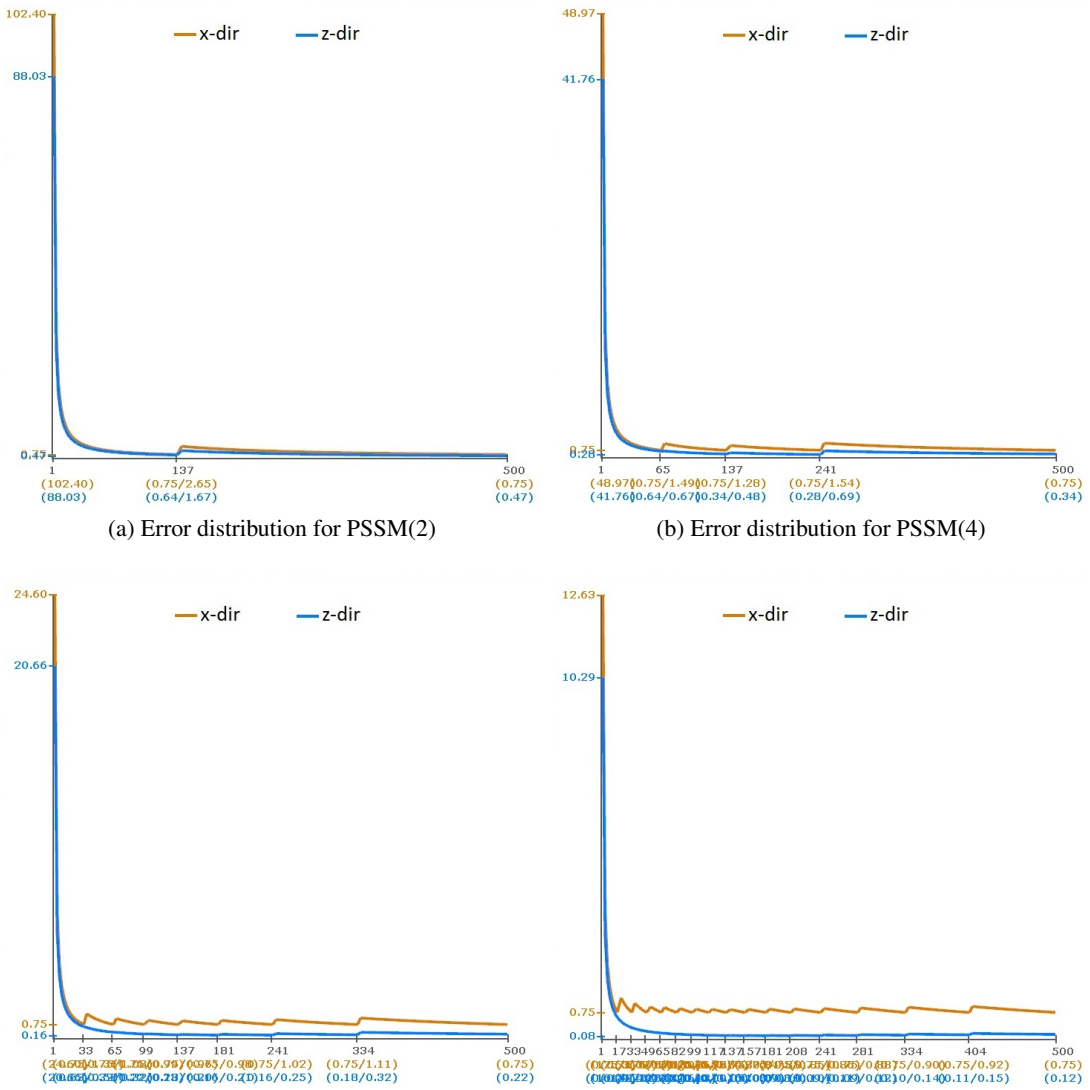
(a) Split selection based on eye-space depth

(b) Split selection based on the best resolution

Figure 3.14: Illustration of split selection based on the eye-space depth (a) and split selection based on the shadow map with the best resolution (b). Red shaded: first split. Yellow shaded: second split.

provided sampling and percentage closer filtering is only supported by Direct3D 10.1 or above. Furthermore to avoid any waste of texture memory the usage of cube maps only makes sense for a number of six splits and requires some additional shader instructions for the texture look up.

The third technique of storing the shadow maps was proposed by Lloyd et al. [LTYM06] and uses a large texture atlas, where all shadow maps are packed into. The only drawback of this



(a) Error distribution for PSSM(2)

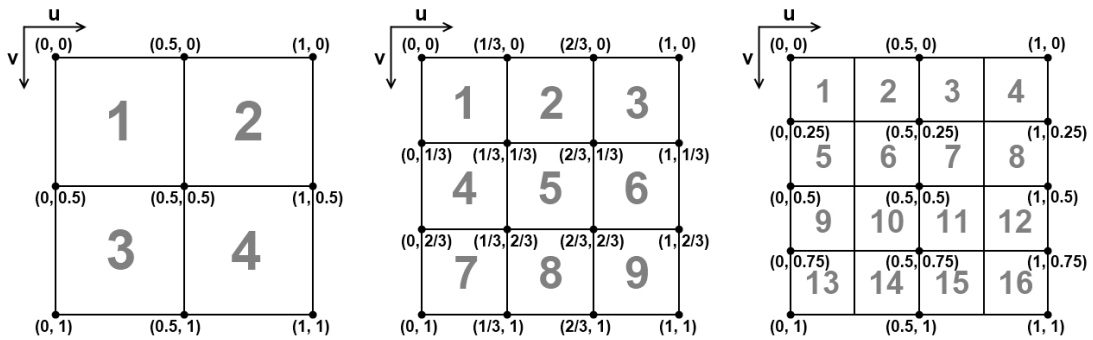
(b) Error distribution for PSSM(4)

(c) Error distribution for PSSM(8)

(d) Error distribution for PSSM(16)

Figure 3.15: Comparison of perspective aliasing error distribution for PSSM [ZSXL06] with an overhead light source using various numbers of splits. Parameters: $\Theta = 30°$.

technique is the imposed restriction of the single shadow map size based upon the chosen atlas partitioning scheme and the maximum allowed texture size of the graphics hardware. Though, according to Zhang et al. [ZZB09] this method comes with some practical advantages which have increased its amount of usage in modern games recently. Dependent on the chosen partitioning scheme for the texture atlas and actual number of splits, the video memory can be used very efficiently. In this context a very common variant is to use 4 splits that are arranged at the 4 corners of a quad (see Figure 3.16a). Especially for very complex shading systems which are



(a) Configuration of the texture atlas where the optimal video memory consumption occurs at the maximum of 4 splits.

(b) Configuration of the texture atlas where the optimal video memory consumption occurs at the maximum of 9 splits.

(c) Configuration of the texture atlas where the optimal video memory consumption occurs at the maximum of 16 splits.

Figure 3.16: Illustration of our 3 implemented atlas strategies including the actual split orders and the corresponding texture coordinates. We limited the maximum split shadow map size to 2048 x 2048, so that maximum texture size allowed by the graphics hardware cannot be exceeded by using 16 splits.

using many texture samplers for other purposes, it can be quite useful that for a texture atlas only one single two-dimensional texture sampler is required. Another great advantage of this technique comes with the fact that it works almost equally on any common graphics hardware and similar for different graphics APIs, which helps developers, in these times of various available GPUs supporting different versions of OpenGL or DirectX, to support as many graphic cards as possible. Since the usage of soft shadow algorithms have become more and more popular, a further advantage of the texture atlas strategy comes with the opportunity that almost any filtering technique can be applied in one rendering pass though some additional shader instructions might be required for special treatments at the split boundaries.

For our framework we have decided to go with the atlas strategy and pack all splits into a single large texture atlas. To offer the user the most possible flexibility without wasting too much video memory and to limit the degradation of usability, we implemented three different partitioning schemes for an actual maximum of 4, 9 or 16 splits and restricted the shadow map resolution for a single split to a maximum of 2048 x 2048 pixels (see Figure 3.16).

3.3.6 Minimizing the number of rendering passes

As mentioned before, one of the main disadvantages of PSSM [ZSXL06] or CSM [Eng06] compared to single split warping algorithms is the requirement of the additional 3D rendering passes to generate the shadow map for each split frustum. DirectX 9 respectively OpenGL 2 GPUs at least feature the opportunity to generate the final shadowed image in one pass, so that for z-partitioning with n splits we need $n+1$ 3D rendering passes - n passes to generate the shadow

maps and 1 pass to project the shadows into the scene. By disabling any kind of shading operations and using hardware depth maps, which were especially introduced for shadow mapping purposes, the performance loss could be minimized. But a certain lack of rendering speed compared to single split warped shadow mapping methods was still present.

After the introduction of DirectX 10 respectively OpenGL 3 graphics hardware Zhang et al. [ZSN07] presented a method called *geometry cloning* to generate the shadow maps for all split frusta in one rendering pass by using the new *geometry shader* stage. Their basic idea is to generate all split frusta and the split-specific focusing matrices as usual and send the complete scene geometry in one rendering pass to the graphics hardware to render all split shadow maps simultaneously into *multiple render targets*.

When all scene objects have to be rendered in one pass, it is no longer possible to perform common *frustum culling* between the single splits. Therefore, to minimize the amount of geometry computations on the GPU, Zhang et al. [ZSN07] performs before the actual shadow map rendering pass an additional pass on the CPU that determines the object respectively shadow caster to split relation based on the bounding boxes of the scene objects. This ensures that objects are only processed for splits where they are lying in or are at least partly visible from the light's point of view.

During the first stage of the actual rendering pass the vertex shader applies the common (not split-specific focused) light's view-projection matrix to the vertices. In the next step the geometry shader generates from the incoming triangles for each split the corresponding triangles by application of the split-specific transformations and setting the correct render target index. The computations in the final pixel shader stage remain the same as for standard z-partitioning methods. In Listing A.1 we show a sample geometry shader program for geometry cloning.

According to Zhang et al. [ZSN07] the great advantage of geometry cloning is that, since each scene object respectively shadow caster is sent only once to the graphics card it comes with a large amount of API overhead (e.g. extra draw calls) reduction. Though for scenes with a large geometry amount they recommend the usage of their introduced method with *instanced drawing* [ZSN07] since processing large amounts of data with the geometry shader can be very expensive which we also noticed in our experiments.

3.4 Combinations

We have mentioned before that most of our used shadow mapping algorithms can be easily combined with each other. According to Lloyd et al. [LTYM06] the combination of warping with z-partitioning works fine and usually leads to great improvements of the shadow quality. However there are some aspects that need to be considered when different algorithms are applied simultaneously.

The calculation of the intersection bodies B for each split works pretty straight forward ana-

log to focusing of a single split configuration. To cover only the visible parts of the scene, first of all the near and far plane of the entire view frustum need to be adjusted to the scene as explained in Section 3.3.1. After the following computation of the split distances according to the current split scheme we construct the split frustum for the current split. Now B can be calculated in the same way as discussed in Section 3.1.1 while using the current split frustum instead of the entire view frustum.

In the next step we compute the perspective warping frustum P for the current split based upon the previously calculated intersection body. As mentioned in Section 3.2.1, we cannot compute the projection center in the same way as discussed in this section. Though this method works for the first split, in most cases it produces bad results for the subsequent splits. Usually the required nearest vertex to the eye e on the intersection body LVS is given by one of the corner points on the near plane of the current split frustum. Since the near plane distances of the distant split frusta are normally very high, the distances of the corner points from the center of the near plane are very high as well. Therefore, computation of C_{start} and the projection center C in the way discussed in Section 3.2.1, usually leads to an asymmetric warping frustum, even for directional overhead light source. For this reason we propose an alternative way to compute the projection center in Chapter 4 which solves this problem in most cases and guarantees a symmetric warping frustum for a common overhead light source. The basic operations to compute P work analog to the steps explained in Section 3.2 whether we are using common LiSPSM or re-parametrized LiSPSM. If we are using arbitrary warping which was added for experimental purposes it is possible to adjust the warping strength for each split independently or based upon the variable first split according to the currently used split scheme.

Once we have applied the LiSP projection matrix for the current split we adjust the resulting light projection to the intersection body B as explained in Section 3.1.2.

A sample configuration of LiSPSM using n_{opt} combined with z-partitioning using the split scheme of PSSM [ZSXL06] is shown in Figure 3.17.

3.4.1 Error analysis

By the combination of warping and z-partitioning and using the equations 2.3 and 2.4 to evaluate the errors, we can see that the overall perspective aliasing error can be further reduced and we are able to minimize the influence of the drawbacks of each technique without any additional costs. This means warping avoids the high error at the near plane distance and z-partitioning minimizes the error maximum at the duelling frusta problem.

For a common split scheme that consists of a uniform and a logarithmic term like for example PSSM ($\lambda = 0.5$) (see also Section 3.3.2) we have noticed a high perspective error at the near plane in case of z-partitioning with uniform shadow mapping. This high error mainly results from the large near plane to far plane ratio of the first split and usually leads to a very strong warped first split and a relatively large discontinuity at the split border between the first and the second split (see Figure 3.18a). Our experiments have shown that the minimal perspective error

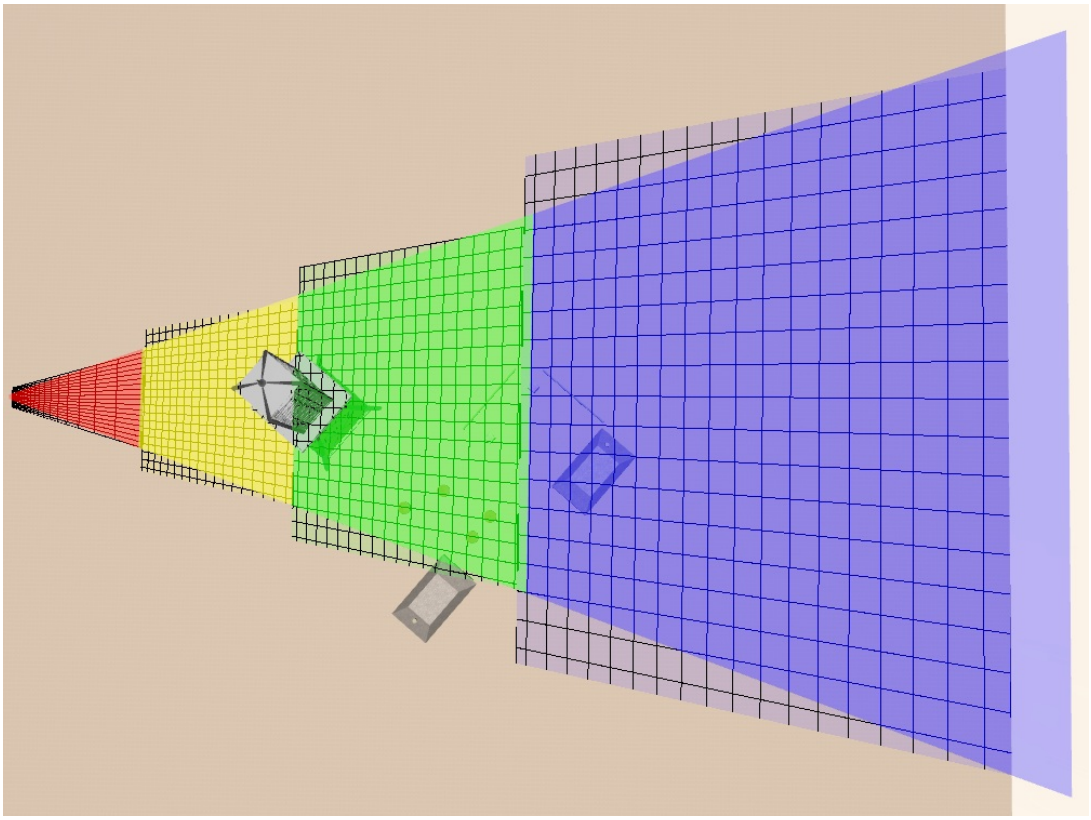


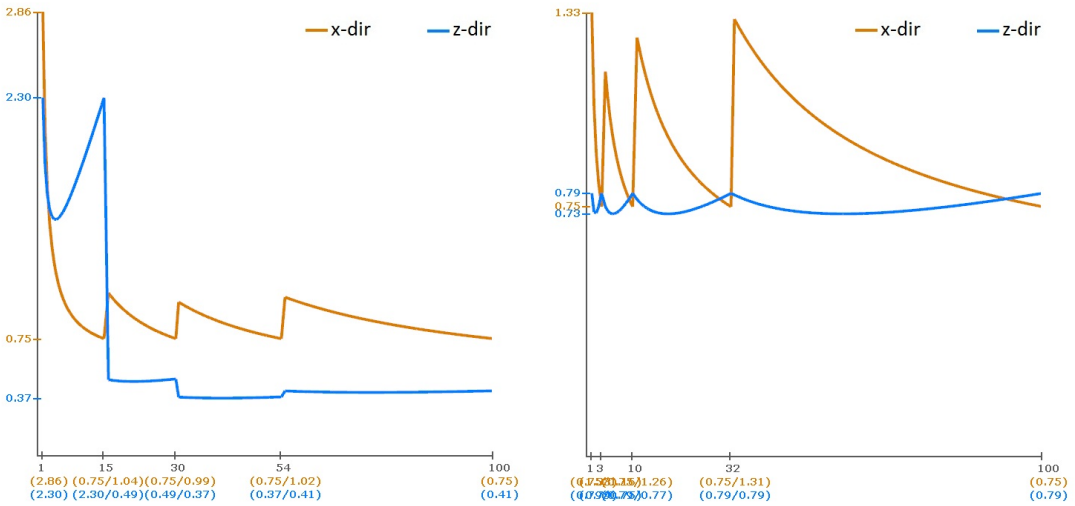
Figure 3.17: Visualization of warping combined with z-partitioning. The warping algorithm corresponds to LiSPSM with n_{opt} and the used split scheme for z-partitioning corresponds to PSSM ($\lambda = 0.5$).

occurs at warped z-partitioning using the logarithmic split scheme (see Equation 3.7). Figure 3.18b shows the resulting perspective error distribution at this configuration using 4 split frusta.

Lloyd [LTYM06] also discusses various combinations of shadow mapping techniques including face partitioning [Koz04] which is not considered in this thesis. He defines a so called *storage factor* $\bar{S}$ as a useful aggregate measure of the errors in x- and z-direction independent of the shadow map and image resolution and analyzes the discussed shadow mapping techniques regarding $\bar{S}$. According to his results this total error $\bar{S}$ results in a minimum for the combined application of warping and z-partitioning (see Figure 3.19) considering an overhead light source. For further results of his error analysis we refer to [LTYM06].

3.5 Reducing shadow flickering

As mentioned in Section 3.1 every shadow mapping algorithm that uses a focused shadow map or somehow changes the orientation of the light-space according to the eye-space (e.g. LiSPSM)



(a) PSSM(4) + warping

(b) Log. z-partitioning + warping

Figure 3.18: Error comparison of different combinations of z-partitioning and warping for an overhead light source. (a) LiSPSM combined with PSSM(4): Notice the unwanted relatively large discontinuity in the distribution of z-direction at the split border between the first and the second split. (b) LiSPSM combined with log. z-partitioning: Especially the error in z-direction is almost uniform distributed along the view frustum. Parameters: $\Theta = 30°$, $\gamma = 90°$.

imposes *flickering* or *swimming* shadow boundaries. This problem is caused by different rasterizations of the shadow map for consecutive frames, when the camera is moving (see Figure 3.20). Since we recompute the shadow map every frame and use an orthographic projection matrix to translate and scale the light projection according to the intersection body, the flickering problem can be mathematically disassembled according to Zhang et al. [ZZB09] into an *offset* problem and a *scale* problem. The offset problem results from different quantization of specific fragment positions across consecutive frames. The scale problem is caused by the changes of the texel size. Considering these observations it is obvious that we need to stabilize the quantization of the positions and the texel size relative to the intersection body respectively the view frustum.

A simple and effective approach to avoid the flickering problem is to apply filtering methods like for example PCF [RSC87]. Though filtering does not actually resolve the problem but hides it, in many cases it is sufficient and produces good results.

Zhang et al. [ZZB09] describes two actual solutions to this problem. The so called *exact solution* focuses the shadow map on the view frustum's respectively split frustum's bounding sphere rather than the intersection body's bounding box. Because of the symmetric characteristic of the sphere, this method stabilizes the world-space size of the shadow map and completely removes the scale problem. The offset problem is resolved by adjusting the light's position in world-space according to the sphere's center. Since at this solution the world-space diameter of

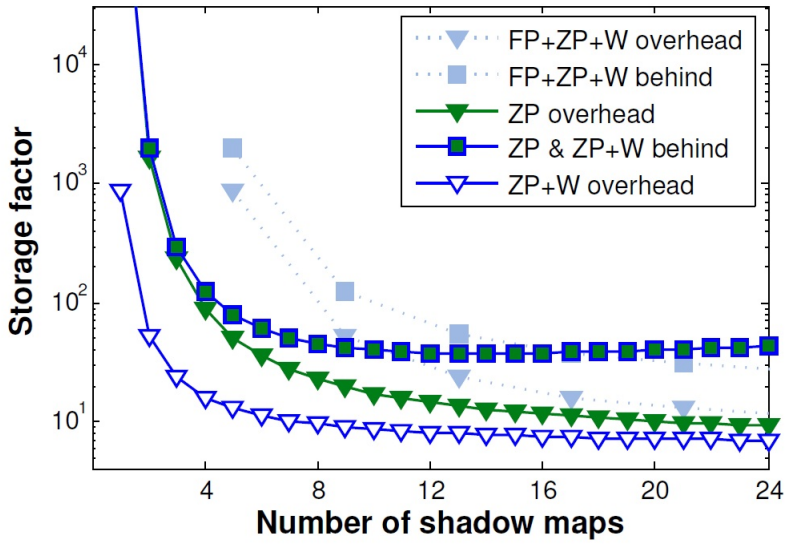


Figure 3.19: The storage factor of different combinations of warping (W), z-partitioning (ZP) and face partitioning (FP) for varying number of shadow maps. Parameters: far plane to near plane ratio $f/n = 1000$, $\Theta = 30°$. Image courtesy of Lloyd [LTYM06].

the bounding sphere needs to remain constant, it is obvious that this technique is only applicable in combination with uniform z-partitioning without any intersection body focusing and view frustum adjustments. Considering these observations in addition to the fact that a sphere is not the best approximation of a split frustum, it can be seen that the big disadvantage of this exact solution is that a large amount of shadow map resolution is wasted.

The second solution to the flickering issue, proposed by Zhang et al. [ZZB09], is the so called *approximated solution*. Its basic idea is to discretize the scaling of the focusing step into a predefined range of discrete levels under the assumption that the scale value varies smoothly across consecutive frames. According to Zhang et al. [ZZB09] we use their empirically found number of 64 levels. The offset problem can be solved similar to the exact solution by adjusting the light's position in world space so that the center of the shadow map contains the center of the intersection body's bounding box.

The approximated solution works much better than the exact solution since it does not waste as much shadow map resolution and it is still possible to combine it with focused uniform shadow mapping as well as with focused uniform z-partitioning, that uses an adjusted view frustum. A small drawback is that some minor swimming can still occur.

Both of the above discussed techniques cannot be used with any warping algorithm because of the non-uniform texel distribution. A possible solution to elimination of flickering for warping algorithms by using temporal coherence across several consecutive frames was introduced

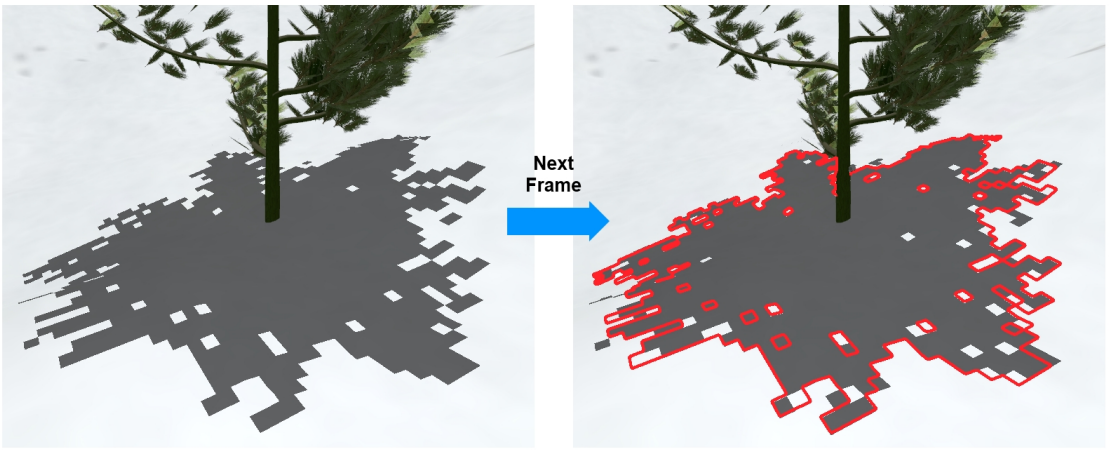


Figure 3.20: Illustration of the flickering problem. The red lines in the right image denote the shadow boundaries of the previous frame, shown on the left side.

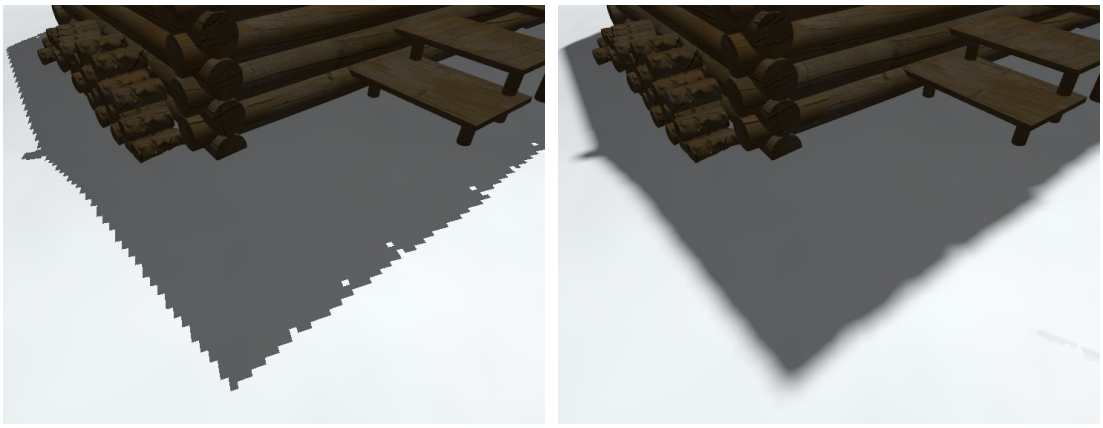
by Scherzer et al. [SJW07].

3.6 Filtering

Soft shadows in games have become more and more popular recently. In the past few years various new soft shadow algorithms (e.g. PCSS [Fer05]) and filtering techniques (e.g. VSM [DL06]) have been introduced. Since the focus of this thesis lies on improving the shadow quality of hard shadow maps and reducing the perspective aliasing, the implementation of soft shadows played a minor role at our work. Though filtering still can be used to hide the jagged shadow boundaries respectively the typical staircase artifacts caused by undersampling nearby the camera, which is why we experimented with a PCF [RSC87] based approach to improve our results.

Modern graphics hardware already provides 2x2 bilinear PCF sampling for common depth maps which however is in most cases not sufficient to hide undersampling artifacts efficiently. Yet, it can be easily used to implement extended PCF based filtering techniques.

According to Fernando's PCSS [Fer05] we implemented an extended PCF based method that uses hardware PCF samples which are distributed on a *Poisson-disk* (see Figure 3.21). For further details on Poisson-disks and their generation we refer to Dunbar and Humphreys [DH06]. Since PCF sampling is only required in shadowed areas, Fernando [Fer05] proposed a technique to reduce the amount PCF samples in areas where no shadows have to be cast. Before the actual filtering process he applies a so called *blocker search* to find all shadow casters in the area of the current. This blocker search works similar to the actual filtering method, except it can be performed using less samples and instead of computing an average shadow value it sums up all shadow casters which are found during the sampling process. If the blocker search results in no



(a) Unfiltered shadows suffering from the typical staircase artifacts.

(b) PCF filtered shadows showing hardly any staircase artifacts.

Figure 3.21: Comparison of unfiltered shadows (a) and PCF filtered shadows using 64 Poisson distributed hardware PCF samples (b). Notice that the staircase artifacts have been almost completely disappeared in the right image.

found shadow casters, the filtering is skipped. Dependent on the occurring shadows in the current image to be rendered, the application of the blocker search increased the frame rate during our experiments up to 15 percent compared to PCF filtering without the prior blocker search.
In addition to the computation of the number of blockers, Fernando [Fer05] computes during the blocker search the average depth value of all found shadow casters for the further calculations of the variable penumbra size of PCSS.

3.6.1 Kernel size

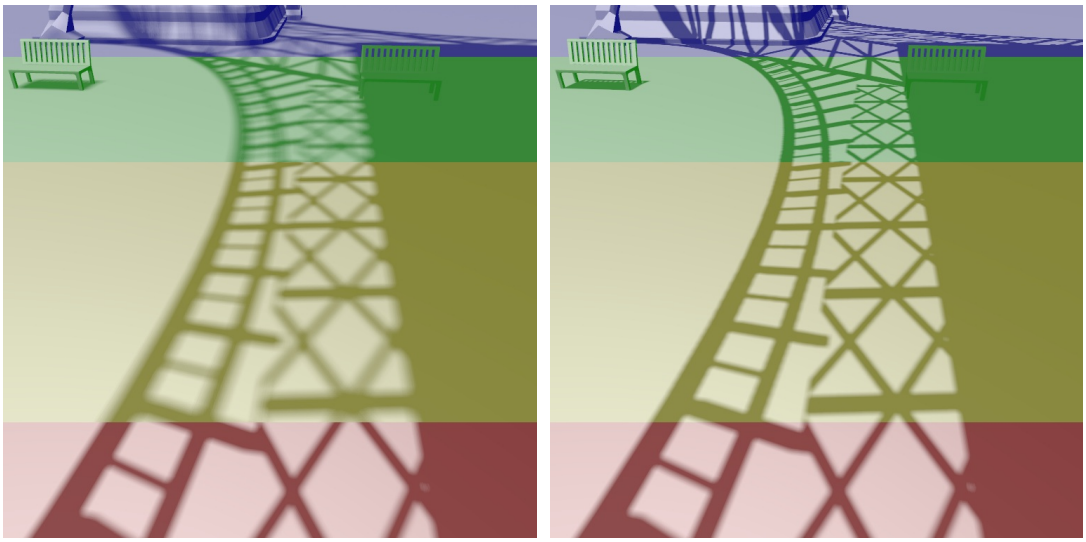
To offer the opportunity to experiment with the softness of the PCF filtered shadows we added a variable parameter r_l that can be modified during the run-time. It controls the basic kernel size of our filtering method which can also be referred to as the diameter of a fictional area light source. Note that the resulting soft shadows of PCF filtered shadow maps are not physically correct.

As discussed previously in this chapter almost any of our implemented shadow mapping algorithms uses focusing on an intersection body. Normally, this involves a scaling of the world-space kernel size of the filter according to the current scale operation of the focusing matrix. For common single split uniform shadow mapping usually this is not a big problem, though it imposes view-dependent changing of the shadow softness. However in case of z-partitioning and warping methods this issue can cause ugly artifacts. Since the single splits of z-partitioned shadow mapping methods are usually differently scaled, the world-space filter kernel size is scaled in the same way which leads to abrupt changes of the shadow softness at the split borders

(see Figure 3.22a). A very simple solution to eliminate the view-dependence of the kernel size for common single split uniform shadow mapping is to scale the kernel size according to the focusing scale transformation. Since our shadow maps are organized in a large texture atlas for z-partitioned shadow mapping we need to apply an additional factor l_{sc} to each split which is based on the currently used atlas partitioning scheme (see Equation 3.9).

$$\begin{aligned} \mathbf{s}_i &= r_l \; l_{sc} \begin{pmatrix} \frac{2}{max_x - min_x} \\ \frac{2}{max_y - min_y} \end{pmatrix} \\ l_{sc} &= \begin{cases} 1/2 & 1 < m \leq 4 \\ 1/3 & 4 < m \leq 9 \\ 1/4 & 9 < m \leq 16 \end{cases} \end{aligned} \tag{3.9}$$

The values min and max correspond to the boundaries of the intersection body bounding box found during the focusing step for the current split i (compare to Equation 3.1) and m stands for the currently used number of splits.



(a) PCF filtered shadows without any split-specific adaptation of the kernel size.

(b) PCF filtered shadows with the proposed adjustments of the kernel size for each split.

Figure 3.22: Comparison of PCF filtered shadows with (b) and without (a) split-specific adjustments of the filter kernel size. Note the constant shadow softness in (b) over all split areas.

The previous discussed methods are hardly applicable to warped shadow mapping techniques because of the non-uniform distributed samples in the shadow map. Especially for very strong warped shadow maps a small kernel size nearby the observer can lead to a very large kernel size distant from the camera. A good workaround to reduce this problem is to minimize the

warping effect to a reasonable level or combine it with z-partitioning and treat it like uniform z-partitioning.

3.7 Summary

In this chapter we have reviewed and analyzed several advanced shadow mapping techniques to produce hard shadows in large scaled virtual environments. We focused on fully hardware-accelerated techniques that deal with minimizing the perspective aliasing error.

Considering different approaches of improving the shadow quality we laid the focus of our analysis on three basis techniques:

- Fitting the shadow map on the scene objects that are relevant for shadow mapping based on the current view frustum configuration.
- LiSPSM which applies a perspective warping frustum to redistribute the shadow map samples according to the current view frustum configuration.
- Z-partitioning which splits the current view frustum along its length into several split frusta and applies for each split part a single shadow map.

Afterwards we discussed the opportunities of combining the different techniques for further shadow quality improvements and avoiding some of the drawbacks of each single method.

Then we dealt with some methods to reduce the imposed flickering artifacts of the used focused shadow mapping techniques to improve the robustness of the final shadows.

At the end of the chapter we briefly discussed PCF as technique for hiding the aliasing artifacts by filtering of the projected shadows.

CHAPTER 4

Optimizations

As mentioned before some steps of the previous discussed shadow mapping algorithms can cause minor problems, if we combine them with other approaches or visualize the interesting aspects of the shadow map creation process. In this chapter we will introduce some optimizations for the discussed shadow mapping techniques, which solve these problems and improve the shadow quality respectively enhance the quality of our applied visualizations.

Additionally, we will discuss an approach to adjust the view-frustum near plane, using the capabilities of the graphics hardware, to further reduce perspective aliasing.

4.1 Projection center for warping

We have explained in the previous chapter, that Wimmer and Scherzer's [WS06] approach to define the projection center C of the perspective warping frustum for LiSPSM involves some robustness issues and usually leads to asymmetric warping frustums for the second and the subsequent splits in case of warped z-partitioning.

Our basic idea to avoid the erratic changes of C caused by the discontinuous changes of C_{start} respectively e is to define C_{start} by using a point that lies on the view vector of the eye. As mentioned in Section 3.2, e corresponds to the nearest vertex to the eye on the body LVS, which denotes the intersection of light frustum (in case of a point light), view-frustum and scene bounding box. Considering the given vertex e, we assume that a point s, lying on the view vector, has the same distance as e to the camera along the current view vector and still lies on LVS. Based on this assumption we define C_{start} as $(s_{ls_x},\ s_{ls_y},\ B_{z_{near}})^T$. Since s always lies on the camera's view vector, its position changes continuously according to the camera movements as well as the position of C_{start}. After the computation of C_{start} we obtain the projection center C as explained before by translating the distance of the free parameter back from the near plane. The new way to define C_{start} also causes the projection center to lie in the center of the cur-

rent split frustum near plane in most cases. Figure 4.1 illustrates the definition of the projection center.

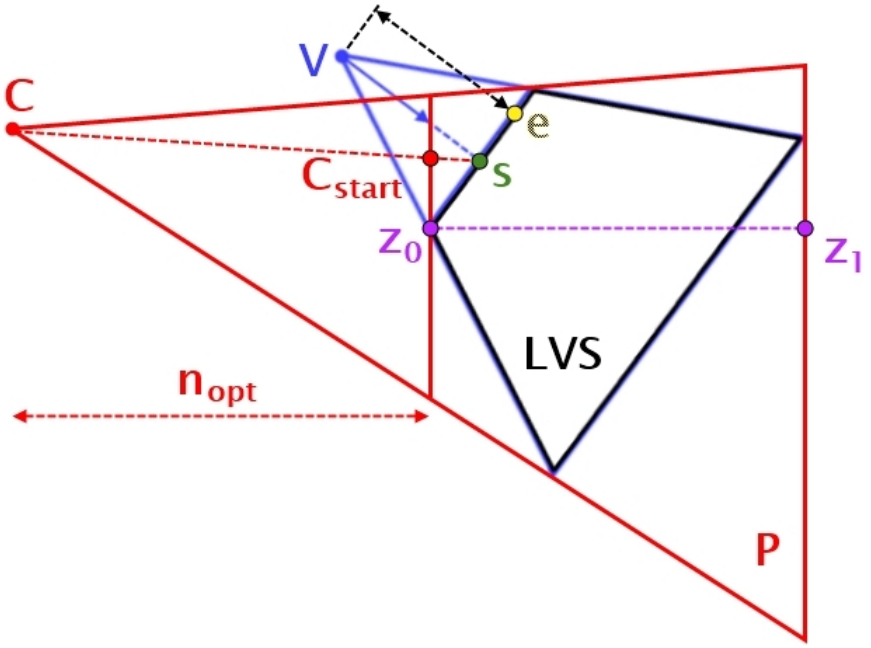


Figure 4.1: Construction of C_{start} and C in light-space from the points e and s on the plane nearest to the camera.

4.2 Pseudo-near plane and pseudo-far plane

One perception in common virtual environments is that the area nearby the viewer is empty in most cases. We have seen that perspective aliasing has a maximum in this region, mainly caused by the far plane distance to near plane distance ratio. Considering this circumstance, the best thing to do would be to increase the near plane distance. This normally comes with a great reduction of the perspective aliasing error. However, increasing the near plane distance too much can lead to near plane clipping of geometry against the view frustum. Thus, in common applications the near plane distance is set to a conservatively low value to avoid near plane clipping, while accepting the higher error.

Lloyd [Llo07] proposes an alternative approach to the previously discussed z-partitioning scheme as well as an alternative warping algorithm. By using a *pseudo-near plane*, this new approach allows the user to reduce perspective aliasing and still keep the near plane distance conservatively low. According to Lloyd [Llo07] the only price to be paid is the possible risk of a high error between the actual near plane distance n_e and the pseudo-near plane distance n_p.

In addition to Lloyd's considerations, we experimented with the new approach of using a *pseudo-far plane*, which is applied additionally to the pseudo-near plane to further reduce perspective aliasing. It can be used in an analog way as the pseudo-near plane and involves the similar possible risk of a high error between the pseudo-far plane distance f_p and the actual far plane distance f_e.

For z-partitioning, combination of the pseudo-near plane with the approach of the pseudo-far plane leads to a new split scheme shown in Equation 4.1.

$$C_i = (1-\lambda)\left(n_p + (f_p - n_p)\,\frac{i}{m}\right) + \lambda\left(n_p\,(f_p/n_p)^{i/m}\right) \quad \lambda \in [0,1]. \tag{4.1}$$

While Lloyd's [Llo07] formulation is based on a logarithmic split scheme, we compute the new split distances by replacing the actual near plane and far plane in Zhang's et al. [ZSXL06] practical split scheme through the values of the pseudo-near and pseudo-far plane.

To define a pseudo-near warping algorithm Lloyd [Llo07] uses the same way as Wimmer et al. [WSP04] for LiSPSM by solving for a free parameter n for the perspective warping frustum where the error at n_p is equal to the error at f. By adding our experimental approach of the pseudo-far plane and conversion of Wimmer and Scherzer's [WS06] revised formulation of LiSPSM we get the new formulation of n_{opt}:

$$n_{opt} = d\,\frac{z_0}{z_1 - z_0} + \sqrt{(z_0 + n_p\,(z_1 - z_0))\,(z_1 - f_p\,(z_1 - z_0))} \quad n_p, f_p \in [0,1]. \tag{4.2}$$

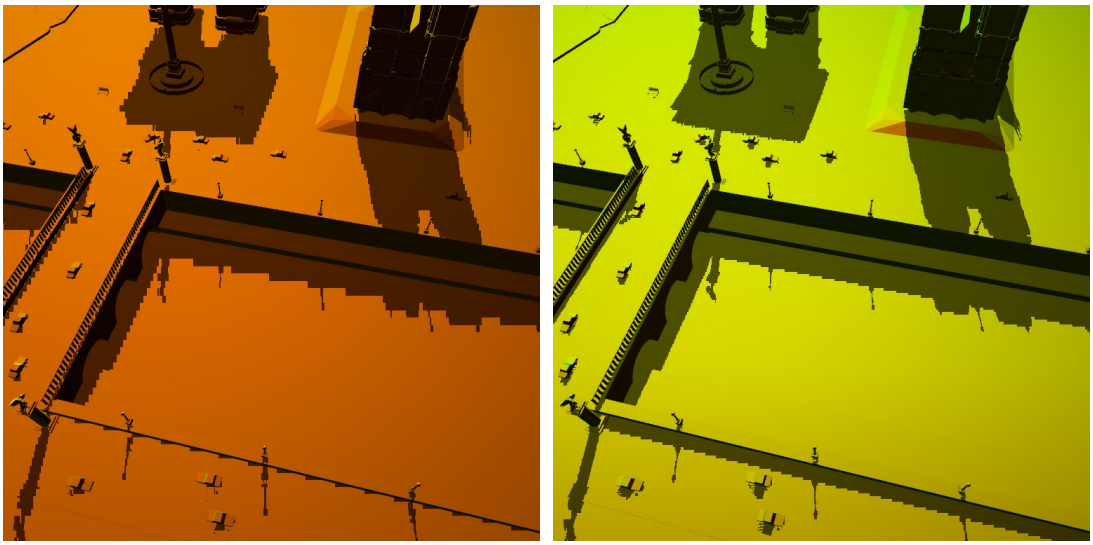
Our experiments have shown that this technique works fine for all possible values of n_p and f_p, and there is no need to distinguish between the two cases $n_p < 2/3$ and $n_p \geq 2/3$ as proposed by Lloyd [Llo07].

Note that for the calculation of the intersection body B, we are still using the actual near plane distance n_e and far plane distance f_e which means the entire view frustum respectively all visible objects are covered by the shadow map. In case of LiSPSM combined with z-partitioning, n_p is only considered for the calculation of the warping frustum of the first split and f_p is only used for warping of the last split. In Figure 4.2 we show a comparison of n_{opt} LiSPSM with and without a pseudo-near and pseudo-far plane.

4.3 Maximizing the near plane distance

In Section 4.2 we have mentioned Lloyd's [Llo07] approach to reduce perspective aliasing by introducing a pseudo near plane, since increasing the camera's view frustum near plane distance usually comes with a large decrease of the perspective aliasing error.

In this section we will discuss the possibilities of increasing the near plane distance itself, since the perception of an empty area nearby the viewer is valid in most applications.



(a) Common n_{opt} LiSPSM

(b) n_{opt} LiSPSM with pseudo-near plane and pseudo-far plane

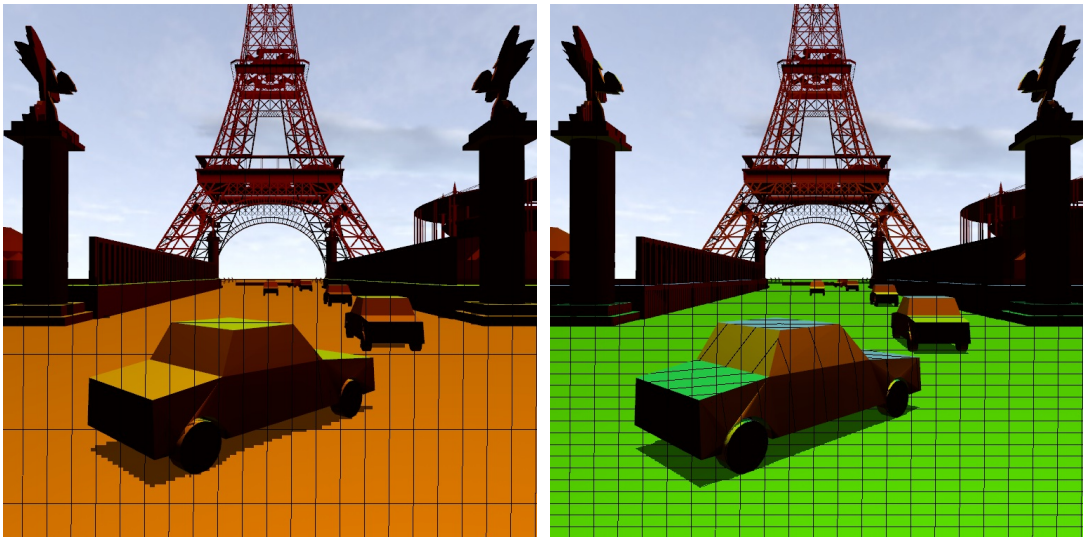
Figure 4.2: Comparison of common n_{opt} LiSPSM with (b) and without (a) the application of a pseudo-near plane and a pseudo-far plane. Both figures include a visualization of the sampling rate via color overlay. Parameters: $\gamma = 90°$, $n_e \approx 1$, $f_e \approx 300$, $n_p \approx 30$ (in eye-space), $f_p \approx 30$ (in eye-space). For the corresponding color mapping of the sampling rate visualization see Figure 5.6.

Obviously the simplest way is to adjust the near plane manually during the runtime. Thus we realized this opportunity in our framework by a slider element which is scaled in a way that the near plane can be varied between a minimum and the current to the scene bounding box adjusted far plane (while focusing is activated). If the camera is located inside the scene bounding box the minimum is set to a fixed conservatively low chosen value, otherwise it results from the adjusted near plane distance computed during the calculation of the convex intersection body. Since we usually do not have any information about the actual location of the scene objects in relation to the camera, which imposes possible near plane clipping for high near plane distances, this method makes only sense for experimental purposes.

Stamminger and Drettakis [SD02] mentioned a new method to adjust the near plane by reading back the depth buffer after each frame. Following this approach, we introduce a method to automatically adjust the near plane distance according to the actual relation between the camera and the nearby scene objects. Similar to Stamminger and Drettakis [SD02], we perform an additional rendering pass to determine all depth values for the current camera position and store them into a linear depth map, before the calculation of the convex intersection body. Afterwards we use the obtained depth map as input for a manually created *mip map chain*. During the computa-

tion of this mip map chain, from the largest to the smallest image, we evaluate for each group of four adjacent fragments the minimal depth (see Listing A.2 for a sample fragment shader source code). Finally we can read back the minimal depth value of the current frame from the smallest tile. This obtained minimal depth can be directly used as new maximal near plane distance for computation of the intersection body to minimize perspective aliasing without any risk of near plane clipping occurrence.

Our experiments have shown that this technique to determine the minimal depth of the current location can result especially for PSM in great improvements of the shadow quality and large reductions of perspective aliasing. Figure 4.3 shows a comparison of PSM with and without the automatic adaptation of the near plane distance including a visualization of the sampling rate and the texel borders.



(a) PSM with conservatively low chosen near plane distance $z_n = 1$

(b) PSM with automatic established near plane distance $z_n \approx 6$

Figure 4.3: Comparison of PSM with conservatively low chosen near plane distance (a) and automatic adjusted near plane distance based upon the current camera to scene location (b) including a visualization of the texel borders and the sampling rate. For the corresponding color mapping see Figure 5.6.

Since this technique requires several additional rendering passes it involves a certain amount of performance loss regarding the real-time capabilities. During our experiments we have seen that the highest costs are involved with the read back of the smallest image of the mip map chain from the graphics hardware to the CPU, which is actually a well known fact in graphics development. The 3D rendering pass to obtain the depth map and the computation of the mip map chain do not involve extensive additional performance degradation. Our analysis of the frame

rate for this technique combined with common PSSM(4) resulted in a minimum of 30 frames per second for each of our test scenes.

CHAPTER 5

Framework

In order to compare and analyze the previous discussed shadow mapping algorithms, we developed a framework within the work on this thesis. This framework features all of the techniques, discussed in the chapters 3 and 4, except for the blocker search of PCF, which we explained in Section 3.6. It the allows us to apply the different algorithms on various scenes and compare the results with respect to performance, robustness and other quality properties. It also provides many opportunities to visualize different aspects of the shadow map creation process. In the following sections we establish the purpose of our framework, give an overview of the concepts, describe the various available features and visualizations, and show some examples.

5.1 Concepts

Most existing shadow mapping systems only feature a few methods to visualize and analyze the different aspects of the shadow mapping process. Moreover, they are often focused on only one specific shadow mapping technique. Because of the large number of different algorithms addressing the same problem like for example perspective aliasing, in most cases it is hard to find the best suitable method based on the used scenario. Even if a system supports several techniques, there is still a need to compare the final results of the different methods in one way or another. Additionally to the rendering of the ordinary projected shadows, useful visualizations are

- for focused shadow mapping the visualization of the convex intersection body from a different point of view to compare the world-space shadow map area at different light directions,
- for warping the visualization of the texel borders or the light view to investigate the warping strength of the shadow map,
- for z-partitioning the visualization of the split areas to find the best split scheme respectively an appropriate split selection strategy,

- for each technique the visualization of the aliasing errors in one way or another to be able to compare the applied algorithms in an analytical way.

Considering the issues above, we focused our implemented framework on providing the most possible flexibility, versatility and the capability to compare and visualize the results from different points of view and in various ways. In the following list we briefly describe the different features and opportunities of our framework:

- **Flexibility:** to find the best solution for a specific configuration we allow the user to adjust almost any of the interesting parameters and apply several techniques simultaneously in a reasonable way via an intuitive usable graphical user interface.

- **Third person view:** in addition to the common first person view, we added the opportunity to interactively investigate the current configuration from a different point of view at any time. The available features in this view and how we implemented them will be explained in detail in Section 5.2.

- **Intersection body:** for investigation of the effects on the used shadow map area while changing the light direction or the light projection, the framework offers the opportunity to visualize the used intersection body. For further details see Section 5.3.

- **Texel borders:** this implemented visualization helps to understand the redistribution of the shadow map samples while experimenting with the application of the various available warping techniques. We explain the benefits of this feature in Section 5.4.

- **Splits:** to find a suitable split scheme for z-partitioning and experiment with different split selection strategies it is possible to visualize the used split areas. See Section 5.5.

- **Light views:** for investigation of the light orientation, the focusing of the light-space, the warping strength we provide an opportunity to visualize the light view respectively all light views in case of z-partitioning. A detailed description of this feature can be found in Section 5.6.

- **Sampling frequency:** for an analysis of the errors in the current configuration over the entire frame, it is possible to visualize the sampling rate. For further details on this visualization see Section 5.7.

- **Perspective aliasing error:** to analyze the actual perspective aliasing error and observe its characteristics for different angles between view vector and light direction, we added a feature to plot its behavior. A full explanation of this feature is given in Section 5.8.

Since we decided to choose a freely available model format (see Chapter 6), our framework can be used for investigation of arbitrary scenes. As mentioned above we will explain our implemented features and visualizations in the following sections in detail.

5.2 Visualization view

Some interesting aspects of the presented shadow mapping algorithms cannot be visualized in a reasonable way from the eye's point of view. A very common practice of shadow mapping frameworks is to offer the user the opportunity to show the light's view depth map as illustrated in Figure 2.1a.

Although this kind of illustration often helps a lot to understand the current configuration of the shadow mapping application, because of the missing shading information and the sometimes confusing orientations of the light source, it cannot be used to visualize all interesting aspects. For this reason we added an additional third person view that allows the user to investigate the current shadow mapping configuration at any time from a completely independent point of view. We call this third person view visualization view and the standard eye view common view in the following sections.

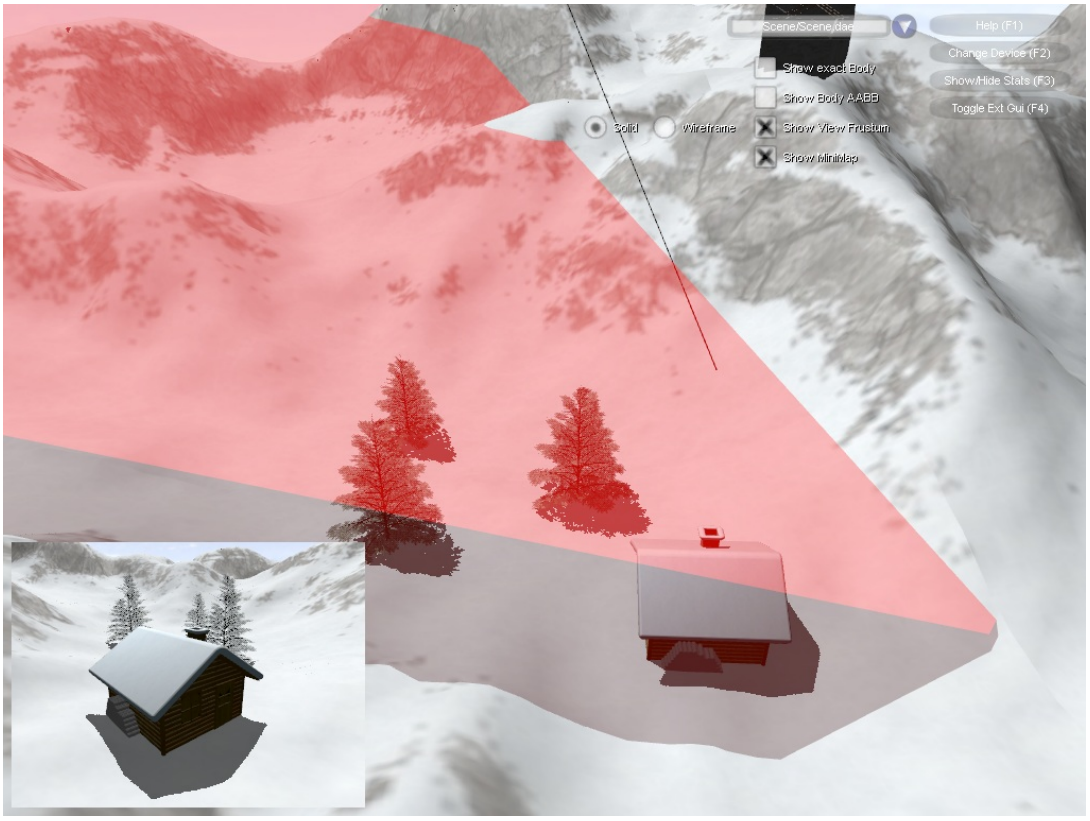


Figure 5.1: Example configuration of the visualization view. It displays the light direction (black line), the eye's point view frustum (red semi-transparent body) and its current view (minimap on the lower left).

For an easy observation of the adjustable light source movements, the current light direction will be visualized in this view. We also added the feature to switch the camera controls in the visualization view. This provides the opportunity to observe the variations of the shadows and other interesting objects from an outside view while moving the eye point. Especially for this scenario, it is also possible to display in the visualization view the current common view, to simplify the navigation in this state. Other objects which can be illustrated in the visualization view are the current eye's point view frustum, the convex intersection body and its world-space axis aligned bounding box. The visualization of the view frustum already represents for focused shadow mapping the view frustum, adjusted to the scene bounding box. In Figure 5.1 we present an example for a common configuration of the visualization view including the current view frustum and the visualization of the common view. Further visualization features of this view will be explained in the following sections.

5.3 Intersection body

Except for standard uniform shadow mapping we need to calculate for each of the presented algorithms a convex intersection body that contains all light rays for the current eye's point of view respectively in case of z-partitioning for the current split part of the view frustum. As discussed in the previous chapters this intersection body represents the essential basis for the focusing step and in case of warping the extension of the perspective warping frustum is based on it. Furthermore, as mentioned in Section 5.1, the intersection body basis almost conforms to the world-space shadow map area in most cases and can be easily used to show the differences between a point light source and a directional light source.

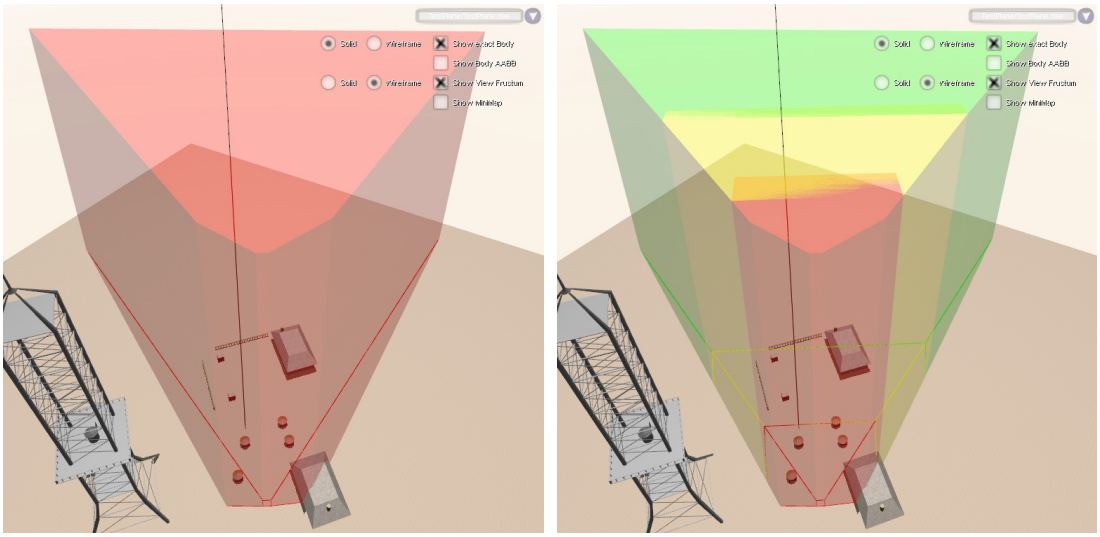
For all these reasons, we added the opportunity to visualize the intersection body respectively in case of z-partitioning all intersection bodies in the visualization view. It allows a simple intuitive analysis of the changing of the shadow map area and in case of z-partitioning it helps to compare the relations between the different world-space shadow map areas of the splits. An example configuration showing the visualization of the intersection body can be seen in Figure 5.2.

Notice that for the computation of the final focusing matrix, only the point cloud of the intersection body is required. For better illustration purposes we used a freely available library implementation [The11] of Barber's et al. [BDH96] *Quickhull* algorithm that computes a *Delaunay triangulation* of the intersection body's point cloud.

5.4 Texel borders

Especially for warping techniques the visualization of the texel borders is a simple way to show the redistribution of the shadow map samples compared to uniform shadow mapping. Moreover, if this visualization is applied in the visualization view, the external borders represent a good illustration of the basis of the perspective warping frustum.

But also for z-partitioning this visualization can be useful for investigation of the changing of



(a) Intersection body for common focused shadow mapping

(b) Intersection bodies for z-partitioning with 3 splits

Figure 5.2: Visualization of the intersection bodies. (a) intersection body for common focused shadow mapping respectively warped shadow mapping. (b) intersection bodies for z-partitioning with 3 split frusta. The corresponding view frustum respectively split frusta are shown as wireframe objects in each figure.

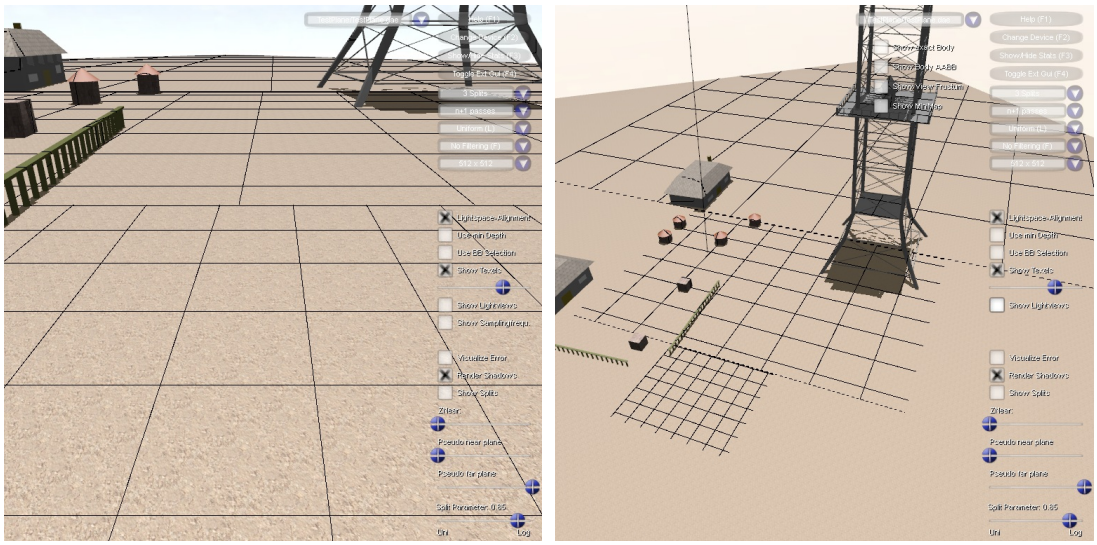
the texel size at the transitions from one split part to another.

Since the external borders of this visualization actually correspond to the world-space shadow map area, it gives a good impression of the wasted shadow map area if it is viewed from the third person view simultaneously with the visualization of the intersection body. Additionally, this visualization can be used to show the unwanted shear effects for light directions that are almost parallel to the view direction.

The fragment shader source code for computation of the grid was introduced by Lloyd [Llo07] and can be seen in Listing A.4. An example of this visualization is shown in Figure 5.3.

5.5 Split selection

Normally it is very difficult to recognize the transitions between the single splits if the shadows are generated using z-partitioning. Though the visualization of the texel borders can be used for this purpose, we additionally implemented a simple method which colorizes the scene according to the current used split selection strategy. The overlaid colors for the single split areas are chosen in the following manner:



(a) (b)

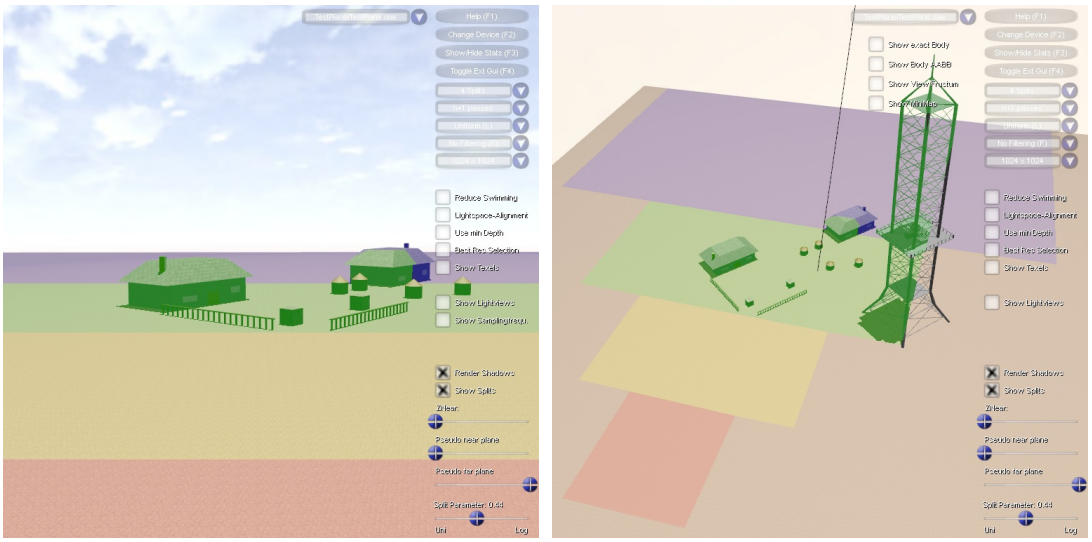
Figure 5.3: Visualization of texel borders for light-space aligned uniform shadow mapping with 3 split frusta. (a) first person view. (b) third person view.

$$color(i) = \begin{cases} \text{red} & i \ mod \ 4 = 0 \\ \text{yellow} & i \ mod \ 4 = 1 \\ \text{green} & i \ mod \ 4 = 2 \\ \text{blue} & i \ mod \ 4 = 3 \end{cases} \tag{5.1}$$

$$i \in [0, \text{MAX\_SPLITS} - 1]$$

This visualization provides an easy possibility to experiment with different numbers of splits and different split schemes, respectively split selection techniques and offers a simple way to find an appropriate partitioning configuration according to the current scenario. Figure 5.4 shows an example configuration using this visualization to examine the current split scheme.

Similar to the visualization of the texel borders this illustration can be used in the visualization view to show the world-space shadow map area for each split and investigate the wasted shadow map space. Since the split parts usually overlap partially, the colorized areas do not always correspond to the exact world-space area of the shadow map, but they still represent a good indicator for it.



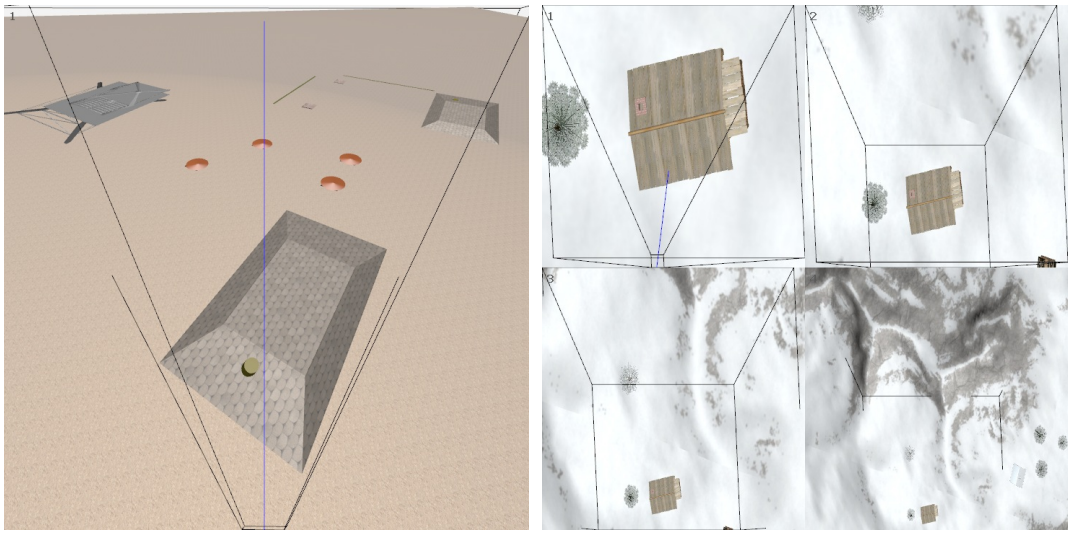
(a) (b)

Figure 5.4: Visualization of used split parts via color overlay. (a) first person view. (b) third person view.

5.6 Light views

We mentioned in Section 5.2 that most shadow mapping frameworks provide a feature to visualize the light's view depth map. Although their significance can be problematic, a shaded visualization of the light view without any illustration of the depth information, used in combination with our other implemented visualization tools, still offers a lot of interesting information. Especially for focused shadow mapping, warping methods or z-partitioning, a visualization of the light view helps to better understand the applied methods. It can be used to examine the current light source orientation, to investigate the focusing amount, to experiment with different warping strategies or warping strengths and to find an appropriate partitioning configuration.

Therefore, instead of visualizing the common depth map, we included a feature to visualize a shaded version of the light view which can be activated at any time. To improve the orientation within the light views and for a better illustration of the warping effect of our used warping techniques, we added to each light view a wire-frame rendering of the current eye view frustum. For single split configurations, we render the current eye view vector in the visualization of the light view and in case of z-partitioning the rendering of the view vector is included into the light view visualization of the first split. In Figure 5.5 we show examples for the light view of LiSPSM and z-partitioning.



(a) Light view of LiSPSM. (b) Light view of z-partitioning with 4 split frusta.

Figure 5.5: Visualization of light views including a wire-frame object of the current view frustum respectively split frustum (black lines) with the corresponding eye view vector (blue line).

5.7 Sampling frequency

The sampling frequency of the shadow map denotes the sampling rate with respect to the screen space and represents a good approximation of the shadow mapping errors over the entire frame. Thus, we added the opportunity to visualize the sampling frequency in the common view via an overlaid color. This visualization allows comparing the amount of perspective aliasing and even projective aliasing for different shadow mapping techniques and different parameter configurations. Figure 5.6 shows the used color mapping.

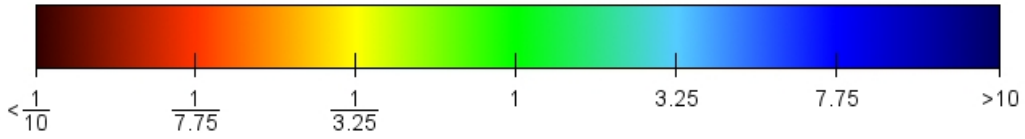


Figure 5.6: Color mapping for the visualization of the sampling frequency. The values represent an approximation of the screen pixel to shadow map texel ratio.

For an independent analysis of both shadow map directions, we added the feature to visualize the sampling frequency in x-direction of the shadow map and in y-direction. For an overall result we use the determinant of x- and y-direction. As mentioned above this kind of visualization approximates the errors for every visible point in the scene. Compared to most other tools

for error analysis, this is actually a great advantage, since most common error metrics focus only on perspective aliasing along the view vector respectively at the center of the screen and cannot be used to analyze the current scenario itself. The major drawback of this method is given by the fact that it can be only used for comparison purposes and is not suitable for an analytical error analysis.

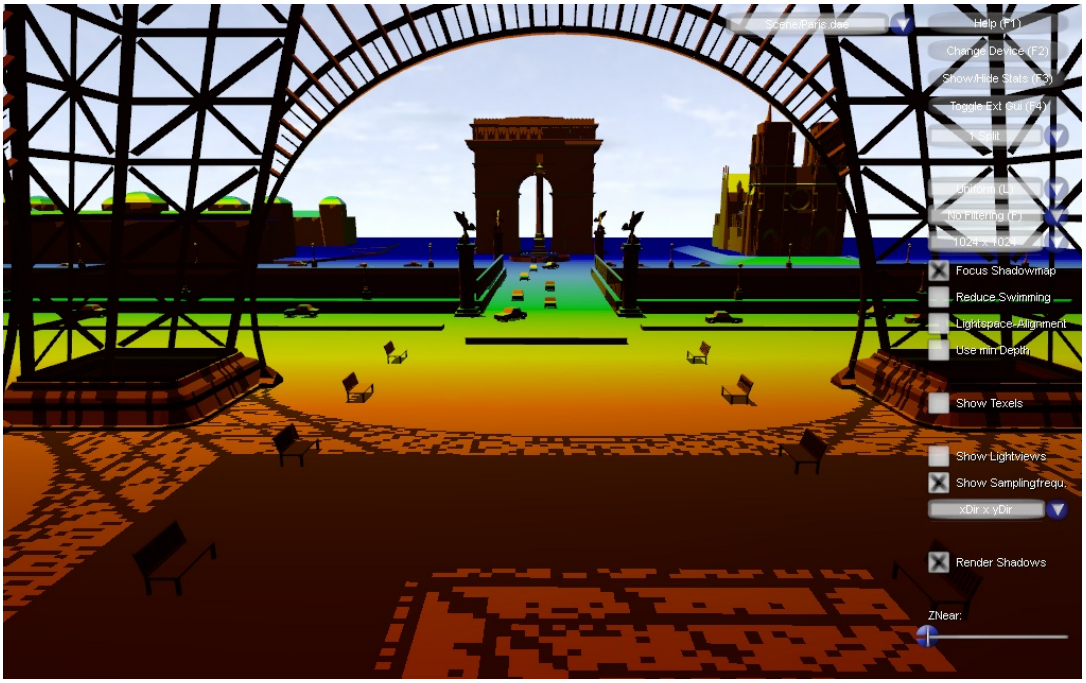


Figure 5.7: Visualization of the sampling frequency. We have undersampling (red shaded) nearby the viewer and oversampling (blue shaded) at the distant regions. Optimal sampling rates are given at the green shaded areas.

A sample fragment shader source code to visualize the sampling frequency can be seen in Listing A.3. An example illustration of the sampling frequency visualization is shown in Figure 5.7.

5.8 Perspective aliasing error

In addition to the visualization of the sampling frequency, which only represents an approximation of the shadow mapping errors, we implemented a feature that allows us to interactively analyze the actual perspective aliasing error along the current view vector over the entire length of the view frustum. Considering only an overhead directional light source or a point light with a fairly narrow field of view, this feature calculates the error values for light-space aligned uniform shadow mapping and any kind of perspective warped shadow mapping whether they are applied in a single split configuration or combined with z-partitioning. The values are computed

by using the error metric discussed in Section 2.3.3, which means the errors for uniform shadow mapping methods are computed by the equations 2.7 and 2.8, for LiSPSM by the equations 2.3 and 2.4 and for PSM by the equations 2.5 and 2.6. Based on the obtained error characteristics, this visualization enables the user to find the best analytical solution for the current view frustum configuration.

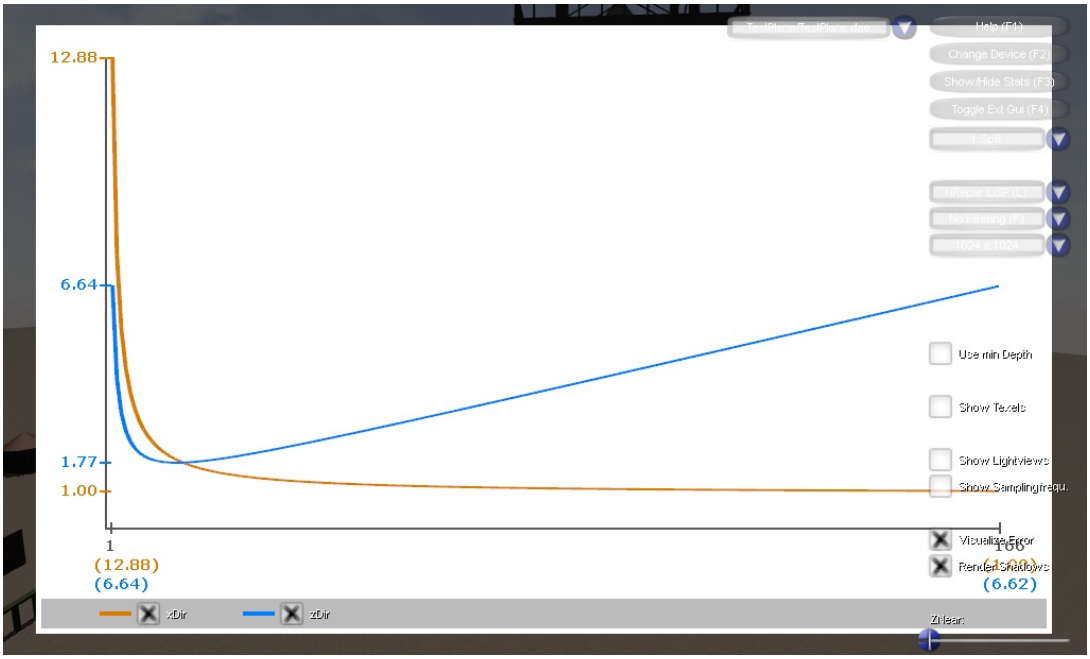


Figure 5.8: Example for visualization of perspective aliasing error. The shown graph corresponds to the error in x- (orange) and z-direction (blue) of LiSPSM for an overhead directional light with $\gamma \approx 90°$.

Figure 5.8 shows an example of this visualization. The x-axis in the shown two-dimensional diagram denotes the eye view depth range currently used for shadow mapping. In case of focused shadow mapping, the plotted values correspond to the near and far plane distances of the adjusted view frustum. If the view frustum has been partitioned using z-partitioning, the values on the x-axis represent the computed split distances and the near and far plane distances. We also added a visualization of the pseudo-near or pseudo-far plane. But they are only displayed if their values are greater than zero. Beneath the shown depth values on the x-axis we additionally display the corresponding perspective error values and in case of z-partitioning the plotted error values include the errors right after the split borders.

The values on the y-axis correspond to the error in light-space x- and z-direction calculated according to the used error metric. The displayed values represent the minima and maxima of the corresponding curve and are used to compute the scaling along the y-axis to achieve an optimal representation of the graphs within the available area.

CHAPTER 6

Implementation

6.1 Implementation overview

Our introduced framework was implemented as a *Multi-threaded dynamic linked C++ application*. Using dynamic linking gave us the opportunity to implement several parts of the application independently from the other parts. This means switching to another rendering API or another model format can be relatively simply realized by implementing the corresponding library using our provided interfaces. We will give a brief overview over the code structure in the next section.

In our introduced framework we used for rendering native DirectX 10, shipped with the DirectX SDK [Mic11]. The window and input handling is done by a combination of the common Win32 API and the DirectX Utility Library (DXUT) which also comes with the DirectX SDK. Using DXUT gave us the great advantage of a freely available *Graphical User Interface* (GUI) library which provides handling- and rendering-interfaces to various common GUI-elements like buttons, checkboxes, drop-down menus and much more.

As model format for our test scenes we decided to use *Collada* [Khr11] in the version *1.4.1*. Compared to other common model formats like for example 3ds or FBX, Collada provides the following advantages:

- the API specification is completely freely available and its usage is completely free of charge,
- besides the whole scene geometry and all texture information it is able store shaders and effects, physics and animations,
- its memory requirement is quite low, since only actually used features are stored in the files,
- it defines an XML-based schema for easy transportation of 3D assets between different applications,

- the files are human-readable

6.2 Code structure and reuse

As mentioned in Section 6.1 we tried to offer certain flexibility for possible further developments by using dynamic linked libraries which are called by the actual main application during the runtime. This allowed us to decouple specific parts of the framework like for example model-loading, scenegraph-traversal or rendering from the main program. We were also able to implement some of the algorithms presented in this thesis independently from any rendering framework.

Considering the current development status, our framework is structured in the following way:

- *Common*: this library provides some basic functionalities and utilities like file in- and output, 3D vector calculations and error- and event-logging.
- *SceneGraph*: the SceneGraph library maps our internal scene structure and provides the abstract interfaces for model-loading, view frustum culling and rendering. It also implements most of the computations for the different warping algorithms and offers basic functionalities to compute the convex intersection body used for focusing the shadow map respectively finding the perspective warping frustum.
- *PluginCollada*: is an implementation of the abstract model-loading interface from the SceneGraph-library. Its only purpose is to open a given Collada file and load the necessary data according to the internal scene structure.
- *D3dRenderSystem*: this library implements the abstract rendering interface from the SceneGraph library. It offers most of the DirectX-specific functionalities and utilities like for example shader-loading, vertex buffer object creation and rendering by using the core functions of the DirectX SDK. Some of the presented algorithms like for example z-partitioning and focusing the shadow map, and the calculation of the perspective shadow mapping error values are also implemented in this library.
- *GraphicsEngine*: the GraphicsEngine library corresponds to the main interface for the application to access custom functionalities of the rendering framework or choose between the different loaded scenes.
- *Viewer*: is the main application and triggers the window creation, the scene loading and the rendering process by calling the previous described libraries. Additionally it is responsible for every window- and user input event handling by using the core functionalities of DXUT.

The implementation of the main application Viewer focuses on visualizing and experimenting with the algorithms presented in this thesis. Thus, optimizing the code of the rendering loop in the Viewer to get the best performance sometimes played a minor part. By using the implemented interfaces of the previous introduced libraries, the Viewer can be easily replaced by

any other DirectX 10 application without the requirement of recompiling the libraries where the main parts of the algorithms are implemented.

CHAPTER 7

Results

In this chapter we present some results of our experiments by comparing the discussed shadow mapping techniques using two game-like example scenes. We also present performance measurements based on a rendering viewport the size of 1024 x 768 and a 1024 x 1024 shadow map. The used light source corresponds to an overhead directional light. Our benchmarking system consisted of an Intel®Core™i7 920 and an Nvidia®GeForce®GTS 250 with 512MB video memory.

7.1 Sights of Paris

The first scene consists of some famous sights of Paris in France arranged on an almost entirely flat terrain (see Figure 7.1). The scene bounding box measures approximately 850 x 800 x 330 meters (w x l x h). The total triangle amount of this scene corresponds to almost 255K triangles.

The greatest challenge of this scene regarding shadow mapping is represented by the, compared to the other scene objects, very high and complex model of the Eiffel Tower.

Detailed views of the shadow quality achieved with our framework are shown in Figures 7.2-7.3. The size of the shadow map in these screen shots was set to 512 x 512 for each split.

7.2 Winter scenery

The second scene represents a winter scenery consisting of some trees and cabins arranged on a hilly terrain (see Figure 7.4). The bounding box of this scene measures approximately 500 x 500 x 100 meters (w x l x h). The total triangle amount of this scene corresponds to almost 380K triangles.

The most challenging parts of this scene are the very detailed tree models and the relatively

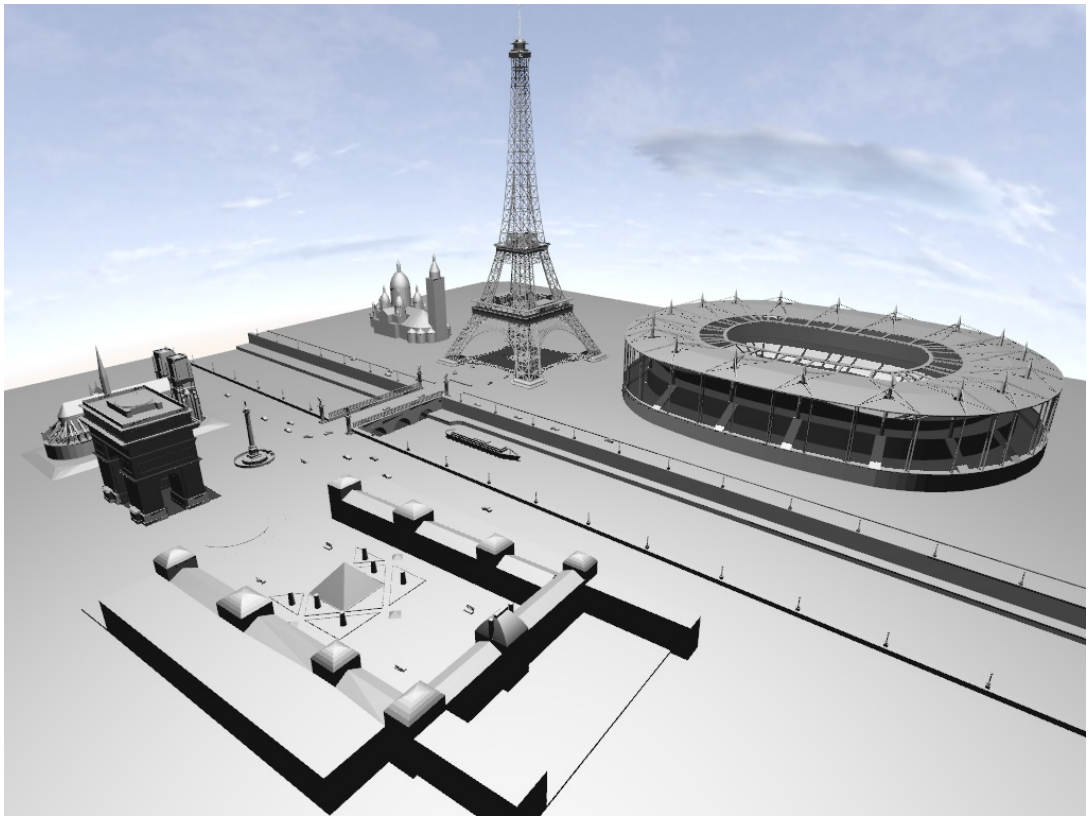


Figure 7.1: Screen shot of the test scene 'Sights of Paris'.

hilly terrain denoting the major shadow receiver.

Detailed views of the shadow quality achieved with our framework are shown in Figures 7.5-7.6. The size of the shadow map in these screen shots was set to 512 x 512 for each split.

7.3 Benchmarking results

The tables 7.1 and 7.2 show the benchmark results taken in the previously discussed example scenes. Since the performance of LiSPSM is as good as the performance of uniform shadow mapping, we limited our tests to LiSPSM and its combination with an automatic adjusted near plane (see Section 4.3), with PSSM (see Section 3.3) and with geometry cloning (see Section 3.3.6).

It can be seen that the additional pre-rendering pass to obtain the minimal depth before the shadow map creation pass and the final rendering pass at 'Automatic Adjusted zNear + LiSPSM' causes the frame rate to decrease by almost the half of the value without an automatic adjusted near plane (see 'LiSPSM'). This drawback results from the expensive read back of the

BENCHMARKING RESULTS		
Applied techniques	*Number of 3D passes*	*Framerate*
LiSPSM	1 + 1	143 fps
Automatic adjusted zNear + LiSPSM	1 + 1 + 1	80 fps
PSSM(4) + LiSPSM	4 + 1	90 fps
PSSM(8) + LiSPSM	8 + 1	60 fps
PSSM(16) + LiSPSM	16 + 1	36 fps
Automatic adjusted zNear + PSSM(4) + LiSPSM	1 + 4 + 1	52 fps
Automatic adjusted zNear + PSSM(8) + LiSPSM	1 + 8 + 1	37 fps
Automatic adjusted zNear + PSSM(16) + LiSPSM	1 + 16 + 1	23 fps
Geometry Cloning PSSM(4) + LiSPSM	1 + 1	76 fps
Geometry Cloning PSSM(8) + LiSPSM	1 + 1	31 fps
Geometry Cloning PSSM(16) + LiSPSM	1 + 1	13 fps

Table 7.1: Benchmarking results for scene 'Sights of Paris'.

BENCHMARKING RESULTS		
Applied techniques	*Number of 3D passes*	*Framerate*
LiSPSM	1 + 1	191 fps
Automatic adjusted zNear + LiSPSM	1 + 1 + 1	78 fps
PSSM(4) + LiSPSM	4 + 1	140 fps
PSSM(8) + LiSPSM	8 + 1	94 fps
PSSM(16) + LiSPSM	16 + 1	56 fps
Automatic adjusted zNear + PSSM(4) + LiSPSM	1 + 4 + 1	55 fps
Automatic adjusted zNear + PSSM(8) + LiSPSM	1 + 8 + 1	40 fps
Automatic adjusted zNear + PSSM(16) + LiSPSM	1 + 16 + 1	28 fps
Geometry Cloning PSSM(4) + LiSPSM	1 + 1	55 fps
Geometry Cloning PSSM(8) + LiSPSM	1 + 1	22 fps
Geometry Cloning PSSM(16) + LiSPSM	1 + 1	8 fps

Table 7.2: Benchmarking results for scene 'Winter scenery'.

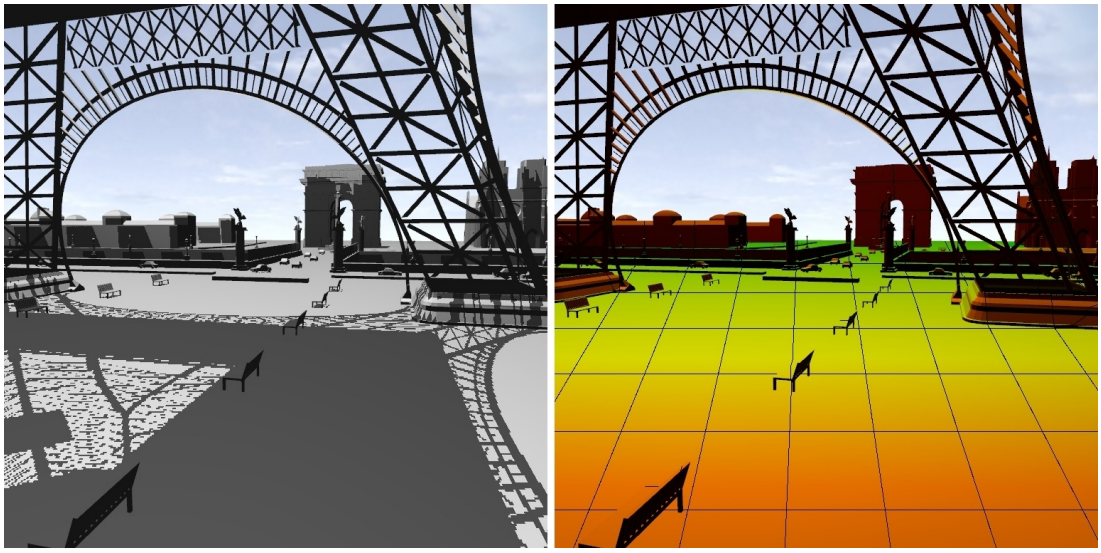


Figure 7.2: Results from the scene 'Sights of Paris'. LiSPSM with n_{repar} and 1 split. (Left) common shadow rendering. (Right) visualization of sampling rate and scaled texel grid.

depth buffer. An analog behavior can be observed for 'Automatic Adjusted zNear + PSSM(x) + LiSPSM' compared to 'PSSM(x) + LiSPSM'.

Furthermore, we can see that the advantage of geometry cloning by sending the geometry only once to the graphics card does not take effect, since the overall amount of geometry, processed through the geometry shader, might be too high (see for example 'Geometry Cloning PSSM(4) + LiSPSM' compared to 'PSSM(4) + LiSPSM'). As mentioned in Section 3.3.6, Zhang's et al. [ZSN07] approach that uses instanced drawing could solve this problem.

7.4 Comparison of pseudo-near plane and adjusted near plane

In Chapter 4 we presented two methods to reduce perspective aliasing via an increase of the near plane distance. The first approach takes advantage of the empty region nearby the camera and uses a pseudo-near plane to compute the warping parameter of n_{opt} LiSPSM. The second approach computes the minimal depth of the current configuration and adjusts the near plane distance before the computation of the convex intersection body.

In this section we will compare those techniques by using our visualization tools and briefly discuss the obtained results. Figure 7.7 shows a common configuration of n_{opt} LiSPSM visualizing the sampling rate and shows the perspective error graph.

In the Figures 7.8 and 7.9 we present the results of n_{opt} LiSPSM using a pseudo-near plane respectively an adjusted near plane distance at the same scene configuration as in Figure 7.7.

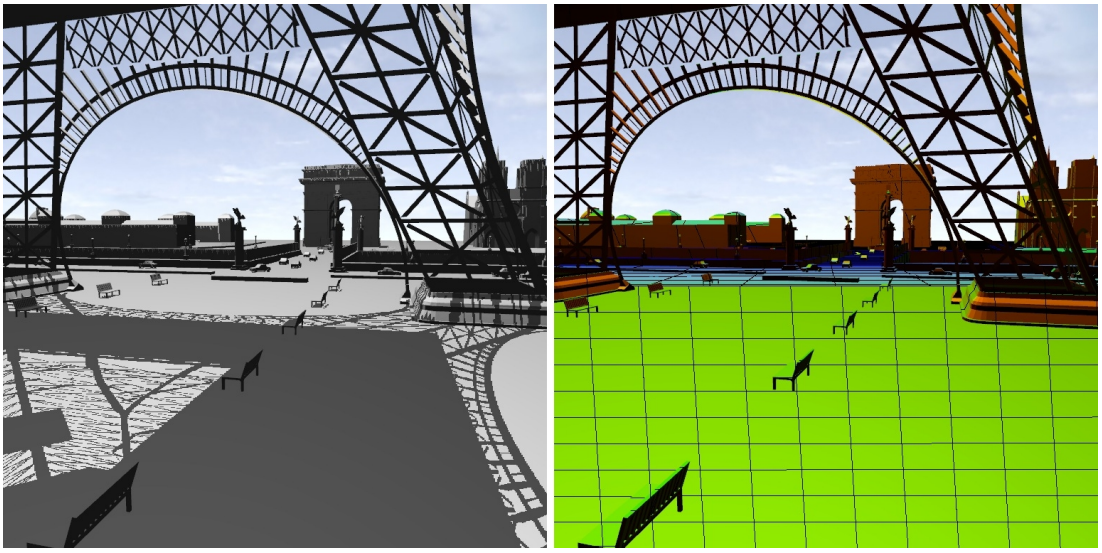


Figure 7.3: Results from the scene 'Sights of Paris'. PSSM(4) combined with PSM with automatic adjusted near plane distance. (Left) common shadow rendering. (Right) visualization of sampling rate and scaled texel grid.

It can be seen, that the error values at the pseudo-near plane distance (*PN* in Figure 7.8b) is a little bit higher compared to the error values at the automatic adjusted near plane distance, but on the other hand the frame rate is almost as high as for common LiSPSM contrary to the other technique. The high error values in Figure 7.8b between the near plane distance and *PN* need not be considered in this case, since according to Figure 7.9b this region is empty.

Summarizing, it can be said that for this certain scene configuration both techniques improve the quality of common LiSPSM, as represented by the applied visualizations, whereas each technique has its benefits and drawbacks.

7.5 Visualizing the warping effect and the shadow map area

Figure 7.10 shows a sample configuration of focused uniform shadow mapping. On the right side we visualize the world-space shadow map area and can see that almost half of the shadow map resolution is wasted since a large part of the shadow map only covers objects that are lying outside of the intersection body (shown as red semitransparent body).

In Figure 7.11 we present an example for the possible tremendous quality improvements of PSM in combination with an adjusted near plane and demonstrate the warping of the shadow map area on the right side in this figure. It can be seen that there is hardly any area of the shadow map wasted.

Figure 7.4: Screen shot of the test scene 'Winter scenery'.

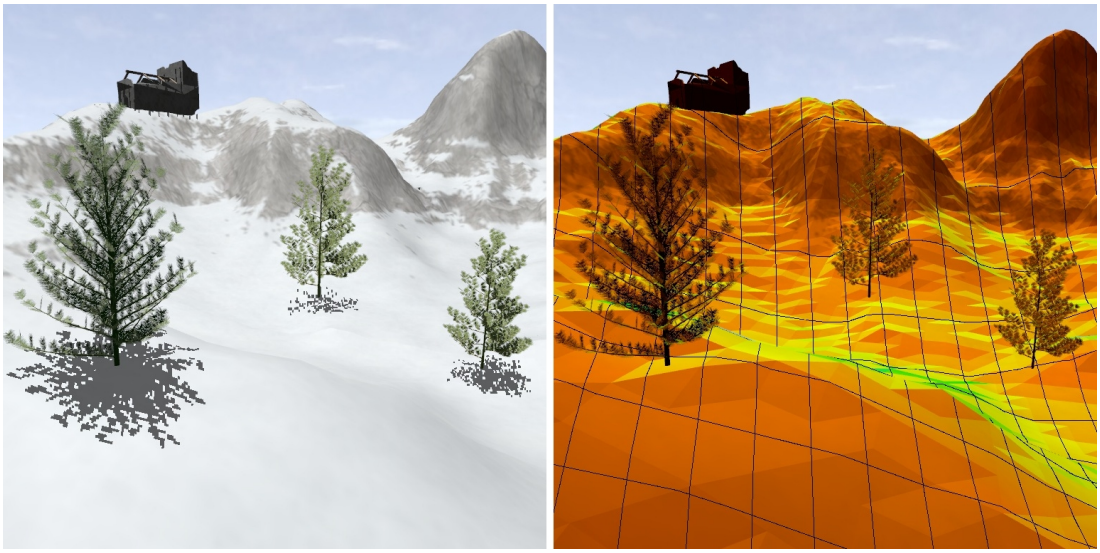


Figure 7.5: Results from the scene 'Winter scenery'. LiSPSM with n_{repar} and 1 split. (Left) common shadow rendering. (Right) visualization of sampling rate and scaled texel grid.

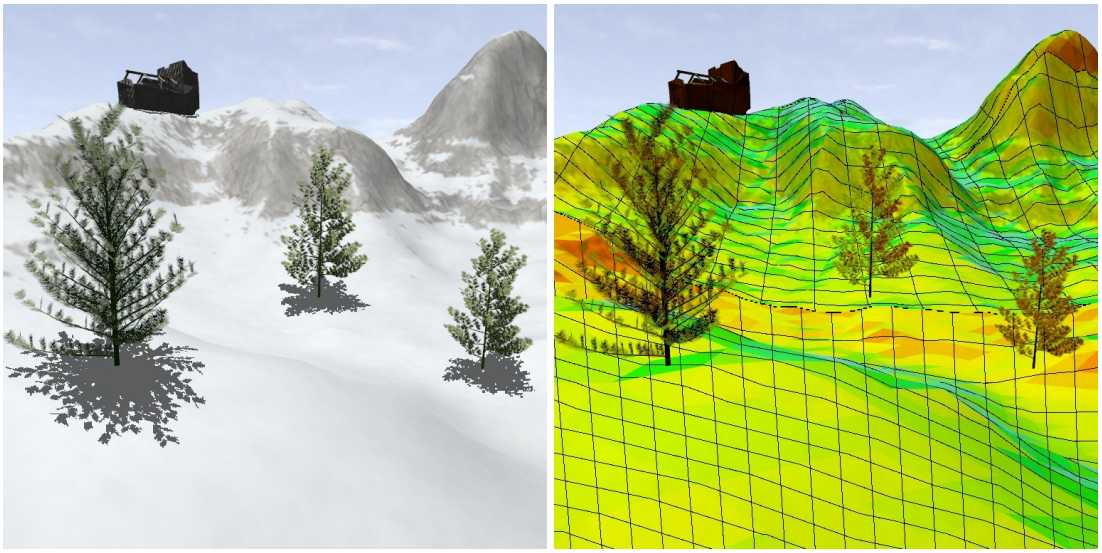
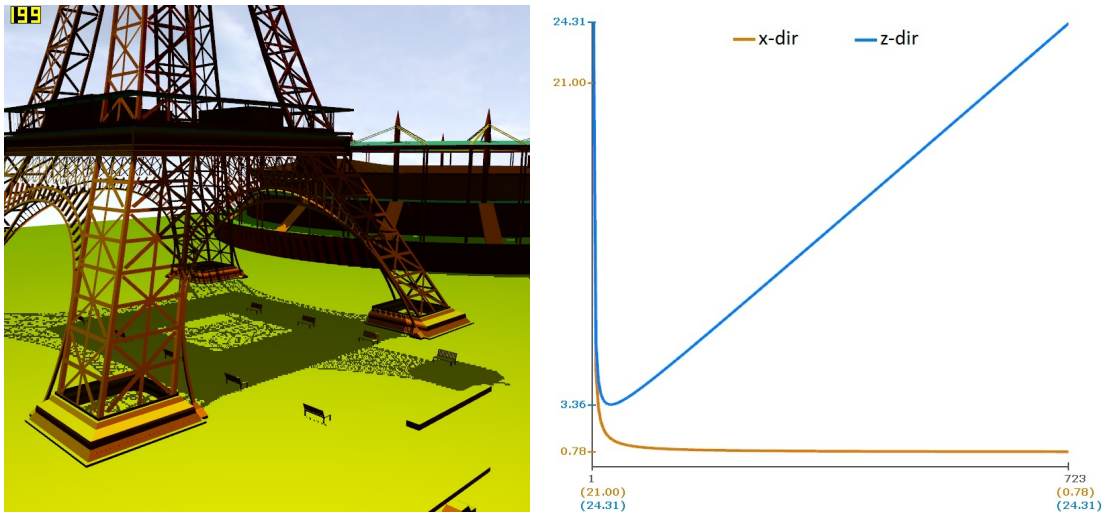


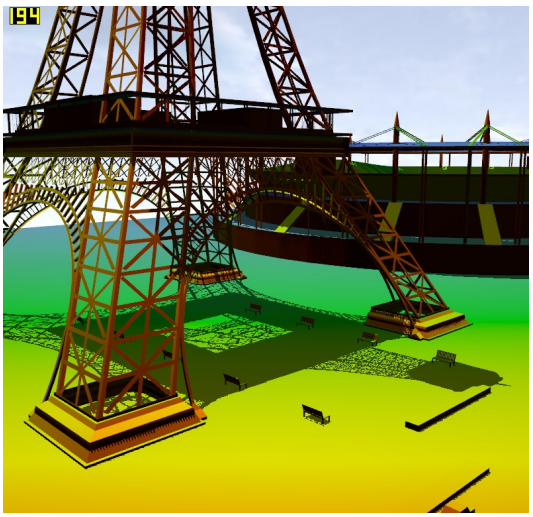
Figure 7.6: Results from the scene 'Winter scenery'. PSSM(4) combined with PSM with automatic adjusted near plane distance. (Left) common shadow rendering. (Right) visualization of sampling rate and scaled texel grid.



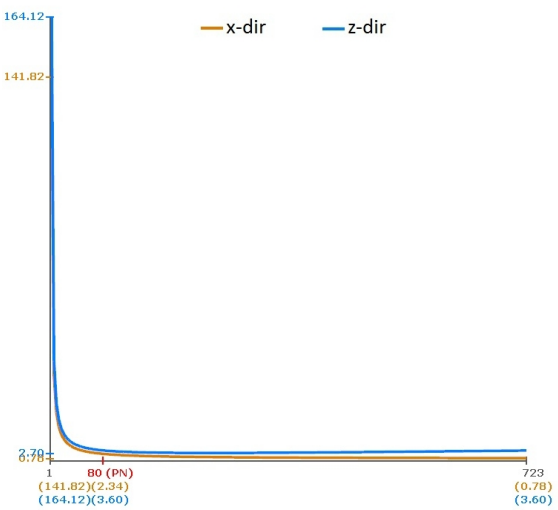
(a) Sampling frequency of n_{opt} LiSPSM

(b) Error graph of n_{opt} LiSPSM

Figure 7.7: Results for common n_{opt} LiSPSM. (a) includes the visualization of the sampling rate. (b) shows the error graph for the current configuration.

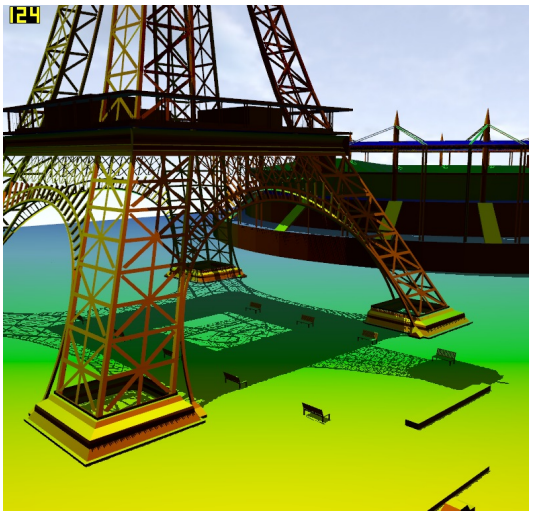


(a) Sampling frequency of n_{opt} LiSPSM using a pseudo-near plane

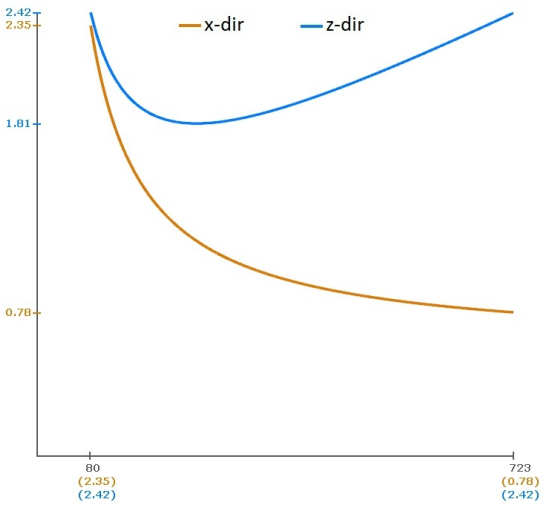


(b) Error graph of n_{opt} LiSPSM using a pseudo-near plane

Figure 7.8: Results for n_{opt} LiSPSM using a pseudo-near plane. (a) includes the visualization of the sampling rate. (b) shows the error graph for the current configuration.

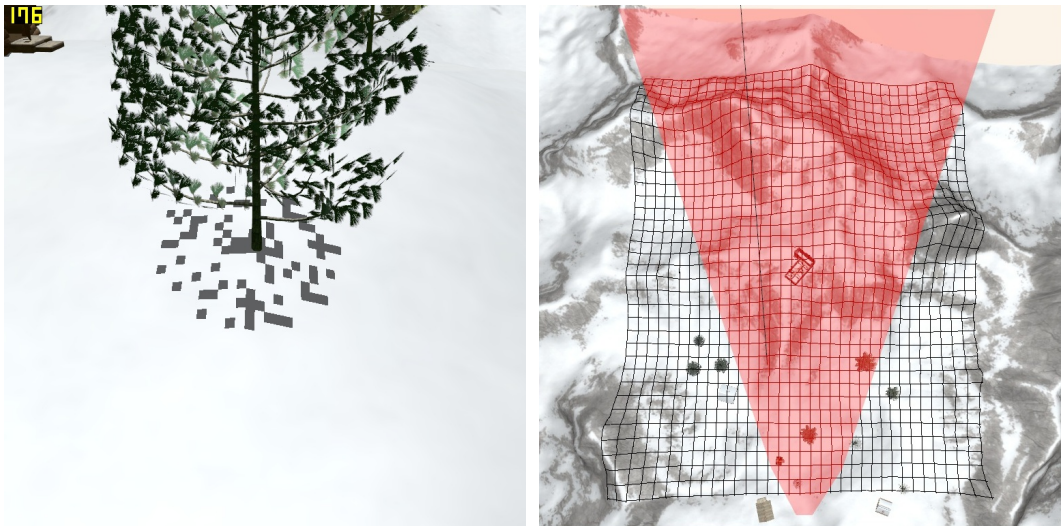


(a) Sampling frequency of n_{opt} LiSPSM using an adjusted near plane



(b) Error graph of n_{opt} LiSPSM using an adjusted near plane

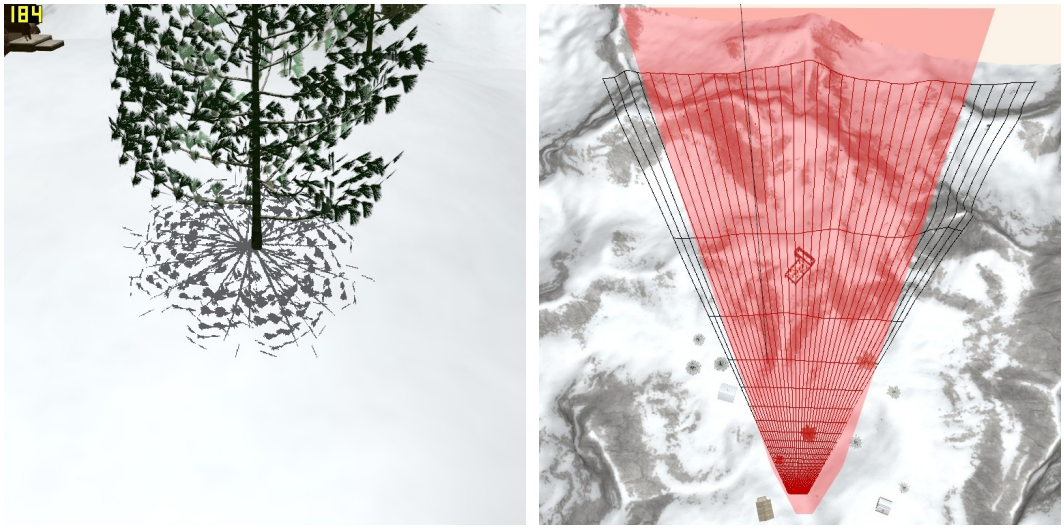
Figure 7.9: Results for n_{opt} LiSPSM using an adjusted near plane. (a) includes the visualization of the sampling rate. (b) shows the error graph for the current configuration.



(a) Uniform shadow mapping using an adjusted near plane

(b) Third person view of uniform shadow map using an adjusted near plane

Figure 7.10: Results for uniform shadow mapping using an adjusted near plane. (a) shows the common view including shadow rendering. (b) shows the third person view visualizing the world-space shadow map area.



(a) PSM using an adjusted near plane

(b) Third person view of PSM using an adjusted near plane

Figure 7.11: Results for PSM using an adjusted near plane. (a) shows the common view including shadow rendering. (b) shows the third person view visualizing the world-space shadow map area and demonstrating the warping effect.

CHAPTER 8

Conclusion

8.1 Summary

In this thesis we reviewed various shadow mapping techniques starting with Williams' [Wil78] basic shadow mapping algorithm and proceeding with several advanced real-time shadow mapping methods like focusing, warping and z-partitioning.

We implemented a framework that allows the application of each technique itself or reasonable combinations of different techniques and focuses on large virtual game-like outdoor scenarios. Moreover, it offers many opportunities to visualize different steps of the shadow creation process and helps to compare and analyze the shadow quality and the remaining aliasing errors from different points of view.

In the following we will summarize the main improvements of our framework compared to existing shadow mapping systems:

Flexibility Most of the existing systems do not offer many possibilities to experiment with different scenes and adjust various parameters. In fact, many systems are optimized on exactly one scenario using fixed settings.
We tried to design our framework in a way that offers the user the most possible flexibility. First of all we added a simple solution to load different arbitrary scenes simultaneously and offer the opportunity to switch between them during the application runtime. In the same way, we included the feature to change the window- and basic view frustum initialization parameters just by editing the corresponding initialization-file.
During the runtime, almost all parameters and configurations can be adjusted between sense making limits.

Visualizations A major contribution of our framework is represented by the ability that almost any interesting aspect of the currently applied shadow mapping techniques can be visualized in

one way or another. As mentioned before, it is not always easy to understand the occurring errors or problems, which is why our implemented visualization features can help figuring out the current issue and provide the possibility to analyze it in different ways.
Especially the so called 'Visualization view', which offers the opportunity to investigate the current configuration from a third person view at any time, provides many ways to find the best shadow mapping method for the current chosen scene.

Error analysis Since the visual impression of the projected shadows sometimes can be misleading, the two implemented methods two analyze the aliasing errors of the current configuration can be very useful to perform a comparison or an analytical analysis of the used shadow mapping techniques:

- Though the first method, the visualization of the **sampling rate**, does not represent an exact error value, it provides a good way to compare different shadow mapping techniques and illustrates the perspective errors and even the projective errors over the entire image.

- The second method, which is based on Lloyd's [LTYM06] error metric and shows the actual **perspective aliasing error** along the view vector, is one of the major improvements of our framework compared to other shadow mapping systems.

Summarizing, the discussed methods in this thesis provide a useful overview over appropriate shadow mapping techniques for large virtual outdoor environments. The presented framework enables game developers to experiment with different approaches of shadow mapping and find the best solution regarding the real-time capabilities, the shadow robustness and the shadow quality.

8.2 Future work

Since shadow mapping is still a hot research topic and this thesis does not cover all techniques introduced so far, there are many possibilities for improvement and future work to shadow mapping in general and to our framework as well. In the following sections we will list some possible enhancements, which we considered during the work on this thesis:

8.2.1 Future work to shadow mapping

Complete elimination of shadow map aliasing Most of the techniques used in this thesis focus on minimizing the perspective aliasing error. However, our experiments have shown that even by the combination of the most appropriate solutions, shadow map aliasing cannot be completely eliminated, leading to unwanted aliasing artifacts like for example jagged shadow boundaries. While these remaining artifacts can be hidden by the application of filtering like for example PCF, it would be beneficial to introduce a method that eliminates all shadow map aliasing for all light directions over the entire view frustum. An alternative partitioning approach or a different rasterization of the final image could be used to achieve this goal.

Complete elimination of shadow flickering In Section 3.5 we dealt with the reduction of the flickering shadow boundaries imposed by the different rasterizations of the focused shadow map. We have seen that the flickering cannot be completely removed without wasting too much shadow map resolution. An approach that uses the temporal coherence of consecutive frames could solve this problem.

Realistic soft shadows As the title of this thesis already says, it focuses on rendering of hard shadows which can only be generated by an ideal point respectively directional light source. However, in reality every light source is represented by a body with definite extents and somehow can be seen as an area light source which leads to soft shadows with a variable penumbra that changes based on the light source - shadow caster - shadow receiver distances. In the past few years there have been several approaches introduced, which try to generate physically correct soft shadows by considering an appropriate sampled area light source to create the penumbra. However, this is still one of the most challenging research areas of shadow creation in virtual environments.

8.2.2 Future work to the presented framework

Third person visualization of the sampling frequency As discussed in Chapter 5 we included a feature to our framework, that visualizes the sampling frequency of the shadow map in the common first person view and therefore provides the opportunity to compare the occurring aliasing errors for several configurations. Unfortunately, we did not find a way to apply this visualization in the third person 'Visualization view', since its implementation depends on the current screen-space in the actual development state. A possible solution to this problem might be to compute the sampling rate independent of the current screen-space.

Other advanced shadow mapping techniques The presented shadow mapping framework can be relatively easy extended with other advanced hard shadow mapping approaches like for example face-partitioning [Koz04] and Alias-Free Shadow Maps [AL04], or advanced filtering techniques like for example Variance Shadow Maps [DL06] and Exponential Shadow Maps [AMS+08].

APPENDIX A

Implementation details

A.1 Geometry cloning

In Chapter 3 we discussed the possibilities to minimize the amount of rendering passes for z-partitioning. According to Zhang et al. [ZSN07] we implemented a variant of geometry cloning for our experiments. In Listing A.1 we present our geometry shader program, written in Microsoft's shading language HLSL.

The outer `for`-loop runs through all splits which cover the current scene object and sets the viewport index corresponding to the current used texture atlas region. The inner `for`-loop transforms the vertices of the current triangle according to the current split.

```
1  struct GSDepthMapOut
2  {
3     float4 pos : SV_POSITION;    // position
4     float2 tex : TEXCOORD0;      // texture coordinate
5     float depth : Depth;
6     uint vpIndex : SV_ViewportArrayIndex; // viewport index
7  };
8
9  [maxvertexcount(NUMSPLITS * 3)]
10 void GSSMmain(triangle VSDepthMapOut input[3], inout
       TriangleStream<GSDepthMapOut> stream)
11 {
12   // for all splits which contain the current object
13   for (int split = g_firstSplit; split <= g_lastSplit; split++)
14   {
15     GSDepthMapOut output = (GSDepthMapOut)0;
16
17     // set the viewport index corresponding to the correct area in
```

```
18      // the texture atlas according to the current split
19      output.vpIndex = split;
20
21      [unroll] for (int vertex = 0; vertex < 3; vertex++)
22      {
23        output.pos = mul( input[vertex].pos, g_mLiSP[split] );
24        output.pos = mul( output.pos, g_mCrop[split] );
25        output.tex = input[vertex].tex;
26        stream.Append(output);
27      }
28      stream.RestartStrip();
29    }
30  }
```

Listing A.1: Geometry shader source code for z-partitioning with $1 + 1$ rendering passes using geometry cloning.

A.2 Mip map chain to retrieve minimal depth

As discussed in Chapter 4, we need to manually generate a mip map chain for obtaining the minimal depth of the current configuration to adjust the near plane distance according to this depth value. The fragment shader program shown in Listing A.2 corresponds to the 2D rendering pass, which we implemented to generate this mip map chain.

This program simply samples four adjacent texels in the given depth map and returns the minimal value.

```
1  float2 MinMipMap_PS(PostProcess_PSIn input) : SV_Target
2  {
3    float2 offset = 1.0f / g_mapSize;
4
5    float depth = depthMap.Sample(SampPointWrap, input.tex).r,
       depth1;
6
7    depth1 = depthMap.Sample(SampPointWrap, input.tex + float2(.0f,
       offset.y)).r;
8    depth = min(depth, depth1);
9
10   depth1 = depthMap.Sample(SampPointWrap, input.tex + float2(
       offset.x, .0f)).r;
11   depth = min(depth, depth1);
12
13   depth1 = depthMap.Sample(SampPointWrap, input.tex + float2(
       offset.x, offset.y)).r;
14   depth = min(depth, depth1);
```

```
15
16   return (float2)depth;
17 }
```

Listing A.2: Fragment shader source code for the generation of the mip map chain to compute the minimal depth of the current frame.

A.3 Sampling rate

The sample fragment shader source code, shown in Listing A.3, presents a way how the sampling frequency of the shadow map can be visualized as discussed in Chapter 5. The approach to compute the determinant of both shadow map directions for an overall result of x- and y-direction was proposed by Lloyd [Llo07].

The two operations at the beginning compute the partial derivative of the given texture coordinates with respect to the screen-space x- and y-coordinate. Then we calculate dependent from the current chosen `error`-selection the determinant or the length of the computed vectors. Finally, we look for the corresponding interval in `iv`, perform a linear interpolation of the assigned colors in `clr` and return the final RGB-value.

```
1  cbuffer cbConstant {
2    const float3 iv[] = {
3      float3(.0, 1/7.75, 1/7.75),
4      float3(1/7.75, 1/3.25, 1/3.25-1/7.75),
5      float3(1/3.25, 1, 1-1/3.25),
6      float3(1, 3.25, 3.25-1),
7      float3(3.25, 7.75, 7.75-3.25),
8      float3(7.75, 10, 10-7.75)
9    };
10   const float3 clr[] = {
11     float3(0.2, 0, 0),
12     float3(1, 0.2, 0),
13     float3(1, 1, 0),
14     float3(0, 1, 0),
15     float3(0.3, 0.8, 1),
16     float3(0, 0, 1),
17     float3(0, 0, 0.4)
18   };
19 };
20 float4 VisualizeSampling(float2 uv, float texMapSize, float
       splitFactor, int error) {
21   float2 dSdX = 1.0f / splitFactor * texMapSize * ddx( uv );
22   float2 dSdY = 1.0f / splitFactor * texMapSize * ddy( uv );
23   float area = .0f;
24
```

```
25    if (error == 0) // determinant of x- and y-direction
26      area = abs(dSdX.x * dSdY.y - dSdY.x * dSdX.y);
27    else if (error == 1) // sampling rate for x-direction
28      area = length(float2(dSdX.x, dSdY.x));
29    else // sampling rate for y-direction
30      area = length(float2(dSdX.y, dSdY.y));
31
32    float3 result = (float3)1.0f;
33    [unroll] for (int i = 0; i < 6; ++i) {
34      if (area >= iv[i].x && area < iv[i].y) {
35        result = lerp(clr[i], clr[i+1], (area-iv[i].x) / iv[i].z);
36        break;
37      }
38      else
39        result = clr[6];
40    }
41    return float4(result, 1.0);
42  }
```

Listing A.3: Fragment shader source code to visualize the sampling frequency in the shadow map for the current fragment.

A.4 Texel borders

In Chapter 5 we discussed the advantages of visualizing the texel borders. Lloyd [Llo07] introduced a simple way to compute and draw a scalable grid that corresponds to the texels. In Listing A.4 we show a sample fragment shader source code for this visualization.

At first we calculate the scaled texel size and the scaled texture coordinates inside the shadow map. Then we compute the scaled texture coordinates with respect to the screen-space, using the partial derivatives. Finally, to avoid unwanted stretching and compressing, we scale the grid line width by the length of screen-space texture coordinates and retrieve if the scaled texture coordinates lie on a line or not.

```
1  float4 VisualizeTexels(float2 uv, float texMapSize, int size,
       float splitFactor)
2  {
3    float4 color = (float4)1.0f;
4
5    float texelSize = size * splitFactor / texMapSize;
6    float lineWidth = 1.0f;
7
8    // compute grid using the scaled texelsize
9    // from "Logarithmic Perspective Shadow Maps" [Llo07]
10   float2 scaledTC = uv / texelSize;
```

```
11
12    float2 dS = float2(ddx(scaledTC.x), ddy(scaledTC.x));
13    float2 dT = float2(ddx(scaledTC.y), ddy(scaledTC.y));
14    float2 m = frac( scaledTC );
15
16    if( m.x < lineWidth * length(dS) || m.y < lineWidth * length(dT)
          )
17      color = float4(.0f, .0f, .0f, 1.0f);
18
19    return color;
20  }
```

Listing A.4: Fragment shader source code for computation of the scaled grid to visualize the texel borders.

APPENDIX B

Framework details

B.1 Main configuration and scene loading

The initialization file to change the main configuration of the application - `config.ini` - lies in the same directory as the execution binaries. It provides the opportunity to configure the configuration baseline of the window resolution, the screen mode (windowed or fullscreen) and to set the parameters for the perspective camera view frustum. The possible settings are shown in Table B.1. Also in the same directory as the execution binaries lies the initialization file

Window settings		
Setting	Enclosing tags	Type
Window width	$< width > < /width >$	integer > 0
Window height	$< height > < /height >$	integer > 0
Screen mode	$< fullscreen > < /fullscreen >$	0: windowed 1: fullscreen

Camera view frustum		
Setting	Enclosing tags	Type
Field of view	$< fov > < /fov >$	integer > 0
Near plane distance	$< zNear > < /zNear >$	integer > 0
Far plane distance	$< zFar > < /zFar >$	integer $>$ near plane distance

Table B.1: Overview of possible configuration settings of our framework.

that specifies the scenes to be loaded at the program start - `scenes.ini`. It contains some additional commented out details of the compatible model format and the list of filenames of the various scenes to be loaded.

B.2 Main structure

There are actually 5 main areas on the screen where the user is provided with information about the current state of the application or can change its variables. An illustration of the interesting regions highlighted with different colors can be found in Figure B.1. The red highlighted

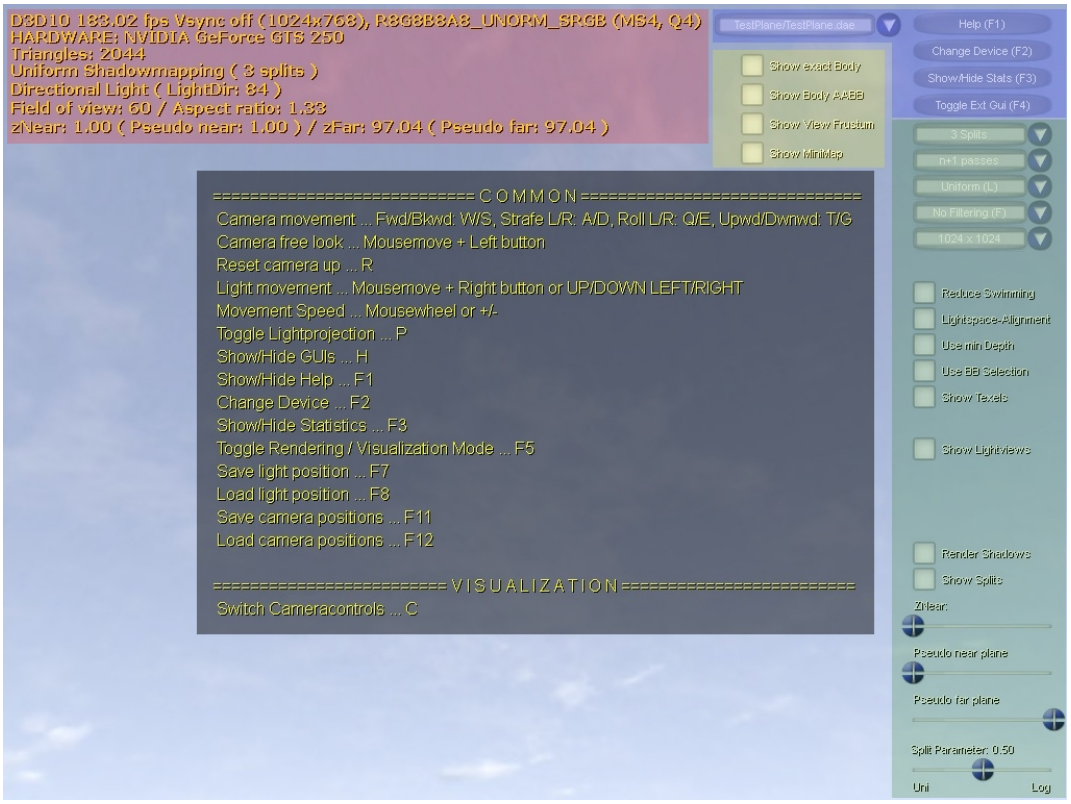


Figure B.1: Illustration of the interesting information- and controlling areas in our framework.

area contains all relevant statistics and informations like for example frame rate, used graphics hardware and current active shadow mapping algorithms. The blue highlighted *graphical user interface* (GUI) offers the opportunity to

- switch the current rendered scene
- show an overlay with additional helping informations on available keyboard and mouse inputs (black highlighted)
- change the device settings
- blend in an extended GUI to change the specific parameters of the current used shadow mapping techniques (green highlighted).

The yellow highlighted GUI is only visible during the third person view which will be described in Section 5.2.

List of Figures

1.1 Shadows generated with CryENGINE 2

2.1 Example visualization of the light's view depth map 6
2.2 Self shadowing artifacts caused by resampling inaccuracies 7
2.3 Projection aliasing artifacts caused by very sparsely sampled surfaces 8
2.4 Jagged shadow boundaries caused by perspective aliasing 9
2.5 Illustration of shadow map aliasing 11
2.6 Generalized illustration of shadow map aliasing 12
2.7 Illustration of error changes in both directions 14
2.8 Perspective aliasing error distributions along the length of the view frustum 15

3.1 Comparison of intersection body computation 18
3.2 Computation of convex intersection body 19
3.3 Example visualization of the convex body 20
3.4 Comparison between standard shadow mapping and focused shadow mapping . . . 21
3.5 Comparison of the perspective error between standard shadow mapping and focused shadow mapping 22
3.6 Example configuration of the perspective warping frustum 23
3.7 Parameterizing of the view frustum for a directional light 27
3.8 Error comparison of LiSPSM techniques 28
3.9 Comparison of the perspective aliasing error distribution between focused uniform shadow mapping and light space perspective shadow mapping 29
3.10 Schematic illustration of z-partitioning with an overhead light 30
3.11 View frustum splitting with and without near and far planes adjusting 31
3.12 Comparison of split schemes for z-partitioning with two split parts 32
3.13 Split selection 33
3.14 Illustration visualization of split selection based on the eye-space depth and split selection based on the shadow map with the best resolution 33
3.15 Comparison of perspective aliasing error distribution for PSSM 34
3.16 Illustration of our 3 implemented atlas strategies 35
3.17 Visualization of warping combined with z-partitioning 38
3.18 Error comparison of different combinations of z-partitioning and warping 39

3.19 The storage factor of different combinations of warping, z-partitioning and face partitioning . . . 40
3.20 Illustration of the flickering problem . . . 41
3.21 Comparison of unfiltered shadows and PCF filtered shadows . . . 42
3.22 Comparison of PCF filtered shadows with and without split-specific adjustments of the filter kernel size . . . 43

4.1 Construction of C_{start} and C in light-space . . . 46
4.2 Comparison of common n_{opt} LiSPSM with and without the application of a pseudo-near plane and a pseudo-far plane . . . 48
4.3 Comparison of PSM with conservatively low chosen near plane distance and automatic adjusted near plane distance . . . 49

5.1 Example configuration of the visualization view . . . 53
5.2 Visualization of the intersection bodies . . . 55
5.3 Visualization of texel borders . . . 56
5.4 Visualization of used split parts via color overlay . . . 57
5.5 Visualization of light views . . . 58
5.6 Color mapping for the visualization of the sampling frequency . . . 58
5.7 Visualization of the sampling frequency . . . 59
5.8 Example for visualization of perspective aliasing error . . . 60

7.1 Screen shot of the test scene 'Sights of Paris' . . . 66
7.2 Results from the scene 'Sights of Paris'. LiSPSM with n_{repar} and 1 split . . . 68
7.3 Results from the scene 'Sights of Paris'. PSSM(4) combined with PSM with automatic adjusted near plane distance . . . 69
7.4 Screen shot of the test scene 'Winter scenery' . . . 70
7.5 Results from the scene 'Winter scenery'. LiSPSM with n_{repar} and 1 split . . . 70
7.6 Results from the scene 'Winter scenery'. PSSM(4) combined with PSM with automatic adjusted near plane distance . . . 71
7.7 Results for common n_{opt} LiSPSM . . . 71
7.8 Results for n_{opt} LiSPSM using a pseudo-near plane . . . 72
7.9 Results for n_{opt} LiSPSM using an adjusted near plane . . . 72
7.10 Results for uniform shadow mapping using an adjusted near plane . . . 73
7.11 Results for PSM using an adjusted near plane . . . 73

B.1 Illustration of the interesting information- and controlling areas in our framework. . 86

Bibliography

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

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

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

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, December 1996.

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

[DL06] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 Symposium On Interactive 3D Graphics and Games*, I3D '06, pages 161–165, New York, NY, USA, 2006. Association for Computing Machinery (ACM).

[Eng06] Wolfgang Engel. Cascaded shadow maps. In Wolfgang Engel, editor, *ShaderX 5 – Advanced Rendering Techniques*, volume 5 of *ShaderX*, pages 197–206. Charles River Media, December 2006.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, number 35 in SIGGRAPH '05, New York, NY, USA, 2005. Association for Computing Machinery (ACM).

[FFBG01] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 387–390, New York, NY, USA, 2001. ACM.

[Khr11] Khronos Group. COLLADA - Digital Asset Exchange Schema for Interactive 3D. http://www.khronos.org/collada/, 2011.

[Koz04] Simon Kozlov. Perspective shadow maps: Care and feeding. In Randima Fernando, editor, *GPU Gems – Programming Techniques, Tips and Tricks for Real-Time Graphics*, GPU Gems, pages 214–244. Addison-Wesley Longman, 2004.

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

[Llo07] Brandon Lloyd. *Logarithmic Perspective Shadow Maps*. PhD thesis, University of North Carolina at Chapel Hill, August 2007.

[LTYM06] Brandon Lloyd, David Tuft, Sung-Eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In Tomas Akenine-Möller and Wolfgang Heidrich, editors, *Proceedings of the Eurographics Workshop/Symposium on Rendering, EGSR*, pages 215–226, Aire-la-Ville, Switzerland, June 2006. Eurographics Association.

[Mic11] Microsoft Corporation. DirectX Developer Center. http://msdn.microsoft.com/en-us/directx/, 2011.

[RSC87] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *ACM SIGGRAPH Computer Graphics*, 21(4):283–291, July 1987.

[Sch05] Daniel Scherzer. Robust shadow maps for large environments. In *Proceedings of the Central European Seminar on Computer Graphics 2005*. Eigenverlag, 2005.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics (TOG) (Proceedings of ACM SIGGRAPH 2002)*, 21(3):557–562, July 2002.

[SJW07] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50. Eurographics Association, June 2007.

[SKvW+92] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *SIGGRAPH '92 Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 26(2):249–252, July 1992.

[The11] The Geometry Center Home Page. Qhull code for Convex Hull, Delaunay Triangulation, Voronoi Diagram, and Halfspace Intersection about a Point. http://www.qhull.org, 2011.

[TQJN99] Katsumi Tadamura, Xueying Qin, Guofang Jiao, and Eihachiro Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. In *Proceedings of the International Conference on Computer Graphics*, page 166. IEEE Computer Society, 1999.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH '78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, 12(3):270–274, August 1978.

[WS06] Michael Wimmer and Daniel Scherzer. Robust shadow mapping with light space perspective shadow maps. In Wolfgang Engel, editor, *ShaderX 4 – Advanced Rendering Techniques*, volume 4 of *ShaderX*, pages 313–330. Charles River Media, March 2006.

[WSP04] Micheal Wimmer, Daniel Scherzer, and Werner Purgathofer. Light space perspective shadow maps. In Alexander Keller and Henrik W. Jensen, editors, *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)*, pages 143–151. Eurographics Association, June 2004.

[ZSN07] Fan Zhang, Hanqiu Sun, and Oskari Nyman. Parallel-split shadow maps on programmable GPUs. In Hubert Nguyen, editor, *GPU Gems 3 – Programming Techniques for High-Performance Graphics and General-Purpose Computation*, volume 3 of *GPU Gems*, pages 203–235. Addison-Wesley Professional, August 2007.

[ZSXL06] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, VRCIA '06, pages 311–318, New York, NY, USA, 2006. Association for Computing Machinery (ACM).

[ZZB09] Fan Zhang, Alexander Zaprjagaev, and Allan Bentham. Practical cascaded shadow maps. In Wolfgang Engel, editor, *ShaderX 7 – Advanced Rendering Techniques*, volume 7 of *ShaderX*, pages 305–329. Charles River Media, March 2009.