FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Real-Time Shadows for Large-Scale Geospatial Visualization

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Michaela Niedermayer
Matrikelnummer 01227211

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr. Werner Purgathofer, TU Wien, Vienna, Austria
Mitwirkung: Dipl.-Ing. Dr.techn. Jürgen Waser, VRVis Research Center, Vienna, Austira
Dipl.-Ing. Daniel Cornel, VRVis Research Center, Vienna, Austria

Wien, 23. Oktober 2018

Michaela Niedermayer

Werner Purgathofer

Real-Time Shadows for Large-Scale Geospatial Visualization

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Michaela Niedermayer
Registration Number 01227211

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Dr. Werner Purgathofer, TU Wien, Vienna, Austria
Assistance: Dipl.-Ing. Dr.techn. Jürgen Waser, VRVis Research Center, Vienna, Austira
Dipl.-Ing. Daniel Cornel, VRVis Research Center, Vienna, Austria

Vienna, 23rd October, 2018

Michaela Niedermayer

Werner Purgathofer

Erklärung zur Verfassung der Arbeit

Michaela Niedermayer
2041 Hetzmannsdorf 52

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

Wien, 23. Oktober 2018

Michaela Niedermayer

Acknowledgements

I would like to thank my family for the financial and overall support during my studies. Further, I would like to thank my advisors Jürgen Waser and Daniel Cornel for their patience and the good cooperation. This work was enabled by the Competence Centre VRVis. VRVis is funded by BMVIT, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

Kurzfassung

Hochwasser ist ein oft unvorhergesehenes Ereignis das jederzeit eintreten kann und enorme Schäden anrichtet. Um die Gefahren frühzeitig zu erkennen und darauf reagieren zu können verwendet man Simulationen, wie im Entscheidungshilfe-System Visdom. Die Echtzeit-Anwendung ist praktisch und bietet viele Möglichkeiten, wie z.B. Simulationen vom Ansteigen des Flusses über gewisse Zeit oder dem Errichten von Barrieren. Es fehlt jedoch an einer lebensnahen Darstellung der gesamten Szene, die wichtig ist, da die Visualisierung der Ergebnisse auch für Nicht-Experten wie Entscheidungsträger oder die breite Öffentlichkeit verständlich sein muss. Konkreter definiert fehlen in der Anwendung bisher Schatten, obwohl diese besonders gut geeignet sind, um Relationen von Objekten zueinander zu erkennen.

Diese Bachelorarbeit behandelt die Implementierung von Schatten in Visdom, um die Szene realer wirken zu lassen. Dies ist sehr komplex für solch große Szenen über Großstädte und Landstriche und bedarf viel selbst entwickelten Strategien, da es keine fertigen Lösungen dafür gibt. Wir präsentieren eine Adaptierung des Algorithmus *cascaded shadow mapping*, die eine gute Möglichkeit bietet die Szene zu unterteilen, um damit bessere Ergebnisse in der Schattenqualität zu erzielen. Weitere Verbesserungen werden präsentiert, um die visuelle Qualität der Schatten zu erhöhen und vorkommende Artefakte zu vermeiden. Das Ergebnis dieser Arbeit ist die flexible Darstellung weicher Schatten für eine Vielzahl unterschiedlich großer Szenen in Echtzeit, die den Realismus und räumlichen Eindruck erhöhen und die Performance nicht zu stark beeinflussen.

Abstract

A flood is an often unforeseen event, which can happen at any time and causes enormous damage. Simulations are used to predict risks and respond to them early, like in the decision-support system Visdom. The real-time application is useful and offers many possibilities, e.g., simulations of the rising of the river over a certain time or the building of barriers. However, there is a lack of a true-to-life representation of the entire scene, which is important, as the visualization of the results must be understandable also for non-experts, such as decision-makers or the general public. More precisely, shadows are missing in the application so far, although they are particularly well suited to recognize relations of objects to each other.

This bachelor thesis covers the implementation of shadows in Visdom to increase the realism of the scene. This is very complex for such big scenes on city or even country scale and requires many self-developed strategies, as there are no ready-made solutions. We present an adaptation of the *cascaded shadow maps* algorithm, which provides a good way to subdivide the scene for better results in shadow quality. Further improvements are presented to increase the quality of the resulting shadows, as well as avoid occurring artifacts. The result of this work is a flexible visualization of soft shadows for a variety of different-sized scenes in real time, which increase the realism and spatial perception and further do not influence the performance too much.

Contents

Kurzfassung **ix**

Abstract **xi**

Contents **xiii**

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Aim of the Work . . . 2
- 1.4 Methodology . . . 3
- 1.5 Structure of the Thesis . . . 3

2 State of the Art **5**
- 2.1 Basic Algorithms . . . 5
- 2.2 Cascaded Shadow Maps . . . 8
- 2.3 Soft Shadows . . . 9

3 Methodology **11**
- 3.1 Depth Map and Standard Shadow Mapping . . . 11
- 3.2 Cascaded Shadow Mapping . . . 12
- 3.3 Improvements . . . 14

4 Implementation **17**
- 4.1 Setup . . . 17
- 4.2 Shadow Pass . . . 18
- 4.3 Shading Pass . . . 19
- 4.4 Parameters . . . 22

5 Results and Discussion **25**
- 5.1 Visual Results . . . 25
- 5.2 Performance Results . . . 26
- 5.3 Discussion . . . 29

6 Conclusion **33**

Bibliography **35**

CHAPTER 1

Introduction

1.1 Background and Motivation

Floods are dangerous events that can occur at any time of the year, due to different causes, including heavy rainfall, snow melting, river overtoppings, dike breaches and much more. It is often not possible to predict such events. The financial costs of these natural events amount to several million Euros, e.g., the floodwater in Austria in 2013 caused damage in the amount of EUR 870 million [Lic14]. Visdom is a software framework used for decision support in flood management [vis]. It helps to simulate and visualize floods over a specific period of time in different cities or countries, e.g., for Austria or for the city of Cologne, Germany. In different scenarios, several situations can be simulated to determine, which buildings would be in danger and what kind of barriers would help to save these objects before they are damaged. These simulation results are mapped to the geospatial domain in a 3D visualization. The visualization should not be too complex that important results can be easily distinguished, but it should be true-to-life that also non-experts are able to understand it quickly. The visualization has already a high degree of realism in many areas of the application, but not in illumination.

1.2 Problem Statement

For a true-to-life presentation of the visualization results, shadows are missing in Visdom, although they would be well suited to get knowledge about the relations of the objects to each other. Shadows are the best solution to determine the relative location of objects. They anchor objects in the scene, which fixes the problem that they sometimes seem to “float” in a scene without shadows. Further advantages of shadows are that they indicate the position of light sources and give a greater sense of depth to the scene. For example, if the top down camera is used in a scene with many buildings, without shadows it is not possible to say, if a building is higher than another one or how high the buildings are

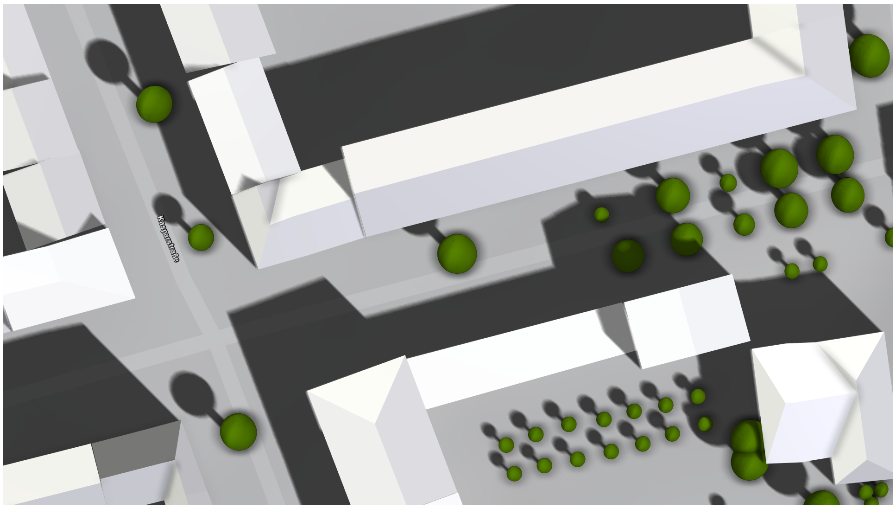


Figure 1.1: This figure shows a small view in Visdom with the top down camera. Through the shadows, the highs of the buildings can be determined.

in general. As can be seen in Figure 1.1, the length of shadows allows to determine the relative height of the buildings.

It is a complex task to implement shadows for such large scenes. There exist no ready-made solutions that could be implemented without adaptions to our use case. Common algorithms for shadow calculation have different limitations. These lead to artifacts in diverse situations, which we do not want to have in our visualization. Some of the artifacts are well-known (e.g., incorrect self-shadowing or perspective aliasing) and solutions for these can be found in the literature. However, the solutions suggested for synthetic cases do not always work in real-world scenarios, so robust solutions have to be adapted or developed. In addition, some artifacts stem from faults in the geospatial data gathered from different sources over which we have no control. These are the challenging problems, since existing approaches do not tackle these issues and, therefore, own solutions need to be developed.

1.3 Aim of the Work

The visualization of the flood results is an important step, since it needs to be understandable also for non-experts like decision-makers or the general public. Therefore, the aim of this work is the implementation of high-quality shadows without artifacts in the 3D visualization to increase the realism and spatial perception for large-scale scenes including an entire city or country. A further goal is to render the shadows without reducing the performance significantly.

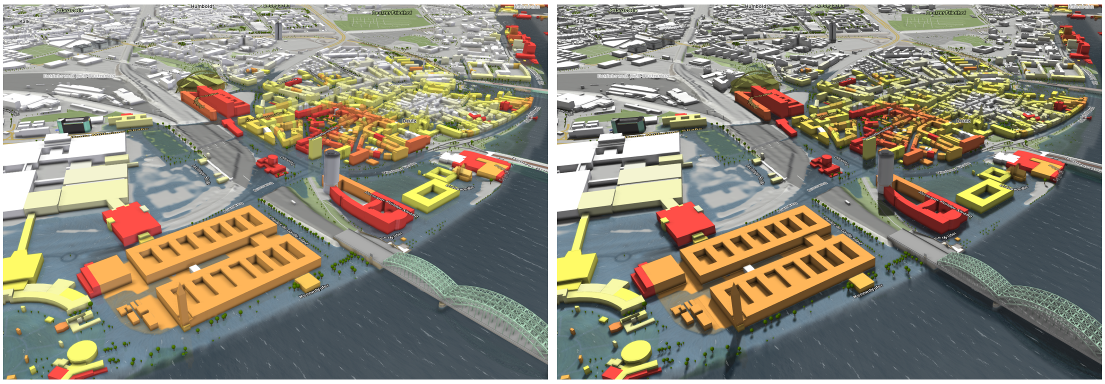


Figure 1.2: A scene view from the city of Cologne without and with shadows.

1.4 Methodology

This thesis covers the implementation of shadows with an adapted cascaded shadow maps (CSM) algorithm and further improvements to increase the quality of the shadows. The first step is the implementation of standard shadow mapping with two render steps to create a single depth map and render the shadows. The succeeding change to cascaded shadow maps is a challenge, as it is normally not used for orthographic projection and for such large scenes. Therefore, some adjustments in CSM need to be done to improve the technique according the circumstances. The other improvements are implemented to avoid occurring artifacts and further increase the quality of the shadows. These improvements include a definable range of shadows, automatic calculations of values, which changes according the size of the scene, and soft shadow techniques, to smooth the transition between shadow and light. With various user parameters, the shadows can be adapted even more precisely to the scene and thus achieve even better quality. The most important criterion is that the algorithm returns good-looking soft shadows by high performance, independent of the size of the scene. In Figure 1.2 the difference between a scene with and without shadow can be seen.

1.5 Structure of the Thesis

The next chapter (chapter 2) describes the state of the art in real-time shadow rendering techniques. At first, the two common shadow algorithms are explained. Afterwards an extension of one of the algorithms is described, as well as two soft shadow techniques, needed to smooth the resulting hard shadows. In chapter 3, the theoretical background of the implemented techniques is explained, including the two passes of the shadow mapping algorithm, the change from the standard method to cascaded shadow mapping and all further improvements to enhance the quality of the resulting shadows. In the following chapter 4, the implementation of the two render passes of shadow mapping is clarified in detail. In addition, an overview of all occurring artifacts can be found with a short description and the used solution. Furthermore, the difficulties in the implementation are discussed, just as the parameters, which can be changed by the user. Chapter 5 covers the results and performance tests of the implementation. The final chapter gives a conclusion of the thesis and of future work in this area.

CHAPTER 2

State of the Art

2.1 Basic Algorithms

The two best known algorithms for shadows are shadow mapping and shadow volumes [ZSXL06]. Both methods are common techniques for rendering real-time shadows. In the following subsections (subsection 2.1.1 and subsection 2.1.2) both methods are described, combined with their advantages and disadvantages.

2.1.1 Shadow Volumes

The shadow volumes technique was introduced by Franklin C. Crow in 1977 [Cro77]. A 3D shadow volume is cast out by the position of the light source and the shadowing object. A shadow volume designates the region, which lies in the shadow of the object and is a closed polyhedron. All geometry that lies inside the shadow volume is in the dark. The algorithm works in the object space and needs the connectivity information of all polygons to calculate the silhouette of each shadow caster. For this algorithm a 3D point-in-polyhedron inside/outside test is done, which means a ray is cast through the point of interest. The start point of the ray have to be outside the volume. The intersections of the ray are counted from the start point to the known point with polyhedron faces, for a front face add +1, for a back face subtract 1. The counting in this step is similar to the non-zero winding rule. The known point determines the exactly algorithm. If the known point is the eye-point, the *z-pass* algorithm is taken, if the point is in infinity, the *z-fail* algorithm is used [Wim17].

How to determine which algorithm should be taken is not given. The algorithms are equivalent, but z-fail is in reversed order. Z-pass is faster, but there is a problem with this algorithm when there exists a shadow volume in front of the near clip plane. This could lead to mistaken count and it does not exist a robust solution for this problem. The z-fail algorithm is slower and there exists a problem too, this time at the far clip

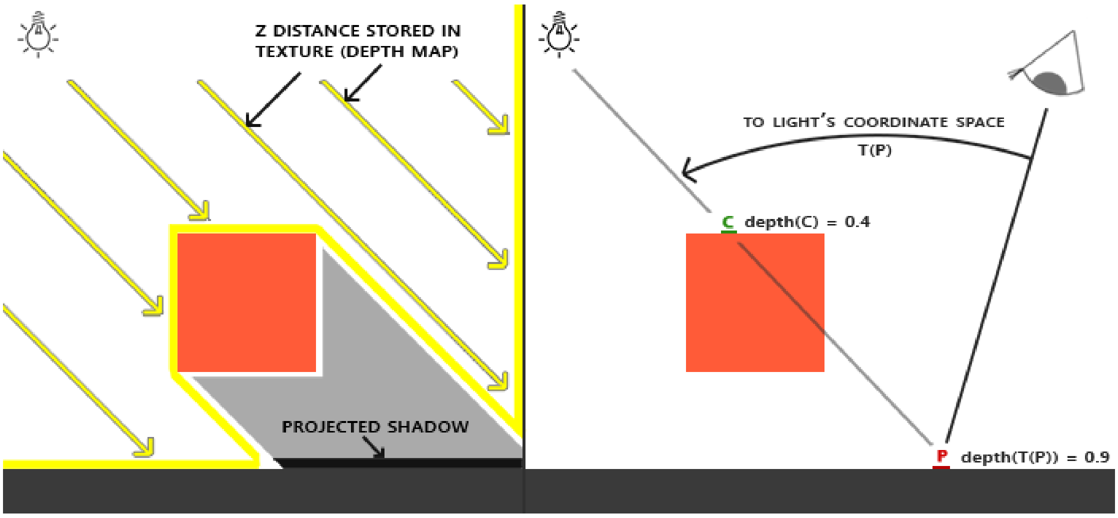


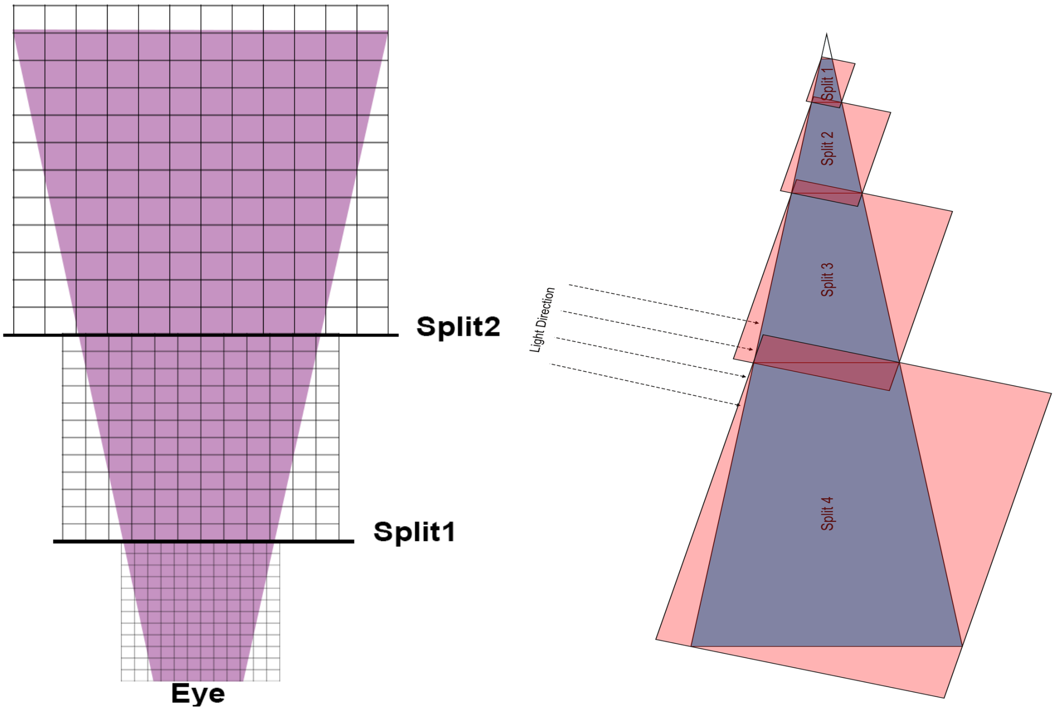
Figure 2.1: This figure shows the basic steps of the shadow mapping algorithm. Image from [dV14].

plane. Nevertheless, as opposed to z-pass, for this problem exists a robust solution with infinite shadow volumes. A further idea would be to switch between both techniques dynamically, depending on whether the scene camera lies in a shadow volume or not.

In contrast to shadow maps, shadow volumes can also be used for omnidirectional lights unproblematically and have pixel-accurate shadow boundaries. Further advantages are the arbitrary receivers, automatic self-shadowing and broad hardware support. The main disadvantage of shadow volumes is the vast fill-rate intensity. In addition, it is often difficult to get the right algorithm (z-fail / z-pass) and it does not work for arbitrary casters like smoke or fog [Wim17, SWP11].

2.1.2 Shadow Mapping

Shadow Mapping is a technique introduced by Williams et al. in 1978 [Wil78]. It is done in image space with just one additional render pass. The algorithm works in two steps, which can be seen in Figure 2.1. In the first step a depth map from the point of view of the light source, i.e., in light space, is created, commonly a directional light. In this render pass no color information is needed, as only the distance of the camera to any point that is visible from the light source in the scene is stored. The first pass is also called the *z-buffer pass*. In the second render pass, called the *shading pass*, the scene is viewed from the observer's eye, which is the camera of the scene. The values of the x, y and z coordinates are mapped to the light space. The z-values of the same point of interest of the first and second render pass are then compared with each other; these values named z-eye and z-light are both in light space. If z-eye is larger than z-light, the fragment is in shadow and gets shaded accordingly [Wil78]. Because shadow mapping is done in image space, some limitations must be observed. It has to be ensured that all objects that may



(a)

(b)

Figure 2.2: (a) Cascaded Shadow Mapping splits. (b) The sections of the depth maps with some wasted area outside the view frustum. Image from [Lea14].

cast a shadow in the viewed scene are also part of the depth map from the first render pass. A big issue that often occur in image space algorithms is aliasing (perspective aliasing, projection aliasing). Aliasing is an artifact caused by under-sampling. These staircase artifacts occur mostly because of the low resolution of the depth map near the observer's eye. Another problem that should be mentioned is incorrect self-shadowing of objects, which happens after the transformation from observer's eye space to light space. Caused by the quantization of z-buffer surfaces, the transformed point is not lying right on the surface of which it is a part, but above or below [Wil78]. A bias can be subtracted from the incorrect self-shadowed point after transformation, so the point is lying fully in light.

The major advantage of shadow mapping over shadow volumes is efficiency and speed. Although a shadow map requires two render passes, the algorithm is very fast. As shadow mapping is significantly faster than shadow volumes in many cases, it is state of the art for real-time applications e.g., games. When the incorrect self-shadowing is corrected with using a bias it can be seen as an advantage, because self-shadowing is an important factor for shadow algorithms. A further advantage is, that the technique is independent of scene complexity since there is no geometry-processing needed and no additional meshes to generate and render. In addition, the depth map can sometimes be reused, e.g., for

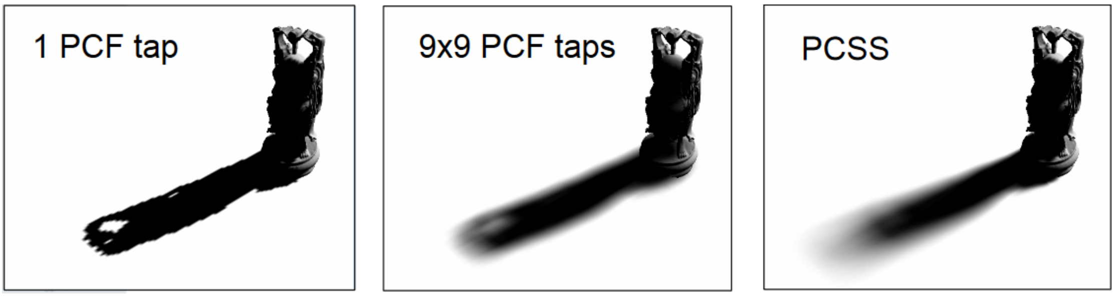


Figure 2.3: A shadow of a statue with no use of PCF (left), PCF with a 9×9 kernel size (middle) and PCSS (right). Images are taken from [Bav08].

z-culling. Beside all the aliasing, shadow mapping is also problematic for omnidirectional lights.

2.2 Cascaded Shadow Maps

It is an important problem to produce high-quality shadows in large environments for real-time applications. A solution for this challenging problem is to use cascaded shadow maps (CSM). CSM is one of many algorithms, which uses multiple shadow maps. Other similar methods are Parallel-Split Shadow Mapping (PSSM) [ZSXL06], plural sunlight buffers [TQJN99] and z-partitioning [LTY$^+$06]. Cascaded Shadow Maps is a partitioning method where the view frustum is split into different parts. The splits are parallel to the near plane of the scene camera and for every new part an own shadow map will be generated. The first shadow map covers objects near the camera, which casts a detailed shadow. The last split generates a shadow map that captures the objects in the distance with coarse resolution. There can also be some shadow maps in between, depending on how many splits are done in the view frustum. An example of these splits can be seen in Figure 2.2a. Through the separate depth maps, the resolution in each subfrustum is higher and aliasing artifacts are reduced, which means the shadow qualities are greatly improved. CSM is often seen as an discretization of *Perspective Shadow Maps (PSM)* [SD02]. As can be seen in Figure 2.2b, there is a big part of the shadow map wasted, especially in the far light frustum, because it is outside of the camera view frustum. The idea of PSM is to wrap the light frustum to exactly coincide with the view frustum [Dim07]. CSM is an algorithm that can be described in two steps [FJ17]. First, for every light's frustum, the scene depth from the lights point of view is rendered. In the second step, the scene is rendered from the camera's point of view. In which shadow map the lookup is taken depends now on the fragment's z-value.

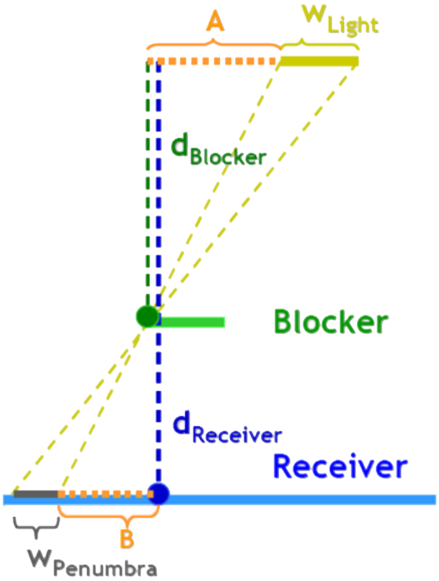


Figure 2.4: The graphical scheme of the percentage closer soft shadow algorithm. Image from [Fer05]

2.3 Soft Shadows

2.3.1 Percentage Closer Filtering (PCF)

Traditional shadow mapping produces hard shadows, which means that a pixel can either be lit or fully in shadow. This causes sharp shadow edges and a lot of aliasing. Percentage closer filtering reduces the problem of shadow map aliasing [LX11]. After computing the visibility at sample locations, the binary shadow value first must be filtered for PCF to produce new shadow values. The next step is to calculate the shadow average in the filter kernel. The new shadow values can be in the range between zero and one, but without the requirement to be exactly zero or one. After applying the new values, the shadow edges are no longer sharp, they have a smooth transition from lit to dark [KSA+15]. The larger the size of the PCF kernel, the softer are the resulting shadows. A comparison of a shadow with using PCF and no filter can be seen in Figure 2.3. Even if the shadow borders are smooth, the shadowmap texels are still visible sometimes. A possibility to reduce this too, is to use *stochastic sampling* in combination with PCF. In stochastic sampling the shadow map is multisampled and sample positions are chosen pseudo-randomly on the basis of a statistical distribution. Another approach to calculate soft shadows that also includes the PCF filtering is called Percentage Closer Soft Shadows and is described in the next subsection 2.3.2.

2.3.2 Percentage Closer Soft Shadow (PCSS)

Percentage closer soft shadows is a technique introduced by Fernando in 2005 [Fer05]. The method is based on the observation that the shadow becomes softer when the PCF kernel size increases, which means the filter size varies intelligently to achieve a plausible degree of softness as well as hard contact shadows [LX11]. The algorithm consists of three steps: the blocker search, the penumbra estimation and the PCF filtering [Fer05]. The whole function can be seen in Figure 2.4. In the first step the depths that are closer to the light source than to the receivers (i.e., points being shaded) are searched and averaged. The size of the search region depends on the light size and the receiver's distance from the light source. The second step (penumbra estimation) is the most important one, because here the penumbra width is estimated, using parallel planes approximation, with the following equation:

$$w_{\text{Penumbra}} = (d_{\text{Receiver}} - d_{\text{Blocker}}) \cdot w_{\text{Light}} / d_{\text{Blocker}}. \tag{2.1}$$

In the last step, a PCF filtering is performed using a kernel size proportional to the penumbra estimation. PCSS is generating perceptually accurate soft shadows. In addition, no pre-/post-processing or additional geometry is needed. A further advantage is that PCSS uses just a single light source sample and can seamlessly replace the existing shadow mapping shader code in an application, but with much better results. In Figure 2.3 you can see an example how the shadow with PCSS would look like.

CHAPTER 3

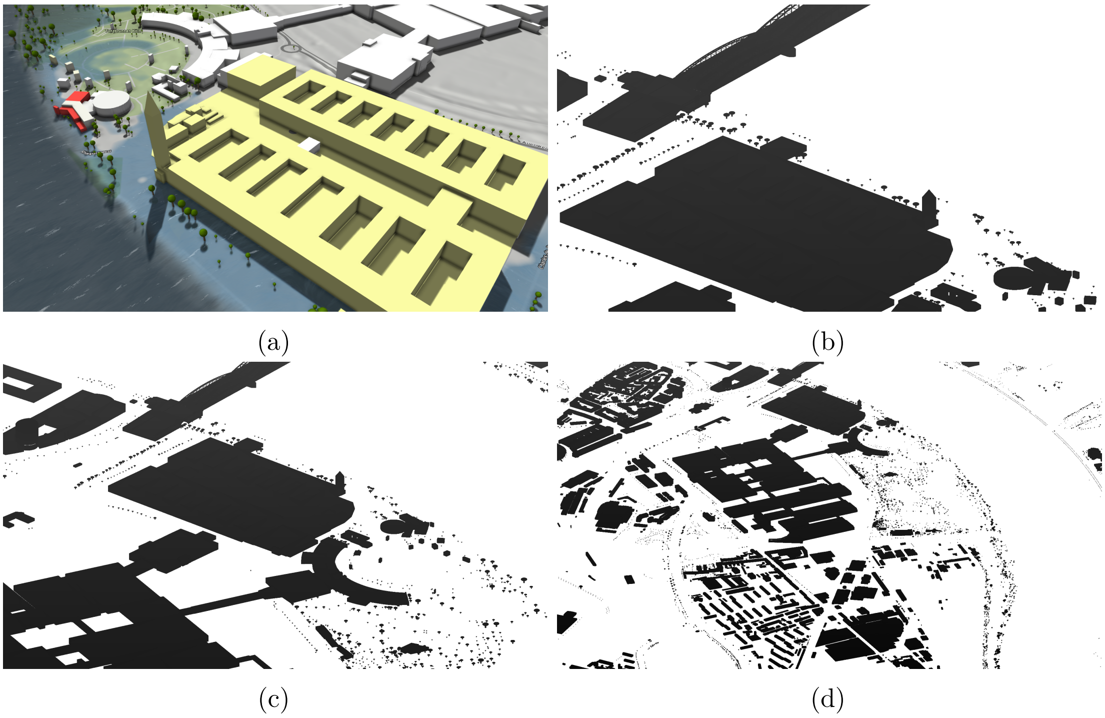
Methodology

In this chapter, the theoretical background of the implementation is explained. The goal is to implement a shadow mapping algorithm (section 3.1) to render good looking shadows in large environments, e.g., for the city of Cologne, Germany. The implementation includes many methods to calculate the necessary values automatically, suitable for the actual scene and view. When shadow mapping worked fine, cascaded shadow maps (section 3.2) are introduced for better results in big scenes. Otherwise, the shadow is very blocky due to the big texel size when just one shadow map is used for the whole scene. After cascaded shadow mapping was implemented, the results looked much better, but still not good enough when a large scene is rendered. Therefore, some improvements are required (section 3.3), including soft shadows, automatic calculations and other corrections (e.g., blending between the cascades, fade out of shadow). To generate soft shadows, the techniques percentage closer filtering (PCF) and percentage closer soft shadows (PCSS) are described.

3.1 Depth Map and Standard Shadow Mapping

The first step is to create a single depth map for basic shadow mapping. The generation of the depth map is the first render pass of the shadow mapping algorithm. The created texture stores the scene from the light source's "view". For shadow mapping, a directional light is used. This kind of light is comparable with a sun and has therefore no position in the scene, but a direction in which the light is cast. The depth map includes all kinds of meshes (buildings, bridges, etc.) and trees. The terrain is not included, which means the terrain does not cast shadows. A result of a depth map from the application can be seen in Figure 3.1b, Figure 3.1c and Figure 3.1d, already divided in the cascades, which are explained in section 3.2.

For all the shadow calculations, a new camera is added with the light's position, so that all predefined matrix calculation methods in the camera class can be used. After



(a) (b) (c) (d)

Figure 3.1: (a) Scene with the camera view. (b-d) The depth maps of three different cascades of image (a).

generating the depth map, the second render pass can now be done. This is the normal rendering but with an additional shadow calculation method. The first step in the method is the transformation of each visible surface position to the light space and subsequent projection to the image space of the corresponding cascade's shadow map. The normalized distance to the light source is stored as the current depth. As the depth map is also in the range [0,1], the closest depth from the light's point of view can now be determined with a lookup to the shadow map. The last step is to compare the depths. If the current depth is larger than the closest depth, the shadow is one.

3.2 Cascaded Shadow Mapping

Cascaded Shadow Mapping is a technique where multiple depth maps are used instead of one to address perspective aliasing. In our application, one to five cascades are possible. As described in section 2.2, the view frustum is split into different parts. The splits are located along the z-axis, but there is a key problem in how to specify the split positions. There are three schemes that can be seen in Figure 3.2. In the uniform split scheme the planes are split uniformly, which means the distribution of perspective aliasing errors are the same as for standard shadow maps [ZSXL06]. The logarithmic split scheme provides the theoretically even distribution of perspective aliasing errors. The problem in this

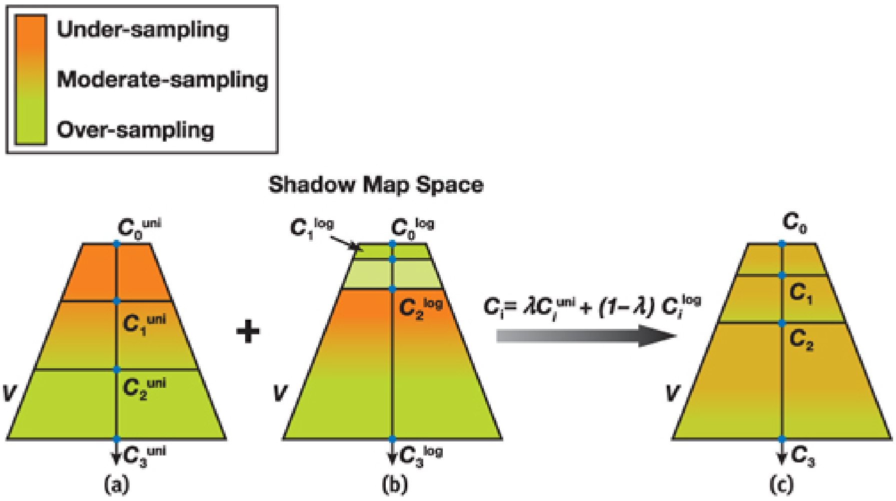


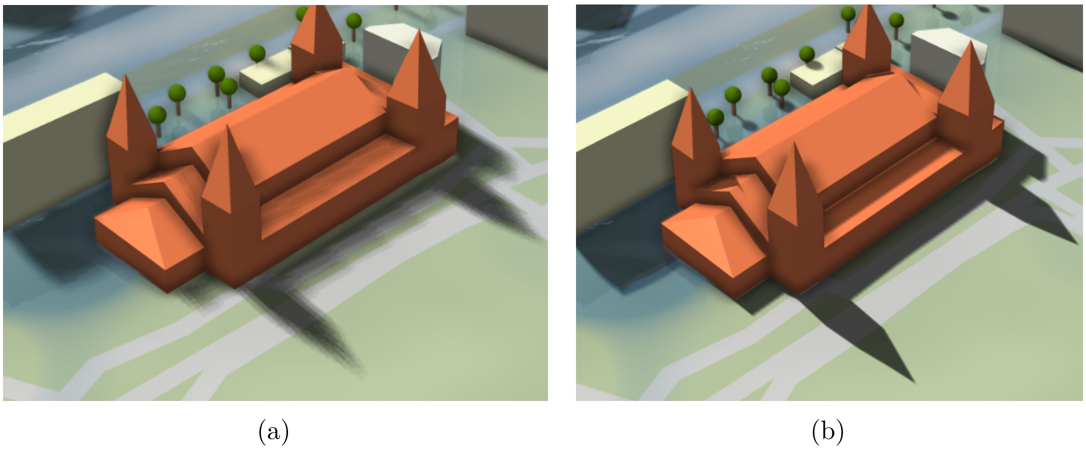
Figure 3.2: The different split schemes for multiple depth maps. Image from [Ngu07].

scheme is the length of split parts near the viewer, which are too small and therefore over-sampled. The third scheme, called *practical split scheme*, provides a moderate sampling rate in the whole view frustum and is therefore used in our cascaded shadow mapping algorithm. It can be seen as a combination of the uniform and logarithmic scheme.

$$C_{\mathrm{i}} = \lambda * (n * (f/n)^{\mathrm{i/m}}) + (1-\lambda) * (n + (f-n) * (i/m)). \tag{3.1}$$

λ is the amount in percent of how much the logarithmic part is taken. The equation of the split position for cascade i (from overall m cascades) is Equation 3.1, at which n is the near plane and f is the far plane.

The next step in the cascaded shadow mapping algorithm is to compute an orthographic projection for each subfrustum. In the standard shadow mapping method a camera object for the shadow is created to easily compute the matrices with the given scene camera. Since it would be too complicate to store all the values of each individual cascade, especially with all the further calculations and improvements that will be described in the next section, a *cascade camera* is created for each split part to easily store values like near/far plane, view matrix, projection matrix, etc. Furthermore, an additional camera is created for each cascade which faces the cascade from the direction of the directional light source, in the following called a *shadow camera*. All of these cameras are updated every time the original scene camera is changing. All the steps done for standard shadow mapping are now done for each cascade. From a list of cascade splits along the view direction, it can be decided for each visible surface position to which cascade and therefore shadow map it corresponds. In Figure 3.3, you can see the difference of the scene when using standard shadow mapping (Figure 3.3a) and cascaded shadow mapping with four cascades (Figure 3.3b).



(a) (b)

Figure 3.3: (a) A screenshot of a small part of the scene without cascaded shadow mapping. (b) The same scene when four cascades are used.

3.3 Improvements

3.3.1 Soft Shadows

One of two soft shadow techniques can be chosen in the application, the *Percentage Closer Filtering (PCF)* and the *Percentage Closer Soft Shadows (PCSS)*. If the PCSS checkbox in the client is unchecked, the PCF is automatically chosen as default shadow-method. For this technique, the size of the filter kernel can be selected in the application during run time. It is also possible to select a size of 1×1 -filter kernel, which means no PCF filtering is done. The filtering itself is done as described in the previous subsection 2.3.1.

For PCSS, two parameters can be set in the application at run time, namely, the light size and the number of samples for the included PCF algorithm. First, the blocker search is done with a fixed number of samples. After this, the penumbra size is calculated with the selected values in the client. The last step is the PCF filtering with included stochastic sampling. Through the combination with stochastic sampling, the texels are less visible. The best-looking result when using soft shadows is shadowing with PCF with a 9×9 -filter size, but this is also the most expensive regarding the performance. More details to the results and performance can be found in chapter 5.

3.3.2 Further improvements

In this subsection, several improvements integrated in the application are explained. All of them can be activated and deactivated in the client during run time. The first one is necessary to hide the cut between the cascades. Sometimes, due to the resolution changing, it is strongly visible where the next cascade begins, e.g., in Figure 3.4. A blending between the cascades is added. For all the fragments in the blend area of a user-defined size, a second shadow value is calculated. Both shadow values are linearly

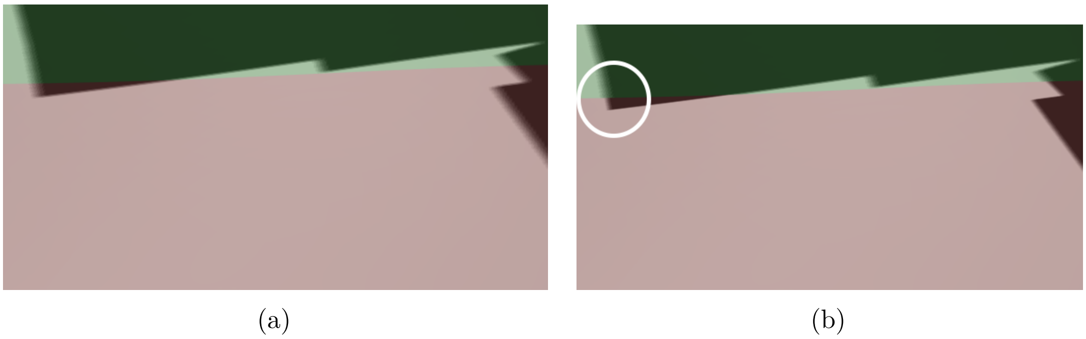

(a) (b)

Figure 3.4: A small scene view from Visdom. The colors show the different cascades, where red is the first one and green the second. (a) Blending between the cascades with a 0.05 blend area size. (b) No blending between the cascades, which can be seen exactly in the circle.

combined to get the final shadow result. Another improvement, which generates better shadow results, is a fade out of the shadow. Within a user-defined *fade out range*, the shadow intensity is gradually reduced and beyond that range, no shadows are drawn in order to reduce the cascade size. This is a big improvement for the shadow quality, since just the shadow range and fade out area is rendered in the depth maps instead of the whole scene.

CHAPTER 4

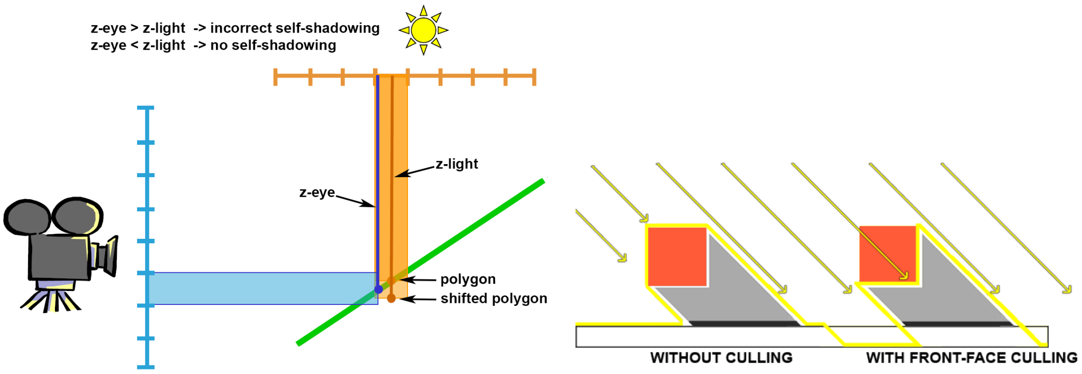
Implementation

The whole shadow mapping algorithm with methods and improvements explained in chapter 3 is implemented in Visdom with the graphics API OpenGL. The code is written in the programming language C++.

4.1 Setup

There exist three important parameters (z-near, z-far, orthozoom factor) needed for the orthographic projection matrix. The values of these parameters vary for a good-looking shadow, depending on the actual view. Since the view and therefore the amount of rendered polygons changes a lot when zooming in and out, these three values are calculated automatically. Z-near and z-far of the shadow camera of each cascade are the first two values computed in the same method. In the last section the splitting into the cascades is explained, with the splits we get the z-near and z-far values for the cascade cameras. These values are required to get the frustum corners of the actual cascade. After multiplying the frustum corners with the view matrix from the shadow camera of this cascade, we get the coordinates in view space and can store the minimum and maximum z-values.

The orthographic projection in the application is not a standard OpenGL projection matrix, but an reversed one, which means that it maps the far plane to zero instead of one [Dep15]. The orthozoom factor is a value, which does not exist in a normal orthographic projection. This value is used in the application to scale the whole scene before the projection, to map the scene to a much smaller area. For the calculation of the orthozoom factor, first the bounding box of the scene camera of one cascade is used, to get the intersection volume. In the next step, the intersection volume is projected onto the near plane of the scene camera. Further, a new bounding box around the points on the plane is built, to get the width and the height of this bounding box. The width


z-eye > z-light -> incorrect self-shadowing
z-eye < z-light -> no self-shadowing
z-light
z-eye
polygon
shifted polygon
WITHOUT CULLING
WITH FRONT-FACE CULLING

(a) (b)

Figure 4.1: (a) An example can be seen, why incorrect self-shadowing appears. Image from [Wim17]. (b) This figure demonstrates which values are taken for the depth map without and with front face culling. Image from [dV14].

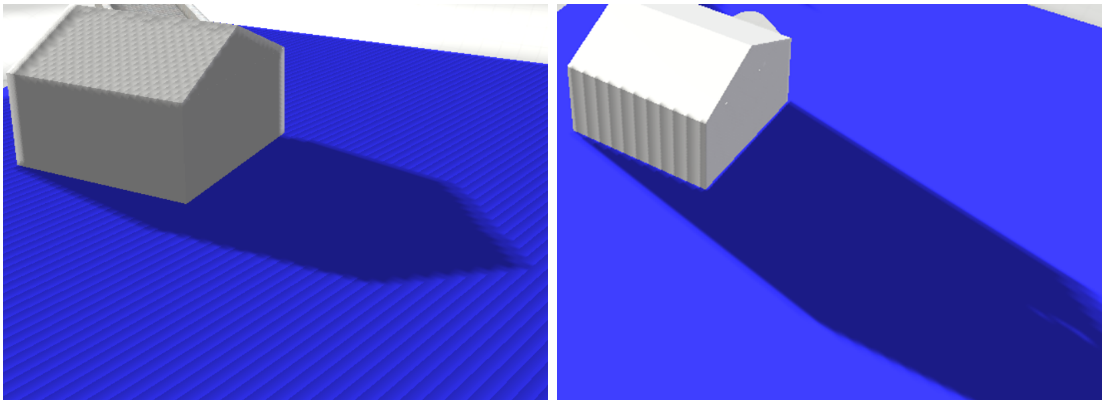
and the height are each divided by two. The maximum of these two values is then set as the orthozoom factor.

4.2 Shadow Pass

The program required to generate the depth map consists of a vertex and a fragment shader. A framebuffer is used to store the rendered result in a texture. First, the 2D texture is created with a depth render target, as no color buffer is needed in this render pass. This texture is then attached as the framebuffer's depth buffer. For the depth map generation an attachment descriptor is used to store more than one texture, as each cascade has an own depth texture. In the fragment shader the cascade index of the actual fragment is defined. This index is used to get the correct depth map. The render logic and the vertex shader are the same as for rendering meshes. The fragment shader is simpler, because it only stores the depth values. In the vertex shader the vertex coordinates are transformed from world space to the cascade's corresponding light space before projection.

The OpenGL API offers a function to cull faces for the rendering, so they are not drawn in the application. For surface rendering, it is usual to cull the back faces, which means much less triangles needs to be rendered. For the depth map creation, the value is set to front face culling. If the front face is culled, the shadow map stores the depth value of the back face instead of the front face, which can be seen in Figure 4.1b. It does not change anything for solid objects if the depth is taken from the front or back face, as it does not matter if shadows are inside the objects.

Through the difference in the depth, there are no longer artifacts on the roofs of the buildings. Furthermore, the incorrect self-shadowing explained in subsection 2.1.2 and



(a) (b)

Figure 4.2: (a) Example of incorrect self-shadowing. (b) The incorrect self-shadowing is reduced, but shadow acne appears on the backface of the building due to front face culling.

can be seen in Figure 4.1a is not visible any longer, because the z-value of the light is now always larger than the z-value of the eye, if the shaded point is not in shadow. An example of a small scene with and without incorrect self-shadowing can be seen in Figure 4.2. The only exception are objects which do not have any depth itself (e.g., a plane), as the front face culling would not change anything regarding the depth. After using front face culling, there are still some artifacts on the back faces of the meshes, named shadow acne, which can be seen in Figure 4.2b. It occurs when the shadow receiver is orthogonal to the shadow map plane (i.e., projection aliasing). A solution for this problem is implemented by calculating the dot product of the directional light direction and the normal of the shaded point. If the result is larger than or equal to zero the fragment is in shadow.

Another OpenGL integrated method, which is set before rendering, is the polygon offset parameter. In this function, the fragment's depth value is offset after the interpolation from the depth values of the corresponding vertices. It can be used to fix the incorrect self-shadowing already in the first render pass, instead of using a bias within the shader of the second render pass. The polygon offset parameter is often used as a solution for the incorrect self-shadowing, but with the use of front face culling, the incorrect self-shadowing did also not occur any longer, and it is a more robust solution.

4.3 Shading Pass

This render pass is the common shading pass with an additional shadow calculation. The computation is done in the fragment shader. Shadow camera parameters such as light-space matrices and near and far bounds are provided with a uniform buffer object. In the shader, the fragment's depth is used to find out in which cascade the shadow lookup is done. To compare the fragment with the corresponding fragment in the depth map, a transformation to the light screen space is required. The fragment's world-space position is transformed to the shadow map's image space, which maps the distance to the light source to the range [0, 1]. The z-coordinate of this value is stored as the current depth. The built-in texture-function is called, to get the closest depth from the light's point of view. A parameter of this function is the cascade index to determine, which depth map is chosen. The closest and current depth are compared to each other. If the current depth is larger than the closest depth, the result is one, and otherwise it is zero. This algorithm returns hard shadows, which means the resulting shadow is a binary value of zero or one and causes hard lines between the dark and lit regions. The soft shadow algorithms PCF and PCSS are then used to smooth the transition and to get other shadow values between zero and one.

In the percentage closer filtering algorithm, the user can select a filter size. Possible values are 1, 3, 5, 7 or 9, where a size of 9 would mean a 9×9 filter size. In two for-loops, all shadow values of the neighbors of the actual fragment are summed up and averaged. The PCSS algorithm can be divided into three parts. In the first part, the blocker search, a fixed number of blocker samples are used. We set this value to five, by default. With the light size, the near plane of the cascade, in which the fragment is lying, and the current depth, a search width is defined. This search width combined with stochastic sampling gives an offset, when the closest depth is calculated. If the current depth is larger than the closest depth, a new blocker is found. The result is the sum of the blocker closest-depth values divided by the number of found blockers. After the penumbra size calculation with the Equation 2.1, a PCF filtering is done. The user can set the number of the PCF samples for PCSS. The offset for the calculation of the current depth is the penumbra size combined with stochastic sampling. For every sample where the current depth is larger than the closest depth, an amount of one is added to the sum, which is at the end divided by the number of samples. The result is the amount of shadow, which means the returned value lies between zero and one and is multiplied with the diffuse and specular colors.

4.3.1 Difficulties

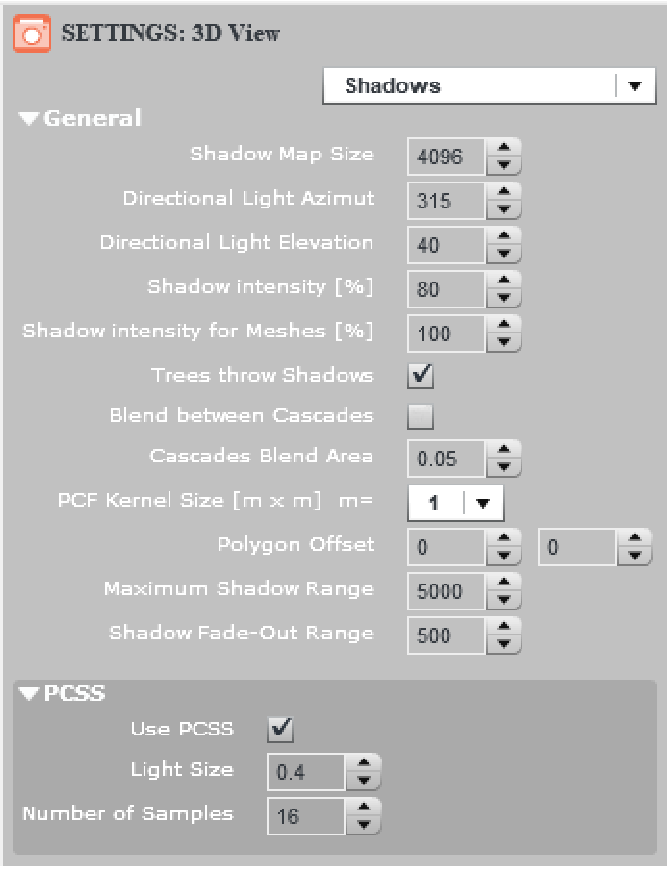
The first difficulty is the integration of the algorithms into an existing system with a large codebase and many performance optimizations. All tutorials for shadow mapping work with the original OpenGL API functions, which cannot be used in the project, because there are already predefined methods that should be used, with the same and additional functionality. Further, it is not always simple to find the correct solution if an artifact is appearing. Most shadow mapping implementations rely on parameter tweaking and magic numbers such as a polygon offset to hide artifacts in a particular scene, which is not a robust solution for a system with arbitrarily sized scenes. In a real-world application with geospatial data and geometry from different sources artifacts occurring, which would not occur in synthetic cases. For this reason own strategies needed to be found to handle all of them. Some solutions are found very late in the

Artifact, Problem	**Description, Occurence**	**Improvement**
incorrect self-shadowing	Occurs when the transformed point (from eye-space to light-space) is not lying right on the surface, especially if z-eye is larger than z-light it causes incorrect self-shadowing.	Front face culling removes the artifacts.
Hard shadows	Hard shadow edges through the binary shadow value.	Use of PCF or PCSS to get shadow value between zero and one.
Staircase/ Jaggies	Low resolution of the depth map near the observer's eye.	Use of cascaded shadow mapping and implementation of a maximum shadow range.
Projection Aliasing	Occurs when shadow receiver is orthogonal to the shadow map plane. Artifacts on back faces of meshes can be seen, which look like shadow acne.	Calculating the dot product of the directional light direction and the normal of the shaded point. If the result is larger or equal to zero, set the shaded point to be in shadow.
Perspective Aliasing	If the distribution of values in the depth map is the same, objects near the eye are under-sampled and objects far away are over-sampled. It is associated with the staircase artifact.	Use of the practical split scheme in cascaded shadow mapping.
Visible cascade change	The cut between two cascades is visible.	Implementation of a blending between the cascades, so the cut cannot be seen any longer.

Table 4.1: Overview of all artifacts and problems occurred during the implementation, with a short description and their solution.

implementation process, i.e., the front face culling. Before front face culling is used, a lot of different values for the OpenGL depth offset value parameter are tried, to fix the incorrect self-shadowing. In addition, many values for a bias instead of the offset parameter are tried.

A big implementation step is the change from standard shadow mapping to cascaded shadow mapping. It took some time to find out, that it is the easiest way to store all the values (i.e., view-matrix, projection-matrix, z-near/z-far) for every cascade in an extra camera, although this means to create eight new camera objects, one for every cascade from the observer's eye and one for every cascade from the light's point of view. Another big challenge is to find an automatic calculation for the orthozoom factor, as this value does not exist in a normal orthographic projection matrix. Some different calculations are tested, combined with some existing methods from the application, but there is always a part of the shadows truncated. In addition, the problem is not always obvious and



SETTINGS: 3D View

Shadows

General

Shadow Map Size	4096
Directional Light Azimut	315
Directional Light Elevation	40
Shadow intensity [%]	80
Shadow intensity for Meshes [%]	100
Trees throw Shadows	☑
Blend between Cascades	☐
Cascades Blend Area	0.05
PCF Kernel Size [m x m] m=	1
Polygon Offset	0 0
Maximum Shadow Range	5000
Shadow Fade-Out Range	500

PCSS

Use PCSS	☑
Light Size	0.4
Number of Samples	16

Figure 4.3: A screenshot of the settings, which the user can change for the shadows in the client.

sometimes only noticeable in few particular views. This led to the assumption that the problem is fixed, while it was still there and appeared again much later. Furthermore, the cascaded shadow mapping algorithm is normally used in perspective projection, so many calculations had to be derived by ourselves to use them for orthographic projection.

4.3.2 Artifacts

As already mentioned there are several artifacts appearing during the implementation. In Table 4.1, an overview of the artifacts can be found, together with a short description of how and why they occur, as well as how they are fixed in the implementation.

4.4 Parameters

The user can change several shadow mapping parameters during run time. In Figure 4.3 the default values for these parameters can be seen. The depth map is always quadratic, so just one value can be set for the size. This value is important, as a too small shadow map size causes more aliasing and a too big size causes poor performance. The azimuth and elevation directional light are defined as degrees and determine the direction of the directional light. Azimuth has a range from 0 to 360° and can be seen as a rotation around the world space up-vector. The elevation angle ranges from 0 to 90° and is specified as the angle over the horizontal plane.

As already described in section 3.3 the user can further set values for: intensity of shadow, shadow range & fade out range, the blend area and all settings for soft shadows with PCF and PCSS. The default value for the intensity is set to 80, because the scene is very dark with 100 percent. The blend area has a default value of five percent, but the blending is only done when the checkbox above it, is checked. The default setting of the blending between cascades is set to false, as the edge between the cascades is mostly not visible in big scenes and therefore, it is not worth the decrease of performance. As soft shadow technique, Percentage Closer Soft Shadows is chosen, since the results regarding the quality are better. PCF would reach better results with using a 9×9 -filter kernel size, but this is again not worth the reduction of performance. The user can select, whether the trees should throw shadows or not. This is implemented, because there is a pre-existing issue with their geometry in the system.

CHAPTER 5

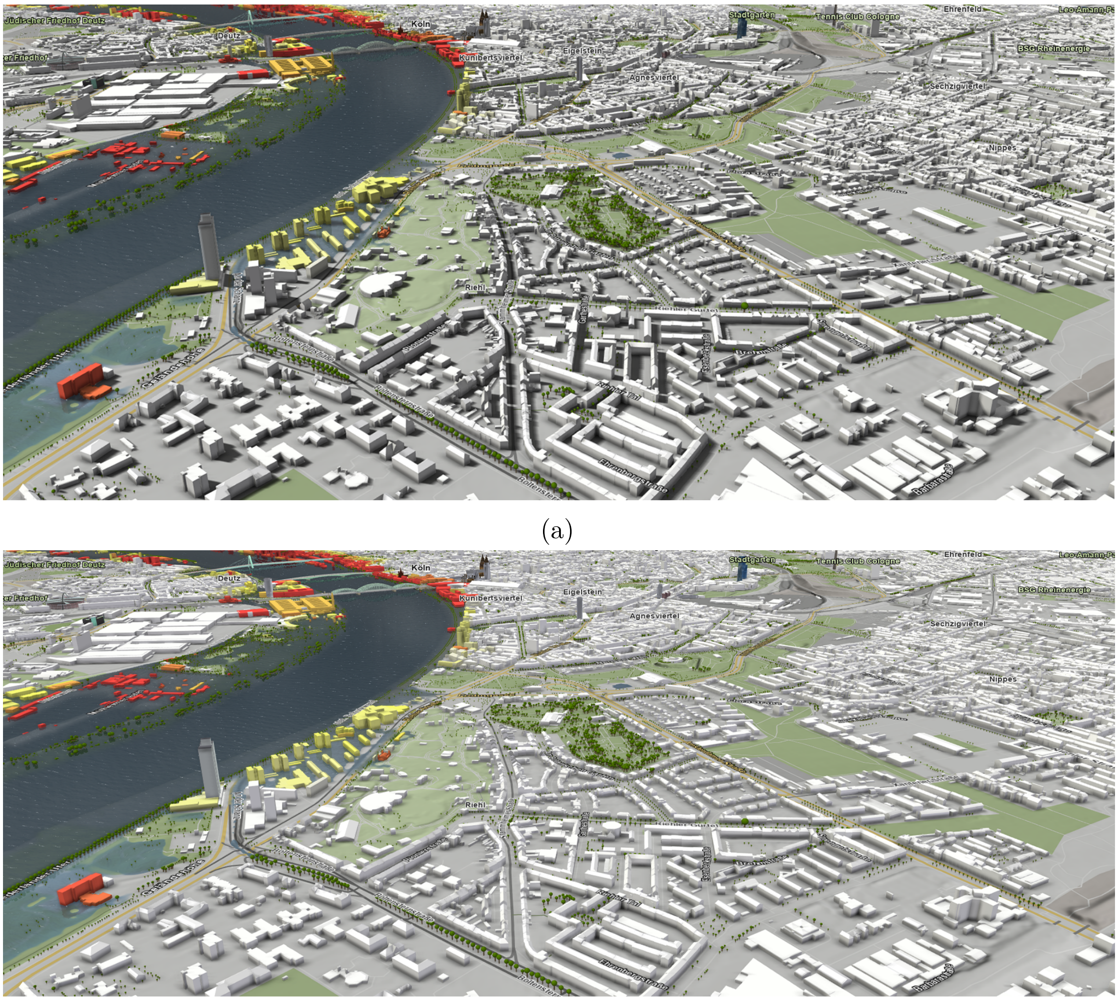
Results and Discussion

In this section, the results of the implemented algorithm are shown and explained. First, the visual results are shown with short descriptions about what can be exactly seen in the resulting images. After this, the performance results are presented which are measured with the same camera settings as in Figure 5.1. In the last section, which is the discussion, a critical review of the results is done.

5.1 Visual Results

The results show the implemented shadow mapping algorithm in the city of Cologne, Germany. This city provides a good opportunity to show the results, as it is a big scene with high buildings, a river, detailed bridges, trees and some special buildings, e.g., the dome. The shadows make the scene much more realistic, especially in a direct comparison. Through cascaded shadow mapping and a limited shadow range, the shadow quality increases a lot. A result of a big scene with and without shadows can be seen in Figure 5.1, where the shadows are restricted to a range of 5 kilometers. Shadows are cast by trees and various kinds of meshes. That includes objects that are dynamically added at run time, e.g., walls and sandbag barriers, which are used to save endangered buildings from the floods. As soon as they are added to the scene, the shadow appears. The terrain receives shadows, but does not cast shadows itself. This could be an add-on in future work.

The algorithm shows objects with little details very accurately, e.g., one of the bridges which can be seen in Figure 5.2a. It is visible that the bridge is built out of many thin metal bars. Further, you can see in this figure the shadow of the bridge in the water twice. This happens as one shadow is on the ground of the water (on the terrain), and the other one is on the surface of the water. In Figure 5.2b trees with their shadows are shown. In the figure a dynamic floodwall can be seen, which is inserted next to the river



(a)

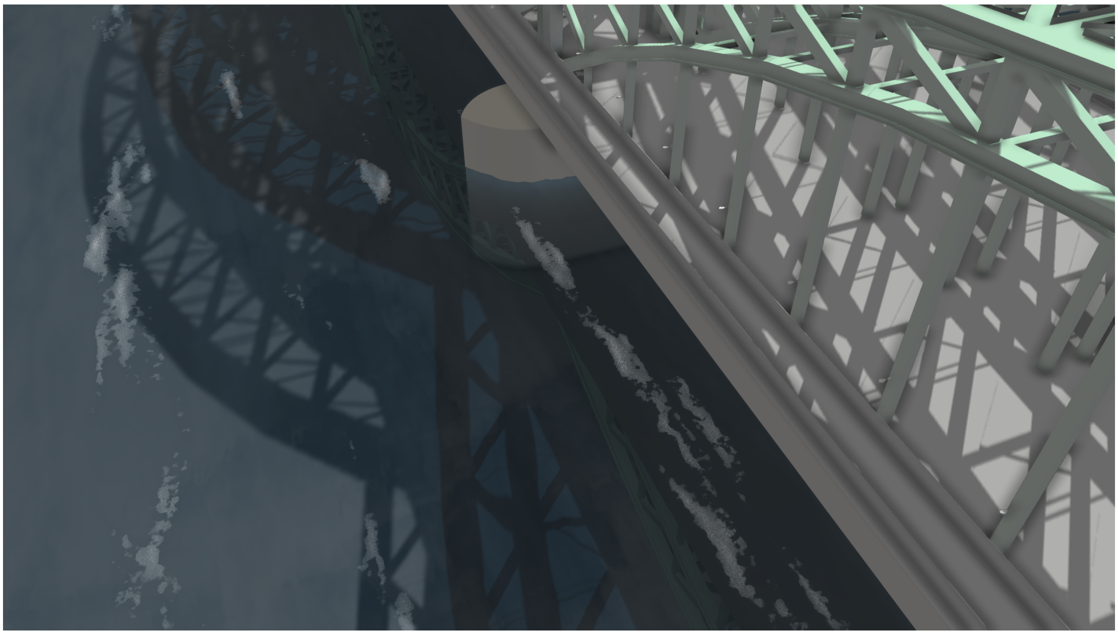
(b)

Figure 5.1: (a) Shadow range of 5 km and fade out range of 500 m. (b) No shadows.

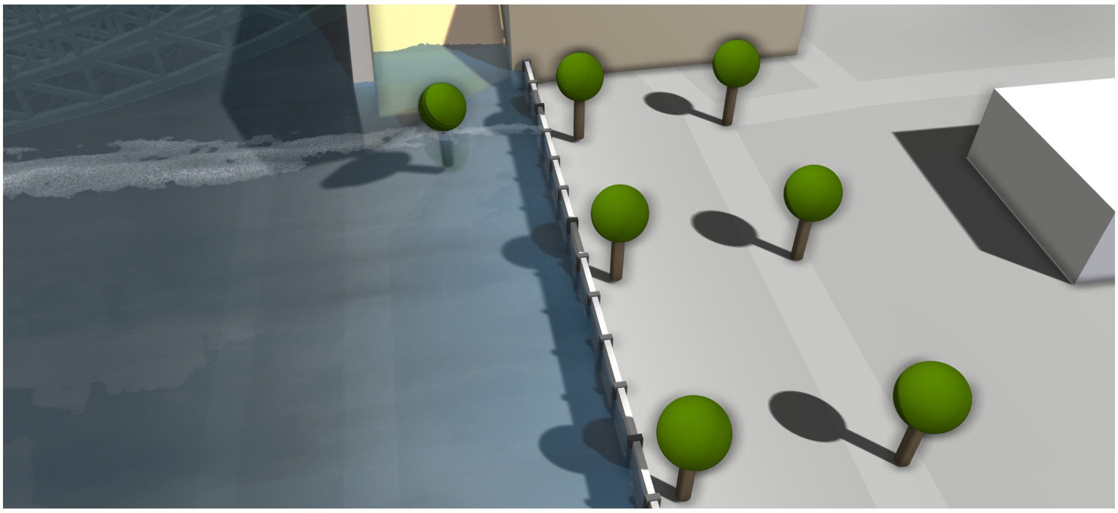
to protect the buildings around it. Beside this, the figure shows again the double shadow in the water area.

5.2 Performance Results

The computer used for testing the implementation has an integrated Intel Core i5-4690K processor with 3.50GHz and a memory of 32 GB RAM. As graphic board, the NVIDIA GeForce GTX 1070 is used. The operating system is Microsoft Windows 10 x64. The elapsed time on the GPU is measured with a pipeline statistics query from the OpenGL API. Performance tests are done for both render passes. The measured time of the first render pass (the depth map creation), includes the change of the viewport and culling,



(a)

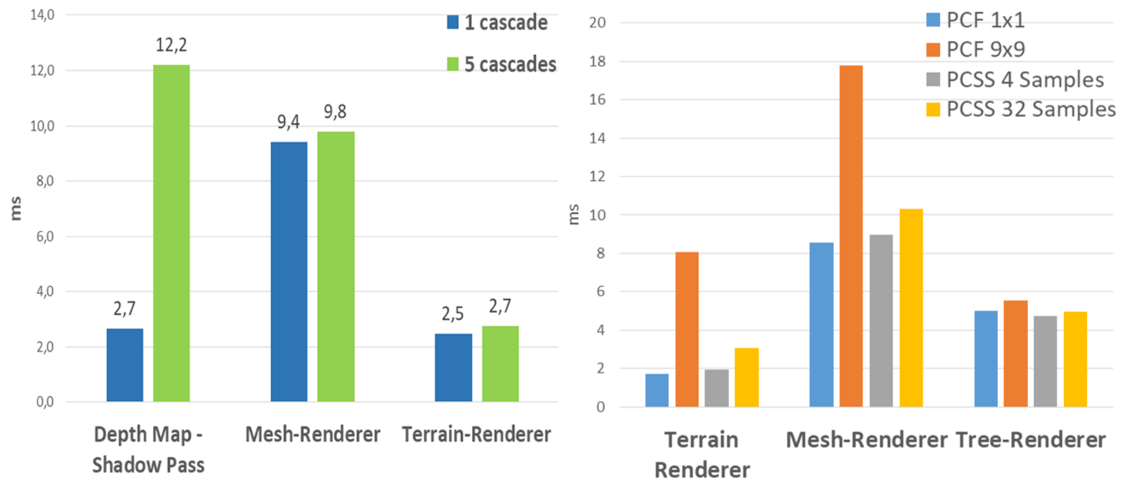


(b)

Figure 5.2: (a) A bridge with many shadow details. (b) Shadow of trees and double shadow in the water.

the binding of the program where all parameters are set and the rendering of the depth map itself. In all render classes (terrain, mesh and tree), the performance results of the shading pass include just the drawing of the elements, which is the actual rendering.

An interesting result shows the difference between the use of one and five cascades. This calculation includes the depth map generation as well as the shading pass for the meshes



(a) (b)

Figure 5.3: (a) A diagram, which shows the different performance for using one or five cascades. The diagram includes both render passes. (b) A diagram, which shows the performance for the use of different soft shading algorithms with different filter sizes and samples, measured in the shading pass for the terrain, the meshes and trees.

and for the terrain. As can be seen in Figure 5.3a, there is not a big difference in the elapsed time between the amount of cascades for the second pass, but in the shadow map creation the value for five cascades is more than four times as high as for just one cascade. This is expected, as the scene geometry has to be rendered four more times. The diagram, shown in Figure 5.3b, shows the elapsed time (y-axis) of the mesh, terrain and tree renderers (y-axis) when different soft shading techniques are used. The best looking results are gotten with the use of the PCF technique with a 9×9 filter size, but as can be seen in the figure the elapsed time to render this, is on average much higher than for the filter size 1×1 or for the other techniques. There is just a minimal difference between PCF 1×1 filter size and PCSS with four samples, although a PCF size of 1×1 means there is no soft shading done. With increasing size of the samples in PCSS, also, the elapsed time is increasing, but in proportion the raise is very low.

Another criterion, which influences the shadow quality, is the size of the depth map. The larger the size, the better is the resolution of the shadow. However, as can be seen in Figure 5.4 the elapsed time for the creation of the depth map is also increasing depending on the size. In our application, we set a shadow map size of 4096×4096 as default, as the proportion between performance and shadow quality is the best. A result of a comparison between a shadow map size of 1024 and 4096 can be seen in Figure 5.5.

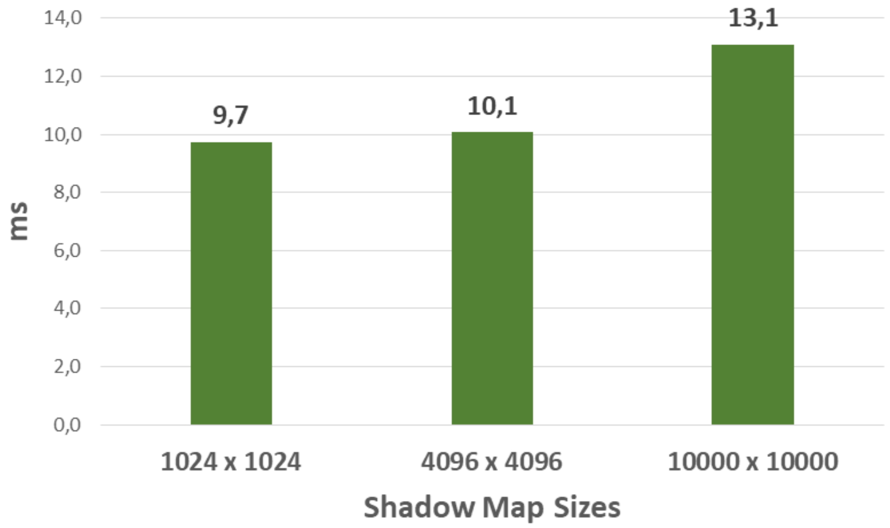


Figure 5.4: A diagram, which shows the elapsed time of the depth map generation for different shadow map sizes.

5.3 Discussion

A difficult point in the shadow quality is that it depends much on the direction of the scene camera. When a scene is viewed top-down, the viewport includes just a few objects, and therefore just a few objects need to be rendered in the depth map. However, if the scene is viewed horizontally, the viewport of the camera can contain the whole scene in worst case. This leads to the problem that the entire scene needs to be rendered into the depth map and the positions near the camera are extremely under-sampled. Many improvements for this problem are added to the implementation, which are CSM, a practical split scheme in CSM and at least a maximum range for the shadow. An example of such a horizontally viewed scene can be seen in Figure 5.5a. In this view, CSM is already used, but staircases in the shadow are visible, as no shadow range was used and the depth map has a resolution of 1024, which is too small for this scene view. With the usage of a maximum shadow range or a higher resolution of the depth maps, the result is much better, i.e., in Figure 5.5b where a shadow map size of 4096 is used.

All mentioned improvements are very helpful to overcome the problem, but it is still visible that the shadow quality in a horizontal view is worse than in a top-down view. Although it is visible, the improvements decreased this artifact to a minimum and therefore it is not annoying any longer. To implement the feature of a maximum shadow range is helpful in many ways. It improves the shadow quality and a little bit the performance when terrain and meshes are rendered. Nevertheless, the improvement has a disadvantage one has to accept. In a top-down view, it can happen for tall buildings that shadows are still visible on the roof, but not on the ground.

One of the two soft shading techniques should always be used when rendering shadows,

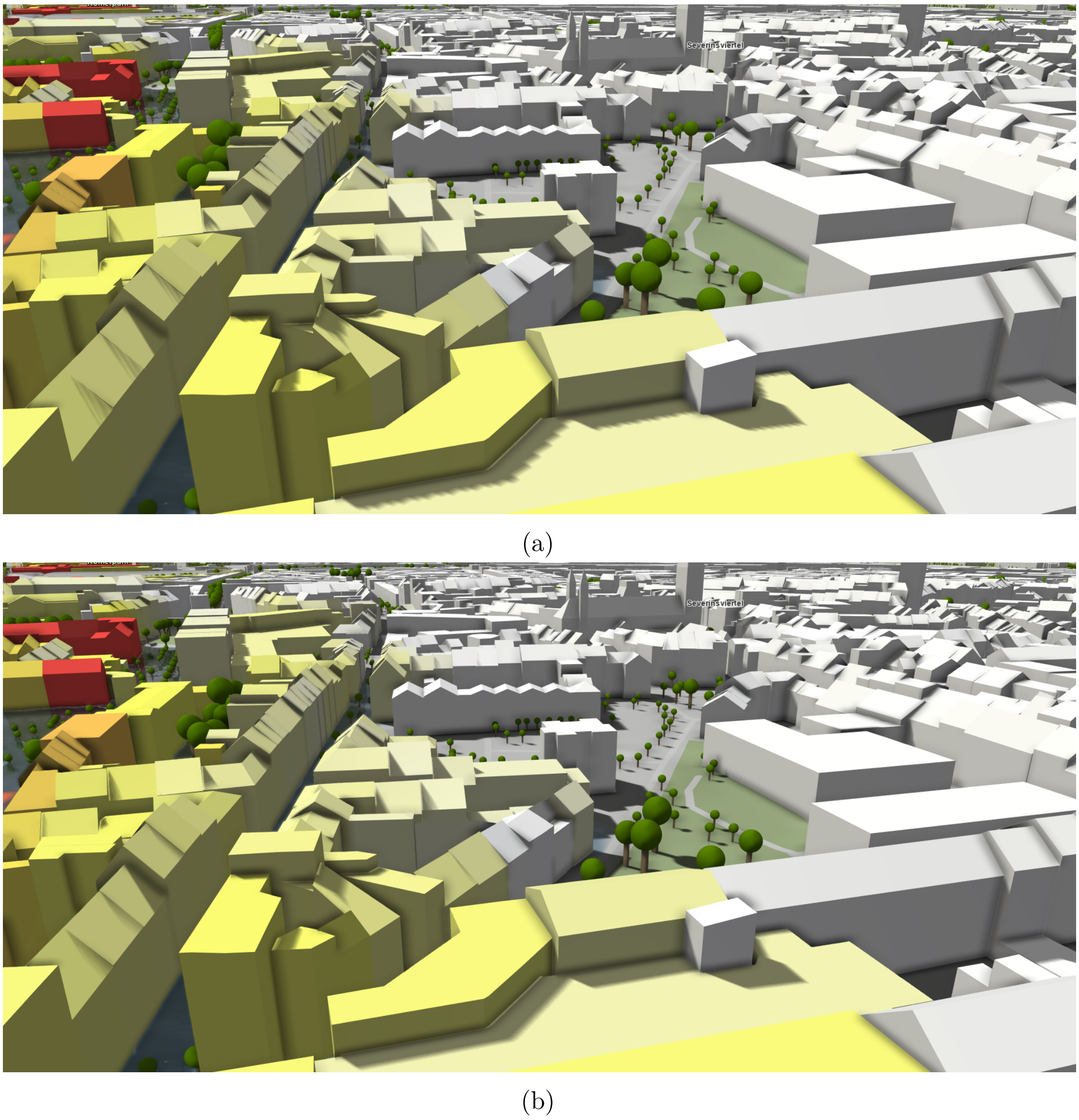


Figure 5.5: A horizontally viewed scene. (a) A shadow map size of 1024, which causes artifacts. (b) Much better results with shadow map size of 4096.

at which PCSS has a better proportion between performance and shadow quality. PCF with a filter size of 9×9 is the best choice for screenshots, since it gives the best looking results, but as described in 5.2 it has a too low performance to use it as default filtering technique.

The default number of cascades is set to four, out of five possible cascades. The number of cascades can be changed in the code, but there does not exist a scheme what number of splits returns the best results. As already showed in Figure 3.3a, the use of one depth map for the whole scene returns shadow with staircases, which is no acceptable shadow quality. When using two cascades for the whole scene the result looks very similar. We found out that four cascades give a good-looking shadow and do not reduce the performance too much.

CHAPTER 6

Conclusion

This thesis covers the implementation of real-time shadows for large-scale geospatial visualization. An adapted version of the cascaded shadow maps technique is used to reach a good quality of the shadows and at interactive frame rates. Several improvements helped to reach the desired quality and a good performance. They further removed occurring artifacts. Examples of these artifacts are the incorrect self-shadowing, fixed with the use of front face culling, perspective aliasing, fixed with the use of a specific split scheme in CSM, and a visible cascade transition, which is solved by blending between the cascades. As shadow mapping produces hard shadows, we integrated two soft shadow techniques, which are PCF and PCSS, where PCSS is preferred as the filter size and therefore the degree of softness varies intelligently according to the surroundings. The result is a flexible visualization of real-time shadows, which improve the realism of the scene. An add-on that should be done in future work is casting shadows of the terrain. The used technique for shadow calculation is easy to extend in further renderer classes, which may be added in the application in future work and should also receive shadows.

Bibliography

[Bav08] Louis Bavoil. Advanced soft shadow mapping techniques. `http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_SoftShadowMapping.pdf`, February 2008. Seen on 30.01.2019.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA, 1977. ACM.

[Dep15] Depth precision. `http://dev.theomader.com/depth-precision/`, May 2015. Seen on 01.02.2019.

[Dim07] Rouslan Dimitrov. Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*, 2007.

[dV14] Joey de Vries. `https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping/`, 2014. Seen on 13.02.2019.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[FJ17] Margarita N Favorskaya and Lakhmi C Jain. Large scene rendering. In *Handbook on Advances in Remote Sensing and Geographic Information Systems*, chapter 9, pages 281–320. Springer, 2017.

[KSA+15] Hoshang Kolivand, Mohd Shahrizal Sunar, Ayman Altameem, Amjad Rehman, and Mueen Uddin. Shadow mapping algorithms: applications and limitations. *Applied Mathematics & Information Sciences*, 9(3):1307, 2015.

[Lea14] Craig Leach. failengine. `http://www.leachgamedev.com/failengine/`, 2014. Seen on 01.02.2019.

[Lic14] Petra Lichtenberger. Century floodwater in 2013 in austria: one year later. *GMX magazine*, 2014.

[LTY+06] Brandon Lloyd, David Tuft, Sung-eui Yoon, Dinesh Manocha, et al. Warping and partitioning for low error shadow maps. In *Rendering Techniques*, pages 215–226, 2006.

[LX11] Lu Liu and Shuangjiu Xiao. Real-time soft shadows for large-scale virtual environments. In *2011 International Conference on Multimedia Technology*, pages 5464–5467, July 2011.

[Ngu07] Hubert Nguyen. *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’02, pages 557–562, New York, NY, USA, 2002. ACM.

[SWP11] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. *Computer Graphics Forum*, 30(1):169–186, 2011.

[TQJN99] Tadamura, Xueying Qin, Guofang Jiao, and Nakamae. Rendering optimal solar shadows using plural sunlight depth buffers. In *1999 Proceedings Computer Graphics International*, pages 166–173, June 1999.

[vis] Visdom. `http://www.visdom.at/`. Seen on 26.01.2019.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’78, pages 270–274, New York, NY, USA, 1978. ACM.

[Wim17] Michael Wimmer. Shadows. `https://www.cg.tuwien.ac.at/courses/Realtime/lectures/VU.WS.2016/index.html`, 2017. Seen on 13.02.2019.

[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. ACM.