Karlsruhe Institute of Technology

Moment-Based Cascade Splits for Sample Distribution Shadow Maps

Bachelorarbeit von

Alexander Schipek

An der Fakultät für Informatik
Institut für Visualisierung und Datenanalyse,
Lehrstuhl für Computergrafik

September 10, 2020

Erstgutachter:	Prof. Dr.-Ing. Carsten Dachsbacher
Zweitgutachter:	Prof. Dr. Hartmut Prautzsch
Betreuender Mitarbeiter:	Dr. rer. nat. Christoph Peters

Contents

Abstract

English

Shadow computation is important for real-time rendering. Without shadows many scenes would look rather boring and unrealistic. Computing realistic shadows is not easy, especially for real-time applications. Therefore the technique we want to improve on is the real-time shadow technique shadow mapping, more precisely cascaded shadow mapping for parallel global light sources. Cascaded shadow mapping tries to improve the effective resolution by splitting the view frustum in ranges of depth, each assigned to their own shadow map, that encloses said range. These ranges of depth are thus called cascades. In order to achieve an improvement in visual quality, at the cost of manageable overhead, we introduce a technique to compute the split positions for cascaded shadow mapping for four cascades. Non-adaptive split computation suffers the problem of leading to bad results for many scenes. We want to improve this by considering the depth distribution of the camera, thus falling into the category of sample distribution shadow maps. The depth distribution consists of the depth buffer values of the camera and provides us with a spatial knowledge about the geometry of the scene. Adaptive split computation methods, like finding the minimum and maximum of the scene depth and redefining near and far plane, are better than non-adaptive methods but still suffer from problems like being slow or generating empty cascades. In order to improve on these problems, we use a construct from stochastics called moments. The n-th power moment consists of the sum over all samples of a given distribution, taken to the power of n and divided by the number of samples. We use eight power moments in order to estimate the given depth distribution, resulting in a proper partition of the view frustum. In order to compute the moments, we propose a reduce shader. The main goal of this technique is to find depth intervals or clusters, where all values are adjacent. Placing a split between two of these intervals is a good idea, because, if we also use bounding box fitting in order to minimize the size of the generated cascades, then we neither end up with cascades being inflated excessively, nor with cascades being empty. In order to do that, we calculate the moment-based reconstruction of the cumulative histogram. This reconstruction acts like a smooth density estimation. Calculating the splits consists of calculating the minima found in the gradient of the reconstruction. We approximate this gradient by using moments, resulting in the reciprocal of a polynomial. In order to find the minima we just have to find the maxima of this polynomial. This consists of finding the roots of the derivative of the polynomial and checking for maxima. For finding these roots we use Laguerre's method [Pre07]. Should we not find enough minima, in order to set three splits, we will set splits between the cascades generated by the already found splits, such that the maximum of all approximated screen areas for the new cascades becomes minimal. We thus try to distribute the screen evenly among the cascades, under the consideration, that we use any minima we found as a split. We recommend warping the depths before that computation in order to minimize the impact of the perspective projection. Excluding pixels that look at no geometry, for instance, is achieved with a binary frame buffer texture that stores the validity of a pixel in the color channel. For the shadow computation we use percentage closer filtering with a 3×3 kernel and bilinear interpolation.

Deutsch

Schattenberechnung ist wichtig für Echtzeit-Rendering. Ohne Schatten würden viele Szenen langweilig und unrealistisch aussehen. Realistische Schatten zu berechnen ist nicht einfach. Das gilt vor allem für Echtzeitanwendungen. Deshalb versucht unsere Technik eine Verbesserung der Echtzeit-Schatten-Technik Shadow Mapping, genauer Cascaded Shadow Mapping für parallele globale Lichtquellen, zu bieten. Cascaded Shadow Mapping versucht die effektive Auflösung durch das Aufspalten des Sichtkegels der Kamera in Tiefenintervalle, welche ihre eigene Shadow Map zugewiesen bekommen, zu erhöhen. Diese Tiefenintervalle nennt man dann Kaskaden. Um eine Verbesserung der visuellen Qualität zu erreichen, entwickeln wir eine Technik zur Split-Punkt-Berechnung für Cascaded Shadow Mapping für vier Splits. Diese Technik darf nur einen überschaubaren Anteil an Zusatzaufwand benötigen. Nicht-adaptive Split-Berechnung leidet an der Tatsache, dass es viele Szenen gibt, in denen die Splits schlecht sind. Wir wollen das verbessern, indem wir die Tiefenverteilung der Kamera berücksichtigen. Daher fällt unsere Methode in die Kategorie Sample Distribution Shadow Map. Die Tiefenverteilung besteht aus den Werten des Tiefenpuffers der Kamera, die uns räumliches Wissen über die Geometrie der Szene zur Verfügung stellen. Adaptive Split-Berechnungen, wie das Finden der minimalen und maximalen Szenentiefe und Umdefinieren der Near- und Far-Ebenen, sind besser als nicht-adaptive Methoden. Allerdings leiden sie immer noch an Problemen, wie langsamer Berechnung oder leeren Kaskaden. Um diese Probleme zu verbessern, verwenden wir ein Konstrukt aus der Stochastik, namens Momente. Das n-te Moment besteht aus der Summe über alle Stichproben der Verteilung hoch n und geteilt durch die Anzahl an Stichproben. Wir verwenden acht Momente, um die Tiefenverteilung zu approximieren. Das resultiert dann in einer hinreichend guten Aufspaltung des Sichtkegels der Kamera. Um die Momente zu berechnen, schlagen wir einen Reduce-Shader vor. Das Ziel dieser Technik ist es, Tiefenintervalle oder Cluster zu finden, in denen die Tiefen nah beisammen sind. Das Platzieren eines Splits zwischen zwei dieser Intervalle ist eine gute Idee, weil wir dann, unter der Verwendung von Bounding Box Fitting zur Minimierung der Kaskadengröße, weder unnötig große, noch leere Kaskaden bekommen. Um das zu erreichen, berechnen wir die momenten-basierete Rekonstruktion des kumulativen Histogramms. Diese Rekonstruktion verhält sich wie eine geglättete Dichtenschätzung. Um nun die Splits zu erhalten, gilt es, die Minima des Gradienten dieser Rekonstruktion zu berechnen. Wir approximieren diesen Gradienten mit den Momenten, was in dem Kehrwert eines Polynoms resultiert. Um die Minima zu errechnen, müssen wir nur noch die Nullstellen der Ableitung dieses Polynoms berechnen und herausfinden, ob diese Maxima sind. Zur Berechnung der Nullstellen verwenden wir Laguerre's Methode [Pre07]. Sollten wir nicht genügend Minima finden, um drei Splits zu setzen, so setzen wir Splits zwischen bereits generierten Kaskaden, sodass das Maximum aller approximierten Bildschirmflächen der neuen Kaskaden minimal wird. Dementsprechend versuchen wir den Bildschirm so gleichmäßig wie möglich auf die Kaskaden aufzuteilen, unter der Berücksichtigung, dass wir alle Minima als Splits nehmen, die wir finden. Wir empfehlen es, die Tiefen vor der Berechnung zu modifizieren, um den Einfluss der perspektivischen Projektion zu minimieren. Das Auslassen von Pixeln, die beispielsweise keine Geometrie beinhalten, ist durch eine binäre Frame Buffer Textur gewährleistet, die die Validität des Pixels im Farbkanal speichert. Für die Schattenberechnung verwenden wir Percentage Closer Filtering mit einem 3×3 Kern und bilinearer Interpolation.

1. Introduction

The realism of virtual environments depends on good shadows. Without shadows, the three dimensional projection of the environment is harder to comprehend for the user, as shadows provide depth and structure to the scene. For instance, if we have a shadow from a global light source, we are able to estimate the position and shape of the object causing it without seeing that object explicitly. Distant geometry is easier perceived as such, if the scene provides shadows. These shadows should also be as detailed as possible in order to be more realistic.

In many real-time applications shadows are computed with shadow maps. Shadow maps store the depth buffer, seen from the perspective of a light source and use that to compare it with the transformed depth of a surface point. If the transformed depth is greater than the depth in the buffer, then we have a shadow at that surface point.

We can always raise the resolution of the shadow map, but doing so is expensive. That is why we are looking for an alternative to raise the effective resolution of shadow maps.

Looking at global parallel light sources, like the sun, we encounter the problem that the resolution of a shadow map is limited. In order to improve that we use cascaded shadow maps.

Cascaded shadow maps try to improve the effective resolution of shadow maps by splitting the view frustum in intervals of depth. Each interval then gets its own shadow map. These intervals are thus called cascades.

We improve the effective resolution further by minimizing the bounding box size of these shadow maps. This is accomplished by bounding box fitting. Bounding box fitting consists of a compute shader that seeks the minimum and maximum coordinate for every axis in the cascade. With these values the enclosing bounding box is cropped to a minimal size.

The effective resolution depends now directly on the split positions. Therefore we have to compute the best split positions in order to get the best results.

Methods that do not adapt to the current scene lead to bad splits for many scenes. That means that they are not of interest for interactive applications.

Adaptive methods that evaluate the depth buffer of the camera, in order to work with the distribution arising from it, fall into the category of sample distribution shadow maps.

One example would be to find the minimum and maximum value of depth and redefining the near and far plane for the split computation. This method is, generally speaking, better than methods that do not adapt but it also suffers from two major problems.

First, it could be the case that all cascades but two (the first and last cascade), become obsolete because they only enclose empty space.

Second, it could lead to cascades being inflated by empty space. For instance, we have a scene with a balcony and look down to the street. If we are unlucky, one cascade could be exactly between balcony and street, and contains pixels of both. This would lead to a big cascade enclosing empty space with the exception of these few pixels at the borders. Therefore these pixels have a poor effective resolution resulting in bad shadows.

Bounding box fitting is not able to improve any of these two cases much because the cascade is either empty or enclosed with geometry but mostly empty. This leads to either the loss of the shadow map or, worse, bad shadows for pixels at the border between empty space and geometry.

Our approach solves these problems by estimating the depth distribution of the camera with moments. Moments are stochastic constructs used to describe and reconstruct distributions. The n-th moments of a distribution is the sum over all samples of that distribution taken to the power of n and divided by the number of samples.

We compute the points where least geometry gets added, for instance a big depth interval containing empty space, by using moments. These points are good positions for splits because they automatically partition the depth in intervals of adjacent geometry, so the bounding box fitting minimizes them properly.

We find these splits by setting up the moment-based reconstruction of the cumulative distribution. This reconstruction acts as a smoothened density estimation. The big empty spaces manifest themselves as minima in the gradient of said reconstruction.

From our moments we get the gradient as the reciprocal of a polynomial. We thus seek for maxima of the polynomial in order to find the minima of the gradient. Finding maxima for a polynomial includes seeking roots of said polynomial. Therefore we use Laguerre's method [Pre07]. Our method thus guarantees that no cascades are left unused due to a poor placing of splits.

Even if the number of these intervals is less than the number of splits we need, we get good splits too. In this case, we try to minimize the approximated area of visible geometry per split by placing the remaining splits in one or more of these intervals, such that the maximal area of the resulting cascades becomes minimal.

In order to do that, we approximate the area by using our reconstruction again. The reconstruction describes the lower and upper bound of all distributions that have the same moments. If we take the average, we have a sufficiently good approximation of the cumulative histogram, thus we use it to compute the approximate screen areas. We now just have to invert that function in order to place our remaining splits. We invert the reconstruction by using a method called bisection [Pre07, p. 449].

By doing so, we achieve good looking close-ups of character models with detailed shadows without losing too much of the important background detail or adding resources for that matter alone (see Figure 1.1).

This alone might be interesting for real-time applications like games that show many close-ups of characters.

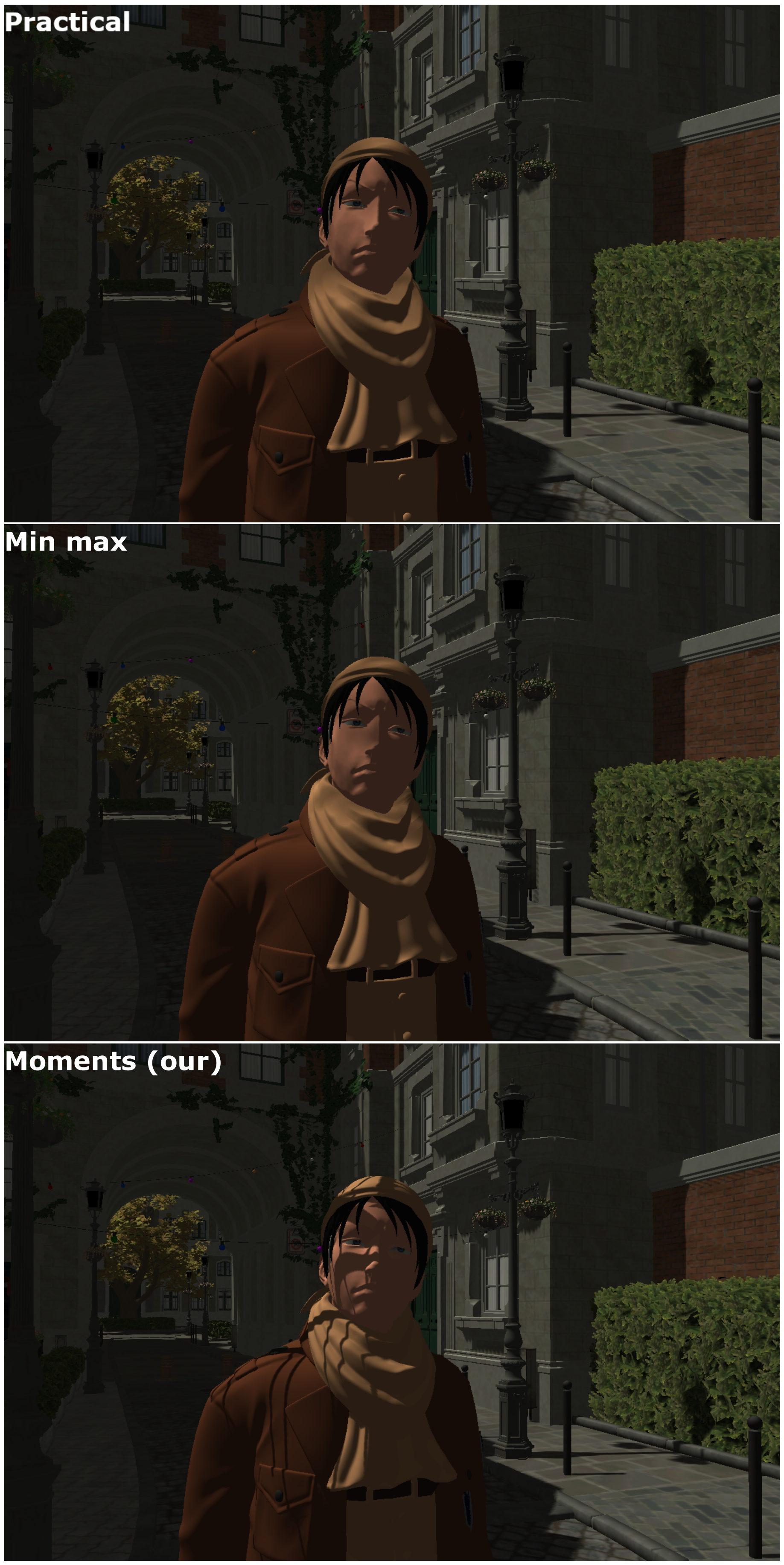


Figure 1.1: A close-up of a character model in a scene, rendered with three different schemes of split computation.

2. Background

This chapter explains the basic methods used to produce shadows generated by a parallel light source. Additionally it explains the idea of an efficient use of allocated resources for shadow computation. First we introduce shadow mapping in order to understand the basic computation method and aliasing arising from it. After that we discuss cascaded shadow mapping. Therefore we also explain some split techniques, cascade overlapping and bounding box fitting. Last we explain the concept of moments in order to establish fundamental knowledge, needed for upcoming chapters.

2.1 Shadow Mapping

In order to compute shadows we want to know how much light is visible at a given surface point p. Therefore we define a visibility function $V(p)$ like:

$$V(p) = \begin{cases} 0 & \text{if the line segment between } p \text{ and the light source is occluded by geometry} \\ 1 & \text{else} \end{cases} \tag{2.1}$$

This equation can be solved by casting a ray from the position of the light in direction of the surface point we want to shade. If we take the nearest intersection point $p_{intersect}$, we get the following equation:

$$V(p) = \begin{cases} 0 & \text{if } \|p_{intersect} - p_{lightsource}\| < \|p - p_{lightsource}\| \\ 1 & \text{else} \end{cases} \tag{2.2}$$

In order to position the camera and define the view frustum, we have to calculate the view and projection matrices. These matrices are used to transform a point from model space to view and then to projection space. If we limit our attention to spot lights with an opening angle of less than 180 degrees or parallel lights, we already have all the information needed to compute the matrices for the light source. We thus use the depth of the surface points seen from the perspective of the light instead of their distance to the light.

This is a common problematic solved by a depth buffer, first introduced by Wolfgang Straßer [Str74, c. 6 p. 1]. The depth buffer is a texture, rendered from a given perspective that stores the depth values of the scene. If we use a depth texture as shown in Figure 2.1a from the perspective of the light source, we get a picture like Figure 2.1b.

We now have to compute the depth of the surface point seen from the light source. Therefore we need to transform the given coordinates of the point into the system of the light source.

$$\begin{aligned} p_{ws} &= Mp \\ p_{ls} &= PVp_{ws} \end{aligned} \tag{2.3}$$

Where p is the position of the point and M is the so called model matrix of the scene, transforming points from model space to world space. Multiplying these two, results in the world space position p_{ws} of the point. V is the view matrix and P the projection matrix of the light source. By multiplying them with the world space position we get the light space position p_{ls}. We rearrange the equation like this:

$$p_{ls} = PVMp = A_{MVP}p \tag{2.4}$$

This equation shows us that we only have to perform one matrix multiplication on the GPU with a pre-multiplied matrix A_{MVP}. By doing so we get the homogeneous coordinates of the point in light space. We now have to get the u- and v-coordinates in order to make a lookup in the shadow map. We also need the z-component, describing the depth of the point seen from the perspective of the light:

$$p_{uvz} = \frac{\frac{p_{ls}.xyz}{p_{ls}.w} + 1}{2} \tag{2.5}$$

We now just have to check whether the z-coordinate is greater than the value of the depth texture at the given position, in order to solve Equation (2.2), hence the name shadow map [Wil78], for the texture.

$$V(p_{uvz}) = \begin{cases} 0 & \text{if } T(p_{uvz}.u, p_{uvz}.v) < p_{uvz}.z \\ 1 & \text{else} \end{cases} \tag{2.6}$$

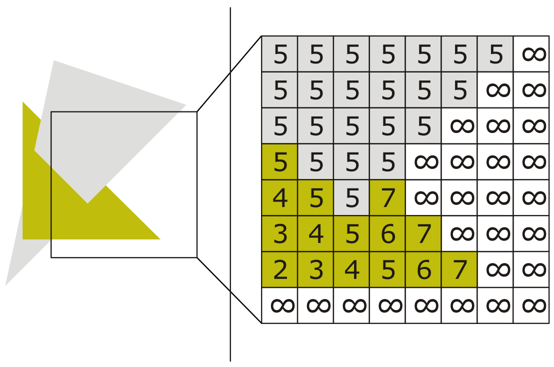
Where $T(p_{uvz}.u, p_{uvz}.v)$ represents the shadow map lookup at texture coordinate $p_{uvz}.u$ and $p_{uvz}.v$.

With such an implementation we encounter a problem called shadow acne (Figure 2.2a). The problem consists of the light source being tilted. Because of the fixed resolution of the shadow map, the position of the middle of these texels has to be mapped on the continuous geometry discretely.

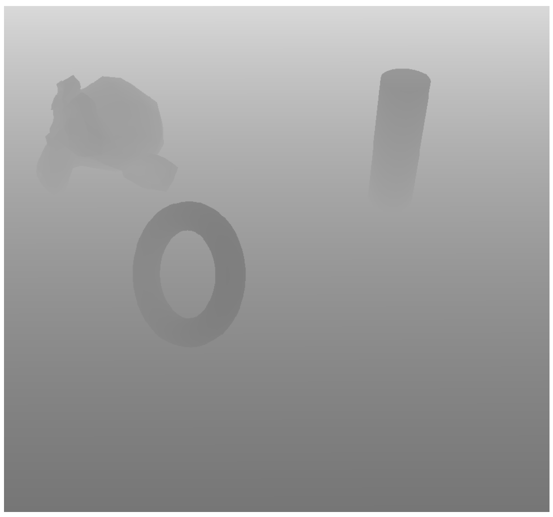
Figure 2.2c shows that these texels are not representing the ground properly. Positions where the texels are above the ground lead to a shadow because the depth of the geometry is higher than the depth stored in the corresponding texel.

In order to fix this, we add a shadow bias that submerges the entire texels in the geometry, as illustrated in Figure 2.2c, which results in Figure 2.2b. The new and final equation with shadow bias s_{bias} is:

$$V(p_{uvz}) = \begin{cases} 0 & \text{if } T(p_{uvz}.u, p_{uvz}.v) + s_{bias} < p_{uvz}.z \\ 1 & \text{else} \end{cases} \tag{2.7}$$



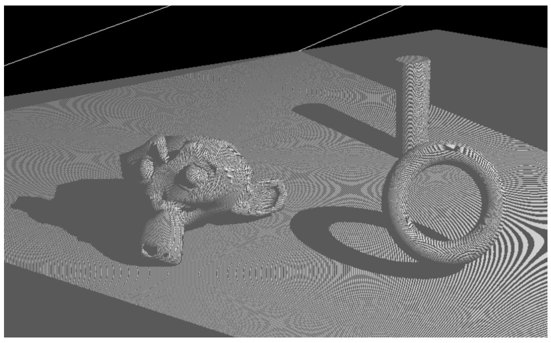
(a) `https://commons.wikimedia.org/wiki/File:Z-buffer.svg`

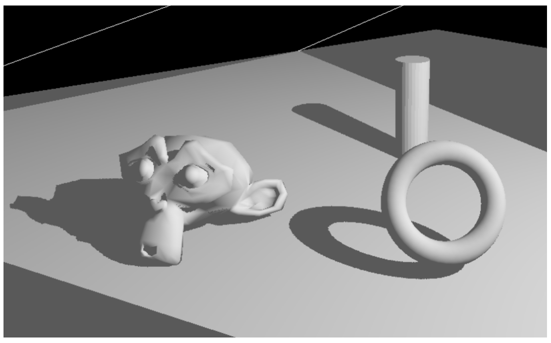


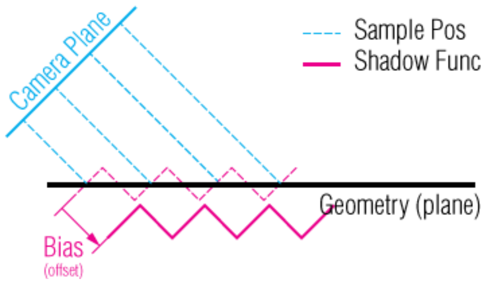
(b) Source: Computer Graphics exercise 7 2019 KIT

Figure 2.1: (a) Depth buffer texture with two triangles.
(b) Shadow map (white = far, black = near)

Shadow mapping can be used for any kind of point light source. Even omnidirectional lights can be computed by six shadow maps, generating a cube map around the light source. In this thesis we explicitly look at parallel light sources only.


(a) Source: Computer Graphics exercise 7 2019 KIT


(b) Source: Computer Graphics exercise 7 2019 KIT



(c) https://i.stack.imgur.com/SyZdy.png

Figure 2.2: (a) Shadow acne
(b) After adding a shadow bias
(c) Reason for shadow acne

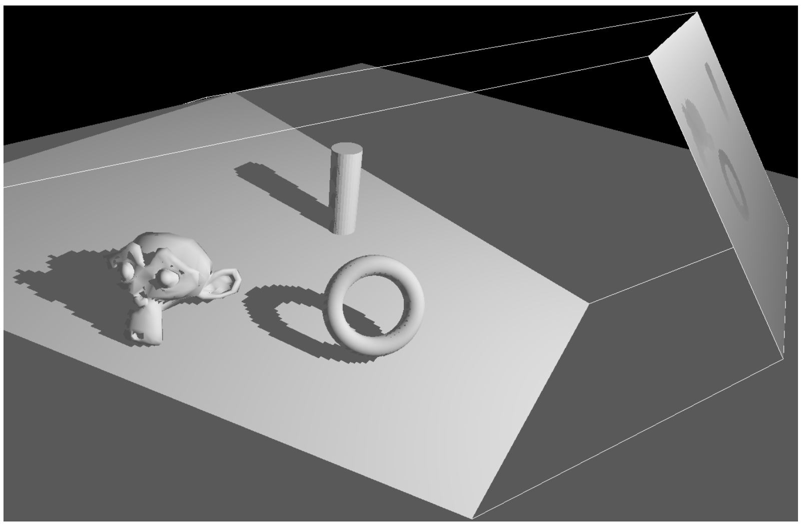
2.1.1 Aliasing

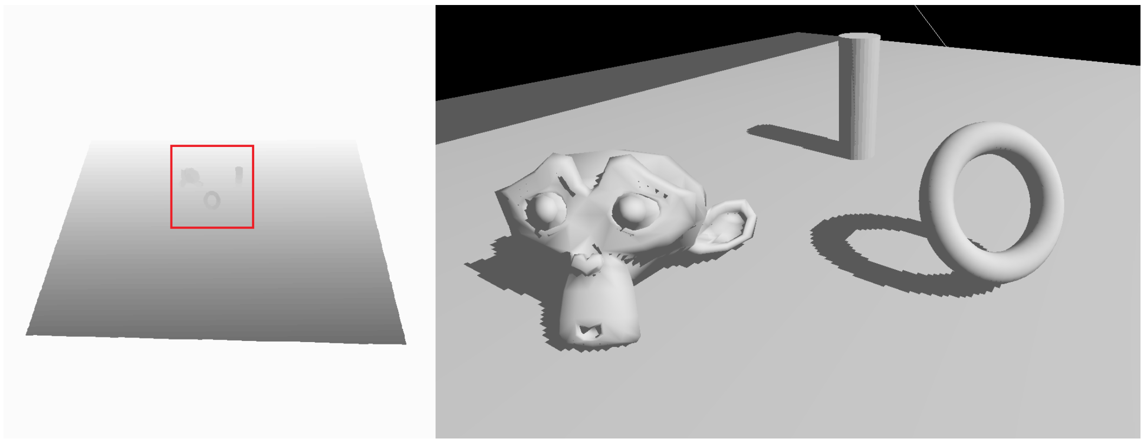
The quality of the resulting picture is determined by the resolution of the shadow map and the position and form of the frustum generating it. If we change the resolution from 1024×1024 to 128×128, we can see the problem resulting from a lower resolution (Figure 2.3a).

The problem is a result of magnification. Many pixels are mapped to the same texel, thus resulting in jagged edges.

This can also be achieved by positioning and forming the frustum of the light source in such ways that the effective resolution becomes 128×128. As seen in Figure 2.3b the resulting image has the same problem as if it had a lower resolution for the shadow map (Figure 2.3a).

Another way to achieve something similar is to use a flat light source, for instance a flashlight we put on the floor. As shown in Figure 2.3c there is aliasing even though the resolution is 1024×1024 and the frustum includes mainly the needed content.


(a) Shadow map resolution of 128×128


(b) Left: shadow map (contrast is a bit higher and the red box indicates the old content of the shadow map), Right: result

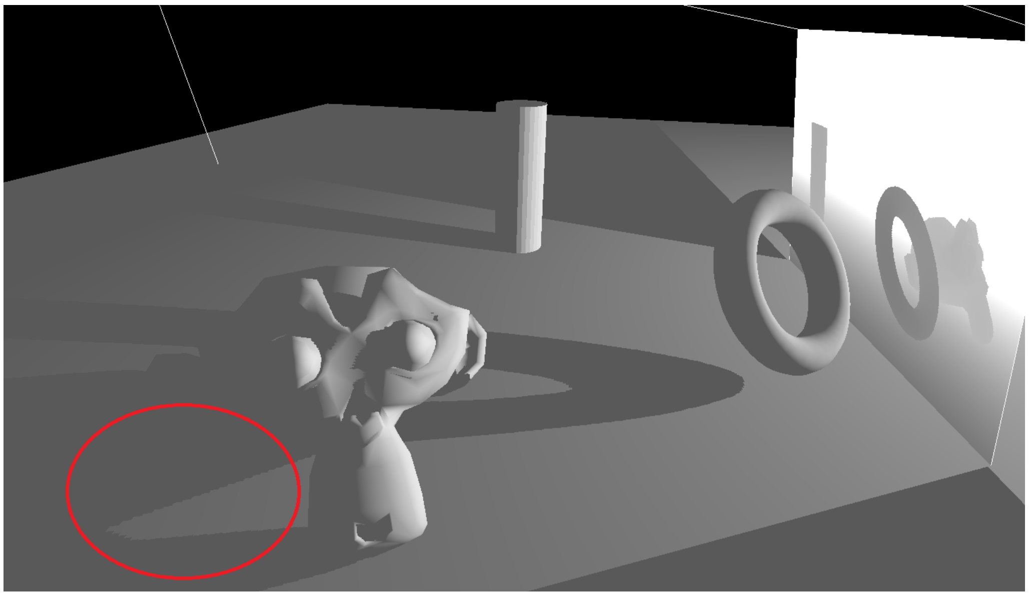

(c) Aliasing caused by a flat light source, the red circle indicates aliasing

Figure 2.3: Source: Computer Graphics exercise 7 2019 KIT

2.1.2 Percentage Closer Filtering

In order to counter some aliasing we use a technique called percentage closer filtering [RSC87]. Percentage closer filtering works as follows:

For every pixel we look at a kernel region with a size of N. In this region we want to compute the average of all pixel values, being one if they are in light and zero if they are in shade. It is important that we take the average of the binary result (shadow or not) and not the average of the depths. If we took the average depth, we would end up with wrong shadows arising from averaged geometry. If we use percentage closer filtering correctly we get a result like Figure 2.5.

Figure 2.4 illustrates the computation of the shadow with a line. We can see that the edges of the line get blurred, losing detail and sharpness but gaining a much smoother look. Figure 2.5 shows percentage closer filtering in the scene from earlier with a shadow map size of 128×128 resulting in much smoother shadows.

We improve the quality even more by interpolating bilinearly (Figure 2.6). If we let the hardware calculate the shadow computation, by explicitly setting up a shadow map texture, we can activate the hardware accelerated bilinear interpolation for this texture. Doing so gives us the results seen in Figure 2.5.

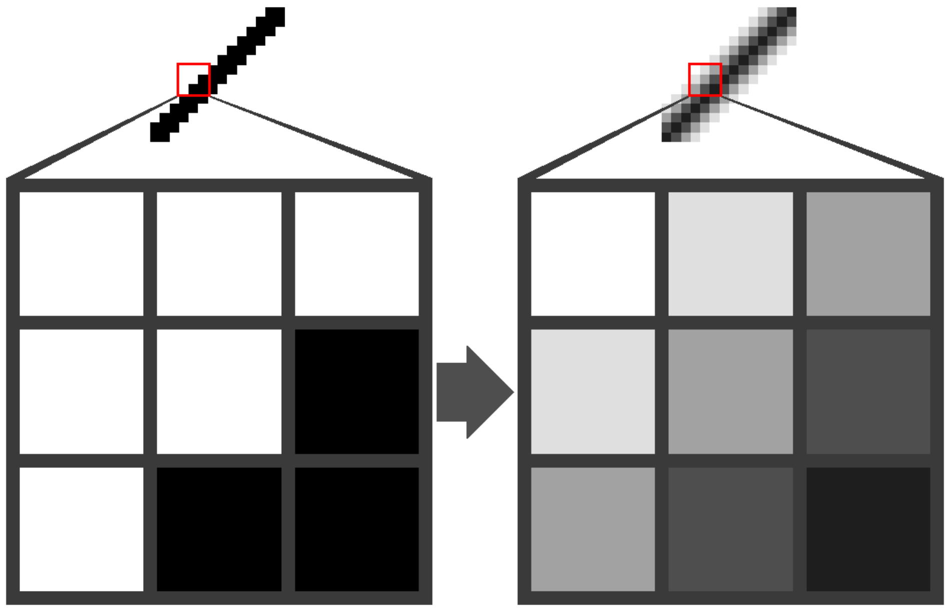


Figure 2.4: Left: shadow of a line without percentage closer filtering
Right: shadow of the same line with a 3×3 percentage closer filtering kernel.

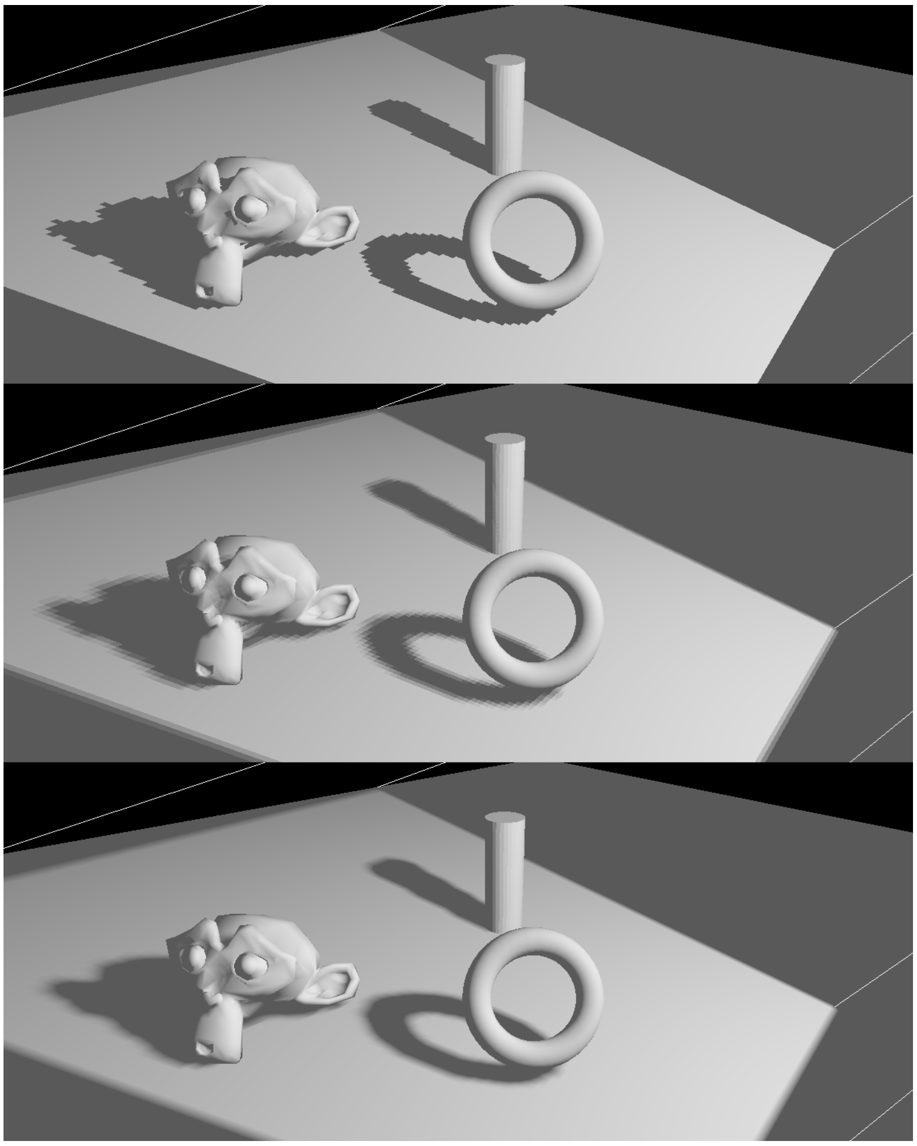


Figure 2.5: All three images use a 128×128 shadow map.
Image 1 is rendered without percentage closer filtering.
Image 2 is rendered with a 3×3 percentage closer filtering kernel but without bilinear interpolation.
Image 3 is rendered with a 3×3 percentage closer filtering kernel and hardware accelerated bilinear interpolation.
Source: Computer Graphics exercise 7 2019 KIT

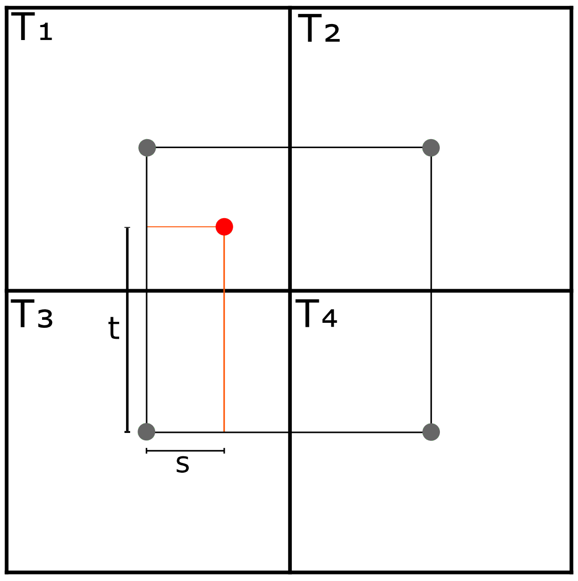


Figure 2.6: Bilinear interpolation is calculated as:
$T = (1-s)tT_1 + stT_2 + (1-s)(1-t)T_3 + s(1-t)T_4$

2.2 Cascaded Shadow Mapping

We want to compute global light sources. A good example for that is the sun. The light emitted by the sun must travel a long distance. Therefore the angle between these light rays is small. So small in fact that we approximate it with zero. That means that the light is parallel, letting us use an orthogonal projection for the shadow map.

This is advantageous because the orthogonal projection consists of a cuboid bounding box, thus making it easy to surround geometry with it. In order to surround an object, we just need its vertices. With these vertices and the light direction we then compute the enclosing projection. We do this by finding the minimum and maximum coordinates of all vertices for the given axis and use them to generate the orthogonal projection matrix. But what should be enclosed?

Of course the first idea might be the entirety of geometry in the scene. This would lead to aliasing as described in Section 2.1.1, where the frustum encloses mainly unimportant geometry, thus lowering the effective resolution of the shadow map. The question is, what is the most important geometry? The answer is simple, the geometry the camera sees. So it is a good idea to enclose the view frustum. This is done by multiplying the vertices of a unit cube with the inverse view projection matrix of the camera, resulting in the generation of the view frustum in world space. We activate depth clamping for the shadow map rendering in order to capture all geometry that is outside the view frustum that might throw a shadow.

As we can see in Figure 2.7 the resulting image is not that good looking. It still suffers from the problem that the effective resolution is low. In order to improve that further we split the view frustum into so called cascades [ZSXL06, Eng06]. We do this by splitting the view frustum in intervals of depth, creating new partial frusta that get their own shadow map.

Looking at Figure 2.8 it represents a rather perfect scenario, where no cascade overlaps into the frustum of another cascade. Normally that is not the case. The shadows of the pillars are more detailed than before. In Figure 2.9 we can see for instance the gap between the sphere and ring represented in the shadow of the front pillars.

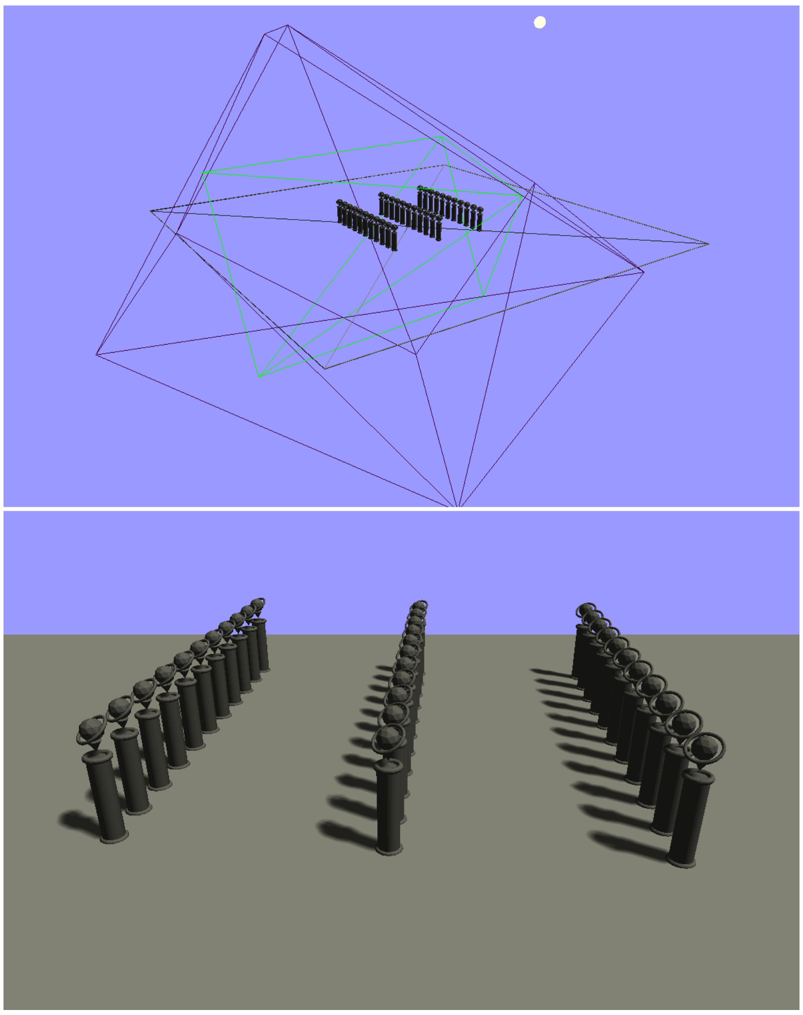


Figure 2.7: Top: frustum of the light (purple) enclosing the view frustum (green)
Bottom: resulting image using a 512×512 shadow map and 3×3 percentage closer filtering

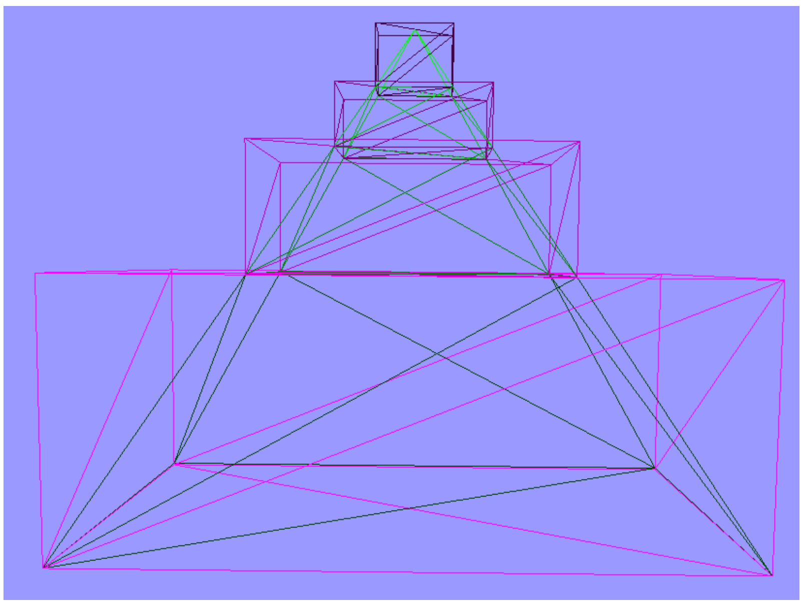


Figure 2.8: Cascades of light (shades of purple) enclosing the view frustum (shades of green). The cascade splits are fixed at 0-12.5%, 12.5-25%, 25-50%, 50-100% of the view frustum

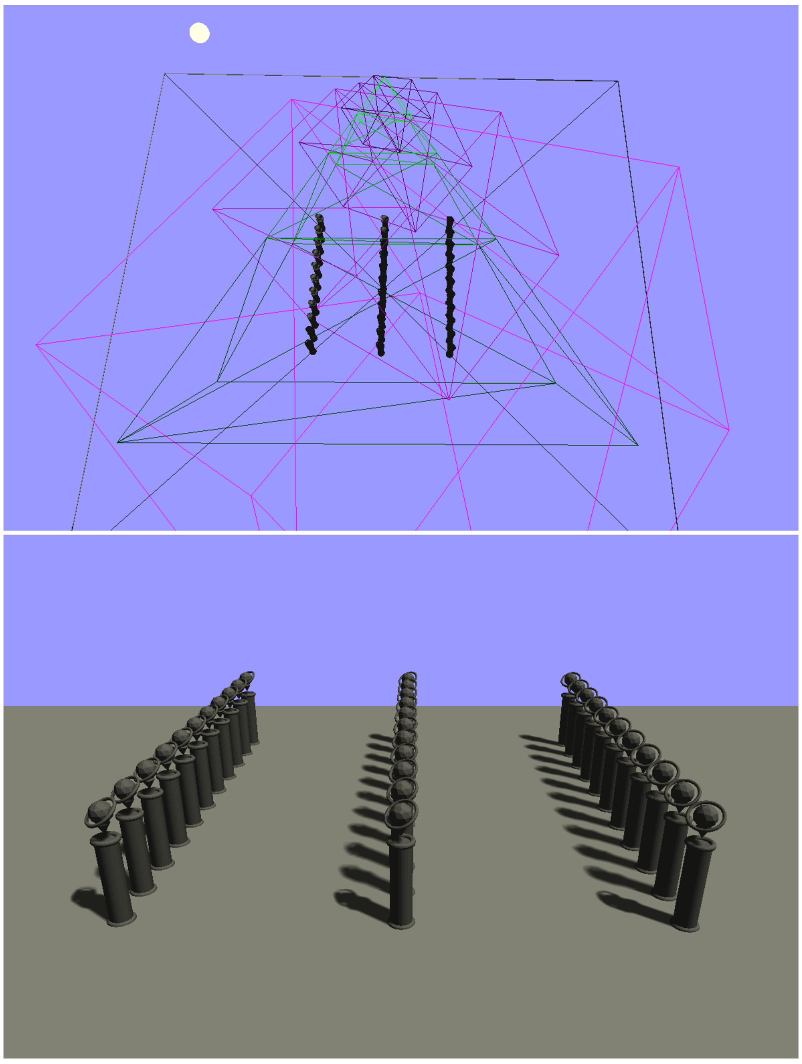


Figure 2.9: Top: cascades of the light (shades of purple) enclosing the view frustum (shades of green).
Bottom: resulting image using four (512×512) shadow maps (one for each cascade) and 3×3 percentage closer filtering.
The cascade splits are fixed at 0-12.5%, 12.5-25%, 25-50%, 50-100% of the depth range of the view frustum

2.2.1 Cascade Overlapping

A subtle but in some situations bad looking problem occurs at the border between two cascades as seen in Figure 2.10a. This cut exists because the shadow seen from the shadow map of the third split is different from the shadow seen from the fourth split.

This is made more natural looking by making the partial frusta of these splits a bit deeper, forcing them to overlap at a given depth. There we interpolate the shadow value to smoothen the border between two splits. Figure 2.10b shows that by doing so the hard cut from Figure 2.10a has been smoothened.

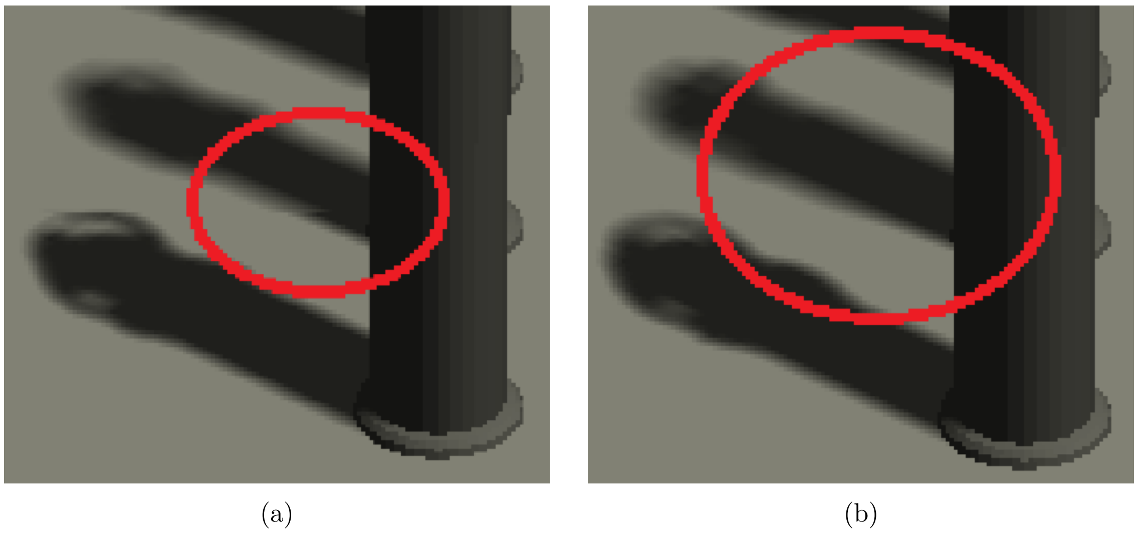


Figure 2.10: (a) Zoomed version of Figure 2.9 bottom image, highlighting the problem occurring at the border of two splits
(b) Figure 2.10a with overlapping splits

2.2.2 Bounding Box Fitting

If we have a look at Figure 2.9 again, we can see that approximately a third of the screen is filled with blue void. We also can see that a part of the partial frustum of cascade 3 looks at nothing as well.

Figure 2.11 shows us that the third cascade does not have to enclose the entirety of its frustum split. The front 50% have no geometry whatsoever, thus we raise the effective resolution by enclosing only the cropped frustum split.

We crop it by finding the minimum and maximum in every direction of every partial frustum, thus cropping it in x-, y- and z-direction, minimizing the size of the bounding box. With these values we build a scale and transformation matrix that improves our shadow maps by fitting the part of the view frustum that includes geometry [LSL11].

In order to get the minimum and maximum of every coordinate for every cascade efficiently, we use a reduce shader [Har07] to compute them for a given depth image from the perspective of the camera. With these values we then build our scale transformation matrix for every cascade and end up with a result like Figure 2.12.

As we can see the bounding boxes are much smaller resulting in more detailed shadows.

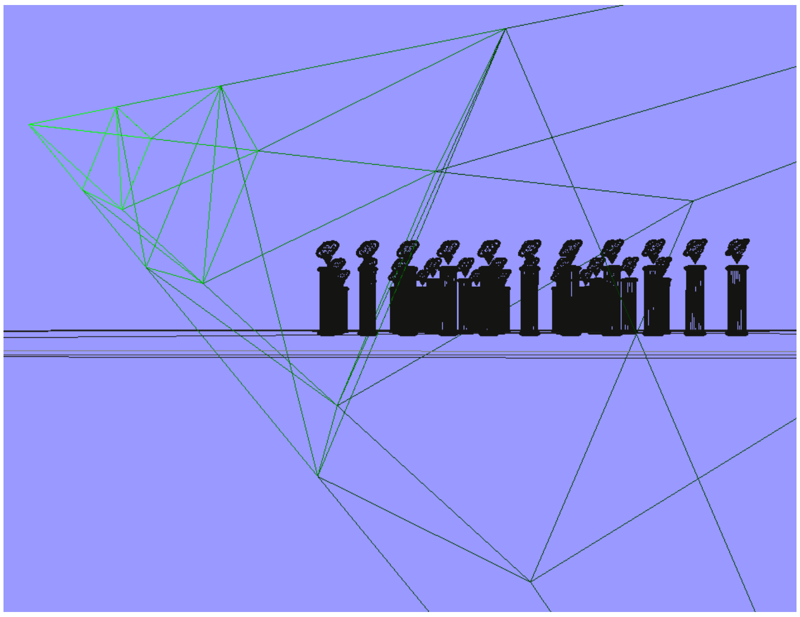


Figure 2.11: Excessive bounding box size

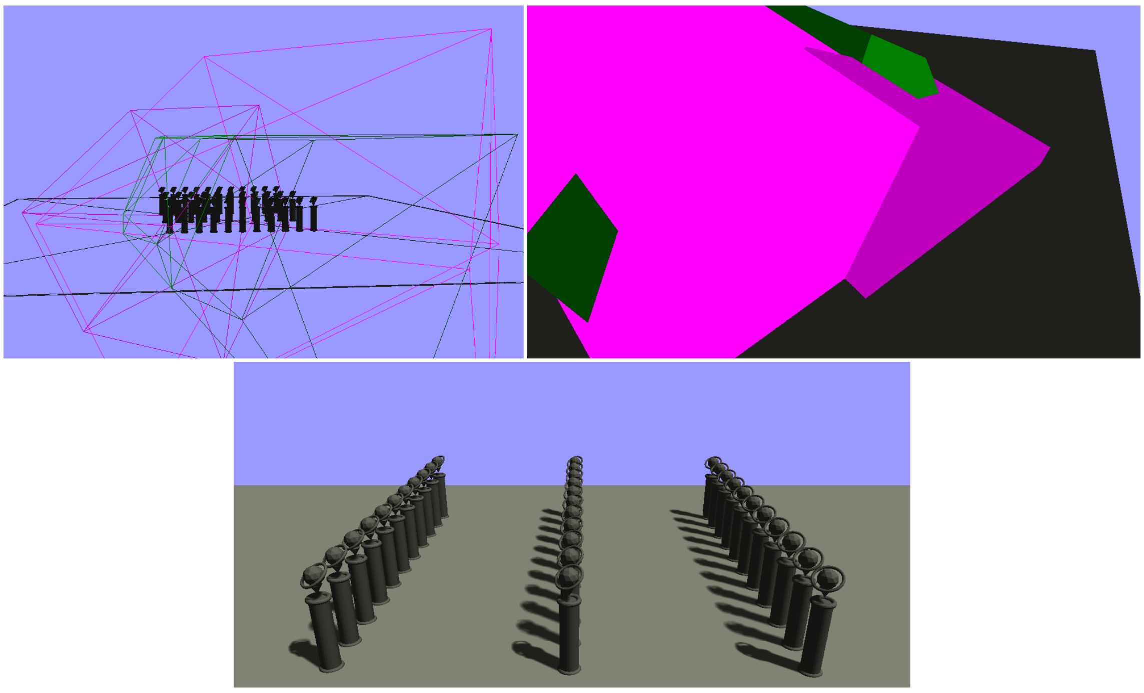


Figure 2.12: These three images show the effect of bounding box fitting for:
1. The z-coordinate of the frustum and the frusta of the light
2. The frusta of the light being much smaller
3. The shadows

2.2.3 Split Methods

As setting the splits at fixed positions is a bad idea for many scenes, especially for interactive applications, we want to look at adaptive methods. Therefore we are looking at the methods first introduced by Zhang et al. [ZSXL06], and discuss their shortcomings:

A good idea could be to minimize aliasing arising from the view frustum. The camera normally has a perspective projection. That means that objects further away are exponentially smaller. Their cascades do not need a high resolution in order to make them look properly shadowed. This means that a logarithmic split scheme should lead to a minimization of aliasing error for every cascade:

$$C_i^{log} = n(f/n)^{i/m} \tag{2.8}$$

Where n is the near plane, f is the far plane, i is the index of the current cascade and m is the number of all cascades. There is one problem with this method because, if we would implement it like that, we would have a small cascade at front being mostly wasted because it is rather rare that geometry appears this close to the near plane.

Another method arises by splitting the view frustum uniformly:

$$C_i^{uniform} = n + (f - n)i/m \tag{2.9}$$

This should lead to the worst distribution of aliasing because close geometry needs a much higher resolution in order to throw good shadows than smaller looking geometry in the back as discussed for Equation (2.8).

Combining these two methods leads to a scheme called practical split scheme [ZSXL06]. We combine these two methods by adding them and dividing by two:

$$C_i = \frac{C_i^{log} + C_i^{uniform}}{2} = \frac{n(f/n)^{i/m} + n + (f - n)i/m}{2} \tag{2.10}$$

These schemes lead to a split result like Figure 2.13. In Figure 2.12 and Figure 2.14 we see that two of the four Cascades are completely useless. This also results in a worse outcome for Figure 2.14 than the fixed method.

If we want to use our four shadow maps efficiently we do not want them to look at nothing. In order to do this we have to take a look at the distribution of depths arising from the depth buffer of the camera, giving it the name sample distribution shadow maps [LSL11].

One idea discussed by Lauritzen et al. [LSL11] is to find the minimum and maximum value of depths seen from the perspective of the camera in order to use them as the new near and far planes. Therefore we use the same technique discussed for bounding box fitting (Section 2.2.2). The reduce shader calculates our values efficiently resulting in Figure 2.15.

As we can see the results have improved, but there still are scenes where one or more cascades are wasted. For instance, a scene where we are on a balcony of a house looking down at the streets. If we see just a pixel of the balcony this min max method will clamp it between the balcony and the street. A big part of the view frustum is thus in mid air, resulting in one or maybe two cascades unutilised. Figure 2.16 clearly represents the limits of min max clamping. The view frustum was split into two usable cascades only (first cascade light and last cascade dark green). We are wasting two 512×512 shadow maps again which is an inefficient resource management.

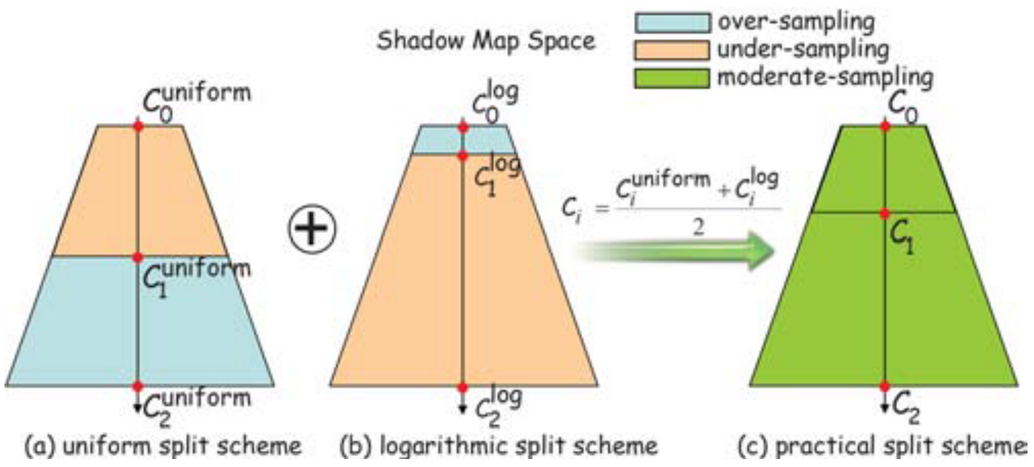


Figure 2.13: Prcatical split scheme
Source: [ZSXL06]

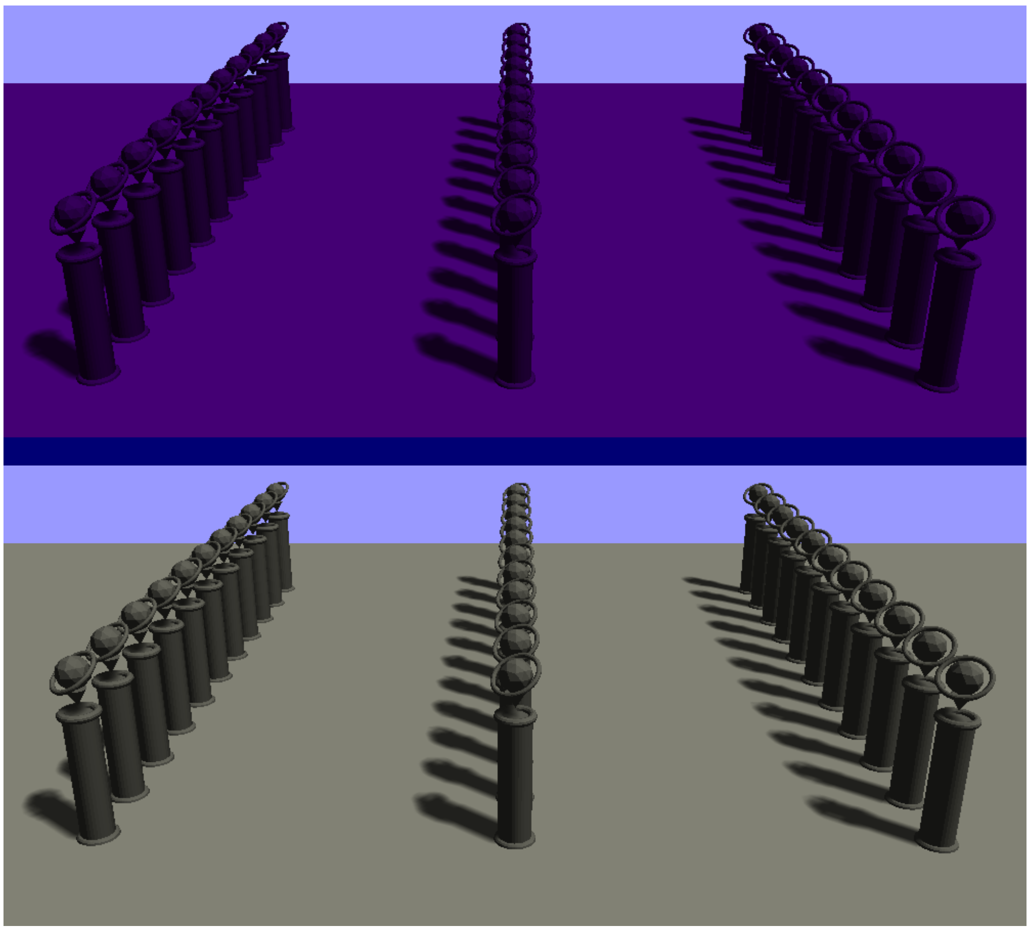


Figure 2.14: Practical split scheme result. Top: color coded cascades, Bottom: resulting shadows

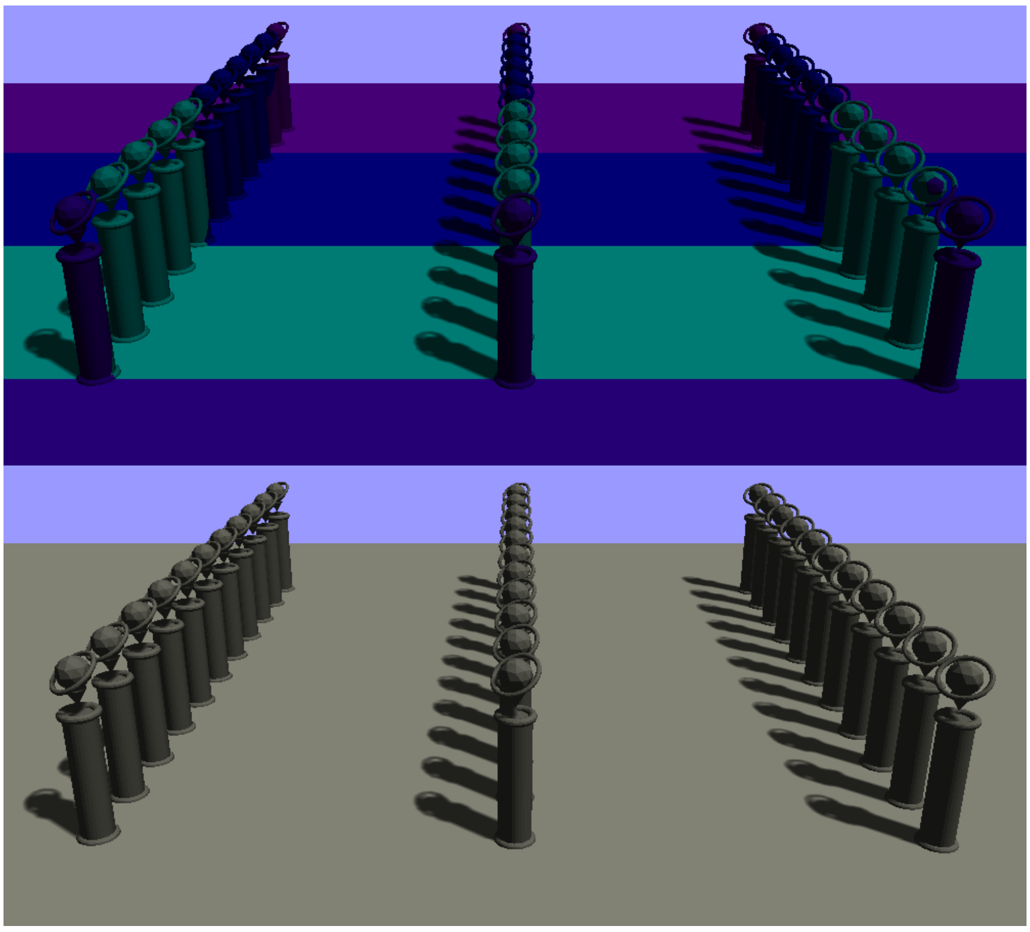


Figure 2.15: Practical split scheme with min, max clamping. Top: color coded cascades, Bottom: resulting shadows

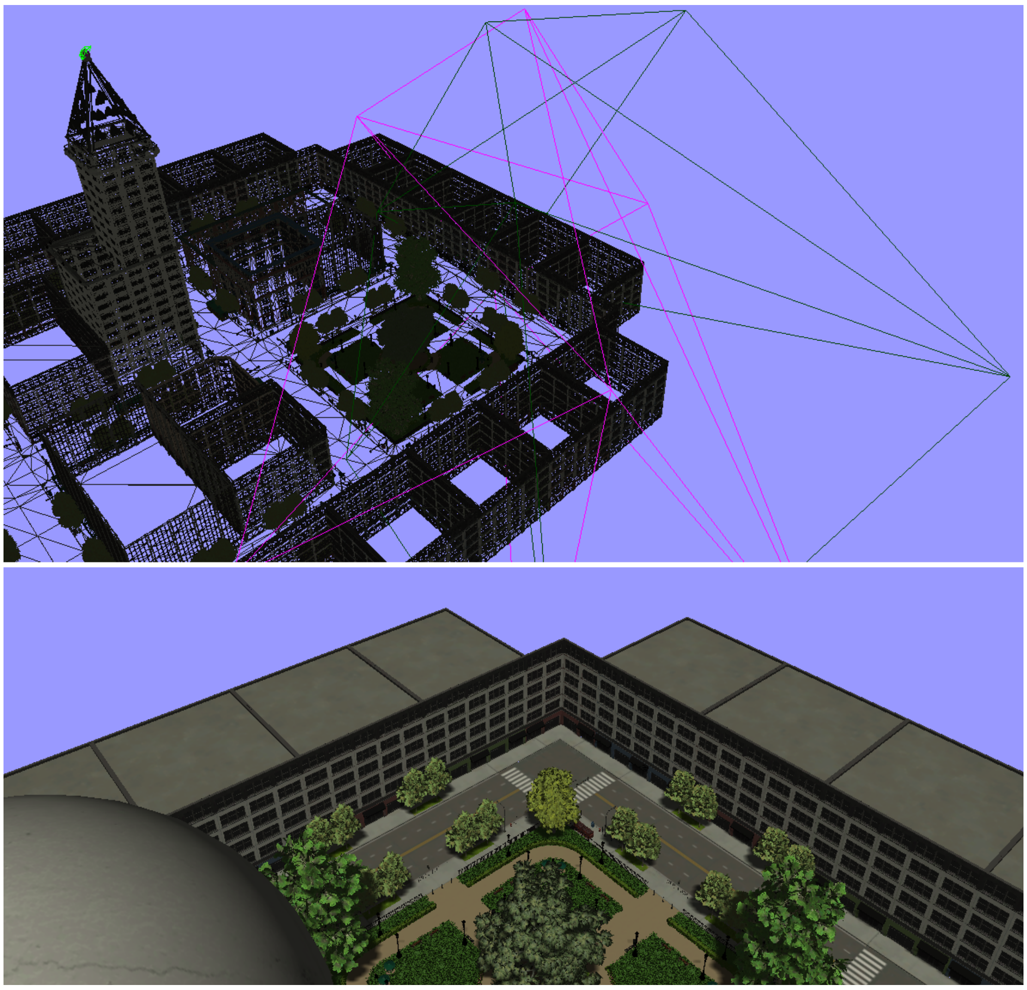


Figure 2.16: Practical split scheme with min, max clamping worst case scenario. Top: resulting frusta, Bottom: resulting shadows

2.3 Moments

As described before we want to analyze the depth distribution of the screen. This distribution consists of $N \in \mathbb{N}$ pixels with depth values $x_1, \ldots, x_N \in \mathbb{R}$. It is difficult to work with all depth values but it becomes easy if we work with a small number of moments. In order to work with moments, we define the problem as a probability space and a real random number x of the distribution. Therefore we define the cumulative depth distribution function for the depth values of our screen at depth x_f as:

$$\frac{1}{N}\sum_{i=1}^{N} \delta_{x_i}(x_f) \tag{2.11}$$

with $\delta_{x_i}(x_f)$ being defined as:

$$\delta_{x_i}(x_f) = \begin{cases} 1 & \text{if } x_i < x_f \\ 0 & \text{else} \end{cases} \tag{2.12}$$

We thus call $\mathbb{E}(x^r)$ for $r \in \mathbb{N}$ the r-th power moment of x [Geo15, p. 117]. The first raw moment is called the mean, the second central moment is called variance:

$$\begin{aligned} \mathbb{E}(x) &= \frac{1}{N}\sum_{i=1}^{N} x_i \\ \mathbb{V}(x) = \mathbb{E}([x - \mathbb{E}(x)]^2) &= \frac{1}{N}\sum_{i=1}^{N} (x_i - \mathbb{E}(x))^2 \end{aligned} \tag{2.13}$$

In order to estimate our depth distribution, we first need to compute the moment vector b containing m moments:

$$\begin{aligned} b &= (\mathbb{E}(x), \mathbb{E}(x^2), \ldots, \mathbb{E}(x^m))^T \\ &= \left(\frac{1}{N}\sum_{i=1}^{N} x_i, \frac{1}{N}\sum_{i=1}^{N} x_i^2, \ldots, \frac{1}{N}\sum_{i=1}^{N} x_i^m\right)^T \end{aligned} \tag{2.14}$$

If we want to estimate our cumulative distribution using our moment vector only, we want to calculate the lower bound of all distributions with the same moments. Therefore we need to generalize our depth distribution function at position $x_f = z_f$:

$$\sum_{l=0}^{K} w_l \delta_{z_l}(z_f) \tag{2.15}$$

Where z_l are the support points of the distribution, w_l the associated weights and $K \in \mathbb{N}$ the number of support points. We now have to solve the following minimization problem in order to find a suitable distribution with our moments:

$$\inf\{S(z < z_f) | S \text{ distribution on } \mathbb{R} \text{ with moment vector } b\} \tag{2.16}$$

This non-trivial problem has been solved by the Chebyshev-Markov inequality [KNL77, p. 125]. This inequality provides us with the information that we only need $\frac{m}{2}+1$ support points [PK15]. This leads to the new cumulative distribution:

$$\sum_{l=0}^{\frac{m}{2}} w_l \delta_{z_l}(z_f) \tag{2.17}$$

As the moments for these distributions remain equal, we have a new equation for our moments too:

$$b_j = \sum_{l=0}^{\frac{m}{2}} w_l z_l^j \tag{2.18}$$

This equation defines a system of linear equations with m equations and $m+2$ variables. If we take into consideration that we in fact have the zeroth moment and can set $z_0 = z_f$ [Pet17, p. 60], we end up with $m+2$ equations and $m+2$ variables. This system leads to solutions, as demonstrated by Peters et al. [PK15].

The lower bound of all distributions containing the same moment vector b is thus equal to:

$$S(z < z_f) = \sum_{l=0}^{\frac{m}{2}} w_l \cdot \delta_{z_l}(z_f) \tag{2.19}$$

The upper bound is therefore approximated by adding w_0, as we modify Equation (2.12) by changing $<$ to $\leq$:

$$S(z \leq z_f) = w_0 + \sum_{l=0}^{\frac{m}{2}} w_l \cdot \delta_{z_l}(z_f) \tag{2.20}$$

Figure 2.17 shows this process in the field. It illustrates w_l and z_l for a given distribution and z_f. We can calculate the upper bound by adding w_0, because w_0 happens to be the so called maximal point mass $w(z_f)$. The maximal point mass is defined by the following equation:

$$w(z_f) = \sup\{S(z = z_f) | S \text{ distribution on } \mathbb{R} \text{ with moment vector } b\} \tag{2.21}$$

This equation also leads to the same system of linear equations as finding the lower bound [1]. We therefore end up with the following equation:

$$w(z_f) = S(z = z_f) = \sum_{l=0}^{\frac{m}{2}} w_l \begin{cases} 1 & \text{if } z_l = z_f \\ 0 & \text{else} \end{cases} \tag{2.22}$$

We set $z_0 = z_f$ and can thus say that $w(z_f) = w_0$.

[1]From a personal conversation with Christoph Peters

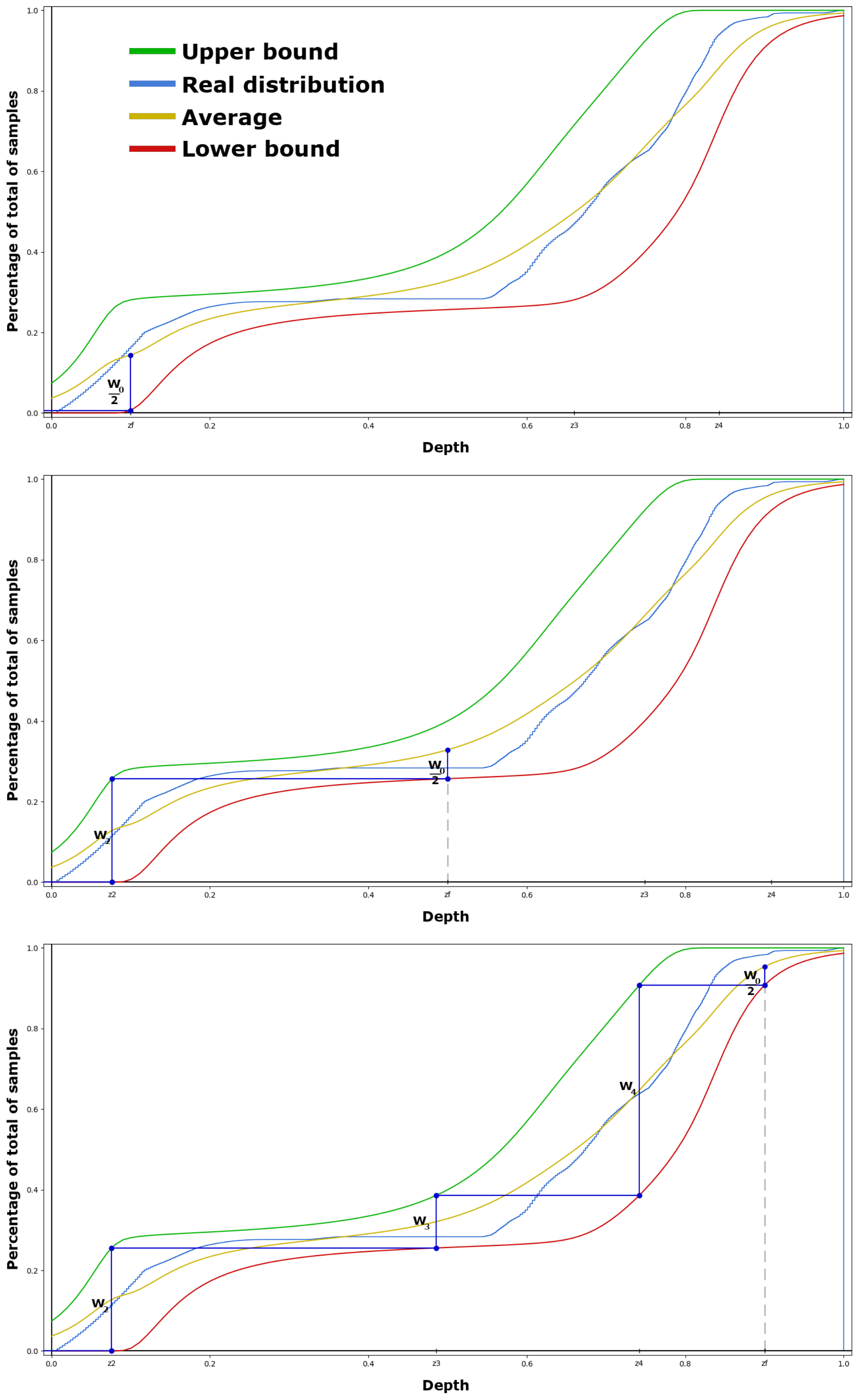


Figure 2.17: Example for the reconstruction of a distribution showing us the maximum point mass w_l and the support points z_l for a given depth z_f.

3. Related Work

There are many techniques to improve shadow quality for real-time applications. We can also find many techniques using moments to achieve their goals. We describe some of these techniques here to get a better understanding of shadow calculation and the usage of moments in general.

3.1 Variance Shadow Mapping

Variance shadow mapping [DL06] is introduced as an alternative to percentage closer filtering. It enables the possibility for the shadow texture to be filtered directly. This is beneficial, as there are many hardware accelerated possibilities to filter a texture directly, thus allowing for many fast methods of antialiasing.

The main idea behind variance shadow maps is to store the first two moments of a distribution of depth in the shadow map. Filtering this texture results in a texture containing the first two moments of the distribution.

From these two moments the mean and variance arise as:

$$\begin{aligned} \mu &= \mathbb{E}(x) = b_1 \\ \sigma^2 &= \mathbb{E}(x^2) - \mathbb{E}(x)^2 = b_2 - b_1^2 \end{aligned} \tag{3.1}$$

Where b_1 and b_2 represent the first and second moment, μ represents the mean and σ^2 the variance. Calculating the shadow intensity now consists of using the Chebyshev-Markov inequality [KNL77, p. 125] in order to get an upper bound for the depth z_f as:

$$P(x \geq z_f) \leq p_{max}(z_f) = \begin{cases} \dfrac{\sigma^2}{\sigma^2 + (z_f - \mu)^2} & \text{if } z_f > \mu \\ 1 & \text{else} \end{cases} \tag{3.2}$$

In order to utilize what we learned in Section 2.3, we modify this equation a bit:

$$\begin{aligned} 1 - P(x < z_f) &\leq p_{max}(z_f) \\ P(x < z_f) \geq 1 - p_{max}(z_f) &= \begin{cases} 1 - \dfrac{\sigma^2}{\sigma^2 + (z_f - \mu)^2} & \text{if } z_f > \mu \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3.3}$$

Therefore we compute the lower bound of the moment-based reconstruction. Peters et al. [PMWK17] proves that this equation, also known as Cantelli's inequality is equivalent to the moment-based reconstruction provided by their paper.

A problem arising from this technique is light leaking or light bleeding. This happens because p_{max} represents a lower bound of the brightness. Thus there are situations where p_{max} does not represent the ground truth correctly. As it is a lower bound, the problem manifests itself in shadows as fragments of light that should not be there.

3.2 Moment Shadow Mapping

As the name suggests, moment shadow mapping [PK15] improves the visual quality of shadows by using moments. It does so by providing a so called moment shadow map. This moment shadow map allows for efficient texture filtering and antialiasing, as a variance shadow map does because it is an extension of a variance shadow map.

It consists of four moments being stored in the four color channels of the texture. If we want to filter this shadow map linearly, we do so now because we will not lose too much information. Examples for these filter operations would be a two-pass Gaussian blur or the generation of mipmaps.

In order to calculate the shadow intensity, we compute the lower bound of the reconstruction generated by the four moments, as described in Section 2.3. We achieve this by utilizing the Chebyshev-Markov inequality [KNL77, p. 125]. If we do so, we reduce the occurrence of shadow acne, as we underestimate the ground truth as sharply as possible. As we use four moments, we also minimize the problem of light leaking because we approximate the ground truth closer for any moment we add (see Figure 3.1).

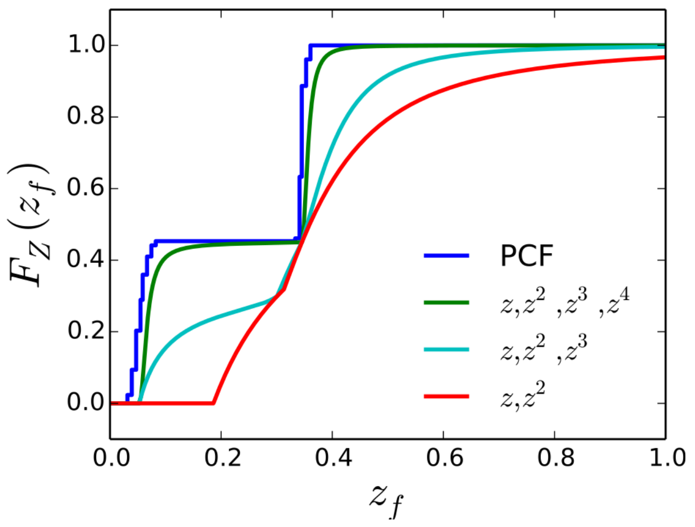


Figure 3.1: Lower bounds for a cumulative distribution (blue) obtained by two, three and four moments

Source: [PK15]

3.3 Moment-Based Order-Independent Transparency

Computing transparent geometry correctly would normally consist of sorting before blending it. This is not efficient especially for dynamic scenes. Therefore it is desirable to find a technique that does not need to sort the geometry. Moment-based order-independent transparency [MKKP18a] does just that.

First, we need to establish that transparent geometry is rendered twice. The first pass is used to determine the transmittance function per pixel, the second one to composite transparent surfaces properly [MKKP18a, p. 4].

The transmittance at a depth z_f is defined as:

$$T(z_f) = \prod_{l=0, z_l < z_f}^{n-1} (1 - \alpha_l) \tag{3.4}$$

where α_l is the alpha value of the frame at the depth of the support point z_l (see Section 2.3). Münstermann et al. [MKKP18a] explain that rather than working with the transmittance we should work in logarithmic domain. Therefore we define the absorbance such that:

$$A(z_f) = -\ln T(z_f) = \sum_{l=0, z_l < z_f}^{n-1} -\ln(1 - \alpha_l) \tag{3.5}$$

We thus get a distribution function like the one described in Section 2.3:

$$\sum_{l=0}^{n-1} -\ln(1 - \alpha_l)\delta_{z_l}(z_f) \tag{3.6}$$

From here, we know that we can use our moments to approximate the absorbance per pixel and thus get the transmittance.

As transparency can occur in complicated constellations, we need a better approximation of the distribution in order to capture it more precisely. Moment-based order-independent transparency, thus uses either eight power moments or four trigonometric moments.

The first render pass stores the transmittance as a vector of moments:

$$b = \sum_{l=0}^{n-1} - \ln(1 - \alpha_l)\mathbf{b}(z_l) \tag{3.7}$$

Where $\mathbf{b}(z) = (1, z, z^2, \dots, z^m)^T$.

The second render pass is used to render and shade all transparent surfaces and compute the absorbance and with it the transmittance. The result is thus stored in order to blend the surfaces correctly by using the transmittance.

This technique is interesting for this thesis, as we also intend to use eight power moments.

3.4 Shadow Map Parametrization

3.4.1 Perspective Aliasing Analysis

Stamminger et al. [SD02] provide a good analysis for the aliasing error arising from the perspective projection (Figure 2.3c). The error consists of the following equation:

$$d = d_s \frac{r_s}{r_i} \frac{\cos\beta}{\cos\alpha} \tag{3.8}$$

The variables of this equation are described by Figure 3.2.

We build upon this analysis in more detail in Section 4.3

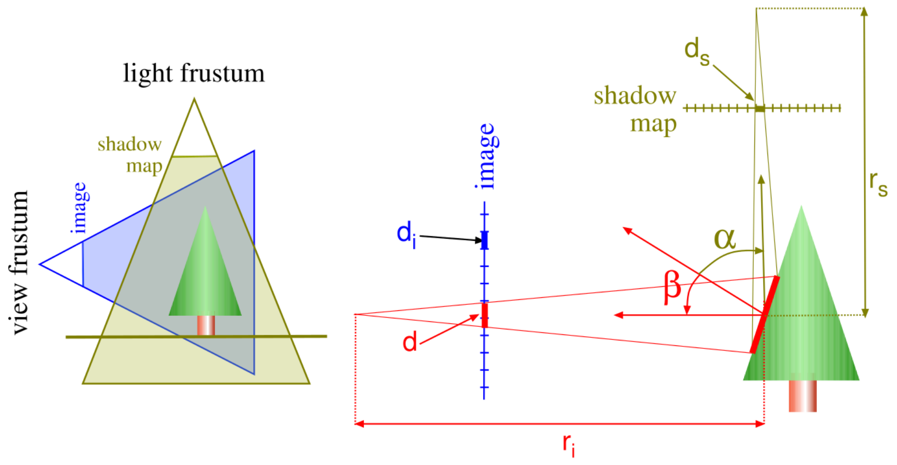


Figure 3.2: Shadow map projection quantities
Source: [SD02, p. 2]

3.4.2 Trapezoidal Shadow Maps

One method that is built upon the analysis provided by perspective shadow maps is the method of trapezoidal shadow maps [MT04]. It solves the problem of resolution being bad for lights that are far away. Another problem it solves is the one that plagues perspective shadow maps the most, the polygon offset problem.

The resolution problem for light sources that are far away is solved by maximizing the utilization of the shadow map, by enclosing as much of the view frustum as possible. We therefore have to define a so called normalization matrix N. This matrix transforms points from post-perspective space to the so called N-space. An example for this matrix would be the orthogonal projection matrix that achieves the enclosing of the view frustum with a bounding box in Figure 2.7. The N-space for this example is thus called bounding box space.

Bounding boxes do not enclose a view frustum with perspective projection ideally, as you can see in Figure 3.3. Martin et al. [MT04] suggest the use of a trapezoidal approximation for this enclosure. This trapezoidal enclosure also solves the problem of polygon offset, if implemented correctly.

In order to explain the calculation of that trapezoidal approximation we follow the steps provided by Martin et al. [MT04, p. 5]:

Therefore we first gain the vertices of the view frustum. We do this by multiplying the vertices of a unit cube with the inverse view projection matrix of the camera, as we did in Section 2.2. We then transform these vertices into light space (post-perspective space) by multiplying them with the view projection matrix of our light source. We compute the center line l that passes through the center of the near and far plane of the frustum. After that, we compute the two dimensional convex hull of these vertices. We continue by calculating the top line l_t that is orthogonal to l and touches the convex hull. At last, we obtain the base line l_b that is parallel to l_t and also touches the convex hull.

In order to compute the matrix N we only have to get the sides of the trapezoidal. Therefore we call λ the distance between l_t and l_b. We map l_t to $y = +1$ and l_b to $y = -1$ along the center line l. We indicate the focus region [MT04] as a distance p_L on l. This distance is thus mapped by $y = \xi$ and $\delta' = 1 - \xi$. In order to compute the intersection

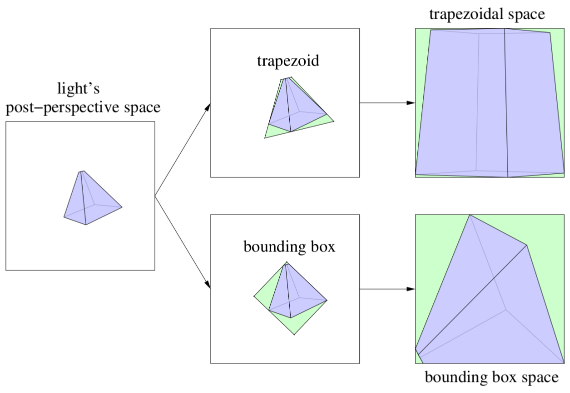


Figure 3.3: An example for trapezoidal approximation
Source: [MT04, p. 3]

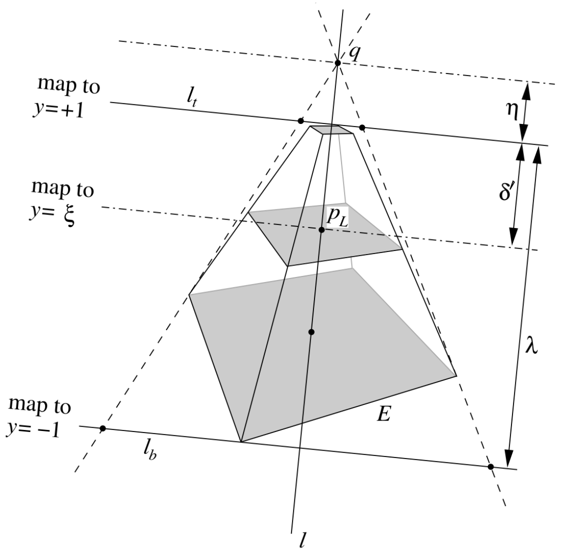


Figure 3.4: One dimensional projection problem to compute q.
Source: [MT04, p. 6]

point q between both sides and the distance η from the top line, we only need to solve the following equation further explained by Figure 3.4:

$$\begin{pmatrix} \frac{-(\lambda+2\eta)}{\lambda} & \frac{2(\lambda+\eta)\eta}{\lambda} \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \delta' + \eta \\ 1 \end{pmatrix} = \begin{pmatrix} \tilde{\xi} \\ \omega \end{pmatrix} \tag{3.9}$$

With $\xi = \dfrac{\tilde{\xi}}{\omega}$, we get the solution $\eta = \dfrac{\lambda\delta' + \lambda\delta'\xi}{\lambda - 2\delta' - \lambda\xi}$. Therefore we can compute q and with q we get our two sides and can thus calculate our matrix N.

3.4.3 Irregular Z-Buffer Shadows

Another method to minimize perspective aliasing is to use irregular z-buffer shadows [JLBM05]. It does so by storing surface points seen from the camera in a two dimensional data structure. We therefore store many points in some texels and less to no points in others.

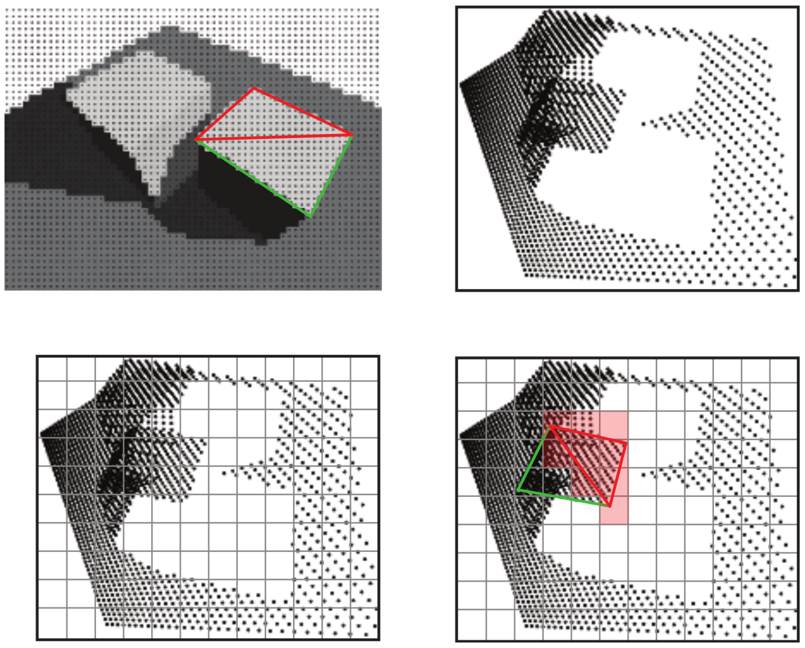


Figure 3.5: Generation of an irregular z-buffer
Source: [MHH18, c. 7 p. 260]

In order to detect shadows, we just compare the depth of a rasterized triangle with all points inside the texels it touches (see Figure 3.5).

By doing so, we achieve the benefit of ray tracing, consisting of handling arbitrary directions dynamically. We are also able to utilize the rasterization pipeline efficiently. In order to reduce the number of points for texels near the camera, cascaded shadow mapping can be used.

3.4.4 Sample Distribution Shadow Maps

Sample distribution shadow maps [LSL11] describe techniques of shadow map computation that take the depth distribution of the camera into consideration. These techniques compute depth partitions that are used for cascaded shadow mapping.

The easiest implementation is to find the global minimum and maximum of the depth distribution by using a reduce shader.

This results in a smaller view frustum that can now be partitioned by using former techniques like for example the logarithmic scheme (Equation (2.8)). We can see the results in Figure 3.6.

The adaptive logarithmic scheme is described as being similar to the logarithmic scheme except it avoids the gaps in the distribution. Therefore it is required to compute a histogram of the distribution. Computing that histogram is expensive.

Another scheme is described as the K-means scheme. It consists of finding clusters in the depth histogram and placing splits around them. It finds these clusters by utilizing the K-means algorithm, also known as Lloyd's algorithm [Llo82]. This algorithm consists of minimizing the sum over all variances of the $K \in \mathbb{N}$ found clusters Q_k with mean $\mathbb{E}(x_k)$. As we know from Section 2.3, we can calculate the variance for finite real sample sets as

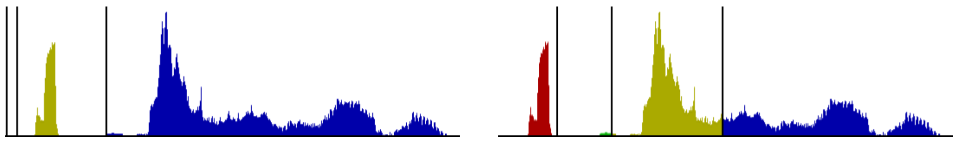

(a) Basic logarithmic (b) SDSM logarithmic

Figure 3.6: Source: [LSL11]

a sum and thus need to minimize the following equation:

$$\sum_{k=1}^{K} \sum_{i=1}^{|Q_k|} (x_{k_i} - \mathbb{E}(x_k))^2 \tag{3.10}$$

The algorithm consists of three steps [Mac02, p. 286 Algorithm 20.2]:

Initialization

Set K means $\mathbf{m_k}$ to random values.

Assignment step

Each data point x_i is assigned to the cluster Q_l with the nearest mean. This is achieved by comparing the Euclidean distance between the data points and the means $\mathbf{m_k}$.

Update step

We need to update the means $\mathbf{m_k}$ such that they represent the mean of the current cluster constellation:

$$m_k = \frac{1}{|Q_k|} \sum_{i=1}^{|Q_k|} x_{k_i} \tag{3.11}$$

Repeat assignment and update

We repeat these steps until the assignments do not change anymore.

Unfortunately this method is rather slow.

Bounding box fitting (Section 2.2.2) as discussed before is an important step to implement sample distribution shadow maps, as it minimizes the frusta sizes notably.

Sample distribution shadow maps are the predecessors for our technique, as we try to improve on the approach using the histogram. As discussed before we want to accelerate and improve the results by using moments.

4. Moment-Based Cascade Splits

This chapter explains the idea of moment-based cascade splits and how to achieve an efficient computation of such. It explains that by first introducing the ideas of using local minima in the moment-based reconstruction (section 4.1) and equiarea splitting (section 4.2).

After that, we discuss how we can improve our results by warping our values beforehand (section 4.3) and last we explain how to implement these techniques efficiently (section .4.4).

As discussed before, in order to gain good split positions, we want to have a look at the depth distributions arising from the frame buffer of the camera. If we plot these values as a cumulative histogram, we might get something like Figure 4.1.

The histogram in Figure 4.1 shows us the distribution of the depth. We can see gaps between spikes of depth. We may not want a split at every place where the histogram is zero but we want splits where it is for a long long range of depth. A split at these positions is good in most cases because then the cascades include only one depth interval. An example would be the rooftop in Figure 4.1 represented by the first spike in in the histogram.

It is not a problem if many cascades would land in the same depth interval. A problem would be if one cascade would be assigned to many intervals, thus inflating the bounding box, resulting in a bad effective resolution.

These gaps manifest themselves in the cumulative histogram Figure 4.1 as points where the derivative of said histogram is minimal. As we only want big gaps we have a benefit of using the smooth reconstruction provided by our moments.

Therefore we approximate these histograms in Figure 4.1 as described in Section 3.2.

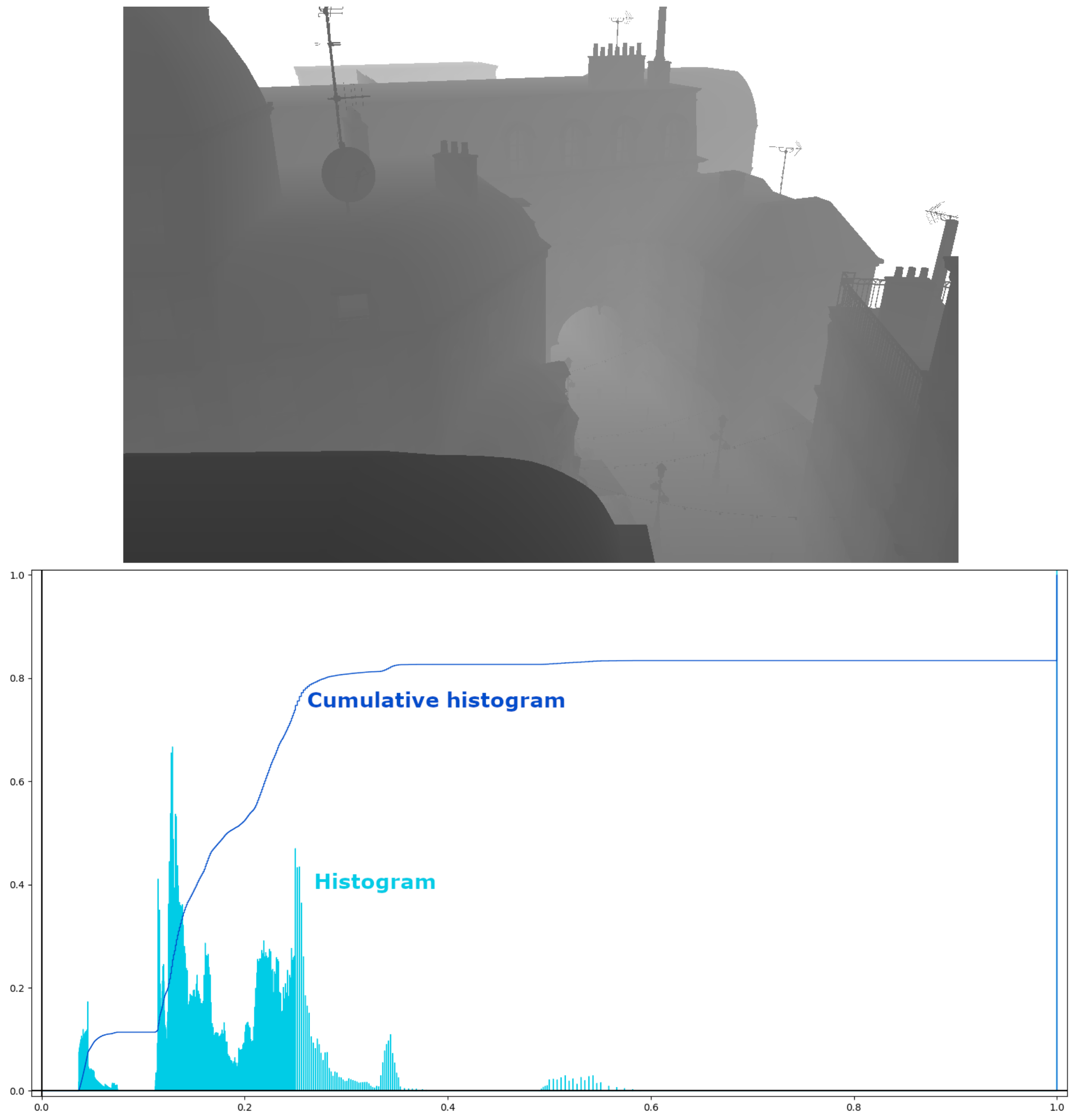


Figure 4.1: Exemplary depth buffer image with histograms

4.1 Local Minima Splitting

From Peters et al. [PMWK17] we learned that the approximation of the cumulative histogram is achieved by solving two systems of linear equations and finding the roots of a polynomial (algorithm 1). We thus get weights $w(z)$ representing the height between the lower and upper bound at a given depth z:

$$\begin{aligned} b(z) &= (1, z^1, \dots, z^{\frac{m}{2}})^T \in \mathbb{R}^{\frac{m}{2}+1} \\ w(z) &= \frac{1}{b^T(z)B^{-1}b(z)} \end{aligned} \tag{4.1}$$

Where B is the Hankel matrix generated from m moments b_i like:

$$B = \begin{pmatrix} b_0 & b_1 & \cdots & b_{\frac{m}{2}} \\ b_1 & b_2 & \cdots & b_{\frac{m}{2}+1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\frac{m}{2}} & b_{\frac{m}{2}+1} & \cdots & b_m \end{pmatrix} \in \mathbb{R}^{(\frac{m}{2}+1)\times(\frac{m}{2}+1)} \tag{4.2}$$

The minimal derivatives of the histogram are located at positions where the lower and upper bounds are as close together as possible. This has to do with the fact that this distance is the maximal point mass (described in section 2.3). That means we just have to find the minima of $w(z)$. Therefore we need the derivative of $w(z)$:

$$\begin{aligned} w^{'}(z) &= \frac{\partial}{\partial z}\frac{1}{b^T(z)B^{-1}b(z)} \\ &= -\frac{2b^{'T}(z)B^{-1}b(z)}{(b^T(z)B^{-1}b(z))^2} \end{aligned} \tag{4.3}$$

We do not get a division by zero here because we know that the maximum point mass is a value between zero and one (see Section 2.3). Thus the reciprocal can not be zero. This is also the case for the squared reciprocal.

We now just have to find the roots of the derivative:

$$\begin{aligned} 0 &= w^{'}(z) \\ \Leftrightarrow 0 &= -\frac{2b^{'T}(z)B^{-1}b(z)}{(b^T(z)B^{-1}b(z))^2} \\ \Leftrightarrow 0 &= b^{'T}(z)B^{-1}b(z) \end{aligned} \tag{4.4}$$

This equation represents nothing more than finding the roots of a polynomial. Note that by multiplying both sides with a negative value we have to test this polynomial for maxima, not minima.

In order to show that $b'^T(z)B^{-1}b(z)$ is a polynomial, let $C := (c_{j,k})_{j,k=0}^{\frac{m}{2}} := B^{-1}$ for:

$$
\begin{aligned}
0 &= b'^T(z)B^{-1}b(z) = b'^T(z)Cb(z) \\
&= \left(\sum_{j=0}^{\frac{m}{2}} j z^{j-1} c_{j,k} \right)_{k=0}^{\frac{m}{2}} \left(z^k \right)_{k=0}^{\frac{m}{2}} \\
&= \sum_{k=0}^{\frac{m}{2}} \sum_{j=0}^{\frac{m}{2}} j z^{j-1} c_{j,k} z^k = \sum_{k=0}^{\frac{m}{2}} \sum_{j=0}^{\frac{m}{2}} j c_{j,k} z^{k+j-1} = \sum_{j=1}^{\frac{m}{2}} \sum_{k=0}^{\frac{m}{2}} j c_{j,k} z^{k+j-1} \\
&= \sum_{i=0}^{m-1} \left(\sum_{j=\max\left(1,i+1-\frac{m}{2}\right)}^{\min\left(i+1,\frac{m}{2}\right)} j c_{j,(i+1-j)} \right) z^i \\
&= \sum_{i=0}^{m-1} a_i z^i = 0
\end{aligned}
\tag{4.5}
$$

As we can see we get a polynomial with a degree of $m-1$.

Finding the root of a polynomial is a well discussed problem. For efficiency reasons we use Laguerre's method [Pre07, p. 468]. Laguerre's method is beneficial as it has a cubic convergence for most cases. That is the case because it consumes two derivatives of the polynomial for each step in order to compute the roots. Cases where it can not find a root are rather rare [Pre07, p. 467] and can be fixed by breaking the cycle. In our test cases, the number of iterations never exceeded five iterations and the average number of iterations was approximately between two and three.

If we find a root, we thus eliminate it from the polynomial by using polynomial division [Pre07, p. 204]. We divide the polynomial by a polynomial of degree one containing the same found root, like $(x - x_{root})$. After iterating this method $m-1$ times, we end up with a vector containing our roots.

We now just have to set the positions of the splits to the positions of the maxima by looking at the derivative of the polynomial at these roots.

This results in the roots found in Figure 4.2.

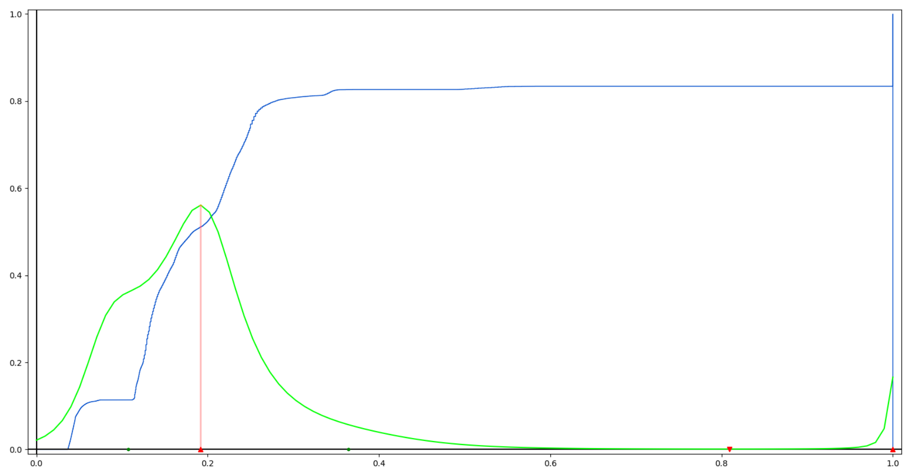


Figure 4.2: The roots found for Figure 4.1: Red triangles indicate minima and maxima for the green curve. The triangles point up for a maximum and down for a minimum. The green dots represent imaginary roots. The green curve represents $w(z)$.

4.2 Equiarea Splitting

If we want to have four cascades (three splits), that is a problem too, because as of now we only set splits in places where minima of the maximal point mass are. The polynomial we described before has a degree of $m - 1$. If we seek maxima as described before, the degree of the polynomial alone limits us to exactly $m - 1$ extrema. If we get imaginary extrema, they always come in pairs as the complex number and the conjugated of said number. Thus we can get one extremum at worst. This one extremum can only be a minimum, as the maximal point mass diverges to negative infinity in both directions. We end up with zero maxima at worst.

At best we can get $\lfloor \frac{m-1}{2} \rfloor$ maxima, also because the maximal point mass diverges to negative infinity. Therefore we get between zero and three maxima for eight moments, thus zero to three minima of the maximal point mass, leading to splits.

That means we need to have an exit strategy for the cases that do not offer enough minima. In these cases we use the moments to calculate the approximation of the given areas, thus trying to set splits such that the number of pixels per cascade is distributed evenly.

Doing that consists of following Peters et al. [PMWK17, algorithm 1] in order to get a function that estimates the cumulative distribution for a given depth (like Figure 2.17). If we do so and always take the average between lower and upper bound we get a picture like Figure 4.3.

In order to invert this function, we need to find roots again but this time not from polynomials. Therefore we use a technique called bisection [Pre07, p. 449]. By doing that we estimate positions where the area has a given value. For instance if we want to partition the entire screen so that each of the four cascades has 25% of the screens pixels, we could do so now (as seen in Figure 4.4).

The main idea now is to take as many minima as we can and set a split there. If there are splits left that need to be set, we try to partition these cascades such that all splits are set. We try to end up minimizing the maximum size of all resulting cascades by setting these splits.

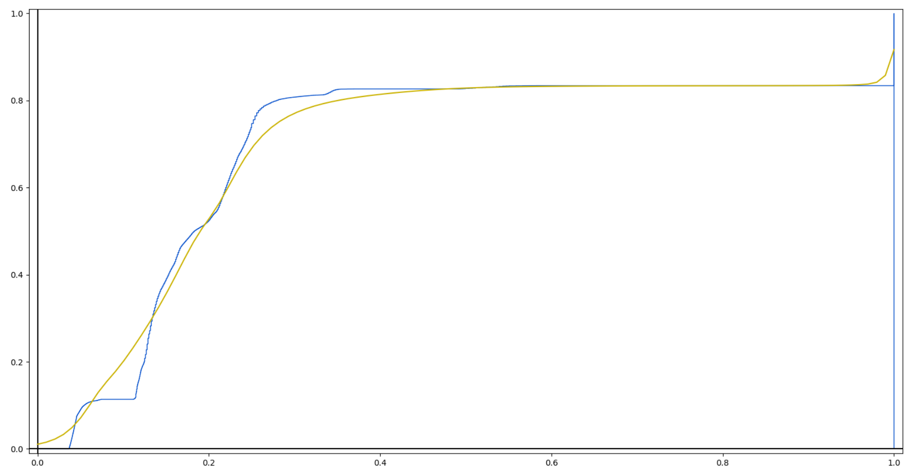


Figure 4.3: Estimated cumulative distribution (yellow)

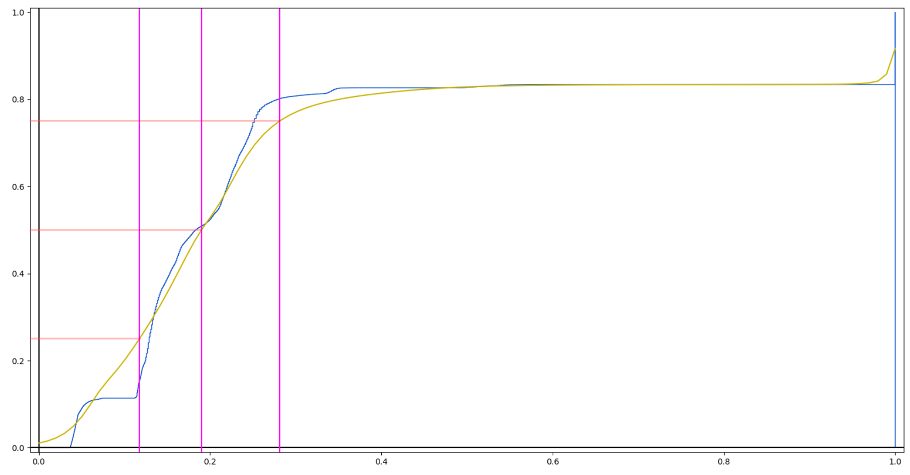


Figure 4.4: Splits (magenta), if every cascade would include 25% of the screens pixel

If we also include that cascades have to include a predefined percentage of pixels, we end up with a result for Figure 4.2 seen in Figure 4.5. We do this by merging cascades defined by a minima that do not include a given percentage of the pixel of the screen. We do this before calculating the remaining splits in order to find new splits with these newly merged cascades. We test this with our reconstruction.

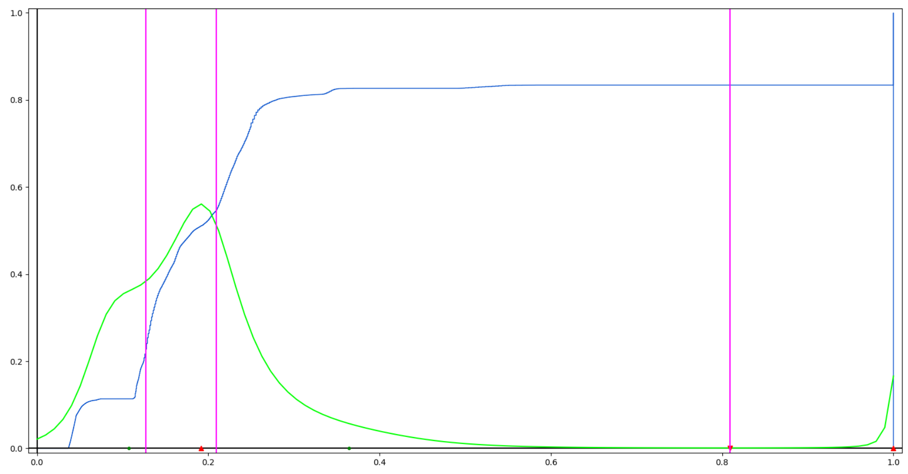


Figure 4.5: Splits (magenta) resulting from our technique (without warp)

4.3 Depth Warp

As we can see in Figure 4.5, we only get one minimum and that minimum is not even good. The pixels separated by that split are pixels looking at geometry and pixels that do not look at geometry. We do not need cascades where no geometry lies. Another problem is, that even by using eight moments, there are many scenes where the roots of the polynomial are imaginary even though the possibility of a minimum in that position exists. These roots are useless for our computation.

Our depth samples are plagued by the perspective transformation, as it applies a function on them, resulting in a non-linear distribution (see Figure 4.7). This also impacts the maximal point mass, as it gets even smoother than we want to.

We are thus able to achieve better results by warping the depth values before calculating the moments with them. An idea would be to minimize the impact of the perspective projection by looking at a flat plane. The function generated by perspective projection is equal to the following equation [SD02]:

$$d = d_s \frac{r_s}{r_i} \frac{\cos\beta}{\cos\alpha} \tag{4.6}$$

Where β is the angle between the camera direction and the surface normal, r_i is the distance from the camera to the intersection, r_s is the distance from the intersection to the light source, α is the angle between the surface normal and the direction of the light (as seen in Figure 3.2).

Stamminger et al. [SD02] states that ”for directional light sources, d is independent of r_s”. We ignore $\cos\alpha$ too because it would be rather difficulty to keep track of it for the inverse warp. If we have a camera at height h and a vector $(r_i, 0, 0)^T$ as seen in Figure 4.6, we get the following equation by setting all constant (and ignored) values to one:

$$\begin{aligned} d &\sim \frac{1}{r_i}\cos\beta \\ &= \frac{1}{r_i}\frac{h}{\|(r_i,0,0)^T - (0,0,h)^T\|} \\ &= \frac{1}{r_i}\frac{h}{\|(r_i,0,-h)^T\|} \\ &= \frac{1}{r_i}\frac{h}{\sqrt{r_i^2+h^2}} \end{aligned} \tag{4.7}$$

We also ignore h in the numerator, as it only applies a linear impact, resulting in:

$$d \sim \frac{1}{r_i}\frac{1}{\sqrt{r_i^2+h^2}} \tag{4.8}$$

If we now want to warp these values, we have to solve the following problem. We have the function $D(z)$ we want to compensate for. This function is embedded in the given distribution of depths. We want an equal distribution for a flat floor, thus we want $D(z)$ to become one for any depth.

First we define $D(z)$ as:

$$D(z) = \frac{1}{z}\frac{1}{\sqrt{z^2+h^2}} = \frac{1}{z\sqrt{z^2+h^2}} \tag{4.9}$$

After that we need to model a function $z(y)$ for:

$$\int_{y_0}^{y_1} D(z(y))z'(y)dy = \int_{z_0}^{z_1} D(z)dz \tag{4.10}$$

With $z_0 = z(y_0)$ and $z_1 = z(y_1)$. Our function $z(y)$ also needs to solve the following equation:

$$\begin{aligned} & D(z(y))z'(y) = 1 \\ \Leftrightarrow\quad & z' = \frac{1}{D(z(y))} \\ \text{in our case}\quad \Leftrightarrow\quad & z' = \frac{1}{\frac{1}{z(y)\sqrt{z(y)^2+h^2}}} = z(y)\sqrt{z(y)^2+h^2} \end{aligned} \tag{4.11}$$

The solution of this problem is not trivial:

$$\begin{aligned} z(y) &= -\frac{2h^2e^{hy}}{h^2e^{2hy}-1} \\ y(z) = z^{-1}(z) &= \frac{\log\left(\frac{z}{h\left(\sqrt{h^2+z^2}+h\right)}\right)}{h} \end{aligned} \tag{4.12}$$

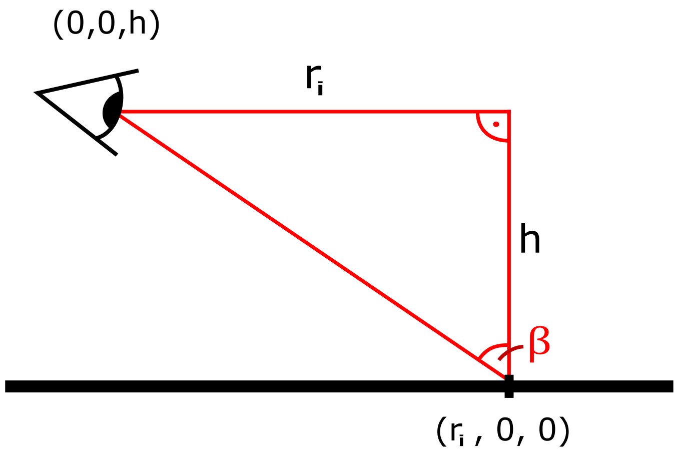


Figure 4.6: Notation for Equation (4.7)

Where h is the ratio between camera height and view frustum depth. If chosen poorly, this value can lead to wrong warping, thus deteriorating our results.

If we look at a flat ground plane like Figure 4.7, we can see that the warped distribution is almost linear, what should be expected for the given scene. Applying $y(z)$ on the depth values before generating the moments and clamping the results between the minimum and maximum of all warped depths, leads to better results than not doing it. If we want the splits generated with these values, we have to apply $z(y)$ in order to warp them back.

If we have a look at Figure 4.1 again and calculate our technique using the warp, we get the result seen in Figure 4.8. As we can see the minimum is now exactly between two big intervals of depth, where no geometry lies. The other splits are good as well, as they partition the scene in the back. This is the result we want.

An example where we get more minima by using this warp is described in Figure 4.9.

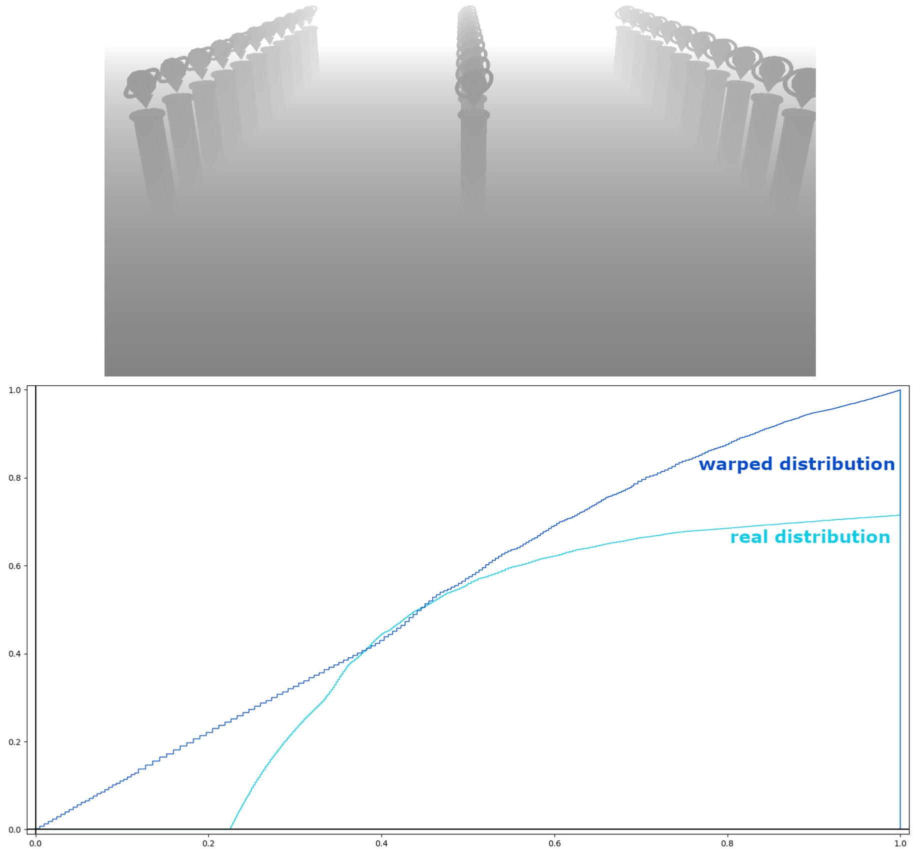


Figure 4.7: Top: Depth buffer, Bottom: real and warped depth distributions

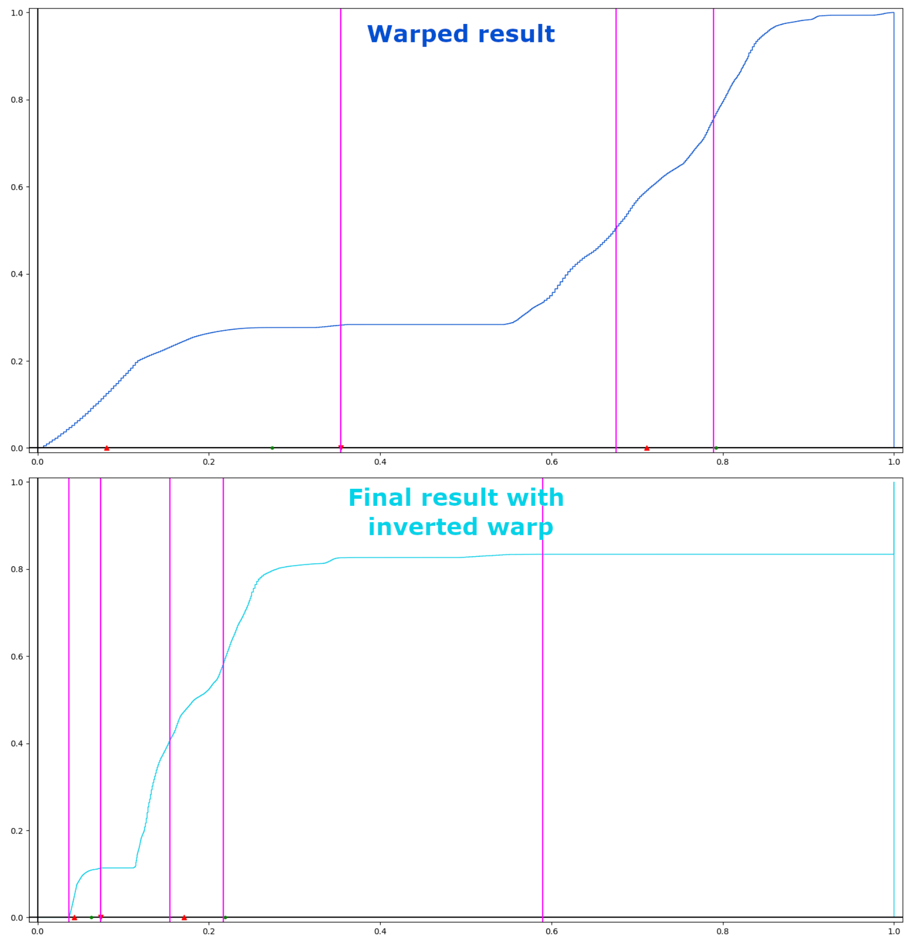


Figure 4.8: Both images are generated for Figure 4.1. Top: resulting warped splits and warped distribution, Bottom: the final result.

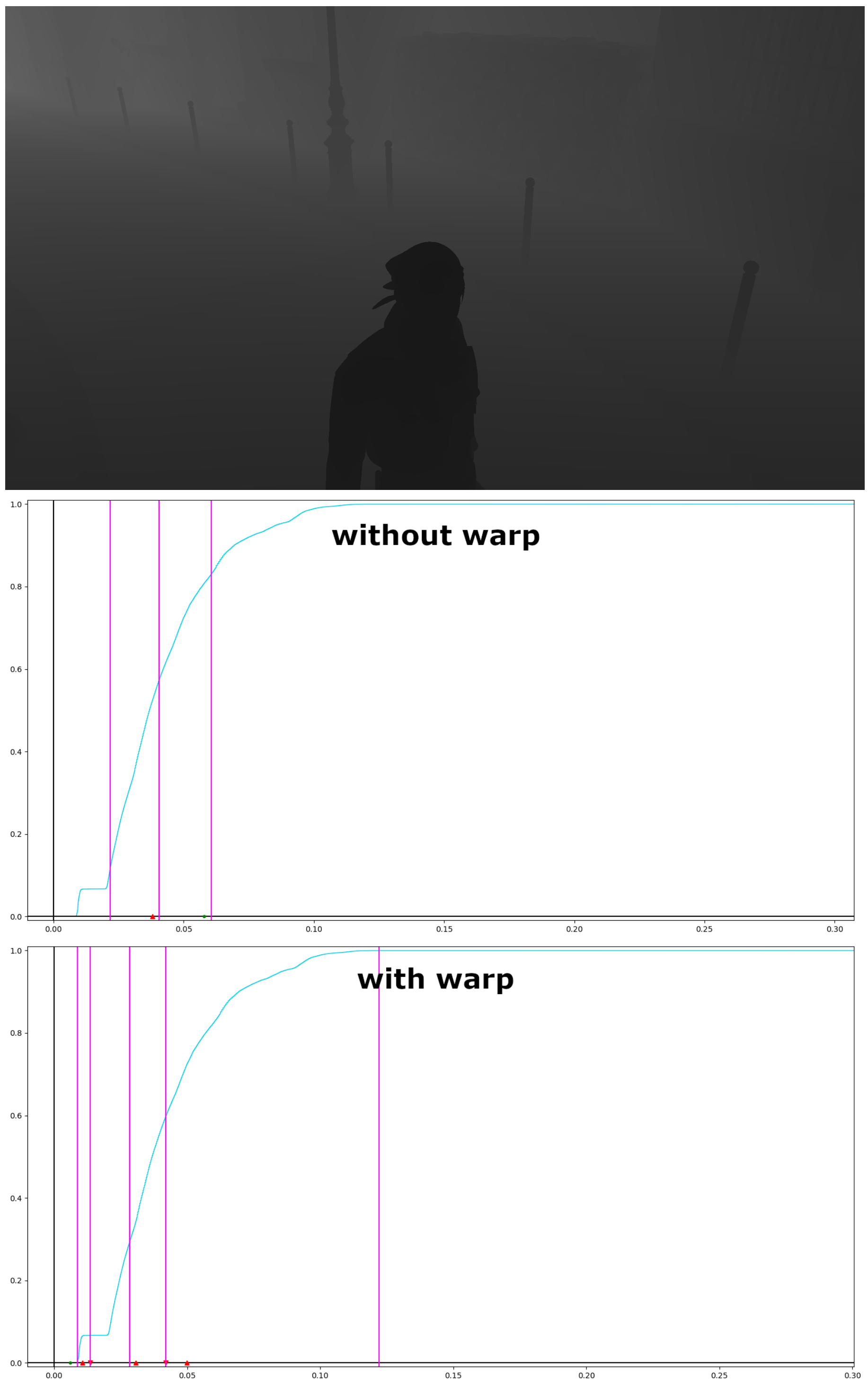


Figure 4.9: Results (with and without warping) for a scene described by the depth buffer

4.4 Computation of Warped Moments

We want to increase the chances that a real root is found, by invalidating pixels that look at nothing. We do this by using a color attachment texture for the frame buffer object of the depth texture. We then tell the fragment shader to assign the validity in form of color (one being valid, zero being invalid). By doing so we automatically get rid of all pixels that show no geometry. We exclude all pixels where the cosine of the angle between the surface normal and the light direction (the diffuse value) is negative as well because they are in shade anyway.

In order to compute the moments efficiently, we use a compute shader [Har07]. As we already established, we use four passes in order to reduce a 4096×4096 image to two 1×1 textures, storing the moments inside their color channels. Therefore we use two result textures for each pass. These textures have four channels containing one 32 bit float each. For the first pass the resolution is 512×512, the second pass 64×64, the third pass 8×8 and the fourth pass ends up with a resolution of 1×1.

This could be optimized further by using only 4 result textures in total (two 512×512 and two 64×64) and restricting the writing and reading on the correct pixels per pass, but this is not really necessary as we would only save 65 pixels. It is a good idea if we had images bigger than 4096×4096.

In order to calculate the moments, we only need to look at the first pass. The job of the other passes is to sum everything up we give them. Prior to the moment calculation, the first pass needs to check whether we are inside the picture or not. If we are not, it writes the value zero and the thread group calculates the 8×8 field normally. We also do this, if any of the pixels inside of the image is invalid (tested by the valid texture).

If we are inside the picture and have a valid pixel, we get the depth value of said pixel. First, we have to linearize this depth. Therefore we have to take a look at what happens to the z component of a position p if we multiply it with our projection matrix:

$$P_{cam}p = \begin{pmatrix} \frac{S}{a} & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & -\dfrac{f+n}{(f-n)} & -\dfrac{2fn}{(f-n)} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ -d_{lin} \\ 1 \end{pmatrix} = \begin{pmatrix} * \\ * \\ \dfrac{f+n}{(f-n)}d_{lin} - \dfrac{2fn}{(f-n)} \\ d_{lin} \end{pmatrix} \tag{4.13}$$

Where a indicates the aspect ratio, $S = \dfrac{1}{\tan\left(\frac{v_{fov}}{2}\right)}$ and v_{fov} is the field of view in radians. We get the definition of the matrix P_{cam} from the documentation of the GLM-Library. If we now divide our z-coordinate with the w-component we get:

$$2d - 1 = \frac{\frac{f+n}{(f-n)}d_{lin} - \frac{2fn}{(f-n)}}{d_{lin}} = \frac{f+n}{(f-n)} - \frac{2fn}{d_{lin}(f-n)} = \frac{f+n-\frac{2fn}{d_{lin}}}{(f-n)} \tag{4.14}$$

We therefore get the following linearization:

$$d_{lin} = \frac{2nf}{f+n-(2d-1)(f-n)} \tag{4.15}$$

After that we transform the value that is now between $[n, f]$, so it will be between zero and one:

$$z = \frac{d_{lin} - n}{f - n} \tag{4.16}$$

We then warp the linear value as described in Section 4.3. If we want to minimize the numerical error of the moment calculation, we want the depth values to be between minus one and one. In order to do this, we either need to know the minimum and maximum of the warped values before hand or compute them in this reduce shader. If we compute them in this reduce shader, we have to apply them afterwards on the computed moments with binomial theorem. We also need the number of valid pixels in order to get the zeroth moment.

Then we take it to the power of the needed moments. We thus get $z, z^2, \ldots, z^m$ with $m = 8$ because our implementation uses eight moments. After that, we let the threads sum our moments up in a shared variable. By doing so, we get the sums of the 8×8 area [Har07, s. 15]. We could improve this even more [Har07], so there is even more potential to this technique in regards to efficiency.

These values are then stored across the four channels of both textures ($T_1(z, z^2, z^3, z^4)$ and $T_2(z^5, z^6, z^7, z^8)$). This pass is thus finished.

At the end of the computation, we have two textures storing our moment vector b (b' if only one reduce is used), containing eight warped moments.

If we ought to use one reduce shader only, we need a third texture per pass containing the minimum, maximum of the warped values and the number of valid pixels. We then have to apply the following equation upon the calculated moments:

$$\begin{aligned}
b_i &= \frac{1}{N}\sum_{k=1}^{N}\left(\frac{d_k - d_{min}}{d_{max} - d_{min}}\right)^i \\
&= \frac{1}{N(d_{\max} - d_{\min})^i}\sum_{k=1}^{N}(d_k - d_{\min})^i \\
&= \frac{1}{N(d_{\max} - d_{\min})^i}\sum_{k=1}^{N}\sum_{l=0}^{i}\binom{i}{l} d_k^{i-l}(-d_{\min})^l \\
&= \frac{1}{N(d_{\max} - d_{\min})^i}\sum_{l=0}^{i}\sum_{k=1}^{N}\binom{i}{l} d_k^{i-l}(-d_{\min})^l \\
&= \frac{1}{(d_{\max} - d_{\min})^i}\sum_{l=0}^{i}\binom{i}{l}\left(\frac{1}{N}\sum_{k=1}^{N} d_k^{i-l}\right)(-d_{\min})^l \\
&= \frac{1}{(d_{\max} - d_{\min})^i}\sum_{l=0}^{i}\binom{i}{l} b'_{i-l}(-d_{\min})^l
\end{aligned} \tag{4.17}$$

Where b_i are the moments we want, N is the number of valid pixels, $d_{\min}$ is the minimum, $d_{\max}$ the maximum value arising from the warp and b'_i are the resulting moments without applying the minimum and maximum values.

Using binomial theorem like that should theoretically lead to the same result as calculating it in two passes. In practical terms, the numerical error is bigger with this method.

Therefore this error in the moments can cause the Hankel matrix described in Section 4.1 to gain negative eigenvalues, thus losing the property of being positive definite. If that happens, the Cholesky decomposition does not work anymore.

In order to improve that behavior, Münstermann et al. [MKKP18b] suggests to use a bias α for the moments. Therefore the new moments are calculated as seen in the following equation:

$$\begin{aligned} b &= (1-\alpha)b + \alpha b^* \\ \text{with} \quad b^* &= (0, 0.75, 0, 0.676666667, 0, 0.63, 0, 0.60030303)^T \end{aligned} \tag{4.18}$$

For eight power moments, using 256 bits, Münstermann et al. [MKKP18b] proposes a value of $5 \cdot 10^{-5}$ for α. Depending on the situation this number can vary but should always be set as low as possible in order to gain a good result. We estimated after some tests that $5 \cdot 10^{-5}$ leads to good results for our computations. While lower numbers introduce more temporal incoherence, higher numbers distort the computation of minima too much.

5. Results

In order to evaluate our technique, we compare resulting images for various scenes under consideration of visual quality and run-time.

All images and results were rendered with a machine of following specification: NVIDIA GeForce GTX 1080, Intel core i7-9700K, 16GB RAM. The test framework uses OpenGL version 4.5. The bistro [Lum17] consists of 2894153 vertices. The city with the clock tower [NHB17] consists of 12088390 vertices. The character we see consists of 872560 vertices.

Figures 5.1 to 5.7 illustrate how our method performs in comparison to a fixed set of splits and the min max method from sample distribution shadow mapping [LSL11]. All techniques use bounding box fitting. The fixed cascades are set to 0 - 12.5%, 12.5 - 25%, 25 - 50%, 50 - 100% of the size of the view frustum. For the min max method, we set the cascades by using the practical scheme from Section 2.2.3.

If we have a look at Figure 5.1 and Figure 5.2, there is a noticeable improvement of the shadow cast by an antenna at the top of the building towards the character's face. We can see that the images produced by fixed and min max are not able to cast that shadow properly. The antenna in the shadow map of these methods only consists of a few pixels. Our technique dedicates the first cascade to the character model alone, thus having a much higher resolution for the antenna, producing a much better image. We can not even tell whether there is anything casting a shadow in (a) for fixed and min max. In (b) we can barely see an implication of the antenna for min max. If we look very closely, we can detect that the shadow quality for distant geometry, like the leaves on the tree, is lower in our technique than in fixed and min max. This barely impacts the visual quality of the image because we do not need that much detail for geometry that is that far away and fills less than seven percent of the visible pixels anyway.

Looking at Figure 5.3, we notice that the shadows, cast by the railing, remain as detailed for our moment-based approach as in the min max approach, but the shadows at the streets are much more detailed. This happens because in this scene we lose a cascade for min max and two cascades for fixed splits. We lose these cascades for fixed splits because the view frustum is so big that only two cascades are visible. We lose one for min max because it is exactly between the balcony and the street. We do not lose any by using our moment-based approach because it finds a minimum between balcony and street. Therefore we have the advantage of having detailed near shadows from the railing and detailed far shadows from, for instance, the street poles or lanterns. If we have a look at the balcony across the street, we can also detect an improvement of the shadows cast by the railing of that balcony.

Figure 5.4 illustrates that our moment-based approach is the only approach that is able to show thin shadows cast by the bumper of a scooter or even by cables in the scene. As we can see fixed splits can not work with this scene whatsoever. They can neither show any shadows on the scooter, nor can they cast proper shadows at the facade. The min max method can shade the facades properly but fails at the scooter. We can see that it casts some shadows like the tip of the hand break, the headlamp and the rear of the scooter. These shadows still are jagged around the edges and lack detail. The best improvement can be seen at the bumper and in the middle of the scooter. The only method that casts the shadow of a thin cable and the bumper was our method. The quality of the facade does not suffer much either. It is a bit worse than in min max but that is barely noticeable. The focus for this scene lies on the scooter anyways.

If we look at Figure 5.5, we can see that our approach is the only approach making good and efficient use of all shadow maps for the shown scene. While the fixed splits cause the second cascade to be empty, we end up with two empty cascades for min max (the second and third). Our techniques guarantees that every cascade consists of geometry. We can also define the minimal amount of geometry we want to include per cascade. Therefore this image shows one of the biggest benefits of using our technique.

The run-times are measured in milliseconds as full frame-times and are listed in Table 5.1. As we can see, our method produces better shadows at reasonable overhead. An overhead of approximately 0.4 ms for full HD and 1.05 ms for 4K is pretty reasonable, as the results are much better in comparison.

In order to estimate how much of this overhead is due to the reduce, we calculate the reduce but use the fixed split scheme, as it has no noticeable overhead. We end up with the results from Table 5.2. If we use these numbers, we can estimate that the reduce has an overhead of approximately 0.16 ms for full HD and 0.81 ms for 4K. As the optimization of the reduce is a well discussed problem and we do not use the most efficient implementation, we can accelerate this technique even more.

Our method still has some problems as we can see in Figure 5.6. The temporal coherence is not too great. As we can see a little camera movement can cause our method to change fast. This happens when a minimum, we first found and used for our split computation, is not found at the same place or at all for a slightly modified scene. As we can see, we lose the detail of the antenna in the character's face but gain detail at the street light in the back. That means that the overall detail of the scene is not gone but rather shifted along the new found cascades. The moment bias helps, but there still are situations in which we can clearly see a jump in the splits. This temporal incoherence is a problem that still needs to be improved upon.

If the problem is too harsh for the given scene, we suggest to only use the equiarea split method without seeking minima, as it is more robust.

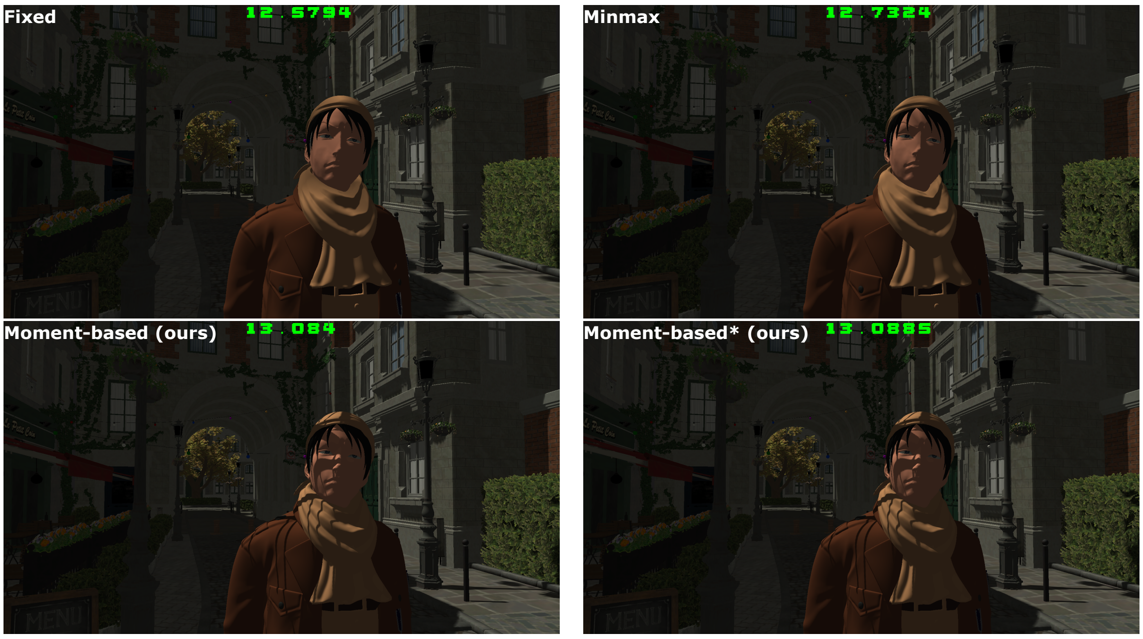
Looking at the shadows cast by the red canopy towards the windows in Figure 5.7, we can see a problem regarding the exclusion of pixels with a negative diffuse value. Other than that, our shadows are still richer in detail, especially if we look at the method without the pixel invalidation based on diffuse value. Taking the shadow cast by the railing of the building as an example, we can see that the edges are way sharper in our moment-based methods than in the fixed or min max methods. We can also see a big improvement if we look at shadows cast by the plants of the street lights. In our techniques we can distinguish the small tendrils of the plants. In fixed and min max we can not see them as detailed. On the other hand we lose some detail regarding the tables at the bistro, especially for

Shadow map resolution	Fixed	Minmax	Moment-based (one reduce)	Moment-based* (one reduce)	Moment-based (two reduces)
Figure 5.1					
512×512	12.5794	12.7324	13.084	13.0885	13.1125
1024×1024	12.6069	12.8369	13.2284	13.2259	13.2621
Figure 5.2					
512×512	21.2609	21.7354	22.7739	22.6954	22.914
1024×1024	21.4249	21.9224	22.987	22.9147	23.1368
Figure 5.3					
512×512	10.8819	11.0106	11.4376	11.4142	11.4585
Figure 5.4					
512×512	10.1756	10.3181	10.6408	10.6575	10.6739
Figure 5.5					
512×512	18.0523	18.0313	18.8124	18.8057	18.8433
Figure 5.7					
512×512	7.3694	7.5307	7.8011	7.8714	7.8362

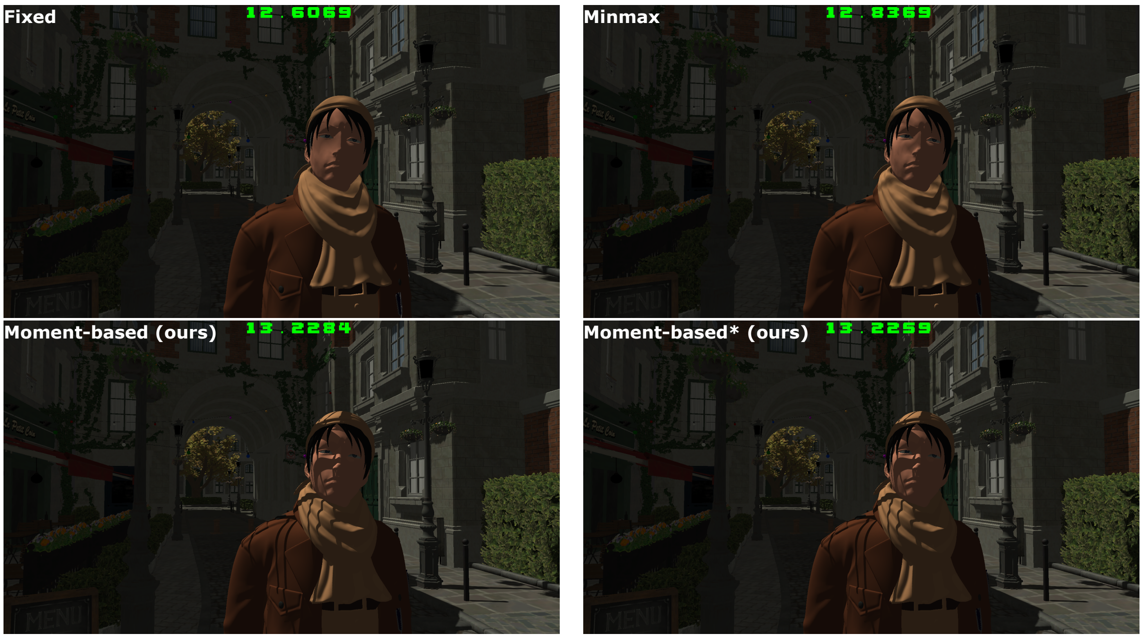
Table 5.1: Full frame times for given scenes and shadow map resolutions, measured in milliseconds. The * indicates that no pixel is invalidated based on a negative diffuse value.

our method with invalidation based on diffuse value. We also lose details in the tree at the back.

Getting rid of the invalidation based on diffuse value is not the best idea either as it is profitable for some scenes, like Figure 5.1 and Figure 5.2.

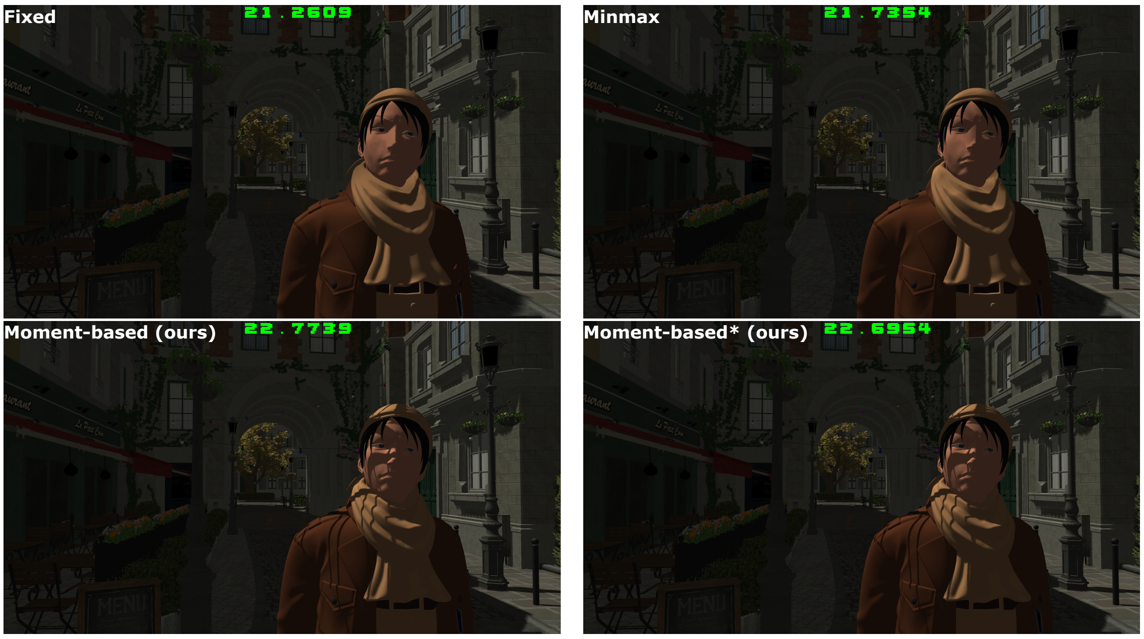


(a) Rendered with four cascades holding 512×512 shadow maps

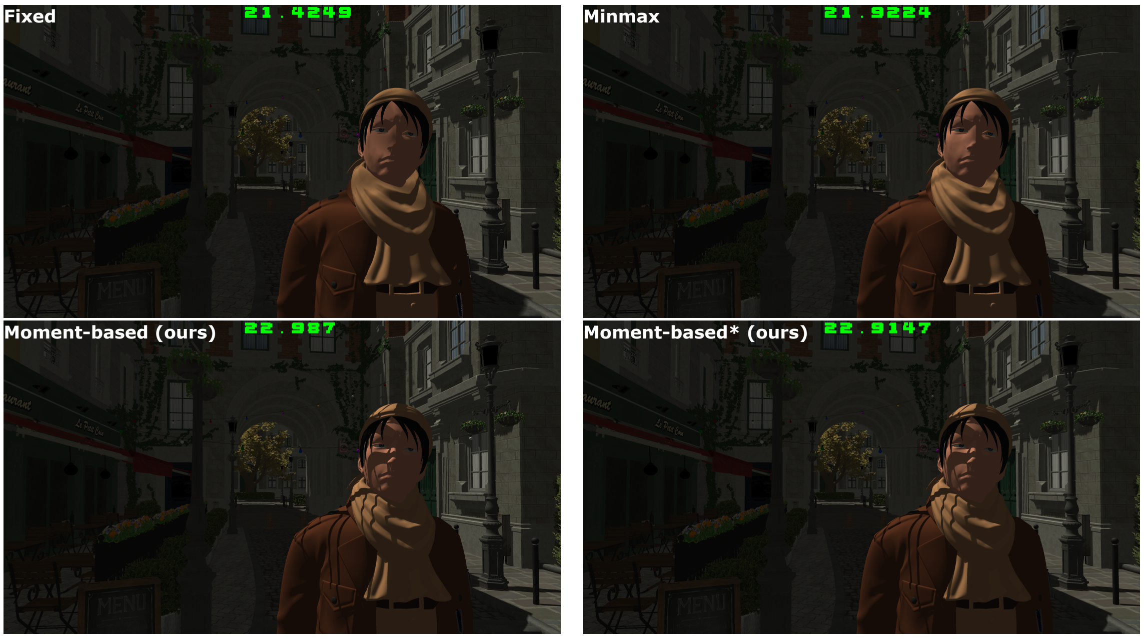


(b) Rendered with four cascades holding 1024×1024 shadow maps

Figure 5.1: Illustration of full frame times at 1920×1080 for a close-up of a character standing near a bistro [Lum17]. The * indicates that no pixel is invalidated based on a negative diffuse value.



(a) Rendered with four cascades holding 512×512 shadow maps



(b) Rendered with four cascades holding 1024×1024 shadow maps

Figure 5.2: Illustration of full frame times at 3840×2160 for a close-up of a character standing near a bistro [Lum17]. The * indicates that no pixel is invalidated based on a negative diffuse value.

	Moment-based (one reduce)	Moment-based* (one reduce)	Moment-based (two reduces)
Figure 5.1a			
Original	13.084	13.0885	13.1125
Fixed	12.8493	12.8349	12.8975
Result	0.2347	0.2536	0.215
Figure 5.2a			
Original	22.7739	22.6954	22.914
Fixed	22.5336	22.4372	22.6976
Result	0.2403	0.2582	0.2164

Table 5.2: Calculation of difference between moment-split computation and fixed computation in order to estimate the overhead of the split computation without the reduce. The * indicates that no pixel is invalidated based on a negative diffuse value.

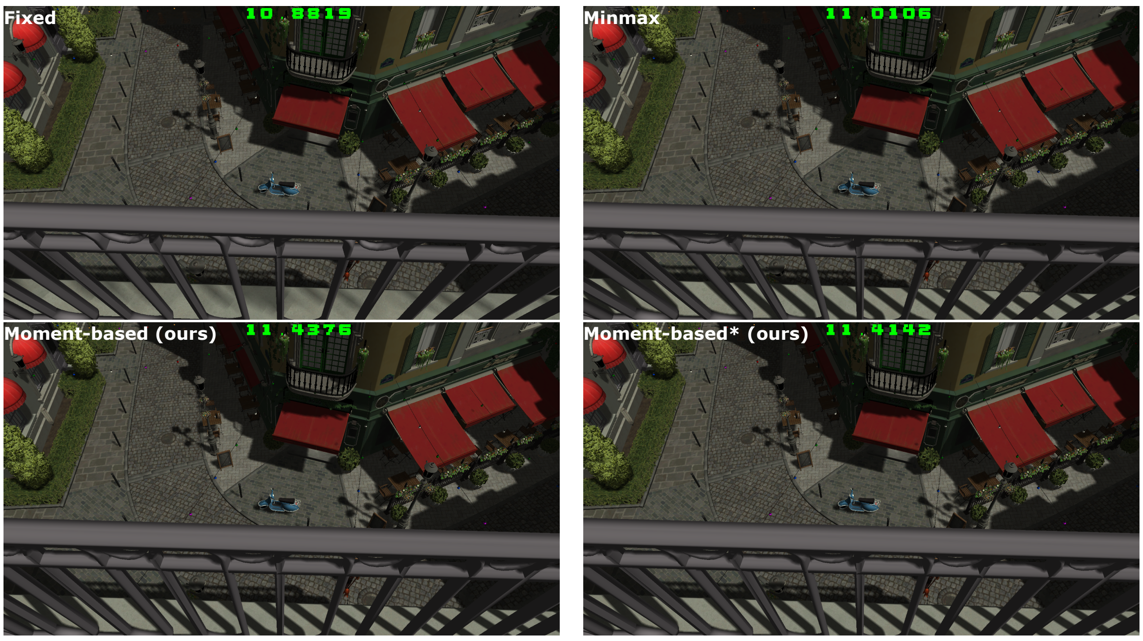


Figure 5.3: A scene seen from a balcony in 1920×1080 [Lum17]. The * indicates that no pixel is invalidated based on a negative diffuse value.

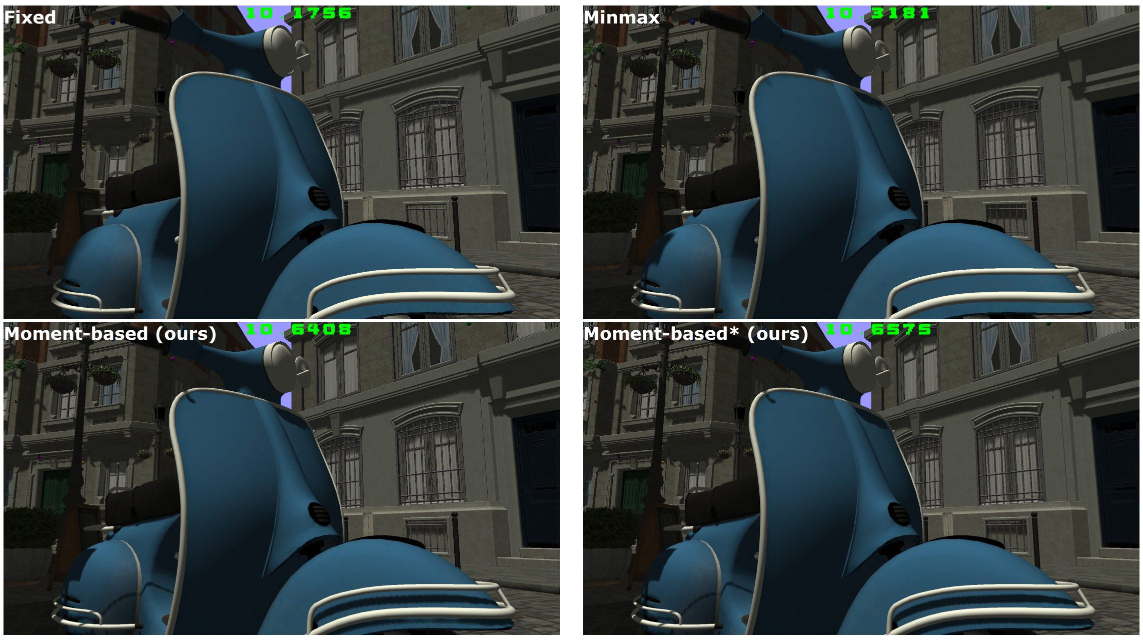


Figure 5.4: A a close-up of a scooter in 1920×1080 [Lum17]. The * indicates that no pixel is invalidated based on a negative diffuse value.

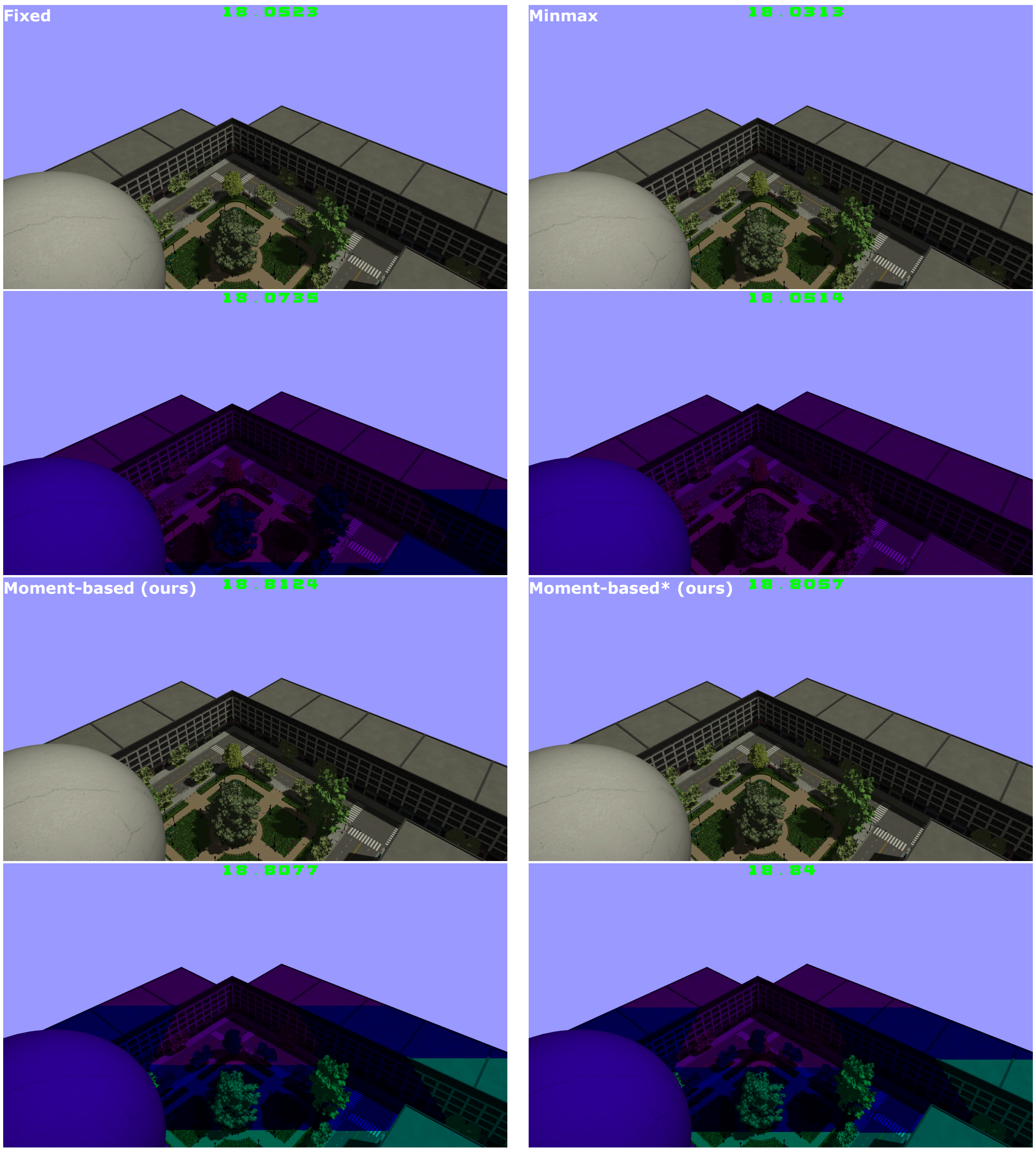


Figure 5.5: The worst case scenario for min max clamping in 1920×1080 [NHB17] (as described in Figure 2.16). The images below show the color coded cascades. The * indicates that no pixel is invalidated based on a negative diffuse value.

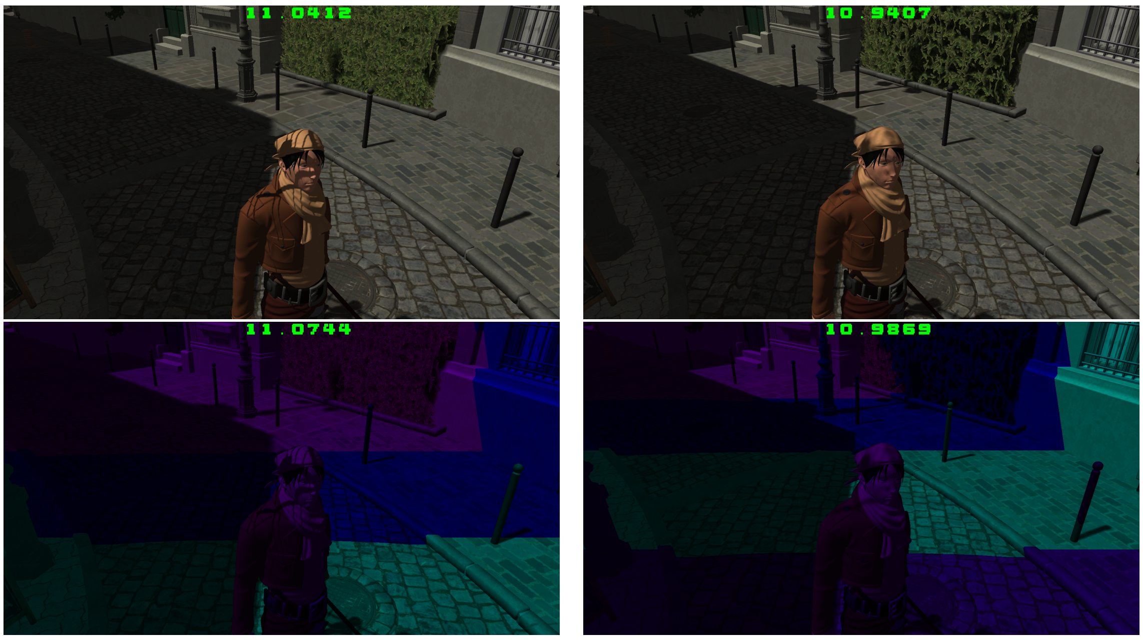


Figure 5.6: Illustration of possible temporal incoherence for minimal camera movement at 1920×1080 using our moment-based technique [Lum17].

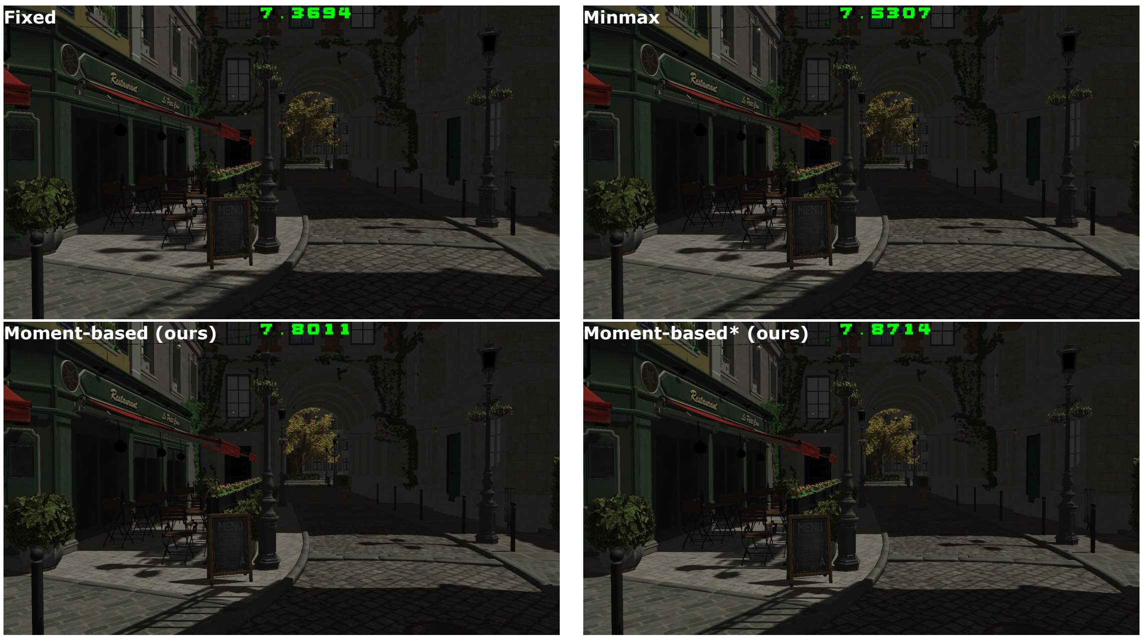


Figure 5.7: A common situation in video games rendered at 1920×1080 near the bistro [Lum17]. The * indicates that no pixel is invalidated based on a negative diffuse value.

6. Conclusions

Moment-based cascade splits are a fast and good alternative to existing split methods. We can achieve better results for many scenes with our technique with just a little overhead.

By using the moment-based reconstruction of the depth distribution, we can compute good split positions. As the reconstruction smoothens the cumulative distribution, we only get up to three positions that behave good for our purpose because these positions are located in the major gaps only. In contrast to that, we would get many minima, one for every tiny gap, by using the exact cumulative histogram.

These positions generate the most efficient cascades for the use of bounding box fitting. We thus have efficiently minimized cascades that improve the visual quality significantly.

Being able to guarantee that there are no unused cascades is another big benefit of using our technique. This property makes our technique even more efficient.

Moment-based splits are especially useful for video games that show many close-ups of objects or characters. It is also favorable to use our technique for scenes with multiple depth stages like skyscrapers, stairways or tall buildings in general.

Looking forward to future research on the subject of moment-based split computation, the temporal incoherence and adaptive exclusion of pixels still need improvement.

As of now this method marks another victory for the usage of moments in real-time rendering applications.

7. Acknowledgments

First and foremost, I would like to thank Christoph Peters for helping me write this thesis by creating the idea, giving me good feedback and always providing me with helpful comments on how to solve a problem.

I also thank Christian Bender, Kilian Kraut and Nicole Schaal for spell-checking my thesis.

Last, I thank the person who wrote the frame-work of the computer graphics exercise at KIT in 2019, as I used parts from this frame-work in order to load 3D-objects and used it to compute some figures in this thesis.

Bibliography

[DL06] W. Donnelly and A. Lauritzen, “Variance shadow maps,” in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ser. I3D ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 161–165. [Online]. Available: https://doi.org/10.1145/1111411.1111440

[Eng06] W. Engel, *Shader X5: Advanced Rendering Techniques.* USA: Charles River Media, Inc., 2006.

[Geo15] H.-O. Georgii, *Stochastik.* Berlin, Boston: De Gruyter, 2015. [Online]. Available: https://www.degruyter.com/view/title/496362

[Har07] M. Harris, “Optimizing parallel reduction in cuda,” 2007. [Online]. Available: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

[JLBM05] G. S. Johnson, J. Lee, C. A. Burns, and W. R. Mark, “The irregular z-buffer: Hardware acceleration for irregular data structures,” *ACM Trans. Graph.*, vol. 24, no. 4, p. 1462–1482, Oct. 2005. [Online]. Available: https://doi.org/10.1145/1095878.1095889

[KNL77] M. G. Krein, A. A. Nudel’man, and D. Louvish, *The Markov moment problem and extremal problems*, ser. Translations of mathematical monographs. Providence, RI: American Mathematical Society, 1977. [Online]. Available: https://cds.cern.ch/record/2623258

[Llo82] S. P. Lloyd, “Least squares quantization in pcm,” *IEEE Trans. Inf. Theory*, vol. 28, pp. 129–136, 1982.

[LSL11] A. Lauritzen, M. Salvi, and A. Lefohn, “Sample distribution shadow maps,” in *Symposium on Interactive 3D Graphics and Games*, ser. I3D ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 97–102. [Online]. Available: https://doi.org/10.1145/1944745.1944761

[Lum17] A. Lumberyard, “Amazon lumberyard bistro, open research content archive (orca),” July 2017, `http://developer.nvidia.com/orca/amazon-lumberyard-bistro`. [Online]. Available: http://developer.nvidia.com/orca/amazon-lumberyard-bistro

[Mac02] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms.* USA: Cambridge University Press, 2002.

[MHH18] T. Möller, E. Haines, and N. Hoffman, *Real-Time Rendering, Fourth Edition.* A K Peters/CRC Press, 2018, vol. Fourth edition. [Online]. Available: http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d1855548%26site%3dehost-live

[MKKP18a] C. Münstermann, S. Krumpen, R. Klein, and C. Peters, “Moment-based order-independent transparency,” *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 1, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3203206

[MKKP18b] C. Münstermann, S. Krumpen, R. Klein, and C. Peters, "Moment-based order-independent transparency," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 1, no. 1, pp. 7:1–7:20, May 2018.

[MT04] T. Martin and T.-S. Tan, "Anti-aliasing and Continuity with Trapezoidal Shadow Maps," in *Eurographics Workshop on Rendering*, A. Keller and H. W. Jensen, Eds. The Eurographics Association, 2004.

[NHB17] K. A. Nicholas Hull and N. Benty, "Nvidia emerald square, open research content archive (orca)," July 2017, `http://developer.nvidia.com/orca/nvidia-emerald-square`. [Online]. Available: http://developer.nvidia.com/orca/nvidia-emerald-square

[Pet17] C. Peters, "Moment-based methods for real-time shadows and fast transient imaging," Dissertation, University of Bonn, 2017, to appear.

[PK15] C. Peters and R. Klein, "Moment shadow mapping," in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, ser. i3D '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 7–14. [Online]. Available: https://doi.org/10.1145/2699276.2699277

[PMWK17] C. Peters, C. Münstermann, N. Wetzstein, and R. Klein, "Improved moment shadow maps for translucent occluders, soft shadows and single scattering," *Journal of Computer Graphics Techniques (JCGT)*, vol. 6, no. 1, pp. 17–67, Mar. 2017. [Online]. Available: http://www.jcgt.org/published/0006/01/03/

[Pre07] W. H. H. Press, Ed., *Numerical recipes : the art of scientific computing*, 3rd ed. Cambridge [u.a.]: Cambridge University Press, 2007, includes bibliographical references and indexHier auch später erschienene, unveränderte Nachdrucke. [Online]. Available: http://swbplus.bsz-bw.de/bsz267301596inh.htm

[RSC87] W. T. Reeves, D. H. Salesin, and R. L. Cook, "Rendering antialiased shadows with depth maps," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 283–291. [Online]. Available: https://doi.org/10.1145/37401.37435

[SD02] M. Stamminger and G. Drettakis, "Perspective shadow maps," *ACM Trans. Graph.*, vol. 21, no. 3, p. 557–562, Jul. 2002. [Online]. Available: https://doi.org/10.1145/566654.566616

[Str74] W. Straßer, *Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten*, ser. Heinrich-Hertz-Institut für Nachrichtentechnik Berlin, West: Technischer Bericht. Technische Universität Berlin, 1974. [Online]. Available: https://books.google.de/books?id=biStNwAACAAJ

[Wil78] L. Williams, "Casting curved shadows on curved surfaces," in *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '78. New York, NY, USA: Association for Computing Machinery, 1978, p. 270–274. [Online]. Available: https://doi.org/10.1145/800248.807402

[ZSXL06] F. Zhang, H. Sun, L. Xu, and L. K. 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*, ser. VRCIA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 311–318. [Online]. Available: https://doi.org/10.1145/1128923.1128975

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen.

Karlsruhe, den September 10, 2020

(Alexander Schipek)