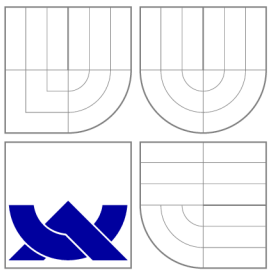
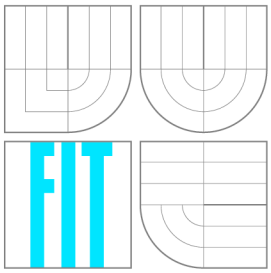


BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

HIGH-QUALITY SHADOW RENDERING FROM COMPLEX LIGHT SOURCES

KVALITNÍ ZOBRAZENÍ STÍNŮ PRO SLOŽITÉ SVĚTELNÉ ZDROJE

DOCTORAL THESIS
DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE

Ing. JAN NAVRÁTIL

SUPERVISOR
VEDOUCÍ PRÁCE

Prof. Dr. Ing. PAVEL ZEMČÍK

BRNO 2015

Abstract

In interactive applications, shadows are traditionally rendered using the shadow mapping algorithm. The disadvantage of the algorithm is limited resolution of depth texture which may lead to unpleasant visual artifacts on shadow edges. This work introduces an approach that is based on the improved texture warping. It allows for rendering a scene with the complex light sources, reduce the artifacts on the shadow boundaries and also improve the quality of the shadows regardless of the type of the scene and its configuration

Abstrakt

V interaktivních aplikacích jsou stíny tradičně zobrazovány s pomocí algoritmu založeným na stínových mapách. Nevýhodou toho algoritmu je, že stínová mapa, reprezentovaná texturou, má pouze omezené rozlišení. To může vést k nepěkným vizuálním artefaktům objevujících se na hranách stínů. Tato práce představuje postup, který je založen na vylepšené deformaci textury. To umožní zobrazit scénu obsahující složité světelné zdroje, zredukovat artefakty na hranicích stínů a také vylepšit kvalitu stínů bez ohledu na typu scény a její konfiguraci.

Keywords

realistic rendering, shadow maps, shadow rendering, omnidirectional light sources, aliasing, warping, shadows, GPU

Klíčová slova

realistické zobrazování, stínové mapy, zobrazení stínů, všesměrová světla, aliasing, deformace, stíny, GPU

Bibliographic citation

Jan Navrátil: *High-Quality Shadow Rendering from Complex Light Sources*, doctoral thesis, Brno, Brno University of Technology, Faculty of Information Technology, 2015.

High-Quality Shadow Rendering from Complex Light Sources

Declaration

I declare that this dissertation thesis is my original work and that I have written it under lead of Prof. Dr. Ing. Pavel Zemčík. All sources and literature that I have used during elaboration of the thesis are correctly cited with complete reference to the corresponding sources.

.....
Jan Navrátil
August 31, 2015

Acknowledgment

I would like to express my thanks to my supervisor Prof. Dr. Ing. Pavel Zemčík for his guidance and helpful advices during my whole studies and for encouraging me to realize my ideas. I am happy that he forced me to finish the thesis. Secondly, I would like to thank all my colleagues from Graph@FIT research group without whose professional and personal help this work would not have been possible. Special thanks goes to Roman Juránek and Markéta Dubská for their support, never ending discussion and proof-reading. You are good friends. Also, I like to thank Tomáš Milet for the long sleepless nights in the office and for his extensive knowledge of OpenGL and GPU programming. Last but not least, I would like to thank my wife Hanka for her love, her support and for all the hard work she had to do without me. The final thanks goes to my kids Lukáš and Linda for their patience. I owe them plenty of hours when I could not be with them.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

One of the real life tasks which benefit from computational performance of computers is the generation of images, animation and visualization. Computers can synthesize images in a nearly photorealistic quality and in real-time. Realistic rendering of global illumination has been considered the most time consuming part of the computer graphics for many years. The most difficult part of the evaluation is computing light transport and visibility correctly for every point of the entire scene.

In realistic image rendering, numerous visual phenomena have to be taken into account. They appear as a consequence of light scattering in the scene, e.g. caustics, reflections, and shadows. Each of these topics is worth discussing. In this thesis, the shadows for interactive applications will be further investigated.

Shadows constitute an important part of computer graphics rendering methods because they allow enhanced perception of the depth relations in the scene. Shadows help human viewers to correctly perceive object positions in the virtually created scene. Virtual scenes are often very dynamic, so if we want to achieve high quality shadows the algorithm has to be robust and work regardless of the scene's configuration, specifically the light and camera position. An example can be seen in applications for modeling and visualization where developers require fast and accurate shadow casting, independent from light types, camera position and scene complexity.

Shadow rendering in 3D applications has been investigated for many years. Various approaches have been published and their usage depends on the application and on the required quality of results. The main challenge is to evaluate a visibility between a rendered point and a light source. The highest quality is achieved with off-line rendering techniques, such as Ray Tracing or Radiosity. However, their rendering times are far from interactive rates. It can take hours or days to produce an high quality realistic image with radiosity or ray tracing. Frequently used algorithms in interactive applications are shadow volumes or shadow mapping. The shadow mapping algorithm is fast and easy to implement on GPUs despite its limitations in resolution and consequently in quality of rendering.

This thesis is mainly focused on resolving issues that appear in a shadow mapping algorithm. This algorithm renders the shadows in two steps. In the first step, the discrete representation of the scene is stored into a depth texture from light point of view. Then, the values in the texture are used for shadow computation from camera point of view. The representation of the scene is discretized because of the limited resolution of the texture. The resolution provides a number of samples that can be used for shadow computation. Since the textures are rectangular, sam-

ples are evenly distributed. This may produce artifacts on shadow boundaries and thus decrease the overall visual quality of the rendered image.

The purpose of this work is improvement of shadow quality in the shadow mapping algorithm. Some methods try to reduce the aliasing artifacts by adapting the distribution of samples to the current scene configuration. This mostly depends on the mutual position of the camera, the scene, and the light source. Typically, in outdoor scenes, modelling the sun as a complex light source is not very efficient, and actually not necessary. In this case, a directional light source is used as an approximation. For this simple light source, the parameterization of sampling distribution is very straightforward and easy to implement. In order to render shadows from more complex light sources, different approach needs to be employed in comparison to methods that deal with a directional light source or a spotlight.

When dealing with complex light sources in the shadow mapping algorithm, the representation of a scene in the depth texture has to be modified, and the texture generation process as well. Therefore, shadow quality improvement techniques that are successfully used with simple light sources are no longer directly usable.

The goal of this thesis is to introduce an approach that is able to render a scene with complex light sources, reduce aliasing artifacts on the shadow boundaries and also improve the quality of shadows regardless of the type of the scene and its configuration. The main contributions are improved shadow quality through better sampling of the scene, utilization of the shadow map warping for sampling improvement and evaluation of shadow quality.

The structure of the thesis is as follows. Overview of computer image synthesis methods and introduction to shadow rendering algorithms is presented in Chapter 2. Chapter 3 focuses on techniques that are based on the shadow mapping algorithm and it describes approaches for improving quality of shadows. Advanced techniques for complex light sources are also introduced. Improved texture warping and core of the thesis is presented in Chapter 4. Experimental results are discussed in Chapter 5.

CHAPTER 2

Global Illumination and Shadows

When people observe an image of some object, such as a room or a scene, they most likely want the image to look like a real photograph. It is mainly related to the images generated by a computer. Computers have to simulate at least how the scene is illuminated by a light. Computation of global illumination is considered as the most difficult and time consuming task in 3D computer graphics. However, high quality images can be produced, such that they are hardly distinguishable from real photographs.

The time complexity arises from a fact that these approaches simulate natural behavior of the light. Today, such simulation is usually based on geometric optics model where light travels in straight lines. The most difficult part of this simulation is evaluating mutual „visibility“ of every two points on the scene surface and summing up their light contribution. During the years, most of the expensive parts of global illumination have been replaced by simple models, or approximated by less complex algorithms capable of running in real-time [31]. This allows to render scenes with dynamic content, light sources and cameras, while retaining a plausible level of realism.

Shadow rendering is one of the areas where physical-based rendering is being replaced by simple algorithms in order to run the application at interactive rates. Shadows play an important role in any computer graphics image. They help to perceive spatial relationships between objects in a scene (see Figure 2.1), and they allow to understand significant visual and depth effects. Algorithms for computation of shadows and methods for improving the quality of rendered images are discussed further in the thesis.

Section 2.1 presents the most common approaches used for computation of global illumination. Algorithms for rendering shadows in interactive applications are described in Section 2.2. Finally, the Shadow Mapping algorithm, its basic concept and constraints are discussed in Section 2.3.

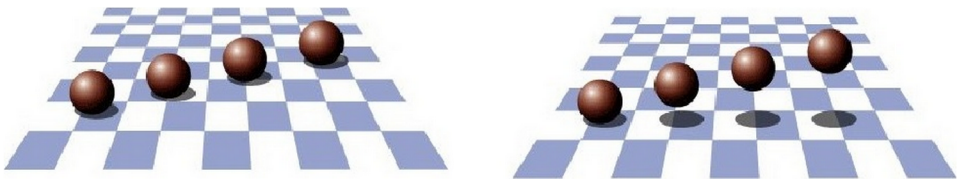


Figure 2.1: Shadows are important in computer graphics. Illustrations of the spatial relationship. Image courtesy of Mark Kilgard.

2.1 Realistic Image Synthesis

Methods for the computation of global illumination are able to synthesize images in photo-realistic quality, because they simulate physical rules of light distribution. This consists of light transport from a light source to an object in a scene or how the light is reflected from various kinds of materials, All these phenomena have to be taken into account and stored in the global scene description. This includes geometry data, properties of materials and specification of shape and properties of light sources. Physical based description of the properties is defined by a model. For better understanding of the properties, *radiometric quantities* [9] explain the physical measurements related to light energy. Based on this information, an algorithm evaluates the interaction of light emitted from the light source with the geometry in the scene.

The solution for computing a light transport through the scene is described in the following equation:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega}) \tag{2.1}$$

Here, L_o is output radiance of a point x in direction $\vec{\omega}$. L_o is computed from a radiance the point emits L_e, and a radiance L_r the point receives from surrounding geometry as the light travels through the scene.

Radiance is the most important quantity for global illumination. It can be thought of as the number of photons arriving per time at a small area from a given direction (see Figure 2.2).

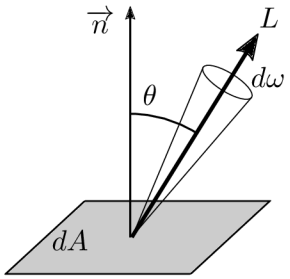


Figure 2.2: Radiance L expresses how much power arrives at (or leaves from) a certain point on a surface, per unit solid angle dw, and per unit projected area dA.

The interaction of the light with the geometry is described by the reflection model. The model is defined by *Bidirectional Reflectance Distribution Function* (BRDF) which expresses a relation between incident radiance L_i and reflected radiance L_r [9]:

$$f_r(x, \vec{\omega}', \vec{\omega}) = \frac{dL_r(x, \vec{\omega})}{L_i(x, \vec{\omega}')(\vec{\omega}' \cdot \vec{n})d\vec{\omega}'} \tag{2.2}$$

Here, x is point on the geometry, and $\vec{\omega}$ and $\vec{\omega}'$ are directions of the reflected and incident radiance L_r and L_i, respectively (see Figure 2.3). Note that $(\vec{\omega}'.n)$ is $cos\theta$, and expresses a geometric relation between a normal vector in the point x and incident direction $\vec{\omega}$.

The computation of reflected radiance in all directions is done by integrating L_i:

$$L_r(x, \vec{\omega}) = \int_\Omega f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}')(\vec{\omega}' \cdot \vec{n})d\vec{\omega}' \tag{2.3}$$

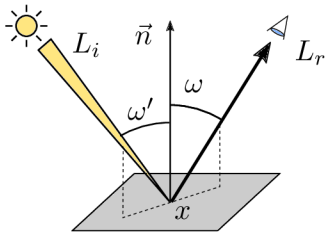


Figure 2.3: Bidirectional reflectance distribution function.

where Ω is the set of all directions, n is a normal in the point x.

When L_r in Eq. 2.1 is replaced by Eq. 2.3, it produces a complete description of the light transport known as *rendering equation* [13]:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_{\Omega} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}')(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}' \tag{2.4}$$

The rendering equation gives the mathematical basis for all global illumination algorithms (see Figure 2.4).

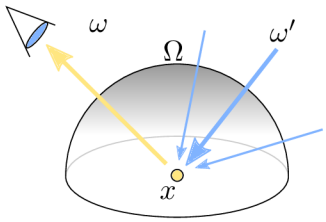


Figure 2.4: Illustration of light surface interaction defined by the rendering equation.

Most of the existing algorithms can be categorized into two basic groups. Firstly, *point sampling* approaches where the scene is evaluated independently for every pixel of the output image. Secondly, *finite elements* methods where the scene is divided into set of elements and the illumination is computed with respect to mutual relations of the elements. The following text gives a brief overview of representative techniques from for each group. Specifically, *Ray Tracing* and *Radiosity* approaches are presented.

2.1.1 Ray Tracing

The recursive Ray Tracing algorithm was firstly introduced for computer graphics in 1980 [38]. The idea of the algorithm is to trace rays emitted from a viewer through a scene, and investigate intersections with scene objects and light sources (see Algorithm 1). The main disadvantage is that it can solve only the basic tasks of global illumination: direct illumination, reflection and refraction of light. To solve more advanced effects such as indirect illumination, depth of field or motion blur, the basic algorithm has to be improved (see Figure 2.5). However, the Ray Tracing algorithm is a basic approach from which more advanced algorithms are developed.

```
Data: Scene geometry
Result: Rendered image
1 foreach object pixel in the output image do
2     foreach object in the scene do
3         if ray intersects an object then
4             select the frontmost intersection;
5             recursively trace the reflection and refraction rays;
6             calculate color;
7         end
8     end
9 end
```

Algorithm 1: The basic Ray Tracing algorithm.

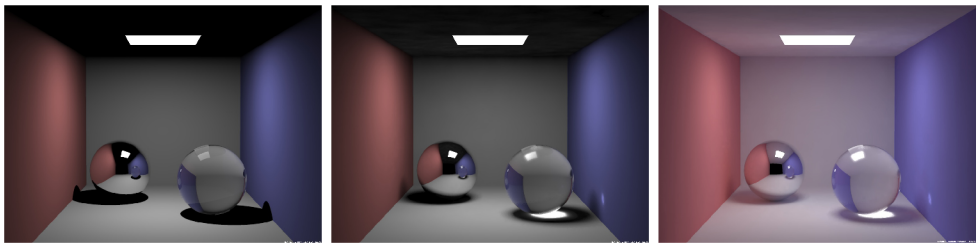


Figure 2.5: Sample scene rendered with various global illumination algorithms. Left to right: Ray Tracing, Distributed Ray Tracing, Photon Mapping [11].

2.1.2 Distributed Ray Tracing

To compute the advanced effects mentioned above, the basic Ray Tracing algorithm has to be extended with, for instance, Monte Carlo integration. In this case, multiple rays are stochastically distributed in order to simulate all possible paths of the rays. The initial idea comes from the *Distributed Ray Tracing* algorithm [5]. Figure 2.6 illustrates the key idea of the algorithm and it shows that the scene is oversampled with additional rays that have to be traced and then their contributions are averaged.

The rays can be distributed in various situations in order to get various effects. The most common effects are:

- anti-aliasing (pixel)
- soft shadows (area light source)
- motion blur (time)
- depth of field (eye)
- glossy reflections (direction of reflected ray)

The biggest disadvantage of this algorithm is that the number of rays has to be chosen wisely. A small number of rays introduces noise, while the number of rays emitted without any limitations can grow exponentially. For instance, Bidirectional

Path Tracing [16] addresses this issue and traces paths simultaneously from a light source and from a viewer.

The Distributed Ray Tracing algorithm increases realism in the rendered image by introducing stochastic sampling. The quality of the image is much better at the cost of increased computation time. The algorithm, however, has a disadvantage which comes from the stochastic rays distribution. It suffers from a large variance which results in a noise in the final image. When the rays distribution is chosen wisely, the noise can be reduced. Also, the performance is decreased with the large number of rays.

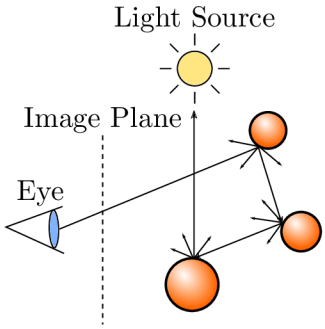


Figure 2.6: Distributed Ray Tracing

All algorithms based on Ray Tracing are point sampling approaches. The scene geometry acts as a „black box“. Paths of the rays are traced through the scene, which is expected to return a color of each currently rendered pixel in the output image. It can be useful for large complex scenes. On the other hand, the point sampling approaches hardly take into account mutual relations between objects in the scene.

2.1.3 Radiosity

Methods based on computation of radiosity solve the global illumination in a scene in a different way than the Ray Tracing algorithm. The methods are based on computation of equilibrium of light transport through a scene. The scene geometry is divided into small elements (patches) that are treated as secondary light sources (see Figure 2.8). Afterwards, during computation of the global illumination, the relations between individual patches as well as their properties are considered. The light scattering is computed by a system of linear equations which are derived from patches. Their relations are given by a *form factor*:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{V(x,x')G(x,x')}{\pi} dA_j dA_i \tag{2.5}$$

which expresses the relation between patches A_i and A_j. V is a simple visibility function that returns 1 if points are mutually visible, 0 otherwise. $G(x,x')$ is a function of geometric relationship between x and x'.

The Radiosity algorithm is generally view-independent as opposed to the Ray Tracing. It is capable of rendering high quality images namely for scenes with diffuse materials. However, it also has some issues that have to be addressed. The most

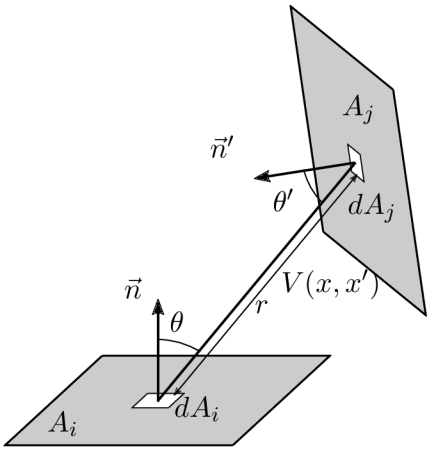


Figure 2.7: Illustration of the form factor equation.

time consuming part is computation of the form factors for every patch in the scene. Many improvements have been developed to address this problem [35], for instance, avoiding computation for distant patches that have negligible impact on the overall result. Since the Radiosity computes global illumination on the patches, the result might be inaccurate when the mesh is generated inappropriately. Consequently, the algorithm does not solve correctly sharp edges in the geometry, e.g. hard shadows are blurred.

Usually, the best results are achieved when the Radiosity is combined with the Ray Tracing, and the scene is rendered in multiple passes [41]. In the first pass, the shadow edges are detected and the grid is generated accordingly. In the second pass, the illumination is computed. The Radiosity algorithm computes indirect illumination and diffuse reflections, and the Ray Tracing computes specular reflections and shadows. Generally, computation of global illumination is a very time consuming process that depends on the scene complexity. However, the result is very realistic and with a high quality.

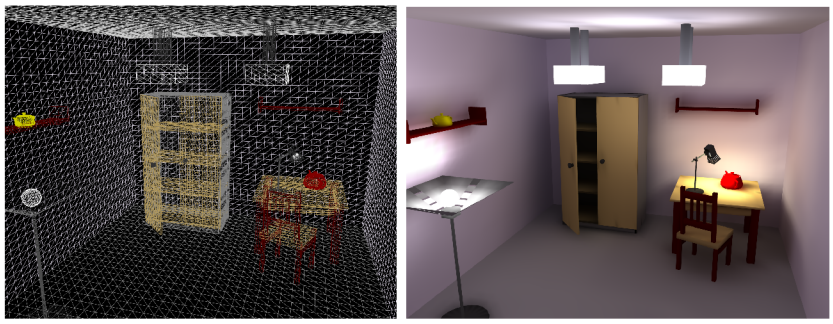


Figure 2.8: (Left) Every polygon in the scene is considered as a secondary light sources. (Right) The Radiosity algorithm simulates light distribution from various light sources and produces a realistic output image. Image courtesy of K. Dudka et al.

2.1.4 Photon Mapping

The idea of photon mapping is slightly different in comparison to Ray Tracing or Radiosity. The main difference can be seen in a different representation of the illumination information. Instead of storing the illumination tightly connected with the geometry, it is stored in a separate data structure called *photon map* [11].

The photon map stores data about photons emitted from a light source as they travel through a scene. It contains positions on a surface where the photons reflect, and also their energy. Separating illumination data from geometry is a crucial concept of the photon mapping algorithm. Firstly, the scene representation is much simpler. Secondly, the photon map can be successfully used even for complex scenes. The photon mapping algorithm can be combined with the Monte Carlo Ray Tracing and it leads to very efficient approach for rendering realistic images (see Figure 2.9). Currently, this is considered to be one of the approaches that generate the best quality images.

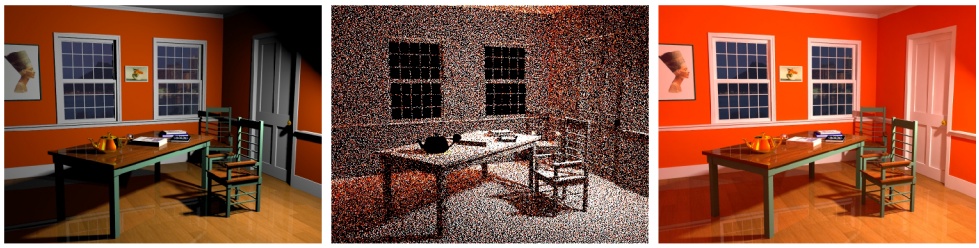


Figure 2.9: The image illustrates differences between the Ray Tracing (left) and photon mapping (right). Photon map is depicted in middle. Images courtesy of Per H. Christensen.

2.2 Shadows in Interactive Applications

Generation of realistic images takes second or hours in the global illumination algorithms. Computation of illumination in interactive applications requires approximation of the most expensive parts of the algorithms [30]. The problem can be divided into two parts.

Firstly, an expensive computation of BRDF on a surface that is currently lit can be approximated by shading models (e.g. Blinn-Phong, Cook-Torrance). In this case, form factors are not needed and the geometric relations are neglected. The shading models evaluate illumination based on position of geometry and light source. The shading models, however, do not provide any information of whether the surface lies in shadow or not.

The second part of the global illumination that has to be approximated is computation of shadows. The global illumination algorithms compute shadows either by evaluating intersection of shadow rays with geometry (Ray Tracing), or it results from a small number of photons in a photon map (photon mapping). Neither of these approaches is applicable in interactive applications without additional simplification [25].

In the following text, an overview of the most popular algorithms that are used

for rendering shadows in interactive applications are presented. Planar shadows and the Shadow Volumes algorithm are briefly introduced in the next sections. The Shadow Mapping algorithm is investigated in detail in Section 2.3 since the main contribution of the thesis is improvement of the algorithm.

2.2.1 Planar Shadows

The simplest algorithm for rendering shadows is based on geometry projection. The idea is to project an object to a plane. The projected planar geometry is then rendered as a separate object, and colored as a shadow - *planar shadows*.

However, the planar shadows algorithm is obsolete nowadays since it has a lot of disadvantages in comparison to modern accelerated algorithms. The shadows can be applied to planes only. Shadows can be reversed due to the projection transform. It is also difficult to blend the shadow with existing texture on the ground. Extra treatment has to be considered when casting shadows on finite planes.

The planar shadows algorithm uses basic operations that are available on graphics hardware from the beginning. Therefore, the algorithm was commonly used in the past when the graphics hardware was lack of acceleration units.

2.2.2 Shadow Volumes

The shadow volume technique for creating per-pixel correct shadows introduces a more general approach than planar shadows [2, 6]. The shadow volume is formed by an occluder, and it is a region in space where objects are occluded.

The basic idea of the Shadow Volumes algorithm is based on counting the number of intersections with the Shadow Volumes. The algorithm works as follows (see Figure 2.10):

1. Cast a ray from a camera through a scene (known as *z-pass* approach).
2. Increment the counter when the ray enters the shadow volume.
3. Decrement the counter when the ray leaves the shadow volume.
4. Repeat Step 2-3 until the ray hits some surface in the scene.
5. If the counter is zero, the surface that is hit by the ray is not in shadow. Otherwise, it is in shadow.

Steps 2 and 3 employs stencil buffer and stencil test. The implementation is very efficient since the stencil buffer support is crucial part of the rendering pipeline.

The algorithm, however, fails when a camera is inside a shadow volume. This issue is addressed by an alternative approach called *z-fail*. The z-fail algorithm (in comparison to z-pass described above) modifies the approach so as the ray is casted from the infinity towards the camera. It counts all entering and leaving intersections with Shadow Volumes until it hits the surface seen from camera. The evaluation of the counter is done in the same way as in the previous the z-pass approach.

As the main disadvantage of Shadow Volumes algorithm is often mentioned high fill-rate requirements for rasterization of the shadow volume geometry. This limitation was addressed by CC Shadow Volumes algorithm [22]. Other improvements are based on object silhouette computation that replace brute-force approach of casting

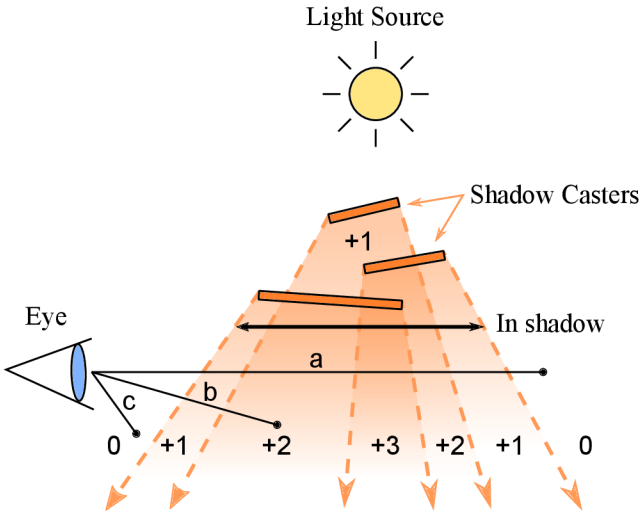


Figure 2.10: Illustration of the Shadow Volumes algorithm. The shadows appear when the counter is greater than zero (view sample b). Otherwise, the object is lit (view samples a and c).

shadow volume from each triangle. The volume is casted from object silhouette instead which considerably reduces amount of shadow volume geometry [24]. As other disadvantages are often considered difficulties to render soft shadows. Although, Assarsson et al. proposed an extension of the classical Shadow Volumes algorithm by introducing penumbra wedges [1].

In conclusion, the Shadow Volumes algorithm provides per-pixel correct results, and it is robust to be used on any model, without any restrictions of complexity, geometry or size. On the other hand, it is computationally expensive in the means of fill-rate and it does not support advanced effects.

2.2.3 Instant Radiosity

The level of realism in interactive application can be further increased. Contemporary graphics hardware supports features that allow for rendering high quality images with advanced illumination effects in real-time. In addition to the shadow rendering using algorithms from previous sections, indirect illumination can be also evaluated in real-time. This could be done under several assumptions. Firstly, high quality images do not need the full simulation of light transport, and only some parts of the simulation need to be computed. Secondly, it is also not necessary to have the accurate simulation in the dynamic scenes, because people cannot see all details if the image dynamically changes its content.

Existing approaches are focused on accelerating the most expensive parts of global illumination on GPUs. Radiosity algorithm introduce approach for distribution of light energy between patches. This leads to indirect illumination in the scene. Keller introduced the Instant Radiosity approach [14] that approximates the indirect illumination by virtual point lights (VPLs) that are generated in the scene. Ritchel showed that VPLs are sufficient to approximate direct and indirect illumination [33], especially when only a single bounce is processed [7].

The Instant Radiosity algorithm generates VPLs by tracing rays from a primary light source. The VPLs are placed in nodes of the path (see Figure 2.11) that is created as the rays travel through the scene. Then an image is rendered including shadows for every VPL using the Shadow Mapping algorithm. In a final step, the images are summed up in order to create the final image as can be seen in Figure 2.12.

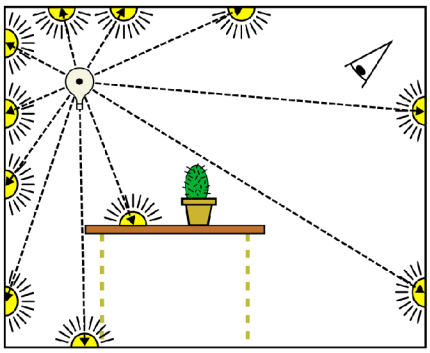


Figure 2.11: Rays are traced from the light source and VPLs are created on hit points [17].

The summing of contributions from each VPL is an iterative process. The first iteration starts on the light source and represents direct illumination. With every iteration, newly generated VPLs representing indirect illumination carry some portion of the light energy that depends on overall number of VPLs.

In the initial version of the Instant Radiosity, a raytracer was needed for exploring the paths. This solution is not suitable for interactive applications. When the shadow maps are used, all computation could be done on graphics hardware. Furthermore, no precomputed solution or data structures is needed, since the algorithm reuses data from previous frame.

A few problems could be seen in the Instant Radiosity algorithm. The data can be shared between frames unless the scene remains static. Dynamic objects and lights are possible, but many rendering passes are required to recompute the new VPLs positions. This could rapidly decrease frame rate. Further, when receiving surface is very close to point light, intensity value in frame buffer is overmodulated (Figure 2.12, right), so additional improvements would be needed. The approach is also limited to diffuse surfaces. The problem with glossy surfaces is that the reflections of VPLs are observed in the final image. The Instant Radiosity algorithm is base for other approaches [7, 33].

2.3 Basics of Shadow Mapping Algorithm

This section investigates basic principles of the Shadow Mapping algorithm [39]. It describes individual steps of the algorithm, its advantages and disadvantages. Further, it provides some implementations details in order to present all necessary aspects that are needed to render shadows on GPU.

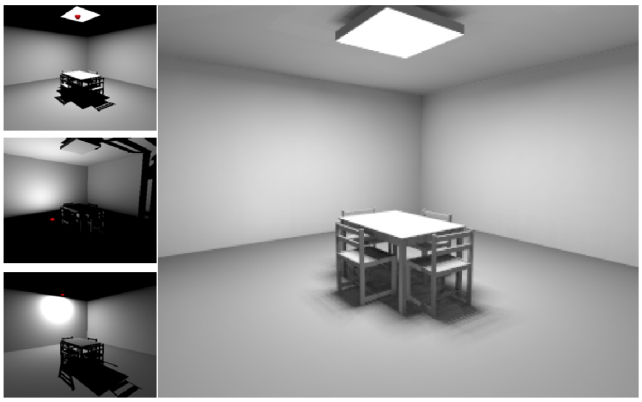


Figure 2.12: A few iteration steps. Red dot is a position of currently rendered VPL. Overmodulated image could be seen on the bottom left. Right is the intermediate image summed after 64 iterations [14].

2.3.1 Shadows in Two Steps

The key concept of all shadow rendering algorithms is that a point on a surface is considered to be visible to the light source if there is no occluder between the surface point and the light source. The visibility test might be a difficult and an expensive task for complex light sources.

In the Shadow Mapping algorithm, three basic types of the light sources can be considered: *directional light source*, *spotlight* and *omnidirectional light source* (see Figure 2.13). However, the algorithm can be used with some additional improvements for complex light sources as well (see Section 3.3). The directional light source is the simplest one. It is used mostly in outdoor scenes where most of the light comes from the sun which is considered to be in infinity. Because of this, all rays can be considered parallel. The spotlight is defined by its position and direction of a „cone“. The light is emitted from a point in space into the directions limited by the cone. The area of the illuminated part of the scene is defined by the field-of-view angle. The omnidirectional light source is also represented as a point in space, but it shines into all directions.

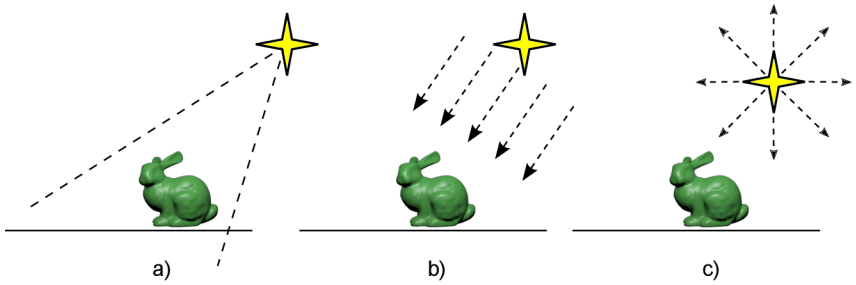


Figure 2.13: (a) A virtual camera for spotlights creates the light view frustum. The frustum covers only a part of the scene based on a direction of the spotlight. (b) Directional lights use orthographic projection, because direction the light rays are parallel. (c) Point light sources cast shadows into all directions.

A basic approach to decide whether some object is occluded by another is to compare its distance to a camera or an observer. In the rasterization pipeline, a depth buffer is used to store information about the distance of the object to the camera. In every pixel, the depth buffer stores only the value of the closest object. In the Shadow Mapping algorithm, the scene is rendered from the virtual camera in the position of the light source. Then, the depth buffer contains information about the distance to the objects that are closest to the light source. This implies that these objects are directly lit by the light source and everything behind them is in the shadow. The subsequent rendering pass from the camera point of view can read the depth information from the depth buffer and decide whether the surface being rendered is in the shadow or not.

To summarize, the Shadow Mapping algorithm renders image in two passes. In the first pass, it simply renders the scene from the light source point of view. The content of the depth buffer is stored in a texture. The texture is called a shadow map or a depth map. In the second pass, every surface point is transformed into the light space and its depth is compared with the depth value in the shadow map. If the depth of the transformed point is greater than the value stored in the shadow map then there is occluder between the point and the light source which means that the point lies in shadow (see Figure 2.14).

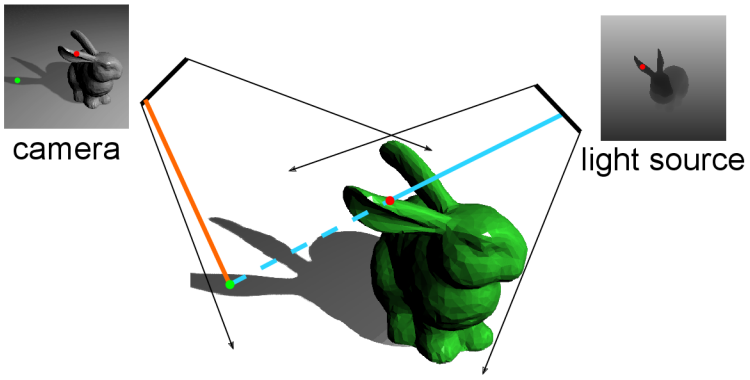


Figure 2.14: Illustration of the basic principle of the Shadow Mapping algorithm. The depth stored in the shadow map (red dot) is less than projected pixel visible from a camera (green dot).

2.3.2 Shadows on GPU

The Shadow Mapping algorithm is easy to implement since it does not require any complex data structures, or an expensive processing. However, a few important concerns have to be taken into account.

During the process of rendering the shadow map, a virtual camera is placed in the position of the light source and the geometry has to be transformed to the light space coordinate system. For this purpose, the model-view-projection (MVP) matrix has to provide an appropriate transformation. The model-view matrix simply transforms the geometry to the light space and projection matrix projects the geom-

etry to the target buffer. It depends on the type of the light source what projection matrix is applied or whether another projection mapping should be used.

Light rays of the directional light sources are parallel, and thus the light can be represented by orthographic projection. Spotlights are represented by a perspective projection, since it emits light from a point in the given direction. Light falloff corresponds to field of view of a camera. Because of the linearity of the transformation, the falloff angle is limited (as the camera field of view is), and the spotlight cannot cover the whole environment. To simulate omnidirectional light sources, multiple perspective projections (see Figure 2.15) or nonlinear transformation must be used.

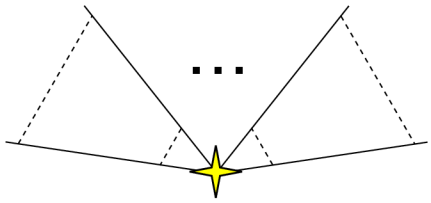


Figure 2.15: Multiple frusta have to be placed next to each other to cover the whole environment.

Naturally, to implement the shadow maps on contemporary computers, one must consider exploitation of GPUs. Contemporary GPUs have already integrated support for accelerated computation of the shadows. For instance in OpenGL, there is API function [1] that allows usage of 3-component vector for sampling the texture with the depth values:

```
float texture( sampler2DShadow sampler, vec3 P);
```

The function fetches the depth value from the shadow map using texture coordinates that are stored in the first two component of vector P. Then, the depth value is compared with the third component that should hold the referencing depth value of currently rendered fragment. The function returns 0, if the P. z value is greater that the value in the texture, otherwise it returns 1. The code snippet written in GLSL can look like:

```
1 vec3 texCoords = lightModelViewProjection * io_ObjSpacePosition;
2 texCoords.xyz = normalize( texCoords.xyz );
3 texCoords.z = (Length - near)/(far - near);
4 vec3 P = vec3( 0.5*texCoords.xy + 0.5, texCoords.z);
5 float shadow = texture( shadowMap, P);
6 vec4 fragColor = shadow*color;
```

From the above fragment of code, it is, hopefully, obvious that the shadow map implementation in contemporary GPU is straightforward and efficient.

[1] `https://www.opengl.org/documentation/glsl/`

2.3.3 Shadow Mapping Issues

The Shadow Mapping algorithm is considered to be very efficient and flexible approach, but it suffers from some issues and visual artifacts including aliasing. As the depth information is usually stored in a texture, the size of the texture represents the total number of depth values that can be fetched in order to compute a shadow. It is common that multiple surface points with different distances to the light source are projected to a single shadow map texel. This incorrect sampling rate leads to unpleasant visual artifacts and aliasing. Section 3.1 explains the aliasing in Shadow Mapping and methods for its elimination in detail. The following text discusses the most common visual artifacts produced by Shadow Mapping that are immediately noticeable.

Artifacts caused by wrongly computed self-shadow that arises on the object surface is called a *surface (shadow) acne*. The depth value in the shadow map is quantized, but points from the surface do not all have the same depth. Consequently, some fragments on the surface lie in shadow while other fragments are considered as lit by the light (see Figure 2.16).

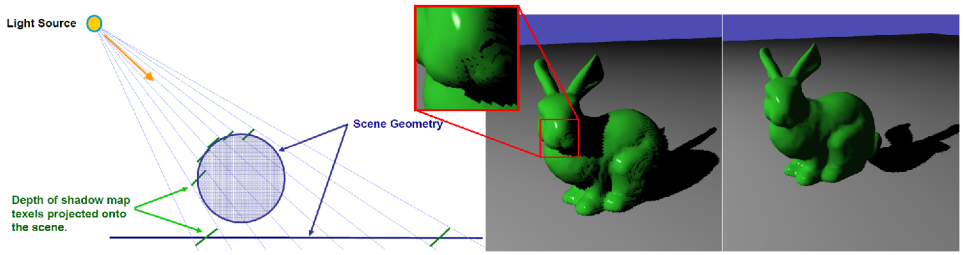


Figure 2.16: Illustration of source of the shadow acne (left). Depicted surface acne (middle) and Peter-panning effect (right).

The solution could be adding some bias to the surface in order to eliminate a difference along the pixels. A slope of the surface might be take into account to achieve a better results. For instance, the polygon offset is successfully used for this purpose. Simultaneously, when rendering objects with closed geometry, the front face culling can be enabled. It causes that the depth map stores distances to the polygons farther from the camera.

However, an excessive usage of the bias could lead to another artifact on shadows. This second common visual artifact, called disconnected shadow, or *Peter Panning*. It makes the shadow detached from the object and the object appears to be floating in the air. This usually happens when the algorithm compares two depth values that are close to each other. When the bias is applied, the shadow test may mistakenly evaluate the fragment to be lit.

To be sure that the shadow test pass for correct fragments, the bias has to be adjusted. Also, the view frustum of the light source has to fit as much as possible in order to improve precision of discrete depth quantization.

The jagged edges caused by a small resolution of the shadow map are another unpleasant visual artifact. The simple solution is to filter the shadow map on multiple samples when fetching the depth value.

Standard filtering that is embedded in graphics hardware cannot be used for this

purpose. When the hardware texture filtering was applied, the fetched depth value would still be one value produced as an average of adjacent texels.

Percentage-closer filtering (PCF) [32] addresses the problem by sampling multiple depth values and count ratio of how many of them pass the depth test. The number of sampled shadow map texels controls how much the shadow edge is blurred (see Figure 2.17). Another solution that improves PCF is to use variance shadow maps [8]. This allows for the hardware texture filtering to be applied on the shadow map with all its additional feature.

Figure 2.17: Detail of a shadow with various number of samples: 1 (a), 4 (b) and 16 (c) [4].

Section 3.3 further in the text introduces two approaches for rendering shadows cast by omnidirectional light sources that can be source of additional issues. The first one is based on the combination of multiple linear projections and it stores depth values into cube maps. The second is based on nonlinear parabolic projection.

However, every nonlinear function embedded into the traditional rendering pipeline „goes against“ the linear interpolation scheme used in the graphics hardware. When the projection transformation represented by a matrix is applied on vertices in the vertex shader, the position, color and other data in fragments are then linearly interpolated in the fragment shader. Generally, any kind of projection function can be applied on vertices. If the results of the projection causes that triangle edges are curved, the linear interpolation makes them straight again. Figure 2.18 illustrates this discrepancy that leads to unwanted artifact for large polygons. The solution for these artifacts is to refine tessellation of the scene. For small polygons, the artifacts are not noticeable.

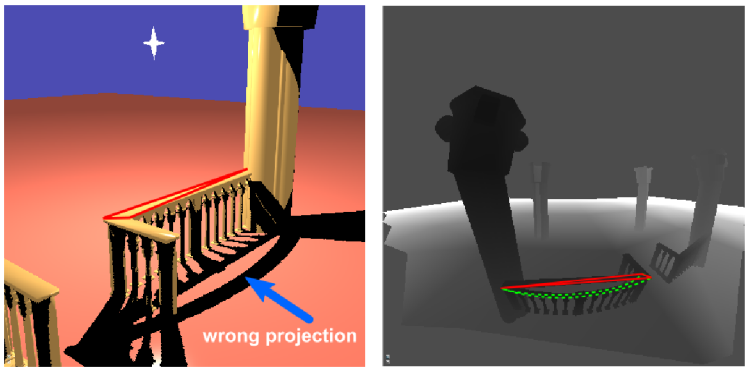


Figure 2.18: Fragments that have to be rasterized between two vertices (left, red triangle) are linearly interpolated in fragment shaders even if the projection is non-linear (right, red triangle). The correct solution of the non-linear parameterization is depicted by green dashed line (right).

CHAPTER 3

Shadow Quality and Complex Light Sources

As the basic Shadow Mapping algorithm has been introduced in Chapter 2, this chapter focuses on visual quality of rendered images. It describes how the quality of shadows is influenced by incorrect sampling and aliasing, and how the aliasing error can be measured. Further, the chapter presents some optimization techniques that eliminate disadvantages of the Shadow Mapping algorithm related to visual artifacts. It shows that the techniques are designed and implemented for simple light sources where they achieve good results.

This work, however, focuses on improvement quality of shadows for complex light sources where the techniques for simple light sources fail. This chapter provides an overview of various methods for rendering shadows cast from omnidirectional light sources. It describes the principles of each method and discusses their advantages and disadvantages.

3.1 Deriving the Error Metric

The aliasing in the Shadow Mapping algorithm is a significant visual artifacts. It appears namely on shadow edges due to low resolution of the shadow map, because a single shadow map texel cannot cover all the details for object further from light sources (see Figure 3.1). The shadow map sampling rate is typically insufficient to handle all the scene details sufficiently well.

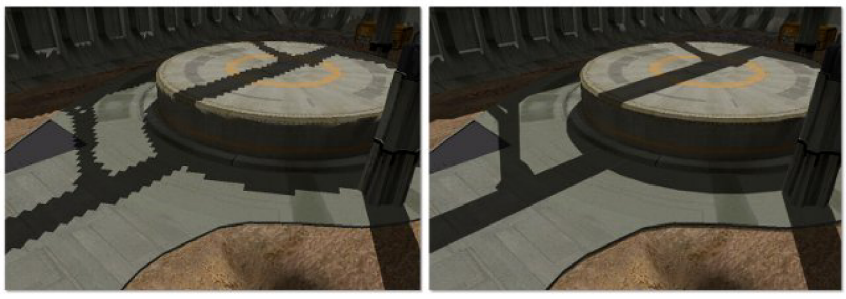


Figure 3.1: (Left) Multiple view samples projected on one shadow map texel. (Right) The mapping is correct, no aliasing is observed [40].

This section describes the origin of the aliasing error in the Shadow Mapping algorithm. Further, it presents an approximation of the aliasing error based on

mismatch between sampling from a camera point of view and a light source point of view (see Figure 3.2). More comprehensive analysis, derivation and quantization of the aliasing error was introduced by Lloyd [19].

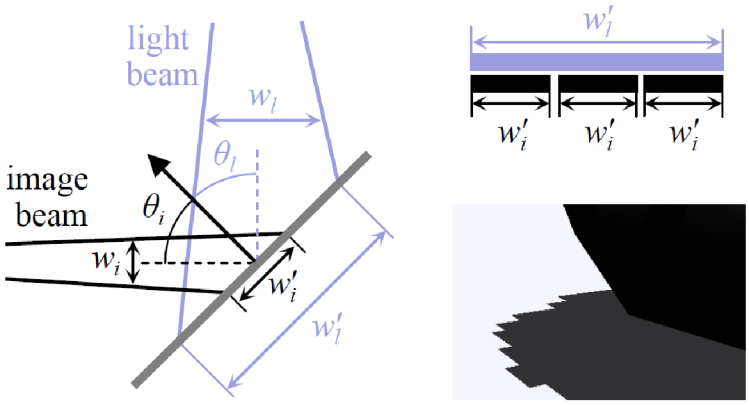


Figure 3.2: Illustration of beams projected from eye and light source and their widths on a surface [21].

3.1.1 Geometric Derivation

The sampling scheme is very similar for both camera and light source. The camera and the light source sample scenes through pixels representing rectangular areas. All the light rays going through the given rectangular pixel and the light source or the camera form a beam defined by the pixel. Given a beam from the camera projects through a pixel onto a scene surface with width w_i'. Similarly, a beam from the light source through a shadow map texel is projected on the surface with width w_l'. The aliasing error on such surface can be approximated by the ratio of the projected beam widths:

$$m = \frac{w_l'}{w_i'} \tag{3.1}$$

As it can be seen in Figure 3.2, the aliasing error does not depend only on beam widths and distance of the surface to the camera or light source but also on surface orientation. Stamminger et al. [36] described these two types of aliasing: *perspective* and *projection*. The aliasing error according to Stamminger can be quantified as:

$$m = \frac{w_l'}{w_i'} \approx \frac{w_l}{w_i} \frac{\cos\theta_i}{\cos\theta_l} \tag{3.2}$$

where w_i and w_l are the widths of the image and light beams at the point of intersection and θ_i and θ_l are the angles between the surface normal and the beam directions.

Perspective aliasing is caused when the shadow map is undersampled because of light source distance while projection aliasing appears when the direction of light rays is parallel to the surface so that shadow stretches along the surface. The perspective aliasing is the most common one in the Shadow Mapping algorithm. It occurs when more than one point on the geometry is projected to the single texel in the shadow map. As can be seen in Figure 3.3, pixels by the near plane are more dense in

post-perspective space than pixels by the far plan. However, the sampling rate of the shadow map remains the same over the entire view frustum. Because of this, more pixels map to the same texel in the shadow map.

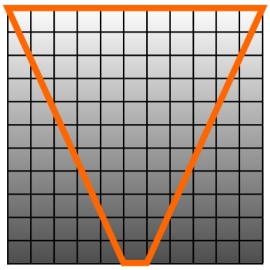


Figure 3.3: One shadow pixel is projected on many samples in front of the camera.

Some methods exist that attempt to reduce the perspective aliasing artifacts on shadow boundaries. The shadow map can be filtered [4] that causes the shadows to be smooth which, however, is not always desired. The „correct“ approach would be to use high shadow map resolution for objects near to the camera and low resolution for distant objects. Naturally, the high resolution is not necessary for shadows far from camera as the fine scene details are not visible due to perspective projection.

In some approaches [10], multiple shadow maps with different resolutions are used. They are stored in a hierarchy based on resolution and they adapt to the level of detail desired in individual locations of the rendered frame. Unfortunately, this technique needs multiple rendering passes and some additional data structures, so acceleration in hardware is not efficient.

The projective aliasing is not easy to eliminate and since it is less intrusive, existing approaches neglect it. The following sections describe how the perspective aliasing can be reduced which leads to elimination of jagged edges in the output image. It happens when the width of the image beam w_i equals the to the light beam width w_l.

3.1.2 Scene Sampling

The Shadow Mapping algorithm works with two types of samples. View samples are pixels that correspond to points on a scene surface described by their 3D position (and other properties such as color, normal vector etc.). They are generated by sampling the scene from a camera point of view. Shadow samples are generated by sampling the scene from a light source point of view. In both cases, the sampling is performed using an orthogonal grid with a predefined resolution.

However, multiple view samples can be projected onto one shadow sample and then aliasing can be observed in a final image as jagged edges of the shadows. This is caused by uniform rasterization of a texture produced by a graphics hardware. One solution is to parameterize the sampling using a warping function. The function enlarges important parts of a scene in order to increase shadow sampling rate. This technique increases a probability that shadows for different view samples are resolved by different shadow samples. There are two types of the warping function - *global* and *local*. The global warping function can be defined by a transformation matrix. This warping function mostly depends on a mutual position of a camera, a light

source and geometry and ignores properties of view samples [36]. The local warping function is derived from properties of view samples and scene analysis [12, 34]. These approaches are described in detail in the following sections.

3.2 Methods for Reducing Aliasing

Some methods exist to reduce the aliasing errors caused by the sampling mechanisms used in the Shadow Mapping algorithm. As the shadow map size is typically given by the hardware limitations, these methods exploit non-uniform sampling of the shadow maps either through non-linear mapping or using discrete smaller maps with different resolutions.

3.2.1 Perspective Shadow Maps

Perspective Shadow Maps [36] differ from standard shadow maps in that they are generated after perspective transformation, i.e. in normalized device coordinates. It causes reduction of the perspective aliasing (see Section 3.1) on the shadow boundaries in a rendered image.

The transformation is projective and thus can be represented by a matrix in homogeneous coordinates. This matrix is used to project a current viewing frustum to a unit cube. The same matrix is applied on the view frustum before rendering of the shadow map. After that, the shadow map is generated by parallel projection of the transformed space (see Figure 3.4). In the post-perspective space, the unit cube is sampled with the same sampling rate. Objects that are closer to a camera receive the same amount of samples as the distant objects that are smaller and thus not need a full detail of the shadow.

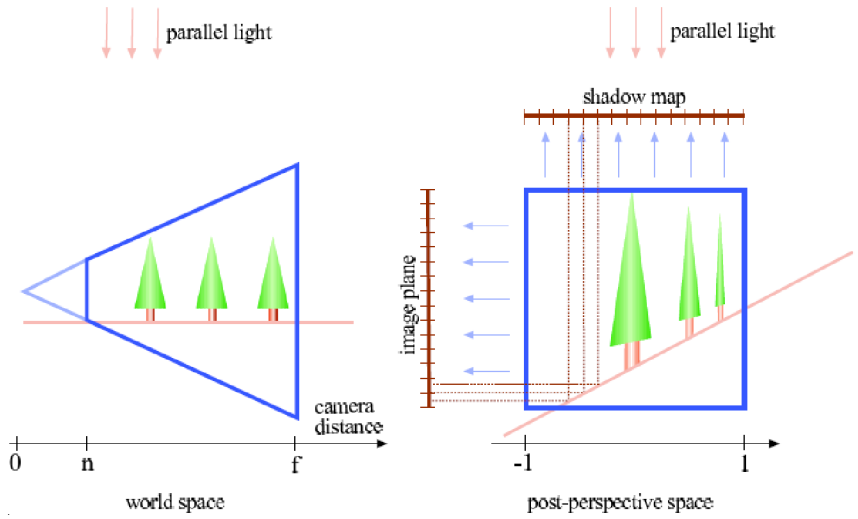


Figure 3.4: Shadow map for a parallel light is generated after perspective transformation [36].

When transforming a scene to the post-perspective space, not only the camera view frustum needs to be projected, but also objects outside the view frustum. They can also cast shadows on other objects visible in the view frustum and consequently

they have to be rendered into the shadow map. An extra attention has to be taken on shadow casters behind the camera, because the perspective transformation projects all objects behind the camera beyond the infinity plane. These objects are then not rendered into the shadow map.

Stamminger et al. suggested solution to virtually move the camera backwards in order to get all objects that participate on shadows in front of the camera. This solution modifies the post-perspective space so as it may eventually decrease the effect of the algorithm on the perspective aliasing. In the worst case, it degrades to the standard shadow map algorithm where the perspective aliasing is still presented.

Further, the algorithm has to deal with the light sources in transformed space since the same projection matrix is applied on the light sources as well. The light source may change in transformed space and it depends on its type and on the initial position related to the camera. Directional light sources becomes point light sources in post-perspective space mapped to an infinite plane. In extreme case when the direction of the light source comes from behind the camera, the transformed point light source is inverted. It means that objects are in wrong order. Naturally, this may affect the computation of shadows. In this case, it is necessary to invert the depth test so as the furthest point is considered to be lit. All other points with smaller depth are in shadow.

The similar situation is observed for spotlights as well. Stamminger et al. described some most common cases. For instance, when the spotlight is on the same plane as a camera, it is converted to the directional light source in post-perspective space.

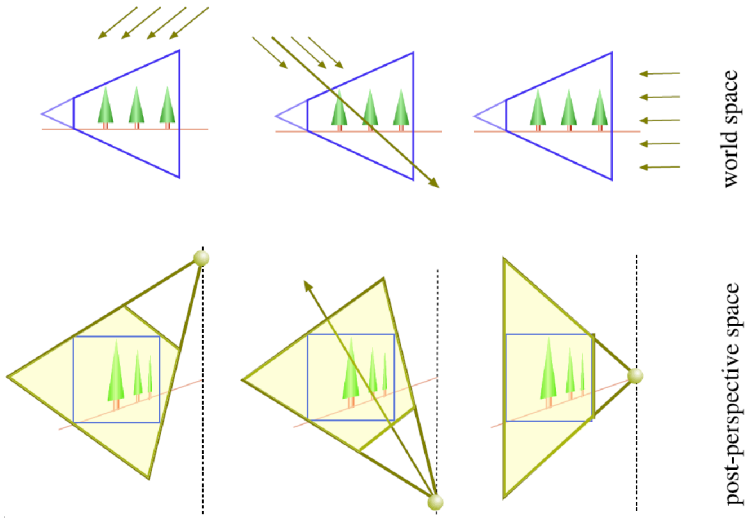


Figure 3.5: Directional light source in world space (top) and in post-perspective space (bottom) [36].

Although, PSMs significantly decrease the perspective aliasing, the projection aliasing is not treated. The approach is view-dependent and requires additional processing to resolve some corner cases. Also, the depth quantization is neglected, the surface acne and self-shadowing are emphasized due to non-uniform scaling. This could be partially eliminated by setting a reasonable offset. But this setting has to be done by user depending on the scene complexity.

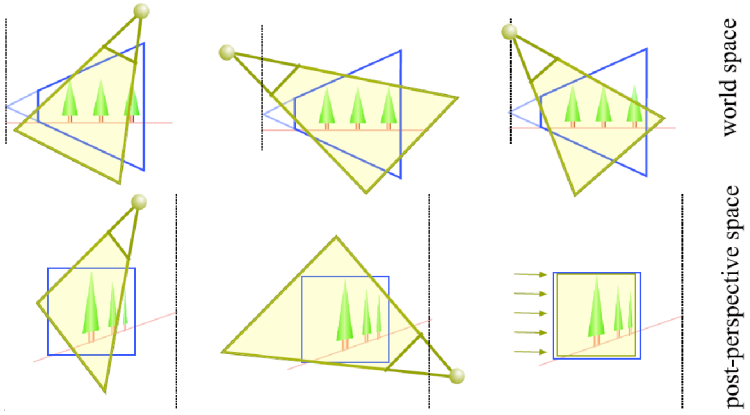


Figure 3.6: Point light source in world space (top) and in post-perspective space (bottom) [36].

3.2.2 Parallel-Split Shadow Maps

The perspective aliasing has its maximum values close to the near plane. With increasing distance from the camera, the aliasing decreases until it reaches a distance where the shadow map starts to be oversampled (see Figure 3.7). From this point, the shadow map resolution is used inefficiently. This can be easily observed in large environments where the shadow map for a directional light source has sample the entire scene.

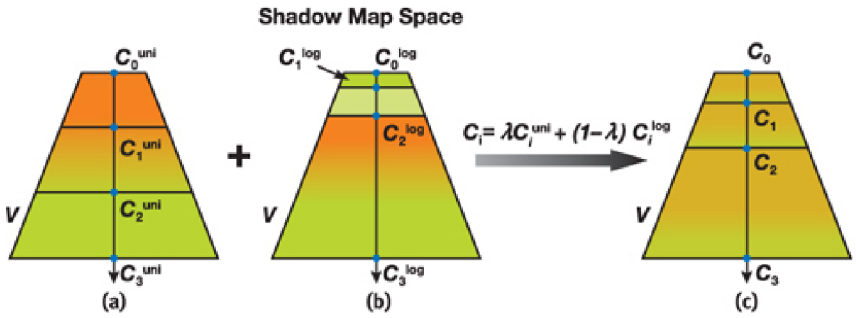


Figure 3.7: Practical Split Scheme combines uniform and logarithmic schemes [40].

The idea of the *Parallel-Split Shadow Maps* approach [40] is to split the view frustum in a certain distance from the camera into several parts in order to minimize the oversampled areas and thus make use the shadow map efficiently. For each part, an independent shadow map is rendered. In this way, the sampling rate of the shadow map is optimally distributed and the perspective aliasing is reduced. The algorithm is performed in the following steps:

```
1 Split the view frustum into predefined number of parts;
2 foreach part do
3 |   Derive the projection matrix;
4 |   Render the shadow map;
5 end
6 Render a scene with shadows;
```

The number of parts could be relatively small (3-5 parts). The main issue of the algorithm is to find the split positions so as the perspective aliasing is reduced, or at least it is constant throughout the entire frustum.

Zhang et al. showed that this requirement is fulfilled only with the *practical split scheme*. The scheme determines the split positions by equation:

$$C_i = \lambda C_i^{log} + (1 - \lambda) C_i^{uni} \qquad 0 < \lambda < 1 \tag{3.3}$$

where C_i^{log} and C_i^{uni} are split positions, and λ is the weight that controls the split positions according to requirements of the application (the default value suggested by Zhang is $\lambda = 0.5$). The practical split scheme combines two basic schemes - logarithmic and uniform.

In the uniform scheme, the split positions are evenly distributed along the view direction. The distribution of the alias error in each part is the same as in the standard shadow map. However, the difference between undersampled and oversampled areas are not so distinctive (see Figure 3.7). The split positions are determined from the properties of the view frustum by equation:

$$C_i^{uni} = n + (f - n)\frac{i}{m} \tag{3.4}$$

The logarithmic split scheme allows an even distribution of the perspective aliasing. The main disadvantage of this approach is that the determined split positions are close to the near plane. It reduces the aliasing in front of the camera, but it is still observable further in the scene (see Figure 3.7). The scheme is defined by:

$$C_i^{log} = n \left(\frac{f}{n}\right)^{\frac{i}{m}} \tag{3.5}$$

In both uniform as well as logarithmic split scheme, the perspective aliasing is still presented. Therefore, Zhang et al. suggested to combine these two approaches, because it was shown that they together produce the best results.

In order to achieve the best results, some steps have to be done prior to generating the shadow map in Step 3 of the algorithm. To gather only the part of the scene that belongs to the current split view, the projection matrix has to be defined individually for each part. The projection matrix has to cover the entire split view as well as all shadow casters outside the camera view frustum. In this case, the projection matrix definition is scene-independent. The scene-dependent approach is more restrictive and it takes geometry into account. The projection matrix is defined only to cover objects that can cast shadows (see Figure 3.8).

The rendering of the shadow maps may introduce performance bottleneck. In the traditional approach where the shadow rendering was not accelerated, the shadow

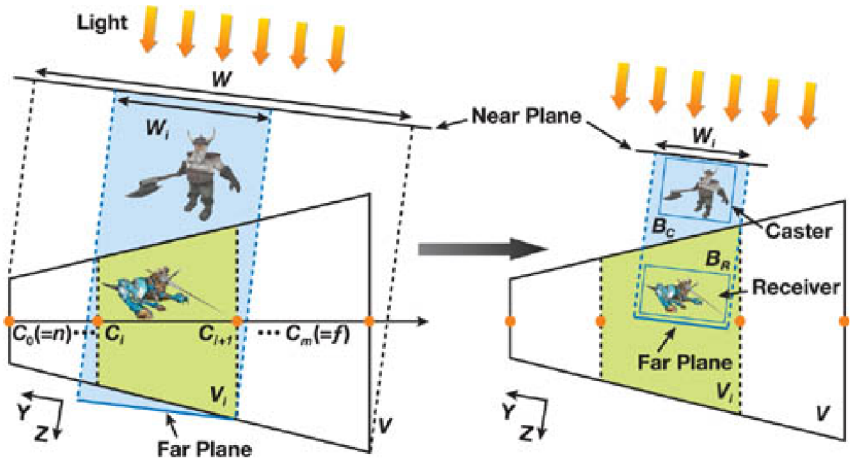


Figure 3.8: Definition of projection matrix is scene-dependent (left), or scene-independent (right) [40].

maps for each part had to be rendered separately. With contemporary GPUs, this step can be performed in one render pass with help of e.g. MRT (multiple render targets) or geometry shaders.

PSSMs are considered to be very efficient approach for rendering shadows in large environments. Zhang et al. showed how the technique is implemented for directional light sources and he also claimed that the technique can be used also with point light sources. In this case PSSMs, it is thought of as spotlights where light is emitted from a single point, but it is culled by a frustum. Use of PSSMs for omnidirectional light sources are not directly applicable without major improvements.

When implementing this technique, a programmer should focus on areas in a scene where two parts are connected. Visual artifacts can be seen due to different resolution of shadow maps in both parts (see Figure 3.9). However, it can be easily fixed with a shadow map filtering, e.g. with PCF, as explained in Section 2.3.3.

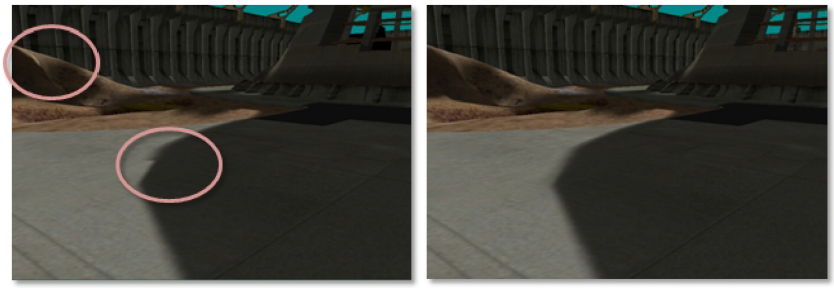


Figure 3.9: Illustration of artifact on boundary of two shadow maps [40].

3.2.3 Rectilinear Texture Warping

Rosen [34] introduced an adaptive shadow mapping approach that also addresses the aliasing issue. He suggested the *Rectilinear Texture Warping* (RTW) technique that is capable of rendering quality shadows. Unlike CSMs, the RTW uses only one shadow map to cover the entire scene and a set of importance functions for adaptive scene sampling. The shadow map can be generated per-frame, and it supports fully dynamic scenes. As the camera and the light source moves, the RTW adaptively changes the sampling rate, whereas the standard shadow map remains unchanged as it can be seen in Figure 3.10.

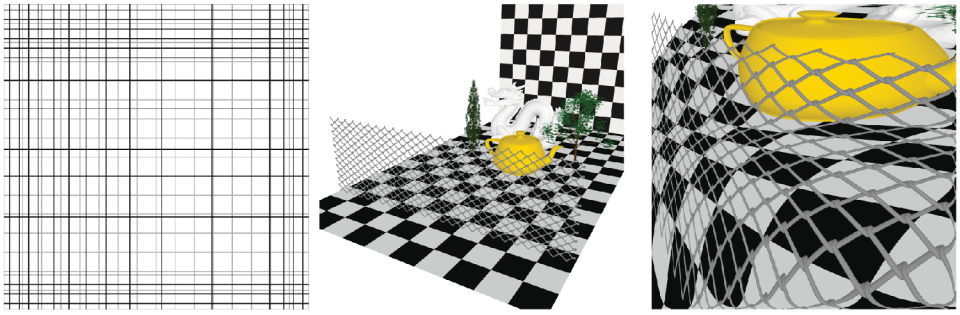


Figure 3.10: Illustration of rectilinear warping scheme [34].

The crucial step in the algorithm is creating the importance map. The importance can be analyzed in three different ways:

- **Forward** - Firstly, the depth map is rendered from the light point of view. Then, the depth map is analyzed and the importance map is built.
- **Backward** - The depth map is rendered from camera point of view, projected to the light space and analyzed.
- **Hybrid** - Combination of most valuable results from both approach in cost of higher computation time.

The importance analysis in all options is performed using an arbitrary number of analytical and heuristic-based functions. The output of the analysis is the *importance map* (Step 1 of the Algorithm 2) which serves as an input to next steps of the algorithm:

```
1 Build the importance map
2 Convert 2-D importance map into 1-D warping maps
  begin
3 |   Collapse rows/columns to 1-D importance maps
4 |   Blur importance maps
5 |   Build warping maps from importance maps
  end
6 Render the RTW shadow map
7 Render the output image from the desired view
```

Algorithm 2: RTW algorithm

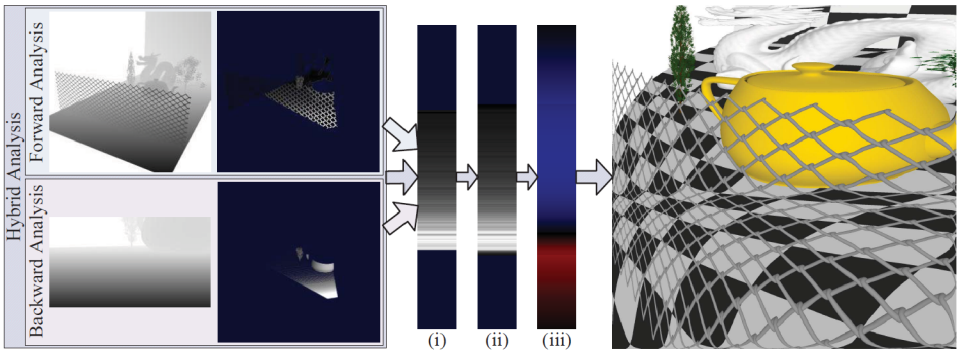


Figure 3.11: From left to right, steps of Rectilinear Texture Warping algorithm [34].

In the Step 2, the importance map is further processed in order to get the warping map. Firstly (in Step 3), the maximal importance value from every row and every column is stored into two 1D importance maps. These maps are blurred in Step 4 in order to smooth the differences between adjacent samples and ensure coherency. Finally (in Step 5), the positions are shifted according to value in the 1D importance map. The computed offsets are then used to build the warping maps.

The Steps 6 and 7 are well known steps from the standard Shadow Mapping algorithm. Step 6 renders the shadow map and Step 7 computes shadows. However, in both of these steps, the newly built warping map is used for rendering of the shadow map as well as computation of shadow map coordinates when the shadow is computed.

As mentioned above, multiple importance functions can be used for the analysis. Rosen implemented the following functions:

- **Desired view (DV)** function evaluates only those pixels that are visible in current camera view frustum. It is not needed to compute shadows outside the frustum.
- **Distance to eye** function measures distance of a point to the camera and focuses on the points that are closer since they need more detail.
- **Shadow edge** function ensures that higher sampling is used only on shadow boundaries where the aliasing error is the most noticeable.
- **Surface normal** function gives increased importance to objects facing the camera.

The biggest disadvantage of the RTW algorithm is the rectilinear grid. In order to maintain high quality shadows, the algorithm selects the maximal importance value from the importance map for a given row and column, respectively. This may introduce unneeded resolution for the remaining part of the row or column. In the worst case, it may lead to decrease in quality of the output. Another disadvantage that is common for all warping approaches is that the scene has to be finely tessellated. The warping of the shadow map curves the long edges of triangles (see Section 2.3.3). This artifact is not visible when the triangles are reasonably small.

Contemporary GPUs supports hardware tessellation. It slightly increases processing time, but also improves quality of the rendered image.

Jia et al. [12] introduced *Distorted Shadow Mapping* (DSM) algorithm that detects shadow silhouettes from depth discontinuities in the standard shadow map. It does not employ any regular grid, and increases the sampling rate locally. However, the DSM algorithm does not consider any other view information, and it relies only on information from the shadow map. Hence, some important details might be missing.

3.3 Omnidirectional Shadow Mapping

Section 2.3.2 discussed how shadows rendering for the whole environment with traditional projection transformations becomes problematic. Therefore, in case of omnidirectional light sources, alternative approaches must be used.

3.3.1 Cube Shadow Maps

In order to create shadow maps for an omnidirectional light source, the Cube Shadow Maps algorithm proposes to point the virtual camera into six directions. The view direction of the virtual camera should be oriented along directions defined by the axes of the local coordinate system of the cube: positive X, negative X, positive Y, negative Y, positive Z and negative Z. This is almost identical to the way how a cube map for environment mapping is generated except that in this case depth values are stored instead of color.

Basics of the Cube Shadow Maps

The faces of the cube represent shadow maps and directions of the faces shows the particular direction for the virtual camera (see Figure 3.12). In order to cover the whole environment, the traditional Shadow Mapping algorithm exploits cube maps to visualize shadows cast from point lights. To fill the data in the cube shadow map, six render passes have to be performed. The GPUs generally support the cube shadow maps which are thus easy to implement.

The biggest disadvantage of the Cube Shadow Maps is that six render passes are often too expensive. This fact can cause rapid decrease of performance for complex scenes with high number of polygons. Even if per-object frustum culling is applied, rendering of shadows is still very expensive in comparison to rendering of the rest of the scene.

Efficient Frustum Culling

King and Newhall [15] introduced method for reducing the number of passes. If the light source is outside the view frustum, then rendering of at least one face of the cube shadow map can be skipped. This leads to a significant effect on the performance.

The following technique for efficient cube face frustum culling (EFC) can be used. The camera view frustum and each cube face frustum are tested for their mutual intersection. Those frusta that do not intersect can be discarded for further rendering because they do not contribute to the final image. The efficient culling of

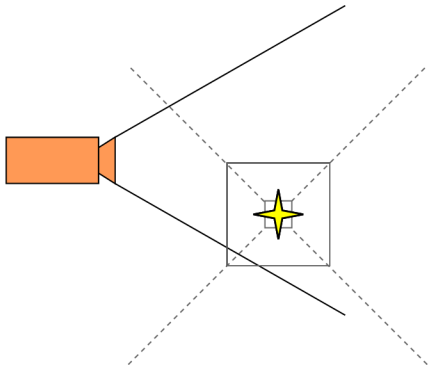


Figure 3.12: Illustration of the Cube Shadow Maps technique. Each face of the cube stores depth values for a certain part of the scene.

arbitrary frustum F against the camera view frustum V works as follows. The frusta are defined by 8 boundary points and 12 boundary edges. To determine whether the two frusta intersect, two symmetric tests have to be performed. Firstly, it should be tested whether a boundary point of one frustum lies inside other frustum (see Figure 3.13a). Secondly, it should be tested whether a boundary edge of one frustum intersects one or more clip planes of other frustum (see Figure 3.13b) [15].

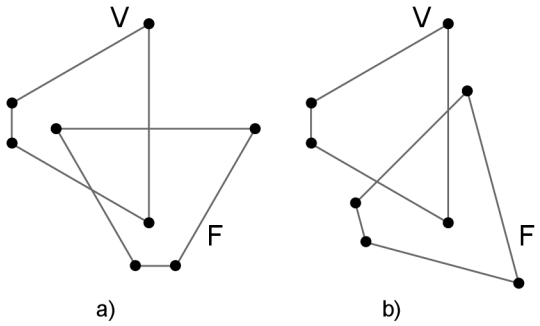


Figure 3.13: A frustum consists of boundary points and boundary edges. Two frusta intersect when (a) at least one boundary point of the frustum F lies inside other the frustum V or (b) at least one boundary edge of the frustum F intersects a face of the frustum V.

For each face of the cube shadow map, it is investigated whether the camera view frustum intersects the cube face frustum and vice versa. If it is not the case, the cube face frustum does not contribute to the scene and it can be omitted from the further processing (see Figure 3.14). It is also necessary to take into account shadow casters outside the view frustum. If the view frustum culling is applied on the shadow caster, the projected shadow that is visible in the view frustum may disappear. On the other hand, culling the shadow caster against the cube face frustum causes that the shadows are rendered outside the camera view frustum. King and Newhall suggested to use frustum-frustum intersection test described above for the shadow casters as well. Since point light sources are used, rays are emitted from a single point towards all shadow casters. This is analogous to the perspective pro-

jections. If the shadow casters are enclosed by bounding objects, frusta representing the projected shadows can be created and then the frustum-frustum test can be applied in this case as well. These tests are performed once per frame.

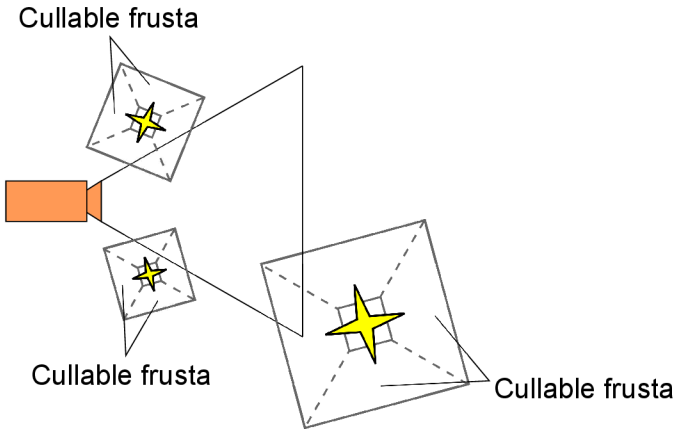


Figure 3.14: If the light source lies outside the camera view frustum, at least one face can be culled.

3.3.2 Dual-Paraboloid Shadow Maps

The following text discusses an alternative approach for rendering shadows cast from omnidirectional light sources. The *Dual–Paraboloid Shadow Mapping* algorithm (DPSM) [3] maps 3D positions of a geometry into 2D map. The Dual-Paraboloid mapping can be used for creating maps of an environment and among other environment mapping approaches, such as cubical or spherical, the algorithm introduces better performance in comparison to the cubical mapping, and better quality in comparison to the spherical mapping. The algorithm is based on two paraboloids attached back-to-back, each capturing one hemisphere. This section introduces principles of the Dual–Paraboloid Shadow Mapping algorithm, and how it can be used for rendering shadows.

Mathematical Background

In principle, the idea is based on a mirror. Imagine a totally reflective mirror in a shape of a paraboloid that reflects incident rays from a single hemisphere into the direction of the paraboloid (see Figure 3.15). The rays may carry some information about the environment (such as color or distance) and the information can be stored into a rectangular map. The 2D coordinates are computed from the point on the paraboloid surface where the ray intersects the paraboloid.

To implement the Dual–Paraboloid Shadow Mapping algorithm on GPU, it is necessary to understand how the mapping actually works. This knowledge will be then used for writing shaders for GPUs. The paraboloid itself is given by:

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2), \quad x^2 + y^2 \leq 1 \tag{3.6}$$

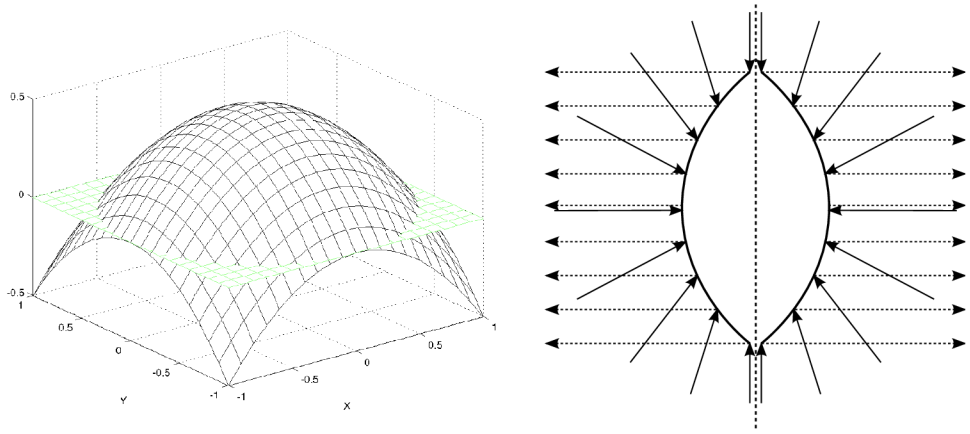


Figure 3.15: (Left) The paraboloid itself. (Right) Two paraboloids attached back-to-back can capture the environment from all directions [3].

The key concept of the paraboloid mapping is that all incident rays are reflected in the same direction. The first task is to find the point on a paraboloid surface where the ray is reflected. This can be computed using the surface normal vector.

A paraboloid surface point P is given by:

$$P = (x, y, f(x, y)) \tag{3.7}$$

To compute the normal vector in P, the tangent vectors have to be computed by taking the partial derivatives of the function with respect to x and y. The resulted cross product gives the normal vector:

$$T_x = \frac{\delta P}{\delta x} = \left(1, 0, \frac{\delta f(x, y)}{\delta x}\right) = (1, 0, -x) \tag{3.8}$$

$$T_y = \frac{\delta P}{\delta y} = \left(0, 1, \frac{\delta f(x, y)}{\delta y}\right) = (0, 1, -y) \tag{3.9}$$

$$N_P = T_x \times T_y = (x, y, 1) \tag{3.10}$$

The derived normal vector for the point P is now known for every point on the paraboloid surface and it expresses the x and y coordinate of the map.

Based on the information mentioned above, the mapping can now be defined. The normal vector for the entire paraboloid surface can be computed as a sum of the incident ray and the reflected ray. As mentioned above, the reflected ray is always going to be $(0, 0, 1)$ for the front paraboloid and $(0, 0, -1)$ for the back paraboloid, respectively. This is a crucial concept that it is the same for a given hemisphere (see Figure 3.15). The normal vector can be computed as:

$$N_P \Leftrightarrow V_{incident} + V_{reflected} \tag{3.11}$$

Based on the Eq. 3.10, the previous equation can be expressed as:

$$N_P = (x, y, 1) \Leftrightarrow V_{incident} + V_{reflected} = V_{sum} \tag{3.12}$$

The final step for getting the x and y coordinates is to divide all components of V_sum by its z part:

$$N_P = \frac{1}{z_{sum}} (x_{sum}, y_{sum}, z_{sum}) = \left(\frac{x_{sum}}{z_{sum}}, \frac{y_{sum}}{z_{sum}}, 1 \right) \tag{3.13}$$

As both x and y coordinates of the paraboloid surface is now computed, they express the point on the surface from which the incident ray is reflected. Also, they express the coordinates to the map where the information from the environment is going to be stored. The following text presents how this concept can be applied to rendering shadows using the Shadow Mapping algorithm.

Depth Texture Generation and Shadow Rendering

When rendering shadows cast from an omnidirectional light source, the Shadow Mapping algorithm requires to render the shadow map for the entire scene. Based on the concept of the Dual-Paraboloid mapping, it needs only two render passes to capture the whole environment.

The omnidirectional shadow rendering works in the same way as the traditional Shadow Mapping algorithm (see Section 2.3). The virtual camera is placed in the position of a light source. According to the position and the orientation of the light source, the appropriate model-view-projection matrix has to be found. In this case, the projection matrix can be identity, because the projection is performed by the paraboloid mapping. The model-view matrix provides information on where is the scene divided into two hemispheres and it also expresses the direction of the paraboloid.

The vertex shader on GPU parametrizes only the geometry vertices. The remaining part of the rendering process is unchanged. It means, that the paraboloid mapping is applied only in the vertex shader and rasterization of polygons are performed in the traditional way. The vertex shader can be written as:

```
1 vec4 vertexEyeSpace = in_ModelViewMatrix * vec4(in_Vertex,1.0);
2 vertexEyeSpace.xyz = normalize( vertexEyeSpace.xyz );
3 vertexEyeSpace.z += 1.0;
4 vertexEyeSpace.xy /= vertexEyeSpace.z;
```

The input geometry is transformed to the light space using model-view matrix. The resulting vector is normalized and it will serves as the incident ray for the paraboloid mapping. The next step is to sum the incident ray with the reflection vector which is $(0, 0, 1)$ (Line 3). Finally, the result is divided by the z part in order to derive the x and y coordinates. To process the vertex further in the pipeline, the z and w coordinates have to be set as well:

```
1 vertexEyeSpace.z = (Length - near)/(far - near); vertexEyeSpace.w = 1.0;
```

The depth value from the z coordinate is stored in the shadow map in a fragment shader. The values from the shadow map will be used in the next step where the shadow is computed.

In the final render pass of the Shadow Mapping algorithm, the depth values are read from the shadow map and compared with the depth of the current fragment. The shadow computation is now performed in the fragment shader. However, the steps for computing coordinates to the shadow map are very similar with the first rendering pass. The same concept of the paraboloid mapping is used as well:

1. Find a vector from the light source to the desired object.
2. Use this vector to calculate s and t coordinates (one pair for each hemisphere).
3. Sample both paraboloid maps with the coordinates.
4. Process the sampled values.

The implementation of all steps in the fragment shader is:

```
1 texCoords.xyz = normalize( texCoords.xyz );
2 texCoords.z += 1.0; texCoords.x /= texCoords.z; texCoords.y /=
  texCoords.z;
3 texCoords.z = (Length - near)/(far - near); texCoords.w = 1.0;
4 return vec3( 0.5*texCoords.xy + 0.5, texCoords.z);
```

The resulting x and y coordinates are normalized in order to sample the texture in range [0..1]. The z coordinates holds the depth of the rendered fragment.

After this step, all the necessary pieces of information are derived for computing shadows using the Shadow Mapping algorithm. The texture coordinates are derived using the paraboloid mapping, and the depth value that is going to be compared with the value stored in the shadow map is also computed.

The Dual-Paraboloid Shadow Mapping minimizes the amount of used memory and the number of render passes that are necessary to cover the whole environment in comparison to . the Cube Shadow Maps technique. Other parameterization can certainly be found but the proposed parabolic parameterization maintains its simplicity and performance, e.g. in GPU implementation [29].

Nevertheless, the DPSM algorithm has also some disadvantages. While in the Cube Shadow Map approach all the transformations needed to create the shadow map are linear, they do not need any extra treatment on GPUs. This mainly concerns the interpolation process between vertex and fragment shader (see Section 2.3.3). When using the DPSM algorithm, the rendered scene needs to be finely tessellated, because the mapping is not linear and it does not work well for large polygons. Unfortunately, it may introduce new bottlenecks and artifacts on the connected parts of front and back paraboloids.

3.3.3 Comparison of Cube Shadow Maps and DPSM

The Cube Shadow Mapping as well as the Dual-Paraboloid Shadow Mapping algorithms are capable of rendering shadows cast from omnidirectional light sources. Since both approaches have some advantages over another, it is worth comparing their properties in detail [28].

Frame Time in Walkthrough

The first measurement shows dependence of the frame time as the camera walk through the scene (see Figure 3.16). The unoptimized variants of the Cube Shadow Mapping algorithm (Cube6) and the Dual–Paraboloid Shadow Mapping (Dual-Paraboloid) show the worst results. In both approaches as all the geometry is rendered in every pass. Naturally, the Cube Shadow Mapping algorithm performed the highest frame time because of the six render passes.

The basic optimization technique introduced the bounding object frustum culling against the camera view frustum, the cube face frustum (Cube6 Optim) and the clipping plane between paraboloids (Dual-Paraboloid Optim). In this case, the same amount of geometry is rendered in both approaches. The overhead for increased number of the render passes for the Cube Shadow Mapping algorithm had no effect on an overall time for a single frame and thus the resulting frame times are similar.

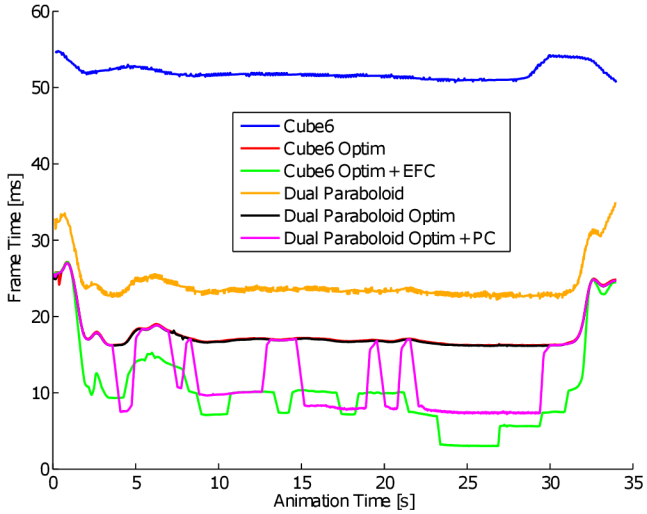


Figure 3.16: Frame times for the walk-through of the scene for all implemented methods.

The Cube Shadow Mapping approach exhibits the best result with the effective cube face frustum culling (EFC) presented in Section 3.3.1. Figure 3.16 shows that the DPSM increased the performance only by skipping one paraboloid wherever appropriate using plane clipping (PC). Otherwise, all of the geometry would have to be rendered in two passes. The Cube Shadow Mapping approach can skip up to five render passes and thus it achieved the best results (e.g. in 25^{th} second of the walk-through). The frame time in the DPSM depends mainly on the amount of rendered geometry and also on the amount of geometry in the given hemisphere. As it can be seen in Figure 3.16, the DPSM saved only 50% of the computation time when it rendered the scene only for one side. However, the Cube Shadow Mapping approach saved up to 83% of the performance. Furthermore, Figure 3.17 shows that the DPSM uses only one paraboloid most of the time and also that the Cube Shadow Mapping approach rarely performed all six passes. This happens when the light source lied outside the camera view frustum.

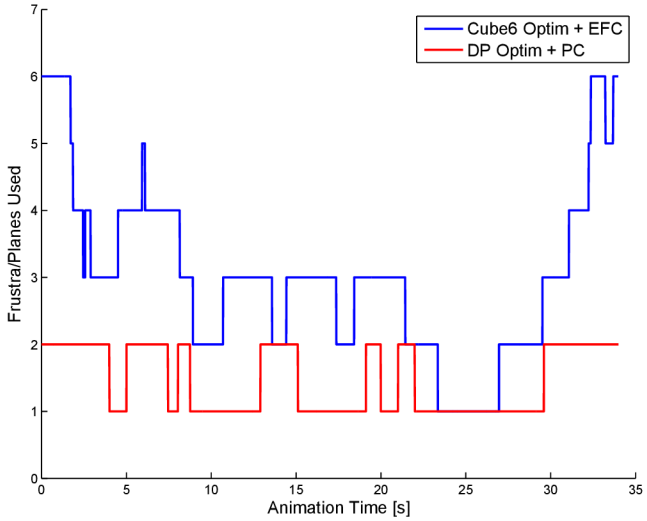


Figure 3.17: The plot shows the number of processed cube faces (blue) and the number of rendered paraboloid sides (red).

Timings of Render Passes

Since the Shadow Mapping algorithm renders shadows in two passes, the benchmark application measured frame times of individual passes for all the implemented methods. The time for final shadow rendering turned out to be equivalent for all methods because it mainly depends on number of rendered polygons. Here, the view frustum culling was employed. The most noticeable differences were in times for rendering the shadow maps.

Figure 3.18 shows that the methods without any optimization had to render all the geometry six times in case of the Cube Shadow Mapping approach (blue), or two times in case of the DPSM algorithm (red), respectively. There are also some differences between methods where the frustum and plane culling is applied. The DPSM algorithm was faster in comparison to the Cube Shadow Mapping approach. An overall amount of rendered geometry was equivalent in both cases so there seems to be some additional overhead in the Cube Shadow Mapping technique.

Generally, the DPSM algorithm was faster when only one paraboloid was processed. The Cube Shadow Mapping technique reached the similar times when only 2 faces were processed. The plot in Figure 3.18 also shows that in 25^{th} second, the Cube Shadow Mapping technique achieved the best results. In this case, only one face was processed which was caused by the position of the light sources relative to the camera (see Figure 3.19).

Effect of Shadow Map Resolution

It was also investigated how the shadow map resolution affects the frame rate. In Table 3.1 and Table 3.2 the results for various shadow map sizes are presented. As it can be seen, the optimization techniques caused the increased frame rate.

Considering shadow map as a texture storing single 32-bit value per texel, memory consumption of the Cube Shadow Mapping approach was from 24MB

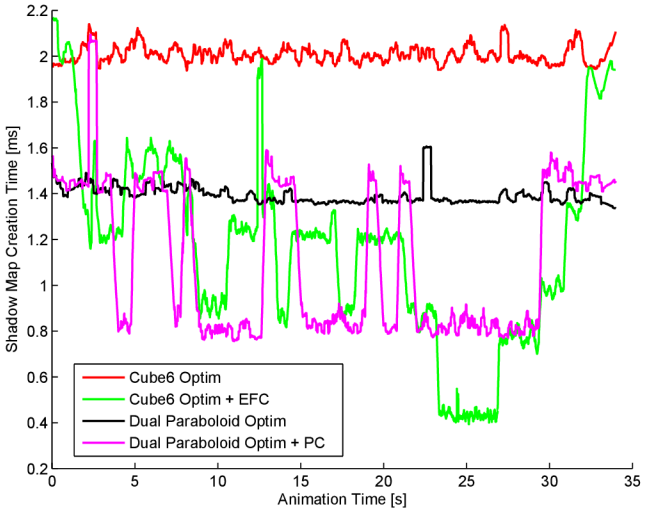


Figure 3.18: Evaluation of the times which all methods spent on the shadow map generation. For better illustration, unoptimized methods are not visible, because they had very poor results in comparison to optimized techniques.

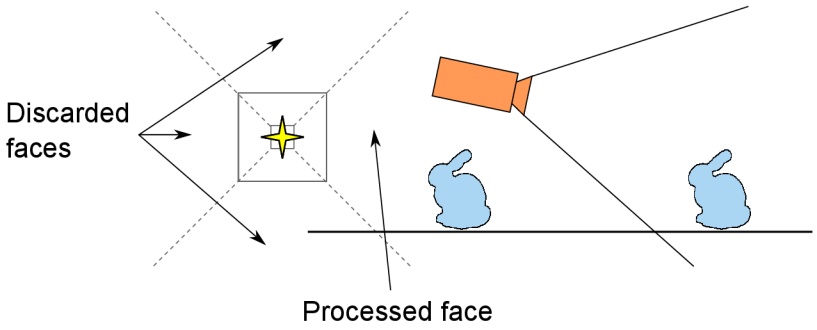


Figure 3.19: An illustration of the situation when only one face is processed during shadow map generation pass. Figure shows that only one cube face frustum intersects with the camera view frustum.

(1024 × 1024) to 384MB (4096 × 4096). Whereas the more computationally intensive Dual–Paraboloid Shadow Mapping approach used one third of memory in comparison to the Cube Shadow Mapping approach (8MB to 128MB). By utilizing efficient frustum culling methods, the computation time can be saved by reducing number of the render passes and size of the geometry data, which also reduced memory utilization (less number of values stored due to frustum culling).

When taking 1024^2 resolution of shadow map as 100% performance for each method, switching to 2048^2 caused performance drop off only by 6.54% in average, but greatly increased shadow quality. Choosing 4096^2 resolution for shadow map took 25.76% performance penalty in average.

The resulting image quality of the Dual–Paraboloid Shadow Mapping technique depends on the geometry of the occluding object. As described in [3, 29], the Dual–Paraboloid mapping causes low-polygonal casters to produce incorrect shadows. Increasing shadow map resolution improves shadow quality but still can

	1024^2	2048^2	4096^2
Cube6	75.71	70.04	47.9
Cube6Optim	150.43	116.76	64.04
Cube6Opt+EFC	188.71	151.67	89.68
DP	167.95	146.62	97.52
DPOptim	207.24	178.67	109.4
DPOptim+PC	208.15	180.24	110.95

Table 3.1: FPS of low-polygonal scene (600K vertices)

	1024^2	2048^2	4096^2
Cube6	19.11	18.38	16.21
Cube6Optim	57.15	51.23	36.50
Cube6Opt+EFC	127.47	114.21	83.38
DP	41.50	39.74	33.17
DPOptim	57.47	54.32	42.85
DPOptim+PC	90.56	86.08	69.58

Table 3.2: FPS of high-polygonal scene (3M vertices)

not match the quality of details achieved by the Cube Shadow Maps approach (see Figure 3.20).

Position of a Light Source Relative to Geometry

The last experiment was focused on position of the light source relative to the geometry. This experiment was inspired by techniques for computation of interactive global illumination [33].

In this case, Virtual Point Lights (VPLs) are generated on the surface to approximate indirect lighting. The reflected light is scattered into all directions. Therefore, some method is required to handle shadows from the reflected light. For this purpose, all of the the geometry data is positioned into one hemisphere relative to the light source. When the geometry is distributed around the light sources, it is useful to use the Cube Shadow Maps technique, because this optimization strategy is better and it can easily manage the number of processed cube map faces. However, when only one hemisphere is needed to render, the DPSM algorithm is more suitable.

Times for rendering of the shadow map was measured in both of the presented techniques. Ritschel et al. [33] employed the Dual–Paraboloid mapping algorithm in their approach. They generated shadow maps for multiple VPLs (256 and more) from simplified geometry.

In Figure 3.21, it can be seen that the DPSM algorithm is approximately two times faster than the Cube Shadow Mapping approach. The results are similar for various levels of the scene complexity. The Dual–Paraboloid mapping algorithm can be used despite its worse accuracy, because indirect lighting produces low-frequency shadows. In this case, the artifacts are blurred.

The experiments showed advantages and disadvantages of both approaches in various context. It also showed that there is no proof which approach is better. Both can be successfully used in various applications of computer graphics.

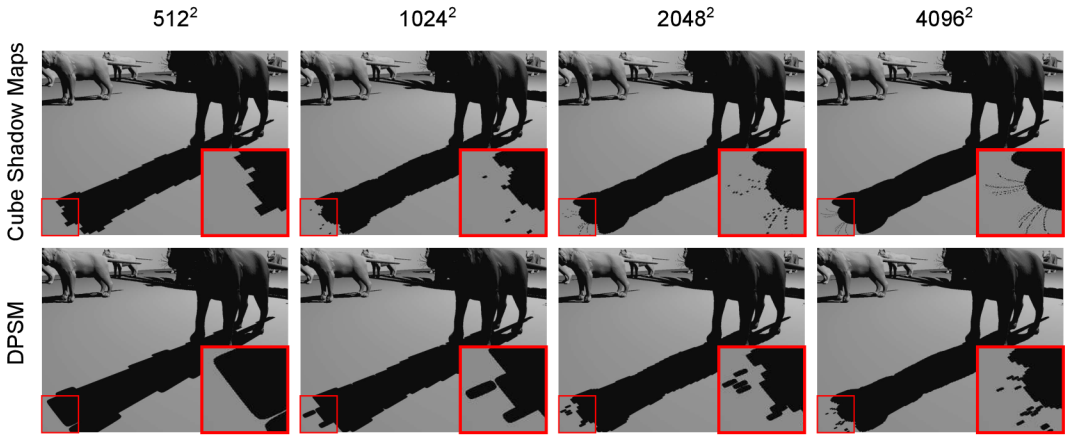


Figure 3.20: Figure shows how the shadow map resolution influences the shadow quality. Since a single paraboloid covers one hemisphere, one shadow map texel is projected on the large area in the scene (in comparison to the Cube Shadow Maps). This leads to worse quality of shadows.

3.3.4 Interactive Global Illumination

Section 2.2.3 presented the idea of Instant Radiosity that is based on generation of Virtual Point Lights for approximation of indirect illumination. This section discusses how the algorithms for omnidirectional shadow rendering can be employed in interactive applications to deal with indirect illumination.

Ritschel et al. presented a low-quality shadow maps (*Imperfect shadow maps*, ISMs) [33] that can be evaluated hundreds per frame. The approach follows observations that the direct illumination needs an accurate visibility test for correct results, however, the indirect illumination can use an approximate visibility queries. It was shown that inaccurate visibility has a minor impact on the result, but leads to significant performance gains because indirect illumination is created by smooth gradients that could mask some errors caused by incorrect visibility testing.

The technique is based on Instant Radiosity [14] and every VPL generates the Imperfect Shadow Map using Dual-Paraboloid Shadow Mapping algorithm. Due to low-frequency nature of indirect illumination, it is not necessary to render the shadow map for the entire scene, but only for a coarse point-based representation (see Figure 3.22). It was shown that it works well for large and fully dynamic scenes and it also enables changes of light, geometry, and material without decreasing of frame rate. In the next step, Pull-push algorithm is used to fill the holes in the shadow map. The resulted Imperfect Shadow Map is not accurate and some depth values are incorrect, but it suffices for a plausible indirect illumination rendering. Since the shadow maps are stored in a large 4096×4096 texture, the pull-push is done in parallel for all shadow maps and it takes only a few milliseconds.

The ISMs approach is very fast for computing global illumination. It could be used for direct and indirect illumination, but it has also some limitations. Due to simplified geometry, it has problem with glossy reflections. It is also not directly scalable to very large scene because of the point-based representation. The points need to be generated reasonably and this is problem in the large scenes. The approach

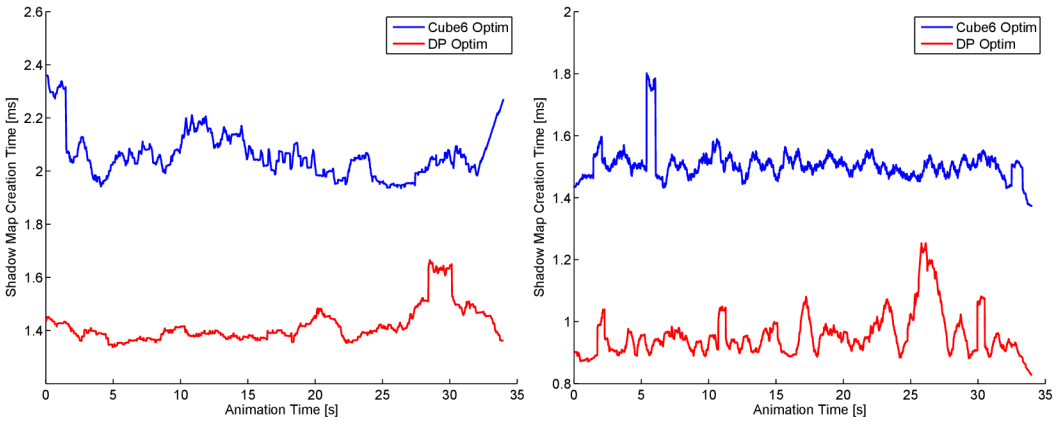


Figure 3.21: Figure illustrates times that the methods of interest spent on generation of the shadow map. In this case, the geometry is placed into one direction from the light source. The scene was represented by points only: 3 millions points (left) and 100k points (right).

has several parameters that need to be chosen, e.g. number of shadow maps, level of pull-push algorithm, etc. which should be adjusted according to scene complexity.

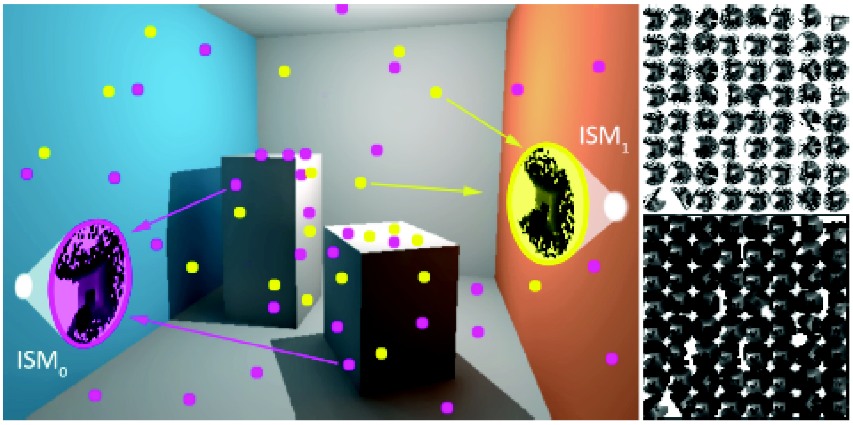


Figure 3.22: Illustration of ISMs concept. Simplified shadow maps (right,top) and interpolated after pull-push(right,bottom) [33]

CHAPTER 4

Improved Texture Warping for Complex Light Sources

The Cube Shadow Maps and Dual-Paraboloid Shadow Mapping are methods that are capable of rendering shadows cast into all directions. Since they both are based on the Shadow Mapping algorithm, they suffer from issues caused by the limited resolution of the shadow map represented by a raster image. The main issue is related to the quality of shadows which is usually decreased by aliasing. The Parallel-Split Shadow Maps is a technique that can reduce aliasing in outdoor scenes and large environments. It is optimized for directional light sources and spotlights. These types of light sources are the most common in outdoor environment.

This chapter introduces a novel technique for improving quality of shadows cast by omnidirectional light sources. It shows how to improve process of shadow map rendering in order to get a better sampling distribution of the scene. It utilizes non-orthogonal warping scheme and it is applicable also for complex light sources.

The core of the thesis can be expressed by the following statement: *Parameterization of shadow map coordinates based on simple scene analysis can reduce aliasing error of the shadows cast by complex light sources.*

Section 3.1 shows that the highest aliasing error can be observed close to the near plane of the camera view frustum. For some scenarios, for instance outdoor scenes lit by the sunlight, the aliasing error can be successfully reduced with the PSSM algorithm (see Section 3.2.2). However, PSSMs do not address the shadow quality for omnidirectional light sources. The shadow quality for this type of light sources is discussed in the thesis. They present one of the three types of light sources that can be usually seen in indoor scenes.

Section 3.3 explains how difficult is to compute shadows for omnidirectional light source. The thesis shows how to improve the quality of shadows regardless of the mutual position of the light source and the camera. The improvements are implemented in both Cube Shadow Maps and Dual-Paraboloid Shadow Mapping algorithm. Moreover, omnidirectional light sources are successfully employed not only for direct illumination, but also as virtual point lights for computing of indirect illumination (see Section 2.2.3).

An example of a critical scenario is when the light source is inside the camera view frustum. The scenario introduces two main challenges. Firstly, shadows have to be cast into all directions. Secondly, the aliasing error is not distributed uniformly but it depends on mutual position of the light source and the camera, and the current scene configuration. The uniform distribution of the aliasing error is observed from the light source point of view when the light source is outside the frustum. This applies to all types of light sources. When the light source is inside the frustum,

the alias error changes unevenly. The approach presented in the thesis handles both of the challenges.

Section 4.1 shows how successive improvements of the Dual-Paraboloid Shadow Mapping algorithm solves the use case described above. Firstly, the solution for the simple case when the light source is outside the camera view frustum is introduced. Then, the scene sampling is refined by exploiting variable sampling ability of the paraboloid. Finally, Section 4.2 introduces a general solution for the case when the light source is inside the camera view frustum.

4.1 Improved Paraboloid Mapping

First of the sampling improvement is based on importance-driven cut of the paraboloid and finding optimal paraboloid orientation. The algorithm adjusts the sampling in order to handle in the best way the important areas detected in the camera view frustum. It is based on the fact that it is not necessary to sample the whole environment and some parts of the scene can be completely excluded from sampling into the shadow map (see Figure 4.1). Then, one of the two paraboloids is not required when the light source lies outside the camera view frustum. In this case, the single paraboloid is sufficient to cover the whole frustum with a single shadow map. Firstly, the basic idea of the algorithm is described. Then, secondly, the steps of the algorithm are explained in detail in the following sections.

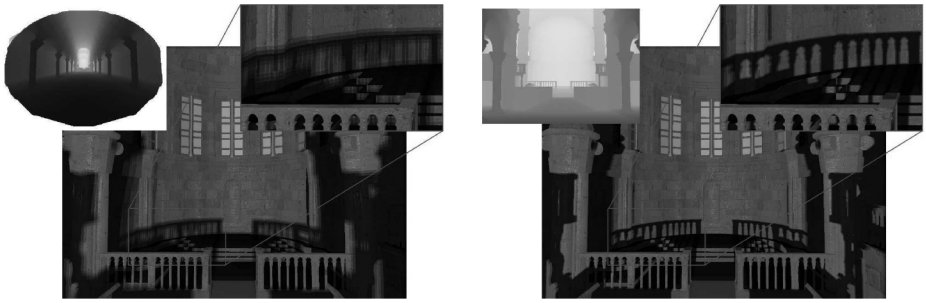


Figure 4.1: Large indoor scene with hemispherical light source and 1024×1024 paraboloid shadow map (left). Same scene with improved paraboloid map of the same size (right)

The original Dual-Paraboloid Shadow Mapping algorithm uses two paraboloids - front and back - in order to capture the environment around the light source from both directions. Each paraboloid samples the scene uniformly and the rotation of the paraboloids remains fixed. The shadow map rendering is performed by calculating the paraboloid projection and such projection can be implemented in a vertex shader [3] (see Section 3.3).

In order to achieve better sampling of the scene, it must be ensured that rendering from the light source point of view with parabolic projection covers the objects lying inside the camera view frustum. It is not necessary to sample the scene parts with objects which receive shadows outside the camera view frustum. Therefore, the algorithm consists of four main steps:

1. Locate clipping planes for the camera view frustum to mark the boundaries

where all potential shadow receivers are present, forming a *truncated view frustum*.

2. Determine an optimal field of view for the scene rendered from the light source point of view to cover the whole truncated view frustum.

3. Find the directional vector which sets the rotation of the paraboloid.

4. Perform a suitable cut on the paraboloid to constrain shadow map rendering only to the selected part of the paraboloid.

These steps generate *Improved Paraboloid Shadow Maps* (IPSM) [37]. Since the optimal orientation of the front paraboloid covers the visible part of the camera view frustum, the back paraboloid is needed only when the light source lies inside the view frustum. In such cases, shadows must cover the entire environment and the parameterization converges to the standard DPSM. Otherwise, one rendering pass can be saved to speed up rendering.

4.1.1 Determining Optimal Coverage

In the 3D space, an area illuminated by a point light source with certain field-of-view and direction has the shape of a cone, as seen in Figure 4.2 (right). Let us call it as a *light cone*. In order to optimally cover the view frustum with the light cone, it is needed to locate the boundaries around the visible objects in the view frustum first, so that shadow sampling will be performed only within its boundaries. They will be referred as *minimal* and *maximal depth clipping planes* respectively. In order to determine the boundaries, the z-values of all transformed and clipped vertices are computed in the eye space to obtain minimum and maximum distances to the camera. In the next text, the view frustum with minimum/maximum depth boundaries is called a *truncated view frustum* (see Figure 4.2, left).

Location of optimal coverage and field-of-view can be done by the following method. Prior to starting the calculation, a light position L is obtained and positions of eight *frustum border points* (FBPs) on the minimal (N_{1-4}) and maximal (F_{1-4}) depth clipping planes of truncated view frustum, as seen in Figure 4.2 (left). The algorithm computes the optimal field-of-view F_v and direction vector for the light cone C_d.

Firstly, the normalized sum of the vectors D_j from the light source to the FBPs is computed (Eq. 4.1). This computation expresses the average direction $\bar{C_d}$ from the light source L to the truncated view frustum.

$$\bar{C_d} = norm(\sum_j D_j) \tag{4.1}$$

In the next step, iteration through all vectors D_j is performed in order to obtain the maximal angle from FBPs to an average direction $\bar{C_d}$ (Eq. 4.2).

$$F_v = max(D_j \cdot \bar{C_d}), 1 \leq j \leq 8 \tag{4.2}$$

This maximal angle defines the field-of-view F_v of the light cone. Since all frustum border points are contained in the light cone, this implies that the whole truncated view frustum will also be covered by the light cone (Figure 4.2, right).

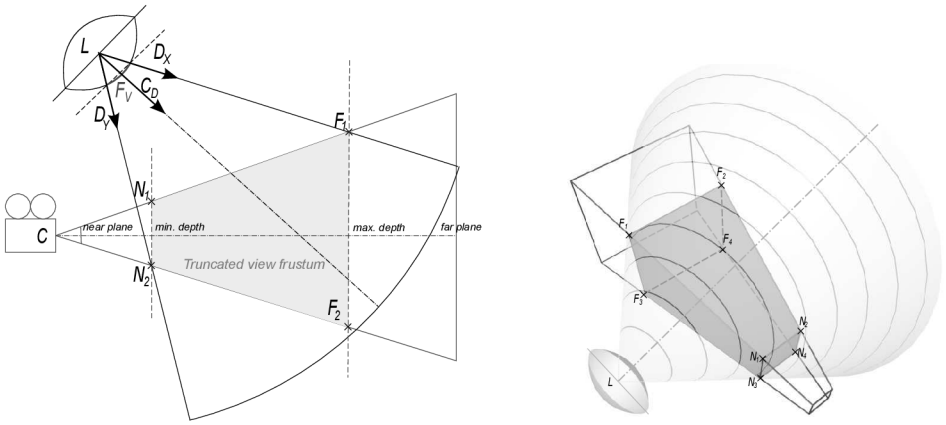


Figure 4.2: Illustration of optimal coverage of the truncated view frustum with the light cone.

This evaluation is performed only once per frame, so it does not have a crucial impact on the performance. It should be noted that the proposed method is the numerical solution and thus it is not optimal.

4.1.2 Paraboloid Cutting and Rotation

After obtaining the field of view F_v and the average direction vector $\bar{C}_d$ using the steps above, the light cone can be defined to cover the truncated view frustum. Since the single light paraboloid has a 180° field-of-view, the cutting scheme that is used for smaller angles can be introduce.

The parameterization for DPSM (see Section 3.3) is exploited to find the zoom factor. An auxiliary vector is defined whose direction is derived from size of the field-of-view F_v. Then, the vector is processed in the same manner as the transformed vertices ($vertexEyeSpace$) in the vertex shader. This processing could be done in 2D because of the symmetry of the paraboloid. The resulted x-axis coordinate expresses the zoom factor Z which is applied to the computed coordinates in the vertex shader: $vertexEyeSpace.xy* = 1/Z$. This operation causes the paraboloid to be cut at the point $(Z, f(x))$ which precisely creates the desired field-of-view F_v (see Figure 4.3), but the resolution of the shadow map texture in the output remains the same.

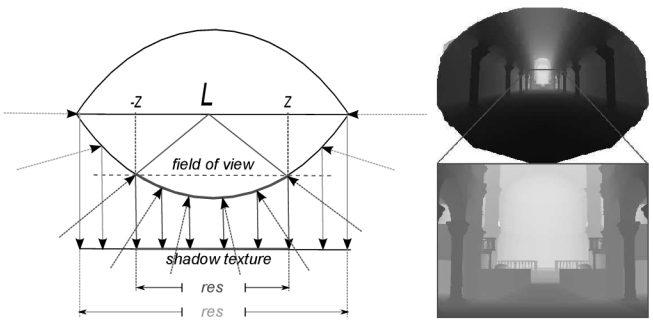


Figure 4.3: Paraboloid cut to constrain shadow map sampling to certain parts of the paraboloid.

4.1.3 Implementation Details

Shaders from the original DPSM can be left almost intact and the solution can be easily implemented into any graphics engine using deferred shading techniques without major changes against the original DPSM.

In order to compute minimal/maximal depth clipping planes, z-values of the transformed vertices are needed. Normal vectors of the surface and depth values are rendered into a floating point render target with the attached texture. This process is common in graphics engines using deferred shading and it can be easily added into the scene rendering pipeline.

This texture is scaled down on the GPU to a small size (e.g. 32×32) so as to reduce data transfer (per-pixel accuracy is not required). The data is then transferred from GPU to CPU. The stored buffer is analyzed and minimum, maximum and average depth values are searched for. Because of the scaling on the GPU and the transferring of only a small block of data to CPU, it has no significant effect on performance.

4.1.4 Skewed Paraboloid Cut for Better Shadow Rendering

The method described below focuses on improvement quality of shadows in areas in front of the camera. The method extends the original Dual-Paraboloid Shadow Mapping (DPSM) algorithm and introduces modified parameterization which increases the density of sampling in some parts of the shadow map. The densely sampled areas should cover mainly the parts of the scene in front of the camera where high visual quality is most important [27]. The approach deals with a case when the aliasing error is distributed evenly through the camera view frustum.

The proposed technique modifies of the original DPSM parameterization using the rotation transformation that refines the density of sampling for some directions $\vec{v}$ (see Figure 4.4). The transformation can be defined by matrix M_c and the new parameterization is expressed as:

$$\vec{h_c} = \vec{d} + M_c \cdot \vec{v} = k \cdot \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \tag{4.3}$$

where $\vec{d}$is direction of the paraboloid, $\vec{v}$ is the incident ray and k is the scaling factor. Equation 4.3 expresses parameterization of one hemisphere using 2D coordinates $(x_c; y_c)$.

In the original DPSM algorithm, the directions of the incident rays fall into the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ relatively to the direction of paraboloid $\vec{d}$. The parameterization generates 2D coordinates in a range $[-1, 1]$ which serve as indices to the texture. Further, the xy-plane divides the scene into two hemispheres. Let us call it a *separation plane*.

The rotation matrix M_c rotates all incident rays $\vec{v}$ so that they are reflected from different points on the paraboloid. The range of the reflection points coordinates is defined by an intersection of the paraboloid and the separation plane (Figure 4.5, left). Orientation of the separation plane determines the range of the directions relatively to the direction of the paraboloid (Figure 4.5, right top). Since the paraboloid has to capture the rays from the same hemisphere and the directions of the rays have

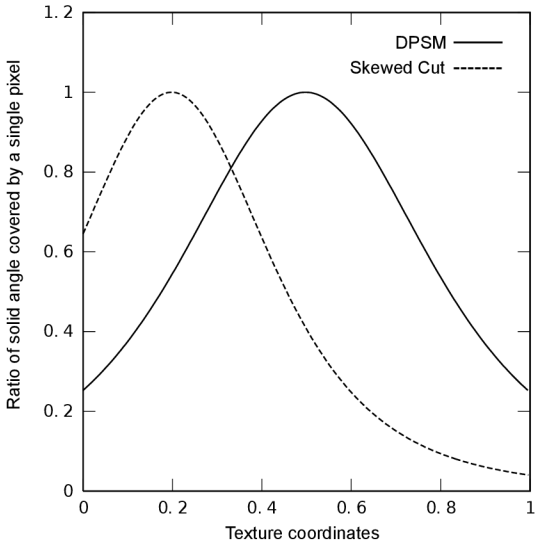


Figure 4.4: the change of solid angle covered by a single pixel versus the angle between the viewing direction over the whole shadow map. Pixels near the edge of the shadow map cover approximately 1/4 of the solid angle covered by center pixels. Using the skewed cut, the sampling density can be further increased on one side of the map.

changed relatively to the direction of the paraboloid, the direction of the paraboloid has to rotate from its initial position as illustrated in Figure 4.5 (right bottom).

By rotating the separation plane by some angle, the *skewed cut* of the paraboloid is performed. It is shown in Figure 4.6 how the shadow map changes when the skewed cut is applied.

The skewed cut is parameterized by the orientation of the separation plane defined by two rotation angles in x and y axes - *cut angles*. Using the cut angles, the rotation matrices can be derived separately for every axis. The matrices are used later in the rendering step. They control directions of the incident rays for the given paraboloid.

The intersection of the paraboloid and the separation plane produces a curve. If the curve is projected on the xy-plane, it obtains a range of $x; y$ coordinates. The minimal and maximal coordinate value in every axis helps to map the coordinates to the appropriate texture. Let us denote the min/max values as *cut parameters*. These coordinates are generated from all incident rays from the single hemisphere using the standard paraboloid mapping approach (see Figure 4.5, left).

As mentioned above, the skewed cut provides the dense sampling on the certain part of the shadow map. Further, single paraboloid covers one hemisphere. To make sure that both paraboloids cover the adjacent regions with their dense sampled parts (see Figure 4.7), the cut angles and the cut parameters have to be derived separately for both paraboloids. The orientation of the light source may change in order to cover the appropriate area in the scene with the densely sampled part of the shadow map.

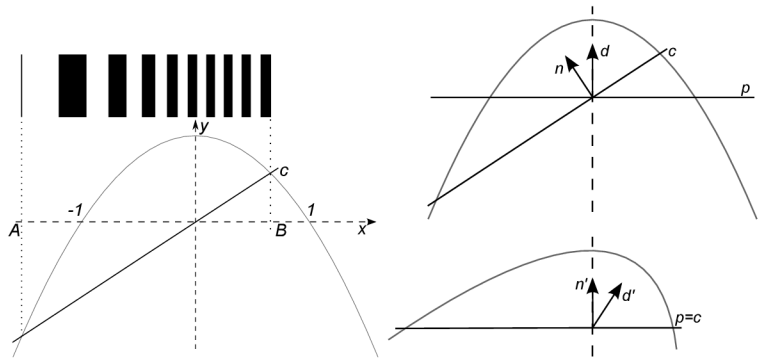


Figure 4.5: (Left) Every stripe denotes the number of pixels (or samples) that can be used to store information from a single solid angle. The skewed cut c produce new coordinates A, B. (Right) d is the direction of the paraboloid, p is the initial separation plane, c is a new separation plane created by the skewed cut and n is the normal of the separation plane.

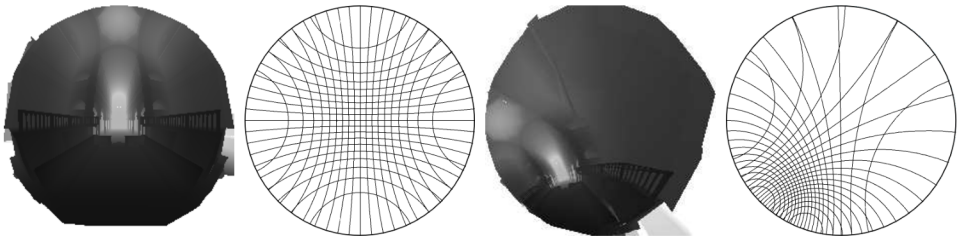


Figure 4.6: Figure shows the resulted shadow map texture of the original DPSM compared to the skewed cut. Every cell of the grid represents how many pixels is used to capture the environment from the given solid angle.

4.2 Improved Non-orthogonal Texture Warping

The approaches described in previous sections still do not address the problem of reducing aliasing in general case. They can improve the quality of shadows for cases where the aliasing error is distributed evenly in the shadow map [37, 27].

This section presents a new approach that solves the problem even for non-uniform distribution of the aliasing error. It presents the key idea for reducing aliasing error for general use case. Section 4.2.2 introduces the solution for this problem which is based on Non-orthogonal Texture Warping (NoTW) scheme. This solution is the contribution of the thesis that has been published in peer-reviewed media [26].

4.2.1 Importance-driven Error Reduction

Section 3.1 defines the aliasing error and shows how it can be measured. It was also shown that the aliasing error can be evaluated for any point in the camera view frustum. The idea of the proposed algorithm is to modify the projection to the shadow map according to value of the aliasing error.

The value of the aliasing error expresses whether the projection of a surface

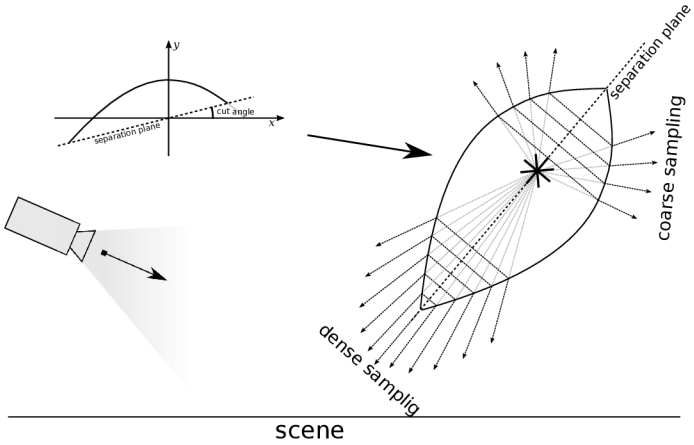


Figure 4.7: Dense sampled areas of both paraboloids should capture the environment close to a camera.

to the shadow map is undersampled (the value greater than 1), or oversampled (the value less than 1). In case of undersampled areas, jagged shadow edges appear. When the aliasing error is projected to the light space, it helps to identify the undersampled and oversampled regions in the shadow map. In these regions, the sampling rate has to be increased or decreased, respectively. the sampling rate can be modified using an improved parameterization of the mapping function. The result of the improved parameterization is that all of the points in the undersampled regions are mapped on a larger area so that the sampling improves while the sampling density of the previously oversampled regions is reduced.

Let us suppose that possible method for projection control is a grid that is placed over the shadow map. By default, the grid cells are rectangular and the projection corresponds to the standard Shadow Mapping algorithm. By changing positions of vertices on the grid, the projection can enlarge the undersampled parts locally and reduce the oversampled parts. Movements of the vertices should be managed so that the reprojected shadow map reaches the equilibrium state. The best result is achieved when the alias error is completely removed so that it equals to 1 in every pixel. However, due to geometric limitations of the shadow map this situation is not achievable. Therefore, the feasible solution is provided when the aliasing error is as constant as possible over the entire shadow map. When the grid reaches a steady state, the shadow map is regenerated with the derived warping function. The same function has to be used in the shadow rendering step. The warping grid projects the surface points on different positions in the shadow map and hence the texture coordinates have to be parametrized using the same warping function.

This section presents the key concept of the NoTW approach. The idea of warping grid illustrates how the projection can be modified. The grid no longer appears in the following text and the improved mapping is derived using a set of warping functions.

The idea of the parameterization of the texture coordinates using warping functions is crucial for the remaining text. It presents the efficient way of improving the shadow quality based on the values of the aliasing error projected to the light space.

4.2.2 Introduction to Improved Texture Warping

Rosen [34] introduced the first method that addressed problem of important regions distributed in the depth texture. He introduced the rectilinear warping maps that could easily control the sampling in particular parts of the depth texture. This could be controlled by importance function and the approach could be used for point light sources without complex modification. Nevertheless, the rectilinear warping schema is not completely local and some parts of a scene may receive resolution higher or lower than required and that situation is not optimal.

Similar approach was published by Jia et al. [12]. They do not limit the approach to rectilinear grid; therefore, they can control the results more precisely. However, this approach needs multiple render passes of the scene to analyze the scene and decides the dividing schema. This can introduce certain issues for complex scenes.

The improved warping parameterization described in this thesis reduces the aliasing artifacts, and it allows to render high quality shadows regardless of a light source or a camera position in the scene.

The approach computes an improved parameterization based on importance driven depth texture warping. It identifies regions in the depth texture where the sampling is not optimal and enlarge this regions in order to get higher sampling rate. Before the traditional Shadow Mapping algorithm, an additional step of generating the non-orthogonal warping functions have to be applied. These functions are used later during the shadow rendering.

The main contributions are:

- Introduction of a novel importance function for determining sampling rate of depth texture. This function extends the set of functions introduced by Rosen et al. [34].
- The Non-orthogonal Texture Warping (NoTW) scheme which leads to better control of importance-based warping without affecting the nearest regions in the texture (in the same row and/or column).

The Non-orthogonal Texture Warping (NoTW) algorithm is partially based on Rectilinear Texture Warping (RTW) approach [34] (see Section 3.2.3 for details). The RTW approach utilizes various properties of view samples, e.g. distance to a camera, normal vector or edge detection. The warping function can be constructed using forward, backward or hybrid analysis.

The first step in the forward analysis is rendering of the scene from the light source point of view. Then, the importance map is computed. In the backward analysis, the G-buffer with the scene's depth and color is rendered from a camera point of view. Then, the importance analysis is performed using samples projected into the light space. The hybrid analysis combines both approaches.

The backward analysis is the fastest method because it requires a scene to be rendered only two times. The first rendering pass is used to create a depth buffer from the camera. The second rendering pass creates a warped shadow map. Its complexity is linear with relation to the number of light sources.

The warping function in RTW is composed of two 1D warping functions that operate in projection plane of a light source (see Figure 4.8). These functions are derived from an *importance map*. The importance map is constructed by projection of view samples onto the projection plane of a light source. Multiple view samples

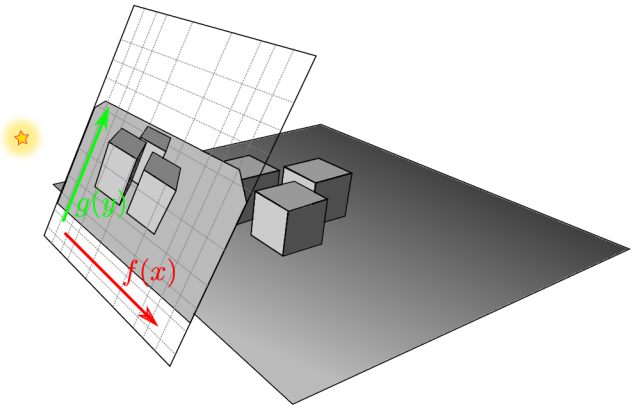


Figure 4.8: Two 1D warping functions enlarge parts of the scene that are important according to the importance map. It is not always optimal with the rectangular grid.

can be projected into one pixel of the importance map. In every pixel, the importance value is computed based on the view sample properties. The 1D warping functions are derived separately for column and rows according to a maximal importance value. Since the functions parameterize vertical and horizontal component of the shadow map separately they produce an orthogonal warping grid.

4.2.3 Shadow Rendering Using Warping Functions

The basic idea of the NoTW algorithm presented in the thesis is to achieve better distribution of view samples in the shadow map. Every shadow sample resolves shadow for all view samples that were projected on it (the detail explanation of view and shadow samples and their relation to the aliasing error are presented in Section 3.1). The ideal situation occurs when one texel from the shadow map samples a surface that is projected onto one pixel in the image space. However, this is hardly achievable in most of the scenes because of the scene complexity, geometry and mutual position of the camera and the light source. Assume that the best result is observed when the number of view samples for all shadow samples is the same.

In NoTW algorithm, the importance map has the same resolution as the shadow map. Every pixel in the importance map stores the number of view samples that were projected onto the given shadow map texel. The importance map can be created by projection of view samples into to the light space and increase a counter by one. This step can be easily accelerated by contemporary GPUs.

The complete algorithm for computing shadow consists of the following steps:

1 Render a scene from a camera point of view to G-buffer
2 Project every view sample into the importance map
3 Compute prefix-sum for every row in the importance map
4 Construct the set of warping functions for rows according to Equation 4.7. Use the prefix-sum from the Step 3
5 Smoothen the set of warping functions, e.g. using weighted average
6 Project every view sample onto the importance map (and increment by 1) leveraging the set of warping functions created in the previous step
7 Repeat the Steps 2-5 for all columns
8 Create shadow map using both sets of warping functions
9 Evaluate shadows in the scene using G-buffer, the set of warping functions and the warped shadow map

Algorithm 3: Non-orthogonal Texture Warping.

The first step is generation of the G-buffer. Apart from other properties, it contains positions of view samples. The importance of the samples is then analyzed. The steps 2-7 are the most important ones and they are used to construct the set of 1D warping functions. The warping functions are derived in different manner than Rosen [34]. For every row and every column, 1D warping function is constructed separately and thus it does not allocate unneeded resolution in other parts of the shadow map. The degree of freedom for warping functions is increased using this approach and the situation illustrated in Figure 4.9 is not possible. The steps are described in detail in the following section.

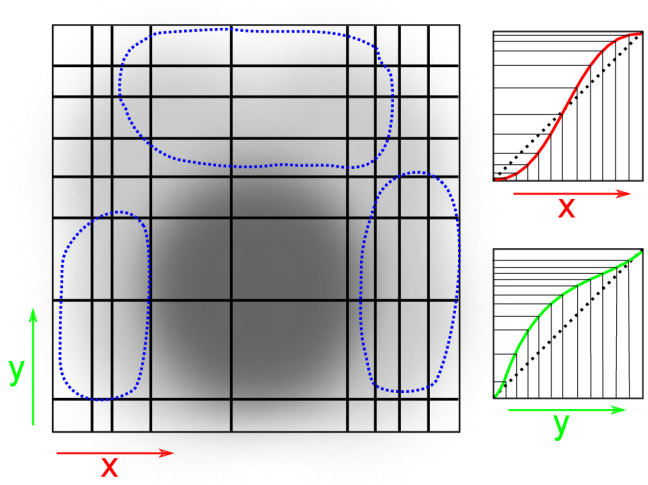


Figure 4.9: Importance map for RTW: Combination of two 1D warping function (left) , two 1D warping function (right) It can be seen that blue parts are oversampled. The larger cells cover more important areas of the shadow map.

4.2.4 Construction of 1D Warping Functions

For one row of the importance map, let us assume a function $f(x)$ that returns the number of view samples on a normalized position x and its corresponding prefix-sum function $g(x)$:

$$n = f(x) \quad x \in \langle 0, 1 \rangle \tag{4.4}$$

$$s = g(x) = \int_0^x f(x)dx \tag{4.5}$$

For evenly distributed view samples in the row, the ratio of the number of view samples on all positions before x, i.e. $g(x)$, and the total number of view samples $g(1) = N$ is equal to ratio of the position x and the row length:

$$\frac{g(x)}{g(1)} = \frac{x}{1} \tag{4.6}$$

Expression $g(x)/g(1) > x/1$ implies that there are more view samples than the number of samples x and thus the area needs to be enlarged to achieve uniform sampling rate. On the other hand, expression $g(x)/g(1) < x/1$ implies that there are less view samples and the area can be smaller.

Now, the warping function can be derived so that it is defined as an offset $o(x)$ that has to be added to the actual view sample position. The offset function is given by:

$$o(x) = \frac{g(x)}{N} - x \tag{4.7}$$

Let us assume that the view sample is projected onto a particular row in the shadow map. Then, a new sample position x' in the row is given by:

$$x' = x + o(x) \tag{4.8}$$

Before the algorithm proceeds with construction of warping functions for columns, the importance map has to be recomputed again. But now, the newly derived set of 1D warping functions for rows are applied. After this step, the number of view samples that have to be redistributed in a given column is nearly constant (see Figure 4.10). When the 1D warping functions for columns are derived, all the view samples are distributed more uniformly.

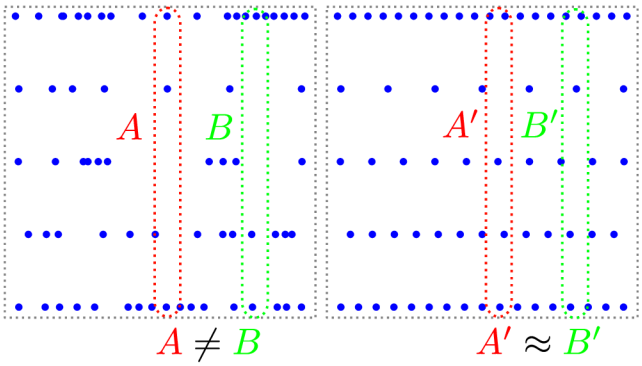


Figure 4.10: (Left) Five rows of the importance map. Blue dots indicate view samples. (Right) the importance map constructed using the set of row warping functions. Columns in the left do not contain the same number of view samples. Columns in the right contains approximately the same number of view samples.

Section 3.2.3 mentioned that the RTW algorithm constructs two warping functions - for vertical and horizontal direction, respectively. This approach is improved in this work by constructing set of warping functions for all rows and all columns at the same time. Nevertheless, these functions have to be smoothed in order to limit the warping amplitude. Otherwise, the large polygons that are linearly rasterized would not be processed by the warping functions correctly. The quality can be controlled by adjusting the size of smoothing window when averaging the warping functions. The wider the window is the smoother are the warping functions. The smoothing step is included in the RTW algorithm as well. It can be implemented, for instance, as a weighted average of the results based on the number of view samples on a row or a column, respectively (see Figure 4.11).

The complete warping function can be expressed as:

$$\begin{aligned} warp(x, y) &= (x + o_x^{(i)}(x), y + o_y^{(j)}(y)) \\ i &= \lfloor y \cdot w \rfloor \\ j &= \lfloor (x + o_x^{(i)}(x)) \cdot w \rfloor \end{aligned} \tag{4.9}$$

where w is the shadow map resolution (number of pixels in one row), $o_x^{(i)}(x)$ is a warping function for i^{th} row, $o_y^{(j)}(y)$ is a warping function for j^{th} column.

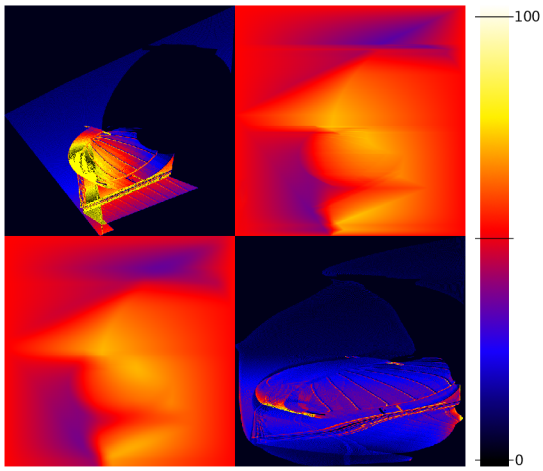


Figure 4.11: (Top, left) Importance map. (Top, right) A set of warping functions for every row of the importance map. (Bottom, left) Smoothed warping functions. (Bottom, right) the importance map after application of row warping functions - importance map for columns. Yellow color in warping functions means positive offset for a particular position in the row.

When both sets of warping functions are applied, the view samples projected onto the projection plane of a light source are better spread as it can be seen in Figure 4.12.

Once both sets of the warping functions are constructed, the shadow map can be rendered (see Step 8 of the proposed Algorithm 3). A surface point with world space coordinate $v = (v_0, v_1, v_2, 1)$ is projected onto the shadow map in Algorithm 4.

Input: **v** - vertex in world space, M - light projection view matrix
Output: **p** - vertex in the shadow map clip space

1 $\mathbf{a} = M \cdot \mathbf{v}$;
2 $\mathbf{b} = ((a_1, a_2)/a_4 + 1)/2$;
3 $\mathbf{c} =$ `warp(b)`;
4 $\mathbf{d} = (\mathbf{c} \cdot 2 - 1) \cdot a_4$;
5 $\mathbf{p} = (d_1, d_2, a_3, a_4)$;

Algorithm 4: Warping function that can be used in vertex / evaluation shader. Steps 1, 2 project vertex into normalized coordinates of shadow map. Step 3 moves vertex according to warping functions. Steps 4, 5 project vertex back into shadow map clip space.

4.2.5 Minimal Shadow Frustum Extension

The Non-orthogonal Texture Warping algorithm is extended with an additional improvement. The technique for finding a *Minimal Shadow Frustum* (MSF) [36] was implemented, and it was extended using rotating caliper (see Algorithm 5). Using this technique, the NoTW algorithm projects only parts of the scene that are visible in the camera view frustum and occluders outside the frustum that cast shadows on objects inside the frustum. However, since the MSF algorithm is complex, it runs on CPU and thus it may influence rendering speed. Moreover, issues caused by precision of floating point operations have to be considered during implementation.

Rosen presented Desired View (DV) function that works similarly to the MSF. However, he did not clearly show how it influences the overall quality. The NoTW algorithm supports the DV as well, but it is only used as pre-process step before computing the importance map. The DV simply finds minimum and maximum view samples coordinates in the importance map. In addition, the MSF rotates the bounding box to an optimal position and adjusts near and far planes. Rosen computes the DV in the RTW approach from the importance map by finding first/last row and column that contains an importance value greater than zero. In the NoTW approach, the DV is computed by parallel reduction over the set of view samples projected into the shadow map space. It does not contribute to warping process, but it only crop the relevant part of shadow map.The DV function can be applied before construction of the warping functions (before the Step 2 of the Algorithm 4).

4.2.6 Summary

This work presents an extension of the Rectilinear Texture Warping algorithm achieved through the improved non-orthogonal warping scheme constructed using the set of 1D warping functions. The novel importance warping functions result in better sampling distribution at the shadow edges.

Standard methods for aliasing reduction globally change sampling rate using partitioning of a scene where directional light sources are commonly used. The Non-orthogonal Texture Warping algorithm changes sampling rate locally and thus it can be used with other kinds of light sources using DPSM or Cube Shadow Maps.

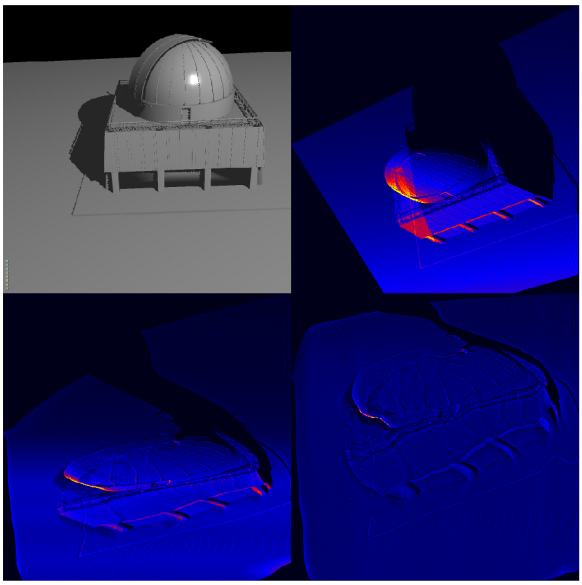


Figure 4.12: (Top, left) Scene rendered from a camera point of view. (Top, right) the importance map created from view samples. (Bottom, left) Reprojected view samples using only row warping functions. (Bottom, right) Reprojected view samples using both sets for warping functions.
It can be seen that importance is more spread across the importance map in the final stage. Black parts of second image are pixels with no view samples. These pixels correspond to those shadow map pixels that are useless - they resolve shadowing equation for invisible parts of the scene. In final image, these black parts almost disappear.

Data: S - convex hull of the scene,
V - convex hull of the camera view frustum,
$\mathbf{L}$ - position of a light source
Result: minimal shadow frustum

1 $SV = S \cap V$
2 $E = convexHull(SV \cup \mathbf{L})$
3 $O = E \cap S$
4 $\mathbf{C} = centerOf(O)$
5 Find near and far plane for $\mathbf{L}$ using $\mathbf{C} - \mathbf{L}$
6 Find silhouette edge of O for $\mathbf{L}$
7 Use rotating caliper algorithm over silhouette edges for finding of minimal frustum
8 Construct view and projection matrix from planes

Algorithm 5: S is a convex hull of the scene, V is a convex hull of the camera view frustum and $\mathbf{L}$ is a position of a light source.

CHAPTER 5

Experimental Results and Discussion

The Non-orthogonal Texture Warping scheme has been evaluated on various scenes. The experiments performed and described in this chapter show that the approach is fast and capable of rendering high-quality shadows for complex light sources. Also, various improvements and extensions that can be used together with the NoTW algorithm are discussed.

5.1 High-quality Shadows

The NoTW algorithm improves the Shadow Mapping algorithm. The most important contribution of the NoTW algorithm is the reduction of the aliasing error in a scene and increasing the quality of rendered shadows. In the Shadow Mapping algorithm, poor quality shadows can be rendered which produces "jagged" shadow edges. In order to evaluate precision of rendered shadows, the Shadow Volumes algorithm was chosen as the ground truth, because it provides sample-precise shadows (see Figure 5.1).

This Section presents various scenes on which the evaluation has been performed. The output images show incorrectly computed shadow pixels in red color.

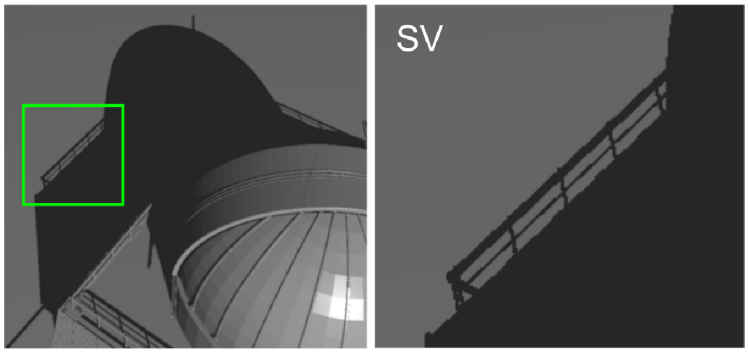


Figure 5.1: The reference image illustrates the Observatory scene (left) and zoomed detail of the image (right) that is used for evaluation of the quality.

5.1.1 Comparison with Standard Shadow Mapping

This section shows differences in quality between the standard Shadow Mapping algorithm as introduced in Section 2.3 and the NoTW algorithm. The results were measured for Observatory scene on 1024×1024 resolution of the output image and with 512×512 resolution for the shadow map. In Figure 5.2, differences from the reference solution are presented.

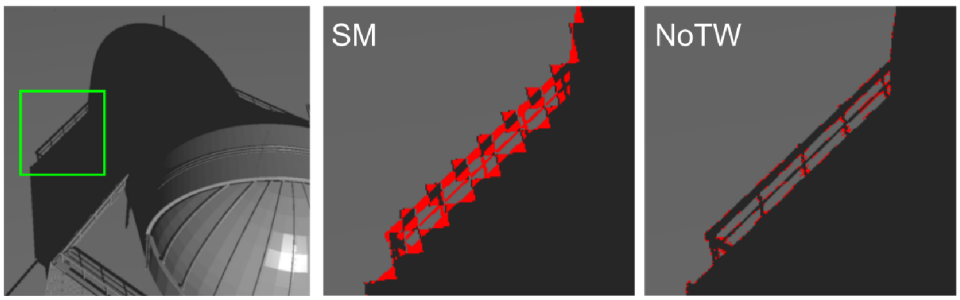


Figure 5.2: (Left) the reference image of Observatory. (Middle) the detail rendered with the Shadow Mapping algorithm. (Right) the same detail rendered with the Non-orthogonal Texture Warping algorithm.

The basic Shadow Mapping algorithm has no ability to focus on the current camera view. It covers the whole scene with the shadow map and the aliasing error in this case is really high. The red pixels in Figure 5.2 (middle) illustrate that many view samples were projected onto a single shadow map texel. The NoTW algorithm, on the other hand, project the view samples more uniformly to the shadow texels.

5.1.2 Comparison with RTW

The Rectilinear Texture Warping (RTW) algorithm is the most similar approach to the NoTW approach and since some improvements of the RTW algorithm are suggested in Section 4.2, the visual quality has been explicitly compared to the RTW algorithm as well. Implementation of RTW algorithm with backward analysis has been used for creation of the importance map. Both the Distance to Eye and the Desired View importance functions were enabled in all reference images (see Section 3.2.3 for more details about the importance functions).

Figure 5.3 (left) shows that the sampling distribution is more uniform in the RTW algorithm in comparison to the standard Shadow Mapping algorithm presented in the previous section.

The results of the algorithms were compared for three scenes (see Figure 5.4). To show that the solution can be adapted to different scenarios and types of light sources, various types of scene (outdoor as well as indoor) have been selected. Note, that in all three scenes the NoTW algorithm produces consistently the best results although in some scene details the results of RTW and NoTW are relatively close to each other.

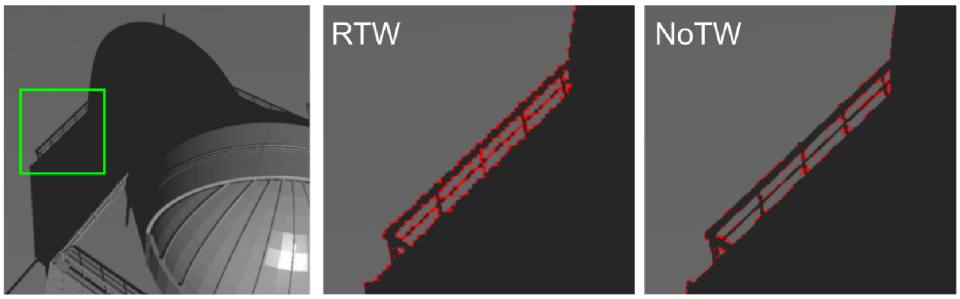


Figure 5.3: (Left) the reference image of Observatory. (Middle) the detail rendered with the Rectilinear Texture Warping algorithm. (Right) the same detail rendered with the Non-orthogonal Texture Warping algorithm.

5.1.3 Limitation of PSSMs

The most widely used technique for shadows rendering is the Parallel-split Shadow Mapping algorithm (see Section 3.2.2). It is very efficient approach that renders high quality shadows namely for large outdoor environments. It has been optimized to be used with directional light sources and spotlights.

The NoTW algorithm omits comparison with the Parallel-Split Shadow Mapping approach since it is not applicable for omnidirectional light sources. The explanation can be shown in Figure 5.5. The omnidirectional light source is supposed to be dynamic and it can move through the scene. The light source is mostly visible in the camera view frustum and this scenario is difficult to address with PSSMs.

The NoTW algorithm is supposed to work with all types of light sources and scenarios. The comparison with approaches that are optimized for a specific use case is not in favor of NoTW; anyhow, the NoTW algorithm compares quite well to these algorithms and the results it produces are comparable.

5.2 Performance

This section presents experiments related to the speed of the Non-orthogonal Texture Warping algorithm (NoTW). Every improvement in quality can bring additional computation cost, however, it still has to maintain interactive rates. Moreover, the Shadow Volumes algorithm defines a lower boundary for speed. In the following text, all approaches have been compared to fully optimized and accelerated Silhouette-based Shadow Volumes approach introduced by Milet et al. [24].

5.2.1 Basic Shadow Algorithms

Table 5.1 shows frame times for all scenes depicted in Figure 5.4. This is a basic performance comparison of the NoTW algorithm with different approaches.

The accelerated Shadow Volumes algorithm (SV) introduced by Milet et al. is the slowest. It can be seen that the frame times depends on the scene complexity. This is a common property of all shadow rendering algorithms and namely the Shadow Volumes. On the other hand, the standard Shadow Mapping algorithm (SM) is the fastest approach, but it has the worst quality of the output as described

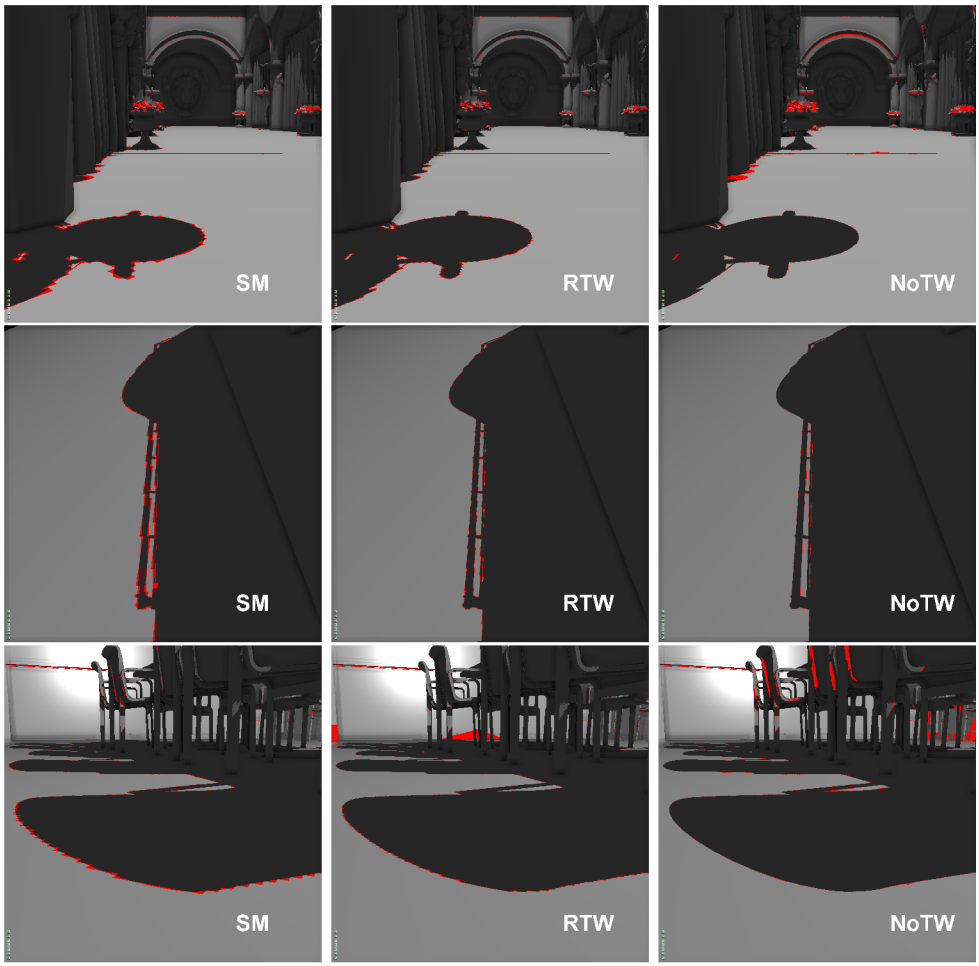


Figure 5.4: Images show differences in quality between techniques based on the Shadow Mapping algorithm and Shadow Volumes algorithm which is considered as ground truth. First scene is Sponza, second scene is Observatory and last scene is Conference room. Times are shown in Table 5.1.

in Section 5.1. The RTW as well as NoTW approaches performs better than SV and the timings are almost equal.

5.2.2 Frame Times of Warping Techniques

The RTW and NoTW algorithms perform almost equally in terms of the frame times. This section discusses the overhead of individual steps in both algorithms. Section 4.2 shows that the idea of the NoTW algorithm is very similar in comparison to the RTW approach. Firstly, the importance map is created using the simple scene analysis. Then, the map serves as an input for deriving of warping functions.

In the NoTW algorithm, the Desired View (DV) function (or Minimal Shadow Frustum extension) is employed only to crop the part of the shadow map where no view samples were projected (see Section 4.2.5). It runs as a pre-process step before the warping functions are derived. However, in RTW algorithm, the DV function is one of the importance functions that contribute to the importance map analysis

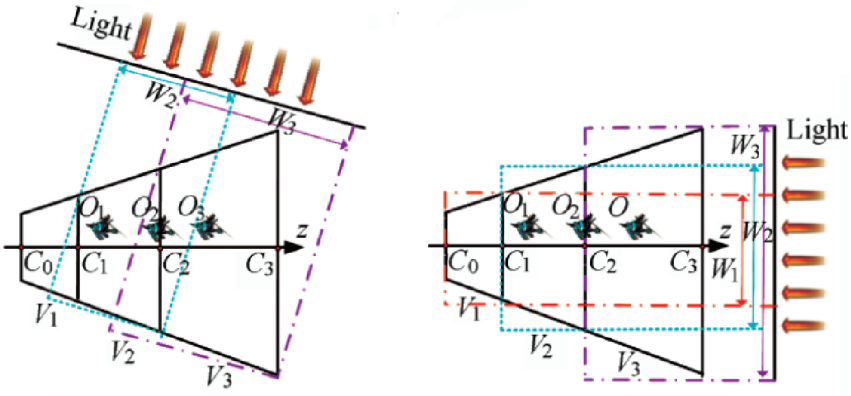


Figure 5.5: Redundant usage of shadow maps resolution when the direction of the light source is almost parallel with the camera view [18].

Scene	Conf. room	Sponza	Observatory
triangles	126665	261978	52583
gbuffer	2.16	2.229	1.84
SV	9.64	18.41	14.96
SM	0.21	0.40	0.16
RTW	3.14	3.47	3.02
NoTW	**3.63**	**3.84**	**3.23**

Table 5.1: Performance comparison of implemented methods for different scenes. Times are in milliseconds.

(see Section 3.2.3). It only adds a weight to a sample in the importance map in the same way as other importance functions.

Figure 5.6 shows what portion of the frame time the algorithms spend on analyzing of the importance map. The plot shows that the NoTW algorithm spends about 15% of time on the pre-processing using the DV function. The overall times for the importance map analysis are slightly better for the NoTW algorithm. However, rendering of the shadow map is more costly, because the warping scheme is a little more complicated in comparison to the RTW algorithm.

5.3 Complex Light Sources

From the Shadow Mapping algorithm point of view, omnidirectional light sources are considered to be complex light sources. They require additional computation steps to be capable of rendering shadows into all direction.

Omnidirectional light sources introduce an advanced use case and it brings additional complexity to the algorithm. The Non-orthogonal Texture Warping algorithm supports also this type of light sources and this section presents visual as well as performance comparison of the Cube Shadow Mapping (CubeSM) and Dual-Paraboloid Shadow Mapping (DPSM) algorithms (presented in Section 3.3) extended with the NoTW scheme. It shows that the NoTW algorithm is applicable

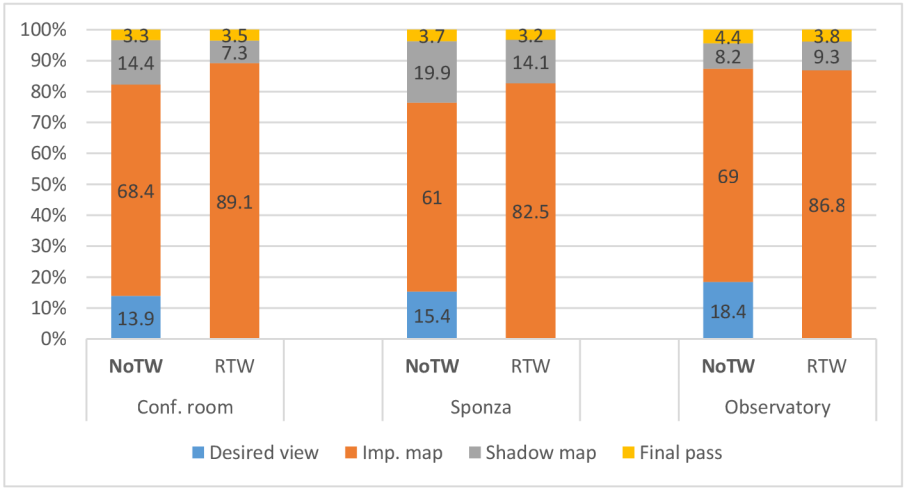


Figure 5.6: Overhead of steps in our algorithm for different scenes. Values are in percent.

to arbitrary use case when it is integrated into existing algorithms for omnidirectional shadow rendering and extended with a zooming feature (e.g. Desired View function or Minimal Shadow Frustum extension).

5.3.1 Omnidirectional Light Sources

To validate robustness of the NoTW algorithm, a simple but still general use case was chosen. The light source is considered as a dynamic object that can be easily visible in the camera view frustum as it travels through a scene. The reason is that in this case, the CubeSM as well as the DPSM have to fully use their resources. In Figure 5.7, one such a use case is depicted.

The next step in evaluation of the visual quality is comparison of the rendered shadow maps (see Figure 5.8). It illustrates how the improved parameterization modifies the shadow map and how the vertices are moved from their initial positions. It is expected that when warping functions are applied, the scene rendered into the shadow map is highly deformed and objects are not be clearly recognizable. Also, it is necessary to apply the same functions in the process of computation shadows when the samples are projected into the light space. The warping scheme in the NoTW as well as RTW algorithm has to ensure that all samples are projected on the correct place in the shadow map.

Finally, Figure 5.9 illustrates the count maps that were analyzed in order to derive the warping functions. Closer look shows that DPSM algorithm is more efficient in using the space available in the map. This is the reason why the DPSM algorithm extended with the NoTW scheme produces better results. Since one side of the paraboloid covers a bigger part of the scene than one cube face frustum, there is more oversampled regions in the shadow map rendered with the DPSM approach. In other words, there is more space where the view samples can be distributed.

However, the warping functions had to be smoothed as described in Section 4.2.4. Therefore, the warping functions do not distribute the view samples over the entire shadow map. The smoothing factor is controlled by the user and it was set manually for each of the testing scenes.

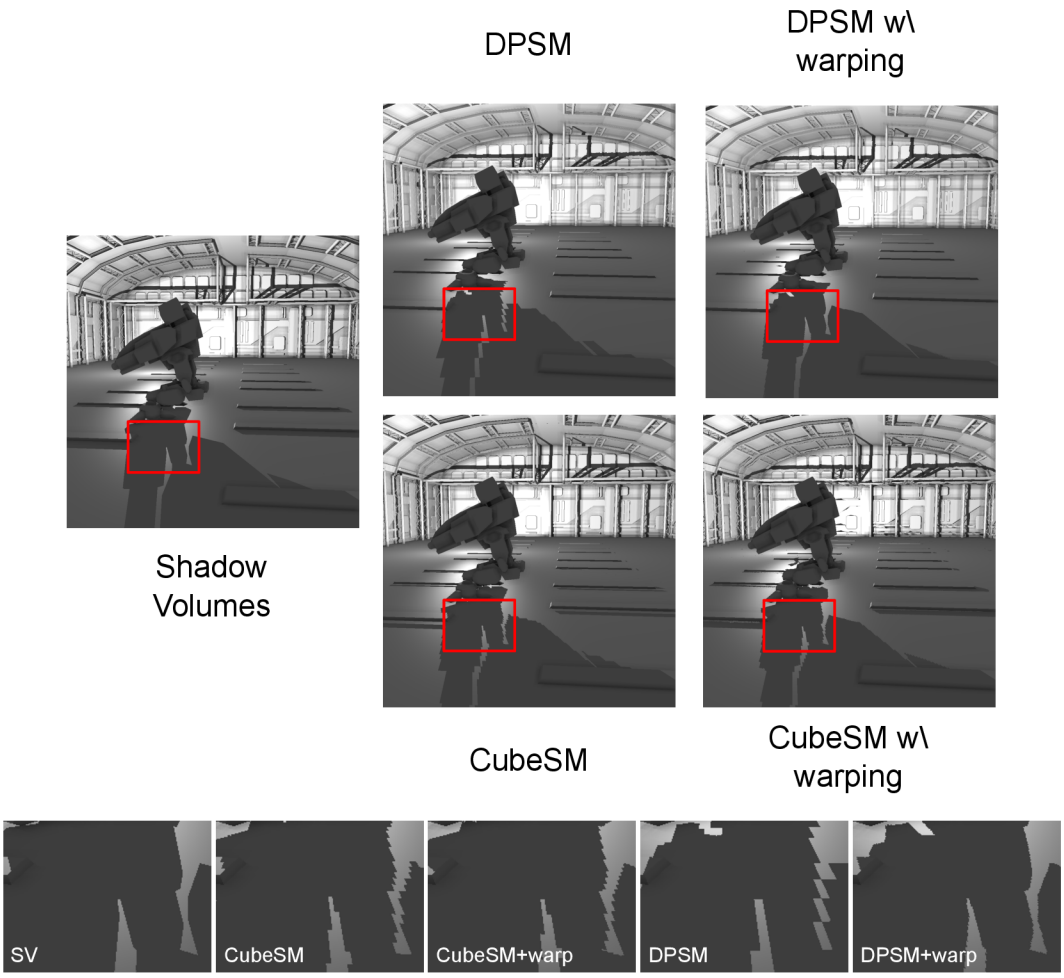


Figure 5.7: Shadow quality can be compared e.g. according to shadow boundaries. „Jagged“ edges shows that the ratio between view samples and shadow samples is high. Shadow Volumes algorithm rendered the reference image.

Performance

Since the Shadow Mapping algorithm consists of various steps, execution times of the steps were also measured for all tested approaches.

The Table 5.2 shows that the biggest impact in NoTW approach is observed in rendering of the shadow map, because of the importance map creation, analysis, and deriving of the warping functions. It has to be noted that even though the Shadow Volumes algorithm is fully optimized and capable of running in real-time, the frame times are not stable between frames. It depends on complexity of the scene and in the worst case, rendering of single frame took 16ms. For shadow mapping-based approaches, the times were stable.

5.3.2 Effect of Desired View

Experimental results also showed that extension of the NoTW algorithm with the Desired View (DV) function (or Minimal Shadow Frustum extension, MSF) is major part of decreasing alias error, but in some situation it is not sufficient. The main

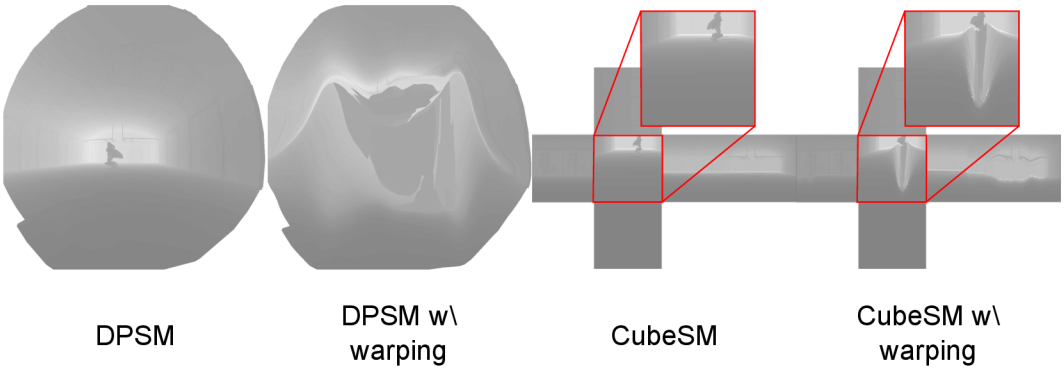


Figure 5.8: Comparison of shadow maps. The warping functions cause the scene is hardly recognizable.

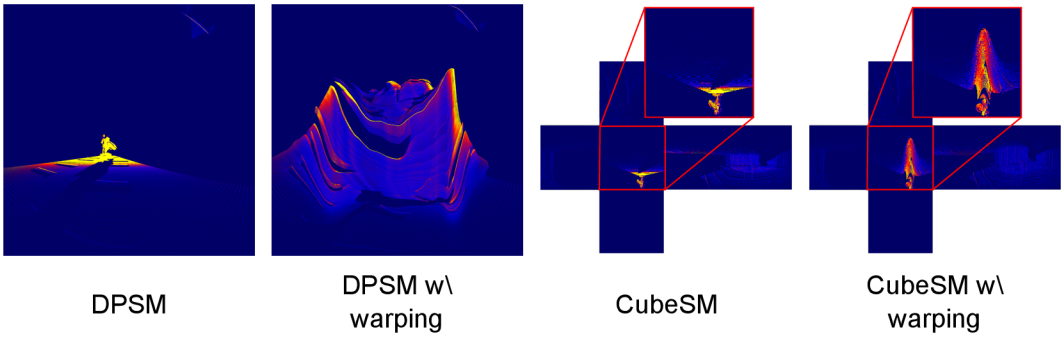


Figure 5.9: Comparison of count maps. For the Cube Shadow Mapping, only one face contains most of the shadow samples.

reason for focusing on these methods is that it should confirm whether the DV or MSF is not sufficient enough to render images of the similar quality. Since Rosen [34] described this importance function, however, he did not show any results.

The MSF or DV perform better than the texture warping techniques when a small part of a scene is rendered. However, in real world scenes the camera renders a bigger part of a scene and in this case the warping techniques perform better (see Figure 5.10 and 5.11). The MSF or DV do not generate the view frustum small enough and thus artifacts on shadow edges are more apparent. The performance of DV and MSF depends on current hardware setup. MSF performs better than DV when running on fast CPU and slow GPU.

Method	Frame time	SM rendering	Shadow computation
SV	4.8*	N/A	N/A
CubeSM	2.5	0.38	0.10
DPSM	2.3	0.16	0.09
CubeSM w/ warping	**5.8**	**3.6**	**0.11**
DPSM w/ warping	**3.4**	**1.2**	**0.10**

Table 5.2: Performance comparison of implemented methods for omnidirectional shadow rendering. Times are in milliseconds.

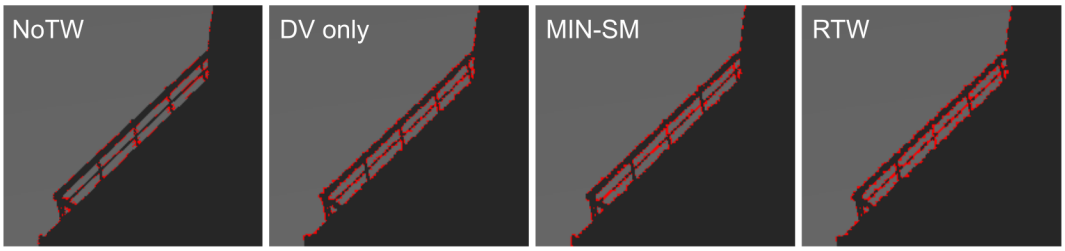


Figure 5.10: (From left to right) complete Non-orthogonal Texture Warping (NoTW) including warping, NoTW with only DV function (no warping applied), Shadow Mapping with MSF extension (MIN-SM), Rectilinear Texture Warping (RTW).

The effect of the DV function is nicely visible in Figure 5.11. First image shows the Observatory scene rendered using the traditional Shadow Mapping algorithm with a directional light source. Second image, shows how the scene is zoomed on the part visible in the camera view frustum when only the DV is applied as a pre-process step in the NoTW algorithm. The RTW includes the DV function in the set of importance functions by default and it is used to analyze the importance map and derive the warping functions. The last image depicts the NoTW algorithm. It is similar to the result from the RTW algorithm but the parameterization is a bit different which leads to lower alias error (see the shadow maps in Figure 5.11).

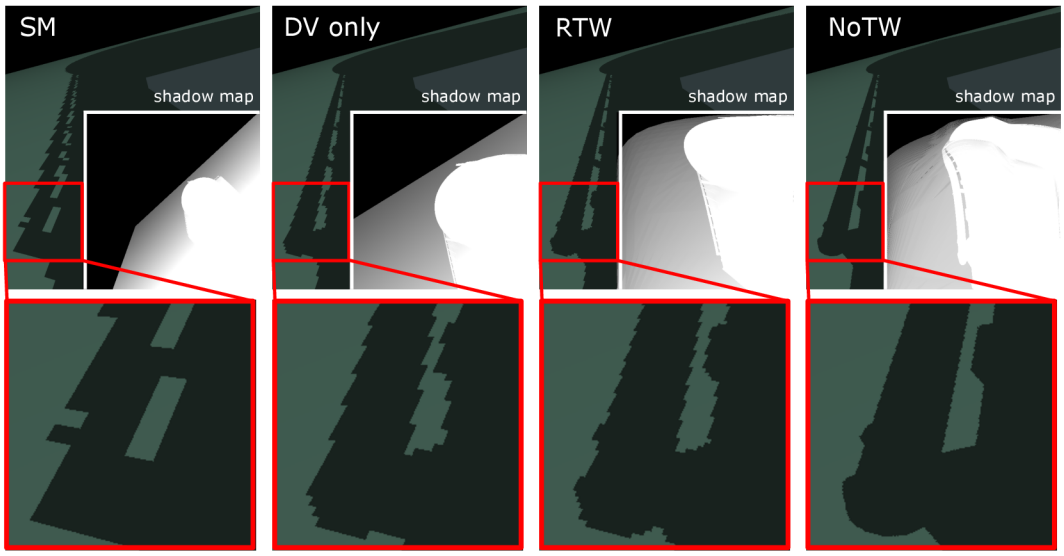


Figure 5.11: Images show shadow maps for Observatory scene. (From left to right) Shadow Mapping (SM), NoTW with only DV function (no warping applied), Rectilinear Texture Warping (RTW), complete NoTW including warping.

The Desired View (DV) function and Minimal Shadow Frustum extension, MSF) help to generate the high-quality shadows (see Figure 5.10) with a small additional cost. However, when the warping techniques employ all their features, the results are even better and the impact on performance is not crucial (see Table 5.3)

The similar effect as the DV function has the Improved Paraboloid Shadow Mapping (IPSM) [37] approach introduced in Section 4.1. The IPSM has been

Method	time per frame
SM	1.596
MIN-SM	1.7
SV	8.750
RTW	3.296
NoTW	**4.708**
NoTW-DV	**2.521**

Table 5.3: Performance comparison of implemented methods. Times are in milliseconds.

designed for optimization of the Dual-Paraboloid Shadow Mapping algorithm in cases when the light source is outside the camera view frustum.

5.4 Discussion

The motivation for this work has been an idea that most of the researchers have been focused on improving shadow quality in the Shadow Mapping algorithm, but only for directional light sources, spotlights and large environments, e.g. Parallel-Split Shadow Maps [40] presented in Section 3.2.2. The Non-orthogonal Texture Warping (NoTW) algorithm has been initially designed for methods involved in omnidirectional shadow casting where improving of the shadow quality has not been explicitly investigated.

Experiments presented in this Chapter showed that the NoTW algorithm is successfully usable in various environments without any modification. Algorithms that depend on view and scene context could be replaced with one solution. Applications can save some time when they do not have to deal with an expensive switching between multiple methods with different data structures and demands on resources.

The initial proof-of-concept has been implemented on contemporary GPUs and the algorithm runs in interactive rates. At this point, there is a space for further optimizations and improvements. The deriving of the warping functions can be further optimized with parallel processing units, e.g. CUDA.

5.4.1 Limitations

However, the solution has also some disadvantages. The NoTW algorithm as well as the RTW algorithm have to deal with the linear rasterization unit. Figure 4.12 (bottom right) shows how the warping functions distorted the space. Nowadays, the rasterization pipeline can handle only the polygonal mesh. If the warping function changes rapidly between two vertices, some errors can be seen (see Figure 5.4 top, right for missing shadows under curtains). Lloyd introduced the nonlinear rasterizer [20] that could replace the traditional rasterization pipeline and allow for processing non-linear data.

In the experiments, a few techniques have been used in NoTW to deal with these errors. Firstly, it utilized the adaptive tessellation provided by OpenGL. The similar improvement was suggested by Rosen et al [34]. Further, the quality can be controlled by adjusting the size of smoothing window. Another solution is to use

weights during smoothing step. It can influence sizes of offset values. In the experiments, these parameters were manually set to fit the current view. When they were set inappropriately, the warping functions do not work correctly so that it totally deforms the shadow map and also produces artifacts and incorrectly computed shadows in the output image.

In the future work, some constraints have to be defined that should be involved in deriving of the warping functions. It should allow for adjusting the parameters automatically and render the output image with the highest quality.

The limitation of the NoTW algorithm is also missing support for Minimal Shadow Frustum extension (see Section 4.2.5). It should perform better that the Desired View function, but due to precision issues in floating point arithmetic it ended only as a prototype with a very poor performance. However, the basic version of the technique has been used in the standard Shadow Mapping algorithm.

5.4.2 Implementation Details

The algorithm has been implemented in OpenGL 4.4 using compute shaders. For creation of the importance map, image atomic operation *imageAtomicAdd* that occurs in OpenGL has been used to save time spent on GPU. The results were measured on a PC running Intel Core i7 4790 with 16GB of memory. The scenes were rendered on a high-end GPU: NVidia GTX 980 and Titan X. Operation system was Linux Ubuntu 14.04.2.

The solution requires additional memory in comparison to the basic Shadow Mapping algorithm. Deferred shading has been used for creation of the G-buffer that requires set of 2D textures. Two one-channel floating point 2D textures have been used for storage of the warping functions with the same resolution as the shadow map. Furthermore, the algorithm requires few textures for storing temporary results - the importance map, prefix sum map and storage for warping functions. The additional memory requirements are thus dependent on the shadow map resolution.For instance, when using the shadow map with resolution $w = 1024$, additional 20 MBytes of the memory needs to be allocated.

The memory requirements can be decreased by using e.g. another format of textures. For instance, 16bit textures for the importance map or prefix-sum map. Also, with increasing number of lights, the memory requirements increase only for storing the warping functions: $8w^2$[bytes] for one light source.

5.4.3 Finite Elements Methods in Shadow Rendering

The sections above showed that the solution presented in Chapter 4 can be successfully used in interactive applications.

However, alternative ways of texture warping were investigated to improve shadow quality, but they did not perform successfully. This section introduces an idea of integrating Finite Element Methods (FEMs) [23] to shadow rendering algorithms. the FEMs should be employed for computing the warping grid that was introduced in Section 4.2. the warping grid manages the projection into the shadow map and moving its nodes controls the sampling rate for a particular area in the shadow map.

Before the solution based on FEMs is introduced, the concept of the warping grid has to be slightly reformulated. the grid has lower resolution than the shadow map. For example, the shadow map with resolution 1024^2 should be covered with 16×16

grid. the FEMs solution is driven by values in grid cells. They contain pixels with the aliasing error values projected to the light space. the definition of the aliasing error in Section 3.1 can be approximated so that the sum of the error values in the cell introduces the minimal area the cell should have in order to minimize the aliasing error (the error in all pixels equals 1). This implies that the cell tends to increase its area to reach the equilibrium state. On the other hand, when the aliasing error in a cell is less than the area of the cell, it tends to decrease its size. From the FEMs point of view, the grid cell is considered to be an element with some „energy“.

The previous assumptions could be used to derive the solution for reducing the aliasing error in the Shadow Mapping algorithm using FEM. the scientific contribution of this approach would inhered in using non-graphic component in computer graphics algorithm (FEMs are mostly used in civil engineering field). Let us proceed to detailed explanation of the process of integration Finite Element Methods into the Shadow Mapping algorithm.

The grid could be considered as a system of springs connected in nodes of the grid. The behavior of the system is defined by Hooke's law:

$$F = k \cdot x \tag{5.1}$$

In FEMs terminology, k is called a *stiffness matrix* and its size depends on the number of nodes in the system. The solution using the FEMs can be divided into the following steps:

1. Divide the body into finite elements.
2. Describe behavior of each element - define stiffness matrix for each element.
3. Assembly - create global stiffness matrix.
4. Solve the system of linear equations.

Computation of matrices in steps 2 and 3 can be solved in two different ways. First option described in the following text is the *Direct Stiffness Method*. It derives the stiffness matrix for each element (Step 2) and assemblies them into the global matrix so as it fulfills the force equilibrium in the system (Step 3).

In Hooke's law, force F depends on displacement x. The same relation has to be define for the warping grid with the aliasing error. Based on the assumption derived at the beginning of this section, the relation between the aliasing error and the area of the grid cell could be given by:

$$e(A) = \frac{A_0}{A}\frac{1}{K} \tag{5.2}$$

where A_0 is the initial area of the grid cell, A is newly computed area, and K expresses the initial measurement of the error in the cell. In the simplest case, K can express the sum of the aliasing error values in all pixels in the cell, or e.g. its square root.

This equation was used to derive the solution using the *Direct Stiffness Method* [23] that leads to the system of equations. However, the experiments showed that the solution is not correct and it caused the warping grid was malformed (see Figure 5.12).

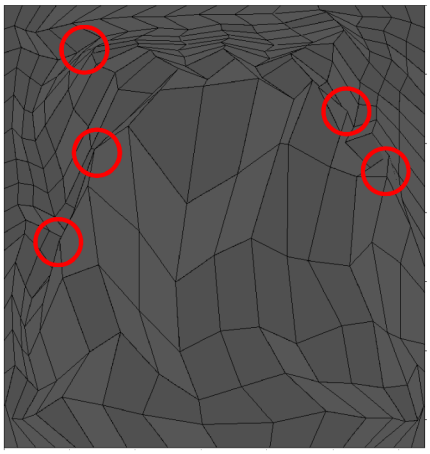


Figure 5.12: Red circles show nodes on the deformed the grid.

Second option is based on *Principle of Minimum Potential Energy*. As mentioned above, the overall error value computed for a particular grid cell can be considered as an "energy" of the cell. The idea of the principle is expressed as:

$$\Pi = \text{Strain energy (U)} - \text{potential energy of loading (W)} \tag{5.3}$$

The strain energy U is caused by displacement in the system whereas energy W is caused by external forces. To be able to solve to solution for the warping grid in the Shadow Mapping algorithm, the strain energy of the grid should be defined. Unfortunately, this requires deep knowledge of the Principle of Minimum Potential Energy in order to adapt it to shadow rendering algorithms introduced in this thesis. This is, however, beyond the scope of this work, and it has not been resolved yet.

CHAPTER 6

Conclusion

The main goal of this work has been improvement of the shadow rendering based on the Shadow Mapping algorithm using Non-orthogonal Texture Warping of the shadow maps. The goal of the work has been achieved and its main contribution is experimental evaluation of the hypothesis that parameterization of shadow map coordinates based on simple scene analysis can reduce aliasing error of the shadows cast by complex light sources.

The experimental results demonstrated that the reducing of aliasing error in shadows could be achieved by modification of projection mapping. This has been evaluated on various use cases. Further details can be found in Chapter 5. Evaluation of the hypothesis included rendering shadows in indoor as well as outdoor scenes with various configurations. The experiments showed that the non-orthogonal warping scheme is applicable to standard Shadow Mapping algorithm and it improved the sampling rate for complex light sources as well.

Results of the work can be applied in various computer graphics applications that rely on quality of shadows in real time. The range of possible applications is from CAD systems e.g. in architecture where shadow rendering is critical for the realistic perception of the buildings to computer games where shadow rendering is nowadays required even in complex scenes and appreciated by the game players as a part of gaming virtual reality.

Future work will be focused on further improvements of robustness and balancing the warping parameters and on better estimation of the error distribution on the shadow maps. Possible other direction of focus would be extensive evaluation of the method on large and complex scenes and measurement of improvement and combination of the warping with other GPU functions. Very interesting direction would be to connect all approaches that employ nonlinear functions with nonlinear rasterization pipeline.

Bibliography

[1] Ulf Assarsson. *A Real-Time Soft Shadow Volume Algorithm.* PhD thesis, Göteborg, Sweden, 2003.

[2] P. Bergeron. A general version of crow's shadow volumes. *Computer Graphics and Applications, IEEE*, 6(9):17–28, Sept 1986.

[3] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *In Proc. of Computer Graphics International*, pages 397–408, 2002.

[4] Michael Bunnell and Fabio Pellacini. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter Shadow Map Antialiasing. Pearson Higher Education, 2004.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 137–145, New York, NY, USA, 1984. ACM.

[6] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, 1977.

[7] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[8] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165. ACM, 2006.

[9] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination.* AK Peters Ltd, 2006.

[10] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 387–390, New York, NY, USA, 2001. ACM.

[11] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[12] Nixiang Jia, Dening Luo, and Yanci Zhang. Distorted shadow mapping. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, pages 209–214, New York, NY, USA, 2013. ACM.

[13] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[14] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56. ACM Press/Addison-Wesley Publishing Co., 1997.

[15] Gary King and William Newhall. Efficient omnidirectional shadow maps. In Wolfgang Engle, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 435–448. Charles River Media, Hingham, MA, 2005.

[16] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings Of Third International Conference On Computational Graphics And Visualization Techniques (compugraphics '93)*, pages 145–153, 1993.

[17] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages 277–286. Eurographics Association, 2007.

[18] Xiao-Hui Liang, Shang Ma, Li-Xia Cen, and Zhuo Yu. Light space cascaded shadow maps algorithm for real time rendering. *Journal of Computer Science and Technology*, 26(1):176–186, 2011.

[19] D. Brandon Lloyd. *Logarithmic Perspective Shadow Maps*. PhD thesis, Chapel Hill, NC, USA, 2007. AAI3289050.

[20] D. Brandon Lloyd, Naga K. Govindaraju, Steven E. Molnar, and Dinesh Manocha. Practical logarithmic rasterization for low-error shadow maps. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '07, pages 17–24, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[21] D. Brandon Lloyd, David Tuft, Sung-eui Yoon, and Dinesh Manocha. Warping and partitioning for low error shadow maps. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, pages 215–226, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[22] D. Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. Cc shadow volumes. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 146–, New York, NY, USA, 2004. ACM.

[23] Daryl L. Logan. *A First Course in the Finite Element Method*. CL Engineering, 5th edition, 2011.

[24] Tomáš Milet, Jozef Kobrtek, Pavel Zemčík, and Jan Pečiva. Fast and robust tessellation-based silhouette shadows. In *WSCG 2014 - Poster papers proceedings*, pages 33–38. University of West Bohemia in Pilsen, 2014.

[25] Tomáš Milet, Jan Navrátil, Adam Herout, and Pavel Zemčík. Improved computation of attenuated light with application in scenes with multiple light sources. In *Proceedings of SCCG 2013*, pages 155–160. Comenius University in Bratislava, 2013.

[26] Tomáš Milet, Jan Navrátil, and Pavel Zemčík. An improved non-orthogonal texture warping for better shadow rendering. In *WSCG 2015 - Full Papers Proceedings*, pages 99–107. Union Agency, 2015.

[27] Jan Navrátil, Pavel Zemčík, Roman Juránek, and Jan Pečiva. A skewed paraboloid cut for better shadow rendering. In *Proceedings of Computer Graphics International 2012*, page 4. Springer Verlag, 2012.

[28] Jan Navrátil, Jozef Kobrtek, and Pavel Zemčík. A survey on methods for omnidirectional shadow rendering. *Journal of WSCG*, 20(2):89–96, 2012.

[29] Brian Osman, Mike Bukowski, and Chris McEvoy. Practical implementation of dual paraboloid shadow maps. In *Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames*, Sandbox '06, pages 103–106, New York, NY, USA, 2006. ACM.

[30] Jan Pečiva, Jaroslav Přibyl, and Jan Navrátil. Close-to-photorealistic lighting for simulations and cad. In *2011 International Simulation Multiconference - SCS (SCSC, SPECTS, GCMS) - SCSC Proceedings (Hard-Copy) 11*, pages 1–2. SCS Publication House, 2011.

[31] Jan Pečiva, Pavel Zemčík, and Jan Navrátil. Mimicking pov-ray photorealistic rendering with accelerated opengl pipeline. In *WSCG'2011 Communication Papers Proceedings*, 19-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, pages 149–156. University of West Bohemia in Pilsen, 2011.

[32] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *SIGGRAPH Comput. Graph.*, 21(4):283–291, August 1987.

[33] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27(5):129:1–129:8, December 2008.

[34] Paul Rosen. Rectilinear texture warping for fast adaptive shadow mapping. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, pages 151–158, New York, NY, USA, 2012. ACM.

[35] Mateu Sbert, Departament D'informtica I Matemtica Aplicada, Departament Llenguatges, and Sistemes Informtics. An integral geometry based method for fast form-factor computation. *Computer Graphics Forum (Eurographics)*, 12:409–420, 1993.

[36] Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Trans. Graph.*, 21(3):557–562, July 2002.

[37] Juraj Vanek, Jan Navrátil, Adam Herout, and Pavel Zemčík. High-quality shadows with improved paraboloid mapping. In *Advances in Visual Computing*, Lecture Notes in Computer Science 6938, pages 421–430. Faculty of Information Technology BUT, 2011.

[38] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

[39] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, August 1978.

[40] Fan Zhang, Hanqiu Sun, Leilei Xu, and Lee Kit Lun. Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*, VRCIA '06, pages 311–318, New York, NY, USA, 2006. ACM.

[41] Kurt Zimmerman and Peter Shirley. A two-pass realistic image synthesis method for complex scenes. In *Eurographics Workshop On Rendering*, pages 284–295, 1995.