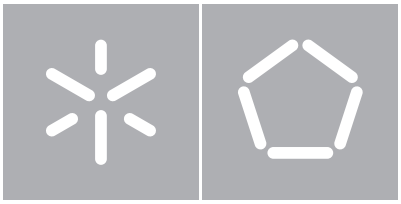
Universidade do Minho
Escola de Engenharia

Fábio da Costa Morais

Rasterization + Ray Tracing: Rendering of Hard Shadows

Outubro de 2015



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Fábio da Costa Morais

Rasterization + Ray Tracing: Rendering of Hard Shadows

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor António José Borba Ramires Fernandes

Outubro de 2015

Acknowledgements

I would like to dedicate this work to my parents for their unconditional support and faith, without it I would have never been able to complete it.

I also would like to express my gratitude to my supervisor, Professor António José Borba Ramires Fernandes, for his contribution and guidance on the research and writing of this document.

Abstract

Due to great technological advances in video cards over the last decade, several classical image-rendering algorithms have recently been adapted to run on GPUs. This made it possible for several ray tracing based global illumination techniques to perform faster and faster, achieving performance levels which are, in some cases, suitable for real-time applications. However, despite these advances, rasterization is still the most widely used technique in the Computer Graphics industry for real time applications due to its efficiency generating images with reasonable visual quality. As the implementation of photorealistic techniques using ray tracing in real-time is still out of reach of today's hardware, there have been several attempts to combine rasterization and ray tracing, to obtain the best of both worlds.

This dissertation seeks to demonstrate the benefits of an approach that combines the efficiency and speed of rasterization, and lighting and visual effects provided by ray tracing. For this purpose, we present an algorithm capable of identifying problematic pixels in shadow map using conservative rasterization techniques, where the triangles are seen larger and smaller than normal. Once these problematic pixels are identified, rays are created for each of these pixels for the ray tracer to correct them. Accordingly two versions of the algorithms have emerged, one that takes into account the adjacency information, explained in more detail in this document, and another in which the adjacency information is ignored, originally developed by Stefan Hertel.

Both versions shown in this study were analysed in terms of image quality, where we determine how many pixels are correct when compared with a method of ray pure tracing, in terms of performance analysing the cost of correcting these problematic pixels using the engine of ray tracing OptiX Prime. Both perspectives are equivalent, only having slight performance differences in the creation of the shadow map or correction of problematic pixels.

Resumo

Devido aos grandes avanços tecnológicos nas placas de vídeo ao longo da última década, vários algoritmos clássicos de renderização de imagens foram recentemente adaptados para correrem em GPU's. Isto tornou possível que várias técnicas de iluminação global, como o ray tracing, serem executadas em tempo real. Mas apesar destes avanços, a rasterização ainda é a técnica mais utilizada na indústria da Computação Gráfica em tempo real devido a sua eficiência no que toca a gerar imagens com qualidade visual razoavelmente boa. Como a implementação de técnicas fotorealista com o uso de ray tracing em tempo real ainda está fora do alcance do hardware de hoje, tem surgido várias tentativas numa forma de combinar a rasterização e ray tracing, de forma a obter o melhor dos dois mundos.

Esta dissertação procura demonstrar os benefícios de uma perspectiva que combina a eficiência e rapidez da rasterização, e a iluminação e os efeitos visuais fornecidos por ray tracing. Para esse efeito, apresentamos um algoritmo capaz de identificar pixéis problemáticos no shadow map com o uso de técnicas de rasterização conservativa, onde os triângulos são vistos maiores e menor que o normal. Uma vez identificados esses pixéis problemáticos, raios são criados para cada um desses pixéis pelo ray tracer para corrigir-lhos. Deste algoritmo surgiram duas versões, uma que toma em consideração a informação de adjacência, explicada em mais detalhe neste documento, e outra onde a informação de adjacência é ignorada, desenvolvida originalmente por Stefan Hertel.

As duas versões mostradas neste trabalho foram analisadas em termos à qualidade de imagem, onde fomos determinar quantos pixéis estão correctos quando comparados com um método de ray tracing puro, em termos de desempenho analisando o custo de corrigir estes pixéis problemáticos usando o motor de ray tracing OptiX Prime. Ambas as perspectivas são equivalentes, apenas apresando diferenças de desempenho na criação do shadow map ou na correcção de pixéis problemáticos.

Contents

1 Introduction 1

1.1 Context 1

1.2 Motivation and Objectives 2

1.3 Outline 3

2 Techniques 4

2.1 Shadow Mapping 4

2.2 Ray Traced Shadows 6

2.3 Conservative Rasterization 8

2.3.1 Implementation 10

3 State of the Art 13

3.1 Trapezoidal Shadow Mapping 13

3.1.1 Increasing Shadow Map Resolution 14

3.1.2 Constructing the Trapezoid 14

3.1.3 Side Lines 15

3.1.4 Focus Region on Shadow Maps 19

3.1.5 Conclusion 20

3.2 Parallel Split Shadow Maps 21

3.2.1 View Frustum Split 21

3.2.2 Logarithmic Split Scheme 23

3.2.3 Uniform Split Scheme 24

3.2.4 Conclusion 25

3.3 Variance Shadow Mapping 25

3.4 Hybrid Methods 26

3.4.1 Hybrid GPU-CPU Renderer 27

3.4.2 Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows 28

4 Conservative Shadow Mapping with Adjacency 30

4.1.1 0-in-Shadow 34

4.1.2 1-in-Shadow 34

4.1.3 2-in-Shadow 36

4.1.4 3-in-Shadow 38

4.1.5 Worst Case Errors 40

4.1.6 Conclusion 41

5 Algorithm Testing 42

5.1 Test Scenes 43

5.2 Ray tracing Scenes 45

5.2.1 Normal Shadow Mapping 49

5.2.2 BGSM Shadow Mapping 54

5.2.3 SGSM Shadow Mapping without Adjacency 60

5.2.4 SGSM Shadow Mapping with Adjacency 66

5.2.5 Conservative Shadow Mapping without Adjacency 71

5.2.6 Conservative Shadow Mapping with Adjacency 77

5.3 Perfomance Testing 83

5.3.1 Best Case 83

5.3.2 Worst Case 89

5.3.3 Average Case 93

6 Conclusions and Future Work 98

7 Bibliography 99

Appendix 104

List of Figures

Figure 1 – A diagram showing the composition of the shadow: the umbra is the interior of the shadow, and the penumbra is the exterior of the shadow. 4

Figure 2 – Shadows provide a valuable clue for the position of the objects. 4

Figure 3 – While distant shadow have great detail, nearby shadows will not have sufficient resolution, resulting in pixelated shadows due to undersampling. 5

Figure 4 – The shadow map resolution for the shadow in the surface is too small, applying the same pixel (red area) of the shadow map to surface seen by the observer (blue area). 6

Figure 5 – Diagram for shadow ray in ray tracing. 6

Figure 6 – A comparison of hard shadows (left) and soft shadows (right). 8

Figure 7 – A comparison of standard (a) and conservative rasterization (b). 8

Figure 8 – (a): overestimated conservative rasterization; (b): underestimated conservative rasterization. 9

Figure 9 – The Optimal Bounding Polygon for a given triangle. 10

Figure 10 – Transformation of the original triangle (orange) to the new triangle (blue); Left: Dilation; Right: Erosion. 11

Figure 11 –Bounding Box Approximation (a) vs. Trapezoidal Approximation (b) 13

Figure 12 - Left: The eye frustum as seen from the light; Middle: example for trapezoidal (top) and bounding box (bottom) approximations; Right: Wastage obtained in the shadow map with approximations. 14

Figure 13 – A 1D homogenous perspective projection problem to compute q. 16

Figure 14 – TSM transformations matrixes: (a) $T1$; (b) R; (c) $T2$; (d) H; (e) $S1$; (f) N; (g) $T3$; (i) The final matrix NT. 19

Figure 15 - For the trapezoid in (a), its corresponding T is shown in (b). In this case, we obtain an over-sampling for a small region of E. (c) For a different trapezoid computed with the 80% rule (having the same top and base lines), its trapezoidal transformation maps the focus region (the upper part of the trapezoid) to within the first 80% in the shadow map. 20

Figure 16 - Split the view frustum into three parts, and shadow maps with the same resolution are generated for the split parts. 21

Figure 17 - Along the z axis the view frustum is split into parts by using the split planes at $\{C\_i \mid 0 \leq i \leq m\}$ 22

Figure 18 - Different types of spilt schemes 23

Figure 19 – Variance Shadow Map light leaking example. 26

Figure 20 – Shadow Demonstration. Left: Pixels that agreed to be shadowed; Middle: Green pixels indicate the marked pixels for the ray-tracer; Right: Shadows computed for the marked pixels by the ray-tracer. 27

Figure 21 – Soft Shadow Demonstration. Left: Soft shadows by the eight shadow maps; Middle: Blue pixels marked for the ray-tracer; Right: Soft shadows computed by the ray-tracer. 28

Figure 22 – Example of the "uncertain" areas (green) when using the Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows. 29

Figure 23 – Geometry Transformations on a pair of triangles that form a plane: (a) Expansion of the triangles in BGSM; (b) Shrinking of triangles in SGSM, without adjacency information; (c) Shrinking of triangles in SGSM, with adjacency information. 31

Figure 24 – Central triangle (orange) and the adjacent triangles (green) provided to the geometry shader 32

Figure 25 – 0-in-shadow case: the original triangle (orange) is shrunk using normal conservative rasterization (blue). 34

Figure 26 - 1-in-shadow case: The new triangle (blue) is composed by $\{b2, a1, v'4\}$. 35

Figure 27 – Three simplified scenarios where the 1-in-Shadow occurs. 36

Figure 28 – 2-in-Shadow case: The new geometry is formed by the triangle $\{b2, a1, b1\}$ and the triangle $\{b1, a0, b2\}$. 37

Figure 29 – Three simplified scenarios where the 2-in-Shadow occurs. 38

Figure 30 – 3-in-Shadow case: The new geometry is formed by the four triangles: $b1, a0, b2$, $\{b2, a1, b1\}$, $b1, a1, a2$ and $\{a2, a1, b0\}$. 39

Figure 31 - Worst Case Scenario: (a) Normal conservative rasterization; (b) Conservative rasterization with adjacency. 40

Figure 32 - The side (left), with (centre) and against (right) viewpoints of the first scene. 43

Figure 33 -The side (left), against (centre) and with (right) viewpoints of the second scene. 44

Figure 34 - The with (left), side (centre) and against (right) viewpoints of the third scene. 45

Figure 35 - Best Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080. 46

Figure 36 - Worst Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080. 47

Figure 37 - Average Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080.Shadow Mapping Errors 48

Figure 38 - Best Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 50

Figure 39 – Worst Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 52

Figure 40 - Average Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 54

Figure 41 - Best Case Result for BGSM; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 56

Figure 42 - Worst Case Result for BGSM; Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 58

Figure 43 - Average Case Result for BGSM; Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 60

Figure 44 - Best Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 62

Figure 45 - Worst Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 64

Figure 46 - Average Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 66

Figure 47 - Best Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096 68

Figure 48 - Worst Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096 69

Figure 49 - Average Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 71

Figure 50 - Best Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 73

Figure 51 - Worst Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 75

Figure 52 - Average Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 77

Figure 53 - Best Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 79

Figure 54 - Worst Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 81

Figure 55 - Average Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096. 83

Figure 56 – Graphical representation of the execution time in the 512x512 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....86

Figure 57 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....87

Figure 58 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....88

Figure 59 – Graphical representation of the execution time in the 512x512 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....91

Figure 60 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....92

Figure 61 - Graphical representation of the execution times in the 1920x1080 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....93

Figure 62 – Graphical representation of the execution time in the 512x512 viewport, showed in tables 29 and 30; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....95

Figure 63 - Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 29 and 30; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....96

Figure 64 - Graphical representation of the execution time in the 1920x1080 viewport, showed in tables 30 and 31; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....97

List of Tables

Table 1 -Information of viewports used for the Bench Scene. 43

Table 2 -Information of viewports used for the Flower Scene. 44

Table 3 -Information of viewports used for the Trees Scene. 45

Table 4 – Optix Prime results for the best case: PFL represents the Pixels Facing the Light and PnFL represents the Pixels not Facing the Light; The PFLs are then split into the pixels in Light and the pixels in Shadow. 46

Table 5 - Optix Prime results for the worst case: PFL represents the Pixels Facing the Light and PnFL represents the pixels; The PFLs are then split into the pixels in Light and the pixels in Shadow. 47

Table 6 - Optix Prime results for the average case: PFL represents the Pixels Facing the Light and PnFL represents the pixels; The PFLs are then split into the pixels in Light and the pixels in Shadow. 48

Table 7 – Normal Shadow Mapping results for the best case; 50

Table 8 - Normal Shadow Mapping results for the worst case; 51

Table 9 - Normal Shadow Mapping results for the average case; 53

Table 10 – BSGM Shadow Mapping results for the best case; 55

Table 11 - BGSM Shadow Mapping results for the worst case; 57

Table 12 - BGSM Shadow Mapping results for the average case; 59

Table 13 – SGSM Shadow Mapping without Adjacency results for the best case; 61

Table 14 - SGSM Shadow Mapping without Adjacency results for the worst case; 63

Table 15 - SGSM Shadow Mapping without Adjacency results for the average case; 65

Table 16 – SGSM Shadow Mapping with Adjacency results for the best case; 67

Table 17 - SGSM Shadow Mapping with Adjacency results for the worst case; 69

Table 18 - SGSM Shadow Mapping with Adjacency results for the average case; 71

Table 19 – Conservative Shadow Mapping without Adjacency results for the best case; 73

Table 20 - Conservative Shadow Mapping without Adjacency results for the worst case; 74

Table 21 - Conservative Shadow Mapping without Adjacency results for the average case; 76

Table 22 - Conservative Shadow Mapping with Adjacency results for the best case; 79

Table 23 - Conservative Shadow Mapping with Adjacency results for the worst case; 80

Table 24 - Conservative Shadow Mapping with Adjacency results for the average case; 82

Table 25 – Execution times of the shadow map rendering for the best case (Side-Trees); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency. 84

Table 26 - Execution times of the ray tracing step for the best case (Side-Trees); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....85

Table 27 - Execution times of the shadow map rendering for the worst case (Against-Flowers); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency. 89

Table 28 – Execution times of the ray tracing step for the worst case (Against-Flowers); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....90

Table 29 - Execution times of the shadow map rendering for the average case (Against-Bench); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency. 94

Table 30 – Execution times of the ray tracing step for the average case (Against-Bench); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.....95

List of Listings

AABB	Axis-Aligned Bounding Box
AO	Ambient Occlusion
HAO	Hybrid Ambient Occlusion
PBRT	Physically Based Ray Tracer
PCF	Percentage Closer Filtering
SSAO	Screen Space Ambient Occlusion
TIR	Total Internal Reflection
VPM	Volume Photon Mapping
FOV	Field Of View
BGSM	Big Geometry Shadow Map
SGSM	Small Geometry Shadow Map
CSMa	Conservative Shadow Mapping without Adjacency
CSMA	Conservative Shadow Mapping with Adjacency

1 Introduction

1.1 Context

Currently, Computer Graphics is divided into several types of image rendering techniques, of which the most prevalent are based on rasterization and ray tracing techniques. These techniques have different philosophies when it comes to image generation.

The rasterization process produces images of reasonable quality with great efficiency, even considering scenarios with a large number of triangles. However, the quality of the images produced by rasterization is not comparable to the quality obtained through algorithms based on ray tracing.

With advances in graphics hardware, ray tracing algorithms have been ported to work on the GPU to speed up image synthesis, as demonstrated by PBRT (Physically Based Ray Tracer) [1], a ray tracing engine with stat-of-the-art technology to generate image for different real-time applications (film, video, gaming, among others). Other engines, especialy designed for real-time ray tracing, have emerged since then, like NVIDIA's Optix™ [2] [3] , AMD's FireRays™ , and Intel's Embree™ .

Despite these advances, the possibility of real-time full ray tracing is still far from current graphical capabilities. In this context proposals that combine rasterization with ray tracing emerged [4]. The main idea is to combine the best of both worlds, the performance of rasterization, and the details that ray tracing is able to provide. Most of the work developed in this field has been mostly about only one aspect like shadows, while some tried to implement all possible ray tracing effects into rasterization using CPU and GPU parallel processing [5, 6].

A famous engine that uses this type of approach is the RenderMan by Pixar, a known engine that uses an implementation of the REYES architecture [7], used for the creation of special effects in the film industry, most known for its role in "*Toy Story*" and "*Finding Nemo*".

For the production of the animation film "*Cars*" [8], the RenderMan was extended to use ray tracing, because the characters in the film required high quality reflections, which was not possible in the rasterization system they had. After this change, Pixar was able to include other effects to film like *ambient occlusion* e *precise shadows*.

After these developments, hybrid algorithms that combined ray tracing and rasterization became a new topic of research. One of the resulting strategies was the use of *deferred rendering*, where the rasterization process gathers the information of the scene and ray tracing uses that information to create visual effects. The rasterization process can have three purposes: to produce an initial image; collect information regarding the scenario (positions and normals) and identify possible areas of intervention for the subsequent steps with ray tracing.

An example of such approach is Stefan Hertel *et al* [9] work, where ray tracing is used to compute the transition between light and shadow in the use of shadow maps. This rendering technique produces high quality images when compared with rasterization, with performance that can be considered acceptable in current graphical technology.

1.2 Motivation and Objectives

Both methods of producing rendered images have their benefits and flaws: rasterization is designed to produce high quality objects, using the proprieties of the object and local approximations of the light, but is incapable of processing visual effects that involve the entire environment, not having easy access to the scenes proprieties, like geometry and placing of the objects, resulting in more complex algorithms to apply little details, that add a new level a detail; ray tracing is designed to produce images with high quality illumination. Due to its implementation based on the studied behaviour of the light in the real world, obtaining visual complex effects like global illumination, reflections and refractions result in simple implementations. However, this comes with a heavy performance impact and high rendering times, since the light effects must take into account the entire scene, and this is not commonly supported directly in hardware graphical pipelines.

The objective of this dissertation is to produce an implementation of a hybrid rendering algorithm, combining the fast and efficient first rendering that rasterization is capable to offer, with the precision that ray tracing offers, hopefully overcoming the individual faults of these methods. This implementation will be analysed from a qualitative and quantitative perspective, comparing with both pure rasterization and ray tracing solutions.

Since the implementation of this dissertation will use the ray tracing engine OptiX™ by NVIDIA, in conjunction with the University of Minho's 3D rendering engine Nau3D, another objective of this dissertation is to add this hybrid approach, testing and improving Nau's support with OptiX™ engine and enhancing the project's capability of rendering high quality 3D environments in real-time.

1.3 Outline

Besides this chapter, this documents will explain the principal concepts used in the shadow mapping and ray tracing, as well as the ideas of conservative rasterization, wich are important for the hybrid method that we'll present.

The second chapter will offers a state of the art of of the most important shadow mapping algorithm that seek to correct the amount of incorrect pixels in the image. Futhermore, it will also present other hybrid method that used rasterization and ray tracing to obtain hard shadows.

The third chapter will present the hybrid method developed in this document, it used a 2 layer shadow map created using the concepts of conservative rastiration to identify problematic areas in a image and sent the information of these areas to the ray tracer to correct them.

The forth chapter will present the testing results of the algorithm. In this chapter will compare to a pure ray tracing and pure shadow mapping algoritmh to analyse the amount of correct and incorrect pixels obtained using our approach, and also analyse the perfomance impact that our hybrid method has compared to normal ray tracing method. Also in this chapter will compared to the work of stefan hertel, another hybrid method that used the same principles as the method presented here.

The fianl chapter will present our conclusions of the algorthim presented here, as well as future work that can be futher added.

2 Techniques

Shadows is the named called to an area not exposed directly to a light source, due to an obstruction of the light by an object, creating a dark silhouette of the obscuring object in the area. Shadow is composed of two components: umbra and penumbra. Umbra is the inner region of the shadow, where the light source is completely blocked. Penumbra is the transition region of the shadow, where the light source is partially obstructed, creating a gradient shadow.

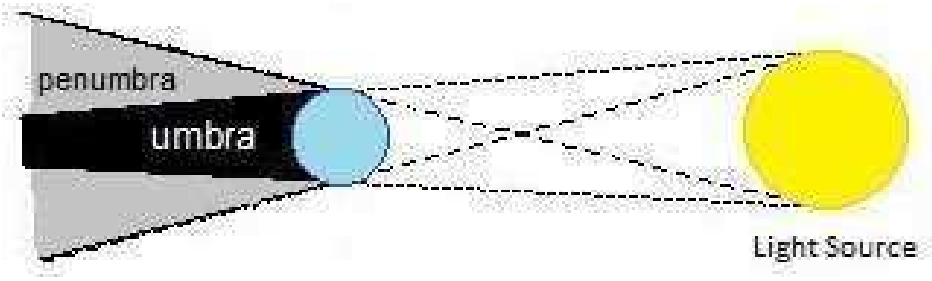


Figure 1 – A diagram showing the composition of the shadow: the umbra is the interior of the shadow, and the penumbra is the exterior of the shadow.

The recreation of this natural phenomenon in rendered images is extremely important, because it establishes a sense of depth, facilitating the determination of spatial relation between objects.

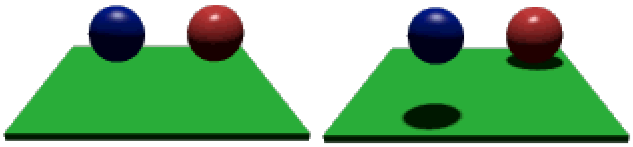


Figure 2 – Shadows provide a valuable clue for the position of the objects.

2.1 Shadow Mapping

The most popular methods in rasterization to generate projected shadow are based on shadow maps [10]. This is a 2-pass algorithm, where the first pass will render the depth values of all the objects visible by the light into a 2D texture (shadow map). These depth values correspond to the distances between to the light source and each visible point by the light, these values being relative to the light point of view. In the second pass of the algorithm, the camera viewpoint is then rendered, where each pixel is transformed to light-space to retrieve the depth value stored in

the shadow map, and compare it to the distance from the corresponding point to the light. If the depth value stored in the shadow map is less than the actual distance to the light, this means that there's another object occluding the point, therefore this pixel is in shadow. Otherwise the pixel is lit.

Since shadows can be determined directly by using the light's point-of-view, the need of auxiliary structures or information about the scene is removed, being an advantage over other methods to create shadows in rasterization. But this technique possesses severe aliasing problems, where shadows lose quality due to subsampling. The most common problems of this technique are Perspective Aliasing and Projection Aliasing (other problems emerge when dealing with large scale environments) [11].

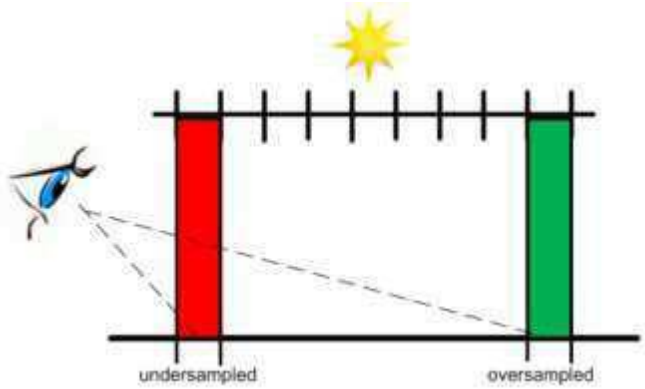


Figure 3 – While distant shadow have great detail, nearby shadows will not have sufficient resolution, resulting in pixelated shadows due to undersampling.

Perspective Aliasing refers to the discrepancies between the perspectives of the eye and the light. In the eye's perspective, the objects closes to the eye are larger than the distant objects, creating inconsistency in the sampling, where the closest shadows are less detailed and the distant shadows get more detailed than required. Figure 3 shows a diagram where the green area represents the oversampling of the shadow and the red area represents the undersampling of the shadow. Projection Aliasing refers to the problem that appears when the surface of an object is almost parallel to the light's direction, making the normal of that surface perpendicular to the light direction. This causes the same depth value of the shadow map to become associated to a large number of the pixels on the surface, resulting in discontinuous. This becomes more apparent when there's camera movement, since the shadows will flicker with each movement.

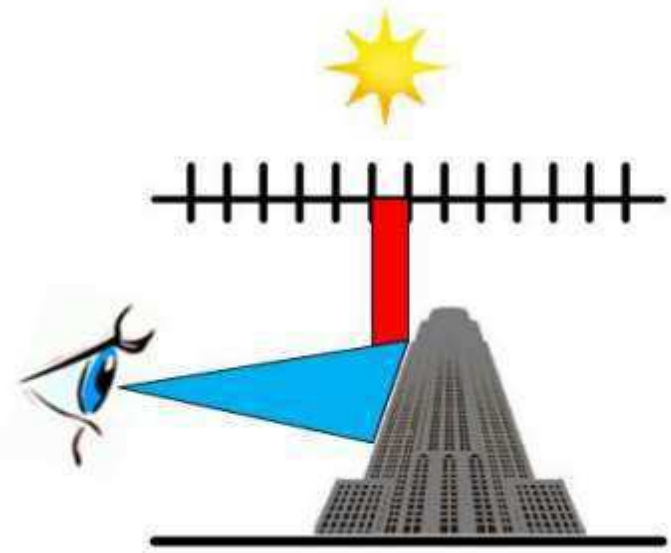


Figure 4 – The shadow map resolution for the shadow in the surface is too small, applying the same pixel (red area) of the shadow map to surface seen by the observer (blue area).

These methods will be detailed in the next chapter.

2.2 Ray Traced Shadows

Considering point lights, shadows are easily created using ray tracing based algorithms. For every pixel in the scene, a ray is sent from its world position to the light source.

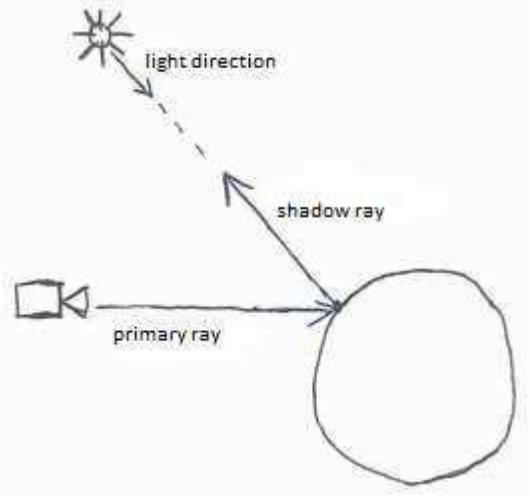


Figure 5 – Diagram for shadow ray in ray tracing.

When a ray from the camera (primary ray) intersects an object, a new ray is created in the intersected point (shadow ray). This ray is sent in the direction of the light source. If the shadow ray intersects any object during its trajectory, then the intersected point is in shadow, if not the intersected point is being illuminated.

This process of creating shadows is simple to implement, gaining projected shadows easily and with great detail.

When considering area light sources, this implies that a single ray sampling the light source is not enough to determine the shadow status. Not only the result would not be correct, but the penumbra effect is not really observable in those conditions.

The reproduction of shadows with area lights needs to determine the percentage of the light source visible for each point in the scenery. That task can be approximated by using multiples samples of the light. This requires several shadow rays in different directions within the light source. The percentage of rays that reach the light give an approximation of the area of visible light. This allows the creation of penumbra effects.

In general, larger number of samples provides better results in the production of penumbra and umbra regions, however, as a direct consequence, the render time of the image will be larger.

The importance of the sampling techniques in ray tracing is because the calculations used determine the exact direction of the shadow ray for each intersected point, which can lead to aliasing problems, where the penumbra edges appear sharp-edged, giving an surreal effect to the shadow. The sampling process allow for multiple shadow rays for each intersected point, blurring the penumbra edge, making it more natural for the observer.

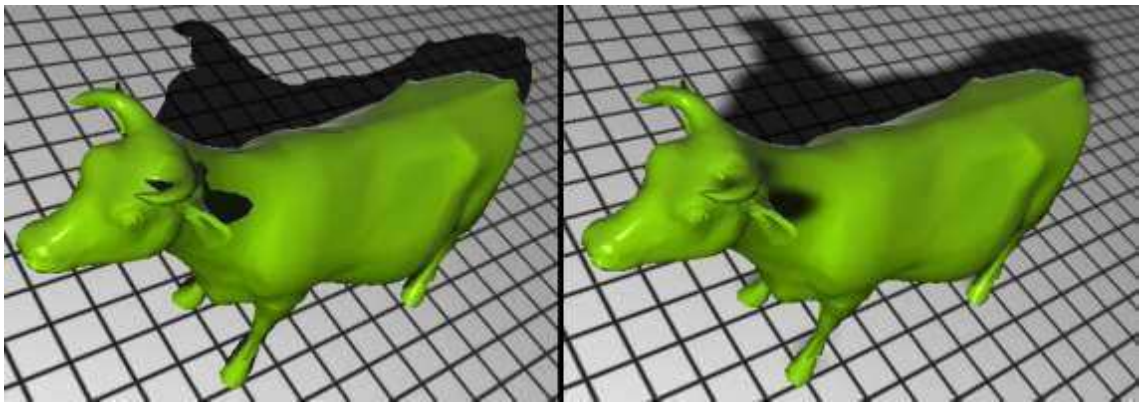


Figure 6 – A comparison of hard shadows (left) and soft shadows (right).

2.3 Conservative Rasterization

Many algorithms of collision detection, occlusion culling and visibility testing for shadow acceleration present the problem of visual artefacts, due to the way rasterization is done in GPU's. This problem is usually fix by increasing the rendering resolution, but the majority of cases, this increase of resolution only reduces the amount of artefacts in the images, because the sample size is larger, but the errors still exist in the image. Another drawback of this solution is the increase of the performance of the algorithms.

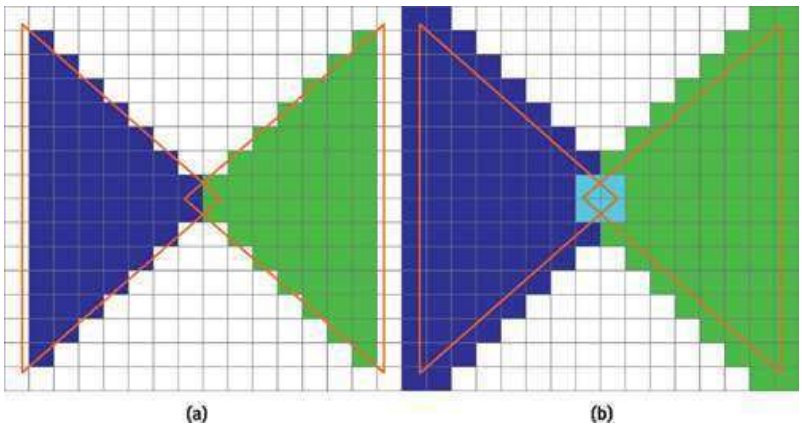


Figure 7 – A comparison of standard (a) and conservative rasterization (b).

The methods presented in Jon Hasselgren show a different approach to sampling artefacts in standard rasterization. The solution presented in his works consists in modifying the polygons before the rasterization process, where a pixel cell (the rectangular region around a pixel in the

pixel gird) moves along the border, applying one of the following transformations, depending on the approach taken to the rasterization process:

- An overestimated conservative rasterization (Figure 8-a), all the pixels caught in the pixel cell are added, dilating the edge.

- An underestimated conservative rasterization (Figure 8-b), all the pixels caught in the pixel cell are erased, eroding the edge.

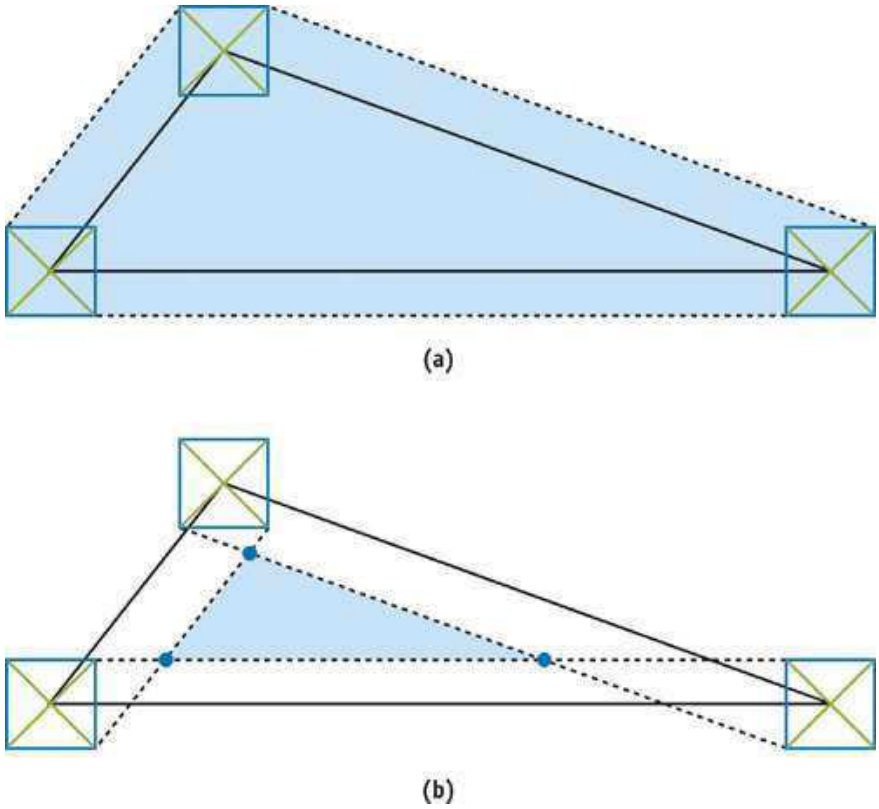


Figure 8 – (a): overestimated conservative rasterization; (b): underestimated conservative rasterization.

Although Jon Hasselgren presents two implementations of the algorithm; this work will focus more on the second implementation, since the first implementation requires multiple output vertices for each input vertex to calculate the optimal bounding box, the second implementation obtains the bounding polygon by intersecting the bounding triangle with an AABB, as showed in Figure 9.

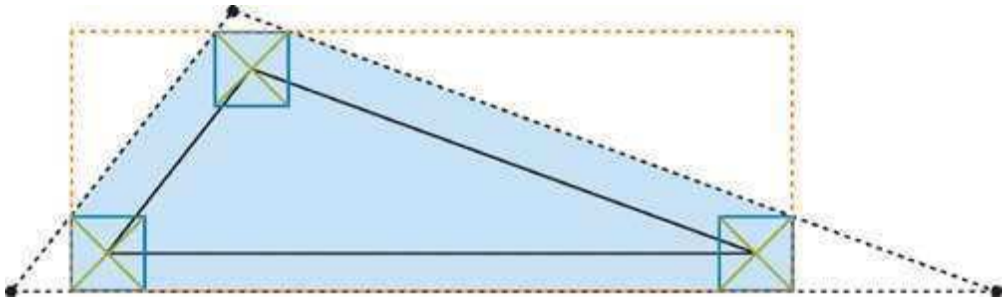


Figure 9 – The Optimal Bounding Polygon for a given triangle.

In order to dilate/erode the edges of a triangle, the vertex information must be available, so the implementation requires a geometry shader. The calculations shown below are done in screen space. This is to reduce the complexity of the calculations to dilate/erode, since the x and y coordinates represent the position of the triangle in the screen, and the z coordinate will indicate if the triangle is visible in the screen.

2.3.1 Implementation

Given a triangle composed of the vertexes v0, v2 and v4, the first step of the algorithm is to calculate the edges of the original triangle in screen space.

$$\begin{aligned} e0 &= v2 - v0 \\ e1 &= v4 - v2 \\ e2 &= v0 - v4 \end{aligned} \tag{1}$$

After all of the edges are calculated, we must calculate the perpendicular vector for each edge (peX), as if they were planes.

$$pe\mathrm{N} = |(-eN.x), (eN.y)| \tag{2}$$

These perpendicular edges represents the direction in which the triangle will be eroded or dilated from, depending of the approach, as showed in the diagrams of Figure 7, the orange triangle represents the original triangle and the blue triangle represents the modified triangle.

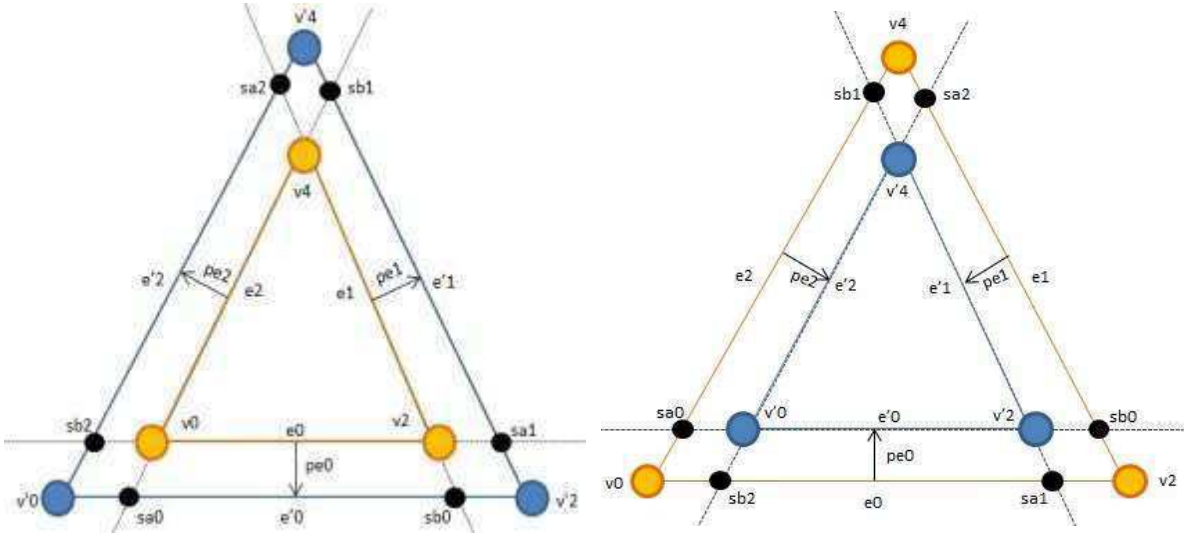


Figure 10 – Transformation of the original triangle (orange) to the new triangle (blue); Left: Dilation; Right: Erosion.

But the calculations done now only apply to the edges, in the shader the transformation apply to the vertex, and since each vertex belongs to two edges, we must calculate the shifted positions of each vertex following the direction of pedge that that vertex belongs to, the following equations show an example for vertex v0, but these calculations apply for all vertexes.

$$\begin{aligned} sa0 &= \mathrm{vec3}(v0) \pm pe0 * pD \\ sb2 &= \mathrm{vec3}(v0) \pm pe2 * pD \end{aligned} \tag{3}$$

Where pD represents diagonal length of the pixel cell. The sa0 is the shifted position of vertex v0 following the direction pe0 and sb2 is the shifted position of v0 following the direction pe2. The $\pm$ means that functions changes depending on the transformation, for erosion the $+$ sign is used and for dilation the $-$ sign is used.

After calculating all shifted positions of the vertex, we can calculate the intersections of the new edges to form the new triangle form by the vertexes v'0, v'2 and v'4. Here the formula for v'0.

$$\begin{gathered} aN = (sbN.y - saN.y) \\ bN = (saN.x - sbN.x) \\ c0 = \ a0 * sa0.x + b0 * sa0.y \\ c2 = a2 * sa2.x + b2 * sa2.y \\ delta = a0 * b2 - a2 * b0 \end{gathered} \tag{4}$$

$$v'0.xy = \frac{(b2 * c0 \ - \ b0 * c2\ , a0 * c2 - a2 * c0)}{delta}$$

After this step, we have obtained the positions of the new vertexes v'0, v'2 and v'4 in screen space, but z component for all the vertexes. The z component of v'x can be easily calculated using the plane equation as follows.

$$Ax + By + Cz + D = 0 \Rightarrow z = \ -\frac{(Ax + By + D)}{C} \tag{5}$$

Therefore, the z coordinate for the v'0 can be calculated using the normal of the original vertex (n) in the next equation:

$$\begin{gathered} D = \ -dot(n, v0) \\ v'0.z = \ -\frac{(n.x * v'0.x + n.y * v'0.y + D)}{n.z} \end{gathered} \tag{6}$$

There might be cases where vertex, both the original and the transformed, are not visible in screen space. This is because the triangle is perpendicular to the camera. Therefore, if the normal of the normal of any vertex is 0, this process is skipped.

3 State of the Art

3.1 Trapezoidal Shadow Mapping

Trapezoidal Shadow Mapping is a new approach of calculating shadows using trapezoidal shadow maps which are derived from trapezoidal approximations of the eye's frustum as seen from the light. It addresses the resolution problems of the standard shadow mapping, resulting in enhanced shadow map resolution for both static and dynamic objects from near and far, with no constraint on the relative positions and motions of the eye and the light. The approach is efficient as only the eight corners of the eye's frustum plus the centres of the near and far plane, rather the scene, are needed to compute a good trapezoidal approximation, thus it scales well to large scenes. Figure 11 shows an example of this approach.

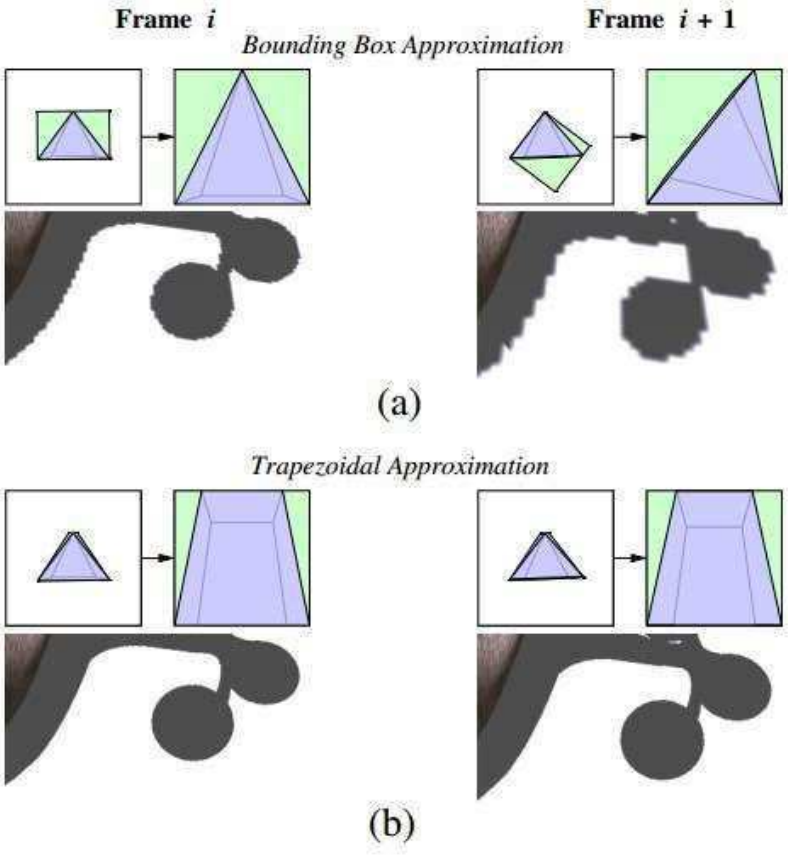


Figure 11 –Bounding Box Approximation (a) vs. Trapezoidal Approximation (b)

3.1.1 Increasing Shadow Map Resolution

A shadow map can viewed as a simple construct that contains two types of positions of the eye's frustum: the positions within and the positions outside the frustum. It is clear that only the former is useful to determine whether pixels are in shadow or not. Increasing the shadow map resolution is to minimize the positions outside the frustum, which are collectively termed as wastage. In other words, a good way to address the resolution problem is to better utilize the shadow map for the area within the eye frustum in Figure 12 (denoted as E). This requires the calculation of an additional normalization matrix N to transform the post-perspective space of the light to an N-space in general, where N refers to the trapezoidal space and bounding box space, respectively in Figure 12. The shadow map is constructed by transforming the pixel into the N-space, rather than into the post-perspective space of the light, for depth comparison.

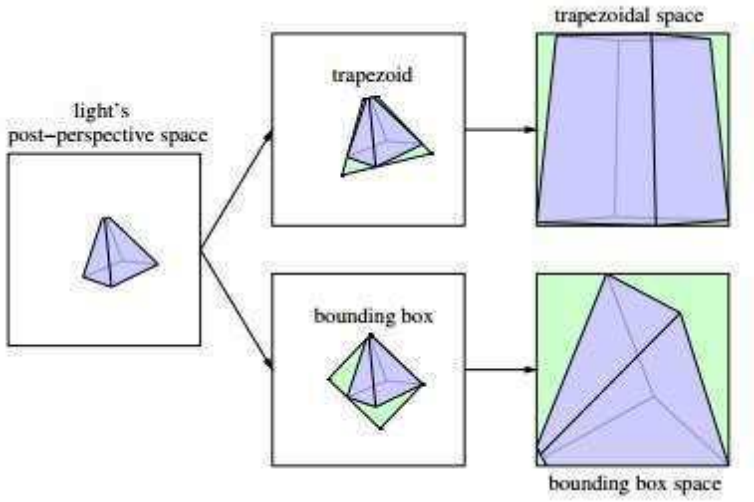


Figure 12 - Left: The eye frustum as seen from the light; Middle: example for trapezoidal (top) and bounding box (bottom) approximations; Right: Wastage obtained in the shadow map with approximations.

3.1.2 Constructing the Trapezoid

Since the trapezoid is recognized to be most similar shape to E, the aim of Martin *et al* works is to construct a trapezoid T to approximate E, with the constraint that each such consecutive approximation results in a smooth transition of the shadow map resolution.

The first step is to find two parallel lines in post-perspective space (L) of the light to contain the base and the top edge of the required trapezoid (T). The aim is to choose the parallel lines such that there is a smooth transition when the eye moves between frames. The algorithm to determine those lines follows these steps:

1. Transform the eye's frustum into L of the light to obtain E;
2. Compute the central line l, which passes through the centres of the near and far plane of E;
3. Calculate the 2D convex hull of E (which can contain up to six vertices on its boundary);
4. Calculate the top line (l_t) that is orthogonal to l and touches the boundary of the convex hull of E. It intersects l at a point closer to the center of the near plane and far plane of E.
5. Calculate the base line (l_b) which is parallel to (and different from) the top line (l_t) (i.e., orthogonal to l too) and touches the boundary of the convex hull of E.

The calculation of the centre line l is important because it allows recalculating the l_t and l_b accordingly to the eye's movement, creating a smooth transition between frames.

3.1.3 Side Lines

The following step is to calculate the side lines of the trapezoid. Assuming the eye is more interested in objects and their shadows within the first δ distance from the near plane. That is, the focus region of the eye is the eye's frustum truncated at δ distance from the near plane. Let p be a point of δ distance away from the near plane with its corresponding point p_L lying on l in L, as seen in Figure 13. Let the distance of p_L from the top line be δ'. The trapezoid has to contain E, so that N_t maps p_L to some point in T.

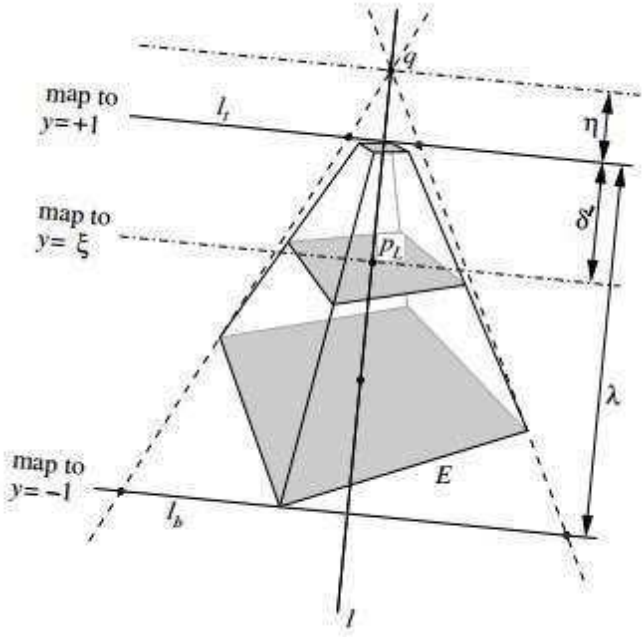


Figure 13 – A 1D homogenous perspective projection problem to compute q.

To do this, we must calculate a transformation matrix N_T which maps the four corners of the trapezoid (t_0,t_1,t_2 and t_3) to the front of the unit cube. This can be done with the following constraints:

$$\begin{aligned}(-1,-1,1,1)^T &= N_T * t_0\\ (+1,-1,1,1)^T &= N_T * t_1\\ (+1,+1,1,1)^T &= N_T * t_2\\ (-1,+1,1,1)^T &= N_T * t_3\end{aligned} \tag{7}$$

A straightforward way to determine n_t with these restrictions is to apply rotation, translation, shearing, scaling, and normalization operations to the trapezoid to map it to the front side of the unit cube. This is achieved by calculating the eight matrices: T_1, R, T_2, H, S_1, N, T_3 and S_2.

As the first step, T_1 transforms the centre of the top edge to the origin (Figure 14-(a)).

$$u = \frac{(t_2 + t_3)}{2} \tag{8}$$

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & -x_u \\ 0 & 1 & 0 & -y_u \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then, the trapezoid T is rotated by applying R around the origin in such a way, that the top edge is collinear with the x-axis (Figure 14-(b)):

$$u = \frac{(t_2 - t_3)}{|t_2 - t_3|}$$

$$R = \begin{vmatrix} x_u & y_u & 0 & 0 \\ y_u & -x_u & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (9)$$

After the rotation, the intersection i of the tow side lines containing the two side edges (t_0, t_3) and (t_1, t_2) is transformed, by applying T_2, to the origin (Figure 14-(c)):

$$u = R * T_1 * i$$

$$T_2 = \begin{vmatrix} 1 & 0 & 0 & -x_u \\ 0 & 1 & 0 & -y_u \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (10)$$

As a next step, the trapezoid has to be sheared with H, so that it is symmetrical to the y-axis (Figure 14-(d)), i.e. that the line passing through the centre of the bottom edge and centre of the top edge is collinear with the y-axis:u

$$= \frac{T_2 * R * T_1 * (t_2 + t_3)}{2}$$

$$H = \begin{vmatrix} 1 & -x_u/y_u & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad (11)$$

Now, the trapezoid is scaled by applying S_1(Figure 14-(e)), so that the angle between the two side lines containing the two side edges (t_0, t_3) and (t_1, t_2) is 90 degrees, and so that the distance between the top edge and the x-axis is 1:

$$u = H * T_2 * R * T_1 * t_2$$

$$S_1 = \begin{vmatrix} 1/x_u & 0 & 0 & 0 \\ 0 & 1/y_u & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{12}$$

The following transformation N (Figure 14-(f)) transforms the trapezoid to a rectangle:

$$N = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix} \tag{13}$$

Then, the rectangle is translated along the y-axis until its centre is coincident with the origin. This is done by applying T_3(Figure 14-(g)). After this transformation the rectangle is symmetrical to the x-axis as well:

$$u = N * S_1 * H * T_2 * R * T_1 * t_0$$
$$v = N * S_1 * H * T_2 * R * T_1 * t_2$$

$$T_3 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\left(\frac{y_u}{w_u} + \frac{y_v}{w_V}\right)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{14}$$

As a last step the rectangle has to be scaled with S_2 (Figure 14-(h)) along the y-axis so that it covers the front side of the unit cube:

$$u = T_3 * N * S_1 * H * T_2 * R * T_1 * t\_0$$

$$S_2 = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -w_u/y_u & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{15}$$

Now the trapezoidal transformation N_T (Figure 14-(i)) can be computed as follows:

$$N_T = S_2 * T_3 * N * S_1 * H * T_2 * R * T_1 \tag{16}$$

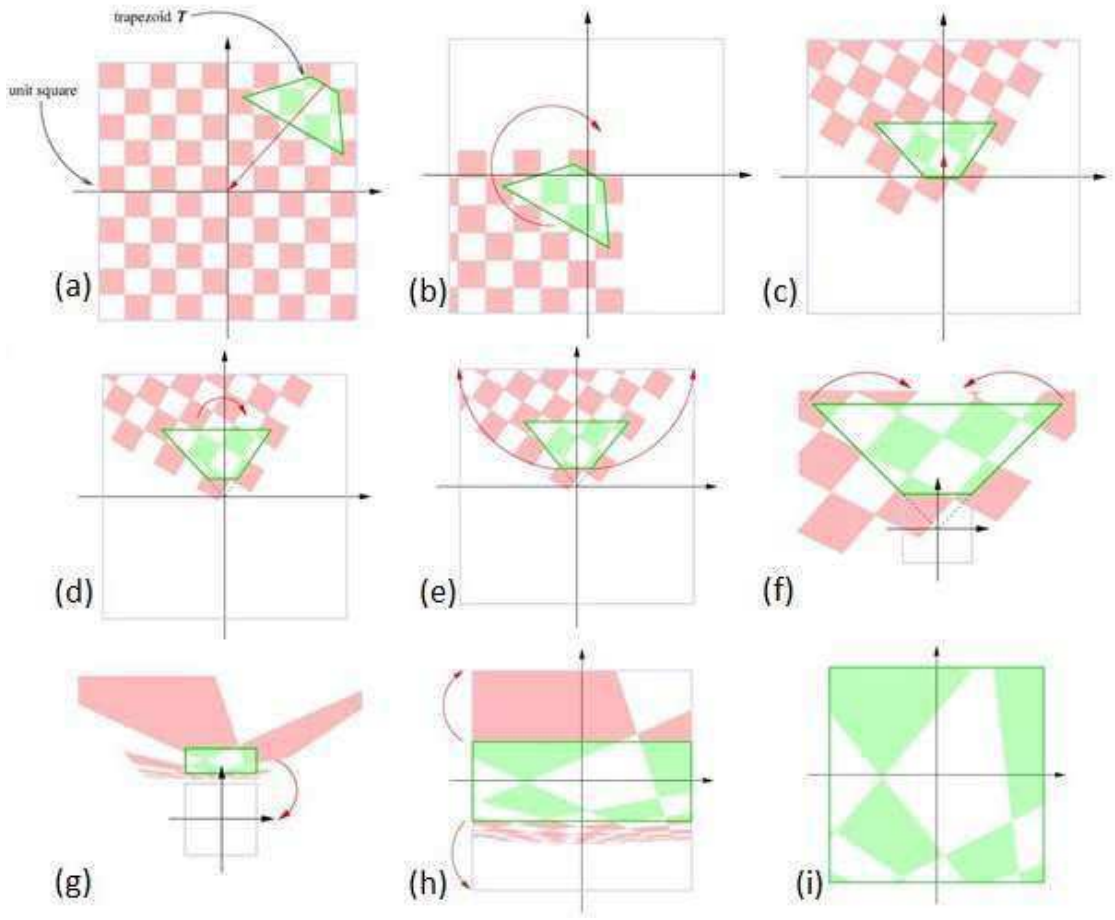


Figure 14 – TSM transformations matrixes: (a) T_1; (b) R; (c) T_2; (d) H; (e) S_1; (f) N; (g) T_3; (i) The final matrix N_T.

3.1.4 Focus Region on Shadow Maps

Figure 4 demonstrates one of the problems of the trapezoidal transformation N_t to T, given a trapezoid containing four triangles, like shown in (a). N_t has the effect of stretching the top edge of the into a unit length. In this case, the top edge is relatively short compared to the base edge, and therefore the stretching results in pushing all the showed triangles towards the bottom of the unit square as showed in (b). This means that the region near the top edge (i.e., close to the near plane) eventually occupies a major part of the shadow map, which results in over-sampling of objects near the eye, sacrificing resolution of the other objects (such as the second to fourth triangles from the top show).

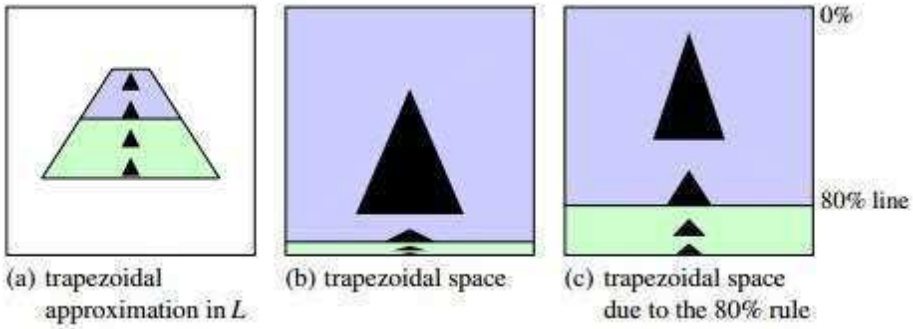


Figure 15 - For the trapezoid in (a), its corresponding T is shown in (b). In this case, we obtain an over-sampling for a small region of E. (c) For a different trapezoid computed with the 80% rule (having the same top and base lines), its trapezoidal transformation maps the focus region (the upper part of the trapezoid) to within the first 80% in the shadow map.

The 80% rule allows to for all the p_L with the distance δ from the top line, when applied the trapezoidal transformation N_t, to be mapped to some point within the 80% line in T. This increases the resolution of the far objects, while maintain a high detail for objects near the eye.

3.1.5 Conclusion

Trapezoidal Shadow Maps shows that it is practical and maps well to graphics hardware. It is a reasonable heuristic to generate shadow maps of good resolution, but the issues on over-sampling and under-sampling remain for various situations such as in the duelling frusta case where the trapezoidal approximation does not have any particular advantage over other approximations.

3.2 Parallel Split Shadow Maps

Parallel split shadow maps is a method to reduce the amount of aliasing error in shadow maps by producing an optimized distribution of shadow map texels. In large-scale environments the shadow map might not have enough resolution in order to create detailed shadows in the scene. PSSM splits the shadow maps in continuous discrete layers, in order to maintain the detail despite the distance of camera.

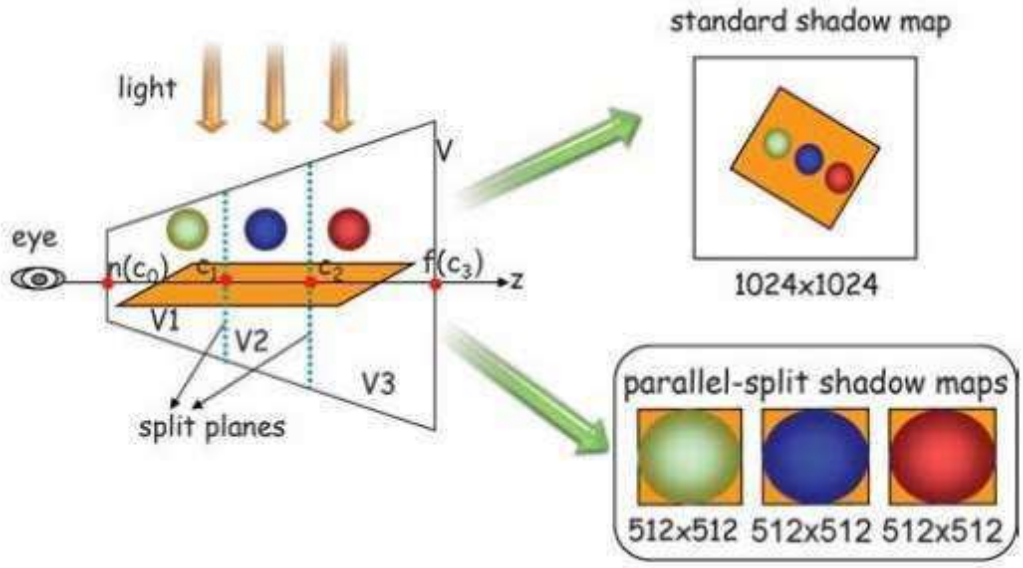


Figure 16 - Split the view frustum into three parts, and shadow maps with the same resolution are generated for the split parts.

The processing steps of the PSSMs scheme are outlined in the following:

1. Split the view frustum into multiple depth parts;
2. Split the light's frustum into multiple smaller ones, each of which covers one split part also the objects potentially casting shadows into the part;
3. Render a shadow map for each split part;
4. Render scene shadows for whole scene.

3.2.1 View Frustum Split

The "splitting" is basically dividing the view frustum in to smaller continuous planes, to which a shadow map will be attributed to each of those planes. These planes are continuous because the

"split" occurs according certain intervals along the z axis. Figure 16 shows an example of m split planes along the z axis.

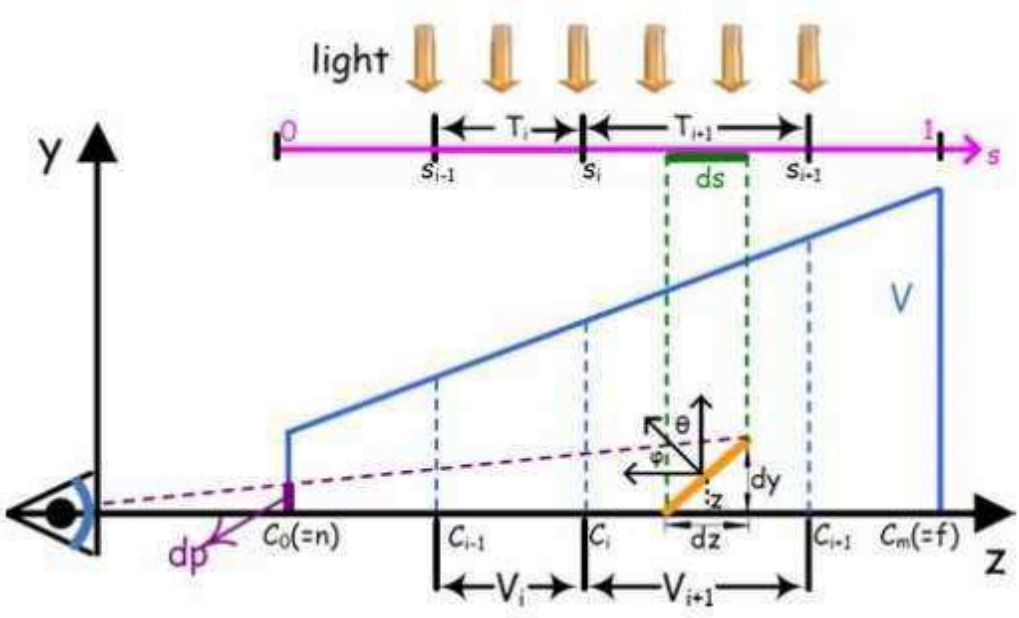


Figure 17 - Along the z axis the view frustum is split into parts by using the split planes at $\{C\_i \mid 0 \le i \le m\}$

In Figure 17, the light beams through a texel with the size $ds \times ds$ in normalized texture space (i.e $s \in [0,1]$) falls on a surface with the length dz in world space. The size of the view beams dp on the screen projected form the surface is approximately ndy/z. φ and θ denote the angles between the surface normal and vector to the screen and the shadow map plane respectively.

Shadow map under-sampling occurs when dp is larger than the pixel size of the screen, this can happen when perspective aliasing (dz/zds) or projection aliasing ($\cos\varphi / \cos\theta$) becomes large. Projection aliasing usually happens when the light's direction is almost parallel to the surface. Since projections aliasing is heavily related to the scene's geometry details, the only solution to the problem is to increase the sampling density, inevitably increasing the scene analysis time. On the other hand, perspective aliasing comes from the perspective foreshortening effect, which can be reduced by applying a global transformation to warp the shadow map texels.

Zhang et al works present three split schemes in order to deal with different distribution of perspective aliasing errors:

- Logarithmic split scheme;
- Uniform split scheme;
- Practical split scheme.

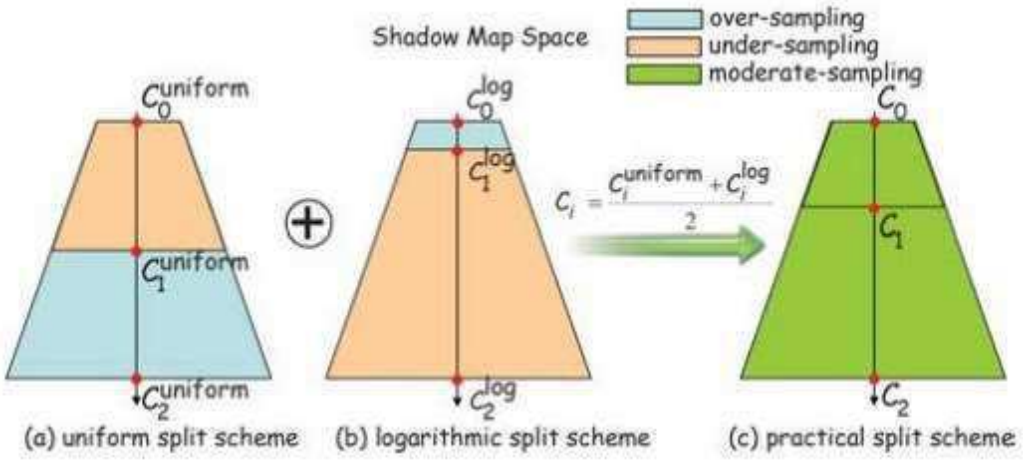


Figure 18 - Different types of spilt schemes

3.2.2 Logarithmic Split Scheme

Logarithmic split scheme is based on optimal distribution of perspective errors, which theoretically evens distribution of perspective aliasing errors over the whole depth range. This scheme assumes that the shadow map accurately covers the view frustum and no piece of the resolution is wasted on invisible parts of the scene.

$$C_i^{log} = n(f/n)^{i/m} \tag{17}$$

The main drawback of this split scheme is that the lengths of split parts near the viewer are too small, so few objects can be included in these split parts. This is because of the main assumption that the shadow map covers the view frustum, which requires that every $z \in [n, f]$ must be mapped to a unique $s \in [0,1]$ in the normalized texture space, but this can't be satisfied in

practice, resulting in under-sampling for parts further from the viewer and over-sampling for parts nearer the viewer.

3.2.3 Uniform Split Scheme

The simplest split scheme is to place the split planes uniformly along the z axis:

$$C_i^{uniform} = n + (f - n)i/m \tag{18}$$

This split scheme results in under-sampling at the points near the view and over-sampling in points further from the view. In contrast to the logarithmic split scheme, uniform split scheme results in the theoretically worst aliasing distribution.

Practical Split Scheme

Practical split scheme consists in combining the previous two split schemes to produce a moderate result between the theoretically optimal and worst sampling densities in the extreme cases, the objects near the viewer and the objects further form the viewer.

In Figure 18, practical split scheme produces moderate sampling for both near and far split parts. Combining equations 17 and 18, the split positions C_i can be calculated by the following equation

$$C_i = \frac{\left(C_i^{log} + C_i^{uniform}\right)}{2} + \delta_{bias}, \forall 0 \leq i \leq m \tag{19}$$

The variable δ_{bias} is a non-negative bias that can be used to accurately adjust clip positions, if necessary for the application.

PSSMs and Scene Rendering

After the lights frustum is split into W_i, each split part V_i is rendered to a shadow map T_i in the W_i space. These shadow maps can have a fixed size, which is helpful for the uniform and practical splits.

With the PSSMs generated in previous step, shadows shadow effects can be now synthesized into the virtual scene. Like standard shadow mapping, each pixel should be transformed into the light space when determining if the pixel is shadowed or not. The differences here are:

1. The correct shadow map T_i must be selected;
2. The pixel has to be transformed into W_i rather W.

After this, for each rasterized fragment, the sampling must come from the appropriate shadow map based on the depth value of the fragment. Since the coordinates are measured in clip space, C_i is transformed to C_i^{clip} below:

$$C_i^{clip} = \frac{f}{f-n}\left(n - \frac{1}{C_i}\right) \in [0,1] \tag{20}$$

Then, the pixel in the fragment buffer with the depth value z^{clip} , if the $z^{clip} \in [C_{index-1}^{clip}, C_{index}^{clip}]$ the shadow map T_{index} is selected. Consequently, this fragment is transformed into the split light's frustum W_{index} for the depth comparison.

3.2.4 Conclusion

While parallel-split shadow maps is an intuitive and simple implementation of shadow rendering, and using the practical split scheme, the quality of the shadows becomes superior to standard shadow map, without complicated scene analysis. However, without using hardware acceleration, the performance drop caused by multiple rendering passes.

3.3 Variance Shadow Mapping

Variance Shadow Mapping is a technique that calculates, besides the usual depth value, the depth-squared values. These values will then be used to calculate the probability of each point being lit or not. But due to the fact that the lower bound of brightness is an approximate value derived from using only one single occluding object , if a scene has a high depth complexity, there might be light leaking artefacts (areas appearing lit instead of shadowed).

Figure 19 – Variance Shadow Map light leaking example.

3.4 Hybrid Methods

There are been developments to combine ray traced shadow effects with rasterization techniques [17], the most common methods use shadow mapping, due to its very efficient algorithm performance, to perform a first pass of the scene, indicating the areas in shadow with the shadow map, followed by a selective ray tracing algorithm to determine the shadow of the pixel that have a unreliable shadow status. This greatly reduces the number of ray being cast and intersection calculations, going better results when compared to shadow maps initial results, with reduced performance times when compared to pure ray tracing solution.

3.4.1 Hybrid GPU-CPU Renderer

The Hybrid GPU-CPU Renderer [18] presented an algorithm to render shadows by mixing shadow mapping and ray tracing. The algorithm creates a shadow map with a bilinear PCF. Then for each pixel in the interpolated result and if the result is 0 or 1 then the four surrounding will agree and the pixel will be lit (1) or shadowed (0). If the result is between 0 and 1, then the four surrounding pixels aren't in agreement the state of the pixel, so the pixel is marked. The ray-tracer will then calculate the shadowing for each uncertain pixel marked in the shadow map.

This algorithm is also adaptable to the type of light source used in the scenery. If the light source is a point light, the method describe before is used. But if the light source is an *area light*, the light source will be replaced by eight point lights, one in the centre of the area and the other seven will surround this area. In this case, the agreement will be done using the eight shadow maps.

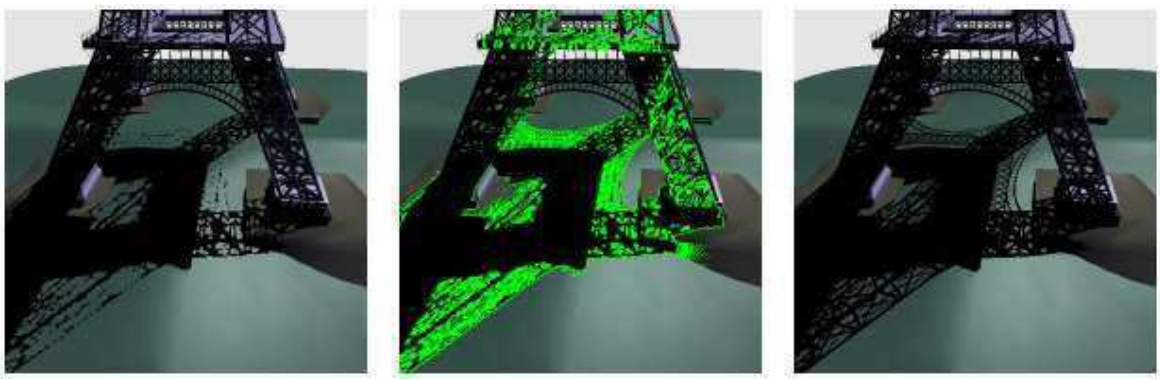


Figure 20 – Shadow Demonstration. Left: Pixels that agreed to be shadowed; Middle: Green pixels indicate the marked pixels for the ray-tracer; Right: Shadows computed for the marked pixels by the ray-tracer.

This algorithm is robust if the shadow map resolution is adequate to the tessellation of the scene, but this is difficult because the scene has to be carefully model using a constant tessellation parameter for all objects. Errors may still occur if the four pixels are in agreement are incorrect, with four pixels indicated that a point is lit, yet the point should be in shadow.

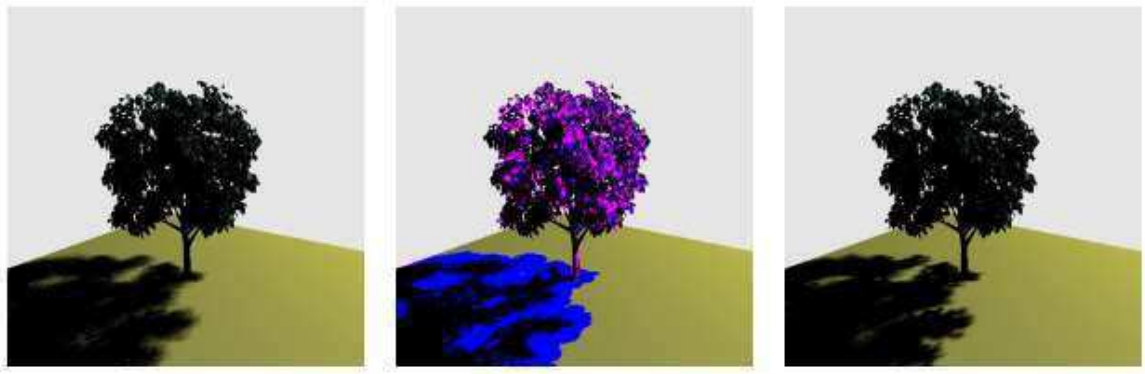


Figure 21 – Soft Shadow Demonstration. Left: Soft shadows by the eight shadow maps; Middle: Blue pixels marked for the ray-tracer; Right: Soft shadows computed by the ray-tracer.

3.4.2 Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows

The Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows is used to create alias-free shadows. It creates a conservative shadow map (CSM) that similarly to the normal shadow map, but in this case a triangle will be saved in a pixel if it overlaps said pixel in any plane, not only in the centre. This is done as shown in Chapter 2.

The CSM consists of a 2 layer texture where one layer contains all the expanded triangles in the scene (BGSM), and the other layer contains all of the shrunken triangles in the scene (SGSM). In the scene rendering step, the pixel will be analysed using both layers in the CSM. If both layers are in agreement with the stat of the pixel, the pixel will be light or shadowed in the final image. If the layers disagree on the state of the pixel, the pixel will be classified as "uncertain". Afterwards, for each "uncertain" pixel discover is sent to a ray tracer to verify the state of the pixel. This ray-tracer will use the information of the triangle saved by the pixel and a kD-tree in order to speed up intersection tests. The information of the depth at which the triangle is found will allow for the ray-tracer to only start testing for intersections from there, as there should be no other triangle between this point and the light source.

As can be seen in Figure 22, there are many areas classified as uncertain that commonly produce correct results using shadow maps, namely the triangle junctions for triangles in light. The performance of this algorithm is highly dependent on the geometry tessellation hence for highly tessellated models a large number of light rays will be required.

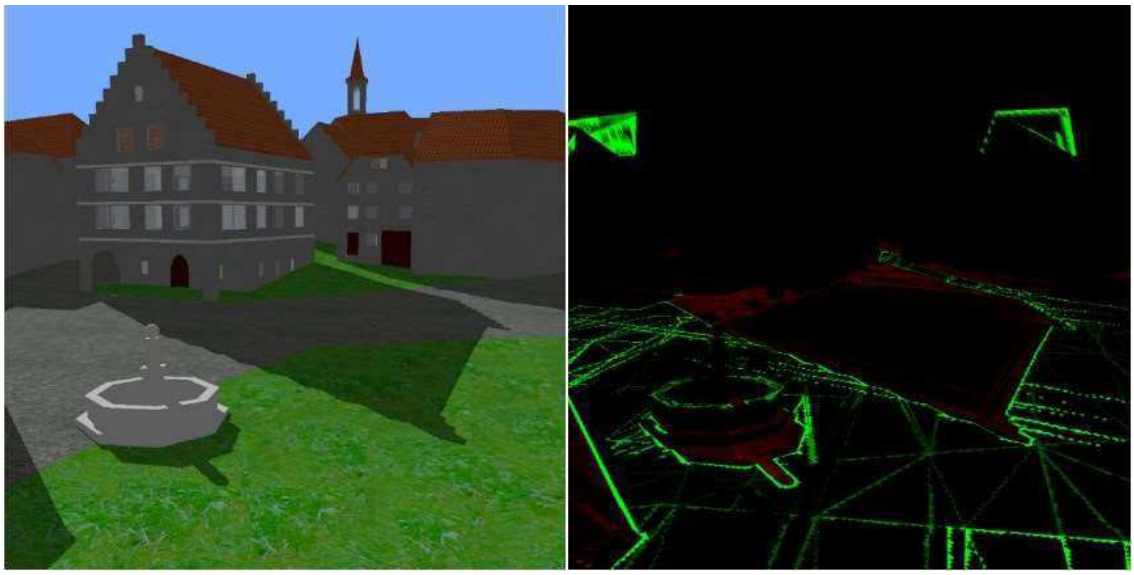


Figure 22 – Example of the "uncertain" areas (green) when using the Hybrid GPU Rendering Pipeline for Alias-Free Hard Shadows.

4 Conservative Shadow Mapping with Adjacency

In Hertel's work, the shadow map is created with 2 layers. One layer views the triangles of the scene larger than normal, therefore each triangle is expanded a certain size λ, which results in the big geometry shadow map (BGSM). The other layer of the shadow map views the triangles in the scene smaller than normal, therefore each triangle is shrunk a certain size λ, which results in the small geometry shadow map (SGSM).

The size λ can be calculated by the following equation:

$$\lambda = \sqrt{\left(\frac{\eta}{l_{sm}}\right)^2 + \left(\frac{\eta}{h_{sm}}\right)^2} \tag{21}$$

Where l_{sm} and h_{sm} are the length and height of the shadow map layer, and η is a user defined variable that indicates the amount of pixels each triangle will expand/shrink.

When the shadows are rendered in to the scene, the shades will look to the corresponding texel in both layers and reach one of the following conclusions.

- If both texels agree that the pixel is in light ($t = 1$), the pixel in the scene will be in light;
- If both texels agree that the pixel is in shadow ($t = 0$), the pixels in the scene will be in shadow;
- If both texels disagree on the state of the pixel, the pixel is classified as "uncertain" and a ray is created to trace the final result.

Although Hertel's work allows for an effective method of determining "uncertain" pixels in scene, the method could go further since many of the "uncertain" pixels are not that uncertain. Figure 23 shows the results of conservative rasterization on a simple plane consisting of two triangles. In the BGSM (a), the triangles are individually expanded, resulting in overlapping in the centre of the plane and the unbinding of the edges of triangles. Since this is a shadow mapping, the overlap is not an issue and the edges can be easily clipped in the fragment shader to reduce further inconsistencies. In the SGSM (b), the triangles are shrunken individually resulting in the tearing of the geometry in SGSM. When comparing the SGSM with the BGSM, these tears will be considered

in light in the SGSM and be shadowed in the BGSM, and as result of the test, will considered as "uncertain" pixels and create rays to be traced to determine if they are shadowed or not. This is an unnecessary test, since even normal shadow map method would have considered shadowed.

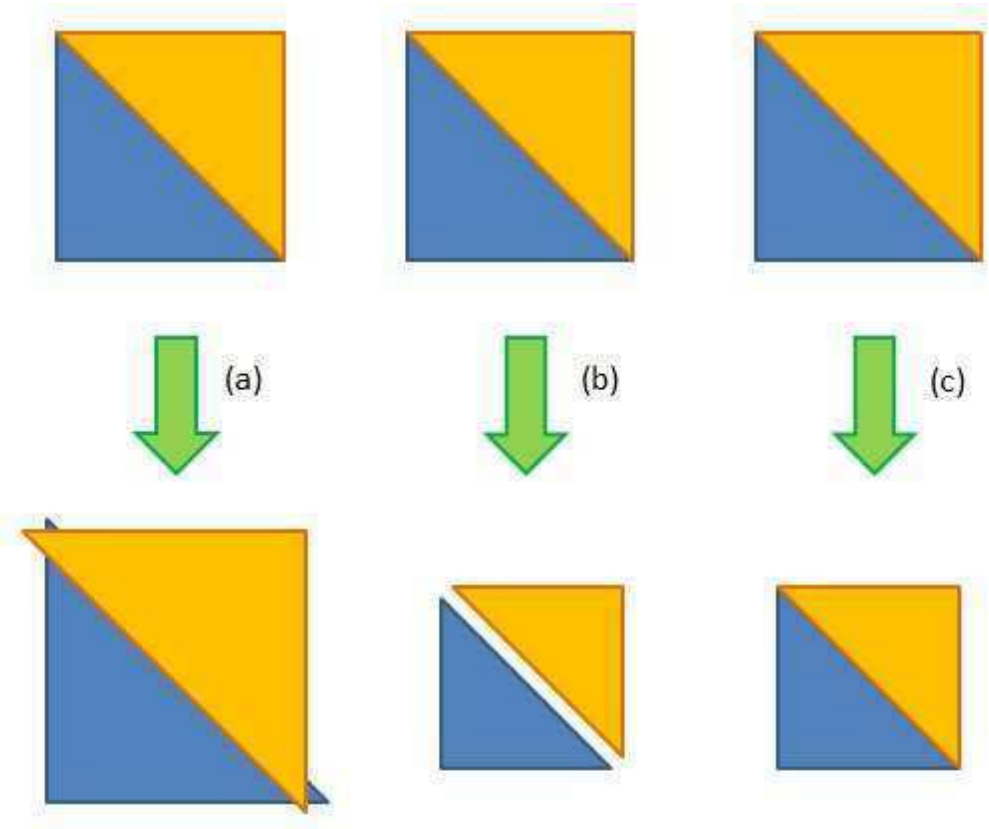


Figure 23 – Geometry Transformations on a pair of triangles that form a plane: (a) Expansion of the triangles in BGSM; (b) Shrinking of triangles in SGSM, without adjacency information; (c) Shrinking of triangles in SGSM, with adjacency information.

This work seeks to reduce the amount of "uncertain" pixels obtained by adding to Hertel's work the adjacency information to produce a more accurate SGSM (c).

OpenGL gives the geometry shader the information of a triangle and all triangles that share an edge with it, as showed in Figure 24.

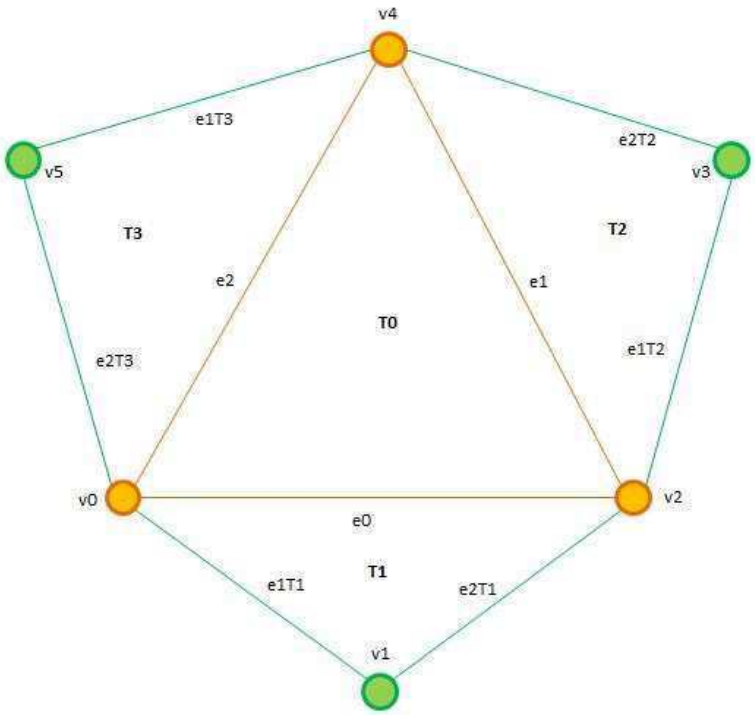


Figure 24 – Central triangle (orange) and the adjacent triangles (green) provided to the geometry shader

The geometry shader receives an array of vertices, $vertex_{adj} = \{v0, v1, v2, v3, v4, v5\}$. As shown in Figure 24, the central triangle $T0$, is formed by $\{v0, v2, v4\}$, T1 is formed by $\{v0, v1, v2\}$, $T2$ and $T3$ are formed by $\{v2, v3, v4\}$ and $\{v4, v5, v0\}$ respectively.

As in conservative rasterization, we can calculate the edges of the new triangles. Following the example of equation (1) in chapter 2.3, the other edges can be calculated like this:

$$\begin{aligned} e1T1 = v1 - v0;\ e2T1 = v2 - v1; \\ e1T2 = v3 - v2;\ e2T2 = v4 - v3; \\ e1T3 = v5 - v4;\ e2T2 = v0 - v5; \end{aligned} \tag{22}$$

These edges are useful to determine if the adjacent triangle exists and determine the facing of the triangles relative to the light. This can be done by the following algorithm for any adjacent triangle ($x \in \{1,2,3\}$):

```
bool txLight //Indicator if the Tx is facing the light/exists

if ( !(length(e1Tx) < 0.001 )){ //Tx exist
          //Calculate the normal of Tx
          normalTx = normalize( cross(vec3(e1Tx), vec3(e2Tx) ));
          if (dot(normalTx, lightDir) > 0)
                    txLight = true; //Tx is facing the light
          else
                    txLight = false; //Tx is not facing the light
          }
else { //Tx doesn't exist
                    txLight = true;
}
```

The first "if" condition is to determine that if the Tx vertex exists. In OpenGL, if adjacent triangle doesn't exist the previous vertex is repeated. For example, if $v1$ doesn't exist, $vertex_{adj} = \{v0, v0, v2, v3, v4, v5\}$. The second "if" condition is to determine the facing of the adjacent triangle, which will effect of the shrinking of the central triangle, to do this we calculate the normal of the triangle Tx and produce the dot of the normal and the light direction. If it's positive, the triangle Tx is facing the light, therefore it is in light. Otherwise, the Tx is not facing the light, therefore it is in shadow.

Any adjacent triangle that doesn't exist will be considered to be in light, in order to simplify the shrinking of the central triangle.

We are only altering the triangles that are in shadow, since these triangles are the ones that the shadow map technique uses to project the shadows into the scene, so we can assume that the central triangle is in shadow. Since each central triangle has three adjacent triangles, at most, we'll have to account the following scenarios:

- None of the adjacent triangles is in shadow;
- One of the adjacent triangles is in shadow;
- Two of the adjacent triangles are in shadow;
- All of the adjacent triangles are in shadow;

For all of these cases, the shrunken edges produced in normal conservative rasterization must be calculated, since these edges are used to create the new triangle.

4.1.1 0-in-Shadow

The simplest case is when none of the adjacent triangles is in shadow. Basically this case is treated like it was a single triangle, following the same process as in normal conservative rasterization, explained in chapter 2.3.

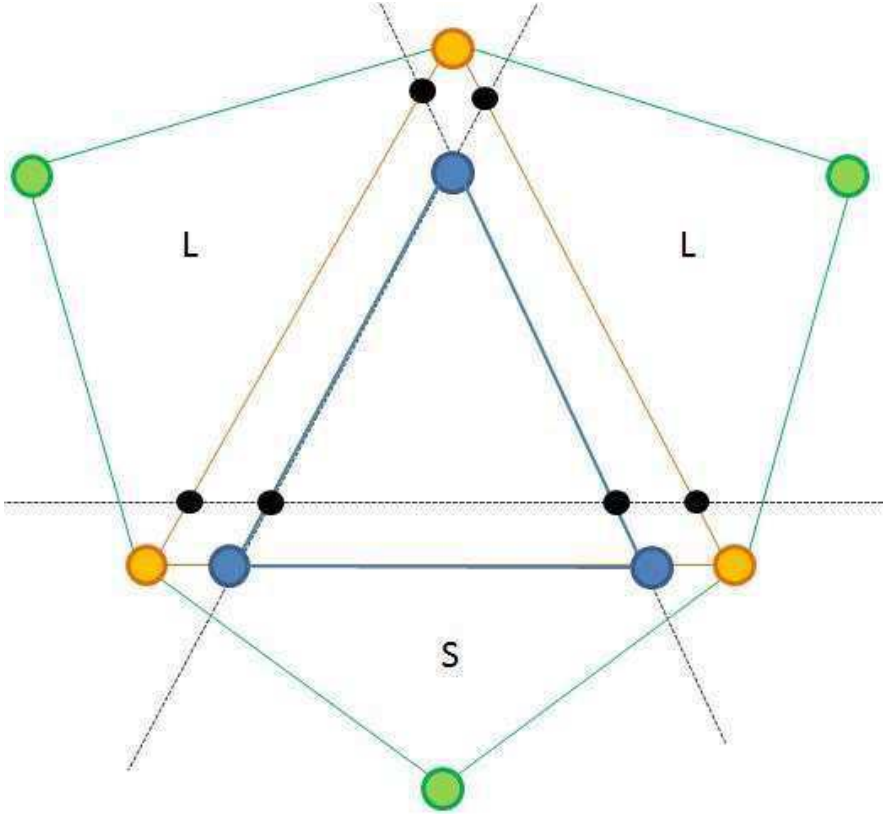


Figure 25 – 0-in-shadow case: the original triangle (orange) is shrunk using normal conservative rasterization (blue).

This mostly occurred in isolated triangles in the scene or in particular geometry constructions where only one triangle is in shadow, while the others are in light. Although this is a rare and mostly unrealistic scenario, it is a possibility and therefore it must be dealt with.

4.1.2 1-in-Shadow

The idea of this case is to shrink only the edges of the triangles that are in light to maintain the connection of the shadowed triangles. To do this we must calculate the intersection between the

new lines formed by the shirking of the lighted edges, and the old line of the shadowed edge, as demonstrated in Figure 26.

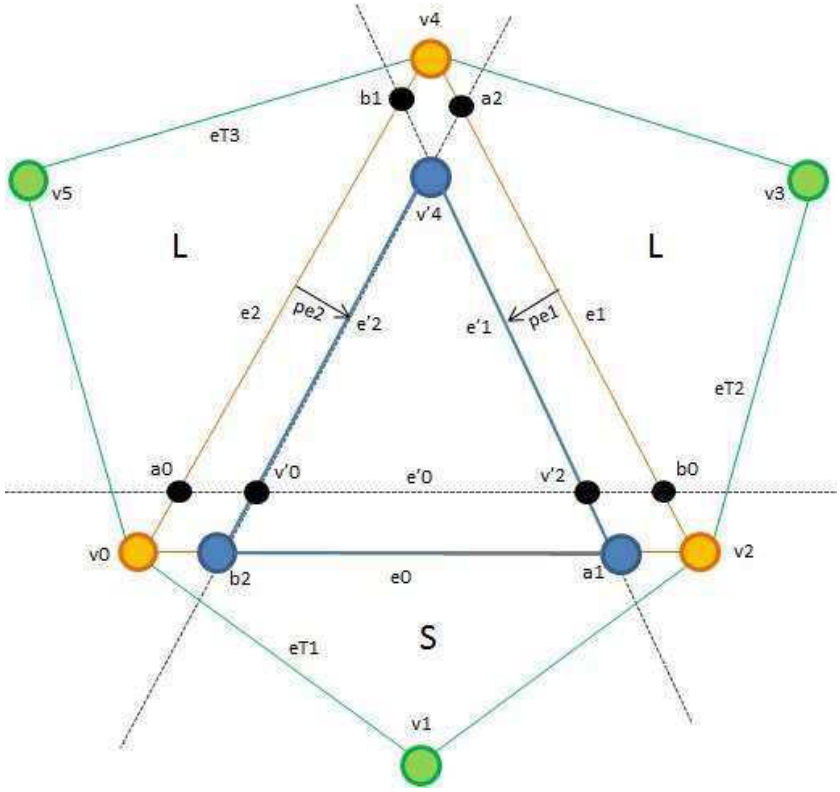


Figure 26 - 1-in-shadow case: The new triangle (blue) is composed by $\{b2, a1, v'4\}$.

Note that bx and ax in figure 4 are not the same points as the sbx and sax in Figure 10. The sbx and the sax are the shifted position of the vertexes of the original triangle, while bx and ax are the intersection points of the new edges. They can be the same, but it's not safe to assume so, due to floating point errors.

Since we already calculated the shrunken triangle, $v'4$ is already taken of. To determine the vertices $b2$ and $a1$, we'll have to find the following line intersections:

- The intersection point of the old edge $e0$ with the new edge $e'2$, which will give us $b2$;
- The intersection point of the old edge $e0$ with the new edge e'1, which will give us $a1$;

Using the *lineLineIntersection* function described in chapter 2.3, these points can be easily determined:

$$b2 = lineLineIntersection(v0, v2, v'4, v'0, n, v0);$$
$$a1 = lineLineIntersection(v0, v2, v'2, v'4, n, v2); \quad (23)$$

After obtaining the new vertices, the geometry shader emits the proper shrunken triangle that maintains the connection with the adjacent shadowed triangle. Figure 27 shows all possible scenarios where only one of the adjacent triangles is in shadow.

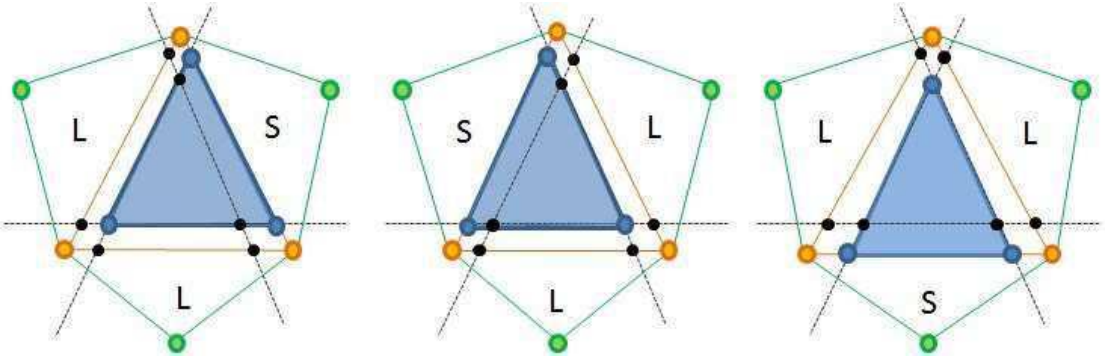


Figure 27 – Three simplified scenarios where the 1-in-Shadow occurs.

These are the easiest cases; there is only the need to calculate two more additional points in order to create the appropriate shrunken triangle.

4.1.3 2-in-Shadow

In these cases, two of the adjacent triangles are in shadow, while the other is in light. The obvious solution is to shrink the edge of the triangle in light. In the example showed in Figure 6, that would result in a new triangle formed by the vertices $\{v0, a1, b1\}$.

But due to the restrictions of the adjacency information in OpenGL, the geometry shader only has access to six vertices at a time, and $v0$ could be part of a triangle that could be in light, but in the current group of vertices we don't have that information. The only safe assumption we can do is consider that $v0$ is in light in another group of $vertex_{adj}$, so we'll cut out $v0$ of the set and creating two new triangles in the geometry shader:

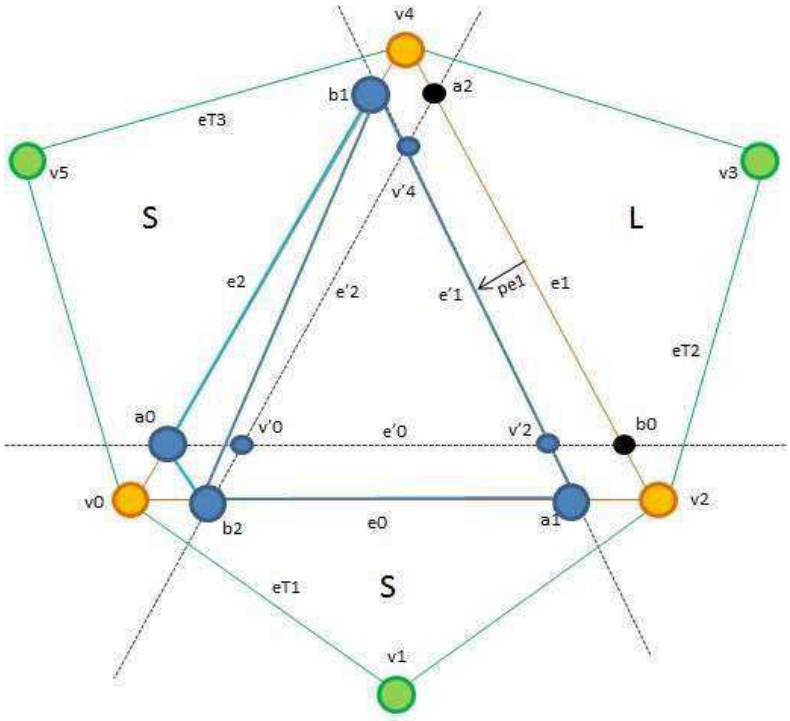


Figure 28 – 2-in-Shadow case: The new geometry is formed by the triangle $\{b2, a1, b1\}$ and the triangle $\{b1, a0, b2\}$.

This will result in the blue and light blue triangles in Figure 28. Like before we'll need to determine the intersections of the old and new edges:

- The intersection point of the old edge $e2$ with the new edge $e'1$, which will give us $b1$;
- The intersection point of the old edge $e0$ with the new edge $e'1$, which will give us $a1$;
- The intersection point of the new edge $e'2$ with the old edge $e0$, which will give us $b2$;
- The intersection point of the new edge $e'0$ with the old edge $e2$, which will give us $a0$;

Using the *lineLineIntersection* function described in chapter 2.3, these points can be easily determined:

$$\begin{aligned} b1 &= lineLineIntersection(v4, v0, sa1, sb1, n, v4); \\ a1 &= lineLineIntersection(v0, v2, sa1, sb1, n, v2); \\ b2 &= lineLineIntersection(v0, v2, sa2, sb2, n, v0); \\ a0 &= lineLineIntersection(v4, v0, sa0, sb0, n, v0); \end{aligned} \tag{24}$$

After obtaining the new vertices, the geometry shader emits two triangles as depicted in Figure 29: a large triangle, containing the central triangle (dark blue), and a smaller triangle, to cut the corner between the adjacent triangles that are in shadow (light blue).

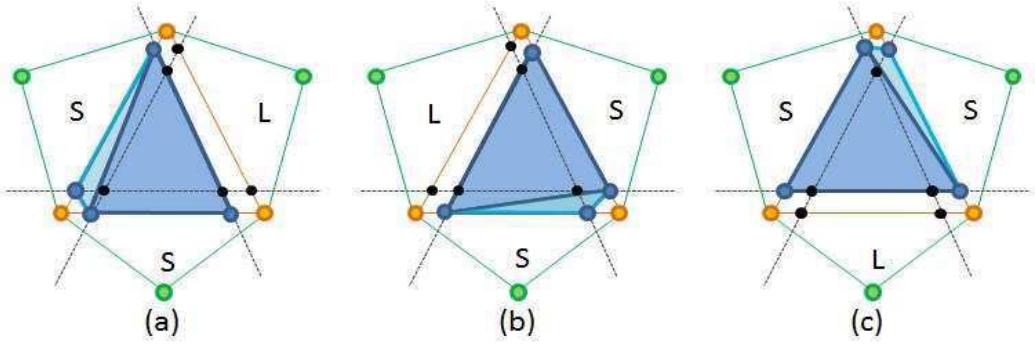


Figure 29 – Three simplified scenarios where the 2-in-Shadow occurs.

These cases are more complex than the previous ones, as they required the emission additional geometry to maintain the connections. In a large number of triangles this method can become more taxing on the GPU.

4.1.4 3-in-Shadow

In this case all the adjacent triangles are in shadow, therefore there is no actual edge to shrink. However the issue that occurred in the 2-in-shadow cases still occurs, we don't know if any of vertexes in the original triangle is in light at another $vertex_{adj}$. Following the same assumption as before, the only thing we can do is to clip the corners of the original triangle, as it's showed in Figure 8. In order to do so, we must calculate all the following intersections:

- The intersection point of the new edge $e'0$ with the old edge $e2$, which will give us $a0$;
- The intersection point of the new edge $e'0$ with the old edge $e1$, which will give us $b0$;
- The intersection point of the new edge $e'1$ with the old edge $e0$, which will give us $a1$;
- The intersection point of the new edge $e'1$ with the old edge $e2$, which will give us $b1$;
- The intersection point of the new edge $e'2$ with the old edge $e1$, which will give us $a2$;
- The intersection point of the new edge $e'2$ with the old edge $e0$, which will give us $b2$;

Using the *lineLineIntersection* function described in chapter 2.3, these points can be easily determined:

$$
\begin{aligned}
a0 &= lineLineIntersection(v4, v0, v'0, v'2, n, v0);\\
b0 &= lineLineIntersection(v2, v4, v'0, v'2, n, v2);\\
\\
a1 &= lineLineIntersection(v0, v2, v'2, v'4, n, v2);\\
b1 &= lineLineIntersection(v4, v0, v'2, v'4, n, v4);\\
\\
a2 &= lineLineIntersection(v2, v4, v'4, v'0, n, v4);\\
b2 &= lineLineIntersection(v0, v2, v'4, v'0, n, v0);
\end{aligned}
\tag{25}
$$

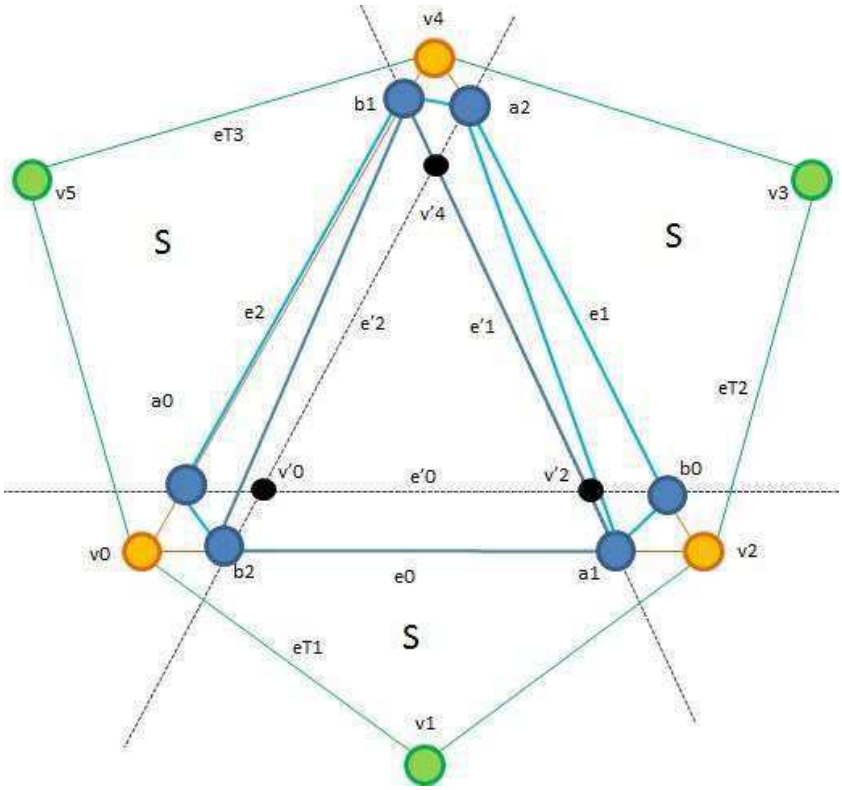


Figure 30 – 3-in-Shadow case: The new geometry is formed by the four triangles: $\{b1, a0, b2\}$, $\{b2, a1, b1\}$, $\{b1, a1, a2\}$ and $\{a2, a1, b0\}$.

After obtaining the new vertices, the geometry shader emits four triangles, depicted by the blue and light blue lines in Figure 8.

This is the worst case, since it's the most common situation when treating shadowed objects. When a side of and objects is shadow, great number of those triangles that are part of the object

will fall in this case, and for each of those triangles, four are emitted to shrink and maintain the connectivity of the object.

4.1.5 Worst Case Errors

Despite these upsides, Figure 31 shows an example where our approach fails to completely maintain the connections of all the triangles in the geometry. The plane formed by four triangles in (b) is shrunk with our approach. While each triangle have the information of the adjacent triangles that share an edge, the triangles with the same colour share only a single vertex, and so don't appear in the adjacency list $vertex_{adj}$, which causes the vertex in the middle to disappear, since each triangle will clip it out, resulting in a hole in the geometry. This is due to the restrictions of the adjacency information of only three adjacent triangles, and the resulting assumptions made in 2-in-Shadow and 3-in-Shadow. A way to counteract this problem is to provide the shader all of the adjacency information of the scene before rendering, but in large scenes, this adjacency list would have to list the adjacency of billions of triangles, which becomes impractical.

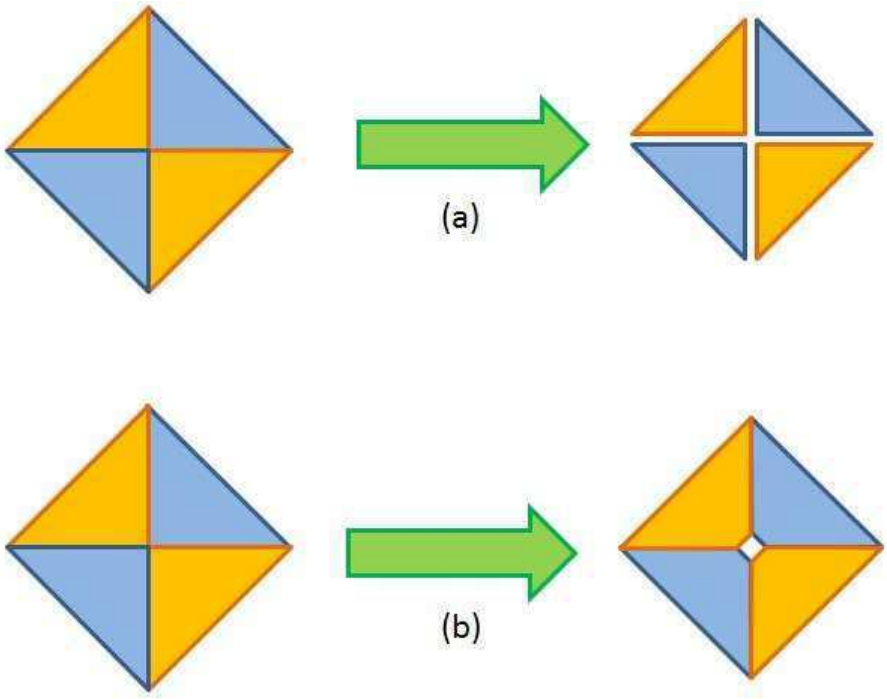


Figure 31 - Worst Case Scenario: (a) Normal conservative rasterization; (b) Conservative rasterization with adjacency.

Still, this provides a superior alternative that normal conservative rasterization, where the triangles would a shrunk inwards and causer tears in the geometry. While at (a), rays would have to be traced from all the pixels in the tears, in (b) the only pixels in the holes would have a ray created to be traced.

4.1.6 Conclusion

This approach to conservative rasterization solves the problem of tearing when shrinking triangles in the geometry. By takin in the adjacency information provided by OpenGL, we can determine efficiently each edges of a triangle are safe to shrink to maintain the connectivity of the triangle. Although these transformations will result in more triangles being emitted by the geometry shader, this will not have an impact in the result of the shadow map, since these overlaps will occur in the places where tears would occur normally, filling them in the fragment shader. The one perceivable impact is in performance, since we are emitting more triangles than in the normal Conservative Rasterization, but this leads to less rays being traced in the following step, as demonstrated in the next chapters.

For future, the possibility of incorporating Embree's ray tracing kernels, developed by Intel, can help determine more accurately the cost of tracing a shadow ray from the scene, because NVIDIA's OptiX Prime does not encapsulate the entire algorithm of which ray tracing is a part, thus, prime cannot refactor the computation for performance. This would also be useful as comparition between NVIDIA's OptiX engine and Intel's Embree.

5 Algorithm Testing

This chapter will start by presenting the scenes used and the ray tracing test results obtained by OptiX Prime. After this, we'll demonstrate the pixels results obtained by using standard shadow mapping, shadow map and ray tracing hybrid presented by Hertel's work, and our approach. As for the specifics of the tests, each test will count the amount of pixels in different states:

- Pixels Facing the Light (PFL), i.e. dot(n,l) > 0
- Pixels Against the Light (PAL), i.e. dot (n,l) < 0
- PFL – Pixels in Light;
- PFL – Pixels in Shadow.

The tables present percentages in relative to the size of the viewport, but also relative to the type of pixels they represent, for example the PFL encapsulate the pixels in light and in shadow, so the percentages will try represent the split between these cases within the PFL. Hertel's and our approach will contain two more states, to determine the amount of "uncertain" pixels found during the rendering steps:

- Pixels in Light – "Uncertain";
- Pixels in Shadow – "Uncertain".

Since each of the "uncertain" pixels will be corrected into light or shadow, these percentages will represent the amount of "uncertain" pixels that appeared in each state.

Finally, each shadow map test will be compared to the Optix Prime results to determine the amount of similarity between the tests.

Tests were also done with shadow map resolutions of 512x512, 1024x1024, 2048x2048 and 4096x4096, with viewport resolutions of 512x512, 1024x1024 and 1920x1080 (FullHD). The view frustum will have the minimum size needed to contain the objects being seen by the camera, including also all the geometry that could influence lighting, from each one of the viewpoints.

5.1 Test Scenes

The following images will show the scenes and that will be used for testing and the various viewpoints that will be used for said tests.

The first scene consists of a scene with two trees, a lamp, a flower box and a bench on a plane. The scene has a total of 55026 triangles. This scene will be called "Bench". Information of light, camera and field of view of this scene can be observed in Table 1.

Viewport		Coordinates		
		x	y	z
Side	Position	-23,277	18,541	30,143
	Direction	0,397	-0,644774	-0,652
With	Position	-37,034573	35,208973	-8,597797
	Direction	0,605439	-0,732089	0,312232
Against	Position	27,214222	27,875109	27,032139
	Direction	-0,560848	-0,777942	-0,283293
Light Direction		0,744	-0,408	0,527
View Frustum		Far: 120,0	Near: 15,0	FoV: 60°

Table 1 -Information of viewports used for the Bench Scene.

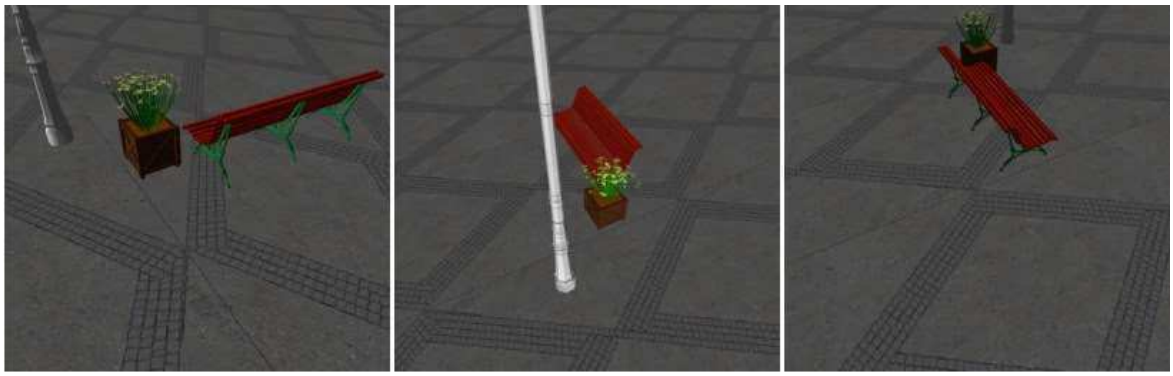


Figure 32 - The side (left), with (centre) and against (right) viewpoints of the first scene.

The second scene, named "Flowers", will also use the same models as the first scene, but will closely observe the shadows cast by the flowers. The flowers are modelled with very small

triangles, allowing the visualization the effect of small geometry on the algorithm. In Table 2 the information of the camera of each viewpoint can be viewed.

Viewport		Coordinates		
		x	y	z
Side	Position	-3,615331	22,376335	2,338565
	Direction	-0,387214	-0,852832	0,350347
With	Position	-17,561422	24,968716	4,010894
	Direction	0,386402	-0,873032	0,297505
Against	Position	-3,263903	24,423452	12,998949
	Direction	-0,239566	-0,958412	-0,155095
Light Direction		0,744	-0,408	0,527
View Frustum		Far: 120,0	Near: 15,0	FoV: 60°

Table 2 -Information of viewports used for the Flower Scene.

Figure 33 -The side (left), against (centre) and with (right) viewpoints of the second scene.

The third scene, called "Trees", will use the same models as the second scene, but will focus attention on an area of the ground where only the shadows of the trees will be seen. Since the trees are constituted by big triangles, this will allow the evaluation of the effect of big triangles on the results. Information of cameras of each viewpoint can be observed in Table 3.

Viewport		Coordinates		
		x	y	z
Side	Position	76,844704	28,391548	-31,870102
	Direction	-0,232891	-0,79644	0,558073
With	Position	42,947086	24,103859	-27,831772

	Direction	0,415959	-0,784187	0,460467
Against	Position	90,805244	35,846294	24,061787
	Direction	-0,421061	-0,852832	0,350347
Light Direction		0,744	-0,408	0,527
View Frustum		Far: 120,0	Near: 15,0	FoV: 60°

Table 3 -Information of viewports used for the Trees Scene.

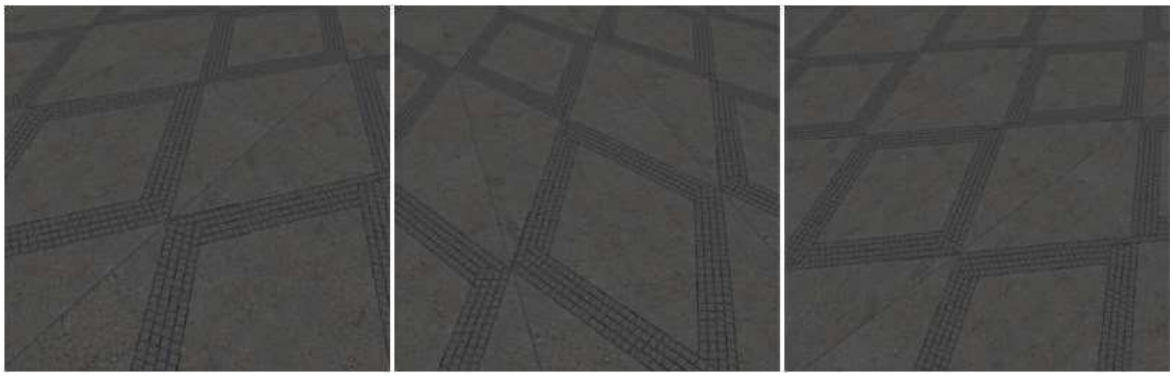


Figure 34 - The with (left), side (centre) and against (right) viewpoints of the third scene.

5.2 Ray tracing Scenes

First set of tests is to determine the number of pixel states obtain in OptiX Prime. Since the ray tracing lighting is the most accurate method to determine the lighting of the scenes, these results will serve as base of comparison of the accuracy for the rest of the methods.

As demonstrated in the following tables, the distribution of the pixels states maintain the same thought the viewport sizes. The major change occurs when the aspect ratio changes to 1.78, as demonstrated in the 1920x1080 viewport, where the ratios drastically change the results.

Table 4 shows the results for the best case test for many of the methods, since it doesn't possess many pixels not facing the light, since it most of the shadows are cast onto the floors by the trees, and the majority are in light than in shadow. The change of ratio increased slightly the amount of pixels in change, as well adding some PnFLs to the image.

Scene	Side
Viewport	Trees
Pixel Types	Viewport Size

	512x512	1024x1024	1920x1080
PFL	262144 (100,00%)	1048576 (100,00%)	2073445 (99,99%)
PnFL	0 (0,00%)	0 (0,00%)	155 (0,01%)
PFL – in Light	168949 (64,45%)	675789 (64,45%)	1320285 (63,38%)
PFL – in Shadow	93195 (35,55%)	372787 (35,55%)	753160 (36,62%)

Table 4 – Optix Prime results for the best case: PFL represents the Pixels Facing the Light and PnFL represents the Pixels not Facing the Light; The PFLs are then split into the pixels in Light and the pixels in Shadow.

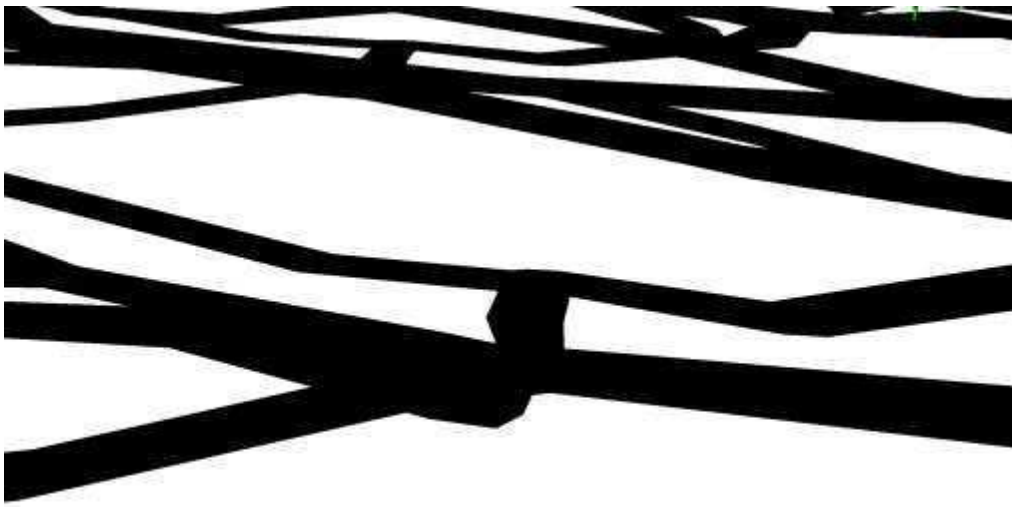


Figure 35 - Best Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080.

Table 5 shows the results for the worst case test for many of the methods, due the flowers possess very small triangles, which causes many of the errors in the shadow map techniques. The change of ratio decreased the amount of PFnL pixels, while increasing the amount of light pixels from almost half (~53%) to more than 2/3 of the image (~70%).

Scene	Against		
Viewport	Flower		
Pixel Types	Viewport Size		
	512x512	1024x1024	1920x1080
PFL	246986 (94,21%)	987819 (94,21%)	2006003 (96,74%)
PnFL	15158 (5,79%)	60757 (5,79%)	67597 (3,26%)
PFL – in Light	130890 (52,99%)	523452 (52,99%)	1399266 (69,75%)

PFL – in Shadow	116096 (47,01%)	464367 (47,01%)	606737 (30,25%)

Table 5 - Optix Prime results for the worst case: PFL represents the Pixels Facing the Light and PnFL represents the pixels; The PFLs are then split into the pixels in Light and the pixels in Shadow.

Figure 36 - Worst Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080.

Table 6 shows the results for the average case test for many of the methods, since the errors in the flowers still exist; these appear in smaller number, not becoming more prominent than the projected shadows of the bench and trees. The change of ratio decreased the amount of PFnL pixels, while increasing the amount of light pixels from ~59% to ~60% of the image.

Scene	Against		
Viewport	Bench		
Pixel Types	Viewport Size		
	512x512	1024x1024	1920x1080
PFL	255894 (97,62%)	1023625 (97,62%)	2045839 (98,66%)
PnFL	6250 (2,38%)	24951 (2,38%)	27761 (1,34%)
PFL – in Light	151058 (59,03%)	604276 (59,03%)	1296685 (63,38%)
PFL – in Shadow	104836 (40,97%)	419349 (40,97%)	749154 (36,62%)

Table 6 - Optix Prime results for the average case: PFL represents the Pixels Facing the Light and PnFL represents the pixels; The PFLs are then split into the pixels in Light and the pixels in Shadow.

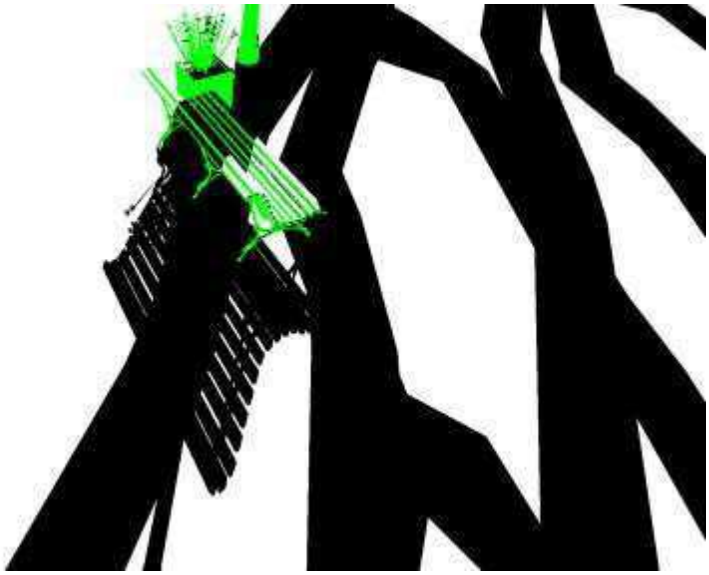


Figure 37 - Average Case Result for Prime; The Green Pixels represent the PnFL pixels; Viewport Size: 1920x1080.Shadow Mapping Errors

These sets of test will demonstrate the amount of pixels in light and shadow obtained by the following techniques. We'll also compare the accuracy of these numbers to determine where the techniques fail, the size of the error in the picture obtained.

For each case, we'll demonstrate the best case, worst case and average case. Since shadow mapping results varies according to the size of the shadow map, for each view port will compare the results for the different sizes of SM.

The PFL and PnFL obtained in the following test are not showed due to these maintain constant thought the tests, which make sense, since the amount of pixels facing the light and not facing the light are not influenced by the shading technique, but the geometry information.

5.2.1 Normal Shadow Mapping

Now the shadow mapping results will be compared against the ray-tracer. Although, it maintains a similar distribution of light pixels (64%) and shadow pixels (36%). The test performed show that the majority of the errors of shadow map happen in the contours of the shadows.

More test results are present in section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	168951 64,45%	168878 64,42%	168973 64,46%	168929 64,44%
	Pixels in Shadow	93193 35,55%	93266 35,58%	93171 35,54%	93215 35,56%
	Light - Correct	167335 99,04%	168094 99,54%	168558 99,75%	168723 99,88%
	Light - Incorrect	1616 0,96%	784 0,46%	415 0,25%	206 0,12%
	Shadow - Correct	91579 98,27%	92411 99,08%	92780 99,58%	92989 99,76%
	Shadow - Incorrect	1614 1,73%	855 0,92%	391 0,42%	226 0,24%
1024 x 1024	Pixels in Light	675861 64,46%	675618 64,43%	675924 64,46%	675793 64,45%
	Pixels in Shadow	372715 35,54%	372958 35,57%	372652 35,54%	372783 35,55%
	Light - Correct	669218 99,02%	672393 99,52%	674183 99,74%	674960 99,88%
	Light - Incorrect	6643 0,98%	3225 0,48%	1741 0,26%	833 0,12%
	Shadow - Correct	366144 98,24%	369562 99,09%	371046 99,57%	371954 99,78%
	Shadow - Incorrect	6571 1,76%	3396 0,91%	1606 0,43%	829 0,22%
1920 x 1080 (FullHD)	Pixels in Light	1321779 63,75%	1319273 63,63%	1320432 63,68%	1320291 63,68%
	Pixels in Shadow	751666 36,25%	754172 36,37%	753013 36,32%	753154 36,32%

	Light - Correct	1304537 98,70%	1311354 99,40%	1316287 99,69%	1318214 99,84%
	Light - Incorrect	17242 1,30%	7919 0,60%	4145 0,31%	2077 0,16%
	Shadow - Correct	735918 97,90%	745241 98,82%	749015 99,47%	751083 99,73%
	Shadow - Incorrect	15748 2,10%	8931 1,18%	3998 0,53%	2071 0,27%

Table 7 – Normal Shadow Mapping results for the best case;

As seen in Table 7, the errors in the contours appear more in the shadow pixels than in the light pixels, with an average difference of 0,38%. The amount of errors decrease with the increase of the shadow map resolution, with ~0,37% decrease in the light pixels and a ~0,50% decrease in shadow pixels. But there is still around 4148 (0,2%) error pixels in the highest shadow map resolution of in FullHD image,

Figure 38 - Best Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene	Against
Viewport	Flowers

Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	131133 53,09%	131384 53,19%	131536 53,26%	131472 53,23%
	Pixels in Shadow	115853 46,91%	115602 46,81%	115450 46,74%	115514 46,77%
	Light - Correct	128612 98,08%	129437 98,52%	130070 98,89%	130367 99,16%
	Light - Incorrect	2521 1,92%	1947 1,48%	1466 1,11%	1105 0,84%
	Shadow - Correct	113575 98,03%	114149 98,74%	114630 99,29%	114991 99,55%
	Shadow - Incorrect	2278 1,97%	1453 1,26%	820 0,71%	523 0,45%
1024 x 1024	Pixels in Light	524600 53,11%	525407 53,19%	526086 53,26%	525736 53,22%
	Pixels in Shadow	463219 46,89%	462412 46,81%	461733 46,74%	462083 46,78%
	Light - Correct	514423 98,06%	517695 98,53%	520181 98,88%	521321 99,16%
	Light - Incorrect	10177 1,94%	7712 1,47%	5905 1,12%	4415 0,84%
	Shadow - Correct	454190 98,05%	456655 98,76%	458462 99,29%	459952 99,54%
	Shadow - Incorrect	9029 1,95%	5757 1,24%	3271 0,71%	2131 0,46%
1920 x 1080 (FullHD)	Pixels in Light	1401605 69,87%	1402284 69,90%	1401876 69,88%	1401839 69,88%
	Pixels in Shadow	604398 30,13%	603719 30,10%	604127 30,12%	604164 30,12%
	Light - Correct	1382439 98,63%	1389470 99,09%	1393259 99,39%	1395775 99,57%
	Light - Incorrect	19166 1,37%	12814 0,91%	8617 0,61%	6064 0,43%
	Shadow - Correct	587571 97,22%	593923 98,38%	598120 99,01%	600673 99,42%
	Shadow - Incorrect	16827 2,78%	9796 1,62%	6007 0,99%	3491 0,58%

Table 8 - Normal Shadow Mapping results for the worst case;

As seen in Table 8, the errors in the contours appear more in the light pixels than in the shadow pixels, with an decrease of 0,38%. The amount of errors decrease with the increase of the shadow map resolution, with ~0,36%(600 pixels) decrease in the light pixels and a ~0,50% (600 pixels) decrease in shadow pixels . But there is still around 9555 (0,46%) error pixels in the highest shadow map resolution of in FullHD image.

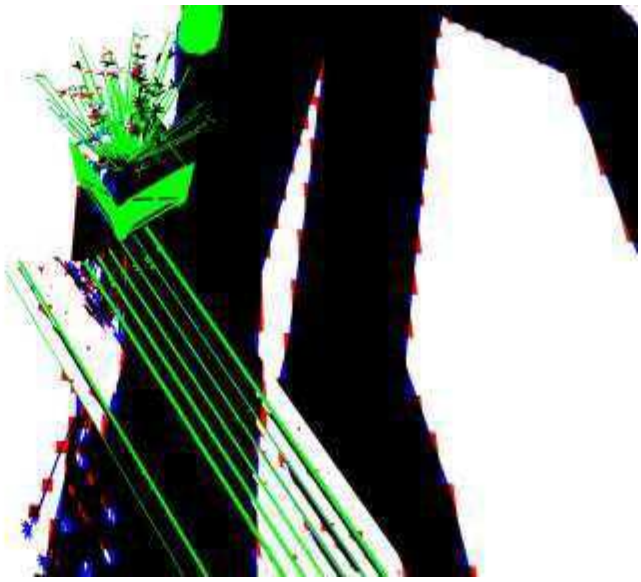


Figure 39 – Worst Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	151978 59,39%	151213 59,09%	151170 59,08%	151186 59,08%
	Pixels in Shadow	103916 40,61%	104681 40,91%	104724 40,92%	104708 40,92%
	Light - Correct	149154 98,14%	149930 99,15%	150431 99,51%	150727 99,70%
	Light - Incorrect	2824 1,86%	1283 0,85%	739 0,49%	459 0,30%

	Shadow - Correct	102012 98,17%	103553 98,92%	104097 99,40%	104377 99,68%
	Shadow - Incorrect	1904 1,83%	1128 1,08%	627 0,60%	331 0,32%
1024 x 1024	Pixels in Light	607887 59,39%	604879 59,09%	604717 59,08%	604788 59,08%
	Pixels in Shadow	415738 40,61%	418746 40,91%	418908 40,92%	418837 40,92%
	Light - Correct	596703 98,16%	599765 99,15%	601761 99,51%	602945 99,70%
	Light - Incorrect	11184 1,84%	5114 0,85%	2956 0,49%	1843 0,30%
	Shadow - Correct	408165 98,18%	414235 98,92%	416393 99,40%	417506 99,68%
	Shadow - Incorrect	7573 1,82%	4511 1,08%	2515 0,60%	1331 0,32%
1920 x 1080 (FullHD)	Pixels in Light	1294206 63,26%	1298218 63,46%	1297078 63,40%	1297634 63,43%
	Pixels in Shadow	751633 36,74%	747621 36,54%	748761 36,60%	748205 36,57%
	Light - Correct	1278358 98,78%	1287139 99,15%	1291451 99,57%	1294148 99,73%
	Light - Incorrect	15848 1,22%	11079 0,85%	5627 0,43%	3486 0,27%
	Shadow - Correct	733306 97,56%	738075 98,72%	743527 99,30%	745668 99,66%
	Shadow - Incorrect	18327 2,44%	9546 1,28%	5234 0,70%	2537 0,34%

Table 9 - Normal Shadow Mapping results for the average case;

As seen in Table 9, the errors in the contours appear more in the shadow pixels than in the light pixels, with an average difference of 0,08%. The amount of errors decrease with the increase of the shadow map resolution, with ~0,52% decrease in the light pixels and a ~0,50% decrease in shadow pixels. But there is still around 6023 (0,29%) error pixels in the highest shadow map resolution of in FullHD image,

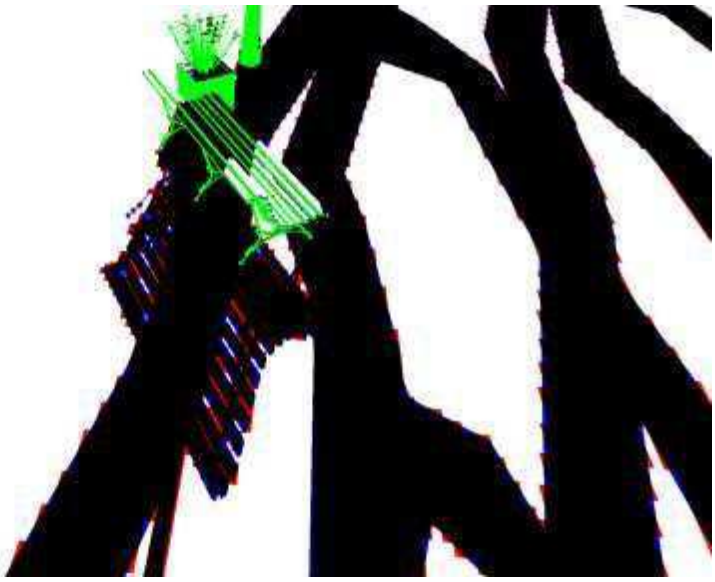


Figure 40 - Average Case Result for Normal Shadow Mapping; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.2.2 BGSM Shadow Mapping

Although this is not a specific method, these tests results show one of the main concepts of 2 layer shadow mapping methods: the pixels in light in the BGSM are always correct. In all three cases, this is proven true.

The number of pixels to expand the triangles (Θ) in all tests was set to 1.

More test results are present in section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	159700 60,92%	164373 62,70%	166581 63,55%	167779 64,00%
	Pixels in Shadow	102444 39,08%	97771 37,30%	95563 36,45%	94365 36,00%
	Light - Correct	159700 100,00%	164373 100,00%	166581 100,00%	167779 100,00%

	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	93195 90,97%	93195 95,32%	93195 97,52%	93195 98,76%
	Shadow - Incorrect	9249 9,03%	4576 4,68%	2368 2,48%	1170 1,24%
1024 x 1024	Pixels in Light	638724 60,91%	657464 62,70%	666366 63,55%	671111 64,00%
	Pixels in Shadow	409852 39,09%	391112 37,30%	382210 36,45%	377465 36,00%
	Light - Correct	638724 100,00%	657464 100,00%	666366 100,00%	671111 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	372787 90,96%	372787 95,31%	372787 97,53%	372787 98,76%
	Shadow - Incorrect	37065 9,04%	18325 4,69%	9423 2,47%	4678 1,24%
1920 x 1080 (FullHD)	Pixels in Light	1227165 59,18%	1273867 61,44%	1296451 62,53%	1308494 63,11%
	Pixels in Shadow	846280 40,82%	799578 38,56%	776994 37,47%	764951 36,89%
	Light - Correct	1227165 100,00%	1273867 100,00%	1296451 100,00%	1308493 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	1 0,00%
	Shadow - Correct	753160 89,00%	753160 94,19%	753160 96,93%	753159 98,46%
	Shadow - Incorrect	93120 11,00%	46418 5,81%	23834 3,07%	11792 1,54%

Table 10 – BSGM Shadow Mapping results for the best case;

As seen in Table 10, there is an increase in the number of shadow pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the shadow pixels as the shadow map resolution is increased (~%). This is because of the pixel size λ of chapter 4, since it is inversely proportional to the height and width of the shadow map, λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the shadows to reduce slightly in the shadow maps.

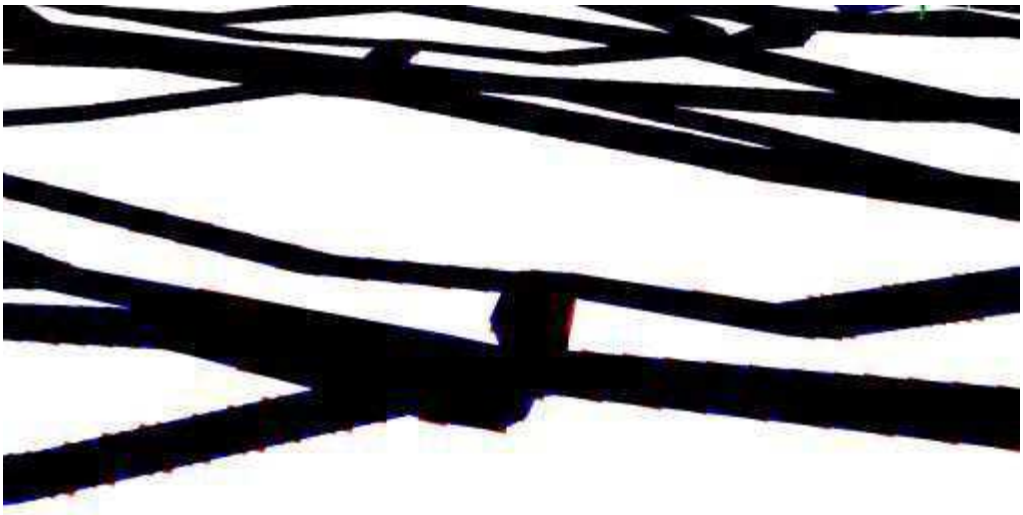


Figure 41 - Best Case Result for BGSM; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Flowers			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	117262 47,48%	123050 49,82%	126589 51,25%	128749 52,13%
	Pixels in Shadow	129724 52,52%	123936 50,18%	120397 48,75%	118237 47,87
	Light - Correct	117169 99,92%	122888 99,87%	126362 99,82%	128445 99,76
	Light - Incorrect	93 0,08%	162 0,13%	227 0,18%	304 0,24
	Shadow - Correct	116003 89,42%	115934 93,54%	115869 96,24%	115792 97,93
	Shadow - Incorrect	13721 10,58%	8002 6,46%	4528 3,76%	2445 2,07
1024 x 1024	Pixels in Light	469026 47,48%	492170 49,82%	506215 51,25%	514926 52,13%
	Pixels in Shadow	518793	495649	481604	472893

		52,52%	50,18%	48,75%	47,87%
	Light - Correct	468643 99,92%	491504 99,86%	505307 99,82%	513672 99,76%
	Light - Incorrect	383 0,08%	666 0,14%	908 0,18%	1254 0,24%
	Shadow - Correct	463984 89,44%	463701 93,55%	463459 96,23%	463113 97,93%
	Shadow - Incorrect	54809 10,56%	31948 6,45%	18145 3,77%	9780 2,07%
1920 x 1080 (FullHD)	Pixels in Light	1296876 64,65%	1339411 66,77%	1367053 68,15%	1383087 68,95%
	Pixels in Shadow	709127 35,35%	666592 33,23%	638950 31,85%	622916 31,05%
	Light - Correct	1296594 99,98%	1338802 99,95%	1366241 99,94%	1381899 99,91%
	Light - Incorrect	282 0,02%	609 0,05%	812 0,06%	1188 0,09%
	Shadow - Correct	606455 85,52%	606128 90,93%	605925 94,83%	605549 97,21%
	Shadow - Incorrect	102672 14,48%	60464 9,07%	33025 5,17%	17367 2,79%

Table 11 - BGSM Shadow Mapping results for the worst case;

As seen in Table 11, there is an increase in the number of shadow pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the shadow pixels as the shadow map resolution is increased (~%). This is because of the pixel size λ of chapter 4, since it is inversely proportional to the height and width of the shadow map, λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the shadows to reduce slightly in the shadow maps (~%).

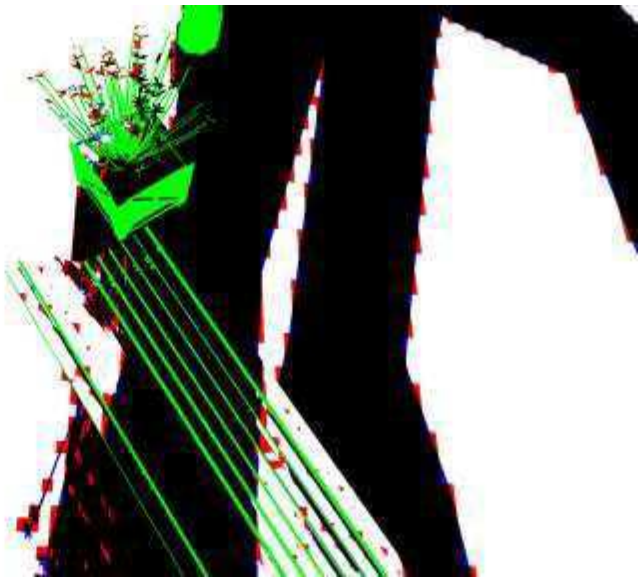


Figure 42 - Worst Case Result for BGSM; Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	141648 55,35%	145423 56,83%	147885 57,79%	149403 58,38%
	Pixels in Shadow	114246 44,65%	110471 43,17%	108009 42,21%	106491 41,62%
	Light - Correct	141606 99,97%	145360 99,96%	147806 99,95%	149311 99,94%
	Light - Incorrect	42 0,03%	63 0,04%	79 0,05%	92 0,06%
	Shadow - Correct	104794 91,73%	104773 94,84%	104757 96,99%	104744 98,36%
	Shadow - Incorrect	9452 8,27%	5698 5,16%	3252 3,01%	1747 1,64%
1024 x 1024	Pixels in Light	566488 55,34%	581669 56,82%	591554 57,79%	597633 58,38%
	Pixels in Shadow	457137	441956	432071	425992

		44,66%	43,18%	42,21%	41,62%
	Light - Correct	566355 99,98%	581445 99,96%	591283 99,95%	597326 99,95%
	Light - Incorrect	133 0,02%	224 0,04%	271 0,05%	307 0,05%
	Shadow - Correct	419216 91,70%	419125 94,83%	419078 96,99%	419042 98,37%
	Shadow - Incorrect	37921 8,30%	22831 5,17%	12993 3,01%	6950 1,63%
1920 x 1080 (FullHD)	Pixels in Light	1209260 59,11%	1248611 61,03%	1270168 62,09%	1282411 62,68%
	Pixels in Shadow	836579 40,89%	797228 38,97%	775671 37,91%	763428 37,32%
	Light - Correct	1209175 99,99%	1248375 99,98%	1269881 99,98%	1282096 99,98%
	Light - Incorrect	85 0,01%	236 0,02%	287 0,02%	315 0,02%
	Shadow - Correct	749069 89,54%	748918 93,94%	748867 96,54%	748839 98,09%
	Shadow - Incorrect	87510 10,46%	48310 6,06%	26804 3,46%	14589 1,91%

Table 12 - BGSM Shadow Mapping results for the average case;

As seen in Table 12, there is an increase in the number of shadow pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the shadow pixels as the shadow map resolution is increased (~%). This is because of the pixel size λ of chapter 4, since it is inversely proportional to the height and width of the shadow map, λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the shadows to reduce slightly in the shadow maps (~%).

Figure 43 - Average Case Result for BGSM; Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.2.3 SGSM Shadow Mapping without Adjacency

Although this is not a specific method, these tests results show the other of the main concepts of 2 layer shadow mapping methods: the pixels in shadow in the SGSM are always correct. In all three cases, this is proven true.

The number of pixels to expand the triangles (Θ) in all tests was set to 1. This is the method used in Hertel's work.

More test results are present in section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	209204 79,81%	190225 72,57%	179873 68,62%	174545 66,58%
	Pixels in Shadow	52940 20,20%	71919 27,43%	82271 31,38%	87599 33,42%
	Light - Correct	168949 80,76%	168949 88,82%	168949 93,93%	168949 96,79%

	Light - Incorrect	40255 19,24%	21276 11,18%	10924 6,07%	5596 3,21%
	Shadow - Correct	52940 100,00%	71919 100,00%	82271 100,00%	87599 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1024 x 1024	Pixels in Light	836946 79,82%	761014 72,58%	719526 68,62%	698102 66,58%
	Pixels in Shadow	211630 20,18%	287562 27,42%	329050 31,38%	350474 33,42%
	Light - Correct	675789 80,74%	675789 88,80%	675789 93,92%	675789 96,80%
	Light - Incorrect	161157 19,26%	85225 11,20%	43737 6,08%	22313 3,20%
	Shadow - Correct	211630 100,00%	287562 100,00%	329050 100,00%	350474 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1688936 81,46%	1519716 73,29%	1423627 68,66%	1373075 66,22%
	Pixels in Shadow	384509 18,54%	553729 26,71%	649818 31,34%	700370 33,78%
	Light - Correct	1320285 78,17%	1320285 86,88%	1320285 92,74%	1320285 96,16%
	Light - Incorrect	368651 21,83%	199431 13,12%	103342 7,26%	52790 3,84%
	Shadow - Correct	384509 100,00%	553729 100,00%	649818 100,00%	700370 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%

Table 13 – SGSM Shadow Mapping without Adjacency results for the best case;

As seen in Table 13, there is an increase in the number of light pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the light pixels as the shadow map resolution is increased (~%). This is same problem showed BGSM Testing, the pixel size λ is inversely proportional to the height and width of the shadow map, as such λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the light to reduce slightly in the shadow maps (~%).

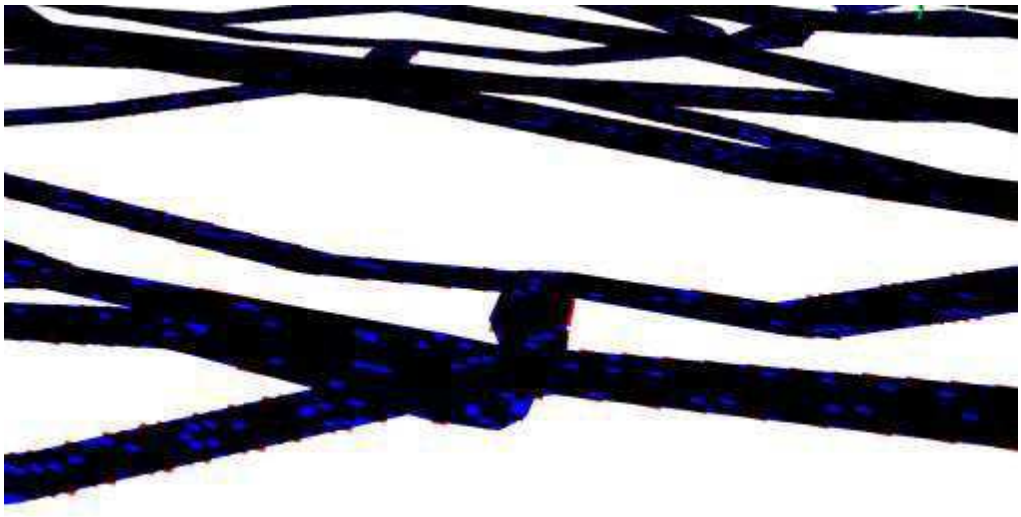


Figure 44 - Best Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Flowers			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	175739 71,15%	159084 64,41%	148869 60,27%	143392 58,06%
	Pixels in Shadow	71247 28,85%	87902 35,59%	98117 39,73%	103594 41,94%
	Light - Correct	130890 74,48%	130890 82,28%	130890 87,92%	130890 91,28%
	Light - Incorrect	44849 25,52%	28194 17,72%	17979 12,08%	12502 8,72%
	Shadow - Correct	71247 100,00%	87902 100,00%	98117 100,00%	103594 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1024 x 1024	Pixels in Light	703074 71,17%	636365 64,42%	595683 60,30%	573420 58,05%
	Pixels in Shadow	284745 28,83%	351454 35,58%	392136 39,70%	414399 41,95%

	Light - Correct	523452 74,45%	523452 82,26%	523452 87,87%	523452 91,29%
	Light - Incorrect	179622 25,55%	112913 17,74%	72231 12,13%	49968 8,71%
	Shadow - Correct	284745 100,00%	351454 100,00%	392136 100,00%	414399 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1710675 85,28%	1592873 79,41%	1516077 75,58%	1474234 73,49%
	Pixels in Shadow	295328 14,72%	413130 20,59%	489926 24,42%	531769 26,51%
	Light - Correct	1399266 81,80%	1399266 87,85%	1399266 92,30%	1399266 94,91%
	Light - Incorrect	311409 18,20%	193607 12,15%	116811 7,70%	74968 5,09%
	Shadow - Correct	295328 100,00%	413130 100,00%	489926 100,00%	531769 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%

Table 14 - SGSM Shadow Mapping without Adjacency results for the worst case;

As seen in Table 14, there is an increase in the number of light pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the light pixels as the shadow map resolution is increased (~%). This is same problem showed BGSM Testing, the pixel size λ is inversely proportional to the height and width of the shadow map, as such λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the light to reduce slightly in the shadow maps (~%).

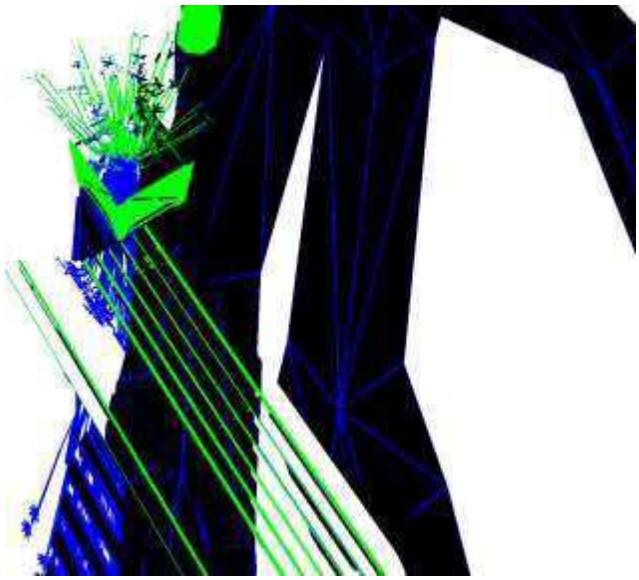


Figure 45 - Worst Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	195282 76,31%	179985 70,34%	168381 65,80%	161806 63,23%
	Pixels in Shadow	60612 23,69%	75909 29,66%	87513 34,20%	94088 36,77%
	Light - Correct	151058 77,35%	151058 83,93%	151058 89,71%	151058 93,36%
	Light - Incorrect	44224 22,65%	28927 16,07%	17323 10,29%	10748 6,64%
	Shadow - Correct	60612 100,00%	75909 100,00%	87513 100,00%	94088 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1024 x 1024	Pixels in Light	781372 76,33%	719876 70,33%	673732 65,82%	647250 63,23%
	Pixels in Shadow	242253 23,67%	303749 29,67%	349893 34,18%	376375 36,77%

	Light - Correct	604276 77,34%	604276 83,94%	604276 89,69%	604276 93,36%
	Light - Incorrect	177096 22,66%	115600 16,06%	69456 10,31%	42974 6,64%
	Shadow - Correct	242253 100,00%	303749 100,00%	349893 100,00%	376375 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1678582 82,05	1532075 74,89%	1432109 70,00%	1374738 67,20%
	Pixels in Shadow	367257 17,95	513764 25,11%	613730 30,00%	671101 32,80%
	Light - Correct	1296685 77,25	1296685 84,64%	1296685 90,54%	1296685 94,32%
	Light - Incorrect	381897 22,75	235390 15,36%	135424 9,46%	78053 5,68%
	Shadow - Correct	367257 100,00	513764 100,00%	613730 100,00%	671101 100,00%
	Shadow - Incorrect	0 0,00	0 0,00%	0 0,00%	0 0,00%

Table 15 - SGSM Shadow Mapping without Adjacency results for the average case;

As seen in Table 15, there is an increase in the number of light pixels in the image than in the normal shadow mapping method, an average of %, in many cases. However, there is a decrease of the light pixels as the shadow map resolution is increased (~%). This is same problem showed BGSM Testing, the pixel size λ is inversely proportional to the height and width of the shadow map, as such λ will decrease if Θ is not increased as well, resulting in smaller expanses. This, however, caused the errors in the light to reduce slightly in the shadow maps (~%).

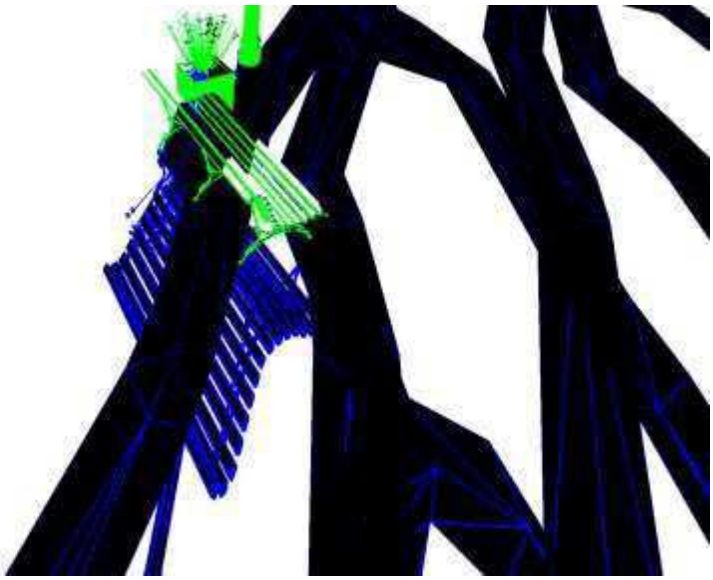


Figure 46 - Average Case Result for SGSM without Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.2.4 SGSM Shadow Mapping with Adjacency

Like in the previous test, the tests results showed here prove that the pixels in shadow in the SGSM are correct, but some cases, errors occur in the shadow map. However, the three cases shown here, this error are not very significant, even on the worse case.

The number of pixels to expand the triangles (Θ) in all tests was set to 1.

More test results are present in section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	181191 69,12%	174086 66,41%	171371 65,37%	170151 64,91%
	Pixels in Shadow	80953	88058	90773	91993

		30,88%	33,59%	34,63%	35,09%
	Light - Correct	168949 93,24%	168945 97,05%	168947 98,59%	168948 99,29%
	Light - Incorrect	12242 6,76%	5141 2,95%	2424 1,41%	1203 0,71%
	Shadow - Correct	80953 100,00%	88054 100,00%	90771 100,00%	91992 100,00%
	Shadow - Incorrect	0 0,00%	4 0,00%	2 0,00%	1 0,00%
1024 x 1024	Pixels in Light	724807 69,12%	696446 66,42%	685591 65,38%	680636 64,91%
	Pixels in Shadow	323769 30,88%	352130 33,58%	362985 34,62%	367940 35,09%
	Light - Correct	675789 93,24%	675768 97,03%	675782 98,57%	675787 99,29%
	Light - Incorrect	49018 6,76%	20678 2,97%	9809 1,43%	4849 0,71%
	Shadow - Correct	323769 100,00%	352109 99,99%	362978 100,00%	367938 100,00%
	Shadow - Incorrect	0 0,00%	21 0,01%	7 0,00%	2 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1446755 69,78%	1372853 66,21%	1345997 64,92%	1332759 64,28%
	Pixels in Shadow	626690 30,22%	700592 33,79%	727448 35,08%	740686 35,72%
	Light - Correct	1320285 91,26%	1320183 96,16%	1320259 98,09%	1320282 99,06%
	Light - Incorrect	126470 8,74%	52670 3,84%	25738 1,91%	12477 0,94%
	Shadow – Correct	626690 100,00%	700490 99,99%	727422 100,00%	740683 100,00%
	Shadow – Incorrect	0 0,00%	102 0,01%	26 0,00%	3 0,00%

Table 16 – SGSM Shadow Mapping with Adjacency results for the best case;

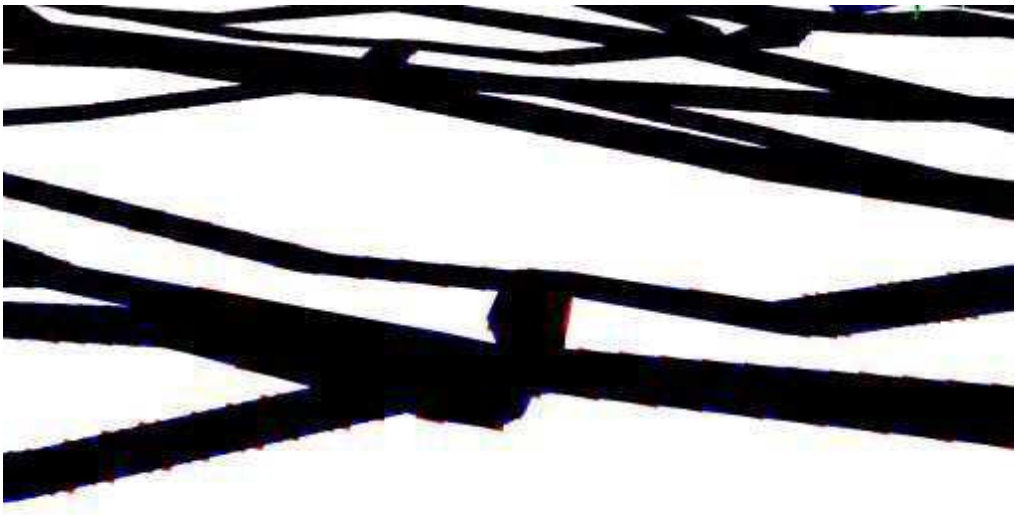


Figure 47 - Best Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096

Scene		Against			
Viewport		Flowers			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	150616 60,98%	144661 58,57%	140993 57,09%	139043 56,30%
	Pixels in Shadow	96370 39,02%	102325 41,43%	105993 42,91%	107943 43,70%
	Light - Correct	130890 86,90%	130889 90,48%	130890 92,83%	130888 94,13%
	Light - Incorrect	19726 13,10%	13772 9,52%	10103 7,17%	8155 5,87%
	Shadow - Correct	96370 100,00%	102324 100,00%	105993 100,00%	107941 100,00%
	Shadow - Incorrect	0 0,00%	1 0,00%	0 0,00%	2 0,00%
1024 x 1024	Pixels in Light	602335 60,98%	578718 58,59%	564043 57,10%	556039 56,29%
	Pixels in Shadow	385484 39,02%	409101 41,41%	423776 42,90%	431780 43,71%
	Light - Correct	523452	523436	523447	523450

		86,90%	90,45%	92,80%	94,14%
	Light - Incorrect	78883 13,10%	55282 9,55%	40596 7,20%	32589 5,86%
	Shadow - Correct	385484 100,00%	409085 100,00%	423771 100,00%	431778 100,00%
	Shadow - Incorrect	0 0,00%	16 0,00%	5 0,00%	2 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1526314 76,09%	1489878 74,27%	1457862 72,68%	1442713 71,92%
	Pixels in Shadow	479689 23,91%	516125 25,73%	548141 27,33%	563290 28,08%
	Light - Correct	1399264 91,68%	1399217 93,91%	1399241 95,98%	1399263 96,99%
	Light - Incorrect	127050 8,32%	90661 6,09%	58621 4,02%	43450 3,01%
	Shadow - Correct	479687 100,00%	516076 99,99%	548116 100,00%	563287 100,00%
	Shadow - Incorrect	2 0,00%	49 0,01%	25 0,00%	3 0,00%

Table 17 - SGSM Shadow Mapping with Adjacency results for the worst case;

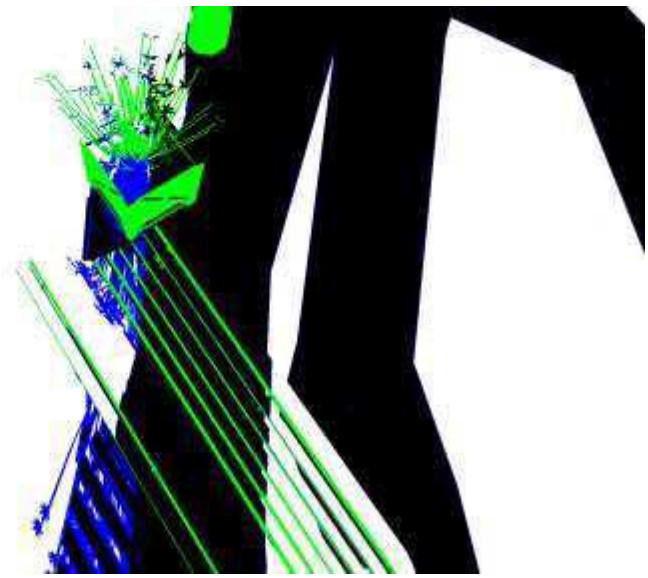


Figure 48 - Worst Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Pixels in Light	175437 68,56%	168081 65,68%	160927 65,80%	157355 63,23%
	Pixels in Shadow	80457 31,44%	87813 34,32%	94967 34,20%	98539 36,77%
	Light - Correct	151058 86,10%	151051 89,87%	151048 89,71%	151055 93,36%
	Light - Incorrect	24379 13,90%	17030 10,13%	9879 10,29%	6300 6,64%
	Shadow - Correct	80457 100,00%	87806 99,99%	94957 100,00%	98536 100,00%
	Shadow - Incorrect	0 0,00%	7 0,01%	10 0,00%	3 0,00%
1024 x 1024	Pixels in Light	701856 68,57%	672305 65,68%	643616 62,88%	629354 61,48%
	Pixels in Shadow	321769 31,43%	351320 34,32%	380009 37,12%	394271 38,52%
	Light - Correct	604276 86,10%	604252 89,88%	604250 93,88%	604262 96,01%
	Light - Incorrect	97580 13,90%	68053 10,12%	39366 6,12%	25092 3,99%
	Shadow - Correct	321769 100,00%	351296 99,99%	379983 99,99%	394257 100,00%
	Shadow - Incorrect	0 0,00%	24 0,01%	26 0,01%	14 0,00%
1920 x 1080 (FullHD)	Pixels in Light	1465035 71,61%	1413356 69,08%	1361801 66,56%	1335043 65,26%
	Pixels in Shadow	580804 28,39%	632483 30,92%	684038 33,44%	710796 34,74%
	Light - Correct	1296685 88,51%	1296622 91,74%	1296669 95,22%	1296664 97,13%
	Light - Incorrect	168350 11,49%	116734 8,26%	65132 4,78%	38379 2,87%
	Shadow - Correct	580804 100,00%	632420 99,99%	684022 100,00%	710775 100,00%
	Shadow - Incorrect	0 0,00%	63 0,01%	16 0,00%	21 0,00%

Table 18 - SGSM Shadow Mapping with Adjacency results for the average case;

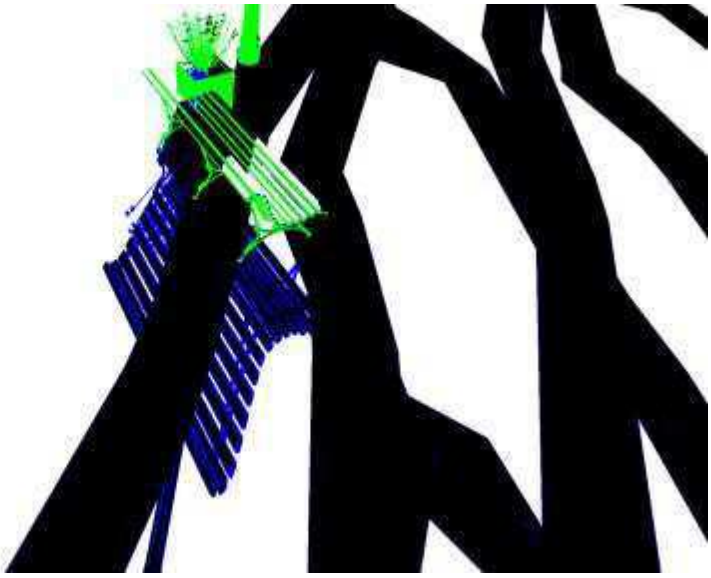


Figure 49 - Average Case Result for SGSM with Adjacency; Blue pixels represent the Light - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.2.5 Conservative Shadow Mapping without Adjacency

In these test, the full algorithm present in Hertel's Work is implemented: the first step the render the shadows with the information of the BGSM and SGSM (without adjacency), and identifying the light, shadow and "uncertain" that they can determine. Following this step, a ray is created for each "uncertain" pixel and sent to the ray-tracer to trace that ray.

The light and shadow pixels found during the first step are not showed in these tables because:

- the number of light pixels found is equal to the number of light pixels found in the SGSM (without adjacency);
- the number of shadow pixels found is equal to the number of shadow pixels found in the BGSM.

This occurs since these are the ones where both shadow maps agree, i.e. the correct light pixels in the BGSM also exist in the SGSM, and the correct shadow pixels in the SGSM also exist in the BGSM. So the "uncertain" pixels are created in the gap between the light pixels of the SGSM and the shadow pixels in the BGSM. This is further analysed in the following tables.

Like before, the number of pixels to expand the triangles (Θ) in all tests was set to 1.

For more detailed information of all the tests made, consult section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - “Uncertain”	10236 6,06%	8538 5,05%	4378 2,59%	2173 1,29%
	Shadow - “Uncertain”	40255 43,19%	21276 22,83%	10924 11,72%	5596 6,00%
	Light - Correct	168949 100,00%	168949 100,00%	1168949 100,00%	168949 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	93195 100,00%	93195 100,00%	93195 100,00%	93195 100,00%
	Shadow - Incorrect	0 0,00%	4 0,00%	2 0,00%	0 0,00%
1024 x 1024	Light - “Uncertain”	40958 6,06%	34113 5,05%	17445 2,58%	8676 1,28%
	Shadow - “Uncertain”	161157 43,23%	85225 22,86%	43737 11,73%	22313 5,99%
	Light - Correct	675789 100,00%	675789 100,00%	675789 100,00%	675789 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	675789 100,00%	675789 100,00%	675789 100,00%	675789 100,00%
	Shadow - Incorrect	0 0,00%	21 0,01%	7 0,00%	2 0,00%
1920 x 1080 (FullHD)	Light - “Uncertain”	103196 7,82%	87300 6,61%	44580 3,38%	22262 1,69%
	Shadow - “Uncertain”	368651 48,95%	199431 26,48%	103342 13,72%	52790 7,01%
	Light - Correct	1320285 100,00%	1320285 100,00%	1320285 100,00%	1320285 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%

	Shadow - Correct	753160 100,00%	753160 100,00%	753160 100,00%	753160 100,00%
	Shadow - Incorrect	0 0,00%	102 0,01%	26 0,00%	3 0,00%

Table 19 – Conservative Shadow Mapping without Adjacency results for the best case;

Figure 50 - Best Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Flowers			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - “Uncertain”	15491 11,84%	14669 11,21%	8566 6,54%	4577 3,49%
	Shadow - “Uncertain”	44849 38,63%	28189 24,28%	17928 15,45%	12391 10,68%
	Light - Correct	130890 100,00%	130890 100,00%	130890 99,96%	130890 99,92%
	Light - Incorrect	0 0,00%	5 0,00%	51 0,04%	111 0,08%
	Shadow - Correct	116096	116091	116045	115985

		100,00%	100,00%	100,00%	100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1024 x 1024	Light - “Uncertain”	61938 11,83%	58356 11,15%	34240 6,54%	18378 3,51%
	Shadow - “Uncertain”	179622 38,68%	112884 24,31%	72022 15,52%	49509 10,67%
	Light - Correct	523452 100,00%	523452 99,99%	523452 99,96%	523452 99,91%
	Light - Incorrect	0 0,00%	29 0,01%	209 0,04%	459 0,09%
	Shadow - Correct	464367 100,00%	464338 100,00%	464158 100,00%	463908 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1920 x 1080 (FullHD)	Light - “Uncertain”	118376 8,46%	109903 7,85%	60529 4,33%	33109 2,37%
	Shadow - “Uncertain”	311409 51,33%	193607 31,91%	116727 19,24%	74568 12,30%
	Light - Correct	1399266 100,00%	1399266 100,00%	1399266 99,99%	1399266 99,97%
	Light - Incorrect	0 0,00%	0 0,00%	84 0,01%	400 0,03%
	Shadow - Correct	606737 100,00%	606737 100,00%	606653 100,00%	606337 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%

Table 20 - Conservative Shadow Mapping without Adjacency results for the worst case;

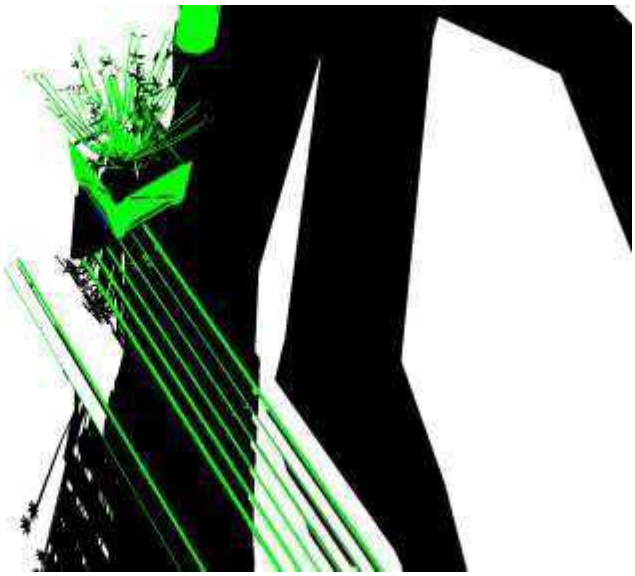


Figure 51 - Worst Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - “Uncertain”	10471 6,93%	9553 6,32%	5423 3,59%	3172 2,10%
	Shadow - “Uncertain”	44224 42,18%	28926 27,59%	17322 16,52%	10746 10,25%
	Light - Correct	151058 100,00%	151058 100,00%	151058 100,00%	151058 100,00%
	Light - Incorrect	0 0,00%	1 0,00%	1 0,00%	2 0,00%
	Shadow - Correct	104836 100,00%	104835 100,00%	104835 100,00%	104834 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1024 x	Light - “Uncertain”	41859 6,93%	38164 6,32%	21908 3,63%	12707 2,10%

1024	Shadow - “Uncertain”	177096 42,23%	115596 27,57%	69452 16,56%	42967 10,25%
	Light - Correct	604276 100,00%	604276 100,00%	604276 100,00%	604276 100,00%
	Light - Incorrect	0 0,00%	4 0,00%	4 0,00%	7 0,00%
	Shadow - Correct	419349 100,00%	419345 100,00%	419345 100,00%	419342 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
1920 x 1080 (FullHD)	Light - “Uncertain”	96939 7,48%	83989 6,48%	46011 3,55%	26135 2,02%
	Shadow - “Uncertain”	381897 50,98%	235390 31,42%	135423 18,08%	78046 10,42%
	Light - Correct	1296685 100,00%	1296685 100,00%	1296685 100,00%	1296685 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	1 0,00%	7 0,00%
	Shadow - Correct	749154 100,00%	749154 100,00%	749153 100,00%	749147 100,00%
	Shadow - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%

Table 21 - Conservative Shadow Mapping without Adjacency results for the average case;

Figure 52 - Average Case Result for CSM without Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.2.6 Conservative Shadow Mapping with Adjacency

In these test, the full algorithm we presented is implemented: the first step the render the shadows with the information of the BGSM and SGSM (without adjacency), and identifying the light, shadow and "uncertain" that they can determine. Following this step, a ray is created for each "uncertain" pixel and sent to the ray-tracer to trace that ray.

The light and shadow pixels found during the first step are not showed in these tables because:

- the number of light pixels found is equal to the number of light pixels found in the SGSM (without adjacency);
- the number of shadow pixels found is equal to the number of shadow pixels found in the BGSM.

This occurs since these are the ones where both shadow maps agree, i.e. the correct light pixels in the BGSM also exist in the SGSM, and the correct shadow pixels in the SGSM also exist in the

BGSM. So the "uncertain" pixels are created in the gap between the light pixels of the SGSM and the shadow pixels in the BGSM. This is further analysed in the following tables.

Like before, the number of pixels to expand the triangles (Θ) in all tests was set to 1.

For more detailed information of all the tests made, consult section XX in the Appendix

Scene		Side			
Viewport		Trees			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - "Uncertain"	10236 6,06%	8534 5,05%	4376 2,59%	2172 1,29%
	Shadow - "Uncertain"	12242 13,14%	5141 5,52%	2424 2,60%	1203 1,29%
	Light - Correct	168949 100,00%	168945 100,00%	168947 100,00%	168948 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	93195 100,00%	93195 100,00%	93195 100,00%	93195 100,00%
	Shadow - Incorrect	0 0,00%	4 0,00%	2 0,00%	1 0,00%
1024 x 1024	Light - "Uncertain"	40958 6,06%	34092 5,04%	17438 2,58%	8674 1,28%
	Shadow - "Uncertain"	49018 13,15%	20678 5,55%	9809 2,63%	4849 1,30%
	Light - Correct	675789 100,00%	675768 100,00%	675782 100,00%	675787 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	372787 100,00%	372787 99,99%	372787 100,00%	372787 100,00%
	Shadow - Incorrect	0 0,00%	21 0,01%	7 0,00%	2 0,00%
1920 x 1080 (FullHD)	Light - "Uncertain"	103196 7,82%	87198 6,60%	44554 3,37%	22259 1,69%
	Shadow - "Uncertain"	126470 16,79%	52670 6,99%	25738 3,42%	12477 1,66%
	Light - Correct	1320285	1320183	1320259	1320282

		100,00%	100,00%	100,00%	100,00%
	Light - Incorrect	0 0,00%	0 0,00%	0 0,00%	0 0,00%
	Shadow - Correct	753160 100,00%	753160 99,99%	753160 100,00%	753160 100,00%
	Shadow - Incorrect	0 0,00%	102 0,01%	26 0,00%	3 0,00%

Table 22 - Conservative Shadow Mapping with Adjacency results for the best case;

Figure 53 - Best Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Flowers			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - “Uncertain”	15491 11,84%	14668 11,21%	8566 6,54%	4575 3,49
	Shadow - “Uncertain”	19726 16,99%	13767 11,86%	10052 8,66%	8044 6,94
	Light - Correct	130890 100,00%	130889 100,00%	130890 99,96%	130888 99,92

	Light - Incorrect	0 0,00%	5 0,00%	51 0,04%	111 0,08
	Shadow - Correct	116096 100,00%	116091 100,00%	116045 100,00%	115985 100,00
	Shadow - Incorrect	0 0,00%	1 0,00%	0 0,00%	2 0,00%
1024 x 1024	Light - “Uncertain”	61938 11,83%	58340 11,15%	34235 6,54%	18376 3,51%
	Shadow - “Uncertain”	78883 16,99%	55253 11,90%	40387 8,70%	32130 6,93%
	Light - Correct	523452 100,00%	523436 99,99%	523447 99,96%	523450 99,91%
	Light - Incorrect	0 0,00%	29 0,01%	209 0,04%	459 0,09%
	Shadow - Correct	464367 100,00%	464338 100,00%	464158 100,00%	463908 100,00%
	Shadow - Incorrect	0 0,00%	16 0,00%	5 0,00%	2 0,00%
1920 x 1080 (FullHD)	Light - “Uncertain”	118374 8,46%	109854 7,85%	119041 4,32%	33106 2,37%
	Shadow - “Uncertain”	127050 20,94%	90661 14,94%	60504 9,65%	43050 7,10%
	Light - Correct	1399264 100,00%	1399217 100,00%	1399241 99,99%	1399263 99,97%
	Light - Incorrect	0 0,00%	0 0,00%	84 0,01%	400 0,03%
	Shadow - Correct	606737 100,00%	606737 99,99%	606653 100,00%	606337 100,00%
	Shadow - Incorrect	2 0,00%	49 0,01%	25 0,00%	3 0,00%

Table 23 - Conservative Shadow Mapping with Adjacency results for the worst case;

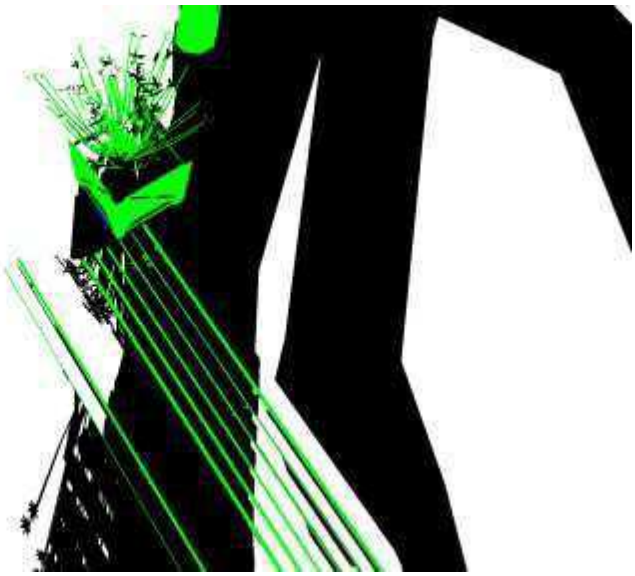


Figure 54 - Worst Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

Scene		Against			
Viewport		Bench			
Viewport Size	Pixel Types	Shadow Map Size			
		512^2	1024^2	2048^2	4096^2
512 x 512	Light - “Uncertain”	10471 6,93%	9546 6,32%	5413 3,58%	3169 2,10%
	Shadow - “Uncertain”	24379 23,25%	17029 16,24%	9878 9,42%	6298 6,01%
	Light - Correct	151058 100,00%	151051 100,00%	151048 100,00%	151055 100,00%
	Light - Incorrect	0 0,00%	1 0,00%	1 0,00%	2 0,00%
	Shadow - Correct	104836 100,00%	104835 99,99%	104835 99,99%	104834 100,00%
	Shadow - Incorrect	0 0,00%	7 0,01%	10 0,01%	3 0,00%

1024 x 1024	Light - “Uncertain”	41859 6,93%	38140 6,31%	21882 3,62%	12693 2,10%
	Shadow - “Uncertain”	97580 23,27%	68049 16,23%	39362 9,39%	25085 5,98%
	Light - Correct	604276 100,00%	604252 100,00%	604250 100,00%	604262 100,00%
	Light - Incorrect	0 0,00%	4 0,00%	4 0,00%	7 0,00%
	Shadow - Correct	419349 100,00%	419345 99,99%	419345 99,99%	419342 100,00%
	Shadow - Incorrect	0 0,00%	24 0,01%	26 0,01%	14 0,00%
1920 x 1080 (FullHD)	Light - “Uncertain”	96939 7,48%	83926 6,47%	45995 3,55%	26114 2,01%
	Shadow - “Uncertain”	168350 22,47%	116734 15,58%	65131 8,69%	38372 5,12%
	Light - Correct	1296685 100,00%	1296622 100,00%	1296669 100,00%	1296664 100,00%
	Light - Incorrect	0 0,00%	0 0,00%	1 0,00%	7 0,00%
	Shadow - Correct	749154 100,00%	749154 99,99%	749153 99,99%	749147 100,00%
	Shadow - Incorrect	0 0,00%	63 0,01%	16 0,01%	21 0,00%

Table 24 - Conservative Shadow Mapping with Adjacency results for the average case;

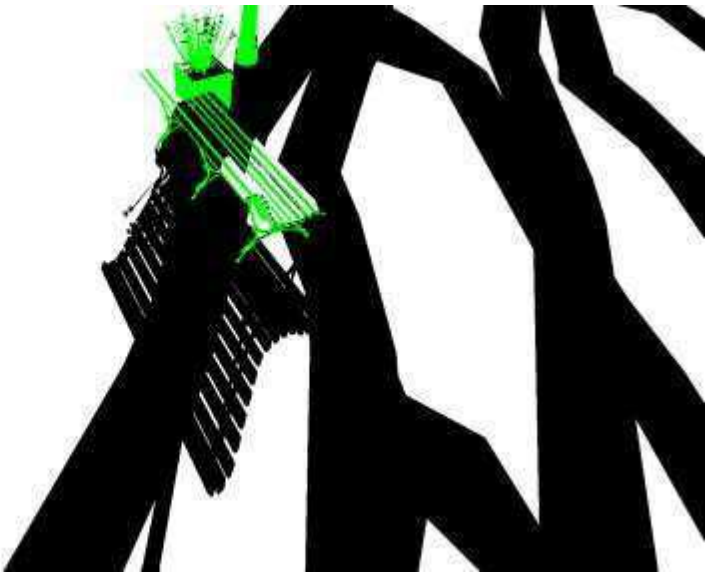


Figure 55 - Average Case Result for CSM with Adjacency; Blue pixels represent the Light - Incorrect and the Red pixels represent the Shadow - Incorrect; Viewport Size: 1920x1080; Shadow Map Size: 4096x4096.

5.3 Perfomance Testing

In this section, we'll compare the execution time of the conservative shadow mapping algorithms, Herel's and ours. We'll first analyse the computation cost involved in the creation of the shadow map texture, for that will compare the algorithms to standard shadow mapping process. Then, we'll analyse the computation costs of traced rays that originate with "uncertain" pixels. We'll use the execution time of the OptiX Prime to set a comparison between the pure ray tracing method and these hybrid approaches.

The examples showed here are same as in the previous sections, following the same parameters. For more tables of the test performed, consult section XX of the Appendix

5.3.1 Best Case

Table 25, shows the amount of time required for the creation of the shadow map for the best case test (Side-Trees). As expected, the conservative shadow maps required more time to create the

shadow map, due to the calculations of the geometry shader, resulting in an average increase of 2ms in the CSM without Adjacency (CSMa) and an average increase of 5ms in the CSM with Adjacency (CSMA).

There is also a great difference of execution time between the CSMa and the CSMA of about 4ms, on average. Due to the amount of additional geometry created to take account the adjacent triangles. It's natural that this difference occurs between the two, although at higher viewport resolutions the differences tend to become less severe.

Viewport Size	SM size	Execution time (ms)		
		NSM	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	1,57	2,31	6,38
	1024x1024	1,48	1,95	6,22
	2048x2048	1,68	2,28	6,83
	4096x4096	1,39	2,01	6,47
1024 x 1024	512x512	2,12	3,06	7,35
	1024x1024	2,09	2,96	7,84
	2048x2048	2,54	3,01	8,52
	4096x4096	2,48	2,93	9,25
1920 x 1080	512x512	3,11	3,76	8,98
	1024x1024	3,05	3,59	9,13
	2048x2048	15,34	16,37	16,76
	4096x4096	3,34	3,91	10,65

Table 25 – Execution times of the shadow map rendering for the best case (Side-Trees); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

Table 26, demonstrates the amount of time required for the ray tracing step for the best case test (Side-Trees). Since the pure ray tracing will render all the pixels in the viewport, these results will demonstrate the differences of execution time to render the "uncertain" pixels vs. all the pixels. As showed, there is a gradual difference between the pure and hybrid (CSMa & CSMA) method, that increases as the viewport size. There is also a significant difference between the CSMa nad CSMA, while in 512x512 viewport the execution times of the CSMa are lower than CSMA, while the larger viewports the execution times of the CSMA are lower or almost equal to those obtained in the

CSMa. This proves the amount of "uncertain" pixels in the tears obtained in the CSMa hinder it's performance.

Viewport Size	SM size	Execution time (ms)		
		Optix Prime	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	14,67	12,47	11,51
	1024x1024	14,14	8,52	11,29
	2048x2048	14,48	7,06	11,54
	4096x4096	14,1	7,92	11,06
1024 x 1024	512x512	48,67	41,12	32,37
	1024x1024	48,58	28,4	25,75
	2048x2048	50,19	22,2	23,61
	4096x4096	49,11	19,24	23,51
1920 x 1080	512x512	98,3	90,69	64,81
	1024x1024	128,75	60,4	54,59
	2048x2048	136,82	49,35	48,17
	4096x4096	128,36	41,77	44,73

Table 26 - Execution times of the ray tracing step for the best case (Side-Trees); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

The following graphics (Figures 56-58) show a visual representation of these differences showed in Tables 25 and 26.

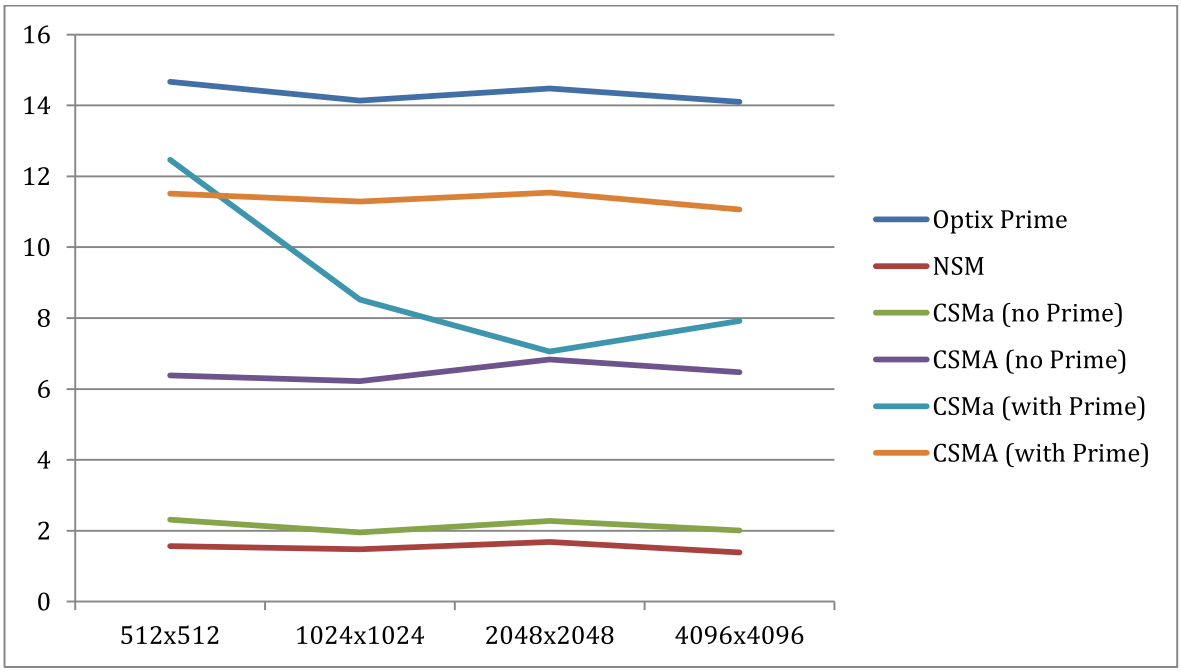


Figure 56 – Graphical representation of the execution time in the 512x512 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

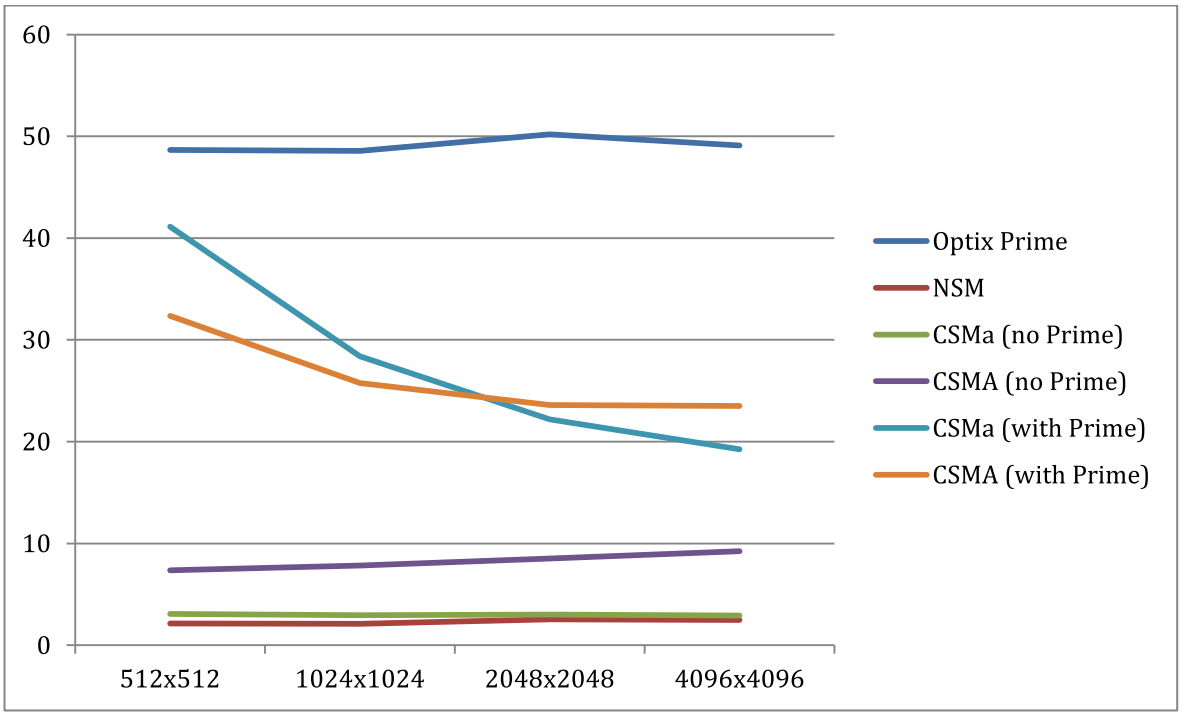


Figure 57 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

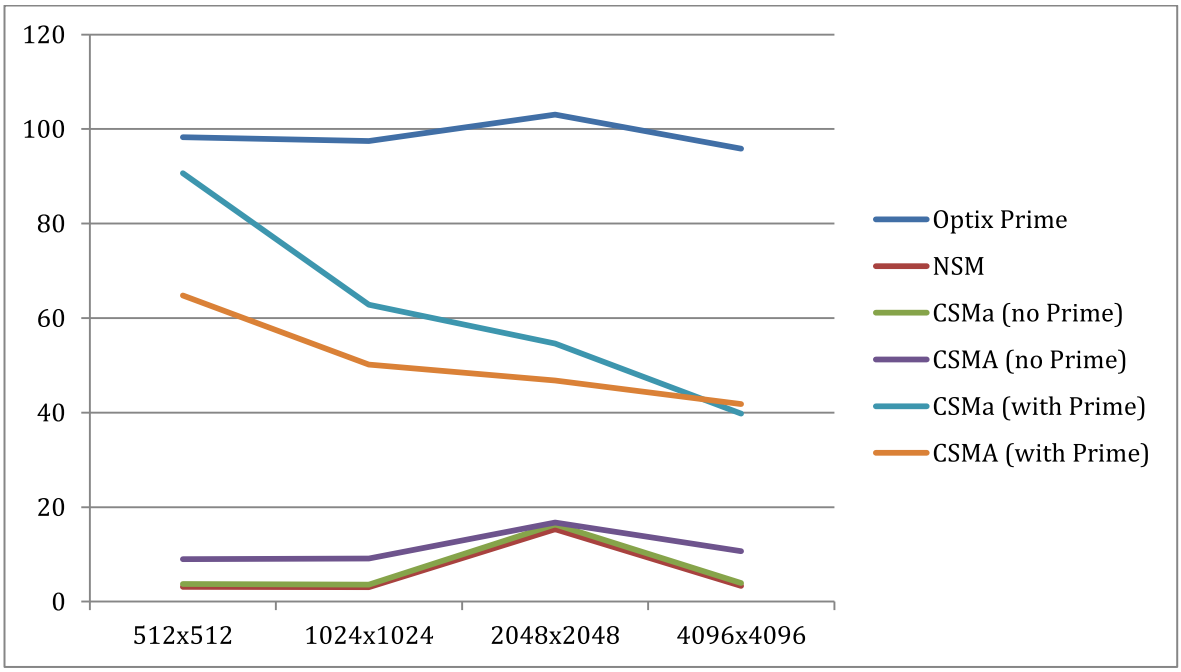


Figure 58 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 25 and 26; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

5.3.2 Worst Case

Table 27, shows the amount of time required for the creation of the shadow map for the worst case test (Against-Flowers). As expected, the conservative shadow maps required more time to create the shadow map, due to the calculations of the geometry shader, resulting in an average increase of 2ms in the CSM without Adjacency (CSMa) and an average increase of 5ms in the CSM with Adjacency (CSMA).

There is also a great difference of execution time between the CSMa and the CSMA of about 4ms, on average. Due to the amount of additional geometry created to take account the adjacent triangles. It's natural that this difference occurs between the two, although at higher viewport resolutions the differences tend to become less severe.

Viewport Size	SM Size	Execution time (ms)		
		NSM	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	1,73	2,41	6,60
	1024x1024	1,58	2,03	6,48
	2048x2048	1,53	2,11	6,94
	4096x4096	1,48	2,11	6,97
1024 x 1024	512x512	2,37	2,98	7,78
	1024x1024	2,36	3,18	8,31
	2048x2048	2,67	3,16	8,91
	4096x4096	2,65	3,26	10,06
1920 x 1080	512x512	3,32	4,05	9,70
	1024x1024	3,49	4,02	9,51
	2048x2048	16,26	16,42	16,8
	4096x4096	3,35	3,88	11,00

Table 27 - Execution times of the shadow map rendering for the worst case (Against-Flowers); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

Table 28, demonstrates the amount of time required for the ray tracing step for the worst case test (Against-Flowers). Since the pure ray tracing will render all the pixels in the viewport, these results will demonstrate the differences of execution time to render the "uncertain" pixels vs. all the pixels. As showed, there is a gradual difference between the pure and hybrid (CSMa & CSMA) method, that increases as the viewport size. There is also a significant difference between the CSMa and

CSMA, while in 512x512 viewport the execution times of the CSMa are lower than CSMA, while the larger viewports the execution times of the CSMA are lower or almost equal to those obtained in the CSMa. This proves the amount of "uncertain" pixels in the tears obtained in the CSMa hinder it's performance.

Viewport Size	SM size	Execution time (ms)		
		Optix Prime	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	28,24	21,62	20,50
	1024x1024	27,72	16,88	19,44
	2048x2048	27,45	15,6	17,26
	4096x4096	27,07	14,5	18,46
1024 x 1024	512x512	84,81	66,86	57,2
	1024x1024	86,24	55,02	48,69
	2048x2048	84,89	44,74	45,29
	4096x4096	86,75	38,77	42,41
1920 x 1080	512x512	127,23	107,23	85,8
	1024x1024	128,75	60,4	54,59
	2048x2048	136,82	49,35	48,17
	4096x4096	128,36	41,77	44,73

Table 28 – Execution times of the ray tracing step for the worst case (Against-Flowers); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

The following graphics (Figures 59-61) show a visual representation of these differences showed in Tables 27 and 28.

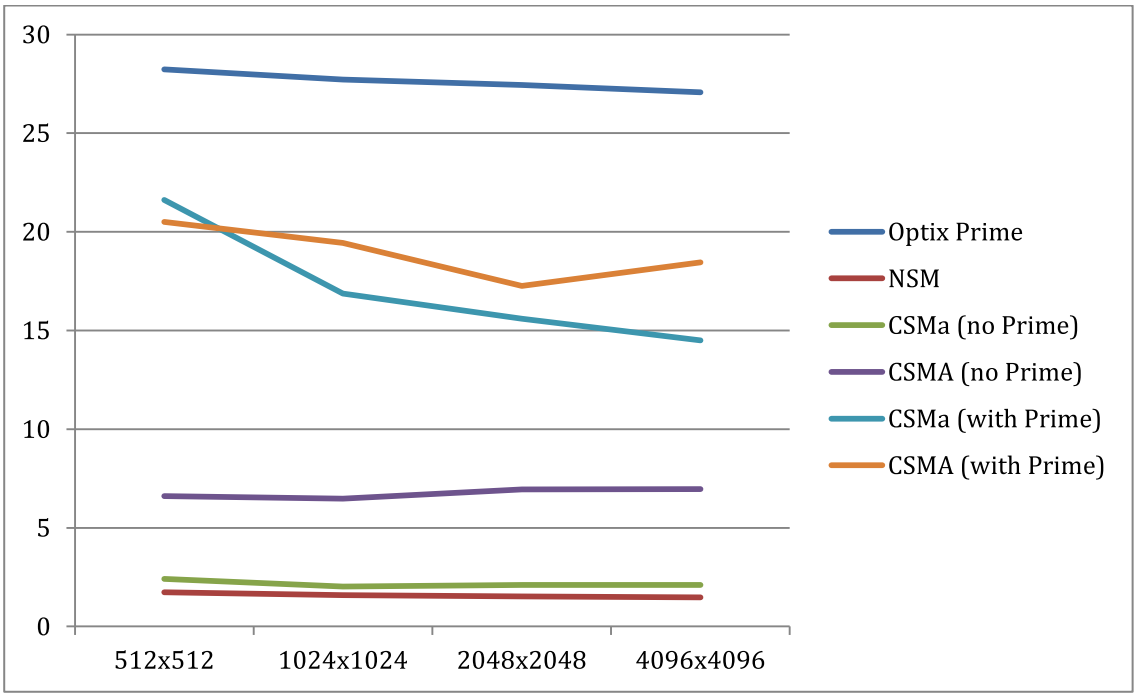


Figure 59 – Graphical representation of the execution time in the 512x512 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

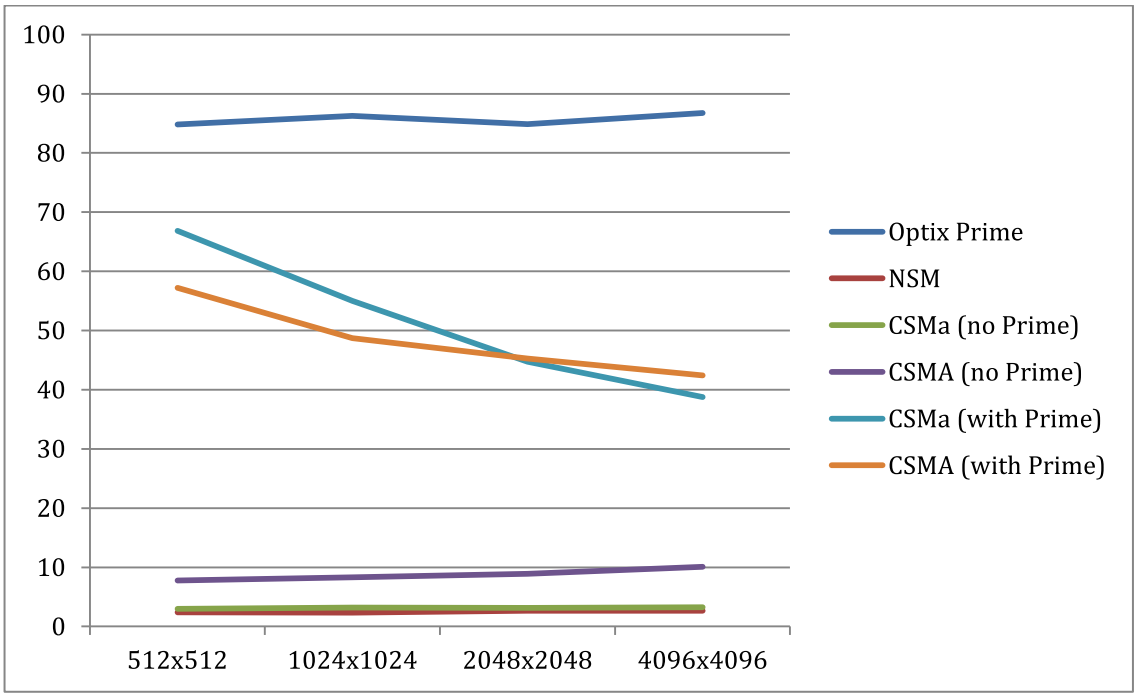


Figure 60 – Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

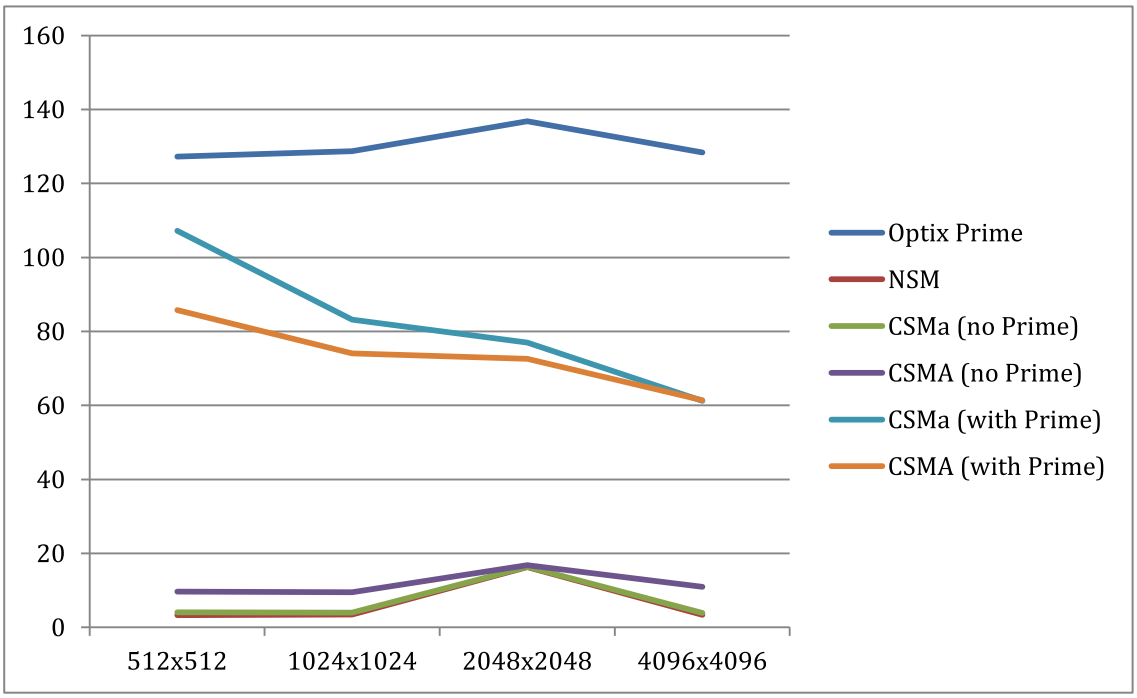


Figure 61 - Graphical representation of the execution times in the 1920x1080 viewport, showed in tables 27 and 28; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

5.3.3 Average Case

Table 29, shows the amount of time required for the creation of the shadow map for the average case test (Against-Bench). As expected, the conservative shadow maps required more time to create the shadow map, due to the calculations of the geometry shader, resulting in an average increase of 2ms in the CSM without Adjacency (CSMa) and an average increase of 5ms in the CSM with Adjacency (CSMA).

There is also a great difference of execution time between the CSMa and the CSMA of about 4ms, on average. Due to the amount of additional geometry created to take account the adjacent triangles. It's natural that this difference occurs between the two, although at higher viewport resolutions the differences tend to become less severe.

Viewport Size	SM size	Execution time (ms)		
		NSM	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	1,63	2,31	6,21
	1024x1024	1,55	2,14	6,34
	2048x2048	1,44	2,11	7,03
	4096x4096	1,50	2,08	6,79
1024 x 1024	512x512	62,50	64,22	66,55
	1024x1024	2,22	2,72	3,85
	2048x2048	2,21	2,73	8,03
	4096x4096	2,12	3,11	9,41
1920 x 1080	512x512	3,21	3,91	9,41
	1024x1024	3,32	3,84	9,53
	2048x2048	3,29	4,04	10,17
	4096x4096	3,24	3,81	10,79

Table 29 - Execution times of the shadow map rendering for the average case (Against-Bench); NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

Table 30, demonstrates the amount of time required for the ray tracing step for the average case test (Against- Bench). Since the pure ray tracing will render all the pixels in the viewport, these results will demonstrate the differences of execution time to render the "uncertain" pixels vs. all the pixels. As showed, there is a gradual difference between the pure and hybrid (CSMa & CSMA) method, that increases as the viewport size. There is also a significant difference between the CSMa and CSMA, while in 512x512 viewport the execution times of the CSMa are lower than CSMA, while the larger viewports the execution times of the CSMA are lower or almost equal to those obtained in the CSMa. This proves the amount of "uncertain" pixels in the tears obtained in the CSMa hinder its performance.

Viewport Size	SM size	Execution time (ms)		
		Optix Prime	CSMa (no Prime)	CSMA (no Prime)
512 x 512	512x512	18,4	14,91	16,36
	1024x1024	17,69	13,05	15

	2048x2048	17,69	11,04	13,08
	4096x4096	17,93	10,91	14,3
1024 x 1024	512x512	61,31	106,5	105,07
	1024x1024	61,77	39,17	37,74
	2048x2048	61,17	31,95	32,76
	4096x4096	60,14	26,64	30,58
1920 x 1080	512x512	99,9	79,94	63,8
	1024x1024	100,29	60,4	54,59
	2048x2048	100,29	49,35	48,17
	4096x4096	101,38	41,77	44,73

Table 30 – Execution times of the ray tracing step for the average case (Against-Bench); CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

The following graphics (Figures 62-64) show a visual representation of these differences showed in Tables 29 and 30.

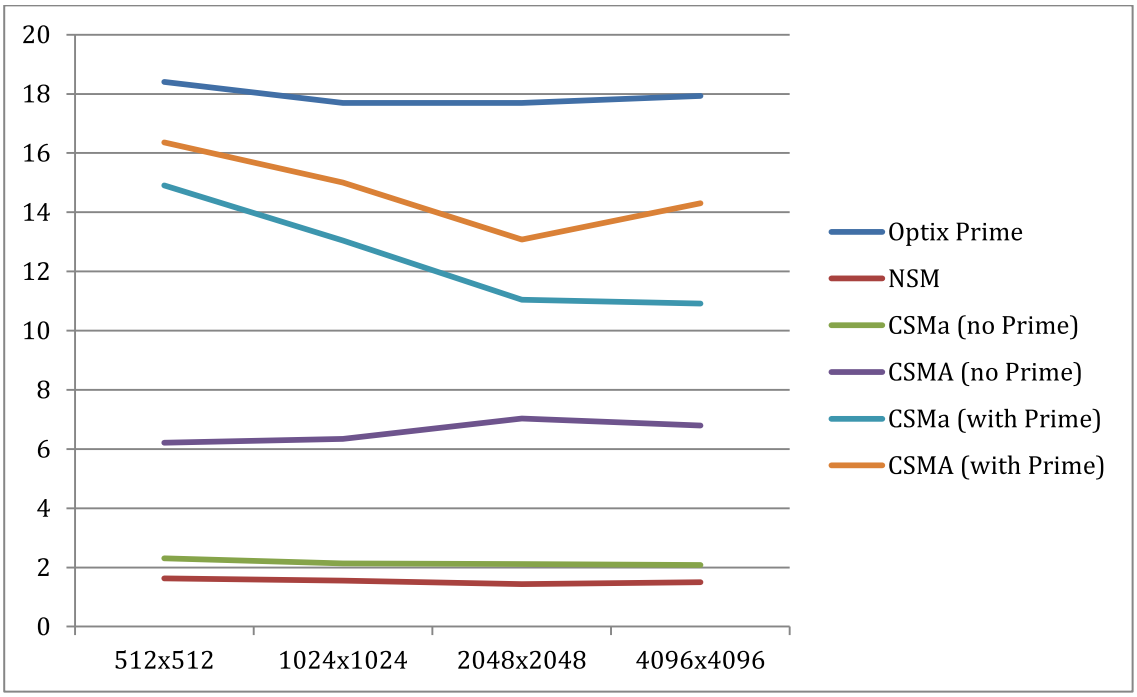


Figure 62 – Graphical representation of the execution time in the 512x512 viewport, showed in

tables 29 and 30; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

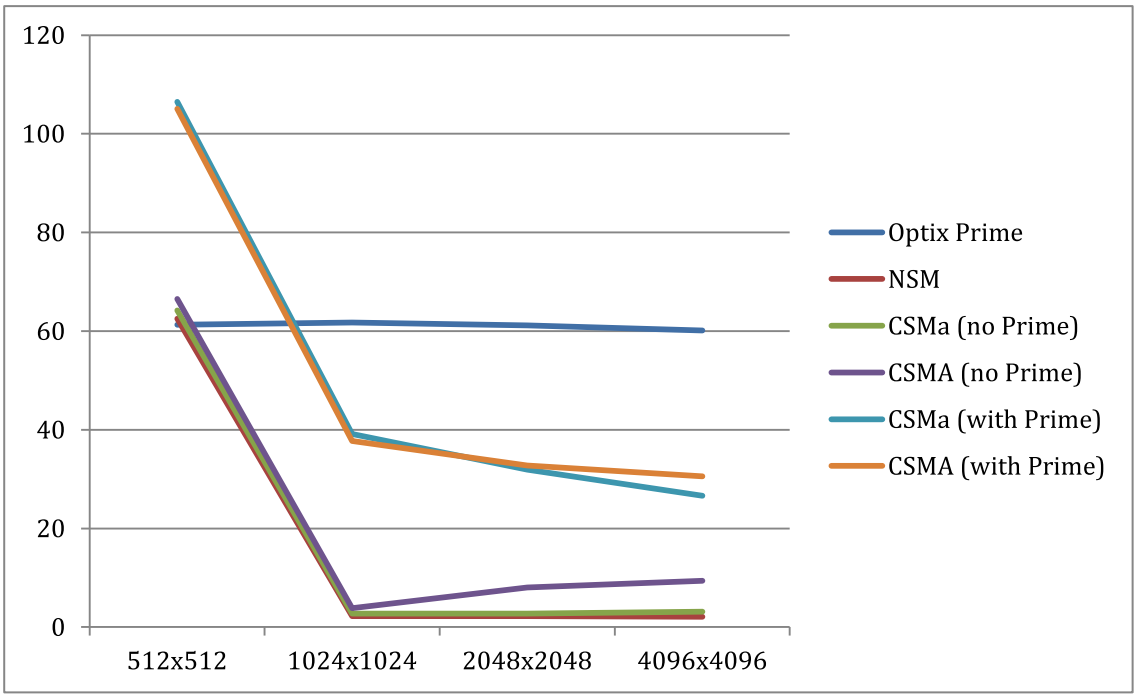


Figure 63 - Graphical representation of the execution time in the 1024x1024 viewport, showed in tables 29 and 30; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

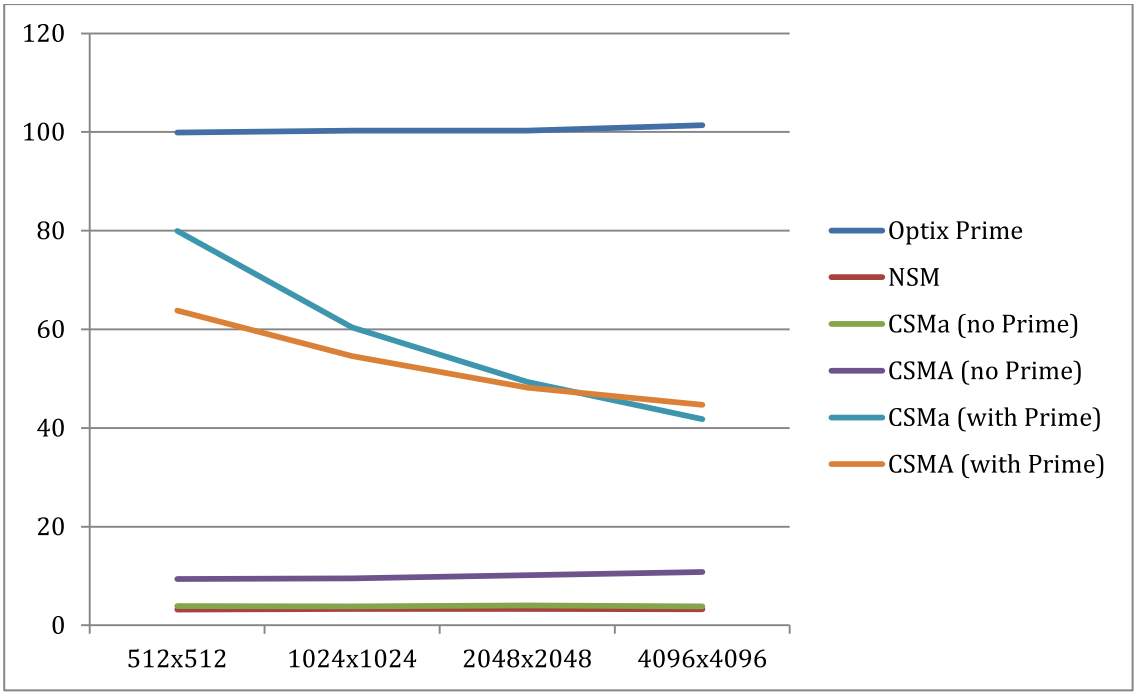


Figure 64 - Graphical representation of the execution time in the 1920x1080 viewport, showed in tables 30 and 31; NSM: Normal Shadow Mapping; CSMa: Conservative Shadow Mapping without Adjacency; CSMA: Conservative Shadow Mapping with Adjacency.

6 Conclusions and Future Work

The rendering of 3D images using pure ray tracing techniques is beyond the reach of current video cards, but certain hybrid ray tracing algorithms have been implemented alongside rasterization processes to produce high-quality images. This allows for the creation of accurate shadows, however shadow mapping techniques still the most time efficient methods to implement, and with many improvements made to the original algorithm, the allow deceptively accurate shadows.

We presented two version of a hybrid algorithm, Conservative Shadow Mapping, where the rasterization produces two shadow maps, with larger (BGSM) and smaller (SGSM) triangles, to determine problematic or "uncertain" pixels in the shadow map, and send these pixels to the ray tracer to correct them. The main difference between versions comes in the SGSM, the algorithm we developed takes account the adjacency information supplied by the geometry shader. Although these algorithms provide approximately the same performance and image quality, there is impact in certain steps, depending is the adjacency information is used or not. Without the adjacency information, more "uncertain" pixels are created, which increased the amount of rays to be created and process in the ray tracer, increasing its execution time. With the adjacency information the number of rays crated is reduces, resulting in a decrease in the execution time, but the creation of the shadow map is delayed to take account all the adjacency cases of the triangles.

7 Bibliography

[1] M. Pharr and G. Humphreys, Physically Based Rendering: From Theory to Implementation, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[2] NVIDIA Corporation, "NVIDIA® OptiX™," [Online]. Available: http://www.nvidia.com/object/optix.html. [Accessed 25 October 2013].

[3] NVIDIA Corporation, "NVIDIA® OptiX™ Ray Tracing Engine," [Online]. Available: https://developer.nvidia.com/optix. [Accessed 25 October 2013].

[4] NVIDIA Research, "Interactive Ray Tracing with CUDA," [Online]. Available: http://www.nvidia.com/content/nvision2008/tech_presentations/Game_Developer_Track/NVISION08-Interactive_Ray_Tracing.pdf. [Accessed 25 October 2013].

[5] J. Cabeleira, "Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time," [Online]. Available: http://www.voltaico.net/files/article.pdf. [Accessed 16 October 2013].

[6] J. Cabeleira and R. F. Prada, "Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time," Universidade Técnica de Lisboa, October 2010. [Online]. Available: https://dspace.ist.utl.pt/bitstream/2295/752012/1/masterThesis.pdf. [Accessed 16 October 2013].

[7] R. Cook, L. Carpenter e E. Catmull, "The Reyes Image Rendering Architecture," *International Conference on Computer Graphics and Interactive Techniques,* pp. 95-102, 1987.

[8] P. Christensen, J. Fong, D. Laur e D. Batali, "Ray Tracing For The Movie "Cars"," *IEEE Symposium on Interactive Ray Tracing,* pp. 1-6, 2006.

[9] S. Hertel, K. Hormann e R. Westermann, "A Hybrid GPU Rendering Pipline for Alias-Free Hard

Shadows," em *Proceedings of Eurographics 2009 Area Papers*, München, Germany, Eurographics Association, 2009, p. 59–66.

[10] A. V. Nealen e W. Heidrich, "Shadow Mapping and Shadow Volumes: Recent Developments in Real-Time," University of British Columbia, 2002.

[11] J. Abrantes e A. R. Fernandes, "Analysis and Proposal of a Shadow Mapping Remix Approach," University of Minho, 2009.

[12] R. Dimitrov, "Cascaded Shadow Maps," 2007.

[13] T. Martin e T.-S. Tan, "Anti-aliasing and Continuity with Trapezoidal Shadow Maps," em *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, Norrköping, Sweden, Eurographics Association, 2004, pp. 153-160.

[14] M. Stamminger e G. Drettakis, "Perspective Shadow Maps," *ACM Trans. Graph.,* vol. 21, n.º July 2002, pp. 557-562, 2002.

[15] F. Zhang, H. Sun, L. Xu e L. K. Lun, "Parallel-split Shadow Maps for Large-scale Virtual Environments," em *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*, Hong Kong, China, ACM, 2006, pp. 311-318.

[16] W. T. Reeves, D. H. Salesin e R. L. Cook, "Rendering Antialiased Shadows with Depth Maps," *SIGGRAPH Comput. Graph.,* vol. 21, n.º 4, pp. 283-291, 1987.

[17] D. L. Moderno and A. R. Fernandes, "Shadow Mapping and Ray-Tracing," University of Minho, 2011.

[18] M. Beister, M. Ernst e M. Stamminger, "A Hybrid GPU-CPU Renderer," University of Erlangen-Nuremberg, 2005.

[19] P. Shanmugam and O. Arikan, "Hardware Accelerated Ambient Occlusion Techniques on GPUs," in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, Seattle, Washington, ACM, 2007, pp. 73-80.

[20] M. Mittring, "Finding Next Gen: CryEngine 2," em *ACM SIGGRAPH 2007 Courses*, San Diego, California, ACM, 2007, pp. 97-121.

[21] C. Reinbothe, T. Boubekeur and M. Alexa, "Hybrid Ambient Occlusion," *EUROGRAPHICS 2009 Areas Papers,* 2009.

[22] J. Kopf, . M. F. Cohen, D. Lischinski and M. Uyttendaele, "Joint Bilateral Upsampling," in *ACM SIGGRAPH 2007 Papers*, vol. 26, San Diego, California, ACM, 2007, p. 96.

[23] B. De Greve, "Reflections and Refractions in Ray Tracing," 2007.

[24] R. Ramamoorthi and P. Hanrahan, "An Efficient Representation for Irradiance Environment Maps," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, 2001, pp. 497-500.

[25] R. Fernando e M. J. Kilgard, "Chapter 7 - Environment Mapping Techniques," em *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc., 2003, pp. 169-198.

[26] S. T. Davids e C. Wyman, "Interactive Refractions with Total Internal Reflection," em *Proceedings of Graphics Interface 2007*, Montreal, Canada, ACM, 2007, pp. 185-190.

[27] S. Prast and A. Fruehstueck, "Caustics, Light Shafts, God Rays," Vienna University of Technology, 2013.

[28] H. W. Jensen and P. H. Christensen, "Efficient Simulation of Light Transport in Scences with Participating Media Using Photon Maps," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 1998, pp. 311-320.

[29] J. T. Kajiya, "The Rendering Equation," *SIGGRAPH Comput. Graph.,* vol. 20, no. 4, pp. 143-150, 1986.

[30] X. Sun, K. Zhou, S. Lin e B. Guo, "Line Space Gathering for Single Scattering in Large Scenes," *ACM Trans. Graph.,* vol. 29, n.º 4, 2010.

[31] B. Tóth e T. Umenhoffer, "Real-time Volumetric Lighting in Participating Media," Technical University of Budapest, 2009.

[32] A. S. Glassner, E. Haines, P. Hanrahan, R. L. Cook, J. Arvo, D. Kirk e P. S. Heckbert, "An Introduction to Ray Tracing," *Academic Press New York,* 1989.

[33] I. Wald, "Realtime Ray Tracing and Interactive Global Ilumination," *Computer Graphics Group, Saarland University,* 2004.

[34] J. Bikker e J. v. Schijndel, "The Brigade Renderer: A Path Tracer for Real-Time Games," *International Journal of Computer Games Technology,* vol. 2013, 2013.

[35] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison e M. Stich, "OptiX: A General Purpose Ray Tracing Engine," *ACM Trans. Graph.,* vol. 29, n.º 4, pp. 1-13, July 2010.

[36] C. W. Everitt e M. J. Kilgard, "Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated," *CoRR,* vol. cs.GR/0301002, 2003.

[37] P. Rademacher, "Ray Tracing: Graphics for the Masses," [Online]. Available: http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html. [Accessed 31 October 2013].

[38] J. Bikker, "Arauna Real-Time Ray Tracing – Nhtv," 31 January 2008. [Online]. Available: http://igad.nhtv.nl/~bikker/. [Accessed 25 October 2013].

[39] D. Pohl, "Quake Wars* Gets Ray Traced," 2 July 2012. [Online]. Available: http://software.intel.com/en-us/articles/quake-wars-gets-ray-traced/. [Accessed 31 October 2013].

[40] S. Laine and T. Karras, "High-Performance Software Rasterization on GPUs," *High Perfomance Graphics 2011,* 2011.

[41] M. Mittring, "Chapter 8 Finding Next Gen – CryEngine 2," [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/Chapter8-Mittring-

Finding_NextGen_CryEngine2.pdf. [Accessed 22 January 2014].

[42] L. Bavoil and M. Sainz, "Multi-Layer Dual-Resolution Screen-Space Ambient," [Online]. Available: http://developer.download.nvidia.com/presentations/2009/SIGGRAPH/Bavoil_MultiLayerDualResolutionSSAO.pdf. [Accessed 22 January 2014].

[43] D. Blythe, "11.2 Environment Mapping," 6 August 1999. [Online]. Available: ftp://ftp.sgi.com/opengl/contrib/blythe/advanced99/notes/node175.html. [Accessed 18 January 2014].

[44] T. Kim, "Interactive Refractions And Caustics Using Image Space Techniques," 28 July 2009. [Online]. Available: http://www.slideshare.net/codevania/interactive-refractions-and-caustics-using-image-space-techniques-1777900. [Accessed 22 January 2014].

[45] D. Kvarfordt and B. Lillandt, "Screen Space Ambient Occlusion," [Online]. Available: http://www.cse.chalmers.se/edu/year/2013/course/TDA361/Advanced%20Computer%20Graphics/SSAO.pdf. [Accessed 22 January 2014].

[46] J. Bikker, "Real-time Ray Tracing through the Eyes of a Game Developer," em *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, Washington, DC, USA, IEEE Computer Society, 2007, pp. 1-10.

[47] M. Wrenninge and N. B. Zafar, "Volumetric Methods In VisualEffects," 8 October 2011. [Online]. Available: http://magnuswrenninge.com/content/pubs/ProductionVolumeRenderingFundamentals2011.pdf. [Accessed 5 February 2014].

Appendix