EFFICIENT HIGH-QUALITY SHADOW MAPPING

By

SAGAR BHANDARE

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2012
© 2012 Sagar Bhandare
To my parents
ACKNOWLEDGMENTS

I would like to thank my chair advisor, Dr. Jörg Peters for guiding me through all the research with valuable advice, support and patience. I am thankful to my supervisory committee members, Dr. Benjamin Lok and Dr. Jeffrey Ho for their feedback. Finally, I would like to thank my parents for their support and affection.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>6</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>7</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>8</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>9</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>9</td>
</tr>
<tr>
<td>1.2 iPASS</td>
<td>10</td>
</tr>
<tr>
<td>1.3 Overview</td>
<td>10</td>
</tr>
<tr>
<td>2 BACKGROUND</td>
<td>11</td>
</tr>
<tr>
<td>3 OPTIMIZATIONS TO SHADOW MAPPING</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Focus on Visible Pixels</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Focus on Visible Shadowed Pixels</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Uniform Depth Values</td>
<td>17</td>
</tr>
<tr>
<td>3.4 Near and Far Planes</td>
<td>18</td>
</tr>
<tr>
<td>4 IMPLEMENTATION</td>
<td>19</td>
</tr>
<tr>
<td>4.1 Generating the Control Texture</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Computing the Bounding Rectangle</td>
<td>19</td>
</tr>
<tr>
<td>4.3 Post-filtering</td>
<td>20</td>
</tr>
<tr>
<td>4.3.1 Bilinear Interpolation</td>
<td>20</td>
</tr>
<tr>
<td>4.3.2 PCF</td>
<td>20</td>
</tr>
<tr>
<td>4.4 Discussion</td>
<td>21</td>
</tr>
<tr>
<td>5 CONCLUSION</td>
<td>23</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>24</td>
</tr>
<tr>
<td>BIOGRAPHICAL SKETCH</td>
<td>26</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Performance comparison</td>
<td>16</td>
</tr>
<tr>
<td>4-1</td>
<td>Bounding rectangle computation</td>
<td>20</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Comparison of basic and frustum-adjusted shadows</td>
<td>9</td>
</tr>
<tr>
<td>2-1</td>
<td>Basic shadow mapping</td>
<td>11</td>
</tr>
<tr>
<td>3-1</td>
<td>Frustum tightening</td>
<td>15</td>
</tr>
<tr>
<td>3-2</td>
<td>Increasing shadow quality</td>
<td>16</td>
</tr>
<tr>
<td>4-1</td>
<td>Filters</td>
<td>21</td>
</tr>
<tr>
<td>4-2</td>
<td>Gallery of <em>Elephants Dream</em></td>
<td>22</td>
</tr>
</tbody>
</table>
Abstract of Thesis Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Master of Science

EFFICIENT HIGH-QUALITY SHADOW MAPPING

By
Sagar Bhandare

December 2012

Chair: Jörg, Peters
Major: Computer Engineering

Shadow mapping is a well-known technique for introducing hard shadows to a scene. It suffers, however, from severe aliasing artifacts such as magnified aliasing of the shadow boundaries.

The thesis represents an extension, including an implementation, of light frustrum adjustment to improve the shadow quality while maintaining its prime advantage of speed. The thesis improves the technique within a real-time animation environment.
Besides focus and perspective distortion, shadows provide important visual cues towards interpreting depth and action in movie scenes. Shadows are a vital component for depth perception in any scene. They define the relationship of objects with the light source and with each other.

1.1 Motivation

Figure 1-1. The Emo model in Elephants Dream [Blender Foundation, 2006] rendered using iPASS (see 1.2) and shadow mapping. A) Basic shadow mapping. B) Shadow mapping with light frustrum adjusted.

Shadow mapping is a well-known and widely used technique in real-time applications to introduce shadows to a scene. It is a topic of intensive research with hundreds of papers published on it (see Chapter 2 for an overview of the literature until now). This thesis' aims are twofold: A) reduce magnified aliasing of shadow boundaries in conventional shadow mapping, and B) provide an efficient GPU implementation of the method with real-time animation.
1.2 iPASS

A brief overview of iPASS is required since the development of this thesis has been done in conjunction with it. Efficient Pixel-Accurate Rendering of Curved Surfaces (iPASS) [Yeo et al., 2012] is a technique to determine optimal GPU tessellation factors so that smooth surfaces are rendered with no noticeable parametric distortion or polyhedral artifacts. Its primary advantage over the commonly used Reyes rendering framework [Cook et al., 1987] is its real-time performance, Reyes being the current industry standard for rendering smooth surfaces in commercial animation movies. iPASS, thus, raises the possibility of achieving cinematic-quality animation in an interactive setting.

1.3 Overview

Chapter 2 describes the problem and prior work on shadow mapping. Chapter 3 explains the key ideas behind the optimizations. These are namely, A) Focus on visible pixels, B) Focus on visible shadowed pixels, C) Uniform depth values and D) Near and far plane adjustment. An efficient GPU algorithm and an implementation are developed in Chapter 4.
CHAPTER 2
BACKGROUND

Shadow maps were introduced in 1978 [Williams, 1978]. Shadow maps are essentially depth maps from the light source. Initially the scene is rendered from the viewpoint of the light source and depth information of the closest fragment for each pixel is recorded in a texture. During the final rendering, each camera-visible pixel is projected back into light-space and a depth comparison is performed against the texture. If the pixel depth is greater than the depth value stored in the texel, the pixel is in shadow, otherwise it is lit.

Because the projection of camera-visible pixels does not conform with the sampling points where shadow map depth was recorded (see Fig. 2-1), we see two kinds of aliasing artifacts: jagged shadow boundaries and incorrect self shadowing. In fact, for the above method, no finite resolution would completely get rid of these problems since the problem is not the resolution itself but the mismatch between shadow map sample and query locations.

Figure 2-1. Shadow maps and aliasing
To address the problem, [Aila and Laine, 2004] and [Johnson et al., 2005] proposed sampling irregularly during shadow map generation. The scene is first rendered from the camera and all visible pixels are projected into light-space. These light-space locations are the locations which will be queried later on. The shadow map is then generated by sampling at exactly these points. Due to lack of hardware support for irregular sampling, a CPU pipeline was used. Further work by [Sintorn et al., 2008] resulted in a hardware implementation, and a later extension by [Pan et al., 2009] provided efficient anti-aliasing.

A large number of methods reparameterize, warp or transform, the scene so that higher sampling densities can be obtained where desired. There are multiple ways to do this. Perspective Shadow Maps [Stamminger and Drettakis, 2002] apply a global transformation along the $z$-axis to award higher sample space to objects close to the camera. A logarithmic transformation along $z$-axis provides an optimal sample rate as shown by [Wimmer et al., 2004], however it is impractical because logarithmic rasterization is required. Other practical warping schemes to redistribute samples towards the near plane [Chong and Gortler, 2004; Chong, 2003; Martin and Tan, 2004; Wimmer et al., 2004] use different perspective transformations. A recent technique by [Rosen, 2012] tries to adapt the warping locally according to scene content to produce a single warped shadow map. A rectilinear warping grid is used here to magnify areas of interest and minify the other parts of a scene. However, this method assumes that the scene primitives are small enough, since if they are not then a primitive, say a triangle, spanning across many grids should not remain a triangle after the warp but it is still rasterized incorrectly as a triangle.

According to [Eisemann et al., 2012], “While warping works very well in some configurations, especially if the light is overhead, there are other configurations where warping degenerates to uniform shadow mapping. A better alternative is to use more than one shadow map.” The basic idea here is to subdivide the view frustrum along
z-axis and render a separate shadow map for each partition. [Engel, 2006; Lloyd et al., 2006; Zhang, 1998] approach the problem along these lines. It has the same spirit of reparameterization along z-axis, however now each partition has a constant sampling density. The intention is to match the visible pixel density to the shadow map texel density. This density decreases as we move away from the camera along the z-axis. These set of approaches give more consistent results than global reparameterization but suffer from other problems such as sudden jumps in shadow quality along the boundaries of the partitions.

The above methods of shadow mapping are scene-independent. A different class of algorithms rely on scene analysis and use that data to refine the shadow map in an optimal way. Adaptive Shadow Maps (ASM) [Fernando et al., 2001] store a tree node in each texel of the shadow map with each node storing multiple samples. The number of samples or refinement are decided based on the camera view and the criteria used for importance sampling. [Arvo, 2004; Giegl and Wimmer, 2007; Lefohn et al., 2007] all use similar ideology to adjust the shadow map resolution. The trade-off with respect to warping methods is an additional scene analysis step. But these methods give better results in general cases.

Shadow ray casting has also been investigated for real-time shadows. In terms of shadow quality, shadow ray casting is considered the benchmark method for hard shadows and is a preferred choice for offline rendering. It does not suffer from the above aliasing problems, however, as with classical ray-tracing, it is not suitable for real-time applications. [Hertel et al., 2009] uses a hybrid GPU pipeline that performs rasterization for pixels that do not fall on shadow boundaries and then switches to ray-casting for the uncertain pixels on the shadow boundaries. [Olsson and Assarsson, 2011] and [Christian Lauterbach and Manocha, 2009] employ similar hybrid strategies to speed up shadow ray-casting. The approach still remains prohibitive for real-time applications.
CHAPTER 3
OPTIMIZATIONS TO SHADOW MAPPING

3.1 Focus on Visible Pixels

A light frustum that is not well-adjusted to current camera view can lead to huge wastage of available resolution and precision. And so we adjust the light frustum to focus only on that part of scene which is currently visible from the camera, [Brabec et al., 2000]. This simple idea can lead to a lot of improvement in shadow quality as greater number of samples are given to parts of scene which are visible (Fig. 3-1).

To start off, we ensure that the initial light frustum covers all of the scene. Then we render the scene from camera and store projected light-space positions \((x', y')\) in a texture called **control texture**. The control texture is then analysed and a bounding rectangle is constructed based on min-max values of the stored \((x', y')\). The bounding rectangle indicates what part of the scene to focus on when generating the shadow map. The transformation needed for this is

\[
T := [x'_{\text{min}}, x'_{\text{max}}] \mapsto [-1, 1], \quad [y'_{\text{min}}, y'_{\text{max}}] \mapsto [-1, 1]
\]

That is, the area inside the bounding rectangle is magnified to cover the entire image plane of the shadow map. The algorithm can be given as:

1. Render the scene from camera and output light-space positions \((x', y')\) to control texture \(C\)
2. Compute the bounding rectangle \(B := [(x'_{\text{min}}, y'_{\text{min}}), (x'_{\text{max}}, y'_{\text{max}})]\)
3. Render the scene from light source and apply computed transformation \(T\) post-projection while rendering to the shadow map \(S\)
4. Apply the same transformation while performing shadow map queries on \(S\)

Note that the initial rendering from camera can be a simple dry run, i.e. without any associated shading/evaluation performed. All that is needed are light-space coordinates of the primitives. Alternatively, it can also be used as the initial deferred rendering pass
Figure 3-1. Shadows become finer as the image plane of the shadow map shrinks. A) No frustum adjustment. B) Frustum restricted to the portion of scene visible from the eye and C) Frustum restricted to shadowed portion of scene where all required shading data are stored along with the light-space positions, the bounding rectangle is computed and shadow map generated, and then a final shading pass is executed.

The bounding rectangle needs to be recomputed when either the light or the camera moves.
3.2 Focus on Visible Shadowed Pixels

Since not all visible pixels will be shadowed, we can tighten the light frustum so that the shadow map covers only those visible pixels that will receive shadow (Fig. 3-1). This extension of the previous method can improve shadow quality further. The method requires an extra rendering pass. Before computing the bounding rectangle, we generate another shadow map from the light and then query this shadow map before we output to the control texture. Pixel positions clearly in front of the shadow map are ignored. This is a good estimate to ensure that only shadowed pixels contribute to the bounding rectangle computation.

We now replace the first step of previous algorithm with the following steps:

1. Render the scene from light source and record shadow map \( S' \)
2. Render the scene from camera and output light-space positions \((x', y')\) to the control texture, discarding pixels whose positions are in front of \( S' \)

![Figure 3-2](image)

Figure 3-2. Results with *Proog* model from *Elephants Dream*. A) No frustum adjustment. B) Focus on visible pixels and C) Focus on visible shadowed pixels. Shadow map contents in lower-left of each image. D) Magnified view of red square

<table>
<thead>
<tr>
<th>Method</th>
<th>Basic</th>
<th>Frustum 1</th>
<th>Frustum 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. performance</td>
<td>240</td>
<td>220</td>
<td>190</td>
</tr>
</tbody>
</table>

Table 3-1. Performance comparison. The term average implies average frame-rate during animation. Figures are in *frames per second* (*fps*).
In other words, we are looking for samples which have a shadow caster-receiver relationship. This view presents us with an alternative approach. We can obviate the additional depth map generation to determine shadowed pixels. For this, we need to maintain a buffer indexed by light-space locations to keep a count of how many visible pixels map to the same location in the shadow map. All locations with two or more samples sufficiently separated will satisfy the shadow caster-receiver requirement.

The main drawback of this approach is that ensuring synchronization of the required accesses to the buffer and atomicity of these accesses has to be done from the pixel shader which is complicated and a non-ideal setup.

This method, however, can give us the smallest possible light frustum dimensions (of the projection plane) for a scene.

### 3.3 Uniform Depth Values

Perspective projection results in depth values which are non-uniformly \((1/z)\) distributed. While it makes sense for the camera view to give higher precision to closer objects and vice-versa, same may not true for the light view. Sometimes, the light source may be far away from camera whereas the objects close to camera generally require greater shadow depth precision. Using uniformly spaced depth values is a better choice, since we then get equal precision at all depths in the light frustum. [Brabec et al., 2000]

Perspective transformation can be written as \((x_p, y_p, z_p, w_p)^T = M_{LightProj} \cdot (x_v, y_v, z_v, w_v)^T\), where \((x_p, y_p, z_p, w_p)\) are projected coordinates and \((x_v, y_v, z_v, w_v)\) are viewspace coordinates. Instead of applying this projection to the \(z\) component, a uniformly varying value of \(z\) in \([0, 1]\) can be computed as,

\[
z_l = -\frac{z_v + near}{far - near}
\]

We replace the post-projection \(z_p\) with \(z_l\). To offset the perspective division that takes place during rasterization, we also pre-multiply \(z_l\) with the homogenous coordinate \(w_p\) to get \(z' = z_l \ast w_p\). Finally \((x_p, y_p, z', w_p)\) are sent to the rasterizer.
3.4 Near and Far Planes

While the first two optimizations concern themselves with obtaining a tight frustum along $x$ and $y$ axes in lightspace, this can be done along the $z$-axis as well. Instead of setting arbitrary values for near and far planes of the light frustum, if we set them such that they tightly bound the visible portion of the scene, a reduced range for depth values is obtained. This gives us a greater sampling density along the $z$-axis. Objects beyond the far plane are not of interest since they would not be visible in camera view anyhow. As for objects closer than the near plane, special handling is required since these objects may cast a shadow on the scene. Depth clipping for these needs to be disabled to avoid clipping them away. The depth for all such objects can be set to zero after projection to ensure they are included in shadow computations and that they cast a shadow on the scene. [Brabec et al., 2000]

Computation of these bounds can be done in tandem with the control texture method explained before, only difference being that now we compute a bounding box with $z$-extent instead of a bounding rectangle.
4.1 Generating the Control Texture

Step 1 from the algorithm in section 3 mentions the control texture. The control texture contains the projected lightspace positions of the visible pixels. These positions are used to determine the bounding rectangle. To generate this texture, we first set the viewpoint to the current camera and pass down the lightspace position from the vertex shader to the pixel shaders, similar to the final rendering pass. The pixel shader only outputs the sample positions to the control texture.

When we want to narrow our focus to just the shadowed pixels, then a depth comparison is performed with a previously computed shadow map and the pixel is discarded if the comparison succeeds i.e. if the pixel is in front of the shadow map. Otherwise, the position is recorded in the control texture.

4.2 Computing the Bounding Rectangle

A DirectX 11 Compute Shader kernel determines the bounding rectangle. The Compute Shader is a new shader stage introduces in DirectX 11 that offers general-purpose computations on the GPU. It can work on arbitrary data input through readable resources (buffers, textures) and the output can be written to writeable resources. Multiple threadgroups can be executed with each threadgroup consisting a maximum of 1024 threads.

Synchronization and communication between different threadgroups is costly when compared to synchronization within a threadgroup. Moreover, for our application, the control texture over which our Compute Shader kernel will run is of limited resolution, say 1024x1024. As a result, no speedup was obtained by dispatching multiple threadgroups instead of a single one. Hence the computation of the bounding rectangle is performed using just one threadgroup. Each thread computes its local bounds from disjoint groups of \( t = \left\lfloor \frac{\text{width} \times \text{height}}{1024} \right\rfloor \) texels and then atomically updates the bounds shared
by other threads in the threadgroup. Finally, one of the threads writes it to an output buffer after all others have finished.

<table>
<thead>
<tr>
<th>Bounding rectangle</th>
<th>Disabled</th>
<th>Enabled - CPU</th>
<th>Enabled - GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. performance</td>
<td>184</td>
<td>70</td>
<td>152</td>
</tr>
</tbody>
</table>

Table 4-1. Effect of bounding rectangle computation on performance. The term average implies average frame-rate during animation. Figures are in frames per second (fps)

4.3 Post-filtering

Although shadow maps are stored as textures, conventional texture filtering techniques cannot be directly applied to them. This is evident since the output of a shadow map is a comparison operation and any filtering of depth values themselves will produce incorrect comparison results. Instead, what we can filter are the results of these comparison operations to give us a smooth gradient from shadowed portions to unshadowed ones. Note that these filtering techniques are not equivalent to soft shadows but merely interpolate or average.

4.3.1 Bilinear Interpolation

For bilinear interpolation, we get the four closest texel values to a sample location in a single fetch operation using the HLSL Gather statement. The 4-tuple is then compared with the sample depth to get four comparison results. HLSL also provides GatherCmp statement to perform both fetch and compare 4-texel values using a single instruction. The comparison results are then interpolated bilinearly across $u$ and $v$ axes to obtain a smoothly varying shadow factor.

4.3.2 PCF

While bilinear interpolation is smooth, it has a very limited filter area and hence only smooths out jagged edge boundaries to a small extent. To obtain a smoother gradation of shadow factors at shadow boundaries, we make use of Percentage Closer Filtering (PCF) proposed by [Reeves et al., 1987]. An $n \times n$ tap PCF filter averages the comparison results over an $n \times n$ regular grid of samples. The output is a fraction
Figure 4-1. Filter quality. A) Point sampling. B) Bilinear sampling. C) PCF 5x5 samples and D) Magnified view of red square denoting the percentage of neighboring texels the sample passes the depth test against and this fraction is used as the shadow factor. Gaussian weighted averaging gives smoother shadows.

4.4 Discussion

We used an NVidia GeForce GTX 690 graphics card and with Intel Core 2 Quad CPU Q9450 at 2.66GHz with 4GB memory to render animated frames from the open-source movie Elephants Dream. The implementation on which the performance numbers are based included iPASS, Approximate Catmull-Clark subdivision (ACC) [Loop and Schaefer, 2008], Skeletal animation and Texture mapping which enable cinematic quality rendering of the movie in real-time. The framework of implementation was DirectX 11 with the associated High-Level Shader Language (HLSL). Figure 4-2 shows screenshots of the implementation.
Figure 4-2. Four scenes from *Elephants Dream* from different viewpoints.
In conclusion, this thesis provides an efficient GPU implementation of various optimizations to basic shadow mapping. The optimizations, which echo the idea of making maximal use of the available resolution and precision, are simple to implement, reduce the aliasing of shadow boundaries and allow for some amount of dynamic refinement of shadows with changes in the camera view and the scene. The efficiency is mainly derived from offloading the bounding rectangle computation from the CPU to the DirectX11 Compute Shader. This minimizes the performance drop incurred from this scene analysis which would otherwise be of a high magnitude. An extension to the existing idea of frustum adjustment is also given that can provide the tightest possible light frustum for some scenes. A real-time demonstration of the implementation runs at above interactive rates, ca. 150 fps.
REFERENCES


BIOGRAPHICAL SKETCH

Sagar Bhandare was born in Belgaum, India. He received his Bachelor of Engineering degree from University of Pune in 2010. His work during his master’s degree focused on computer animation, high quality real-time rendering and GPU techniques.