Casting Light on Shadow Algorithms

David M. Staff III (davidmstaffiii@gmail.com)
Edinboro University of Pennsylvania
March 16, 2014

1 Abstract

Shadows are an important part of any scene rendered in video games and movies. Because of the importance of shadows, a number of methods have been developed for rendering them, some of which also cover other effects such as halos around light sources and crepuscular rays ("god-rays"). The purpose of the research covered in this paper is to understand some of the methods of shadow rendering, show understanding, and to implement at least one of them in OpenGL.

2 Introduction

Graphics, as a field of study, has come a long way since computers started using screens to display output. When video games started out, shadows weren’t important as they were all 2D pixel art games. Then 3D was introduced into gaming. Games became more and more complex, and so did the graphical representation. In the last few years, games have been pushing to look more and more realistic in their rendering. Shadows play a key role in the realism of a scene, and algorithms to render them continue to get more complex and efficient. Yet with all the advancements, shadows are still somewhat stuck when it comes to console gaming. Granted, this may very well be a hardware problem, but if companies have been designing games for consoles such as the Xbox 360 and Playstation 3 for years, they should be finding ways to make all aspects of the game better, including the shadows.
3 Related Work

There are a number of papers on shadow mapping/volume algorithms and rendering with those algorithms in real-time. Stamminger and Drettakis [2002] talk about perspective shadow maps, how shadows should be rendered in higher resolution as they get closer to the camera. It means that more time is spent on the areas that the user can see more detail with more effort, while the further shadows could slowly just become boxes or other simple shapes to draw. Nealen [2002] recompiled known efforts in shadow rendering since no one had done so since Poulin, Fournier, and Woo [Woo et al. 1990] in 1990. Jump ahead to SIGGRAPH Asia 2009 when a course book/paper was written that focuses on real-time interactive solutions [Eisemann et al. 2009]. It is also a guide to making decisions on what algorithms to use for a given problem. In 2010, a paper was published on using light volumes to render volumetric shadows in real time, in addition to covering phenomena such as crepuscular rays and halos around light sources [Billeter et al. 2010]. Chen, Baran, Durand, and Jarosz collaborated on a paper that followed a similar principle: tackle shadows and attack phenomena like crepuscular rays with it. However, Chen and the rest use 1D Min-Max Mipmaps instead of light volumes to do the calculations. In another 2011 paper, Louis Bavoil gives a brief overview of the algorithms for contact-hardening soft shadows, though he doesn’t go into as much detail as other papers regarding his method, and he probably wasn’t allowed to divulge as much since he published under NVidia.

4 Project

My goal with this project is to implement a method of shadow rendering from a recent paper on the subject. After implementing in OpenGL, I will port the shaders to Unity if I have time.

I started into the project with very little knowledge of shadow rendering. In my graphics course at Edinboro University we learned about rendering an image in OpenGL, moving the camera around the scene, and lighting the scene, but we didn’t make it to shadows and textures. When I picked up the project this semester and started to attempt to implement, I discovered that shadow mapping is no easy task, and having a shadow map is the first step in the paper. I found a tutorial online, and when I tried to integrate it with my program from my graphics course, It didn’t cooperate with the version...
of OpenGL I was using, so I had to start anew in order to get the first part of the project.

I ran into difficulties afterward, as other courses demanded my attention, and I am currently stuck going through the tutorial.

5 References

References


The algorithm discussed is similar to accumulation-buffer rendering, researched by Haeberli and Akeley, but renders a single time from the eye’s point of view instead of multiple times.


A more efficient way of computing effects such as volumetric shadows by constructing a polygonal mesh that earound the space that is illuminated.


A real-time algorithm using epipolar rectification to find visible sections by only using 1D height fields.


Focusing on real-time to interactive solutions, the course discusses aspects of algorithms, such as advantages, disadvantages, and suitability for each.

A number of shadow mapping/volume algorithms were developed/refined between 1990, the last time the works in shadows had been compiled, and 2002, when the paper was written.


Shadow maps are very susceptible to aliasing, perspective shadow maps algorithm renders more detail closer to the camera and less further away, decreasing visible aliasing without sacrificing speed.
The Shadow Mapping algorithm that we explored in tutorial 23 and tutorial 24 used a spot light as the light source. The algorithm itself is based on the idea of rendering into a shadow map from the light point of view. This is simple with spot lights because they behave in the same way as our standard camera. The spot light has a location and a direction vector and the area covered by the light grows as we move further away from its source. It is usually used to mimic the behavior of the sun which due to its size and distance seems to cast parallel light rays: In this case, we can no longer use Perspective Projection. Enter Orthographic Projection. The idea here is that of converging all light rays in one spot (the camera), the light rays remain parallel so no 3D effect is created. I Cast ray to light (shadow rays). I Surface point in shadow if the shadow rays hits an occluder object. I Ray tracing is slow, can we do better? (perhaps at the cost of quality). Shadow Algorithms. I We will first focus on hard shadows. I Planar Shadows I Shadow Maps I Shadow Volume. Planar Shadows. I The simplest algorithm: shadowing occurs when objects cast shadows on planar surfaces (projection shadows). y light y light. shadow y=0. Casting shadows in three.js involves 3 parts: the renderer which does the computation, the lights which cast shadows, and objects which receives lights and shadows. Set up the Renderer. The renderer is the one which will compute the shadows positions for your 3D scene. Shadow casting is quite expensive. It is only supported by WebGLRenderer. It uses Shadow mapping, a technique specific to WebGL, performed directly on the GPU. 1. renderer.shadowMapEnabled=true; You can smooth produced shadows with shadowMapSoft. It default to false. On the left, the shadow is crisp, on the right it is soft.