

|  |
| --- |
| AGP Part 2 Overview |
| Author: B00235610 |
|  |
| This document is a single page overview of the project I developed for part 2 of my module assignment. My demo includes deferred rendering, multiple light types, light volumes, height map generation which includes normal averaging, and cube maps to efficiently draw large cube objects in 3D space. It is also worth mentioning that because the lights are rendered with light volumes and they are dynamic as well. |
|  |
|  |

AGP Part 2 Overview

# Demo features & Overview

# User Instructions

• • •

Once starting the application you will presented with a splash screen simply press the start demo button to begin the demo.

When you want to rotate the camera make sure to hold down the left mouse button and use WASD keys for moving. I choose this as it helps make the demo easier to navigate.

To exit, simply close the window or press the escape key.

You can randomize the light and shade of the lights by pressing the randomize lights button. As all the lights are dynamic as a nice benefit of implementing deferred rendering.

The demo consists of many features these include.

* Deferred rendering
* Dynamic and multiple light types
* Height map generation
* Cube maps
* Light volumes

All features are implemented fully but some could be made more efficient. These features specifically where chosen so I could make a rich graphics game demonstration which shows off several advanced techniques. The demo itself renders a 524288 polygon height map where normal’s are calculated per vertex and 49 point lights are rendered in the center along with a simple multi textured house model. Cube maps are used for the skybox.

# Project bottlenecks/failings

There are no major failings in the project everything is rendered at 100+ frames per second and this is with all 49 point lights visible at the center. However I didn’t use any sort of culling mechanism when it comes to shrinking down the data sent to the GPU as with the list of features already implemented it would have been overkill to implementing frustum culling. This is the main bottle neck for the application however it still renders at a very fast rate as it isn’t large enough to be considered a major problem and that was the case for every little problem I encountered.

# Potential improvements

**Improvement 1**

Implementing geometry shader’s to perform culling and breaking down large sets of geometry in to smaller meshes would help cut down the amount out data being rendered into multiple render targets and would therefore help solve the memory bandwidth bottle neck which deferred rendering traditionally suffers from.

**Improvement 2**

Using more efficient light volumes for each point light would also greatly improve performance. By dynamically generating light volumes to match the area of effect it has not just the radius of the point lights reach would significantly improve performance.