IMD3002 Term Project - Radiosity Rendering In Maya

Jake Cataford, Alex Winch

April 24th, 2014

Abstract

This document reports on the progress that we have made toward building an openGL radiosity renderer plugin for Autodesk Maya. It outlines which objectives we have completed and which would require more time.

1 Introduction

This report outlines the process and objectives we wished to accomplish in the creation of this project. It outlines our previous objectives and describes our progress in completing those objectives. It outlines any objectives we have failed to accomplish and the reasons why we have failed to accomplish them. It also describes any methods we used to mitigate problems and expands on the points made in the preceding reports.

2 Objectives

2.1 Part A

This section outlines the objectives this project hoped to accomplish during scope development in the preliminary design stage (Part A)

- This project should enable a user to render a scene from within Maya, without using any third party tools or dependencies
- This project should provide a way for the user to edit and configure the renderer to their personal preference
- This project should allow approximation of global illumination through the radiosity technique
- The project source code should remain in a maintainable, modular state

- The renderer should run efficiently, algorithms within the renderer should have as low of a complexity as possible to reduce render time
- The project GUI should be implemented in as idiomatic of a way as possible. Following the design patterns that the Maya User Interface adheres to
- The renderer will render to a [single node] only to avoid spending too much time dealing with threading implementations

2.2 Part B

This section outlines the objectives completed for the Prototype submission.

- Obtaining the render context from Maya, and rendering pixels to it

2.3 Part C

This section outlines the objectives completed for the Final submission.

- Created an openGL context from within a Maya C++ plugin
- Rendered non-scene objects to context
- Traversed the DAG and found the mesh objects in the scene
- Copy from the openGL context to the render view
- Parsing syntax from a Python script to pass information into plugin
- Created a Python GUI containing render settings with parameters for File Output, Renderable Cameras and Image Size

2.4 Failed Objectives

- Rendering the scene
- Generate lightmaps
- Generate light patches on surfaces
- Calculate incident light and patch emission
- Generate the radiosity loop
- Assign patch emission back to the lightmap
- Blend the radiosity map back onto the final texture mapped scene

3 Method

3.1 Part B

3.1.1 The Renderer

The renderer is a procedural c++ program that harnesses parameters from our GUI and scene information from Maya, then compiles it into an image file by traversing the data through a procedure. This program is not Object oriented because the nature of a renderer is procedural, and there is nothing to gain from having an application state. The renderer processes the data through a variety of algorithms to render the scene geometry and complete the rasterization pipeline. Below, you will find examples of the steps needed to produce the final raster image.

The Technique requires reducing the scene's surfaces into patches, where each patch could be represented by a structure such as:

```
structure Patch:
  color emission;
  color reflectivity;
  color incidentLight;
  color excidentLight;
```

To render the scene, consider this process:

- Load the scene.
- Divide each surface in the scene into patches.
- For each point light, raycast randomly around the scene and initialize the patches with some emission value.
- For each area light, give all patches belonging to the light the emission value of the light.
- For each ambient light, add that light value to the emission value of all patches.
- For each patch, set patch emission to be the average albedo and color value of the pixels in the patch.
- For each pass, loop through the patches in the scene and render the scene from the point of view of the patch. Record the sum of the render to the patch. Then calculate excident light via Incident light by reflectance, plus the previous emission value.

Directional lights require a different method of representing their influence on scene light. We take the normal of the patch, compare it to the direction

of the directional lights in the scene, then set the render background (non-occluded light) with an emission value of the dot product of the normal and light direction. (Lambertian diffuse model).

Patches themselves represent a portion of a global lightmap that needs to be generated in order to map the lighting of the scene onto the geometry. As such, constructing a lightmap is another requirement of this renderer.

3.1.2 The GUI

for the radiosity rendering engine is based on the default Maya dialogue window styling for consistency within the software and the limits of the code.

The GUI is very easy to use, with a focus on intuitive design for the user. At this part of the user experience, the user requires a quick dialogue allowing them to customize their render, without additional options that wouldnt be pertinent to this type of render. For example, because our rendering engine doesnt support animation, render frames options arent included.

The window is quite similar to the mental ray rendering engine window, with a few key changes. Weve included a common tab to allow the user to set all their basic params for their render. For example, under this tab weve included colour management, file output, renderable cameras, image size, scene assembly and render options. However, the primary difference in our rendering engine is the additional radiosity tab. Under the radiosity tab were including the number of radiosity passes, maximum path divisions, point light quality and a checkbox to allow a debug mode.

3.2 Part C

We encountered lots of issues while implementing this plugin, and lots of those issues arose from inexperience with the frameworks and libraries at hand. The first major problem arose from using GLUT to wrangle the openGL context. GLUT did not behave well and had several bugs when working with openGL 3 that made it unusable for this context. We then flipped our wrangler to use SDL and had better success with that framework, we stopped having the artifacts and access errors that came with the GLUT library. However, we began to run into deeper issues when we integrated this into Maya. One of the biggest issues was that Maya's openGL contexts would leak into our contexts, and our buffers would contain the information from Maya's ui. We also made the mistake of using openGL 3 when authoring the plugin and found that it was incompatible with the computers that the school is using.

We managed to traverse the DAG and retrieve the mesh information and transfomation information for the objects in the scene, but didn't have the time to actually render the data.

Debugging also proved difficult because of the long process to go from C++ to Maya and test changes. Maya would frequently crash when our plugin crashed (mostly due to bad access because of the context perversion mentioned earlier) and it would further lengthen the process. One of the things we did to mitigate

this problem was build a standalone openGL application to avoid the process of opening and closing Maya over and over.

We also templated out several classes that were to be used later in the development cycle and made a script with a UI that could trigger our plugin and pass flags in. The parsing of syntax flags was also implemented in the c++ plugin.

4 Design Sketches

Figure 1: Example of common options

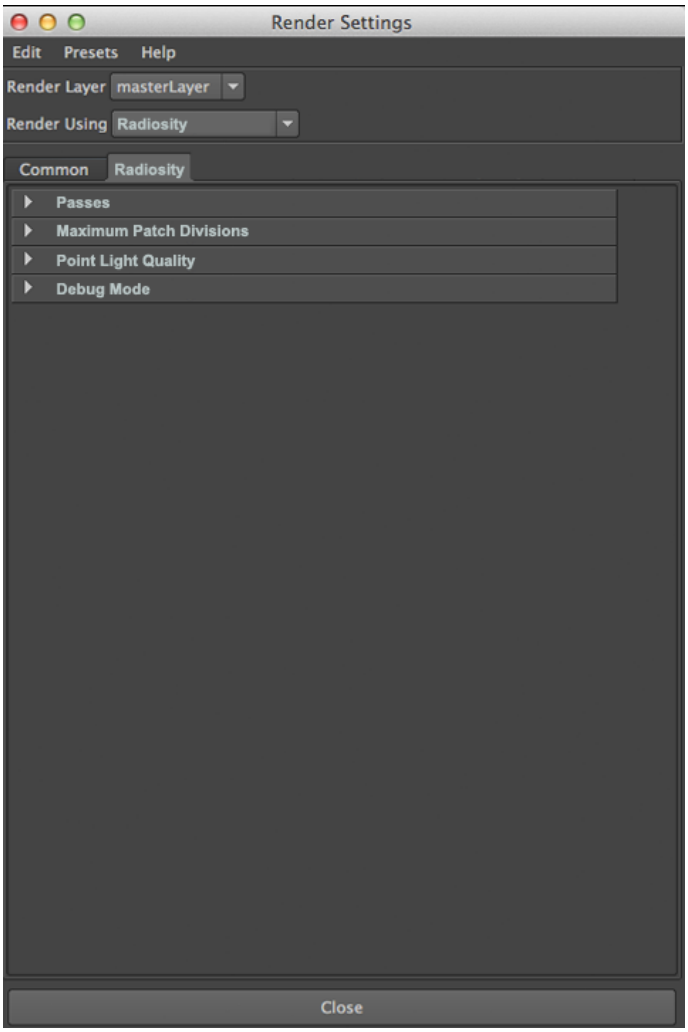

Figure 2: Example of radiosity specific images

5 Citations

OpenMaya included examples were referenced frequently along with the official documentation for OpenMaya and Maya's Python Wrapper for mel. We also referenced the Official OpenGL and SDL documentation frequently. http://nehe.gamedev.net was also used for learning OpenGL 3. Course materials we looked at were used as reference for the theory.