Coursework Report

Alistair Mckay
40207511@napier.ac.uk
Edinburgh Napier University - Module Title (SET08116)

Abstract

This report will outline the creation and implementation of a graphics coursework project. The aim of this project was to create a "the floor is lava" type scene, essentially a living room which has a floor made of lava.

Keywords – Cameras, Texturing, normal mapping, lighting

1 Introduction

The scene featured in this report, which we shall call, "The floor is lava" was created in Microsoft's Visual studio platform in c++ using OpenGl and a graphics framework created and provided by Edinburgh Napier University.

The scene uses several different graphical techniques to create the overall effect. The main effects featured are grey-scale, masking, texturing, lighting and a skybox.

The motivation behind this project was to take what was learned during part one of the coursework and to build on what was learned while building

2 Post processing

To build and improve upon the scene from part one of this coursework, it was decided to utilise several post processing effects.

The first effect that was chosen to be used was wire framing. This was chosen as it is rather simple to implement and gives a cool effect when activated. It also provided to be useful while debugging errors in the scene as it gives us the ability to see clearly where objects are overlapping in the scene.

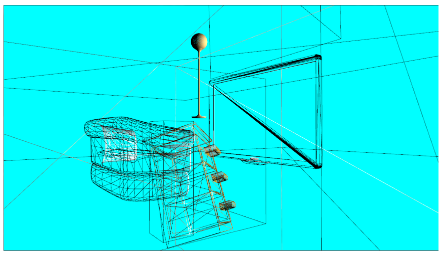


Figure 1: **Wire frame** - The project in wire frame mode

A masking effect was also chosen to be part of the scene. Masking is a technique which, by using a fragment shader, an image is overlayed onto the scene which moves with the camera. This effect is commonly used in AAA titles to do things like have key bindings for various functions displayed on the screen. Masking is also commonly used in shooter titles to display a weapons scope over the scene when the character aims down sight, and example is shown in the below figure.

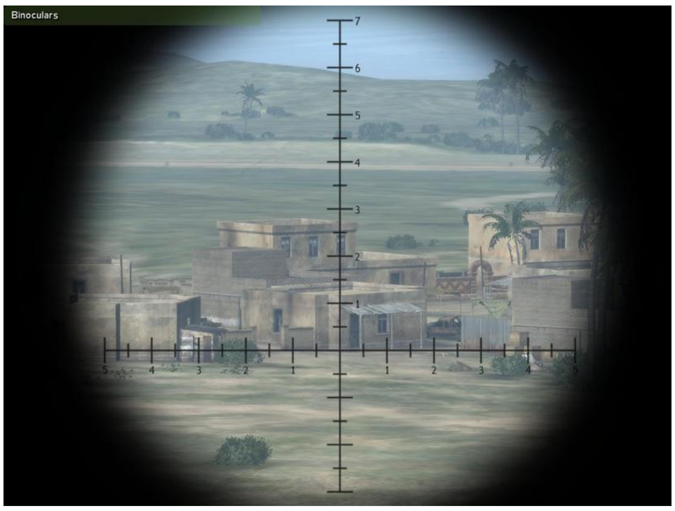


Figure 2: **Arma 3 mask** - The project in wire frame mode

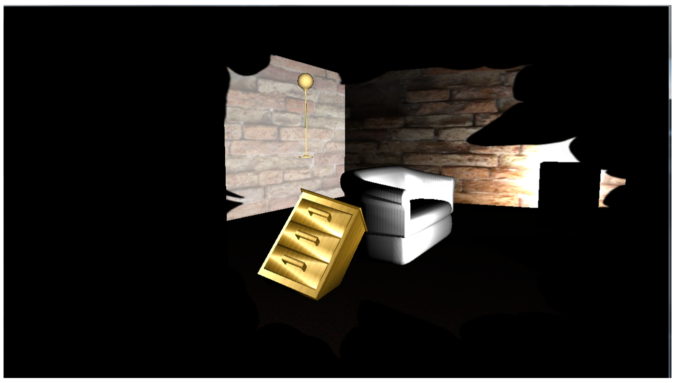


Figure 3: **The mask on this project** - The project in wire frame mode

A grey scale effect was also added to the scene. This is a post processing effect which applies a fragment shader over the scene. A colour, normally grey, is applied so the scene which makes the scene appear to be in black and white. For my scene it was decided to use a red instead of a grey as it was decided that red was more suitable to the aesthetic of this scene.

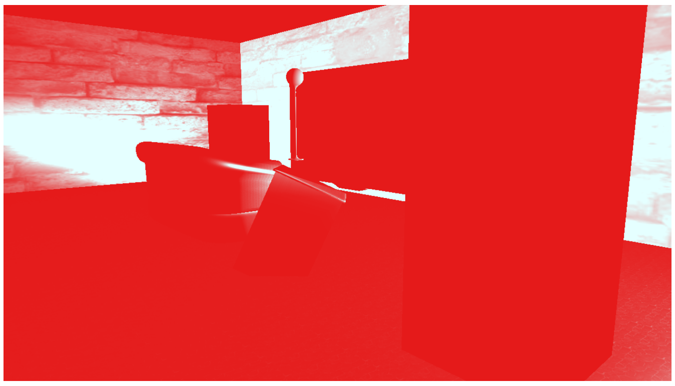


Figure 4: **Grey-scale** - The project in grey scale mode

3 Rendering techniques

During development, there were several rendering techniques added to the scene to improve upon it.

One of the effects used as a sky-box. A sky-box is a rendering technique which basically wraps our entire scene in a cube and textures the inward facing faces with images to give the user an impression that they are not in a box scene but in a richer environment.

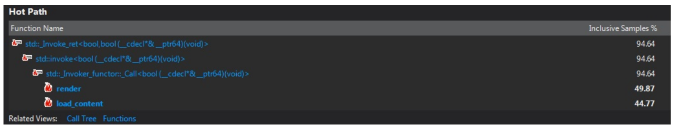
It was decided to add an additional, more complex rendering technique to the scene. The idea behind this technique was to add the ability to display a screen-shot of the current camera view at any given point to the TV. This would be achieved by adding in an additional plane and transforming it to fit inside the bezel of the TV and then display the contents of a frame buffer to the new plane as a texture. Due to time constraints, completing this effect fully was unfortunately unachievable, however this was completed partially and by using keys 2 and 3, we are able to display a frame buffer to a wall and have a shot from the frame buffer displayed there once the effect is deactivated, (see Figure 9).

4 Problems encountered

Throughout the course of this project, several problems were encountered. The main problem encountered was an ongoing issue with the scenes lighting. The scene uses point lights and spotlights in an array which has an issue that means the lights are rendered incorrectly and therefore some textures are not visible. The other issue encountered was an issue where the grey scale effect is rendered incorrectly. An issue was also encountered in which the frame buffer that we used to display the cameras view to the wall displays upside-down. Given more time these issues would be fixed but unfortunately given the time constraints, fixing them is infeasible.

5 Optimisation

Upon completing this project, it was decided that it would be beneficial to investigate the code to improve frame rate and general optimisation .The first thing that done was add to the program, a frame rate counter that outputs to the command prompt as the project is running. This allowed for real time monitoring of the frame rate that the project was achieving and provided the ability to immediately investigate any drops in frame rate that came up.



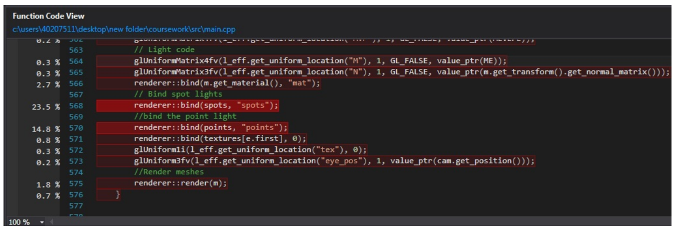
Hot Path

Function Name	Inclusive Samples %
std::_Invoke_ret<bool,bool (__cdecl*&__ptr64)(void)>	94.64
std::invoke<bool (__cdecl*&__ptr64)(void)>	94.64
std::_Invoker_functor::_Call<bool (__cdecl*&__ptr64)(void)>	94.64
render	49.87
load_content	44.77

Related Views: Call Tree Functions

Figure 5: **Hot paths** - The hot paths of the project

The second thing that allowed for investigation of the code was visual studio's inbuilt profiler. The profiler allows for a graphical representation of your code to facilitate easier optimisation. As you can see from the above figure, the render and load content sections of the project were the areas that took up the most of the CPU, or the "Hottest" parts of the project.



Function Code View

c:\users\40207511\desktop\new folder\coursework\src\main.cpp

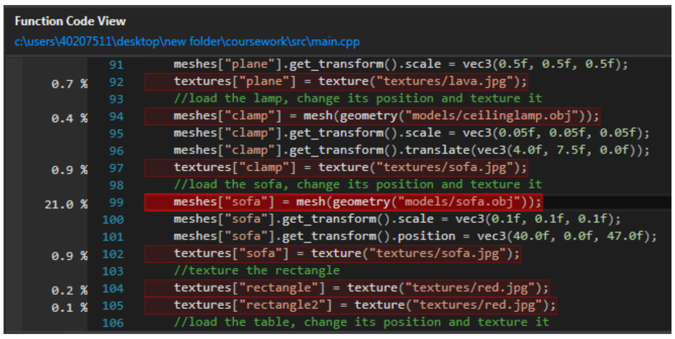
```
                // Light code
 0.3 %          glUniformMatrix4fv(l_eff.get_uniform_location("M"), 1, GL_FALSE, value_ptr(ME));
 0.3 %          glUniformMatrix3fv(l_eff.get_uniform_location("N"), 1, GL_FALSE, value_ptr(m.get_transform().get_normal_matrix()));
 2.7 %          renderer::bind(m.get_material(), "mat");
                // Bind spot lights
23.5 %          renderer::bind(spots, "spots");
                //bind the point light
14.8 %          renderer::bind(points, "points");
 0.8 %          renderer::bind(textures[e.first], 0);
 0.3 %          glUniform1i(l_eff.get_uniform_location("tex"), 0);
 0.2 %          glUniform3fv(l_eff.get_uniform_location("eye_pos"), 1, value_ptr(cam.get_position()));
                //Render meshes
 1.8 %          renderer::render(m);
 0.7 %      }
```

100 %

Figure 6: **Render function** -The projects render function

The first function that we looked at was the render function. As you can see by the above figure, there were certain aspects of the render function using a noticeable amount of the CPU There are two noticeable sections of the render function that are using a significant amount of CPU load. They are the lines where we bind the spot lights and the point lights. These binding functions are only partially operational and both caused a frame rate drop to around 30 frames per second. After investigating further, the frame rate drop was able to be removed from both functions however unfortunately we were unable to completely remove the issues that cause them to take up significant CPU load and work exactly as intended. Given more time the issues with these functions would be completely removed.

After investigating the render function, the load content function was next. As you can see by the above figure, the profiler picked up on several areas of the function that were using a noticeable amount of CPU load. This can be put down to how doing a lot of loading will tend to use a lot of load however there was one part that the profiler picked up that was using a large amount of load. It was assumed that this was down to the amount of polygons in



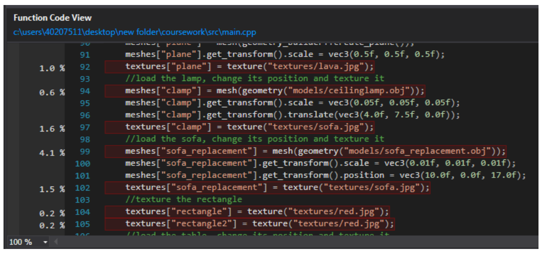
Function Code View

c:\users\40207511\desktop\new folder\coursework\src\main.cpp

```
            meshes["plane"].get_transform().scale = vec3(0.5f, 0.5f, 0.5f);
 0.7 %      textures["plane"] = texture("textures/lava.jpg");
            //load the lamp, change its position and texture it
 0.4 %      meshes["clamp"] = mesh(geometry("models/ceilinglamp.obj"));
            meshes["clamp"].get_transform().scale = vec3(0.05f, 0.05f, 0.05f);
            meshes["clamp"].get_transform().translate(vec3(4.0f, 7.5f, 0.0f));
 0.9 %      textures["clamp"] = texture("textures/sofa.jpg");
            //load the sofa, change its position and texture it
21.0 %      meshes["sofa"] = mesh(geometry("models/sofa.obj"));
            meshes["sofa"].get_transform().scale = vec3(0.1f, 0.1f, 0.1f);
            meshes["sofa"].get_transform().position = vec3(40.0f, 0.0f, 47.0f);
 0.9 %      textures["sofa"] = texture("textures/sofa.jpg");
            //texture the rectangle
 0.2 %      textures["rectangle"] = texture("textures/red.jpg");
 0.1 %      textures["rectangle2"] = texture("textures/red.jpg");
            //load the table, change its position and texture it
```

Figure 7: **ImageTitle** - Model using high load

the model. After replacing the model with an alternative one the amount of load on decreased as you can see by the following figure.



Function Code View

c:\users\40207511\desktop\new folder\coursework\src\main.cpp

```
            meshes["plane"].get_transform().scale = vec3(0.5f, 0.5f, 0.5f);
 1.0 %      textures["plane"] = texture("textures/lava.jpg");
            //load the lamp, change its position and texture it
 0.6 %      meshes["clamp"] = mesh(geometry("models/ceilinglamp.obj"));
            meshes["clamp"].get_transform().scale = vec3(0.05f, 0.05f, 0.05f);
            meshes["clamp"].get_transform().translate(vec3(4.0f, 7.5f, 0.0f));
 1.6 %      textures["clamp"] = texture("textures/sofa.jpg");
            //load the sofa, change its position and texture it
 4.1 %      meshes["sofa_replacement"] = mesh(geometry("models/sofa_replacement.obj"));
            meshes["sofa_replacement"].get_transform().scale = vec3(0.01f, 0.01f, 0.01f);
            meshes["sofa_replacement"].get_transform().position = vec3(10.0f, 0.0f, 17.0f);
 1.5 %      textures["sofa_replacement"] = texture("textures/sofa.jpg");
            //texture the rectangle
 0.2 %      textures["rectangle"] = texture("textures/red.jpg");
 0.2 %      textures["rectangle2"] = texture("textures/red.jpg");
```

100 %

Figure 8: **ImageTitle** - Replacement model using lower load

6 Related work

The inspiration for the scene came from the game that probably everybody played as a child in which the aim was to get around the room by jumping and climbing over the various furniture and by all means, not touching the floor.

Inspiration was also gained from the in-development game from Klei entitled, "Hot lava", which is based on the childhood game. "Hot lava" which is currently in its beta testing phase takes the childhood classic game and takes it a step further with the player being able to see the red hot lava on the ground and become fully immersed in the game of dodging the dreaded lava that has replaced the floor.

7 Future work

Between part one of this coursework and part two many other features were able to be added to improve the overall look and quality of the scene, however there are still some things that could be added. For example, if the projects time-scale were to be increased, a day/night cycle could be added to the scene, there could also be scope for more houses to be added outside of the window, to give the impression that we are inside a house on a street, this would add to the overall realism of the scene.

8 Conclusion

Overall, both parts of this coursework have been interesting to complete and have provided allot of worthwhile graphical knowledge. Part one taught many basic key skills that became very useful while developing part 2. The second part of the coursework built on the skills from part one and enabled us to gain valuable knowledge in core and more advanced concepts that will be of great help while working on future projects.

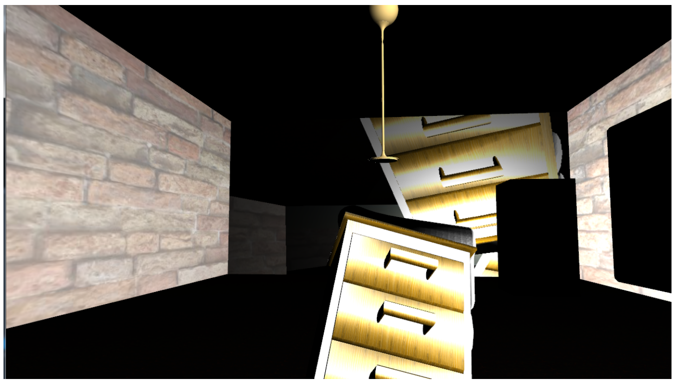


Figure 9: **Additional rendering** - The additional rendering technique

9 References

1. Arma 3 (2013 Bohemian Interactive)

2. Graphics framework (2017 Edinburgh Napier university)