Edinburgh Napier UNIVERSITY

Coursework Report

Emma Parsley
40206111@live.napier.ac.uk
Edinburgh Napier University - Computer graphics (SET08116)

Abstract

This project aims to render a 3D environment in real-time, using OpenGL and C++. The intent is to create a 3D scene shaded as if it where a cartoon

Keywords – OpenGL, GLFW, Shading, lighting, Non-Photorealistic, Outlines, Post-Processing

1 Introduction

The project is a scene of three floating islands shown using non-photorealistic rendering to create a visually interesting cartoon-like environment. The project includes use of normal mapping, to show detail on otherwise flat objects, skybox, to create the appearance of an endless environment, multiple lights using sampled Phong shading to cause the cell shaded look, edge detection to create outlines, post processing, and debug views.

2 Implementation

2.1 Lighting

To create a cartoon-like environment the light was calculated using a cell shading technique. A 1D texture of shades which will be sampled from to decide the colour of the light.

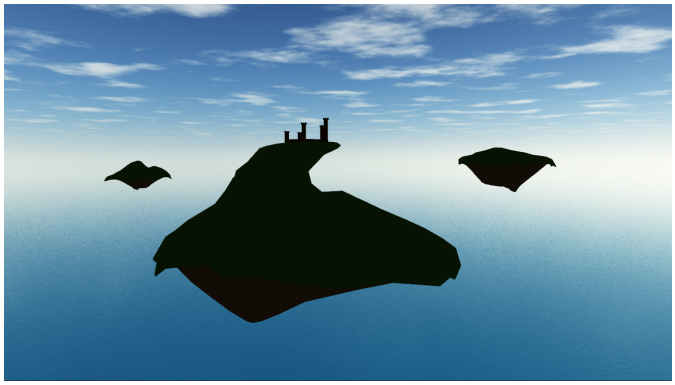

Figure 1: **Ambient Lighting** -Scene with only ambient lighting affecting it

Ambient Lighting Ambient lighting is the background lighting which has no direction and so just shows flat colours for the objects in the scene. See Figure 1.

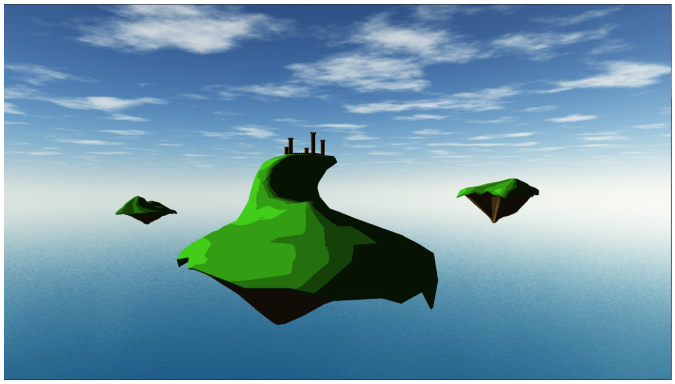

Figure 2: **Directional Lighting** - Scene with only directional lighting affecting it

Directional Lighting Directional Lighting effects the whole scene from one direction. In this project it uses the phong shading model;

$$I = I_a K_a + f_{att} I_p (K_d N.L + K_s (R.V)^n)$$

However to create the cell shading effect the diffuse and specular light are clamped and used as texture coordinates to sample from a 1D texture of shades then multiplied buy the reflection and light colour. See Figure 2.

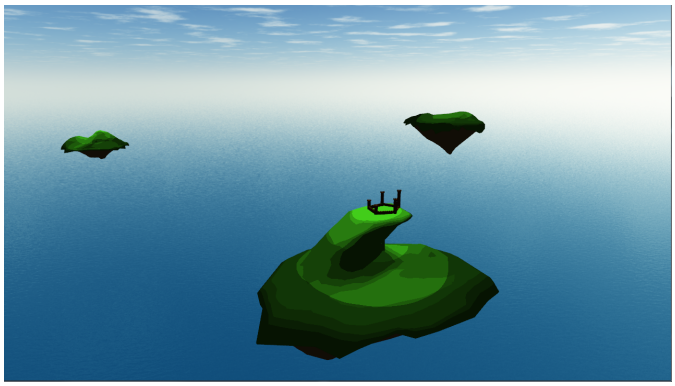

Figure 3: **Spot Lighting** -Scene with only Spot lights affecting it

Spot and Point Lights The scene can also be lit by spot and point lights which are done in a very similar way to the directional light where we clamp;

$$\frac{max(-R.L,0)^p}{k_c + k_l d + K_q d^2}$$

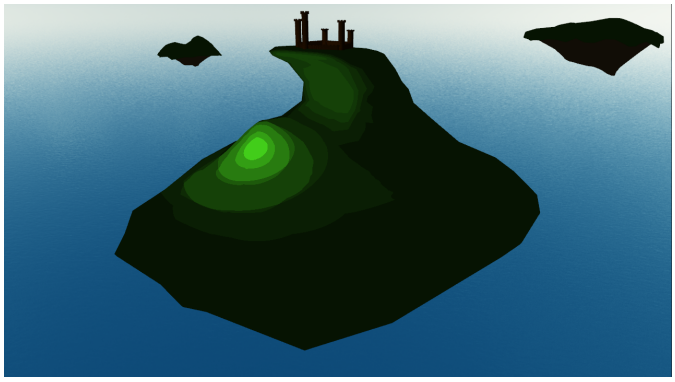

Figure 4: **Point Lighting** - Scene with only Point light affecting it

for spot lights and;

$$\frac{1}{k_c + k_l d + K_q d^2}$$

for point lights to find the texture coordinates to use to sample the shade and multiply it by the light colour and then use the result of that as the light colour for the same calculations we did in directional lighting, including the sampling from the 1D texture again. See figure 3 and figure 4.

Combined Lighting The combined lighting is simply done by adding the calculated colours of each light in the fragment shader. This allows for more than just the shades in the 1D texture to appear making clear the geometry of the object while still restricting the shades enough for it to appear cartoon-like. See figure 5.

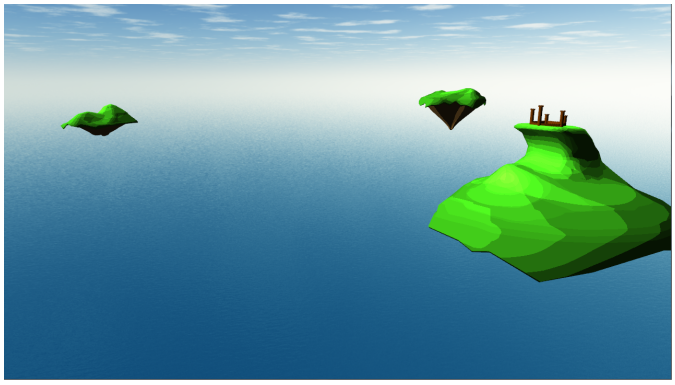

Figure 5: **Multiple Lights** - Scene with all lights affecting it

2.2 Normal Mapping

In order to create detail on the flat textures used within the project normal mapping is used. The use of a normal map allows for this detail without needing to render and display more vertices. The castle in figure 6 has just a flat texture on it but when displayed with the normal map it appears to have bricks.

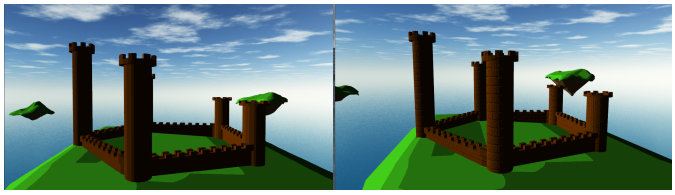

Figure 6: **Normal Mapping** - Left: Castle without normal mapping, Right: castle with normal mapping

2.3 Skybox

To help with the visual pleasing look of this project a skybox was added to create the illusion of an endless environment in the sky above the sea. The skybox is simply a cube scaled up with six seamless textures applied on each face to create the look of the environment. Face culling is disabled on the cube so you can see the textures on the inside, and the cube is moves to the position of the camera so that the user cannot escape the skybox.

2.4 Outlines

Edge Detection and Masking To continue the cartoon look of the project outlines were be added. One way of doing this is using post processing techniques. By first rendering the scene in a frame buffer we can use edge detection on this frame to find the edges. To detect the edges we use a technique called box filtering to calculate the resultant pixel colour from the surrounding pixel colours. If the pixel is a different colour to the surrounding pixels it is an edge. All the edges will be the colour of the original pixels and anything that is not an edge will be black.

Once the edges are calculated we can set all coloured pixels to black and all pixels close to black to white. This will give something we can later use as a mask and overlay on top of the rendered scene. These outlines can be put on top of our current scene by rendering the scene into a separate frame buffer and multiplying a sample from this frame with a sample from the mask created earlier. If the skybox and textures are rendered before edge detection the outlines will appear between the colour changes in those too. See figure 8.

Figure 7: **Just the outlines** - Scene with no objects, just the outlines from edge detection

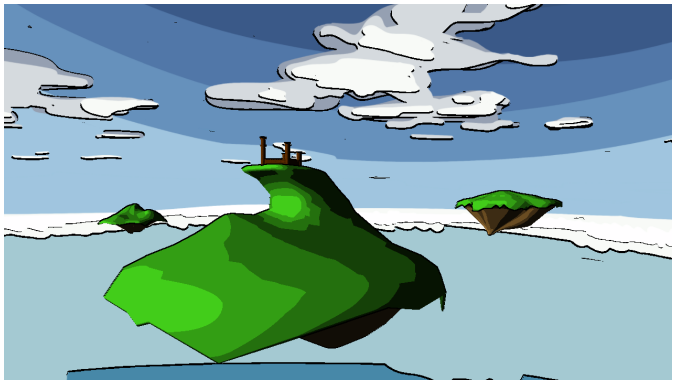

Figure 8: **Edge Detection and Masking** - Scene with edge detected lines masked over it

Silhouette The issue with this edge detection method is it takes three render passes to complete and we can't control the line width of the outlines. Another option for the outlines is to create geometry, this can be done in one pass and the line width is adjustable[1]. For the final project these outlines are left out as some problems were encountered while attempting this.

To do this each vertex of a triangle must be checked to see if it has a front facing normal. If precisely one or two normals are front facing this triangle will need a silhouette segment. A vertex is front facing if

$$\alpha_a = normal_a \cdot camera_a > 0$$

The first two points of the silhouette segment can be calculated using linear interpolation (y_{ab}).

$$y_{ab} = \frac{\alpha_a b - \alpha_b a}{\alpha_a - \alpha_b}$$

The second two points are simple the previous two plus their normal multiplied by a float the value of which will change the size of the lines.

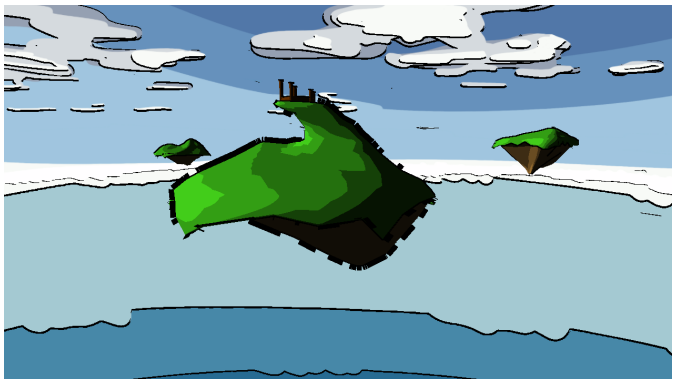

Figure 9: **Silhouette Outline** - Attempted silhouette outlines around objects

The edges of this are still patchy, segments appear to be facing in the wrong directions where they should all face the camera for the optimum effect. Also The outer points of the segment are not connected, a method is needed to make sure there are no gaps between silhouette segments. See figures 9 and 10.

Figure 10: **Wireframe Silhouette Outline** - Attempted silhouette outlines drawn in wire frame so the unconnected parts are more obvious

2.5 Post Processing

As well as edge detection and masking post processing can be used to create all sorts of filters across the screen such as adding the ability to adjust the colours. Rendering the scene into a frame buffer and then rendering this frame onto a quad across the screen with the simple edit of adding the RGB values that will be changed to the end colour of each pixel allows any colour to be added or taken away from the scene.

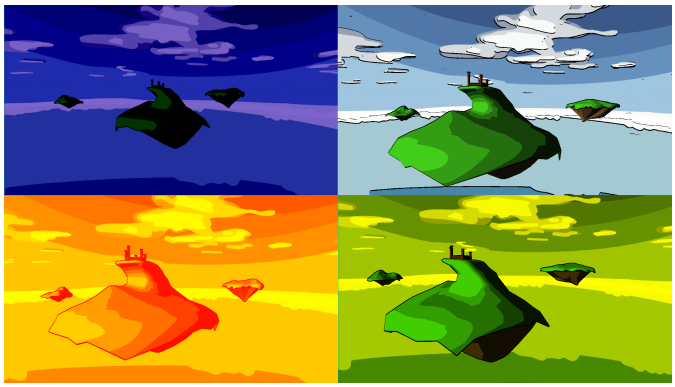

Figure 11: **Adjusted colour values** - Scene with colour values adjusted

2.6 Debug

This scene has the ability to view the objects normals and wireframes to help with any debugging

normals Normals can be created using a geometry shader by simply emitting the original position of each vertex and the position plus the normal in a line strip. Outputting one line for each vertex with these positions creates visible normals. See figure 12.

Wireframes Creating a wireframe is very similar to creating visible normals. Simply output each vertex at it's original position and connect these together in a strip of lines. This will cause each edge of the triangles in the geometry to be drawn out and create a wireframe. See figure 12.

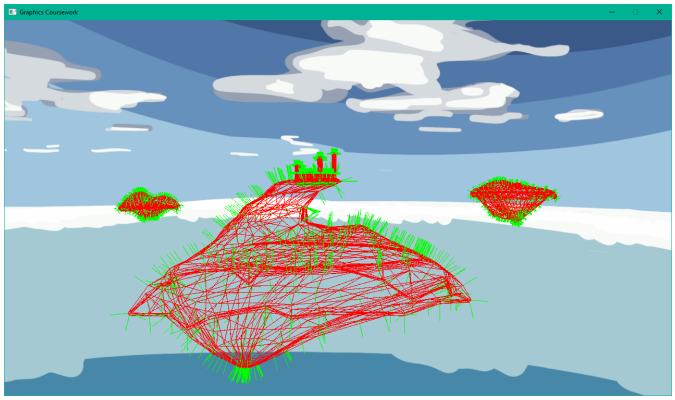

Figure 12: **Wireframe with Normals Showing** - Scene with wireframe of objects and normals showing

2.7 Optimisation

Using visual studio profiling it is obvious that the render function takes up a lot of strain on the CPU, to fix this the code was checked for any unnecessary binds, and everything that didn't need to be within loops was moved out of them.

3 Future Work

Optimisation Although the scene runs fine at the moment it would be good to add more optimisation such as frustum culling to stop unseen objects from being rendered.

Sky Adding clouds to the sky around the islands would add to the overall effect of the scene and visual appeal.

Normal maps Adding normal maps to the islands could create a nice grass-like effect over the current flat texture

Outlines The use of geometry to create silhouetted outlines where the line width can be changed would help really show this cartoon non-photorealistic style.

4 Conclusion

The project works as intended creating an effective non-photorealistic enviroment. The use of skybox, cell shading and edge detection work well together to create the desired effect. However, more could be added and worked on to optimise and visually enhance the current scene.

References

[1] H. Balázs, S. László, and C. Balázs, "Fast silhouette and crease edge synthesis with geometry shaders," p. 4.