Computer Graphics
Coursework Part 1 - Report

Dimitar Hristov
40201757@live.napier.ac.uk
Edinburgh Napier University - Computer Graphics (SET08116)

Abstract

The aim of this project is to create a realistic 3D scene, rendered in real-time. The project is inspired by the series *Games of Thrones*[1] (see **Figure 1**) and previous years projects found on the games website[2] of Napier University. A wide variety of graphics techniques were used to create the 3D scene, from material shading and post-processing effects to reflective and refractive rippling water. This report covers how the scene was implemented and what future work is considered.

Keywords – 3D scene, OpenGL, GLSL, shadows, normal mapping, refraction, reflection, Fresnel effect, distortion, instancing, terrain

Figure 1: **Scenes used as inspiration**

1 Introduction

Scene parts The project is meant to be visually intriguing and to demonstrate core understandings of Computer Graphics principles. The 3D scene consists of:

- realistic water;
- instancing of 800 dragon eggs made with normal mapping around the island;
- island terrain surrounded by the water;
- a miniature model of the Earth and the Moon, rotating around it;
- a wall and a spot light demonstrating shadows;
- a model of a dragon next to the Earth, protecting its egg;
- geometry objects moving with hierarchical transformations (one of the dragon eggs and its' protectors);
- a skybox that brings completeness to the scene and also background;

Graphics effects The graphics effects implemented in this project include:

- reflective and refractive water with implemented Fresnel effect, depth and distortion;
- post-processing effects, including mask, edge detection, sepia, motion blur and wire framing;
- multiple light types (directional, spot and point lights);
- texturing and normal mapping that give high level of details;
- shadows that make the scene more realistic;

There are two types of cameras implemented within the project: *free and target camera*. The free camera allows the user to go around and explore the 3D scene and the four target cameras show the scene from four static points of view.

Further information about these graphics techniques is given later in the report. **Figure 2** shows part of these elements and graphics effects.

Figure 2: **Scene from the project**

2 Related Work

Part of the techniques used in the scene can be found in the workbook for the Computer Graphics module - SET08116 at Edinburgh Napier University[3]. In addition to this, for the water effect an online tutorial[4] was followed and adapted for the purposes of this project. Some of the graphics techniques had to be taken further in order to develop the final 3D scene in an optimized way.

3 Implementation

In order to make the scene alluring, a wide variety of geometry objects were used. These elements are:

3.1 Multiple lights

There are three types of lighting sources implemented in the project: directional, spot and point lights. Together with the normal maps they give to the geometry objects good realistic views.
The Phong shading was used throughout the project. It is a graphics technique that calculates light on a per-pixel rather than per-vertex level. It improves upon Gouraud shading and provides a better approximation of the shading of a smooth surface. On **Figure 3** is shown the difference between the Gouraud - per-vertex (left) and the Phong - per-pixel (right) shading.

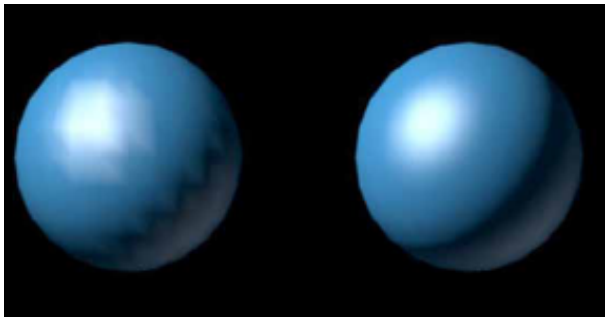

Figure 3: **Gouraud and Phong shading**

Figure 4 shows the Phong equation where the light is white, the ambient and diffuse colours are both blue and the specular colour is white. The same properties are applied to the meshes in the scene and used in the fragment shader when calculating the correct colours to be displayed.

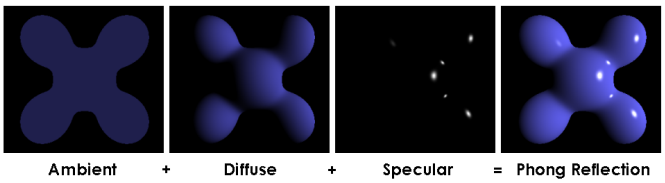

Figure 4: **Phong shading equation**

3.2 Texturing and Normal Mapping

Texturing is the process by which image data can be applied to geometry objects and models to provide more details.
Normal mapping is a technique that allows us to calculate the normals on a per-pixel level and gives a high level of detail to the objects. It creates the illusion that a flat mesh has depth on its surface by reacting with the light in the scene. In the project there are normal maps applied to the Earth, the dragon eggs and to the water. See **Figure 5** for references.

Figure 5: **Normal maps in the scene**

This effect can be achieved with normal map images similar to the one on **Figure 6**. The RGB values of the texture represents the <x,y,z> components of the normal at this point. In addition to the normal, the bi-normal and the tangent are also required for the transformation matrix that is used to calculate the sampled normal. This normal map is used for the effect achieved on the two dragon eggs on **Figure 5**.

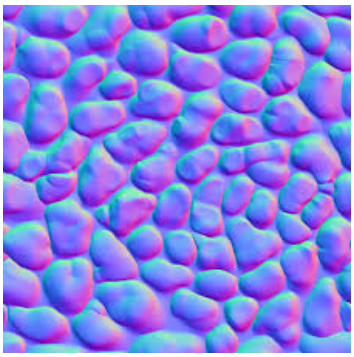

Figure 6: **Normal map**

3.3 Shadows

Shadow mapping uses the depth buffer that captures depth information. This allows us to determine if an object is in shadow based on the light hitting the mesh. **Figure 7** shows an example of how the depth buffer is working to calculate which mesh is lit and which one is in shadow.

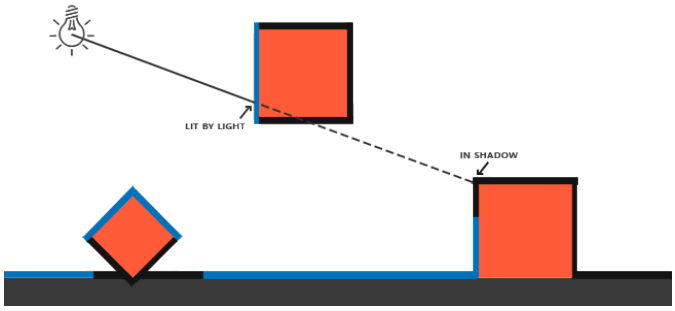

Figure 7: **Depth buffer**

In the scene the shadows are created with one spot light which is casting light on meshes in front of the wall. In order to create more realistic shadows, the projection matrix that is used is with a wider field of view. The normal angle for the FoV is $\pi/4$ and for the shadows it is changed to $\pi/2$. The reason for this is because the camera has a narrower FoV than the cone of the spot light and this may result in clipping. **Figure 8** demonstrates the shadow casting in the project.

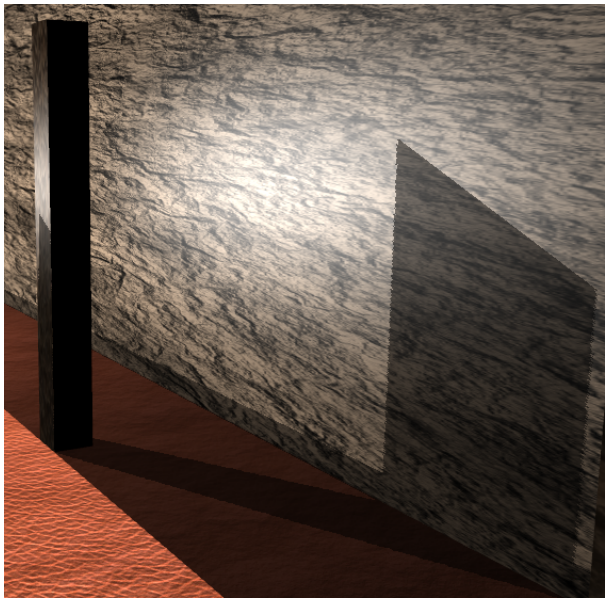

Figure 8: **Shadow**

Apart from creating shadows the depth buffer is also used to determine the depth of the water. By using this we can make the water more transparent in the shallow areas and more murky in the deep areas. More on this can be found in the water section.

3.4 Moving objects

For the implementation of the moving objects the $\sin$ and $\cos$ functions were used. By using their main property (shown on **Figure 9**) the position of the meshes are changed in a predefined range.

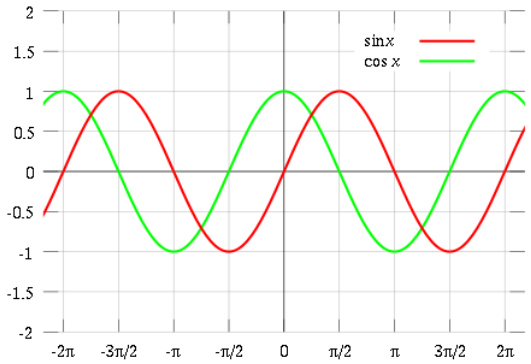

Figure 9: $\sin$ **and** $\cos$ **functions**

3.5 Hierarchical Transformations

Hierarchical transformations is a cheap way of inheriting all the scale, rotation and translation of one mesh by another, all with just one multiplication. In this project the egg protectors are an example of a hierarchical transformations. In order to achieve the final effect the model matrix of the current torus is calculated from the model matrices of the bigger toruses. **Figure 10** shows the result of this graphics effect in the project.

$$[ModelMatrix] = [Translation] * [Rotation] * [Scale]$$

Figure 10: **Hierarchical Transformations**

3.6 Skybox

The skybox is a great graphics effect that brings completeness and background to the 3D scene. The effect is achieved with a cube that has a texture applied to its inner sides and the internal parts of the cube are rendered rather than the external parts by disabling the

cull face. Cube map similar to the one on **Figure 11** can be used for the inner parts of the cube.

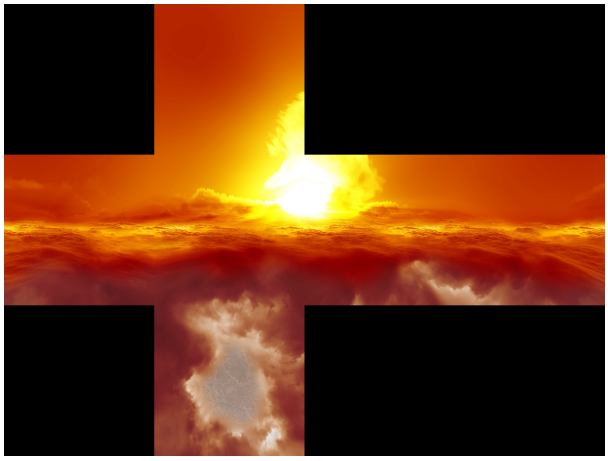

Figure 11: **Cube map used in the project**

The cube map is also used for the reflection part of the water. There we sample the cube map based on the view direction and the surface normal. More on this can be found in the water section.

3.7 Terrain

The terrain is a geometry object that makes the scene even more realistic. It is generated using a height map. **Figure 12** shows an example of a height map. A height map is a texture that contains height data where the pixel colour represents the y-component of the vertex position. The height map from **Figure12** is used in this project. The dark and white pixels represent the low and high parts of the scene, respectively.

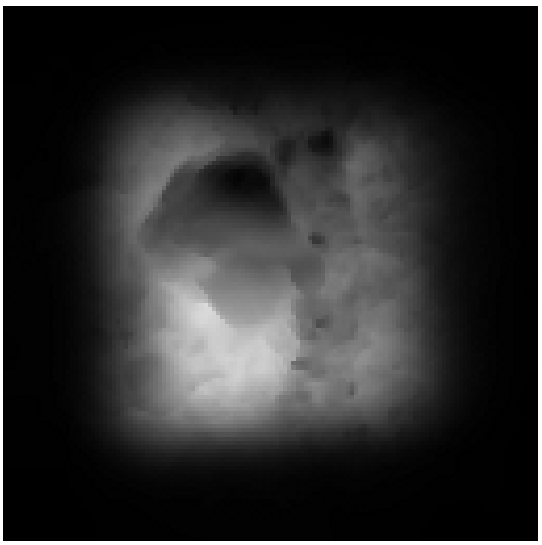

Figure 12: **Height map**

A crucial part of the terrain is multi-texturing and generating texture weights. These two features make the terrain more natural. The texture weights are generated based on the height of the terrain and later used in the shaders to blend between the textures. **Figure 13** shows an example of multi-texturing terrain from the project.

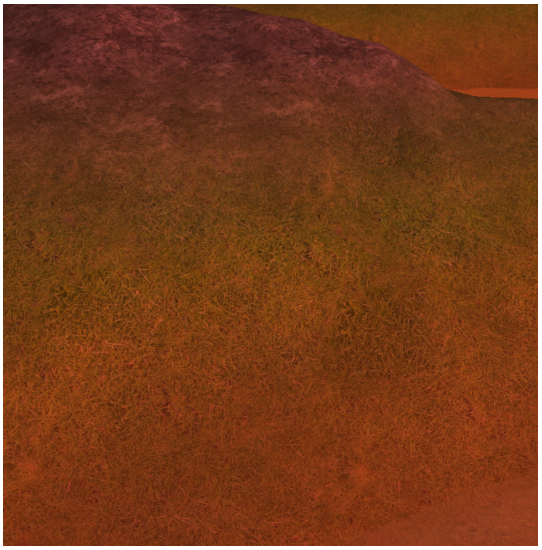

Figure 13: **Multi-texturing terrain. Sand-Grass-Rocks**

3.8 Water

The terrain representing an island is surrounded by water. A wide variety of effects are used on the water to make it more realistic:

3.8.1 Reflection

The reflective part of the water is implemented by sampling a cubemap texture that is used as an environment map. The main difference between a normal 2D texture and a cubemap texture is the way it is sampled. The latter is sampled with 3D directional vector that points in the direction of the cubemap texture that is to be sampled. **Figure 14** shows an example of cubemap reflection.

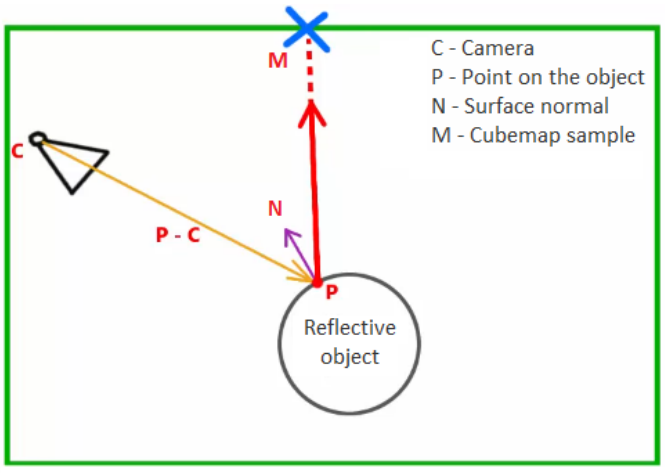

Figure 14: **Reflection with cube map**

In the project, the reflective object is a plane, hence all normals are pointing in the same direction - upwards and the used environment map is the skybox. **Figure 15** shows the reflection of the sun and the clouds on the water.

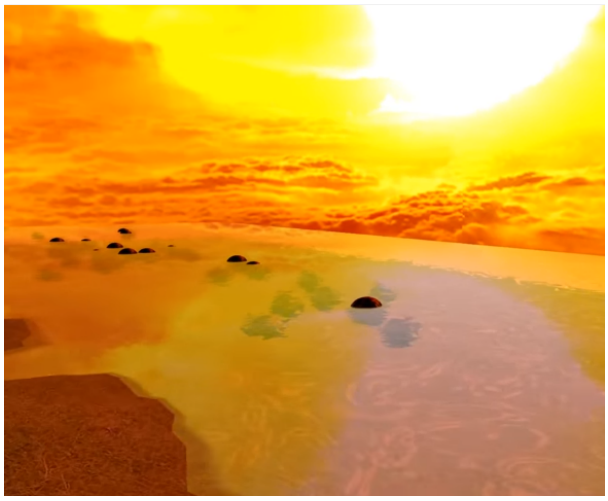

Figure 15: **Reflection of the sun and the clouds**

3.8.2 Refraction

The refraction effect is achieved by rendering the scene to a frame buffer and using the texture later when defining the colour of the water. However, rendering the whole scene twice would be extremely inefficient. For this reason, clipping plane is used and only the refracted parts of the scene are rendered the second time. Clipping planes allows us to specify a 3D plane in the world and all the geometry objects that have negative distance (they are outside this plane) are not rendered. The way to specify if something is inside the clipping plane is by using the dot product of the vertex and the plane. See **Figure 16** for references.

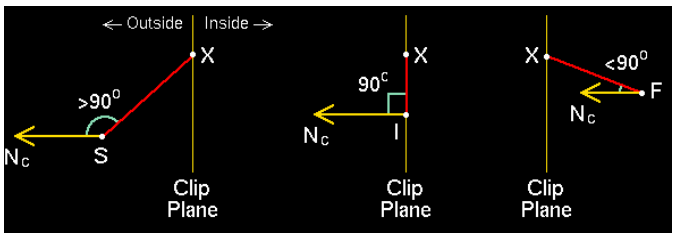

Figure 16: **Distance to clipping plane**

In this case, the level of the clipping plane is the level of the water. **Figure 17** shows the refracted terrain under the water.

Figure 17: **Refraction - terrain under the water**

3.8.3 Distortion effect

The Distortion effect is achieved with a DuDv map (**Figure 18**) which makes the water looks like it is rippling. The DuDv map is just a 2D texture with red and green lobes on it which can be represented as a 2D vector.

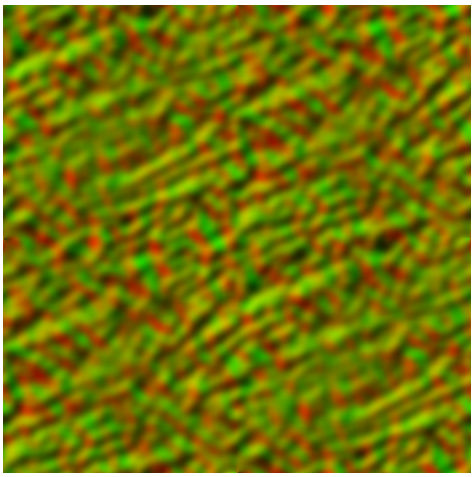

Figure 18: **DuDv map**

The way it is working is by sampling the DuDv map and then distorting the reflection and refraction textures by adding the sampled offsets to their texture coordinates. To increase the level of detail a matching to the DuDv map, normal map texture is used. It is sampled with the distorted texture coordinates and it makes the water more shiny. The result is shown on **Figure 19**. It makes the water more natural by adding this level of detail.

Figure 19: **Distorted refraction texture**

3.8.4 The Fresnel effect

After the reflection and refraction textures are captured and distorted, they are blended based on the Fresnel effect. This determines how much of each texture must be seen based on the viewing angle. When we look straight down into the water it appears to be fairly transparent and when we look across the water surface at a low angle it looks more reflective. The viewing angle is determined from the dot product of the surface normal and the view direction of the camera. The more these two vectors point in the same direction, the more transparent the water is. See **Figure 20** for reference.

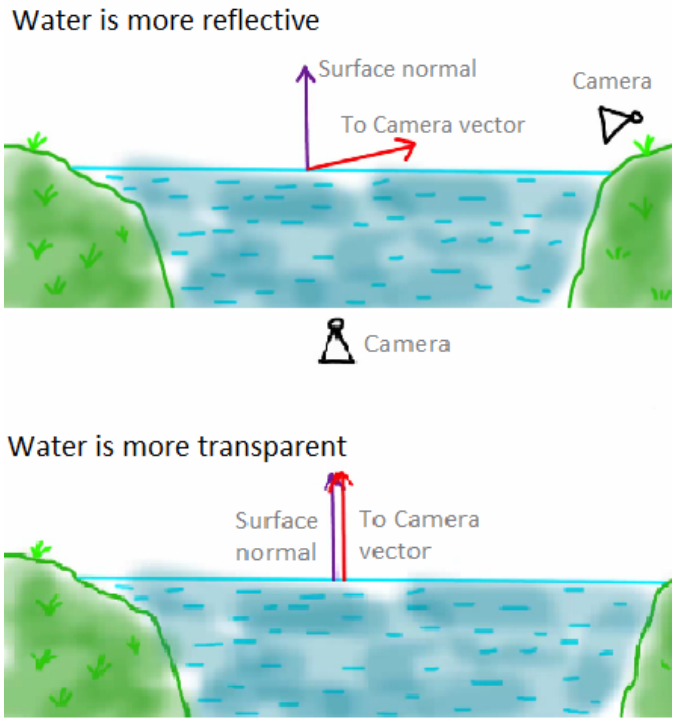

Figure 20: **Viewing angle used for the Fresnel effect**

Figure 21 shows an example of the Fresnel effect in the scene. The area of water that is closer to the camera, seems to be more transparent. The area of water that is further away from the camera is much more reflective due to the large angle.

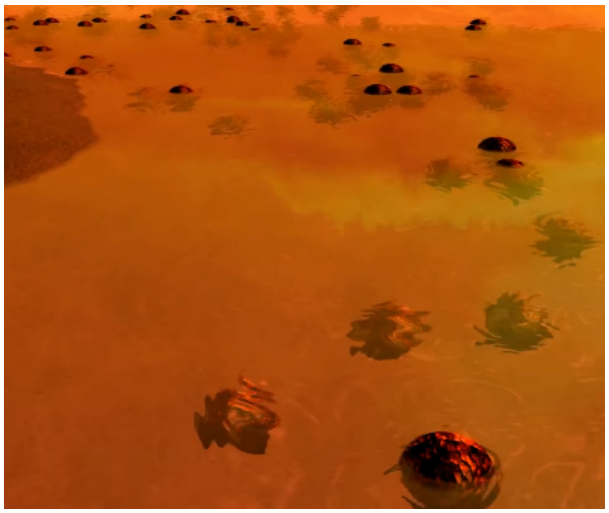

Figure 21: **Fresnel effect**

3.8.5 The depth effect

The depth effect makes the water seem more transparent around the shallow areas and more murky around the deeper areas. For this effect, the depth buffer of the refraction part of the scene and the build-in OpenGL variable - *gl_FragCoord* are used. The depth buffer returns the distance of the nearest object under the water to the camera. And the *gl_FragCoord* returns the window coordinate of the current fragment. Then we subtract one distance from the other and the result is the depth of the water from the perspective of the camera (the amount of water between the camera and the refracted object). See **Figure 22** for reference.

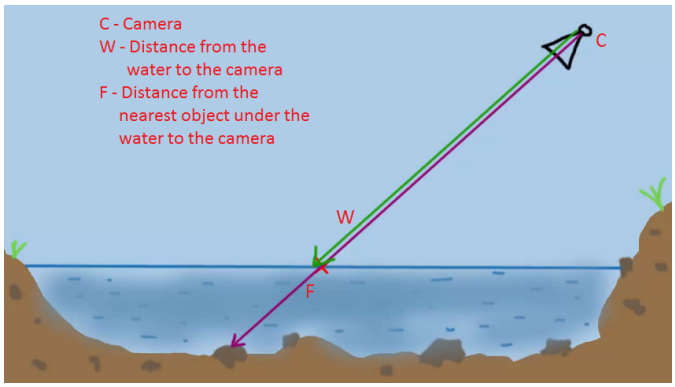

Figure 22: **Water depth**

The tricky part is to get linear depth (the true z value) from the depth buffer by using the near- and far-clip planes. The results of these calculation are shows on **Figure 23**.

Figure 23: **Water depth**

The whiter a pixel is, the further away from the camera it is. The shallow areas of the water are shown in black. In order to achieve the final result, alpha blending is used. **Figure 17** shows an example of this effect from the scene.

3.9 Post-processing effects

Post-processing is a technique where a frame is captured in a texture, different effects are applied to the colours of the pixels and it is rendered back to the screen.

3.9.1 Mask

This is a multi-texturing technique that is achieved with a black and white mask. In the fragment shader both the mask and the texture from the frame buffer are sampled and these colours are multiplied together to get the final colour. In this way only the white parts of the mask are visible in the final image. **Figure 24** shows an example from the project. The effect of zooming in is achieved by changing the FoV angle in the projection of the camera.

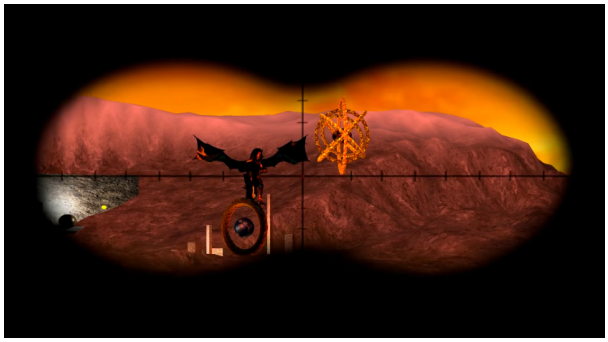

Figure 24: **Binocular mask**

3.9.2 Edge detection

Edge detection is a post-processing effect where only the edges of the geometry objects are visible. For every pixel it is determined if it is part of the edge based on the colours of the pixels around it. Suppose there are 9 pixels as shown on **Figure 25**.

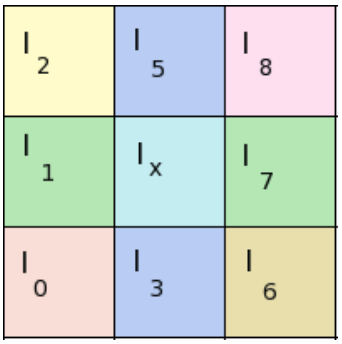

Figure 25: **9 pixels determine if it is an edge**

After applying the edge detection filter, the intensity of pixel I_x is determined from the formula:

$$I_x = \frac{|I_1 - I_7| + |I_5 - I_3| + |I_0 - I_8| + |I_2 - I_6|}{4}$$

In order to be able to calculate pixel intensity we need to find out intensity differences between neighboring pixels and average them. Then the average value is checked against a threshold. If the intensity is greater than the threshold that mean that there is hight contrast and hence this is an edge. **Figure 26** shows the edge detection effect in the project.

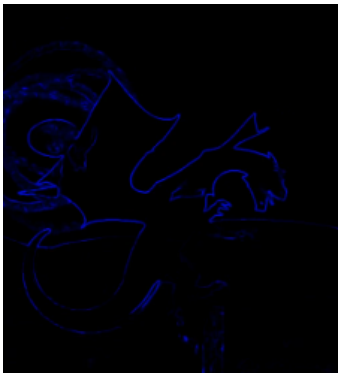

Figure 26: **Edge detection on the dragon**

3.9.3 Sepia

The Sepia post-processing effect is easily achievable by first applying the gray scale effect. In order to do that we need to work out the intensity of an individual pixel. This can be done with the equation: $i = f \cdot c$, where *c* is the vector representing the colour value and *f* = (0.299, 0.587, 0.184), which is the gray scale intensity value and gives different weight to the red, green and blue colours. After this the Sepia effect is achieved by adding the colour (0.314, 0.169, -0.090) to the outgoing gray scale.

3.9.4 Motion blur

Motion Blur is an effect that uses two frame buffers and the textures stored in them to create the required blurring. The Motion Blur can be achieved with the standard multi-texturing approach by blending the current render with the previous one.

3.10 Wire framing

Wire framing is a nice effect that shows the primitive geometry that structures all models and more complex geometry objects. **Figure 27** shows an example of this effect in the project.

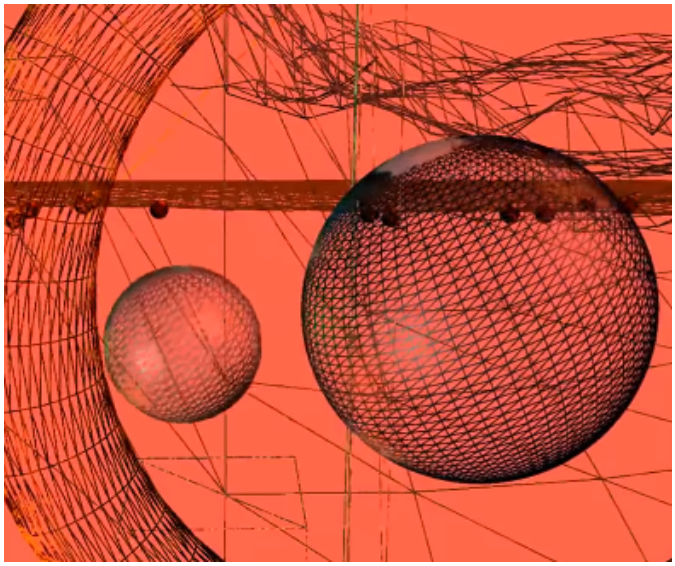

Figure 27: **Wire framing in a sphere**

4 Optimization

The scene in this project consists of quite a lot of geometry objects and additional effects. Doing optimization is crucial in this case in order to improve the performance of the GPU.

4.1 Instancing

One of the used optimization technique is Instancing. Instancing allows us to draw many geometry objects at once with a single render call, saving us all the CPU -> GPU communications each time we need to render the object. With instancing this is done only once and it avoids the performance bottleneck on the relatively slow

CPU to GPU bus. All the required data is sent to the GPU only once and then the CPU renders all the instances without having to continually communicate with the CPU. This method of drawing is really efficient for rendering a large number of geometry objects with the same set of vertex data but with different world transformations. For example, grass or rocks on the seabed. In this project, instancing is used to render 800 dragon eggs around the island under the water.

5 Future Work

The initial plan for the water was to be fully reflective (reflecting the terrain and the rest of the entities from the scene), however, because of the lack of time it was not possible to be implemented. This is something which is definitely considered for future development. In addition to this, the scene will be made more dynamic, with moving clouds and changing day/night times.

6 Conclusion

At this stage the scene was successfully finished with all the required techniques and effects. By undertaking the future work, the scene will became even more interesting and visually appealing.

References

[1] HBO, “Games of Thrones,”

[2] Edinburgh Napier University, “Previous years projects:,” http://games.soc.napier.ac.uk/graphics.html.

[3] Dr K. Chalmers, S. Serrels, “Workbook and practicals:,” https://github.com/edinburgh-napier/set08116.

[4] ThinMatrix, “Online tutorial:,” https://www.youtube.com/user/ThinMatrix.