3D GRAPHICS PROGRAMMING

CI5525

Level2: Outdoor Scene - Report

Bellido Chueco, Daniel
K1925456

CI5525 3D GRAPHICS PROGRAMMING COURSEWORK

REPORT

LEVEL 2, Project 2: **OUTDOOR SCENE**

Recipes

This outdoor scene project has been developed following the recipes:

- 2.4 - The Fog
- 2.7 - Normal Mapping
- 2.11 - Rendering Water
- 2.12 - Particle Systems
- 2.14 - Rigged Characters

The Fog

The implementation of the fog was the first task carried out after creating my terrain in Unity. As it can be seen in the image, the implementation is correct, with a natural look, colour and density that work well with the terrain, the rigged character and the skybox, which goes from having a sunny day texture to being totally gray when the key "1" is pressed.

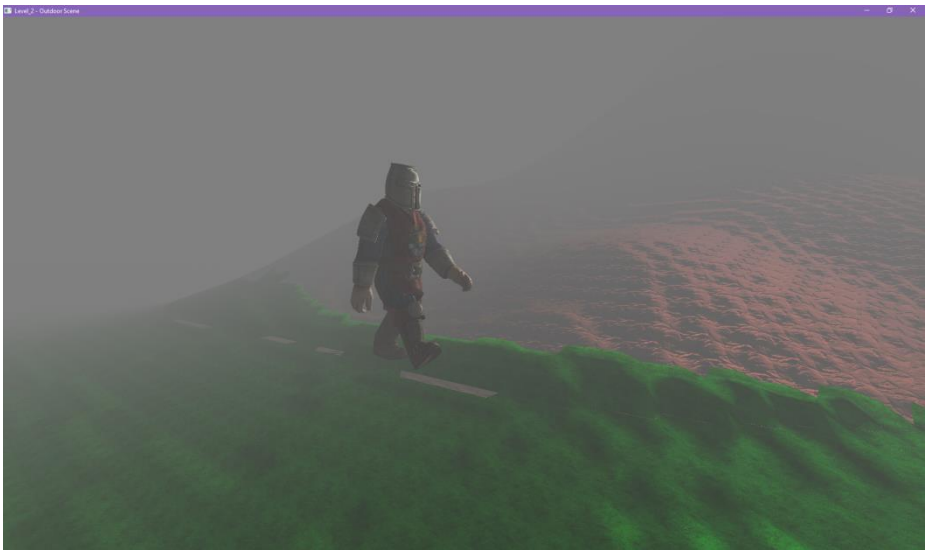

Figure 1 Screenshot taken from a previous version of the project (Backup)

However, fog is only implemented in the basic shader and the implementation of new features in this scene, in particular the water and smoke from the volcano, has caused the fog implementation to be incomplete due to using other shaders. The fog does not cover surfaces such as the smoke from the volcano, the surface of the water, nor the surface of the terrain where the normal map is applied, and, which code was transferred from the basic shader to the terrain shader in order to successfully complete the implementation of the rendering water. This problem could be solved if the fog code implemented in the basic shader were duplicated to the rest of the shaders. But due to other technical complications and lack of time, this implementation became a low priority since it only counted 5% of the total marks. Therefore, the fog has been left in a state that is not suitable for the scene but as it has been worked on and deserves a mention.

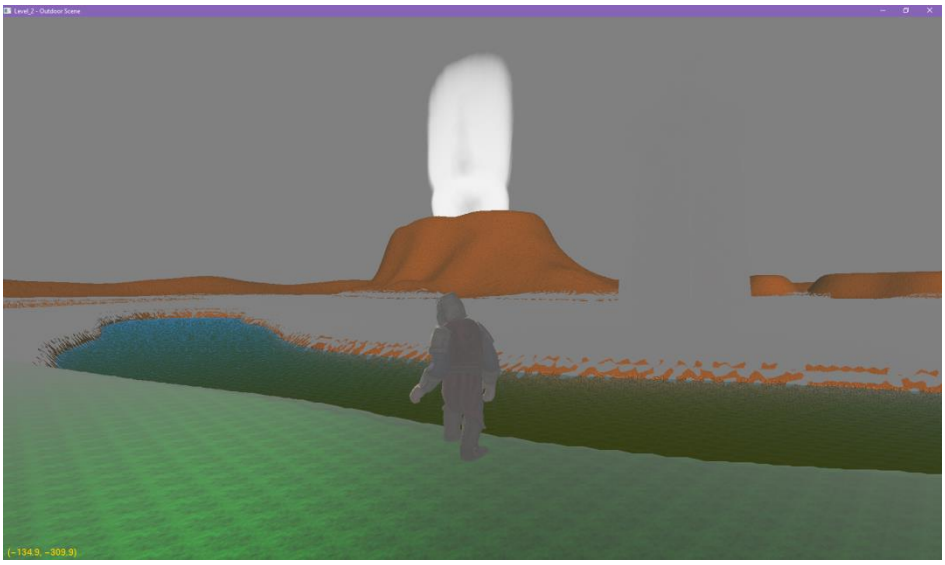

Figure 2 Fog issues in the final version of the project.

Normal Mapping

Normal mapping has been successfully implemented on different objects. The first normal map has been implemented on the ground generated by a custom height-map previously created in Unity. This normal map had been implemented initially in the basic shader but after the implementation of the water rendering it had to be duplicated in the terrain shader in order to carry out the multi-texturing of the lake shore.

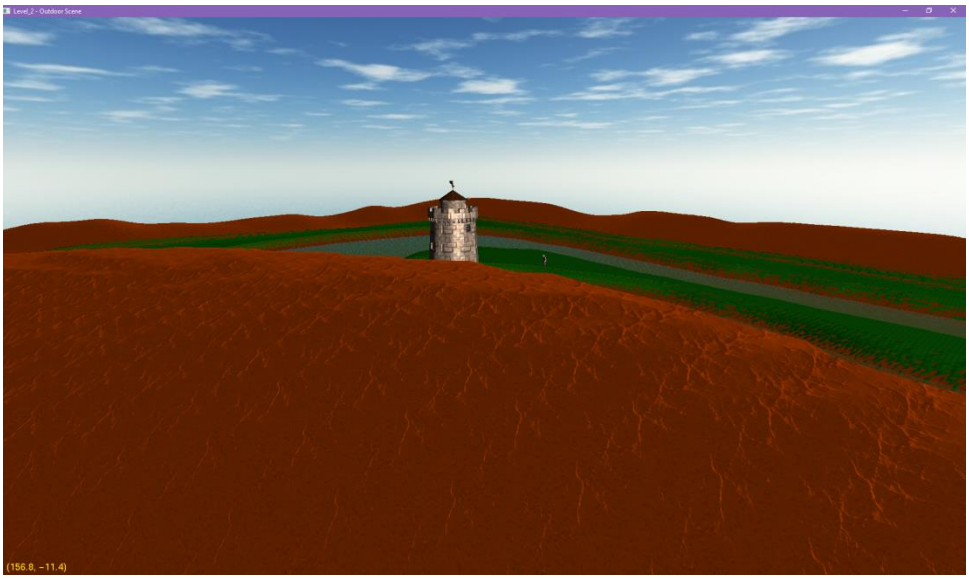

Figure 3 Normal map applied on terrain. View from volcano.

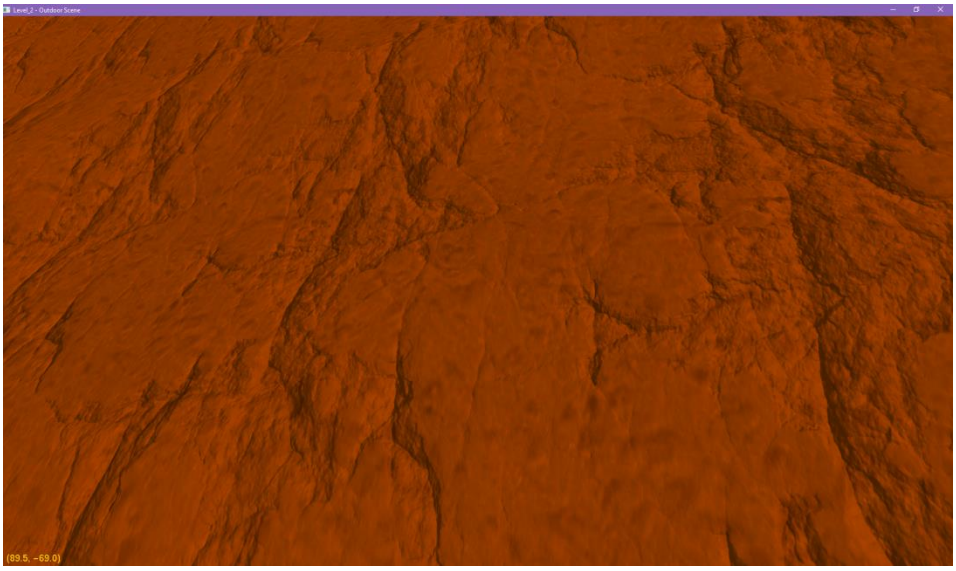

Figure 4 A closer look to the Normal Map on the terrain

The second normal map is applied to the rigged characters to which the multi-texturing technique is also applied.

Figure 5 Multi-textured Rigged Character

The normal map applied is consistent with the composition of the scene and the implementation of the point light located with the sphere of the sky, the sun, intended to be an animated light slowly orbiting the terrain to make the effect of the normal map more visible with changes of light. Being an implementation with lower priority than other techniques, the idea has had to be discarded due to lack of time.

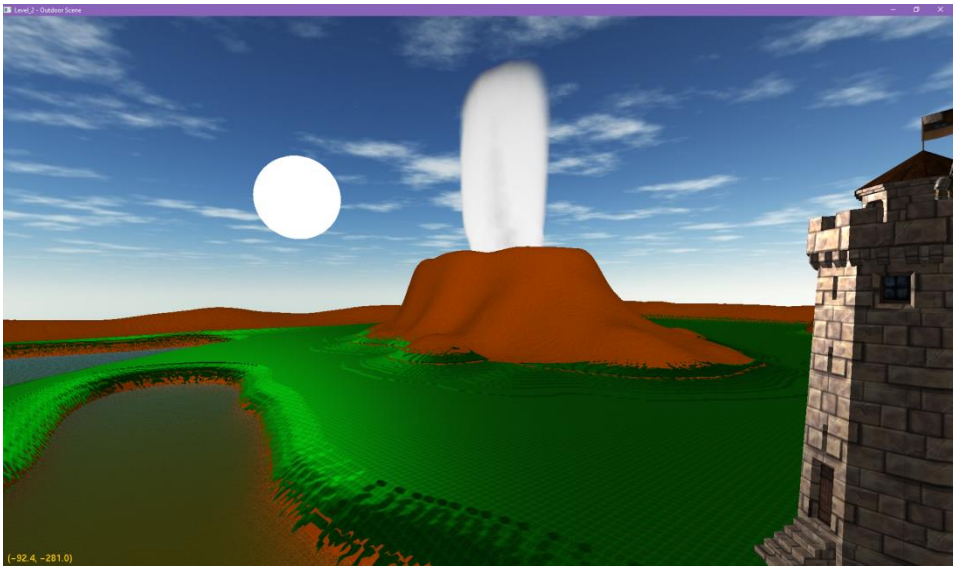

Figure 6 Point light simulating the Sun (Not animated)

Rendering Water

To implement the Lakeside scene it was essential to have an exterior scene with multiple shaders (water and terrain shaders), which has been successfully achieved. In addition, the smooth multi-texturing transition effect between the pebbles and the rock above the water level has been achieved. Additionally, another terrain has been rendered that gives the scene another layer with a different texture without the normal map affecting it.

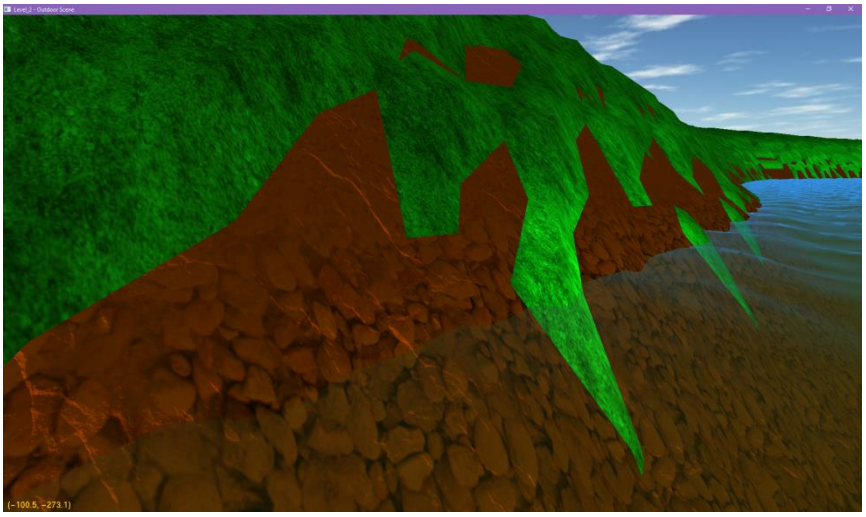

Figure 7 Multi textured shore line

Other implementations on the topic of rendering water are reflected and refracted light combined to provide the colour of the water surface, as well as underwater mist that enhances the colour of the water on the surface. In addition, the use of a completely custom terrain designed to solve the end-of-the-world problem, the composition of the scene remains totally consistent and interesting.

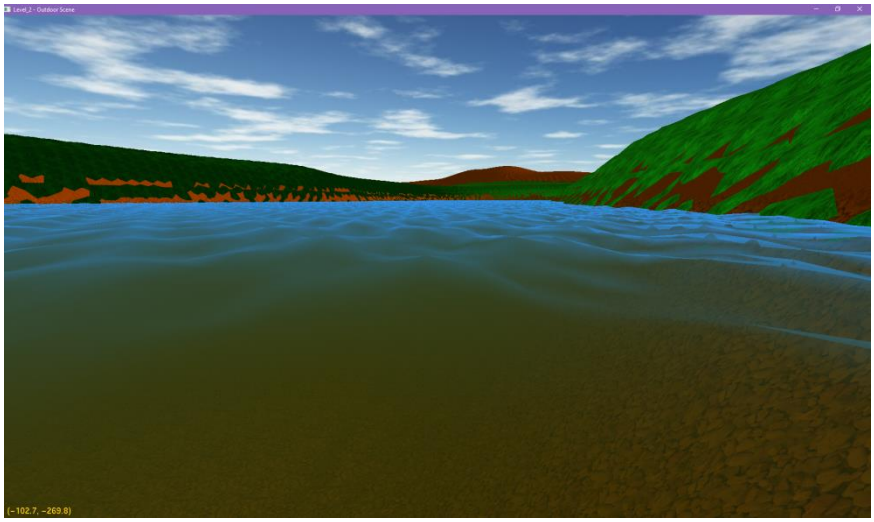

Figure 8 Water rendering with all the expected requirements implemented.

Particle Systems

The proposed particle system for this scene is the implementation of smoke rising from the top of the volcano. These particles have their own shaders and are positioned, scaled, and textured in a way that makes the smoke effect appropriate for the scene.

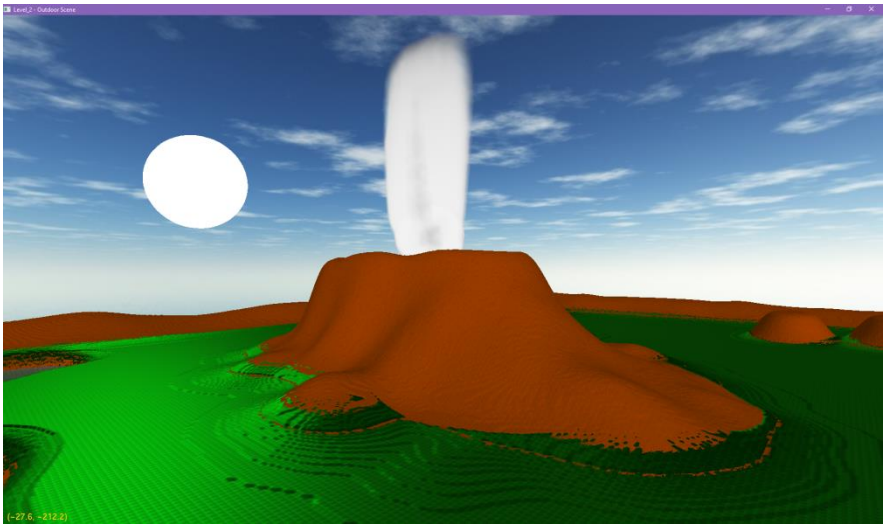

Figure 9 Volcano with particle system (smoke)

Other particle systems were planned to be implemented but the feedback received suggested that it was not necessary to implement other particle systems. But the idea was to implement some fire particles from the top of the volcano and some ash rain.

Rigged Characters

Two medieval-looking guards have been added to the rigged character topic. One of them starts with a walking animation until it reaches the shore of the lake, once there it stops and starts another idle animation as if looking for a way to pass. On the other side of the lake is a watchtower and on top of it is another guard who has another animation as if he wants to get his attention. So there are two characters, a total of three sets of animations, and they are in a meaningful scene.

An attempt has been made to apply an interactive mode from which to control one of the guards with the keyboard, but it has not been possible. It has also been tried to make the guard draw a predetermined route but for strange reasons every time a rotation was applied to apply a change of direction the guard's position changed in a very strange way causing him to teleport to another point on the map.

Thus, it has only been possible to get the camera to automatically follow one of the guards until he stops and changes his animation.

Figure 10 Scene with two animated rigged characters

To get the camera to follow the guard, a function called cameraChase() has been created, which takes a float time as its only parameter, which consists of placing the camera behind the guard. For it to move, simply multiply the value of the X and Z axes by the time parameter multiplied by 2. As a last step, you only have to call this function within the renderScene() function, which makes the "time" parameter increase in each frame.

```
void cameraChase(float time)
{
    matrixView = lookAt(
        vec3(vec3(-185 + (time * 2), 30, -360 + (time * 2))),
        vec3(-165 + (time * 2), 30, -340 + (time * 2)),
        vec3(0.0, 1.0, 0.0));

}
```

Figure 11 Snippet of the function that makes the camera chase the guard