Terrain Flight Project for EDAN35

Charlie Mrad* Melker Gustafsson†

Lund University
Sweden

1 Abstract

Recreating the earth in 3D graphics provides many opportunities to use advanced rendering effects. We chose to do it with a deferred shading renderer that provides us with easy shadows and good performance when using many lights. To increase the realism of the recreation we used geographic data taken from NASA and used techniques for ocean shading that add wave texture and foaming. The results are a demo that features an airplane flying over the earth with colorful oceans and geographically accurate mountains. The performance was never prioritised but if it had been using a deferred shading renderer would have been scrapped in favor of more straight forward rendering methods since we did not take full advantage of the deferred shading.

2 Introduction

We have made a prototype of a game where the user flies an airplane over the earth. We got the project idea from a Youtube video of somebody doing a similar project in Unity [1], and we got very inspired. Our initial plan was to include multiple light sources apart from the sun in the form fire and explosions, hence we decided to use deferred rendering. This project involves several rendering techniques used in the course already but also allowed us to dive deeper into deferred shading and using render passes to accomplish the effects we wanted. We had also initially planned use the jump flooding algorithm [2] to create a the effect of a wave-foam around coastlines but we ended up scrapping the idea when it turned out to not be feasible. Additionally we were also thinking of using compute shaders to create cloud particles that could be interacted with. The effect would have been similar to the one created in [1] but we did not have time to fully implement that.

3 Application

For this project we chose to build upon the deferred shading assignment from the course. We also merged some code from previous assignments and the previous course (EDAF80), such as the parametric sphere mesh generating code.

3.1 Earth

The earth geometry was based of a highly tessellated sphere with around $10800 * 5400$ triangles. We chose this number in order to match the number of pixels present in the height map texture used for the earth. In hindsight, this doesn't help us much since the geometry is generated using a simple UV-sphere technique (i.e mapping a cylinder to the sphere). This produces a big variety of vertex densities that favors the poles and therefore stretches the texture at the poles and also means that vertex density and thus height map accuracy is poor at the equator. An alternative could have been to implement a sphereized cube or a subdivided icosahedron, although the lack of control over face number makes the former more attractive.

3.1.1 Surface

The surface used textures from NASAs visible earth webpage[1] and from JHT's planet pixel emporium[2]. The textures used were all scaled to dimensions 10800 x 5400. We used in total a texture for the diffuse color, specular mapping, normal mapping of land areas and height mapping. All texture data was used when filling the G-buffer like for any other piece of geometry.
However the height mapping works by offsetting the vertices in their normal direction but the height value used is actually blurred across the neighbourhood of the sampled value from the height map. This is done using a simple box-blur algorithm and produces much smoother results when height values start increasing sharply.

3.1.2 Ocean

In order to create the oceans we used two normal maps for wave patterns and scrolled the both in opposite directions of each other so that the water doesn't look like it is flowing in any particular direction. In order to differentiate land from water we checked at the values for the specular map as it so happens that the specular map is set to 0 wherever there is land and 1 otherwise. So if the specular map was greater than 0 at the fragment being rendered then it would apply any water effects if the wave textures were present.
As for the wave-foam patterns around coastlines; that was accomplished by generating a texture iteratively in a prepass we dubbed "pass 0". This pass reads in a texture that is initalized to the specular map from before but with land encoded in the red channel and the oceans encoded in the blue channel. It then looks around each ocean pixel in that texture that hasn't been processed yet and chooses the smallest value of any neighbouring ocean pixel as its value only if any of the neighbouring ocean pixels have been processed already. If a land pixel is found in the neighbours the pixel value is set to be a default coastline pixel value, and if no neighbouring pixels have been processed already then nothing happens. Essentially this process gradually builds up a somewhat square shaped distance map that maps each pixel to a distance from the coastline. The square nature of the distance map is much more evident with small islands than with bigger landmasses.
To finalize the effect the generated distance map is put through a sine function that is phase-shifted by the distance and animated by the elapsed time in seconds. And then in order to add some variation we also factor in some 3 dimensional Perlin noise sampled using the texture coordinates and elapsed time. This all then added on to the diffuse color texture.

3.2 Airplane

The airplane is made using a the Node class from the previous course assignments. It has a movement direction across the surface expressed as an angle and a movement speed. When moving it calculates a vector that is tangential to the surface and rotates it by its movement direction. Then its a simple process of translating the airplane in that direction as well as correcting it back to the altitude it should have so that it does not end up getting further away from the planet. In order to render the plane we created a second

*e-mail: ch3045mr-s@student.lu.se
†me1873gu-s@student.lu.se

[1] https://visibleearth.nasa.gov/collection/1484/blue-marble?page=2
[2] http://planetpixelemporium.com/earth8081.html

rendering loop during the G-buffer pass that considers the nodes in the scene and renders those to the G-buffer as well.

4 Results

Our project work resulted in a prototype of a terrain flight over the earth which can be used as a base for implementing a game. Figure 1 shows our application running in flight mode where the camera follows the airplane which is controlled by the user. Figure 2 shows spectator mode where the camera is disconnected from the airplane and the user is free to inspect the earth from different angles and distances.

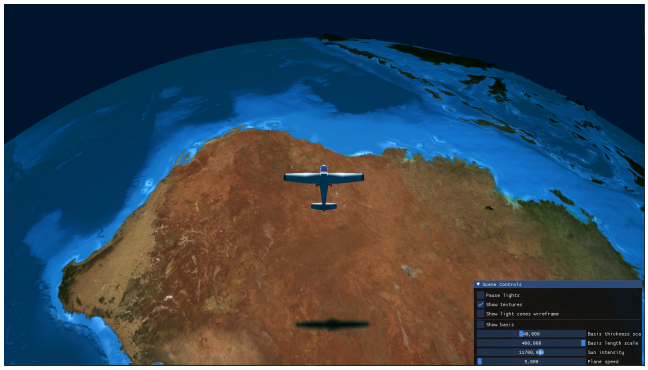

Figure 1: Flight mode

Figure 2: Spectator mode

From a performance perspective, we did not fully utilize the deferred shading technique since only one light source was added, namely the sun. However, if the application was to be extended, the current framework that is used gives room for more light sources to be added efficiently. Apart from the shading technique, we did not have performance in mind while implementing the application.

Our application is not only a base for developing games. It can for example also be extended to be used as a geographical learning tool by adding geographic data of the locations of the earth which has to be pinpointed by the user. This means that our application is a prototype for a broad range of uses.

5 Discussion

We are satisfied with the results and finished most of the objectives that we set up initially. What could be improved is the level of detail, for example by adding clouds, boats on the ocean, or even buildings on the mainland that light up during night which would result in a more immersive experience. Unfortunately time did not allow us to do this.

We tried to set up a particle system that would represent the clouds from a compute shader. Our idea was to represent clouds as a cluster of spheres, where each new sphere would spawn on a random location on the surface of an already existing sphere. A fractal noise value would be computed for each existing sphere to determine which one to add a new sphere to. This would result in clusters of spheres that would look like clouds. Unfortunately we did not get the compute shader to work in the time frame that we had, but instead this is an area for further improvement.

References

[1] Sebastian Lague. *I Tried Creating a Game Using Real-world Geographic Data*. URL: `https://www.youtube.com/watch?v=sLqXFF8mlEU&ab_channel=SebastianLague`. (accessed: 05.11.2021).

[2] Guodong Rong and Tiow-Seng Tan. "Jump flooding in GPU with applications to Voronoi diagram and distance transform". In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 2006, pp. 109–116.