|  |
| --- |
| University of Portsmouth |
| Coursework Report |
| Graphics and Computer Vision – UP918156 |

|  |
| --- |
| 12-20-2019 |

Task One

## Introduction

In this task we are required to design and implement a WebGL application that showcases the Earth with a satellite orbiting around it. This report will discuss the design and implementation of this application as well as highlight the most important components of the application.

## The Requirements

This graphical application has several key requirements. The following will highlight the key components of the application’s requirements:

1. The scene is centred around the Earth as a satellite orbits it
2. The scene is illuminated from the top right corner by a directional light at a 60-degree angle with the horizontal plane.
3. The earth has a radius of 10 units and is mapped with a texture image of the Earth
4. The earth will be animated with a slow rotation
5. The satellite consists of a main cubed body as well as three golden rods
6. Two rods carry blue solar panels
7. One rod carries a golden desk antenna
8. The side of the cube carrying the antenna is black and always faces the Earth
9. The orbit radius and speed of the satellite can be altered using the arrow keys
10. The users will have full control of the viewport using the mouse, alt and shift keys

## The Design & Implementation

In order to create this application, a full implementation of the graphic pipeline is needed. Luckily, this has already been covered during the tutorials portion of the module. Therefore, this project will be based on the foundations laid down by those sessions. This report will focus on the implementation of the requirements as well as provide evidence of the validity of the implementation of these requirements. The following subsections will highlight the implementation of the object models, surface attributes, lighting, animations, interactive controls, scene controls and the outputs and tests of these.

### The Object Models & Surface Attributes

This scene has two main objects, The Earth and the satellite. Starting with the Earth. Three functions are responsible for modelling and drawing the Earth:

1. setupSphereBuffers()

This function creates buffers of the vertices’ positions, normal, textures, and indices. Then, it calculates and populate these buffers with the data for the geometry of the sphere. It should be noted that the algorithm for calculating is based on the code found in the following website:

<https://bl.ocks.org/camargo/649e5903c4584a21a568972d4a2c16d3>

1. drawSphere()

This function is responsible for binding the sphere’s vertex position, normals, textures, and indices buffers to the gpu.

1. drawEarth(Texture)

This function is responsible for pushing the model view matrix to the model view matrix stack, manipulating the model view matrix, uploading the model view matrix and normal matrix to the shader, drawing the sphere with the desired texture, and finally popping the original model view matrix from the stack.

Similarly, the satellite is drawn using three similar functions; setupCubeBuffers(), drawCube(), and drawSata().

Two key differences distinguish the Earth functions from the satellite functions. First, the cube vertices are assigned manually while the sphere’s are calculated using two for loops. This is done because the sphere contains many vertices that can vary depending on the number of longitude and latitude bands.

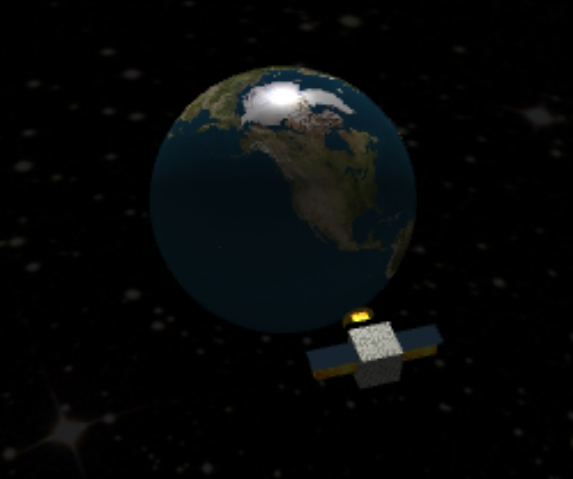
Secondly, the drawSata() function contains the specifications to drawing all the different components of the satellite, due to it being a compound shape with various objects and textures.

Figure 1 Output of the models being drawn to the exact specifications

### Animation & Lighting

This scene has three key animations. Those being, the rotation of the Earth, the rotation of the satellite and the orbit of the satellite.

In order to rotate the Earth we apply a transformation to its model view matrix using the rotateY() function as follows:



The rotation of the satellite is as follows:



Finally, the orbit of the satellite is implemented as follows:



The lighting is implemented in the fragment shader and setupLights() function. The following snippet alters the position of the spot light to be 60 units above the horizon in the Y axes:



The following images showcase the scene at different intervals with the light being applied at 60 units in the Y axis:

Figure 4 Interval 3

### Scene and Orbit Controls

Figure 2 Interval 2

Figure 2 Interval 1

The scene can be manipulated using the mousewheel to move the view port along the Z-axes as well as the shift and alt keys to alter the X-axs and the Y-axes respectively.

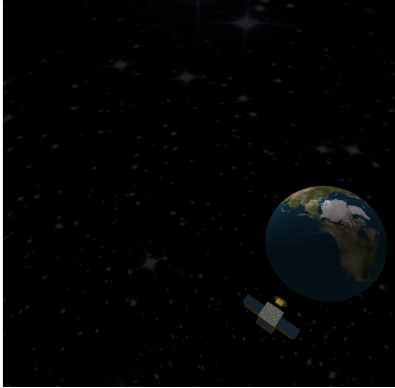
The following images demonstrate those translations:

Figure 5 X-axes

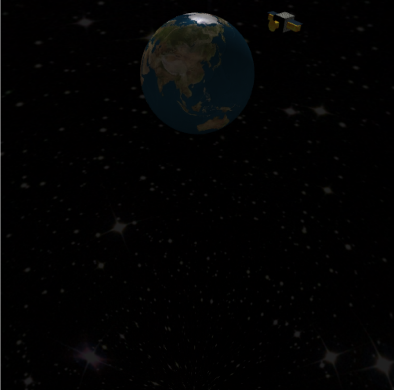
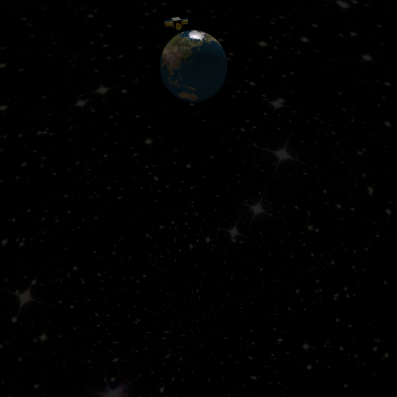


Figure 7 Z-axes

Figure 6 Y-axes

Moreover, the radius of the orbit as well as the speed of the satellite can be altered using the arrow keys. Due to the difficulty of demonstrating the change of speed in the report, a console log is also outputted to demonstrate the current speed. The following images demonstrate the change in radius and speed of the satellite:

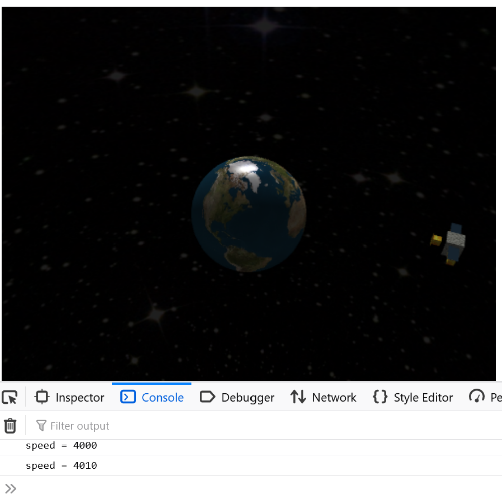
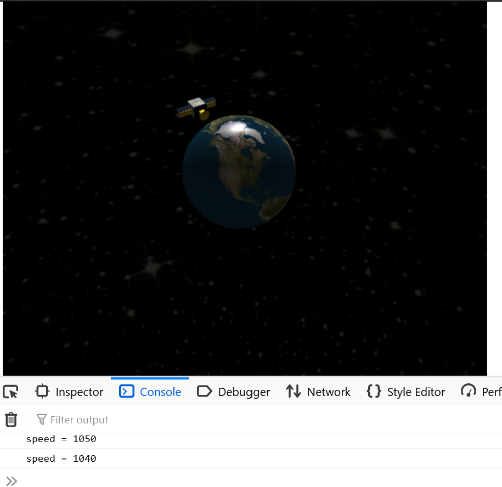


Figure 3 Large and slow orbit

Figure 8 Fast and small orbit

## Difficulties

During the development of this project, two issues were encountered. The first was due to an oversight of assigning the wrong index buffer to the drawElements() function for drawing the spheres. This prevented the spheres from being drawn and throwed an error of the buffer being too small. As for the second issue, the orbit of the satellite was disfigured due to applying the animation inside the drawSata() function as well as in the draw() function. This was fixed by eliminating the unnecessary transformation inside the first function.

## Conclusion

This project used WebGL and Javascript to draw an animated scene that contained the Earth with a satellite orbiting around it. It is my belief that all specifications were met, and that the final application exhibits no known strange behaviour and functions as intended. This project’s implementation was based on the materials provided by this module’s tutorials, in addition to the algorithm of the sphere geometry that was referenced in the code and this report.