UP2024896

Report for task 2 Computer Vision and Graphics

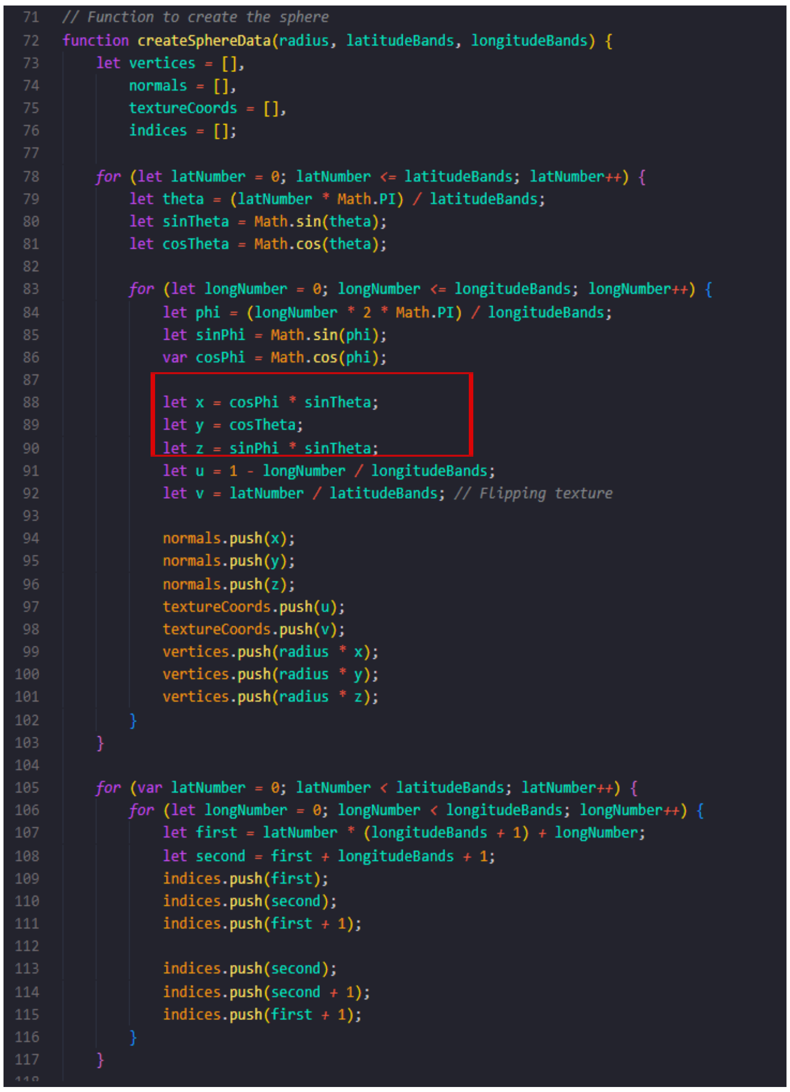
Introduction

The project involves the creation of a WebGL simulation, visualising a satellite orbiting the Earth. This simulation is developed to showcase 3D geometry creation, lightning, texturing, and user interaction.

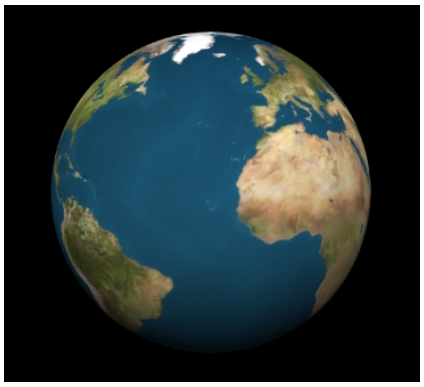
Objective 1: 3D Model of the Earth

Challenges:

My main challenge was figuring out how to implement the algorithm that would generate vertices, normals and texture coordinates for a sphere, ensuring a detailed representation in a 3D space. I will attach the entire function and go through my logical process.



```javascript
// Function to create the sphere
function createSphereData(radius, latitudeBands, longitudeBands) {
    let vertices = [],
        normals = [],
        textureCoords = [],
        indices = [];

    for (let latNumber = 0; latNumber <= latitudeBands; latNumber++) {
        let theta = (latNumber * Math.PI) / latitudeBands;
        let sinTheta = Math.sin(theta);
        let cosTheta = Math.cos(theta);

        for (let longNumber = 0; longNumber <= longitudeBands; longNumber++) {
            let phi = (longNumber * 2 * Math.PI) / longitudeBands;
            let sinPhi = Math.sin(phi);
            var cosPhi = Math.cos(phi);

            let x = cosPhi * sinTheta;
            let y = cosTheta;
            let z = sinPhi * sinTheta;
            let u = 1 - longNumber / longitudeBands;
            let v = latNumber / latitudeBands; // Flipping texture

            normals.push(x);
            normals.push(y);
            normals.push(z);
            textureCoords.push(u);
            textureCoords.push(v);
            vertices.push(radius * x);
            vertices.push(radius * y);
            vertices.push(radius * z);
        }
    }

    for (var latNumber = 0; latNumber < latitudeBands; latNumber++) {
        for (let longNumber = 0; longNumber < longitudeBands; longNumber++) {
            let first = latNumber * (longitudeBands + 1) + longNumber;
            let second = first + longitudeBands + 1;
            indices.push(first);
            indices.push(second);
            indices.push(first + 1);

            indices.push(second);
            indices.push(second + 1);
            indices.push(first + 1);
        }
    }
```



The function loops through latitude bands for each calculating two angles(‘**theta’** and **‘sinTheta’**). ‘**Theta**’ represents the angle from the top of the sphere (north pole) to the current latitude band. Inside the latitude loop, another loop iterates over longitude bands. For each longitude band, angles(‘**phi**’,’**sinPhi**’,’**cosPhi**’) are calculated. ‘**Phi**’ represents the angle around the sphere.

The red rectangle in the screenshot represents the cartesian coordinates, also calculated by the function. To properly place these coordinates in 3D space, multiply them by the sphere's radius. Since the normals point outward from the centre, they are the same as the vertex coordinates (x, y, z) for a sphere with its centre at the origin.

Another challenge during testing was that the earth's texture was applied upside down. It was a straightforward fix, although it took longer than expected. The cause of the issue was the way texture coordinates (u,v) were mapped onto the sphere. To resolve the issue, I have adjusted the calculation of the ‘**v**’ variable.

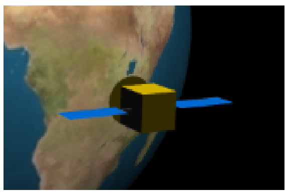
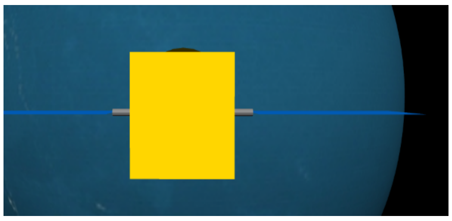
Objective 2: Satellite Model

Challenges:

While creating the satellite model, particularly the main body represented by a cube, was relatively straightforward, challenges arose when developing the other components, especially the rods. Initially, I had planned to implement the rods in a manner that would span from one side of the cube to the other, effectively passing through the main body. This approach minimises the number of draw calls in the rendering function.

However, this initial method led to several complications. For one, the rods were incorrectly inheriting the colour properties of the cube, which was different from the intended design. Additionally, this implementation posed difficulties in correctly positioning the solar panels, as they interacted unfavourably with the rod's placement and rendering.

To resolve these issues, I decided to re-write the rod rendering function. The revised approach involved drawing each rod separately, ensuring they were independent of the cube's properties and correctly positioned relative to the satellite's main body. This change significantly improved the model's realism and allowed for the correct placement and rendering of the solar panels.

Objective 3: Satellite Orbit Mechanics

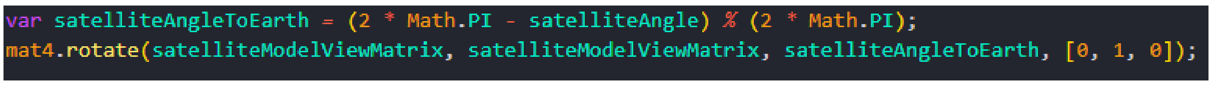
Challenges:

The development of this objective proved to be the most challenging aspect of the entire project. The initial complication arose when the satellite's orbit appeared disproportionately smaller than Earth's. This issue was relatively simple to address, requiring an adjustment to the satellite's orbit radius dynamically based on the Earth's sphere size.

Subsequently, an unexpected behaviour emerged as I integrated keyboard functionality for controlling the satellite's speed. When the satellite's speed increased, it began to spin around its axis, an effect that was tied to the Earth's rotation speed.

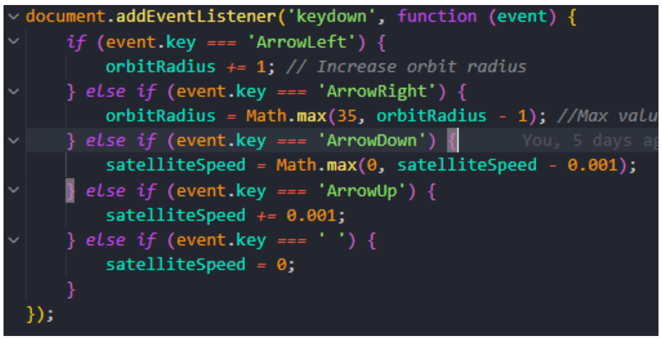
It became clear that the satellite's rotation was incorrectly linked to the Earth's rotational speed, and this linkage was amplified when the satellite's speed was adjusted through the keyboard controls.

It was a complex issue as it included the requirement for the satellite to face the Earth. I implemented a solution that dynamically adjusted the satellite's rotation based on its position in orbit. This was achieved by calculating the angle between the satellite's current position and the Earth's centre. The satellite's model-view matrix was then updated to include a rotation that aligned its facing direction with this calculated angle.



```
var satelliteAngleToEarth = (2 * Math.PI - satelliteAngle) % (2 * Math.PI);
mat4.rotate(satelliteModelViewMatrix, satelliteModelViewMatrix, satelliteAngleToEarth, [0, 1, 0]);
```

Objective 4: Interactivity



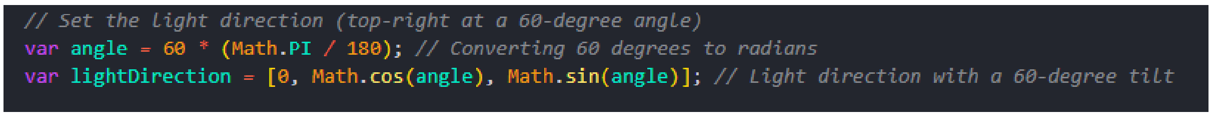
```
document.addEventListener('keydown', function (event) {
    if (event.key === 'ArrowLeft') {
        orbitRadius += 1; // Increase orbit radius
    } else if (event.key === 'ArrowRight') {
        orbitRadius = Math.max(35, orbitRadius - 1); //Max valu
    } else if (event.key === 'ArrowDown') {        You, 5 days ag
        satelliteSpeed = Math.max(0, satelliteSpeed - 0.001);
    } else if (event.key === 'ArrowUp') {
        satelliteSpeed += 0.001;
    } else if (event.key === ' ') {
        satelliteSpeed = 0;
    }
});
```

Challenges:

In this section, there were no significant challenges. However, I have implemented extra interactivity so that the satellite will stop immediately when the space is pressed. Also, I ensured the satellite would not collide with the Earth when we decreased the orbit size by pressing the key.

Objective 5: Lightning

I chose a directional light source positioned at a 60-degree angle from the horizontal plane for the lighting.



```
// Set the light direction (top-right at a 60-degree angle)
var angle = 60 * (Math.PI / 180); // Converting 60 degrees to radians
var lightDirection = [0, Math.cos(angle), Math.sin(angle)]; // Light direction with a 60-degree tilt
```

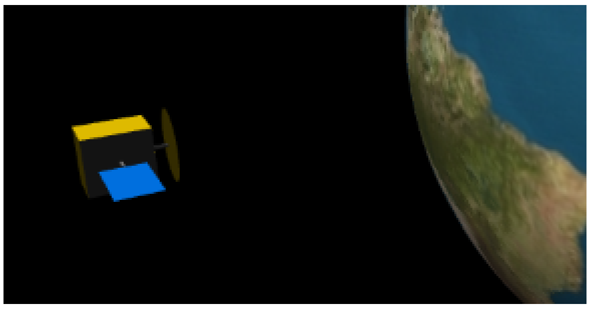
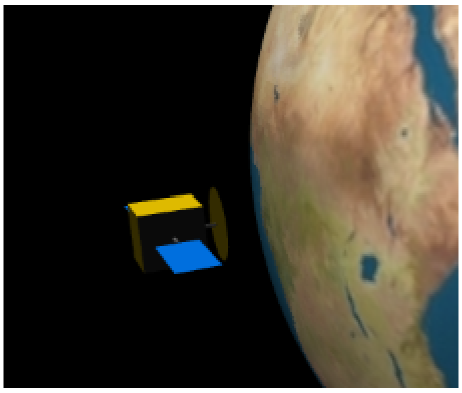
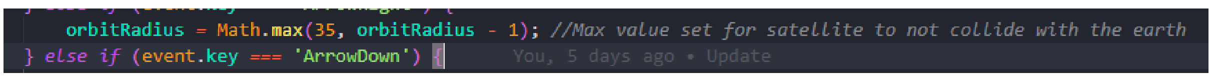
Challenges:

The main difficulty was incorporating the normals into every function that generated data for the objects. I ran into the problem that the lightning didn't work with the Earth, but it did with satellites. A specific issue arose with the lighting effects on the Earth model. While lighting was successfully applied to the satellite, it did not function as expected on Earth. Realising that the lighting needed to be applied directly to the sphere's surface rather than just its texture was the solution.

Testing

Test 1:
Satellite radius change

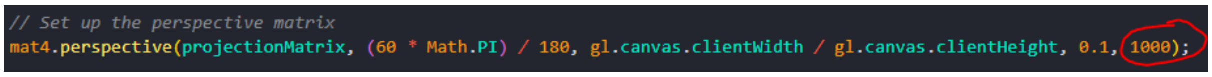
This test shows that the satellite will not collide with the Earth.

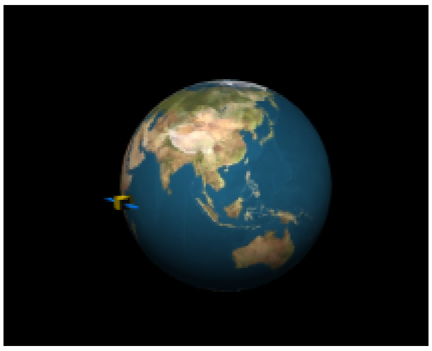
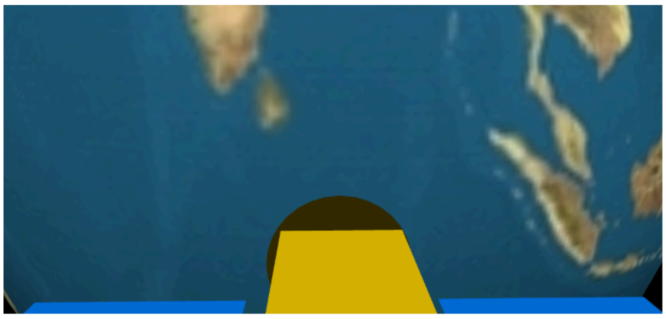
```
    orbitRadius = Math.max(35, orbitRadius - 1); //Max value set for satellite to not collide with the earth
} else if (event.key === 'ArrowDown') {        You, 5 days ago • Update
```

Test 2:
Earth and satellite visibility test

This test is to check the visibility of the objects. If the value were smaller, then the objects like a satellite would not be visible to the user.

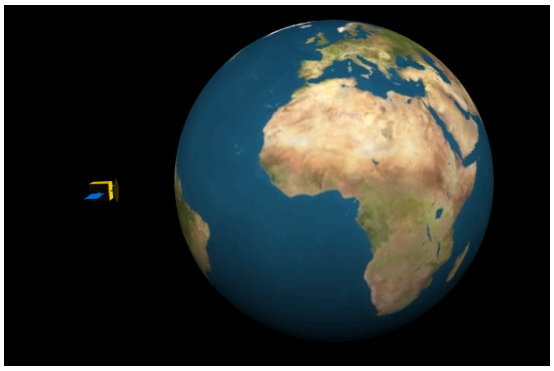
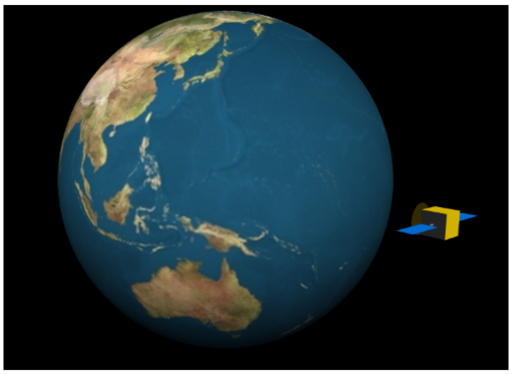


```
// Set up the perspective matrix
mat4.perspective(projectionMatrix, (60 * Math.PI) / 180, gl.canvas.clientWidth / gl.canvas.clientHeight, 0.1, 1000);
```

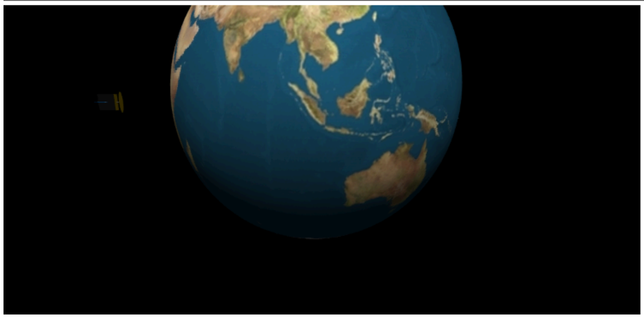
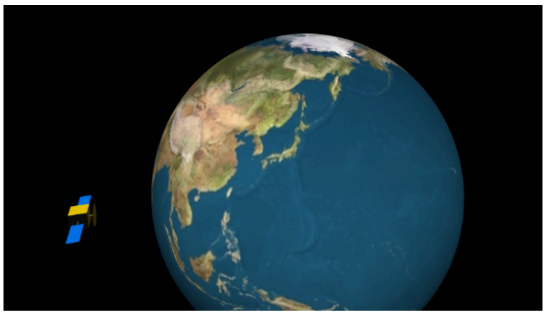
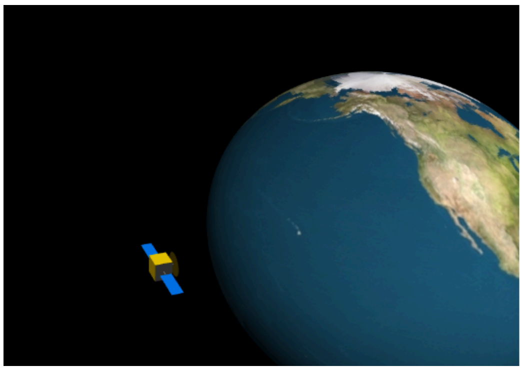
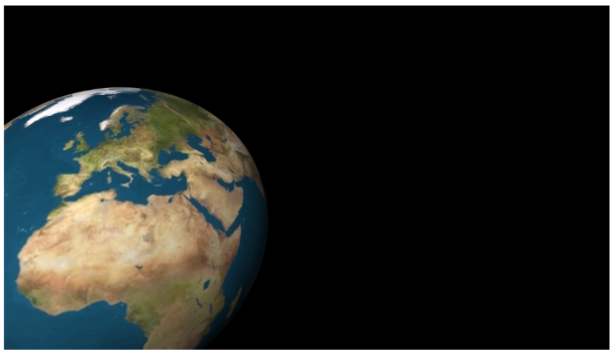



Test 3:
Satellite position change

This test proves the ability to steer the satellite, although it’s difficult to show on a static image.

Test 4:
View change (x and y axis)
This test shows the ability to adjust the scene by the use of a mix of keys,

Test 5:
Test of the mouse control and texture
This test shows the proper texture of the earth, and that rotation by the use of a mouse is possible. North and south poles are present in the picture.