BBM414 Computer Graphics Lab.
Assignment 3 - Report

Berk Bubuş
21945939
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21945939@cs.hacettepe.edu.tr

Overview

In this assignment, we have 2 parts. The first part is about moving and changing some situations about a star with buttons. The second is about creating a solar system with the sun, the earth, and the moon, we have got some properties like scaling, rotation speed, and directions, etc. of these 3 objects. Also, we got buttons for changing these properties.

1 Part 1

In Part 1, at HTML file, I added some buttons according to the pdf. There

```
<div display="block" >
<button id="toggle">Toggle</button>
<button id="speed-up">Speed Up</button>
<button id="slow-down">Slow Down</button>
<button id="color">Color</button>
</div>
```

This function gives a random number, I used it for taking random colors.

```
function generateRandomNumber() {
    var min = 0.0,
        max = 1.0,
        highlightedNumber = (Math.random() * (max - min) + min).toFixed(3);
return highlightedNumber;
};
```

In init function, we got some onclick functions to control direction. And we used lots of generateRandomNumber() to take random colors.

```
window.onload = function init()
{
    document.getElementById("toggle").onclick = function() {
        direction = -direction;
    }
    document.getElementById("speed-up").onclick = function() {
        direction *= 2.0;
    }
    document.getElementById("slow-down").onclick = function() {
```

```
        direction *= 0.5;
    }
    document.getElementById("color").onclick = function() {
        colors = [vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumb
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
            vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber())
        ];
        var cBuffer = gl.createBuffer();
        gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
        gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

        var vColor = gl.getAttribLocation( program, "vColor" );
        gl.vertexAttribPointer( vColor, 3, gl.FLOAT, false, 0, 0 );
        gl.enableVertexAttribArray( vColor );
    }
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL␣isn't␣available" ); }

    //
    //  Configure WebGL
    //
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    //  Load shaders and initialize attribute buffers
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );

    var vertices = [
        vec2( 0.06, -0.48),
        vec2( 0.16, -0.17),
        vec2( 0.48, -0.17),
        vec2( 0.22, 0.02),
        vec2( 0.33, 0.33),
        vec2( 0.06, 0.14),
        vec2( -0.2, 0.33),
        vec2( -0.1, 0.02),
        vec2( -0.36, -0.17),
        vec2( -0.04, -0.17)

    ];

    colors = [vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber()),
        vec3( generateRandomNumber(), generateRandomNumber(), generateRandomNumber())
    ];
    // Load the data into the GPU

    var cBuffer = gl.createBuffer();
```

```
    gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(colors), gl.STATIC_DRAW );

    var vColor = gl.getAttribLocation( program, "vColor" );
    gl.vertexAttribPointer( vColor, 3, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vColor );

    var vBuffer = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

    var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );

    thetaLoc = gl.getUniformLocation( program, "theta" );

    render();
};
```

This render part is in loop. It updates theta at every frame.

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );

    theta += direction;
    gl.uniform1f( thetaLoc, theta );

    gl.drawArrays( gl.LINE_LOOP, 0, 10 );

    window.requestAnimFrame(render);
}
```

2 Part 2

In Part 2, I have an HTML file with lots of body parts. Long story short, there are input parts. I used little CSS file too.

```
<body>
            <div class="row">
        <div class="column">
            <canvas id="glCanvas" width="512" height="512">
            Oops ... your browser doesn't␣support␣the␣HTML5␣canvas␣element
␣␣␣␣␣␣␣␣␣␣␣␣</canvas>
␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row␣marg">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Scale␣Sun
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.1
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="sunScale"␣type="range"␣min="0.1"␣max="2"␣value="1"␣step
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Rotation␣Speed␣Sun
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.001
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="sunRotationSpeed"␣type="range"␣min="0.001"␣max="0.01"␣v
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Clockwise
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="sunRotationDirection"␣type="checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row␣marg">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Scale␣Earth
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.1
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="earthScale"␣type="range"␣min="0.1"␣max="2"␣value="1"␣st
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Rotation␣Speed␣Earth
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.001
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="earthRotationSpeed"␣type="range"␣min="0.001"␣max="0.01"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Clockwise
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="earthRotationDirection"␣type="checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row␣marg">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Scale␣Moon
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.5
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="moonScale"␣type="range"␣min="0.5"␣max="2"␣value="1.25"␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Rotation␣Speed␣Moon
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.001
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="moonRotationSpeed"␣type="range"␣min="0.001"␣max="0.01"␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Clockwise
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="moonRotationDirection"␣type="checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row␣marg">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Earth␣Orbit␣Rotation
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.005
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="earthOrbitRotation"␣type="range"␣min="0.005"␣max="0.05"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Clockwise
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="earthOrbitRotationDirection"␣type="checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
```

```
␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row␣marg">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Moon␣Orbit␣Rotation
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0.01
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="moonOrbitRotation"␣type="range"␣min="0.01"␣max="0.1"␣va
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="column">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Clockwise
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<div␣class="row"␣>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<input␣id="moonOrbitRotationDirection"␣type="checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣␣␣␣␣</div>
␣␣␣␣␣␣␣␣</div>
␣␣␣␣</div>
</body>
```

This file contains column, row and marg classes to use in HTML file.

```
.column {
    float: left;
    width: 33.33%;
}

.row:after {
    content: "";
    display: table;
    clear: both;
}

.marg {
      margin-left: 50px;
      margin-top: 10px;
      padding-right: 100px;
      width: 150%;
}
```

This function helps us to create a star that has its own radius and level. Level symbolizes how many point our star will have.

```
function drawStar(originX, originY, radiusOut, radiusIn, level){
    var positions = [];
    positions.push(originX);
    positions.push(originY);
    var angle = 2*Math.PI/level;
    for(var i = 0; i<level; i++){
        positions.push(originX + radiusOut*Math.cos(angle*i));
        positions.push(originY + radiusOut*Math.sin(angle*i));
        positions.push(originX + radiusIn*Math.cos(angle*(i+0.5)));
        positions.push(originY + radiusIn*Math.sin(angle*(i+0.5)));
    }
    positions.push(originX + radiusOut*Math.cos(angle*0));
    positions.push(originY + radiusOut*Math.sin(angle*0));
    console.log(positions);
    return positions;
}
```

This function calculates our variables and parameters at first of the initialization.

```
function calculate(){
    gl.clearColor(0.0,0.0,0.0,1.0);
```

```
    sun = initShaderProgram(gl, sunShader, fsSource);
    earth = initShaderProgram(gl, earthShader, fsSource);
    moon = initShaderProgram(gl, moonShader, fsSource);

    sunColorLoc = gl.getUniformLocation(sun, "color");
    sunPositionXLoc = gl.getUniformLocation(sun, "posX");
    sunPositionYLoc = gl.getUniformLocation(sun, "posY");
    sunScaleLoc = gl.getUniformLocation(sun, "uScale");
    sunRotationLoc = gl.getUniformLocation(sun, "uTheta");

    earthColorLoc = gl.getUniformLocation(earth, "color");
    earthPositionXLoc = gl.getUniformLocation(earth, "posX");
    earthPositionYLoc = gl.getUniformLocation(earth, "posY");
    earthScaleLoc = gl.getUniformLocation(earth, "uScale");
    earthRotationLoc = gl.getUniformLocation(earth, "uTheta");
    earthOrbitRotationLoc = gl.getUniformLocation(earth, "uThetaOrbit");

    moonColorLoc = gl.getUniformLocation(moon, "color");
    moonPositionXLoc = gl.getUniformLocation(moon, "posX");
    moonPositionYLoc = gl.getUniformLocation(moon, "posY");
    moonScaleLoc = gl.getUniformLocation(moon, "uScale");
    moonRotationLoc = gl.getUniformLocation(moon, "uTheta");
    moonOrbitRotationLoc = gl.getUniformLocation(moon, "uThetaOrbit");
    moonOrbitEarthLoc = gl.getUniformLocation(moon, "uEarthOrbit");

    positionsSun = drawStar(0, 0, 0.25, 0.1, level);
    positionsEarth = drawStar(0, 0, 0.25, 0.1, level);
    positionsMoon = drawStar(0, 0, 0.25, 0.1, level);



}
```

This init function has some assignations too.

```
window.onload = function init() {

    sunRotationSpeed = document.getElementById("sunRotationSpeed");
    earthRotationSpeed = document.getElementById("earthRotationSpeed");
    moonRotationSpeed = document.getElementById("moonRotationSpeed");
    moonScale = document.getElementById("moonScale");
    sunScale = document.getElementById("sunScale");
    earthScale = document.getElementById("earthScale");
    sunRotationDirection = document.getElementById("sunRotationDirection");
    earthRotationDirection = document.getElementById("earthRotationDirection");
    moonRotationDirection = document.getElementById("moonRotationDirection");
    moonOrbitRotationSpeed = document.getElementById("moonOrbitRotation");
    earthOrbitRotationSpeed = document.getElementById("earthOrbitRotation");
    moonOrbitRotationDirection = document.getElementById("moonOrbitRotationDirection");
    earthOrbitRotationDirection = document.getElementById("earthOrbitRotationDirection");

    const canvas = document.querySelector("#glCanvas");
    gl = canvas.getContext("webgl2");
    if(!gl){
        alert("Unable to initialize WebGL2. Your browser or machine may not support it.");
        return;
    }

    calculate();
    draw();
}
```

This function is in loop, basically it changes directions and speeds according to variables.

```
function draw(){
    gl.clear(gl.COLOR_BUFFER_BIT);
    const buffer = initBuffer(gl, positionsSun);

    gl.useProgram(sun);
    gl.uniform1f(sunPositionXLoc, sunPositionX);
    gl.uniform1f(sunPositionYLoc, sunPositionY);
    gl.uniform1f(sunRotationLoc, sunRotation);
    gl.uniform1f(sunScaleLoc, sunScale.value);
    gl.uniform4fv(sunColorLoc, sunColor);

    gl.enableVertexAttribArray(gl.getAttribLocation(sun, "aPosition"));

    gl.bindBuffer(gl.ARRAY_BUFFER, buffer.position);
    gl.vertexAttribPointer(gl.getAttribLocation(sun, "aPosition"), numOfComponents,gl.FLOA
    gl.drawArrays(gl.TRIANGLE_FAN, 0, positionsSun.length/numOfComponents);
    const buffer2 = initBuffer(gl, positionsEarth);

    gl.useProgram(earth);
    gl.uniform1f(earthPositionXLoc, earthPositionX);
    gl.uniform1f(earthPositionYLoc, earthPositionY);
    gl.uniform1f(earthScaleLoc, earthScale.value);
    gl.uniform1f(earthRotationLoc, earthRotation);
    gl.uniform4fv(earthColorLoc, earthColor);
    gl.uniform1f(earthOrbitRotationLoc, earthOrbitRotation);

    gl.enableVertexAttribArray(gl.getAttribLocation(earth, "aPosition"));

    gl.bindBuffer(gl.ARRAY_BUFFER, buffer2.position);

    gl.vertexAttribPointer(gl.getAttribLocation(earth, "aPosition"), numOfComponents,gl.FL
    gl.drawArrays(gl.TRIANGLE_FAN, 0, positionsEarth.length/numOfComponents);
    const buffer3 = initBuffer(gl, positionsMoon);

    gl.useProgram(moon);
    gl.uniform1f(moonPositionXLoc, moonPositionX);
    gl.uniform1f(moonPositionYLoc, moonPositionY);
    gl.uniform1f(moonScaleLoc, moonScale.value);
    gl.uniform1f(moonRotationLoc, moonRotation);
    gl.uniform4fv(moonColorLoc, moonColor);
    gl.uniform1f(moonOrbitRotationLoc, moonOrbitRotation);
    gl.uniform1f(moonOrbitEarthLoc, earthOrbitRotation);

    gl.enableVertexAttribArray(gl.getAttribLocation(moon, "aPosition"));

    gl.bindBuffer(gl.ARRAY_BUFFER, buffer3.position);

    gl.vertexAttribPointer(gl.getAttribLocation(moon, "aPosition"), numOfComponents,gl.FLO
    gl.drawArrays(gl.TRIANGLE_FAN, 0, positionsMoon.length/numOfComponents);

    if(sunRotationDirection.checked){
        sunRotation -= 100 * sunRotationSpeed.value;
    }
    else{
        sunRotation += 100 * sunRotationSpeed.value;
    }
    if(earthRotationDirection.checked){
        earthRotation -= 100 * earthRotationSpeed.value;
    }
    else{
        earthRotation += 100 * earthRotationSpeed.value;
    }
    if(moonRotationDirection.checked){
        moonRotation -= 100 * moonRotationSpeed.value;
    }
```

```
    else{
        moonRotation += 100 * moonRotationSpeed.value;
    }
    if(moonOrbitRotationDirection.checked){
        moonOrbitRotation -= 100 * moonOrbitRotationSpeed.value;
    }
    else{
        moonOrbitRotation += 100 * moonOrbitRotationSpeed.value;
    }
    if(earthOrbitRotationDirection.checked){
        earthOrbitRotation -= 100 * earthOrbitRotationSpeed.value;
    }
    else{
        earthOrbitRotation += 100 * earthOrbitRotationSpeed.value;
    }

    requestAnimationFrame(draw);
}
```

I have 3 different vector shaders to draw sun, earth, and moon.

```
const sunShader = `#version 300 es
    #define PI 3.1415926538
    in vec4 aPosition;

    uniform float uTheta;
    uniform float uScale;
    uniform float posX;
    uniform float posY;

    void main()
    {
        float angle = uTheta*PI/180.0;
        float c = cos(angle);
        float s = sin(angle);

        mat4 rz = mat4(c, s, 0.0, 0.0,
                       -s,  c, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       posX,  posY, 0.0, 1.0);

        mat4 scale = mat4(uScale, 0.0, 0.0, 0.0,
                       0.0,  uScale, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       0.0,  0.0, 0.0, 1.0);
        gl_Position = rz * scale *aPosition;

        gl_Position.z = -gl_Position.z;
    }
`;

const earthShader = `#version 300 es
    #define PI 3.1415926538
    in vec4 aPosition;

    uniform float uTheta;
    uniform float uScale;
    uniform float uThetaOrbit;
    uniform float posX;
    uniform float posY;

    void main()
    {
        float angle = uTheta*PI/180.0;
        float angleOrbit = uThetaOrbit*PI/180.0;
```

```
        float cOrbit = cos(angleOrbit);
        float sOrbit = sin(angleOrbit);
        float c = cos(angle);
        float s = sin(angle);

        mat4 rz = mat4(c, s, 0.0, 0.0,
                       -s,  c, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       cOrbit/2.0,  sOrbit/2.0, 0.0, 1.0);

        mat4 scale = mat4(uScale/1.4, 0.0, 0.0, 0.0,
                       0.0,  uScale/1.4, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       0.0,  0.0, 0.0, 1.0);
        gl_Position = rz * scale *aPosition;

        gl_Position.z = -gl_Position.z;
    }
`;

const moonShader = `#version 300 es
    #define PI 3.1415926538
    in vec4 aPosition;

    uniform float uTheta;
    uniform float uScale;
    uniform float uThetaOrbit;
    uniform float uEarthOrbit;
    uniform float posX;
    uniform float posY;

    void main()
    {
        float angle = uTheta*PI/180.0;
        float angleOrbit = uThetaOrbit*PI/180.0;
        float angleEarthOrbit = uEarthOrbit*PI/180.0;
        float cEarthOrbit = cos(angleEarthOrbit);
        float sEarthOrbit = sin(angleEarthOrbit);
        float cOrbit = cos(angleOrbit);
        float sOrbit = sin(angleOrbit);
        float c = cos(angle);
        float s = sin(angle);

        mat4 rz = mat4(c, s, 0.0, 0.0,
                       -s,  c, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       (cEarthOrbit+cOrbit/2.0)/2.0, (sEarthOrbit+sOrbit/2.0)/2.0, 0.0,

        mat4 scale = mat4(uScale/3.0, 0.0, 0.0, 0.0,
                       0.0,  uScale/3.0, 0.0, 0.0,
                       0.0,  0.0, 1.0, 0.0,
                       0.0,  0.0, 0.0, 1.0);
        gl_Position = rz * scale *aPosition;

        gl_Position.z = -gl_Position.z;

    }
`;
//vec4(0.5, 0.8, 0.9,1.0);
const fsSource =  `#version 300 es
    precision mediump float;
    uniform vec4 color;
    out vec4 fColor;
    void main(){
        fColor = vec4(color.x, color.y, color.z , color.w);
```

```
    }
';
```