A virtual Sun-Earth-Moon system

Evdzhan Nedzhib Mustafa

November 16, 2015

Abstract

Report on Advanced Computer Graphics (CS32310) assignment. Handed out during the 2015/2016 Academic Year, Semester 1.

1 How to run the program

To run open the solarsystem.html file in sensible browser, preferably Firefox.

2 Environment and Parameters

The system was implemented using the three.js (revision 72) library. Development was done exclusively on the Mozilla Firefox browser, version 42. The used OS was Linux Mint 17.2.

Parameters for the system can be found in the javascript/values.js file. The source file contains two sets of values. The first is within the REAL object, and as the name suggest, contains real scientific values of the used variables. The second set of values are in SCALED object, and contains scaled down values, that make it more feasible to actually represent the Sun-Earth-Moon system. One thing to note is that those values do not represent raw values, but rather ratios compared to one earth day and the earth radius. For example, the ratio between the Earth's radius and Moon's radius is kept in the scaled values. The values, both real and scaled, are also appended at the end of this document for completeness.

3 System structure and functionality

The developed system uses the three.js library to create a 3D model of Sun-Earth-Moon. The structure of the program consists of 3 objects of type Mesh, each of which represent the Sun, Earth and the Moon. The meshes are initially positioned at some starting points and are given specific appearance. Then the positions of the meshes are continually changed according to specific rules.

3.1 Meshes

Each Mesh was created using a Geometry and a Material. The obvious choice for Mesh Geometry was the THREE.SphereGeometry. Each Mesh's geometry is tilted by certain degree. This is done to achieve the tilt toward the ecliptic plane. A vector having the same tilt is created, so that when rendering occurs, the mesh can be rotated using that vector. Having the geometries in place, I do construct the materials. Both Earth and Moon use Phong material, while the Sun uses MeshBasicMaterial - so that it shines as if

it were illuminating the light. Using the built-in functions I covered the surface of each sphere with the textures downloaded from specific website (see Acknowledgements).

Having the meshes with proper textures and material, I do move them to appropriate starting positions, from where on the render function does animate the movement of each Mesh. The Sun Mesh is positioned at (3,0,3) on the scene and rotates on its tilted spin axis. The Earth mesh continually rotates around the Sun mesh, and also on its spin axis. Earth's spin axis is tilted by certain degree towards its orbital plane (which is also the ecliptic plane). The Moon mesh, does revolve around the Earth mesh, and on its spin axis. Initially, the self-axis spin speed, and orbital movement speed have the same value, which makes the same side of the Moon face the Earth at all times. If one is to change either of those two values, using the provided UI, one can cause the Moon to show different parts of itself to the Earth. The "Tidal Lock" effect can be observed when both of the spinners, controlling the Moon's rotation around self axis, and rotation around the Earth, have the equal values.

3.2 Moving the Sun, Earth and Moon

The hardest part for me was to correctly understand, what is tilted against what, what rotates around what, and also implement my understanding. As the render functions takes care of animating the object, that's where all the movement happens.

The Sun is easiest to animate. The Sun does rotate around itself, on a tilted axis. To animate that, I used the three.js function 3DObject#rotateOnAxis, which takes a vector and an angle. The said function does rotate the 3DObject around the given vector by the given amount of degrees of the passed angle. The tilt of the vector and the sun mesh, along with the angle to rotate, can be changed at run-time using the provided controls. Changing those values results in different spin angle, or different spin speed.

Having the Sun in place and rotating, I continued with Earth's movement animation. The Earth does revolve around the Sun, and also spins on a specific axis, tilted towards its orbital plane. The spin around a specific tilted axis, was implemented the same way the Sun's spin was - using the 3DObject#rotateOnAxis function. The Earth rotation's around the Sun was done by continually calculating new X and Z positions for the Earth. The basic math I used was to take the Sun's position, and get a new position that is rotated around specific angle degree, and then shifted in certain direction by the Earth's orbital radius. The degree by which I rotate goes from 1 to 360, and back to 1 again - to complete full circle. To calculate X and Z positions, I used the built-in JavaScript functions Math#cos and Math#sin.

Finally, I animate the Moon. The Moon proved to be hardest of to calculate the position of. For the Moon I had several details to consider. First, the Moon's spin axis is tilted by certain angle towards the ecliptic plane. Then, its the orbital plane is also tilted by certain angle to the ecliptic plane (or simply to Earth's orbital plane). Finally, the Moon's spin cycle(around it's spin axis) and rotation around the Earth, had to have the same degree, so that the Moon always presents the same side to the Earth. I had the Moon's geometry tilted by the specified angle towards the ecliptic plane, and as the Sun and Earth, I used the 3DObject#rotateOnAxis function. Now I had to make the Moon orbit around the Earth, with orbit inclined towards the ecliptic. Calculating the positions of the moon in that inclined plane was what I struggled most with in the entire project. I came up with calculation that continually takes Earth's latest position as a Vector. Then I create another Vector to represent the displacement of the Moon for each movement. The displacement is calculated by first rotating around the Y coordinate axis - to get the usual revolve movement. Then the result is rotated again towards the ecliptic plane, that is around the Z axis. If my understand is correct, the second rotation could instead be rotation around the X axis. (This area is still a bit unclear to me.) In the end, I update the Moon's position by adding the displacement vector

to the vector representing the Earth's latest position.

3.3 Light and Shadows

For a light I had several options in three js. The one I was familiar with from the practicals was PointLight, but as it does not cast shadows, I could not use it. The lights that are appropriate for casting shadows are DirectionalLight and SpotLight. I opted to use the more expensive DirectionalLight, as it lights both the Earth and Moon equally. The Light is positioned at the center of the Sun Mesh, and as the Earth rotates, the light continually changes direction to face the it.

For shadows, there are various parameters one has to set for the Light and Mesh objects. For example, in order for a Mesh to receive a shadow, it has to use either MeshPhongMaterial or MeshLambertMaterial. If MeshBasicMaterial is used, the object cannot receive shadow. Parameters that I had to consider for the DirectionalLight, were parameters defining how big the clipping space for shadows is. I had to carefully calculate how far one can place the Earth and Moon, and according to that I set the size of that clipping space. This basically meant that the farthest a shadow can be cast and received, is the sum of the distance between Sun and Earth, distance between Earth and Moon, and the radii of Earth and the Moon.

3.4 Controls

The user can move the camera using the mouse. A control panel is provided through which the user can change various variables, such as tilt axis, speed, velocity etc. A thing to note here is that in order for the Moon to always face the Earth with the same side, Moon's axial spin and orbital movement must have the same speed (which internally is as an angle). There are also several camera modes provided that can be activated using the 1,2,3,4,5 keyboard keys.

Controls panel inital, max/min and increment/decrement values are as follows:

Table 1: Controls values

What	**initial**	**min**	**max**	**increment/decrement step**
Earth-Sun distance	4	1.5	9	0.01
Earth-Moon distance	1	0.5	2	0.01
Sun size	1	0.1	2	0.1
Earth size	0.3	0.03	0.6	0.1
Moon size	0.083	0.0083	0.166	0.1
Sun self axis speed	1	0	20	0.1
Earth self axis speed	1	0	20	0.1
Moon self axis speed	1	0	20	0.1
Earth rotate Sun speed	0.1	0	2	0.1
Moon rotate Earth speed	1	0	20	1
Sun axial tilt	7.5	0	180	1
Earth axial tilt	23.44	0	180	1
Moon axial tilt	1.543	0	180	1
Moon orbit tilt	5.145	0	180	1

4 Acknowledgments

- The images containing the textures for the Earth, Moon, Sun and others are taken from planetpixelemporium.com.
- The three.js library along with the OrbitControls.js file obtained from threejs.org.

5 Self evaluation

I believe I have implemented all basic requirements. I have also implemented all the extensions apart from points g) and h) as I didn't have time to implement them. My overall self - assessment out of is 45/50. I do deduce five marks for not having implemented Elliptical orbits and Kepler's second law of planetary motion.

Table 2: Self Evaluation

Task	documented	implemented		Extension	documented	implemented
1	YES	YES		a)	YES	YES
2	YES	YES		b)	YES	YES
3	YES	YES		c)	YES	YES
4	YES	YES		d)	YES	YES
5	YES	YES		e)	YES	YES
6	YES	YES		f)	YES	YES
7	YES	YES		g)	NO	NO
				h)	NO	NO
				i)	YES	YES
				k)	YES	YES

6 Scientific parameters in values.js

Appending those values representing the used variables in my Sun-Earth-Moon System. Please note that the used notation is JavaSript Object Notation(JSON https://en.wikipedia.org/wiki/JSON).

```
const REAL = {
EARTH_RADIUS: 1.0,                          // earth radii
EARTH_ORBIT_RADIUS: 23500.0,                // earth radii
EARTH_SPIN_PERIOD: 0.9972,                  // earth days
EARTH_ORBIT_PERIOD: 365.26,                 // earth days
EARTH_AXIAL_TILT: 23.44,                    // degrees
EARTH_ORBIT_ECCENTRICITY: 0.0167,           // ?

MOON_RADIUS: 0.277,                         // earth radii
MOON_ORBIT_RADIUS: 60.3,                    // earth radii
MOON_ORBIT_PERIOD: 27.32,                   // earth days
MOON_SPIN_PERIOD: 27.32,                    // earth days
MOON_AXIAL_TILT: 1.543,                     // degrees
MOON_ORBIT_TILT_ECLIPTIC: 5.145,            // degrees
MOON_ORBIT_ECCENTRICITY: 0.0549,            // ?
```

```
    SUN_RADIUS: 109,                            // earth radii
    SUN_SPIN_PERIOD_EQUATOR: 25,                // earth days
    SUN_SPIN_PERIOD_POLES: 27,                  // earth days
    SUN_AXIAL_TILT: 7.5                        // degrees
};

var SCALED = {

    EARTH_RADIUS: 0.3,                          // earth radii
    EARTH_ORBIT_RADIUS: 4,                      // earth radii
    EARTH_SPIN_PERIOD: 1,                       // earth days
    EARTH_ORBIT_PERIOD: 0.1,                    // earth days
    EARTH_AXIAL_TILT: REAL.EARTH_AXIAL_TILT,    // degrees
    EARTH_ORBIT_ECCENTRICITY: REAL.EARTH_ORBIT_ECCENTRICITY, // ?
    EARTH_SCALE: 1,

    MOON_RADIUS: 0.3 * REAL.MOON_RADIUS,        // earth radii
    MOON_ORBIT_RADIUS: 1,                       // earth radii
    MOON_ORBIT_PERIOD: 1,                       // earth days
    MOON_SPIN_PERIOD: 1,                        // earth days
    MOON_AXIAL_TILT: REAL.MOON_AXIAL_TILT,      // degrees
    MOON_ORBIT_TILT_ECLIPTIC:REAL.MOON_ORBIT_TILT_ECLIPTIC,// degrees
    MOON_ORBIT_ECCENTRICITY: REAL.MOON_ORBIT_ECCENTRICITY, // ?
    MOON_SCALE: 1,

    SUN_RADIUS: 1,                               // earth radii
    SUN_SPIN_PERIOD: 1,                          // earth days
    SUN_AXIAL_TILT: REAL.SUN_AXIAL_TILT,         // degrees
    SUN_SCALE: 1,

    CLOUD_SIZE: 0.31,
    CLOUD_OPACITY: 0.3
};
```