CS 32310: 'Virtual Sun-Earth-Moon System'

Due on Thursday, November 19, 2015

James Euesden - jee22

Contents

Introduction

This assignment tasked me with using WebGL to create and display a simple animated model of the Sun-Earth-Moon system [1], requesting the basic features of display (three spheres with textures, orbits and rotations). There was also the option to implement additional functionality (lighting and shading, tilted axial and orbital rotations, etc). The model was not requested to be accurate in sizes, distances or times of rotations, although real world values were provided for reference.

Simulation

Parameters

In order to create the simple model, I used some arbitary values that made the system look somewhat accurate without truly being accurate. This was necessary in order for a viewer to see much of anything. With the changing of the scale of the Earth and Moon for display purposes and the assignment requesting a simple simulation, I took liberties to use values that looked good for the simulation while not impacting the main aspects of it (rotations, spinning on axes, orbital and axial tilts, etc). These parameters are:

```
1  // Large, so we can actually see something with our model.
2  var eccentricityModifier = 50;
3  // For Elliptical orbits/Kepler's 2nd Law. Number of steps for an orbit.
4  var orbitSteps = 1000;
5
6  var earthSize = 10; // from -> 1 * 10;
7  var earthDistanceFromSun = 156; // from -> 23500 / 150;
8  var earthOrbitRotationSpeed = 0.089; // from -> 365.26 / 4000;
9  var earthAxisRotationSpeed= 4.5; // from -> 9.972 / 2;
10 var earthAxialTilt = 23.44;
11 var earthOrbitEccentricity = 0.00167 * eccentricityModifier;
12
13 var moonSize = 6; // from -> 0.277 * 21;
14 var moonDistanceFromEarth = 40; // from -> 60.3 / 1.5
15 var moonAxisRotationSpeed = 1.2; // from -> 2.732 / 2.4;
16 // No Longer needed !
17 // var moonOrbitRotationSpeed = moonAxisRotationSpeed;
18 var moonOrbitEccentricity = 0.00549 * eccentricityModifier;
19 var moonAxialTilt = 1.593;
20 var moonOrbitalTilt = 5.145;
21
22 var sunSize = 20; // from -> 109 / 7;
23 var sunAxisRotationSpeed = 0.05; // from -> 0.25 / 4;
24
25 var starMapSize = 500;
```

As you can see, the Sun is much smaller than it should be, as are the distances between the Sun, Earth and Moon. I felt these numbers gave a good representation for this model, but should not be used if the model were intended to be an accurate representation of the system.

Structure

I have structured my JavaScript code such that there are files for each major component of the model (user interface, Sun-Earth-Moon in one file, Controller, etc) that handles the responsibilty of those components, for initiation and updating on each animation loop. Fror the Sun, Earth and Moon, each is held in their own object, which contains data of their Mesh, a constant reference to their homogeneous coordinates and their update methods.

Most of the values that parameters for controlling speed, size and the tilts of the planets are contained in a single global variable file, for easy access when I was building and testing my simulation. The setup and main rendering loop are in a single Controller file, and this calls to methods in the other files in the order that things should be initialised and updated. This made it easier to ensure the correct ordering of events.

The Earth and Moon mesh objects have also been added into one single Object3D, earthAndMoon, and then that object has been added to the scene, instead of adding the two individually. Originally, this helped me with orbital rotations, acting on both of the objects using one object, however it is not necessary with my final version of the application. I have kept them as part of the same Object3D though, as this increases the efficiency of the rendering, as they are rendered as one item. While not necessary for such a small simulation, if the application were expanded upon in the future to include many, maybe thousands, of additional planets, this style of structure would be much easier to maintain. It would also help group planets and their moons together, meaning they could be added and removed with their satellites easily.

Running the program

For demonstation, go to this address, hosted on my university filestore: `http://users.aber.ac.uk/jee22/cs323/virtual-system/` . For building and testing, I ran the application locally using xampp[2] . You should be able to see the application in full working order in both B59 and C56 using any of the provided browsers. Chrome and Firefox are recommended. No additional downloading is required. There is also a Credits page any textures and code used in the application here: `http://users.aber.ac.uk/jee22/cs323/virtual-system/credits.html`

Basic Features

The earth orbits the centre of the Sun

It does: Picture. It moves it's position on x and z using this update method and with this distance to the Sun. It moves around a set of pre-computed points. Previously moved with this function (sin/cos thing), but now moves by going through these points step by step (see here). More on this in additional features.

The moon orbits the centre of the Earth

It does: Picture of Earth in multiple places with orbit lines on. See my previous thing about the Earth orbiting around the Sun? Same thing applies, except now we also take into account the Earth's current location. See here!

Circular orbits

Orbits are complete, but Elliptical, see Additional point for information. When they were circular though, they were done like this: (Sin/Cos thing, moving on position x and y). If I were to do this in matrix transformations, it would look like this - (Make matrix homogeneous from object geometry, multiply that by shift to pivot point, rotate on axis for orbital step, shift back to position) OR (calculate next orbital step with x and y, then shift it to there with matrix). Seems legit.

The Sun, Earth and Moon shown as texture mapped spheres

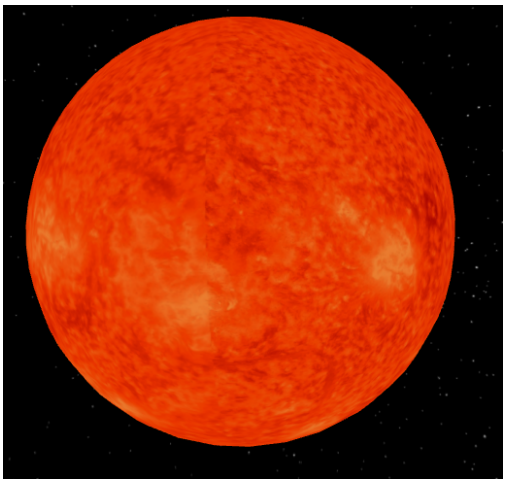
They are. Look at some pictures. The Earth and Moon even have bump maps. The Earth additionally has a specular map! Isn't that cool? textures are from here and credited on my credits page here.

The Sun, Earth and Moon each spinning on their own axes

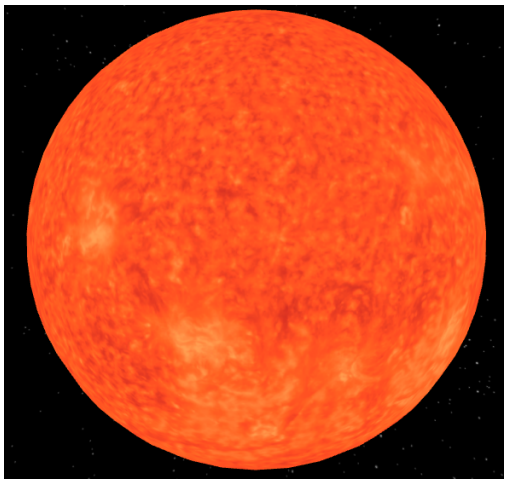
They do, here, look at this picture of the Axis Helper showing that they're spinning on their axes. they do this through matrix transformation of rotation, as can be seen here: We turn Object geometry vertices into homogeneous coordinates (unnecessary for rotation, but for the sake of allowing for future adaptations that may wish to use translate transformations that need the 'w' coordinate too, we do this), multiply by the rotation this step (which is multiplied by the speed of the objects rotation) and then applied back to the Object geometries coordinates. Rotation!

The sun shown as a self-illuminated sphere

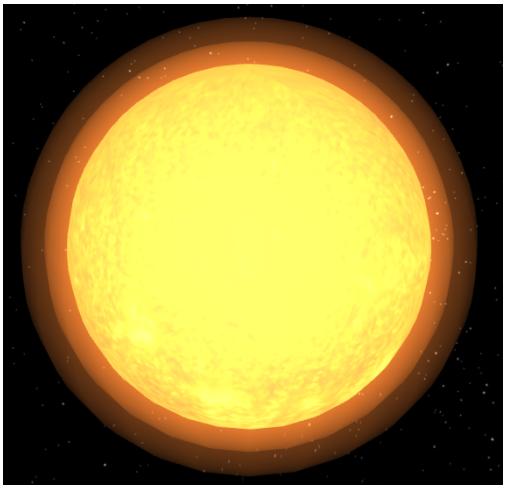
To present the Sun as a self-illuminated sphere, I first used an emissive map, which lights the texture applied to the Mesh, and then added some Ambient lighting, for the purpose of showing the Sun as if it had it's own light and also to help the viewer see the backs of the planets somewhat too. Additionally, I thought it looked good to have a glowing effect around the Sun. I achieved this using the technique by using examples from Lee Stemkoski[3]. I created two glow effects, one larger than the other, to give the impression of the light being more intense as the viewer looked directly at the Sun.



(a) The Sun with just an emissive map for self-illiminating



(b) The Sun with emissive map and ambient lighting.



(c) The Sun with emissive map, ambient lighting and glow effect.

Figure 1: The Sun with the three ways it is presented as self-illuminated.

Earth and Moon lit by a single point light source located at the centre of the Sun

All objects in the scene are lit slightly by a dark AmbientLight for the purpose of the animation. The light source from the Sun is a SpotLight, not a PointLight as requested by the Basic Features. The main reason for this is that Point Lights in three.js do not cast shadows.

In order to light the Earth and Moon with the Spot Light, I have the light starting at the centre of the Sun (0,0,0) and added as part of the SunMesh for rendering efficiency, and its target set as the position of the Earth, tracking it every animation frame to give the illusion of a single light source at the centre of the Sun.

```
sunLight = new THREE.SpotLight(0xfff0e6, 1, 0);
  sunLight.castShadow = true;
  sunLight.shadowBias = -0.0002;
  sunLight.shadowDarkness = 0.5;
  sunLight.shadowMapWidth = 1024;
  sunLight.shadowMapHeight = 1024;

  sunLight.shadowCameraNear = 20;
  sunLight.shadowCameraFar = 200;
  sunLight.shadowCameraFov = 300;
  // awesome for debugging - Shows the Shadow Camera lines
  // scene.add(new THREE.CameraHelper( sunLight.shadow.camera ));

  sunLight.target = earthMesh;
  sunLight.position.set(0, 0, 0);
  sunMesh.add(sunLight);
```

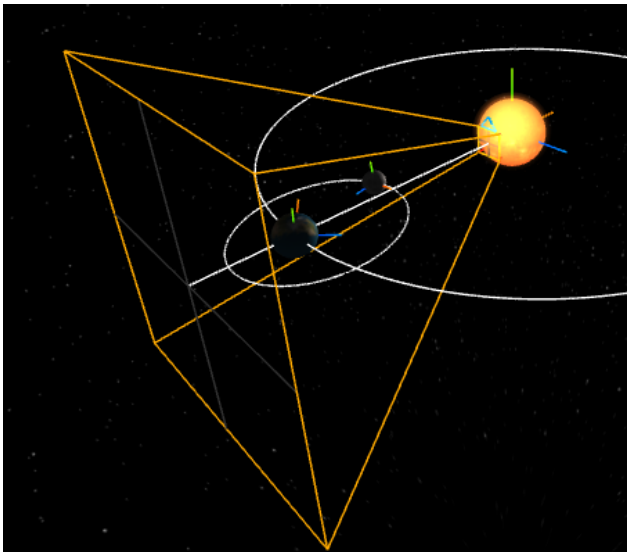


Figure 2: The Sun light Shadow Camera, showing where shadow casting will be computed.

You can see the ShadowCamera of the Spot Light inthe figure above, as displayed by the ShadowCameraHelper. This helper shows us what the catchement area of the cast shadows will be. Computing where shadows should fall is expensive, and so I've made the shadow camera area small, in order to reduce the cost of computation.

Additional Features

User Interface

Shiny UI, look at this picture. You can change speed, move the camera to look at the Sun, Earth or Moon, move the camera around the setting, reset the camera, pause the simulation (pauses the updating of values while still rendering the scene) and turn on or off helpers to visually see the orbital rotations and the axes of the objects. See image

Synchronous orbital and axial rotation of the Moon

Well, I did this at first by keeping the speed of the Moon's rotation and it's orbit the same, which worked perfectly. However, when I switched to pre-computing the elliptical orbit, the orbit and rotation no longer matched. I could compute the new rotation by rotating the Moon on the Y axis by the same amount of time steps and speed as the orbit, but I don't believe it would look too accurate, and with the orbital rotation also titled, it would mean calculating how the x and z axis should also be rotated. While possible to do this, for this simple visualisation, I chose to achieve this by using a tool in the three.js library '.lookAt()'. This method updates the rotation of an object to always face the target. In this case, it keeps the synchronus axial and orbital rotation of the moon so the same face is always pointing towards the Earth. Look at this cool set of images that shows the Moon always facing the Earth! Wow!

Lighting of the Earth and Moon in Phong shading

I didn't write my own Phong shader, but here, I used three.js to use Phone Mesh and have used the SpotLight to get Shadows and stuff. Is that enough? Idk what you want from me. Used a specular map to do this cool reflection on the water thing, while the rest absorbs the light better. Shiny.

Non-illuminated parts of the Earth and Moon to be shown in shadow

Due to the dark colour of the ambient light and the light source in the Sun being quite intense, this effect is achieved, see this picture here. Always darker on the side facing away from the Sun. Cool!

Constant tilt of the Earth's axis

Look, see this with Axis Helper. This is super simple to achieve. We just set the Earth's rotation on the z axis to be 23.4 once, when we create it, and it's like that and always in that direction. It spins upon this axis with the transformation matrix to have the correct axial rotation, but the tilt always stays in the one direction, to be a proper representation of the Earth and how it doesn't wobble on it's axis. See here, on each side of the Sun, the axis stays the same direction.

Constant tilt of the Moon's orbit

Here, look, it does this! See in this image, and in this image on the other side of the orbit, the rotation is a constant tilt. This is achieved through the pre-computed points then being multiplied by a rotation on the z axis (could also be the z axis, but this is what I chose) with a transformation matrix. I chose to use three.js applyMatrix4 with my own rotation matrix. I could have written my own Matrix transformation method, as I did for the rotation on the Y axis, however, since this is multiplying a Vector3 by a Matrix4, and I have already shown that I have mastered the command of matrix transformations, I chose to cut down on the amount of code I would need to write for this simple simulation.

Elliptical Orbits & Non-uniform orbital velocities: Kepler's 2nd Law of Planetary Motion

For both of these additonal points, they tie together. Through calculating one, the other appears. I used the formula provided in the assignment brief [1] in order to achieve this, and my code can be seen here: CODE INSERT LOL. This does this: Some cool stuff where we work out the theta for each angle of rotation, using the semi-major axis of the Earth and its distance from the Sun. This is the same for the Earth. You can see how the result looks here, using the Orbital Lines: During testing, I created just the orbital lines to see what the orbit path was like, and changing the number of time steps changed the speed of the orbit (as it was how many points it had to travel along to reach a full cycle) and changing the eccentricity changed how eccentric the ellipse of the orbit was. We can also see that at certain points, the Earth/Moon moves faster where the points are closer, simulating Kepler's 2nd Law of Planetary Motion.

Eclipse Shadows

Through the use of a SpotLight and its shadow camera, turning the ShadowMap of the renderer on, we get this effect: Here, look at some screenshots of it happening. Very easy to achieve. However, previously encountered issues with double shadows, with a WebGL issue that I couldn't fix in my own code. Struggled for a long time with it (others did too, see this link and this link), until an update to three.js fixed it. Relief at fixing it, also frustration at

it being something I thought was in my code but turned out not to be. Much time wasted. But it works now, and it's super pretty how it does it with the spot light. Here, you can see my testing with the ShadowCamera, how the shadows are presented within the camera, but not outside of it. COOL!

Other features of my choice

Mainly matrix transformations. Previously mentioned them, but I want to express that I wanted to fit them into my assignment. Before I pre-computed the orbits of the Earth and Moon, I was also working on their orbits with matrix transformations, but it became unnecessary once I changed how the orbit was calculated. I still stuck with using matrix transformations for axial rotations, however (where appropriate) and spent some time learning about how to do this. One problem I encountered when working with them was my lighting would not update during an Object's orbit around the light source. This was due to me updating the Geometries vertices but not the face and vertex normals. Once I started updating the Face and Vertex normals (like in three.js own method) see this code here: CODE, it worked as expected. I did test my transformations to check they gave the expected results too (4x4 matrix calculation for each matrix with result HERE PLZ). Many weird attempts with matrix transformations: Working directily with vertices, or working with the Matrix of the object, or working with the Mesh - Flying away Earth, Earth eventually shrinking at the middle (becoming a tube), irregular orbits, etc - Most overcome through looking at the matrix multiply method where I'd inccorectly done the multiplication ordering, or not properly applied the w coordinate on shift translations for the orbital rotations. Happy with the results though.

Sun glow: Just wanted it look cool. While not my own implementation, it was something I felt gave the simulation a little more interest. Code can be found here: SOME CITE. Also cool star map because it looks a little more 'real'. Idea for this came from here: CITE LINK.

Self Assessment

Here is my self-assessment form, including what has been documented, implemented, whether it can be run, if I referenced all my sources and what grade I would give myself out of 50 (and why that mark in text underneath this neat little table, kthnx).

References

[1] H. Holstein and Y. Liu, "A virtual Sun-Earth-Moon system", CS32310 Advanced Computer Graphics Assignment Semester 1 2015, October 14 2015

[2] XAMPP Installers and Downloads for Apache Friends, *apachefriends*, [online], https://www.apachefriends.org/index.html (Accessed 11/06/2015)

[3] Computational Contemplations: Using Shaders and Selective Glow Effects in Three.js, *stemkoski*, [online], http://stemkoski.blogspot.co.uk/2013/03/using-shaders-and-selective-glow.html (Accessed 11/06/2015)

[4] Acknowledgement for code colouring in this report using lstlisting LaTeX: javascript - language option supported in listings - TeX - LaTeX Stack Exchange, *TeX - LaTeX Stack Exchange*, [online], http://tex.stackexchange.com/questions/89574/language-option-supported-in-listings (Accessed: 11/06/2015)