THREE.js Solar System

Nicholas Adamou

November 2018

1 Defining and Rendering a Planet

I began by defining a set of global variables that would be used to calculate the various properties of each planet in the Solar System. To keep the document at minimal length, I am only showing two examples (e.g. Mercury and Saturn).

```
// planet/sun size = 100,000km : 50 units
//      - The Sun is ~696342km, r = sunSize = 348.15.
// AU = 150 mil km : 50 units
// planet orbital radius = 1AU : 1AU
//      - Mercury is 0.4AU from the Sun, thus
//        mercuryOrbitRadius = sunSize + (AU * 0.4)
// planet orbit speed = 1km : 0.02 units

let AU = 50;

...

let sunSize = 348.15;

let mercurySize = 1.2,
    mercuryOrbitRadius = sunSize + (AU * 0.4),
    mercuryOrbitAngle = getRandomNumber(0, 360),
    mercuryOrbitSpeed = 0.8,
    mercuryRotateSpeed = 0.05;

let saturnSize = 29.1,
    saturnOrbitRadius = sunSize + (AU * 9.5),
    saturnOrbitAngle = getRandomNumber(0, 360),
    saturnOrbitSpeed = 0.18,
    saturnRotateSpeed = 0.05;

...

function getRandomNumber(min, max) {
    return Math.random() * (max - min) + min;
}
```

In order to properly define a Mesh for each individual planet in the Solar System, I created a function called createLambertMesh().

```
function createLambertMesh(filePath = '', radius, size) {
    let geometry = new THREE.SphereGeometry(radius, size, size);
```

```
    let material = new THREE.MeshLambertMaterial({
        map: THREE.ImageUtils.loadTexture(filePath),
        shading: THREE.SmoothShading
    });
    return new THREE.Mesh(geometry, material);
}
```

In addition, for planets like Uranus and Saturn, they both are a construction of two individual 3D objects. One being the planet itself and the other being their ring system. Thus, I defined a function called createRingMesh() to achieve this.

```
function createRingMesh(filePath = '', transparent, start, end,
    size) {
    let geometry = new THREE.RingGeometry(start, end, size);
    let material = new THREE.MeshLambertMaterial({
        map: THREE.ImageUtils.loadTexture(filePath),
        shading: THREE.SmoothShading,
        side: THREE.DoubleSide,
        transparent
    });
    return new THREE.Mesh(geometry, material);
}
```

In my init() function, I define the planet's object within 3D space using some of its properties that I defined at the top of the file.

```
//Mercury
mercury = createLambertMesh('assets/images/mercury.jpg',
    mercurySize, 15);
scene.add(mercury);

//Saturn
saturn = createLambertMesh('assets/images/saturn.jpg', saturnSize,
    25);
scene.add(saturn);
//Saturn's rings
saturnRing = createRingMesh('assets/images/saturn-ring.jpg', false,
     saturnRingStart, saturnRingEnd, 30);
saturn.add(saturnRing);
```

To render the planet to the screen I defined a function called animate() which utilizes JavaScript's requestAnimationFrame() function to run the function in a loop. I then constructed a function called render() to put all of the necessary code for drawing each planet and their respective movement within the Solar System.

```
let renderer, scene, camera, radians;

...

let WIDTH = window.innerWidth,
    HEIGHT = window.innerHeight;

...

function init() {
```

```
    scene = new THREE.Scene();

    ...

    camera = new THREE.PerspectiveCamera(70, WIDTH / HEIGHT, 1,
        100000);

    ...

    renderer = new THREE.WebGLRenderer();

    ...
}

...

function animate() {
    requestAnimationFrame(animate);

    render();
}

function render() {

    ...

    //run Mercury's orbit around the Sun
    mercuryOrbitAngle -= mercuryOrbitSpeed;
    radians = mercuryOrbitAngle * Math.PI / 180;
    mercury.position.x = Math.cos(radians) * mercuryOrbitRadius;
    mercury.position.z = Math.sin(radians) * mercuryOrbitRadius;
    mercury.rotation.y += mercuryRotateSpeed;

    ...

    //run Saturn's orbit around the Sun
    saturnOrbitAngle -= saturnOrbitSpeed;
    radians = saturnOrbitAngle * Math.PI / 180;
    saturn.position.x = Math.cos(radians) * saturnOrbitRadius;
    saturn.position.z = Math.sin(radians) * saturnOrbitRadius;
    saturn.rotation.y += saturnRotateSpeed;

    ...

    renderer.render(scene, camera);
}
```

2 Constructing the Asteroid Belt

Much like the other planets, I defined a set of variables at the top of my document that would be used to describe the Asteroid Belt object.

```
let asteroidBelt, asteroid;

...
```

```
4
5  let asteroidOrbitStart = sunSize + (AU * 2.3),
6      asteroidOrbitEnd = sunSize + (AU * 3.3);
```

To properly construct each individual asteroid in the Asteroid Belt, I utilized THREE.js's Object3D class as a root object and added many individual objects to it.

```
1  ...
2
3  function getRandomNumber(min, max) {
4      return Math.random() * (max - min) + min;
5  }
6
7  function init() {
8      ...
9
10     //Asteroid Belt
11     asteroidBelt = new THREE.Object3D();
12     scene.add(asteroidBelt);
13
14     for(let x = 0; x < 3000; x++) {
15         let asteroidSize = getRandomNumber(0.005, 0.5),
16             asteroidOrbit = getRandomNumber(asteroidOrbitStart,
                   asteroidOrbitEnd),
17             yPos = getRandomNumber(-2, 2);
18
19         let geometry = new THREE.SphereGeometry(
20             asteroidSize,
21             getRandomNumber(4, 10),
22             getRandomNumber(4, 10));
23         let material = new THREE.MeshLambertMaterial({color:0
               xeeeeee});
24         asteroid = new THREE.Mesh(geometry, material);
25
26         asteroid.position.y = yPos;
27         radians = getRandomNumber(0, 360) * Math.PI / 180;
28         asteroid.position.x = Math.cos(radians) * asteroidOrbit;
29         asteroid.position.z = Math.sin(radians) * asteroidOrbit;
30
31         asteroidBelt.add(asteroid);
32     }
33
34     ...
35 }
```

Finally, to render the Asteroid Belt I did the following:

```
1  ...
2
3  function render() {
4      ...
5
6      asteroidBelt.rotation.y += 0.0001;
7
8      ...
9  }
```

3 Mouse Controls

In order to add mouse control with zoom I utilized a third-party package called TrackballControls. I implemented this using the following:

```
let mouse = new THREE.Vector2();

let renderer, controls, camera;

...

let WIDTH = window.innerWidth,
    HEIGHT = window.innerHeight;

...

function init() {
    ...

    camera = new THREE.PerspectiveCamera(70, WIDTH / HEIGHT, 1,
        100000);

    controls = new THREE.TrackballControls(camera);
    renderer = new THREE.WebGLRenderer();

    ...

    window.addEventListener('resize', onWindowResize, false);
    renderer.domElement.addEventListener('mousemove', onMouseMove);
}

function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerWidth;
    camera.updateProjectionMatrix();

    renderer.setSize(window.innerWidth, window.innerWidth);
}

function onMouseMove(e) {
    mouse.x = e.clientX;
    mouse.y = e.clientY;
}

function animate() {
    requestAnimationFrame(animate);

    render();
}

function render() {
    controls.update();

    ...
}
```