Solar System Simulation: Exploring Planetary Motion

written by Nirmal Raj Joshi | September 26, 2024

This realistic solar system animation captures the axial rotation, orbital inclination, and time periods of all the planets. It features two viewports: the left displays all the planets, while the right viewport follows the selected planet with a dedicated camera. The green plane represents the orbital plane of Earth and the Sun, helping to illustrate the relative positions of the other planets in relation to Earth's rotation plane.

The script is in Javascript so you can run it in any browser without installing any additional softwares.

Save the code as a HTML file and run in browser to enjoy the animation.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
      <meta  name="viewport"  content="width=device-width,
initial-scale=1.0">
    <title>3D Solar System with Two Viewports</title>
    <style>
        body { margin: 0; overflow: hidden; }
        #controls {
            position: absolute;
            top: 10px;
            left: 10px;
            z-index: 100;
            background-color: rgba(255, 255, 255, 0.8);
            padding: 10px;
            border-radius: 5px;
        }
        #controls label {
            display: block;
            margin-bottom: 5px;
        }
        #controls input, #controls select {
            width: 150px;
```

```
        }
        #container {
            display: flex;
            height: 100vh;
        }
        #mainViewport, #followViewport {
            width: 50%;
            height: 100%;
        }
    </style>
</head>
<body>
<div id="controls">
    <label for="cameraX">Camera X:</label>
     <input type="range" id="cameraX" min="-200" max="200"
value="0" step="1">
    <label for="cameraY">Camera Y:</label>
     <input type="range" id="cameraY" min="-200" max="200"
value="0" step="1">
    <label for="cameraZ">Camera Z:</label>
     <input type="range" id="cameraZ" min="10" max="300"
value="100" step="1">

      <label for="togglePlane">Toggle Earth's Orbital
Plane:</label>
    <input type="checkbox" id="togglePlane" checked>
    <label for="planetSelect">Select Planet:</label>
    <select id="planetSelect">
        <option value="0">Mercury</option>
        <option value="1">Venus</option>
        <option value="2">Earth</option>
        <option value="3">Mars</option>
        <option value="4">Jupiter</option>
        <option value="5">Saturn</option>
        <option value="6">Uranus</option>
        <option value="7">Neptune</option>
    </select>

    <label for="followCamera">Follow Selected Planet:</label>
    <input type="checkbox" id="followCamera">
</div>
```

```
<div id="container">
    <div id="mainViewport"></div>
    <div id="followViewport"></div>
</div>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
<script>
// Scene, Cameras, Renderers
const scene = new THREE.Scene();
const mainCamera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
const followCamera = new THREE.PerspectiveCamera(75, 1, 0.1, 1000);

const mainRenderer = new THREE.WebGLRenderer();
mainRenderer.setSize(window.innerWidth / 2, window.innerHeight);
document.getElementById('mainViewport').appendChild(mainRenderer.domElement);

const followRenderer = new THREE.WebGLRenderer();
followRenderer.setSize(window.innerWidth / 2, window.innerHeight);
document.getElementById('followViewport').appendChild(followRenderer.domElement);

// Light source
const light = new THREE.PointLight(0xffffff, 1.5, 1000);
light.position.set(0, 0, 0);
scene.add(light);

// Sun
const sunGeometry = new THREE.SphereGeometry(5, 32, 32);
const sunMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
const sun = new THREE.Mesh(sunGeometry, sunMaterial);
scene.add(sun);

// Planets data with rotation direction and inclination
```

```
const planets = [
    { name: 'Mercury', size: 0.5, distance: 10, speed: 0.04, tilt: 7, rotationSpeed: 0.01, rotationDirection: 1, inclination: 7.01 },
    { name: 'Venus', size: 1.2, distance: 15, speed: 0.03, tilt: 177.4, rotationSpeed: 0.002, rotationDirection: -1, inclination: 3.39 },
    { name: 'Earth', size: 1.3, distance: 20, speed: 0.02, tilt: 23.5, rotationSpeed: 0.01, rotationDirection: 1, inclination: 0 },
    { name: 'Mars', size: 0.7, distance: 25, speed: 0.017, tilt: 25, rotationSpeed: 0.01, rotationDirection: 1, inclination: 1.85 },
    { name: 'Jupiter', size: 3, distance: 35, speed: 0.013, tilt: 3, rotationSpeed: 0.02, rotationDirection: 1, inclination: 1.31 },
    { name: 'Saturn', size: 2.5, distance: 45, speed: 0.01, tilt: 26.7, rotationSpeed: 0.02, rotationDirection: 1, inclination: 2.49 },
    { name: 'Uranus', size: 2, distance: 55, speed: 0.007, tilt: 97.8, rotationSpeed: 0.015, rotationDirection: 1, inclination: 0.77 },
    { name: 'Neptune', size: 1.9, distance: 65, speed: 0.005, tilt: 28.3, rotationSpeed: 0.015, rotationDirection: 1, inclination: 1.77 }
];

const planetMeshes = [];

// Function to add latitude and longitude lines
function addLatLongLines(planetSize) {
     const latLongMaterial = new THREE.LineBasicMaterial({ color: 0x888888 });
    const latLongGeometry = new THREE.BufferGeometry();
    const positions = [];

    for (let lat = -80; lat <= 80; lat += 20) {
        const phi = (90 - lat) * (Math.PI / 180);
        const thetaStep = 10 * (Math.PI / 180);

        for (let theta = 0; theta <= 360; theta += 10) {
```

```
            const theta1 = theta * (Math.PI / 180);
            const theta2 = (theta + 10) * (Math.PI / 180);

               const x1 = planetSize * Math.sin(phi) *
Math.cos(theta1);
            const y1 = planetSize * Math.cos(phi);
               const z1 = planetSize * Math.sin(phi) *
Math.sin(theta1);

               const x2 = planetSize * Math.sin(phi) *
Math.cos(theta2);
            const y2 = planetSize * Math.cos(phi);
               const z2 = planetSize * Math.sin(phi) *
Math.sin(theta2);

            positions.push(x1, y1, z1, x2, y2, z2);
        }
    }

    for (let lon = 0; lon <= 360; lon += 10) {
        const theta = lon * (Math.PI / 180);
        const phiStep = 10 * (Math.PI / 180);

        for (let phi = 0; phi <= Math.PI; phi += phiStep) {
               const x1 = planetSize * Math.sin(phi) *
Math.cos(theta);
            const y1 = planetSize * Math.cos(phi);
               const z1 = planetSize * Math.sin(phi) *
Math.sin(theta);

             const x2 = planetSize * Math.sin(phi + phiStep) *
Math.cos(theta);
            const y2 = planetSize * Math.cos(phi + phiStep);
             const z2 = planetSize * Math.sin(phi + phiStep) *
Math.sin(theta);

            positions.push(x1, y1, z1, x2, y2, z2);
        }
    }

         latLongGeometry.setAttribute('position',  new
```

```
THREE.Float32BufferAttribute(positions, 3));
                    const    latLongLines    =    new
THREE.LineSegments(latLongGeometry, latLongMaterial);
    return latLongLines;
}

// Create planets with latitude and longitude lines
planets.forEach(planet => {
                   const    planetGeometry    =    new
THREE.SphereGeometry(planet.size, 32, 32);
     const planetMaterial = new THREE.MeshLambertMaterial({
color: Math.random() * 0xffffff });
      const planetMesh = new THREE.Mesh(planetGeometry,
planetMaterial);

    // Position the planet in its orbit, apply inclination
                           planetMesh.position.x      =
Math.cos(THREE.Math.degToRad(planet.inclination))     *
planet.distance;
                           planetMesh.position.y      =
Math.sin(THREE.Math.degToRad(planet.inclination))     *
planet.distance;
    planetMesh.rotation.z = THREE.Math.degToRad(planet.tilt);

    // Add latitude and longitude lines to the planet
    const latLongLines = addLatLongLines(planet.size);
    planetMesh.add(latLongLines);

    planetMeshes.push({
        mesh: planetMesh,
        speed: planet.speed,
        distance: planet.distance,
               rotationSpeed:  planet.rotationSpeed  *
planet.rotationDirection,
        angle: Math.random() * Math.PI * 2
    });

    scene.add(planetMesh);
});

// Earth's orbital plane
```

```
const planeGeometry = new THREE.PlaneGeometry(100, 100);
const planeMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00, side: THREE.DoubleSide, transparent: true, opacity: 0.3 });
const orbitalPlane = new THREE.Mesh(planeGeometry, planeMaterial);
orbitalPlane.rotation.x = Math.PI / 2;
scene.add(orbitalPlane);

// Set up cameras
mainCamera.position.set(0, 0, 100);
followCamera.position.set(0, 0, 100);
followCamera.lookAt(0, 0, 0);

// State variables
let followPlanet = null;

// Animation loop
function animate() {
    requestAnimationFrame(animate);

    // Rotate planets around the Sun
    planetMeshes.forEach(planet => {
        planet.angle += planet.speed;
          planet.mesh.position.x = Math.cos(planet.angle) * planet.distance;
          planet.mesh.position.z = Math.sin(planet.angle) * planet.distance;

        // Rotate planet on its axis
        planet.mesh.rotation.y += planet.rotationSpeed;
    });

    // Update camera positions
    mainRenderer.render(scene, mainCamera);

    if (followPlanet !== null) {
        const selectedPlanet = planetMeshes[followPlanet];
        const planetMesh = selectedPlanet.mesh;
        followCamera.position.set(
            planetMesh.position.x + selectedPlanet.distance *
```

```
Math.cos(planetMesh.rotation.y),
            planetMesh.position.y,
             planetMesh.position.z + selectedPlanet.distance *
Math.sin(planetMesh.rotation.y)
        );
        followCamera.lookAt(planetMesh.position);
        followRenderer.render(scene, followCamera);
    } else {
        followRenderer.render(scene, followCamera);
    }
}

animate();

// Handle window resize
window.addEventListener('resize', () => {
    const width = window.innerWidth / 2;
    const height = window.innerHeight;
    mainRenderer.setSize(width, height);
    followRenderer.setSize(width, height);
    mainCamera.aspect = width / height;
    mainCamera.updateProjectionMatrix();
    followCamera.aspect = width / height;
    followCamera.updateProjectionMatrix();
});

// Camera control with sliders
document.getElementById('cameraX').addEventListener('input',
function() {
    if (followPlanet === null) {
        mainCamera.position.x = parseFloat(this.value);
    }
});

document.getElementById('cameraY').addEventListener('input',
function() {
    if (followPlanet === null) {
        mainCamera.position.y = parseFloat(this.value);
    }
});
```

```
document.getElementById('cameraZ').addEventListener('input',
function() {
    if (followPlanet === null) {
        mainCamera.position.z = parseFloat(this.value);
    }
});

// Toggle Earth's orbital plane visibility
document.getElementById('togglePlane').addEventListener('chang
e', function() {
    orbitalPlane.visible = this.checked;
});

// Handle planet selection
document.getElementById('planetSelect').addEventListener('chan
ge', function() {
    followPlanet = parseInt(this.value);
});

// Handle camera follow toggle
document.getElementById('followCamera').addEventListener('chan
ge', function() {
    if (this.checked) {
        // Enable follow mode
                            const   selectedPlanet   =
document.getElementById('planetSelect').value;
        followPlanet = parseInt(selectedPlanet);
    } else {
        // Disable follow mode
        followPlanet = null;
    }
});
</script>
</body>
</html>
```