**MODELAREA SISTEMULUI SOLAR**

**WebGL & ThreeJS**

Chira Larisa Andreea 932

Pop Ruxandra Paula 936

Toma Răzvan 937

**1. Introducere**

**Scopul Proiectului**

Modelarea Sistemului Solar utilizand WebGL si ThreeJS.

**WebGL & ThreeJS**

ThreeJS este tehnologia principala pe care se bazeaza proiectul nostru, fiind o librarie care permite crearea si prezentarea unor grafici animate 3D in cadrul unui browser web. Acesta este construit pe baza a WebGL, o tehnologie relativ veche in comparativ cu tehnologia web moderna. Cu ajutorul ThreeJs putem sa configuram forme, sa incarcam texturi, sa configuram etc.

De asemenea, in cadrul proiectului nostru am utilizat o librarie numita “canvas-sketch” care ajuta la crearea de arta generativa in JavaScript si browser.

**2. Dispozitii generale**

**Constante si importuri**

Vom avea nevoie de Three.js, OrbitControls si canvas-sketch

Utilizam o constanta care sa retina setarile proiectului si altele pentru dimensiunile ferestrei

**Renderer**

Utilizam WebGLRenderer din Three.js pentru a initializa canvasul si pentru a seta fundalul

**Camera**

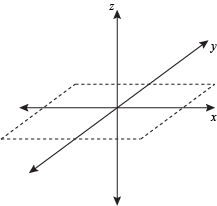
Pentru ca scena sa fie vizibila, trebuie sa setam camera prin care utilizatorii o vor privi. In cazul nostru utilizam o camera de perspectiva, prin care obiectele se vor modifica putin in functie de punctul de vedere

**Orbit Controls**

Permitem utilizatorului sa se deplaseze prin scena cu ajutorul actiunii “drag”

**Lumina**

Pentru inceput, setam sursa de lumina in centrul sistemului nostru solar (Soarele). Avand in vedere spatiul 3D in care lucram, punctul central va fi [0, 0, 0].



[0, 0, 0]

**3. Modelarea sistemului solar**

**Texturi, Materiale si Meshes**

Avand scena de baza creata, incepem sa adaugan obiecte in aceasta. Cream planetele cu ajutorul metodei SphereGeometry din Three.js

const geometry = new THREE.SphereGeometry(1, 32, 16);

Aceasta geometrie reprezinta o implementare a ceea ce forma ar trebui sa fie, si nu forma in sine. Pentru a creea forma, trebuie sa transmitem obiectul geometry ca paramentru la un nou “mesh”.

Un mesh este un obiect care provine din cel mai primitiv obiect din spatiul 3D – poligonul. Astfel, pentru a creea o sfera, transmitem geometria unui mesh, setam pozitia si scala la care trebuie si apoi o adaugam in scena initiala.

const earthMesh = new THREE.Mesh(geometry);

earthMesh.position.set(31, 0, 0);

earthMesh.scale.setScalar(1);

scene.add(earthMesh);

Pentru a face totul mult mai realist, vom utiliza TextureLoader din Three si vom incarca o imagine a proiectiei echirectangulare a Soarelui ca textura.

const earthTexture = loader.load("assets/earth.jpg");

Cu ajutorul acestei texturi vom innitializa un MeshStandardMaterial care ne permite sa mapam textura la un obiect 3D – in cazul nostru sfera.

const earthMaterial = new THREE.MeshStandardMaterial({ map: earthTexture });

Astfel, cand cream meshul, transmitem ca al doilea parametru acest material, alaturi de geometrie.

const earthMesh = new THREE.Mesh(geometry, earthMaterial);

**Rotatia**

Pentru a roti planetele in jurul originii [0, 0, 0] (Soarele), plasam fiecare planeta intr-un container numit Grup.

Astfel obitem urmatoarea sectiune de cod pentru a adauga Pamantul in scena:

const earthGroup = new THREE.Group();

const earthMesh = new THREE.Mesh(geometry, earthMaterial);

earthMesh.position.set(31, 0, 0);

earthMesh.scale.setScalar(1);

earthGroup.add(mercuryMesh)

scene.add(earthGroup);

Pentru a determina Pamantul sa se invarta:

return {

render({ time }) {

const ORP = time / 10;

// Earth's orbital rotation period

const ARP = ORP \* 365;

// Rotation around axis time

earthGroup.rotation.y = ORP;

earthMesh.rotation.y = ARP;

controls.update();

renderer.render(scene, camera);

}

}

Acest procedeu trebuie aplicat pentru fiecare planeta in parte, iar pentru asta putem utilliza o functie astfel incat sa evitam repetia de cod.

function createPlanet(scene, mesh, group, x, scale) {

mesh.position.set(x, 0, 0);

mesh.scale.setScalar(scale);

group.add(mesh);

scene.add(group);

}

**Lumina**

Pentru a ilumina planetele dinspre soare, utilizam PointLight in origine:

const light = new THREE.PointLight("white", 1.25);

light.position.set(0, 0, 0);

scene.add(light);

In schimb pentru a ilumina soarele, vom utiliza SpotLightHelper. Vom privi Soarele ca avand 6 laturi (cub) si vom adauga astfel 6 spotlighturi care impreuna ilumineaza intreg soarele – ca mentiunea ca acestea trebuie plasate inainte de prima planeta.

function createSpotlights(scene) {

var color = 0xFFFFFF;

var intensity = 5;

var distance = 25;

var angle = Math.PI/7;

new Array(6).fill('').forEach((item, i) => {

var spotlight =

new THREE.SpotLight(color, intensity, distance, angle);

var value = i % 2 === 0 ? 25 : -25;

spotlight.position.set(

i < 2 ? value : 0,

i >= 2 && i < 4 ? value : 0,

i >= 4 ? value : 0

);

scene.add( spotlight );

});

}

**Orbite**

Pentru a construi orbita fiecarei planete, apelam functia createOrbit. Aceasta functie ia ca parametrii scena, materialul care va reprezenta culoarea orbitei, raza in directia axei ox, raza in directia axei oy, si unghiul inclinatiei orbitei.

Astfel, functia EllipseCurve creeaza o curba 2d in forma unei elipse, dupa care apelam functia THREE.Line care pe baza geometriei si a materialului va creea obiectul orbita.

Pentru a putea vedea orbitele in reprezentarea noastra, adaugam rotatia specifica fiecareia si le adaugam in scena.

function createOrbit(scene, material, xRadius, yRadius, angle) {  
 const curve = new THREE.EllipseCurve(  
 0, 0, // ax, aY  
 xRadius, yRadius, // xRadius, yRadius  
 0, 2 \* Math.PI, // aStartAngle, aEndAngle  
 false, // aClockwise  
 0 // aRotation  
 );  
 const points = curve.getPoints( 200 );  
 const orbitGeometry = new THREE.BufferGeometry().setFromPoints( points );  
  
 const orbit = new THREE.Line( orbitGeometry, material );  
 orbit.rotateX(1.57 + angle);  
 scene.add(orbit);  
  
}

**Rotatia planetelor pe orbite**

Odata cu includerea planetelor in containerul Group, am adaugat si rotatia in jurul Soarelui. Un ultim pas ar fi cel de a roti planetele pe traiectoria orbitelor si nu paralel cu axa ox.

Pentru realizarea acestei cerinte, am adaugat pentru fiecare grup si o rotatie specifica axei ox, unde unghiul rotatiei corespunde cu cel al inclinatiei orbitei.

Un exemplu de astfel de rotatie ar fi, pentru planeta Mercur:

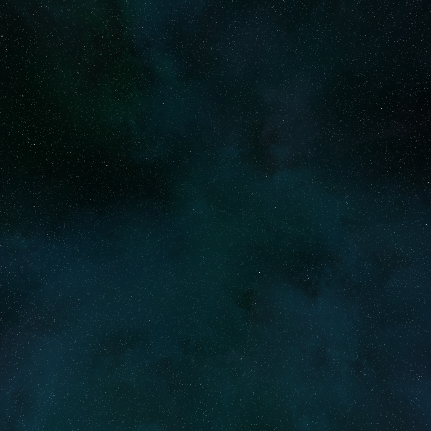
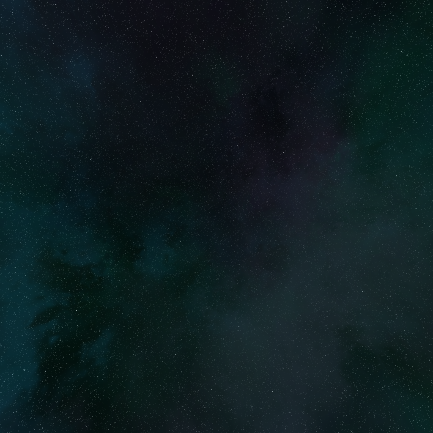
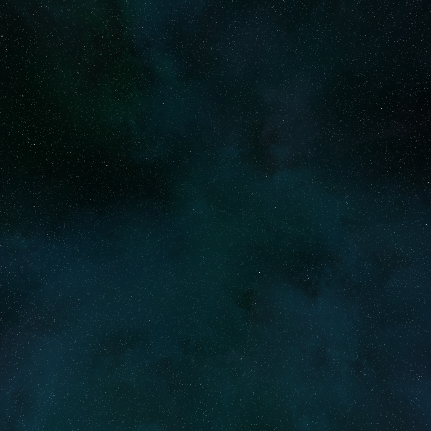
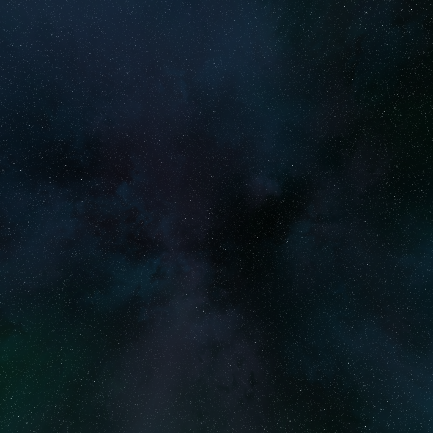
mercuryGroup.rotation.y = ORP \* 1.59; // rotate around the sun  
mercuryGroup.rotation.x = toRad(7.01); // follow the orbit  
mercuryMesh.rotation.y = ARP \* 58.65; // rotate planet

**Skybox**

Pentru a adauga mai mult realism scenei noastre 3D vrem sa folosim un background de tipul **skybox** in detrimentul unui background static cum ar fi o imagine. Un skybox este un cub cu o textura in care punem camera.

Un mod pentru a implementa un skybos este de a folosi un **Cubemap**. Un **Cubemap** este un tip special de textura care are 6 fete ce corespund fetelor cubului.

Exemplu de cubemap folosit:

****

Pentru a folosi aceste imagini vom utiliza CubeTextureLoader:

const loaderCube = new THREE.CubeTextureLoader();

const textuer = loaderCube.load([

` assets/skybox/four/front.png`,

` assets/skybox/four/back.png`,

` assets/skybox/four/top.png`,

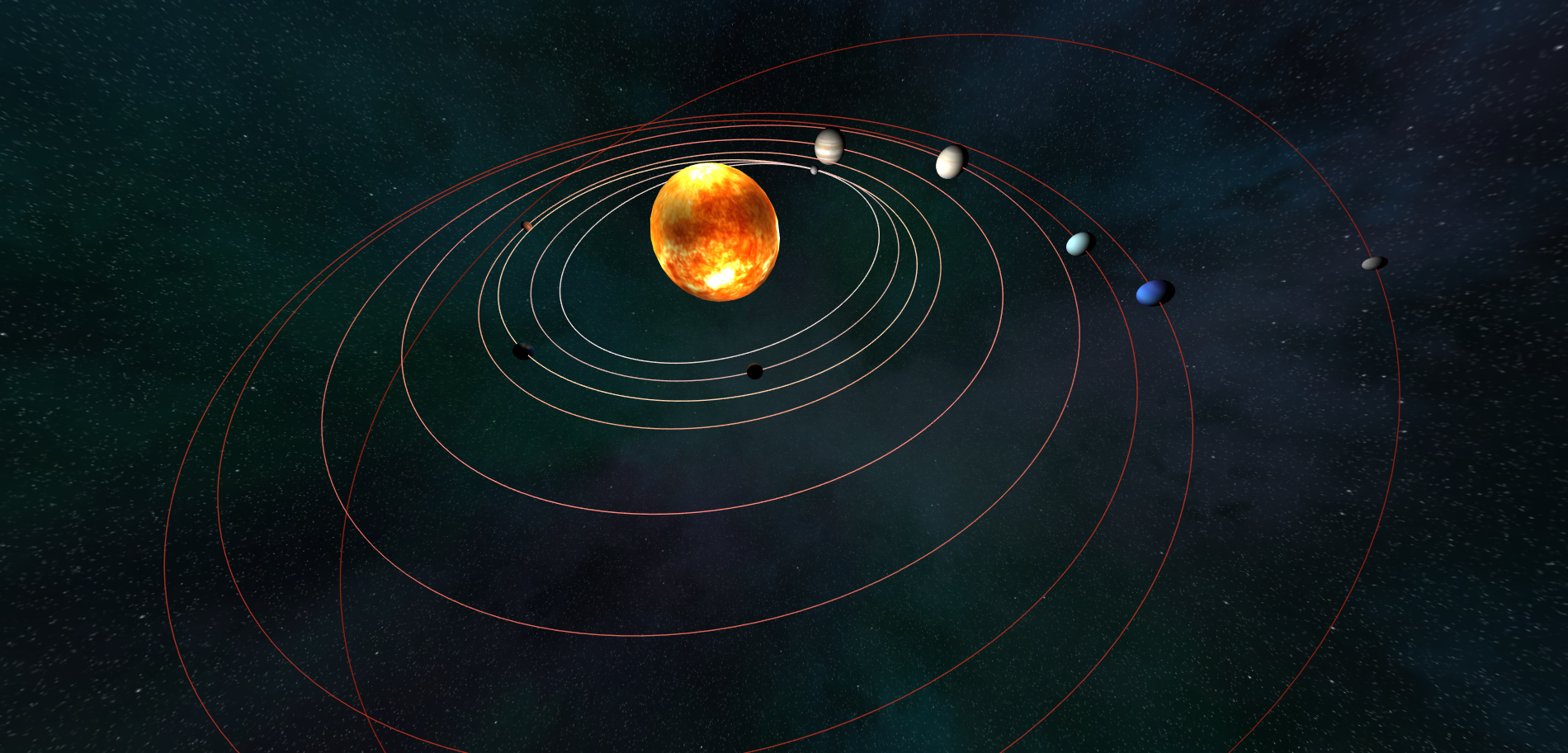
` assets/skybox/four/bottom.png`,

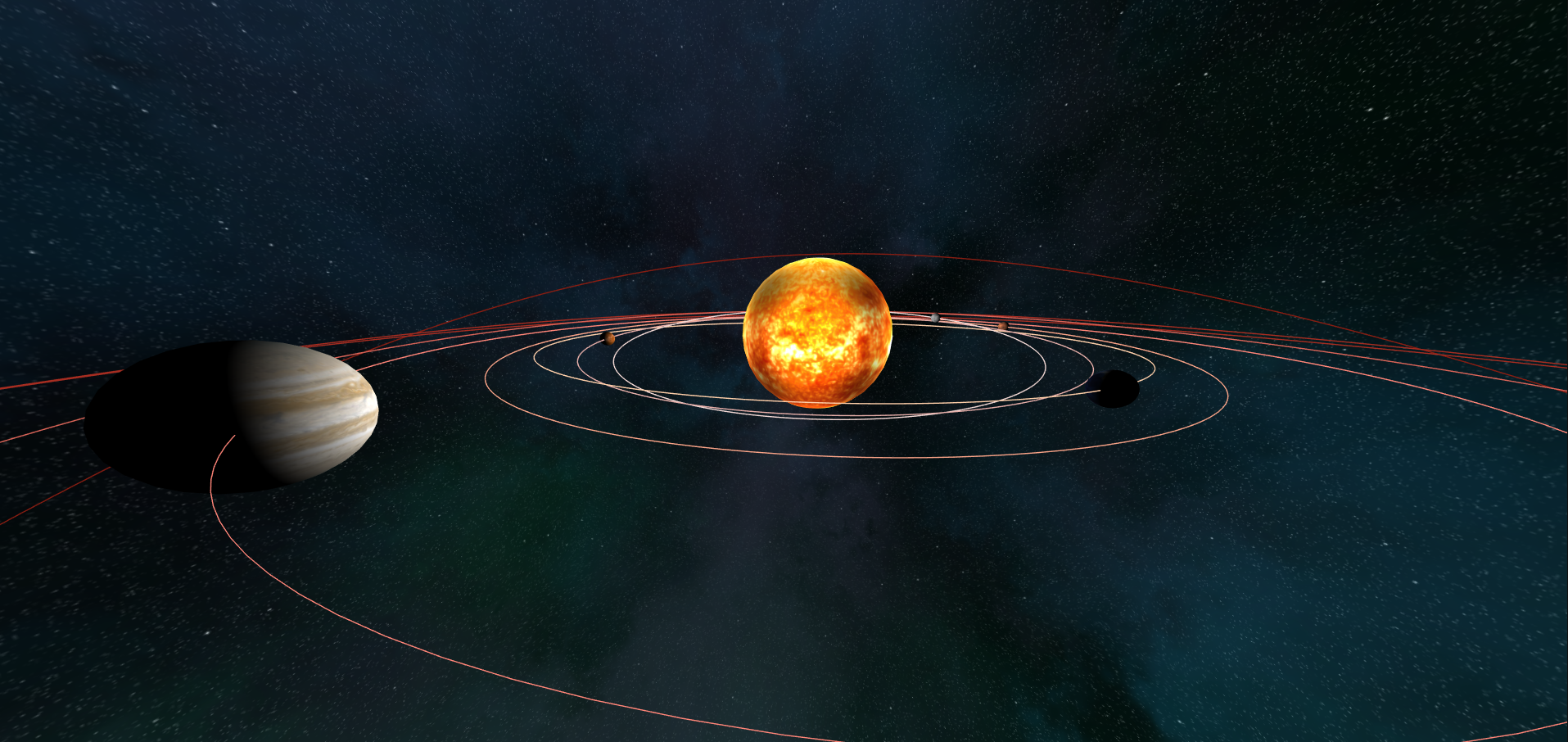
` assets/skybox/four/left.png`,

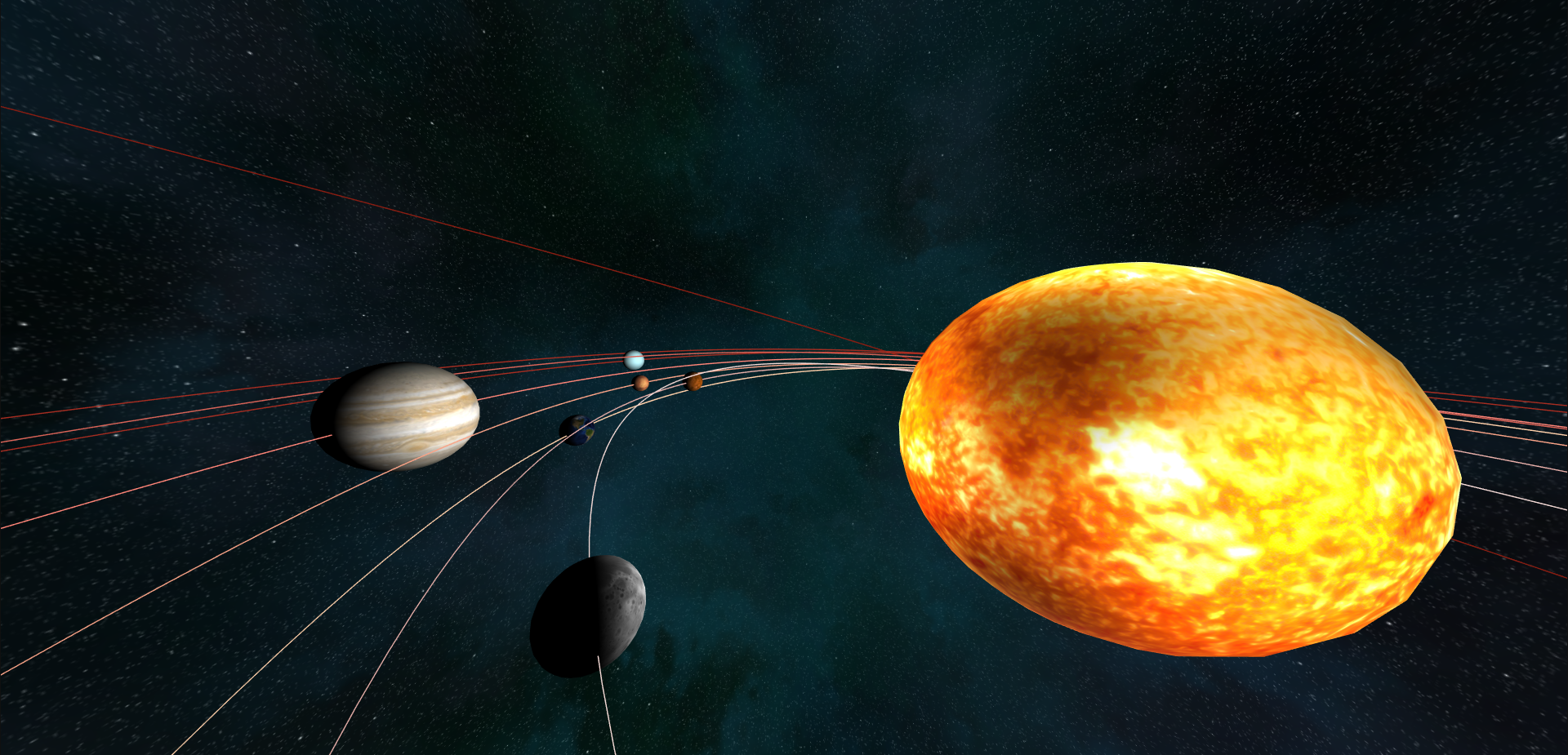
` assets/skybox/four/right.png`,

]);

scene.background = texture;



****

****