Computer Graphics Laboratory
Final Project Documentation

Group Members:
Alvin Justine - 2602127702
Beatrix Marlene Magetanapuang - 2602152370
Gaelen Adelard Jeffry - 2602147036
Rifki Averil Aldean - 2602114246

BE01 - LAB

Computer Graphics
2023/2024 Odd Semester

Solar SysteMF

This file is a documentation for the Computer Graphics final project where students are tasked to create "Solar SysteMF" project within a group of 1 - 4 people. As a Three.js enthusiast, our team creates an interactive, 3D solar system. We decided to leverage Three.js to craft the **sun. eight planets (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus,** and **Neptune**), and a **spaceship** that would allow users to explore the simulated solar system.

1. **Project Structure**

   The following is the **HTML** file of our project which fulfils the project requirement.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Solar SysteMF</title>
</head>
<style>
    body{
        margin: 0;
        padding: 0;
    }
</style>
<body>
    <div id="hovered-object-name" style="position: absolute; top: 10px; left: 10px; color: white; font-size: 18px;"></div>
    <script src="./script2.js" type="module"></script>
</body>
</html>
```

   Figure 1. Required HTML code.

2. **Scene**

   The scene is required to be displayed in **full screen**, so we set the margin and padding to 0.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Solar SysteMF</title>
</head>
<style>
    body{
        margin: 0;
        padding: 0;
    }
</style>
<body>
    <div id="hovered-object-name" style="position: absolute; top: 10px; left: 10px; color: white; font-size: 18px;"></div>
    <script src="./script2.js" type="module"></script>
</body>
</html>
```

Figure 2. Full Screen Style

The scene can be **dynamically resized** to fit the window.

```javascript
window.onresize = () => {
        let w = window.innerWidth;
        let h = window.innerHeight;

        camera.aspect = w / h;
        camera.updateProjectionMatrix();

        renderer.setSize(w, h);
};
```

Figure 3. Window Dynamic Resize Ability code

The scene also has **shadow map enabled** using **PCFShadowMap (default)** as the shadow map type and **anti-aliasing** turned on.

```javascript
    renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setSize(w, h);
    document.body.appendChild(renderer.domElement);

    renderer.shadowMap.enabled = true;
    renderer.shadowMap.type = THREE.PCFShadowMap;
```

Figure 4. PCFShadowMap and anti-aliasing

3. Camera

a) Freely Rotating Camera

It aims to look at the entire scene from the **top**/above perspective.

```
const init = () => {
    scene = new THREE.Scene();

    let fov = 75;
    let w = window.innerWidth;
    let h = window.innerHeight;
    let aspect = w / h;
    let near = 0.1;
    let far = 10000;

    camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
    camera.position.set(640, 480, 240);
    camera.lookAt(640, 320, 0);

    renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setSize(w, h);
    document.body.appendChild(renderer.domElement);

    renderer.shadowMap.enabled = true;
    renderer.shadowMap.type = THREE.PCFShadowMap;

    controls = new OrbitControls(camera, renderer.domElement);
    controls.target.set(640, 320, 0);
    controls.update();
```

Figure 5. Freely Rotating Camera Code

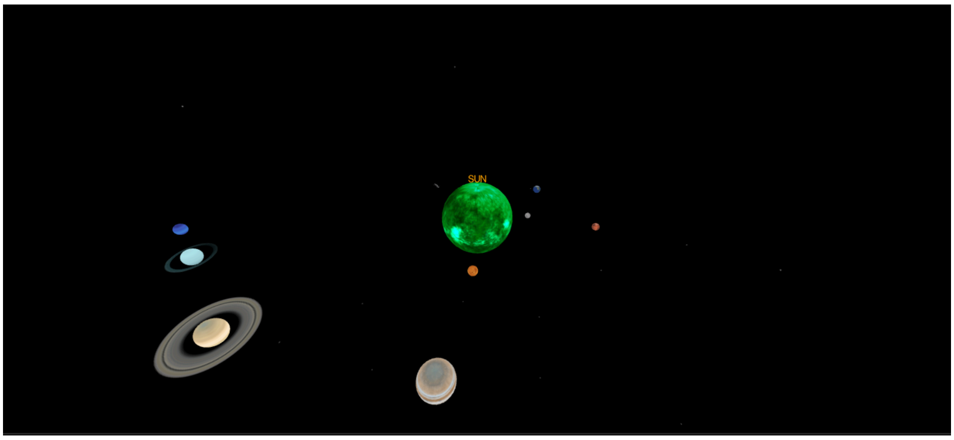


Figure 6. Freely Rotating Camera

b) Third Person Camera

It allows users to traverse the map through the **spaceship** as it centers. Users can **switch** between Freely Rotating Camera by clicking "**Escape**".

```javascript
let isThirdPerson = true;

const updateCamera = () => {
  if (!model) return;

  if (isThirdPerson) {
    const distance = -16;
    const height = 16;

    var offset = new THREE.Vector3(0, height, distance);
    offset.applyQuaternion(model.quaternion);

    camera.position.copy(model.position).add(offset);

    camera.lookAt(model.position);

    controls.target.copy(model.position);
    controls.update();
  } else {
    controls.target.set(640, 320, 0);
    controls.update();
    animate();
    animateText();
  }
};


window.addEventListener('keydown', (event) => {
  if (event.key === 'Escape') {
    isThirdPerson = !isThirdPerson;
  }
});
```

Figure 7. Third-Person Camera Code

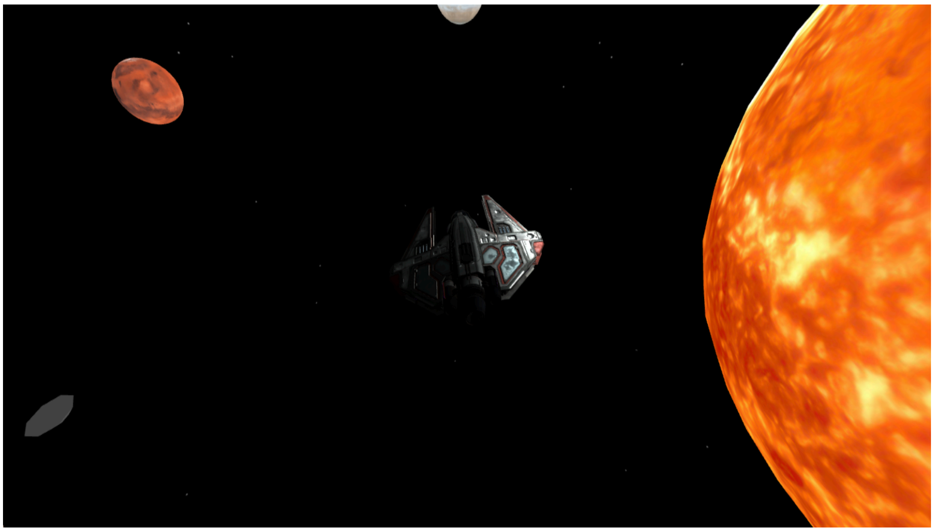
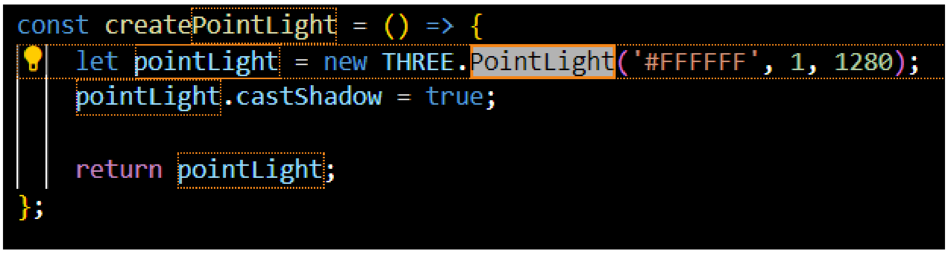


Figure 8. Third-Person Camera

4. Light

a) Point Light

It emits light in **all directions**



```
const createPointLight = () => {
    let pointLight = new THREE.PointLight('#FFFFFF', 1, 1280);
    pointLight.castShadow = true;

    return pointLight;
};
```

Figure 9. Point Light Code

b) Spotlight

Emits a **cone-shaped** beam of light in a specified direction. It will focus on Spaceship's Vector3 (x, y+6, z) and change **dynamically** since the Spaceship's starting position is at. We name the lighting that follows the spaceship "spotLights" while we make another spotlight to illuminate some objects such as satellite and planet rings,

```
const createSpotlights = () => {
  spotlights = new THREE.SpotLight(0xFFFFFF, 8);
  spotlights.castShadow = false;
  spotlights.distance = 8;

  scene.add(spotlights);

  spotlights.target = new THREE.Object3D();
  scene.add(spotlights.target);
};
```

Figure 10. createSpotLights code

```
    let spotLightS = createSpotLightS();
    spotLightS.position.set(750, 326, 0)
    scene.add(spotLightS)

```

Figure 11. Spotlight Code

Figure 12. Spotlight on the Spaceships

5. Objects

a) Sun

The Sun is created out of a **sphere without shadow**.

```
const createSphereNoShadow = (r, texture) => {
    let geo = new THREE.SphereGeometry(r);
    let mat = new THREE.MeshBasicMaterial({
        color: '#FFFFFF',
        map: texture
    });
    let mesh = new THREE.Mesh(geo, mat);
    mesh.castShadow = false;
    mesh.receiveShadow = false;

    return mesh;
};
```

Figure 13. createSphereNoShadow function

```
    sun = createSphereNoShadow(40, sunTexture);
    sun.position.set(640, 320, 0);
    sun.name = "sun";

```

Figure 14. Create Sun

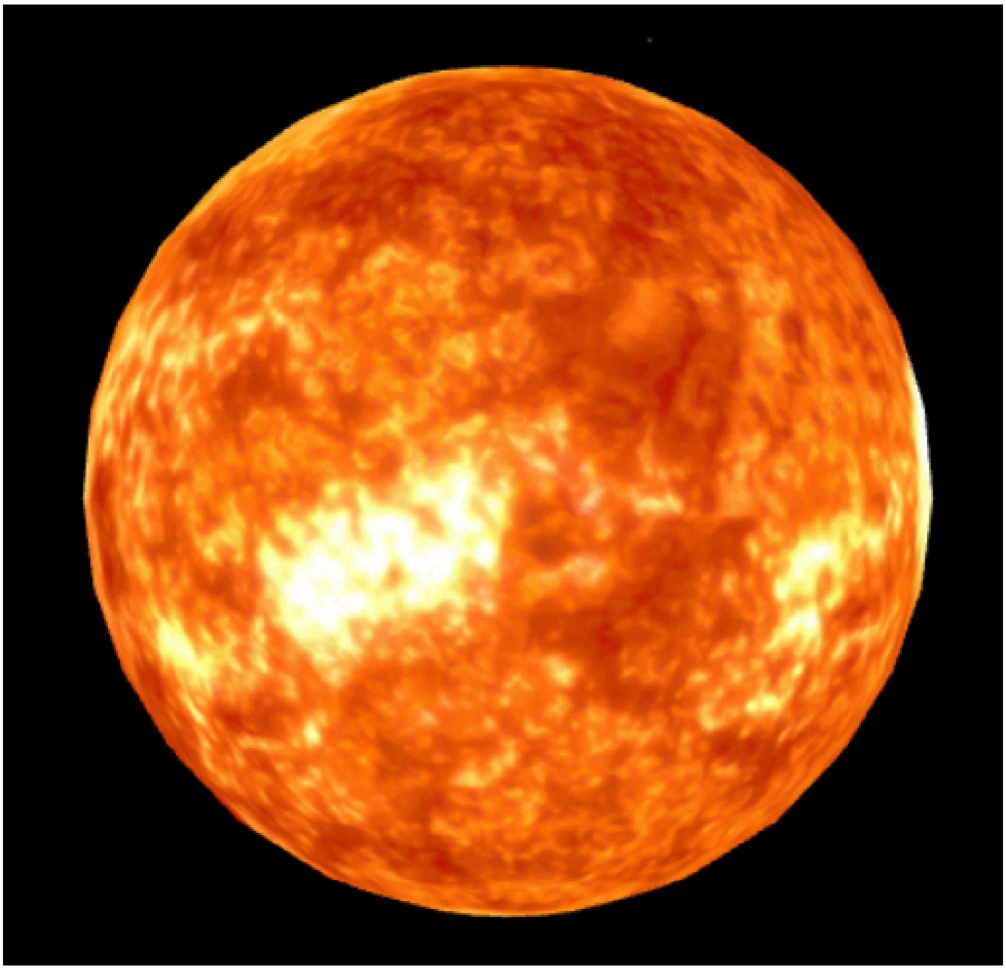


Figure 15. The Sun

b) Planets

The planets are all created by the **createSphere** function

```
const createSphere = (r, texture) => {
    let geo = new THREE.SphereGeometry(r);
    let mat = new THREE.MeshBasicMaterial({
        color: '#FFFFFF',
        map: texture
    });
    let mesh = new THREE.Mesh(geo, mat);
    mesh.castShadow = true;
    mesh.receiveShadow = true;

    return mesh;
```

Figure 16. createSphere function

```
mercury = createSphere(3.2, mercuryTexture);
venus = createSphere(4.8, venusTexture);
earth = createSphere(4.8, earthTexture);
mars = createSphere(4, marsTexture);
jupiter = createSphere(13, jupiterTexture);
saturn = createSphere(10, saturnTexture);
uranus = createSphere(8, uranusTexture);
neptune = createSphere(6, neptuneTexture);
```

Figure 17. Planets creation

Figure 18. Planets

c) Spaceship

The spaceship will allow users to **traverse** the entire screen freely.

```javascript
const moveForward = () => {
  const direction = new THREE.Vector3(0, 0, 1);
  direction.applyQuaternion(model.quaternion);
  model.position.add(direction.multiplyScalar(speed));
};

const rotateLeft = () =>{
  model.rotation.y += 0.03;
}

const rotateRight = () =>{
  model.rotation.y -= 0.03;
}

const rotateUp = () =>{
  model.rotation.z -= -0.01;
}

const rotateDown = () =>{
  model.rotation.z += -0.01;
}
```

Figure 19. Spaceship control

```javascript
const load3DModel = (url) => {
  return new Promise((resolve, reject) => {
    let loader = new GLTFLoader();
    loader.load(
      url,
      (gltf) => resolve(gltf.scene),
      undefined,
      (error) => reject(error)
    );
  });
};
```

Figure 20. Load 3D-Model

```javascript
try {
  model = await load3DModel("./assets/model/spaceship/scene.gltf");
  model.position.set(750, 320, 0);
  scene.add(model);
} catch (error) {
  console.error("Error loading model, ", error);
}
```

Figure 21. Loading the Spaceship's texture

Figure 22. The spaceship

d) Planet's Ring

The planet's rings are made from the **createRing** function

```
const createRing = (innerRadius, outerRadius, thetaSegments, color, texturePath) => {
    const geometry = new THREE.RingGeometry(innerRadius, outerRadius, thetaSegments);
    const textureLoader = new THREE.TextureLoader();
    const texture = textureLoader.load(texturePath);
    const material = new THREE.MeshStandardMaterial({
        map: texture,
        transparent: true,
        side: THREE.DoubleSide
    });
    const ring = new THREE.Mesh(geometry, material);
    ring.receiveShadow = true;
    ring.castShadow = false;

    return ring;
};
```

Figure 23. createRing function

```
uranusRing = createRing(16, 20, 64, '#FFFFFF', './assets/textures/uranus_ring.png');
saturnRing = createRing(16, 32, 64, '#FFFFFF', './assets/textures/saturn_ring.png');
uranusRing.rotation.x = Math.PI / 2;
saturnRing.rotation.x = Math.PI / 2;
uranus.add(uranusRing);
saturn.add(saturnRing);
```

Figure 24. Rings creation

- **Saturn's Ring**

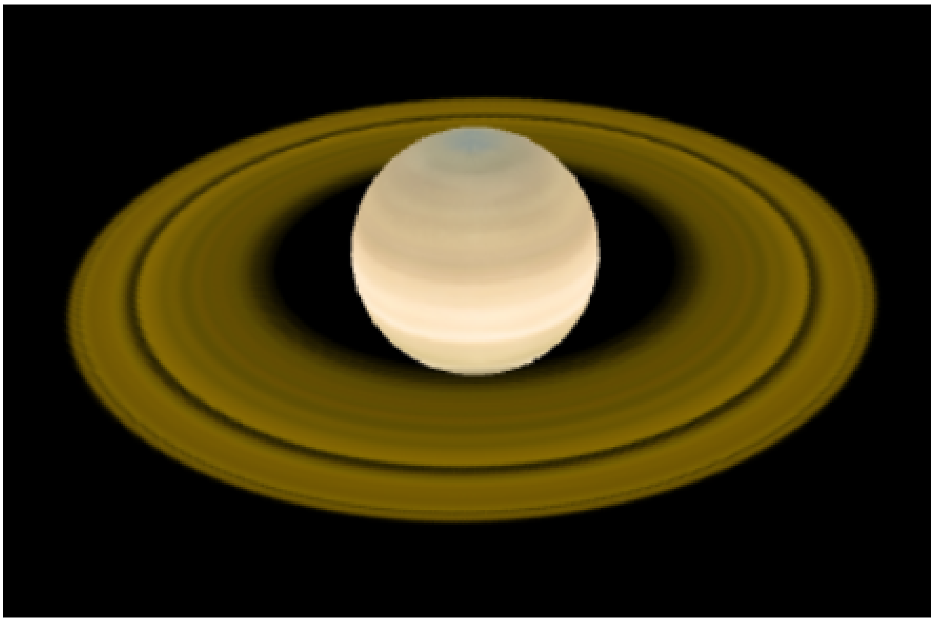


Figure 25. Saturn's Ring

- **Uranus' Ring**

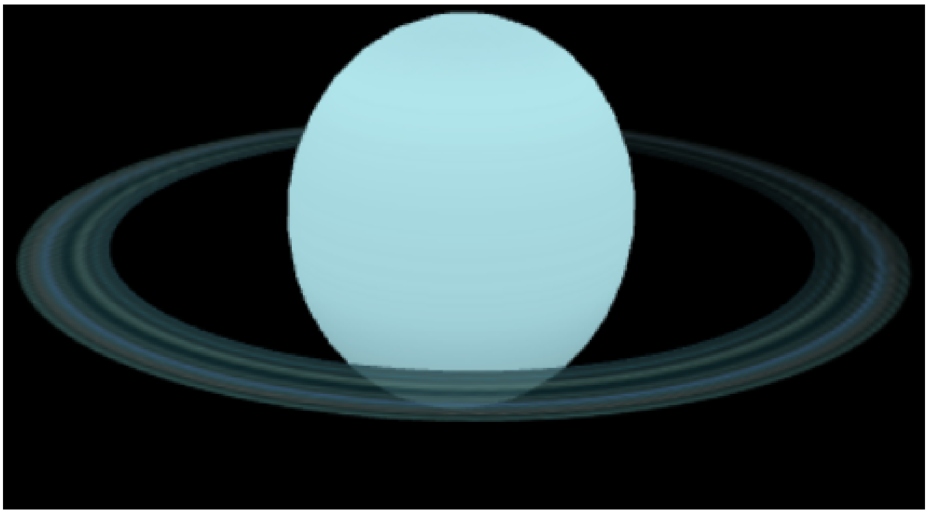


Figure 26. Uranus' Ring

e) Satelite

The satellite is made by the **createCylinder** function

```
const createCylinder = (radiusTop, radiusBot, height, radialSeg, color, metalness, roughness) =>{
    let geometry = new THREE.CylinderGeometry(radiusTop, radiusBot, height, radialSeg);
    let material = new THREE.MeshStandardMaterial({color: color, metalness: metalness, roughness: roughness})
    let mesh = new THREE.Mesh(geometry, material);
    mesh.receiveShadow = true;
    return mesh;
}
```

Figure 27. createCylinder function

```
        Satellite = createCylinder(1, 0.5, 0.4, 8, '#CCCCCC', 0.5, 0.5)
```

Figure 28. Satellite creation

```
                        Satellite.position.x = planet.position.x + 8;
                        Satellite.position.z = planet.position.z;
                        Satellite.position.y = planet.position.y;
                    }
            });
};
```

Figure 29. Satellite position adjustment

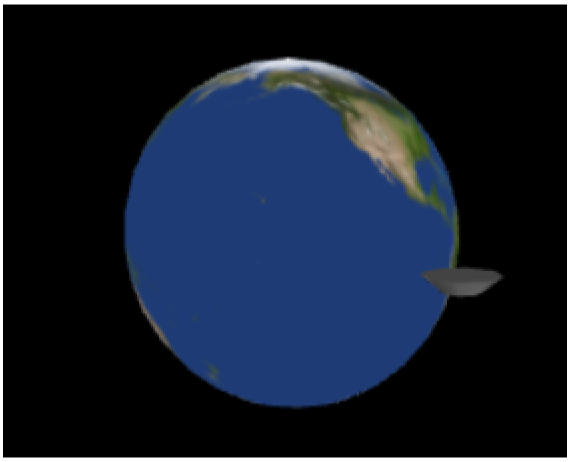


Figure 30. Earth & Satellite

f) Texts

On the **Free-Transform camera**, if a planet is hovered, it will **change color randomly** and display its name **text**. Then if the planet **clicks**, its **rotation** will **speed up** and return to the normal speed after a few seconds.

First, we declare an **array** to store the text variables. We use the **createText** and **createSunText** functions as media to **create the text**. The planet is given the rotation and orbit by the animatePlanets function. The **text** will be brought to the scene when an object is hovered and the planet's **rotation** will speed up when an object is clicked through the animateText function. Furthermore, the animate function let an object **change color randomly** whenever it is hovered.

```
const txt = [
    {
      text: "MERCURY",
      size: 5,
      height: 1,
      pos: new THREE.Vector3(58, 350, 0),
      name: "mercury",
    },
    {
      text: "VENUS",
      size: 3.5,
      height: 1,
      pos: new THREE.Vector3(8, 350, 0),
      name: "venus",
    },
    {
      text: "EARTH",
      size: 3.5,
      height: 1,
      pos: new THREE.Vector3(100, 350, 0),
      name: "earth",
    },
    {
      text: "MARS",
      size: 3.5,
      height: 1,
      pos: new THREE.Vector3(130, 350, 0),
      name: "mars",
    },
    {
      text: "JUPITER",
      size: 3.5,
      height: 1,
      pos: new THREE.Vector3(175, 350, 0),
      name: "jupiter",
    },
    {
```

Figure 31. Text defining

```
const createText = (text, size, height, pos, name) => {
  let loader = new FontLoader();
  loader.load("./three.js-r145-compressed/examples/fonts/helvetiker_regular.typeface.json",
    (font) => {
      let geometry = new TextGeometry(text, {
        font: font,
        size: size,
        height: height
      });
      geometry.center();
      let material = new THREE.MeshBasicMaterial({
        color: "orange",
        transparent: true,
        opacity: 0
      });
      let mesh = new THREE.Mesh(geometry, material);
      mesh.position.copy(pos);
      mesh.name = `${name}_text`;
      textList.add(mesh);
    });
};
```

Figure 32. createText function

```
const createSunText = (text, size, height, pos, name) => {
  const loader = new FontLoader();
  loader.load(
      "./three.js-r145-compressed/examples/fonts/helvetiker_regular.typeface.json",
      (font) => {
        const geometry = new TextGeometry(text, {
          font: font,
          size: size,
          height: height,
        });
        geometry.center();

        const material = new THREE.MeshBasicMaterial({
          color: "orange",
          transparent: true,
          opacity: 0,
        });

        const mesh = new THREE.Mesh(geometry, material);


        const adjustedPos = pos.clone();
        adjustedPos.y += 50;
        mesh.position.copy(adjustedPos);

        mesh.name = `${name}_text`;

        textList.add(mesh);

        const animateText = () => {
          mesh.lookAt(camera.position);
          requestAnimationFrame(animateText);
        };
        animateText();
      }
  );
```

Figure 33. createSunText function

```javascript
const animatePlanets = () => {
    const time = Date.now() * 0.0001;

    Object.keys(speeds).forEach(planetName => {
        let planet = eval(planetName);
        let orbitRadius = orbitalRadii[planetName];
        let speed = speeds[planetName];

        planet.position.x = sun.position.x + orbitRadius * Math.cos(time * speed.orbit);
        planet.position.z = sun.position.z + orbitRadius * Math.sin(time * speed.orbit);
        planet.rotation.y += speed.rotation;
        if (planetName === 'earth') {
            Satellite.position.x = planet.position.x + 8;
            Satellite.position.z = planet.position.z;
            Satellite.position.y = planet.position.y;
        }
    });
};
```

Figure 34. animatePlanets function

```javascript
const animateText = () => {
    const time = Date.now() * 0.0001;

    textList.children.forEach((mesh, index) => {
        const planetNames = Object.keys(orbitalRadii);
        if (index >= planetNames.length) return;

        let planetName = planetNames[index];
        let planet = eval(planetName);

        if (!planet || !speeds[planetName]) return;

        let orbitRadius = orbitalRadii[planetName];
        let speed = speeds[planetName];

        mesh.position.x = sun.position.x + orbitRadius * Math.cos(time * speed.orbit);
        mesh.position.z = sun.position.z + orbitRadius * Math.sin(time * speed.orbit);

        mesh.rotation.y += speed.rotation;
        mesh.lookAt(camera.position);
    });
};
```

Figure 35. animateText function

```
const animate = () => {
  raycaster.setFromCamera(mouse, camera);

  const intersects = raycaster.intersectObjects(scene.children, true);

  if (intersects.length > 0) {
    const firstIntersect = intersects[0].object;

    const allowedObjects = ["sun", "mercury", "venus", "earth", "mars", "jupiter", "saturn", "uranus", "neptune"];

    if (firstIntersect.name !== "spaceship") {
      if (hoveredObject !== firstIntersect) {

        if (hoveredObject && hoveredObject.name) {
          const prevText = textList.children.find(t => t.name === `${hoveredObject.name}_text`);
          if (prevText) prevText.material.opacity = 0;

          if (allowedObjects.includes(hoveredObject.name)) {
            hoveredObject.material.color.set(hoveredObject.originalColor);
          }
        }

        hoveredObject = firstIntersect;

        if (allowedObjects.includes(hoveredObject.name)) {
          hoveredObject.originalColor = hoveredObject.material.color.getHex();
          hoveredObject.material.color.set(getRandomColor());
        }


        if (hoveredObject.name) {
          const newText = textList.children.find(t => t.name === `${hoveredObject.name}_text`);
          if (newText) newText.material.opacity = 1;
        }
      }
    }
  } else if (hoveredObject) {
    if (hoveredObject.name) {
      const textToHide = textList.children.find(t => t.name === `${hoveredObject.name}_text`);
      if (textToHide) textToHide.material.opacity = 0;

      if (allowedObjects.includes(hoveredObject.name)) {
        hoveredObject.material.color.set(hoveredObject.originalColor);
      }
    }
    hoveredObject = null;
  }
};
```

Figure 36. animate function

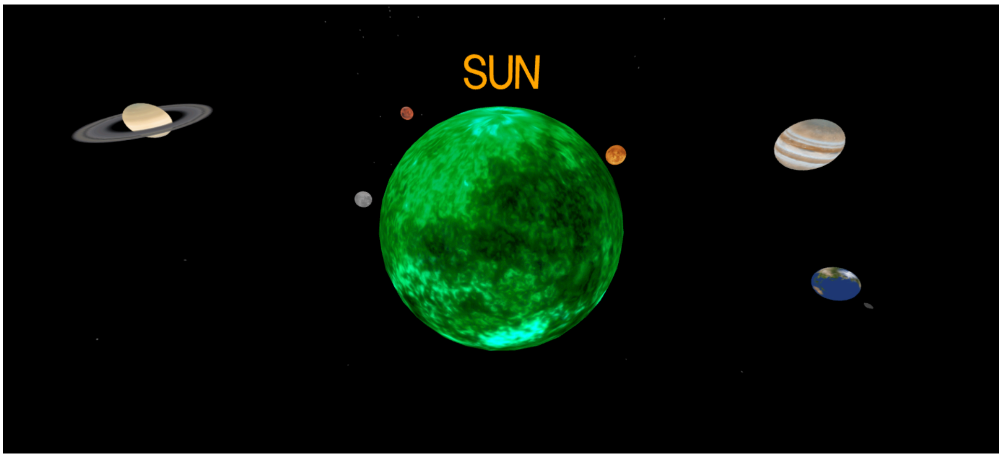


Figure 37. Raycast on hovered objects at Free Rotating camera only

g) Skybox

The skybox is created to give a 3D **background** to the entire scene. For this project, the Skybox displays a black scene with white dots resembling a **starry sky**.

```
const createPathString = () => {
  const basePath = "./assets/skybox/";
  const sides = [
    "right.png",
    "left.png",
    "top.png",
    "bottom.png",
    "front.png",
    "back.png"
  ];
  return sides.map((side) => basePath + side);
};

const createMaterialArray = () => {
  const loader = new THREE.TextureLoader();
  const skyboxImagePaths = createPathString();
  const materialArray = [];

  skyboxImagePaths.forEach((image) => {
    const texture = loader.load(
      image,
      () => console.log(`Loaded texture: ${image}`),
      undefined,
      (err) => console.error(`Failed to load texture: ${image}`, err)
    );
    const material = new THREE.MeshBasicMaterial({
      map: texture,
      side: THREE.BackSide,
    });
    materialArray.push(material);
  });
  return materialArray;
};

const createSkybox = () => {
  let skyboxMat = createMaterialArray();
  let skyboxGeo = new THREE.BoxGeometry(10000, 10000, 10000);
  let skybox = new THREE.Mesh(skyboxGeo, skyboxMat);

  scene.add(skybox);
  console.log("Skybox added:", skybox);
};
```

Figure 38. Skybox code

Figure 39. Skybox