> **User Prompt:**
> write a javascript app that uses three.js to show the locations of exoplanets in relation to the sun

To create this application, we will use **Three.js** for the 3D rendering and **OrbitControls** to allow you to zoom and rotate around the solar system.

Because the real distance between stars is measured in light-years (vastly larger than the size of planets), a 1:1 scale would make it impossible to see anything. Therefore, I have implemented a **Logarithmic Scale** approach: distances are scaled down so you can see the "neighborhood" of our Sun and nearby exoplanetary systems in one view.

You can save the code below as an `.html` file (e.g., `exoplanets.html`) and open it in any modern web browser.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Three.js Exoplanet Explorer</title>
    <style>
        body { margin: 0; background-color: #000; overflow: hidden; font-
family: sans-serif; }
        canvas { display: block; }
        #ui {
            position: absolute;
            top: 20px;
            left: 20px;
            color: white;
            pointer-events: none;
            text-shadow: 1px 1px 2px black;
        }
        .label {
            font-size: 14px;
            background: rgba(0, 0, 0, 0.5);
            padding: 2px 5px;
            border-radius: 3px;
        }
        #instructions {
            position: absolute;
            bottom: 20px;
            width: 100%;
            text-align: center;
            color: #888;
            font-size: 12px;
        }
    </style>
</head>
<body>
```

```
    <div id="ui">
        <h1>Exoplanet Explorer</h1>
        <p>Visualizing nearby planetary systems</p>
        <div id="planet-info">Hover over a planet</div>
    </div>

    <div id="instructions">
        Use Mouse Wheel to Zoom | Left Click to Rotate | Right Click to Pan
    </div>

    <!-- Import Three.js via CDN -->
    <script type="importmap">
        {
            "imports": {
                "three":
"https://unpkg.com/three@0.160.0/build/three.module.js",
                "three/addons/":
"https://unpkg.com/three@0.160.0/examples/jsm/"
            }
        }
    </script>

    <script type="module">
        import * as THREE from 'three';
        import { OrbitControls } from
'three/addons/controls/OrbitControls.js';

        // --- DATA CONFIGURATION ---
        // Distances are in Light Years (LY).
        // We scale them down significantly for visualization purposes.
        const planetsData = [
            { name: "Proxima Centauri b", distance: 4.24, color: 0xff7f50,
size: 0.5 },
            { name: "Barnard's Star b", distance: 6.0, color: 0x98fb98,
size: 0.4 },
            { name: "Wolf 1061c", distance: 14.0, color: 0xadd8e6, size:
0.6 },
            { name: "TRAPPIST-1e", distance: 39.5, color: 0xff69b4, size:
0.4 },
            { name: "Kepler-186f", distance: 582.0, color: 0xdaa520, size:
0.7 },
            { name: "HD 40307g", distance: 42.0, color: 0x9370db, size: 0.5
},
        ];

        // Scaling factor: How many light years = 1 unit in our 3D scene
        const DISTANCE_SCALE = 2;

        // --- SCENE SETUP ---
        const scene = new THREE.Scene();
        const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 2000);
        const renderer = new THREE.WebGLRenderer({ antialias: true });
        renderer.setSize(window.innerWidth, window.innerHeight);
```

```
        renderer.setPixelRatio(window.devicePixelRatio);
        document.body.appendChild(renderer.domElement);

        const controls = new OrbitControls(camera, renderer.domElement);
        camera.position.set(0, 20, 40);
        controls.update();

        // --- LIGHTING ---
        const ambientLight = new THREE.AmbientLight(0x40404  , 0.5); //
Soft white light
        scene.add(ambientLight);

        const sunLight = new THREE.PointLight(0xffffff, 1000, 500); //
Light coming from the Sun
        scene.add(sunLight);

        // --- OBJECTS ---

        // 1. The Sun
        const sunGeometry = new THREE.SphereGeometry(2, 32, 32);
        const sunMaterial = new THREE.MeshBasicMaterial({ color: 0xffdd00
});
        const sun = new THREE.Mesh(sunGeometry, sunMaterial);
        scene.add(sun);

        // Sun Glow (Simple Sprite)
        const spriteMaterial = new THREE.SpriteMaterial({
            map: new
THREE.TextureLoader().load('https://threejs.org/examples/textures/lensflare
/lensflare0.png'),
            color: 0xffaa00, transparent: true, opacity: 0.8, blending:
THREE.AdditiveBlending
        });
        const sunGlow = new THREE.Sprite(spriteMaterial);
        sunGlow.scale.set(10, 10, 1);
        sun.add(sunGlow);

        // 2. Starfield Background
        function createStarfield() {
            const vertices = [];
            for (let i = 0; i < 5000; i++) {
                const x = THREE.MathUtils.randFloatSpread(1000);
                const y = THREE.MathUtils.randFloatSpread(1000);
                const z = THREE.MathUtils.randFloatSpread(1000);
                vertices.push(x, y, z);
            }
            const geometry = new THREE.BufferGeometry();
            geometry.setAttribute('position', new
THREE.Float32BufferAttribute(vertices, 3));
            const material = new THREE.PointsMaterial({ color: 0xffffff,
size: 0.7 });
            const points = new THREE.Points(geometry, material);
            scene.add(points);
        }
```

```
        createStarfield();

        // 3. The Planets
        const planetMeshes = [];
        planetsData.forEach(data => {
            // Create a group to handle rotation/orbit easily
            const orbitGroup = new THREE.Group();
            scene.add(orbitGroup);

            // Draw Orbit Line
            const orbitRadius = data.distance * DISTANCE_SCALE;
            const orbitCurve = new THREE.EllipseCurve(0, 0, orbitRadius,
orbitRadius);
            const points = orbitCurve.getPoints(100);
            const orbitGeometry = new
THREE.BufferGeometry().setFromPoints(points);
            const orbitMaterial = new THREE.LineBasicMaterial({ color:
0x333333 });
            const orbitLine = new THREE.LineLoop(orbitGeometry,
orbitMaterial);
            orbitLine.rotation.x = Math.PI / 2;
            scene.add(orbitLine);

            // Create Planet
            const geometry = new THREE.SphereGeometry(data.size, 32, 32);
            const material = new THREE.MeshStandardMaterial({ color:
data.color, roughness: 0.5 });
            const planet = new THREE.Mesh(geometry, material);

            // Position planet on the orbit ring
            planet.position.x = orbitRadius;

            // Store metadata for interaction
            planet.userData = { name: data.name, distance: data.distance };

            orbitGroup.add(planet);
            planetMeshes.push(planet);
        });

        // --- INTERACTION (Raycasting) ---
        const raycaster = new THREE.Raycaster();
        const mouse = new THREE.Vector2();
        const infoElement = document.getElementById('planet-info');

        window.addEventListener('mousemove', (event) => {
            mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
            mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;
        });

        // --- ANIMATION LOOP ---
        function animate() {
            requestAnimationFrame(animate);

            // Rotate Sun
```

```
            sun.rotation.y += 0.005;

            // Rotate Planets in their orbits (simulated)
            planetMeshes.forEach((planet, index) => {
                const speed = 0.01 / (index + 1);
                planet.parent.rotation.y += speed;
            });

            // Raycasting for hover effect
            raycaster.setFromCamera(mouse, camera);
            const intersects = raycaster.intersectObjects(planetMeshes);

            if (intersects.length > 0) {
                const target = intersects[0].object;
                infoElement.innerHTML = `<strong>${target.userData.name}
</strong><br>Distance: ${target.userData.distance} LY`;
                document.body.style.cursor = 'pointer';
            } else {
                infoElement.innerHTML = "Hover over a planet";
                document.body.style.cursor = 'default';
            }

            controls.update();
            renderer.render(scene, camera);
        }

        // Handle Window Resize
        window.addEventListener('resize', () => {
            camera.aspect = window.innerWidth / window.innerHeight;
            camera.updateProjectionMatrix();
            renderer.setSize(window.innerWidth, window.innerHeight);
        });

        animate();
    </script>
</body>
</html>
```

Key Features of this App:

1. **Logarithmic/Scaled Perspective**: I used a `DISTANCE_SCALE` constant. In real life, Kepler-186f is 582 light years away. If we used true scale, it would be thousands of screens away from the Sun. Here, I've compressed the distances so you can see the "Solar Neighborhood" in one view.
2. **The Sun**: Created with a `MeshBasicMaterial` (so it appears to glow) and an additive-blending sprite to create a "lens flare" effect.
3. **Orbital Paths**: I used `THREE.EllipseCurve` to draw grey rings around the sun, representing the orbital plane of each planet.
4. **Interactivity**:
    - **OrbitControls**: You can left-click to rotate the camera, scroll to zoom in on specific planets, and right-click to pan.

    - **Raycasting**: The app detects when your mouse hovers over a specific planet sphere and updates the HTML UI in the top-left corner with the planet's name and distance.
5. **Starfield**: A custom particle system generates 5,000 random stars in 3D space to provide depth and context.

How to run it:

1. Copy the code above.
2. Save it as `index.html`.
3. Double-click the file to open it in Chrome, Firefox, or Edge. (No local server is required because I used CDN imports).