**CREATING FRONTEND PROCESS**

**ROADMAP**

**🛰️ Phase 1: Setting Up the Project**

1. **Choose the Right Tech Stack**
   * **Frontend:** React + Vite (fast and modern)
   * **3D Graphics:** React-Three-Fiber (Three.js for React)
   * **Data Handling:** satellite.js (to parse TLE and compute positions)
   * **State Management:** Zustand (lightweight alternative to Redux)
2. **Initialize the React Project**
   * Install dependencies and set up Vite
   * Configure Three.js and CesiumJS

**🌍 Phase 2: Implementing 3D Orbital Visualization**

1. **Load and Parse TLE Data**
   * Use satellite.js to extract orbital parameters and position
   * Convert TLE data into 3D coordinates
2. **Render Earth in 3D**
   * Use react-three-fiber to create a rotating Earth
   * Add a realistic atmosphere and lighting
3. **Render Satellites and Debris in Orbit**
   * Use InstancedMesh for high-performance rendering
   * Animate objects in real-time using satellite.js

**🚨 Phase 3: Collision Alerts & UI Components**

1. **Implement Collision Risk Calculation**
   * Integrate ML predictions into UI
   * Visualize collision alerts with connecting lines
2. **Add User Interaction (Click, Hover, Search)**
   * Display satellite details on hover/click
   * Implement search and filter functions
3. **Time Slider & Animation Controls**
   * Fast-forward, rewind, and pause orbital movements
   * Display upcoming close approaches

**🎨 Phase 4: UI/UX Enhancements**

1. **Design a Dark-Themed UI**
   * Floating panels for search & settings
   * Modern buttons and animations
2. **Optimize Performance**

* Use requestAnimationFrame for smooth rendering
* Implement lazy loading for large datasets

**🚀 Phase 5: Deployment & Testing**

1. **Test Interactivity and Performance**
2. **Deploy the App on Vercel or Netlify**

**🚀 Step 1: Setting Up React with Vite**

Vite is a modern tool for building React apps—it’s faster than Create React App (CRA).

1️⃣ Install Node.js

📌 **Check if you have it:** Open a terminal and type: node -v

**Create a New React Project with Vite**

Now, let's create our React project using Vite.

🔹 Open your terminal (Command Prompt, PowerShell, or macOS Terminal) and run:

***npm create vite@latest space-debris-visualization --template react***

Navigate to your project folder: ***cd space-debris-visualization***

Install dependencies: ***npm install***

Start the development server: ***npm run dev***

This will start a local web server. Open your browser and go to **http://localhost:5173/**. You should see the default Vite React page.

✅ **Success!** Your React project is set up! 🎉

**🚀 Step 2: Installing Required Packages**

We need a few libraries to build our 3D visualization. Run the following command in your project folder:

***npm install @react-three/fiber three @react-three/drei satellite.js zustand***

**What Each Package Does:**

* @react-three/fiber → Integrates Three.js with React for 3D graphics
* three → The core 3D rendering library
* @react-three/drei → Useful helpers for 3D scenes (like orbits, cameras)
* satellite.js → Converts TLE data into 3D positions
* zustand → Lightweight state management for handling object selections

✅ **At this point, you have:**  
✔ Installed React + Vite  
✔ Installed 3D rendering and satellite tracking libraries

**Create a 3D Scene Component**

Inside your project folder:  
🔹 Open the **src/** directory.  
🔹 Create a new file: **src/ThreeScene.jsx**  
🔹 Add the following code:

import React from "react";

import { Canvas } from "@react-three/fiber";

import { OrbitControls, Stars } from "@react-three/drei";

const ThreeScene = () => {

return (

<Canvas camera={{ position: [0, 0, 5] }}>

{/\* Background stars \*/}

<Stars />

{/\* Add some light \*/}

<ambientLight intensity={0.5} />

<directionalLight position={[2, 2, 2]} intensity={1} />

{/\* Earth Sphere \*/}

<mesh>

<sphereGeometry args={[1, 32, 32]} />

<meshStandardMaterial color="blue" />

</mesh>

{/\* Controls to rotate the scene \*/}

<OrbitControls />

</Canvas>

);

};

export default ThreeScene;

**2️⃣ Use the 3D Scene in App.jsx**

Now, open **src/App.jsx** and **replace everything** with:

import React from "react";

import ThreeScene from "./ThreeScene";

function App() {

return (

<div style={{ width: "100vw", height: "100vh", background: "black" }}>

<ThreeScene />

</div>

);

}

export default App;

Go to **http://localhost:5173/**, and you should see: ✅ A **3D blue sphere** (representing Earth).  
✅ **Orbit controls** (drag to rotate).  
✅ **Stars in the background** for a space effect.

**Add a Realistic Earth Texture**

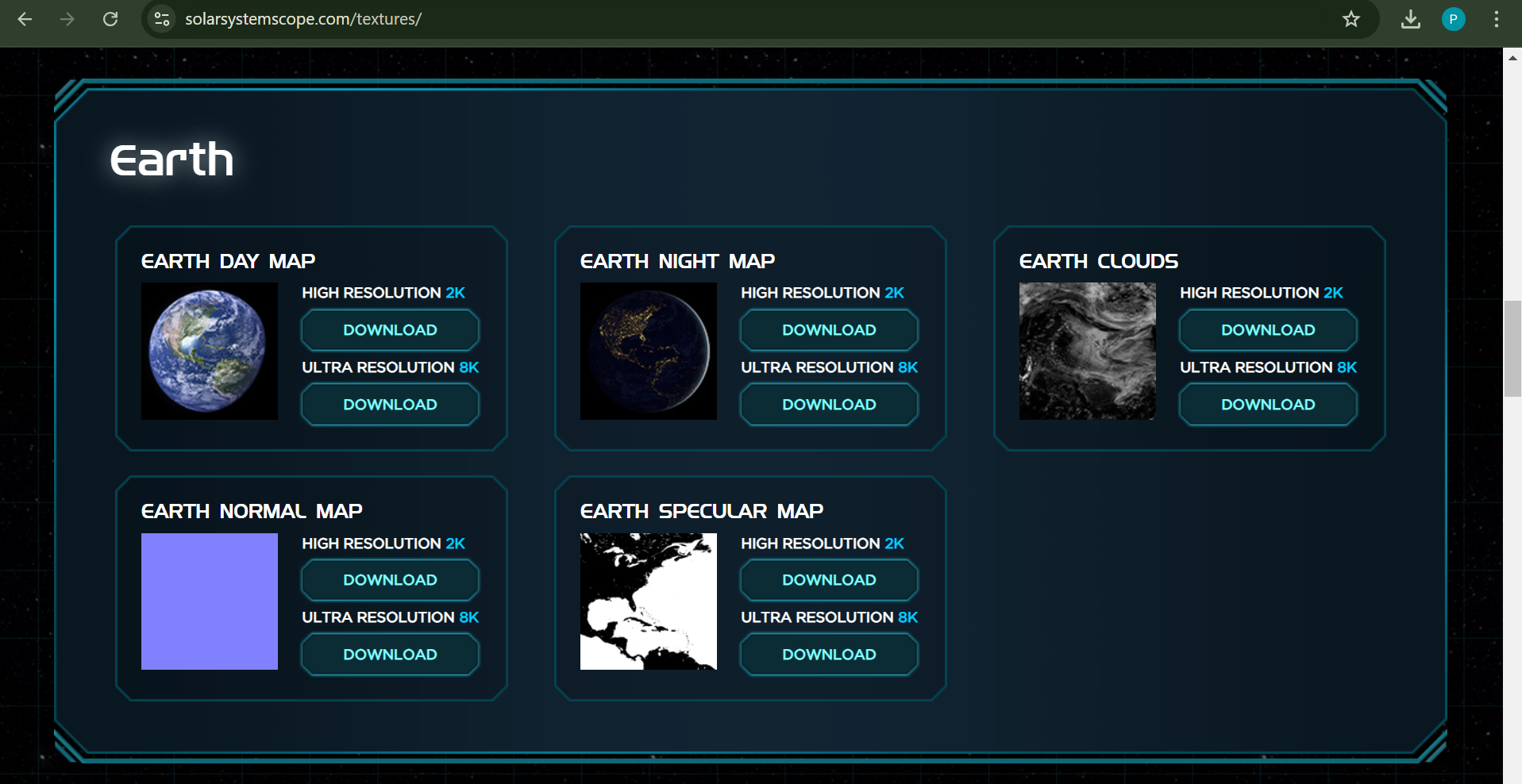
**Install three.js (for better texture handling)**

Run this command inside your project folder:

***npm install three***

**Download Earth Textures**

You'll need **Earth's color texture** (with land) and optionally a **cloud texture** for realism. Use these free high-res textures:



Download Earth texture from above site. Downloaded all expect earth normal map.

**File name: Earth.jsx**

import { useRef } from "react";

import { useFrame, useLoader } from "@react-three/fiber";

import { TextureLoader } from "three";

function Earth() {

  const earthRef = useRef();

  // Load textures (Ensure the files exist in public/textures/)

  const [colorMap, specularMap] = useLoader(TextureLoader, [

    "/textures/earth\_day.jpg",

    "/textures/earth\_specular.jpg",

  ]);

  useFrame(() => {

    earthRef.current.rotation.y += 0.002; // Slow rotation

  });

  return (

    <mesh ref={earthRef} position={[0, 0, 0]}>

      <sphereGeometry args={[1, 64, 64]} />

      <meshStandardMaterial map={colorMap} />

    </mesh>

  );

}

export default Earth;

**File name: App.jsx**

import { Canvas } from "@react-three/fiber";

import { OrbitControls } from "@react-three/drei";

import Earth from "./Earth";

function App() {

  return (

    <div style={{ width: "100vw", height: "100vh", background: "black" }}>

      <Canvas

        camera={{ position: [0, 0, 3], fov: 50 }}

        style={{ width: "100vw", height: "100vh" }}

      >

        <ambientLight intensity={0.5} />

        <directionalLight position={[5, 5, 5]} intensity={1} />

        <Earth />

        <OrbitControls enableZoom={true} />

      </Canvas>

    </div>

  );

}

export default App;

**Steps to Implement Space Debris Visualization**

1. **Get Space Debris Data**
   * Use **NORAD's TLE data** from [Celestrak](https://www.celestrak.com/) or [Space-Track](https://www.space-track.org/).
   * Use **GP data** (latitude, longitude, altitude) if available.
2. **Load & Parse Debris Data**
   * Convert **TLE (Two-Line Element Set)** to **3D coordinates**.
   * Use the satellite.js library for calculations.
3. **Render Debris in Three.js**
   * Represent debris as **small red dots** or sprites around the Earth.
   * Use a **BufferGeometry** for efficiency.
4. **Animate Debris Orbits**
   * Continuously update positions based on orbital mechanics.

**Full Code: Space Debris Visualization in Three.js**

This assumes you're using **React + Three.js + Drei + satellite.js**.

**1️⃣ Install Dependencies**

***npm install three @react-three/fiber @react-three/drei satellite.js d3-fetch***

**Fetch TLE Data (utils/fetchTLE.js)**

import \* as d3 from "d3-fetch";

export async function fetchTLEData() {

  const url = "https://celestrak.com/NORAD/elements/stations.txt";

  try {

    const text = await d3.text(url);

    const lines = text.trim().split("\n");

    if (lines.length < 3) {

      console.error("TLE data seems empty");

      return [];

    }

    console.log("TLE Data Loaded:", lines.slice(0, 6)); // Log first satellite

    const satellites = [];

    for (let i = 0; i < lines.length; i += 3) {

      satellites.push({

        name: lines[i].trim(),

        line1: lines[i + 1].trim(),

        line2: lines[i + 2].trim(),

      });

    }

    return satellites;

  } catch (error) {

    console.error("Error fetching TLE:", error);

    return [];

  }

}

**🚀 What This Does**

✔ **Fetches real-time debris data**  
✔ **Converts TLE to 3D positions**  
✔ **Renders debris as red dots around Earth**  
✔ **Moves debris over time based on orbit calculations**

**File name:spacescence.jsx**

import { useRef } from "react";

import { useFrame } from "@react-three/fiber";

import \* as THREE from "three";

import { useLoader } from "@react-three/fiber";

import { TextureLoader } from "three";

const debrisCount = 100; // Change to add more debris

function SpaceScene() {

  const earthRef = useRef();

  const debrisRefs = useRef([]);

  // Load Earth texture

  const earthTexture = useLoader(TextureLoader, "/textures/earth\_day.jpg");

  // Generate random positions for debris

  const debris = new Array(debrisCount).fill().map(() => ({

    position: [

      (Math.random() - 0.5) \* 10,

      (Math.random() - 0.5) \* 10,

      (Math.random() - 0.5) \* 10,

    ],

    velocity: [

      (Math.random() - 0.5) \* 0.02,

      (Math.random() - 0.5) \* 0.02,

      (Math.random() - 0.5) \* 0.02,

    ],

  }));

  // Animate debris

  useFrame(() => {

    debrisRefs.current.forEach((ref, index) => {

      if (ref) {

        ref.position.x += debris[index].velocity[0];

        ref.position.y += debris[index].velocity[1];

        ref.position.z += debris[index].velocity[2];

      }

    });

  });

  return (

    <>

      {/\* Earth Sphere \*/}

      <mesh ref={earthRef} position={[0, 0, 0]}>

        <sphereGeometry args={[1, 32, 32]} />

        <meshStandardMaterial map={earthTexture} />

      </mesh>

      {/\* Space Debris \*/}

      {debris.map((d, i) => (

        <mesh

          key={i}

          position={d.position}

          ref={(el) => (debrisRefs.current[i] = el)}

        >

          <sphereGeometry args={[0.05, 8, 8]} />

          <meshStandardMaterial color="gray" />

        </mesh>

      ))}

    </>

  );

}

export default SpaceScene;

**Step 1: Load and Parse Satellite/Debris Data**

* We'll start by reading a sample of your dataset.
* Extract key orbital parameters (altitude, velocity, inclination, etc.).
* Format the data for use in the 3D visualization.

**Step 2: Render Satellites and Debris in Orbit**

* Position satellites and debris around Earth based on orbital parameters.
* Use Three.js to animate their movement.
* Ensure orbits are visually accurate.

**Step 3: Handle Click Events for Object Details**

* Allow users to click on a satellite or debris.
* Display basic details (name, altitude, velocity, collision risk, etc.).

**Step 4: Integrate Collision Risk Prediction**

* Connect your trained ML model.
* Show real-time risk assessment when selecting an object.
* **Step 1: Load and Parse Satellite/Debris Data**
* We'll start by reading a sample of your dataset and extracting key orbital parameters.

**1.1 Load the CSV File into React**

Since we're using React, we need to:

* Import the CSV file
* Read it and parse it into a usable format

**Install Dependencies**

We'll use papaparse to parse the CSV file in React. Run this in your project folder:

sh

CopyEdit

***npm install papaparse***

**Create a dataLoader.js File to Read the CSV**

Inside src/utils/dataLoader.js, add:

**Create a dataLoader.js File to Read the CSV**

Inside src/utils/dataLoader.js, add:

***import Papa from "papaparse";***

***export const loadSatelliteData = async () => {***

***const response = await fetch("/data/collision\_risk\_dataset\_preprocessed.csv");***

***const csvText = await response.text();***

***return new Promise((resolve) => {***

***Papa.parse(csvText, {***

***header: true, // Treat first row as headers***

***dynamicTyping: true, // Convert numbers automatically***

***complete: (results) => resolve(results.data),***

***});***

***});***

***};***

 This function fetches the CSV file from public/data/

 Parses it into an array of objects

 Automatically converts numbers for easy use

**Changes Needed:**

1. **Load the CSV data** in App.jsx and pass it to SpaceScene.
2. **Modify SpaceScene.jsx** to receive the data and render satellites.
3. **Modify App.jsx to Load Data**
4. Update your App.jsx to fetch satellite data and pass it to SpaceScene:

import { useEffect, useState } from "react";

import { Canvas } from "@react-three/fiber";

import { OrbitControls } from "@react-three/drei";

import SpaceScene from "./components/SpaceScene";

import { loadSatelliteData } from "./utils/dataLoader"; // Import function

function App() {

const [satellites, setSatellites] = useState([]);

useEffect(() => {

const fetchData = async () => {

const data = await loadSatelliteData();

console.log("Loaded Data:", data); // Debugging: Check data in console

setSatellites(data);

};

fetchData();

}, []);

return (

<div style={{ width: "100vw", height: "100vh", background: "black" }}>

<Canvas camera={{ position: [0, 0, 5], fov: 50 }}>

<ambientLight intensity={0.5} />

<directionalLight position={[5, 5, 5]} intensity={1} />

<SpaceScene satellites={satellites} /> {/\* Pass data \*/}

<OrbitControls enableZoom={true} />

</Canvas>

</div>

);

}

export default App;

✅ **Now satellites data is sent to SpaceScene!**

**2️⃣ Modify SpaceScene.jsx to Render Satellites**

Now, update SpaceScene.jsx to **map over satellites and render them in orbit**.

**SpaceScene.jsx**

**import { useMemo } from "react";**

**import { Sphere } from "@react-three/drei";**

**function SpaceScene({ satellites }) {**

**// Convert satellite orbital parameters to 3D positions**

**const satelliteObjects = useMemo(() => {**

**return satellites.map((sat, index) => {**

**const { altitude, inclination, right\_ascension } = sat;**

**// Convert orbital parameters to 3D coordinates**

**const radius = 1 + altitude / 40000; // Normalize altitude**

**const theta = (inclination \* Math.PI) / 180; // Convert to radians**

**const phi = (right\_ascension \* Math.PI) / 180; // Convert to radians**

**const x = radius \* Math.sin(theta) \* Math.cos(phi);**

**const y = radius \* Math.sin(theta) \* Math.sin(phi);**

**const z = radius \* Math.cos(theta);**

**return { id: index, x, y, z, ...sat };**

**});**

**}, [satellites]);**

**return (**

**<>**

**{satelliteObjects.map((sat) => (**

**<Sphere key={sat.id} args={[0.02, 16, 16]} position={[sat.x, sat.y, sat.z]}>**

**<meshStandardMaterial color="red" />**

**</Sphere>**

**))}**

**</>**

**);**

**}**

**export default SpaceScene;**

**What This Does?**

* **Converts orbital data into 3D positions.**
* **Maps over satellites and renders a small red sphere at their position.**
* **Uses useMemo for better performance.**