An Interactive Learning Journey with Three.js & React Three Fiber

Gamified 3D Solar System Explorer

Houda Mouttalib

@hm43 on GitHub

@houdam on Linkedin

Computer Science Lecturer – EMSI Groupe

PhD Researcher – XR for STEM Education, LPRI EMSI and ENSET Mohammedia

Focus: Web & XR for interactive 3D learning experiences

Google Scholar profile:

Google Scholar

Houda MOUTTALIB

Researcher at LPRI EMSI and ENSET Mohammedia
Verified email at etu.univh2c.ma - Homepage

XR Metaverse Education

TITLE

- **Revolutionizing engineering education: Creating a web-based teaching platform for immersive learning experiences**
  H Mouttalib, M Tabaa, M Youssfi
  Journal of Smart Cities and Society, 1-12
- **Work-in-Progress—Augmented Reality in Higher Education: Case of Electrical Drawings**
  H Mouttalib, M Tabaa, M Youssfi, Y Bounouader
  Immersive Learning Research-Academic, 232-238
- **Towards a platform for higher education in virtual reality of engineering sciences**
  H Mouttalib, M Tabaa, M Youssfi
  International Conference of Innovation and New Trends in Information …
- **Exploring the Horizon: Challenges and Solutions in Integrating Extended Reality (XR) into STEM Education**
  H Mouttalib, M Tabaa, M Youssfi
  International Conference on Smart Applications and Data Analysis, 159-172
- **XR Pedagogical Framework: Leveraging Augmented Reality for Effective Learning in Industrial Training Using CDIO**
  H Mouttalib, M Tabaa, M Youssfi, Y Bounouader
  Immersive Learning Research-Academic, 251-260
- **in Virtual Reality of Engineering Sciences**
  H Mouttalib, M Tabaa, M Youssfi[1]
  New Technologies, Artificial Intelligence and Smart Data: 10th International …

Ressources:

Repo: git clone git@github.com:hm43/**solar_system1**.git

Demo: https://jsnation25.vercel.app/

Workshop Plan

What we will be doing:

- Introduction & objectives
- Initiation to Three.js and React Three Fiber (R3F)
- Solar system context and scene design
- Guided coding: from basic scene to interactive explorer
- Recap, and resources

Workshop Plan

What we will be doing:

- Introduction & objectives
- Initiation to Three.js and React Three Fiber (R3F)
- Solar system context and scene design
- Guided coding: from basic scene to interactive explorer
- Recap, and resources

Introduction & objectives

5 concrete outcomes:

- Why 3D & XR
- Learning Objectives
- What You'll Build
- Why R3F
- Workshop Mindset

Introduction & objectives

Why 3D & XR

Introduction & objectives

Learning Objectives

1. Understand React Three Fiber's declarative paradigm and how it maps React components to Three.js objects
2. Apply scene graph patterns (groups, transforms, hierarchical animation) to build dynamic 3D scenes
3. Implement interactivity via raycasting and pointer events to create responsive, user-driven experiences
4. Recognize performance bottlenecks and optimize using instancing for large-scale data!

Introduction & objectives

What You'll Build

A Gamified 3D Solar System Explorer:

- ✓ Animated planets orbiting the sun with realistic orbital mechanics
- ✓ Interactive raycasting: click planets to reveal facts and complete missions
- ✓ Gamified tasks: order planets by orbital period, discover hidden properties
- ✓ Performance-optimized starfield using instanced rendering

Introduction & objectives

Why R3F

The Challenge with Raw Three.js:
– Boilerplate: scene, camera, renderer setup
– Imperative: you manage lifecycle, updates, re-renders manually
– State management: hard to wire React state into Three.js objects

React Three Fiber Solution:
– Declarative: describe 3D scene as JSX components
– Automatic lifecycle: React handles mounts, updates
– Reactive: React state automatically flows into 3D objects

Introduction & objectives

Workshop
Mindset

Today's focus is not:
✗ Memorizing every Three.js API
✗ Building a production-grade app

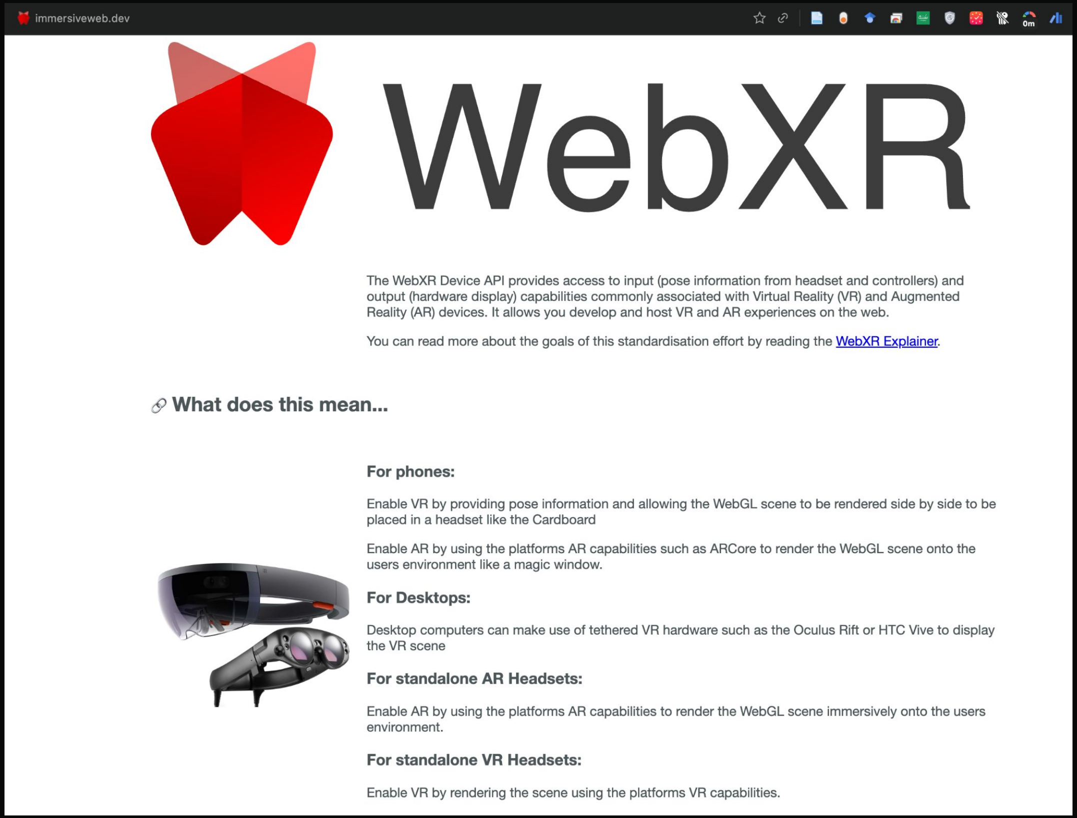
Today's focus IS:
✓ Understanding core patterns (scene graph, animation, interaction, performance)
✓ Seeing how R3F connects React to spatial computing
✓ Building confidence that 3D & XR are reachable with your existing skills

Workshop Plan

What we will be doing:

- Introduction & objectives
- Initiation to Three.js and React Three Fiber (R3F)
- Solar system context and scene design
- Guided coding: from basic scene to interactive explorer
- Recap, and resources

What is WebXR?



immersiveweb.dev

WebXR

The WebXR Device API provides access to input (pose information from headset and controllers) and output (hardware display) capabilities commonly associated with Virtual Reality (VR) and Augmented Reality (AR) devices. It allows you develop and host VR and AR experiences on the web.

You can read more about the goals of this standardisation effort by reading the WebXR Explainer.

What does this mean...

For phones:

Enable VR by providing pose information and allowing the WebGL scene to be rendered side by side to be placed in a headset like the Cardboard

Enable AR by using the platforms AR capabilities such as ARCore to render the WebGL scene onto the users environment like a magic window.

For Desktops:

Desktop computers can make use of tethered VR hardware such as the Oculus Rift or HTC Vive to display the VR scene

For standalone AR Headsets:

Enable AR by using the platforms AR capabilities to render the WebGL scene immersively onto the users environment.

For standalone VR Headsets:

Enable VR by rendering the scene using the platforms VR capabilities.

https://github.com/immersive-web/webxr/blob/master/explainer.md three.js

```js
// The minimal Three.js setup
const scene = new THREE.Scene()
const camera = new
THREE.PerspectiveCamera(75, w/h, 0.1, 1000)
const renderer = new THREE.WebGLRenderer()

renderer.render(scene, camera)
```

React-Three-Fiber

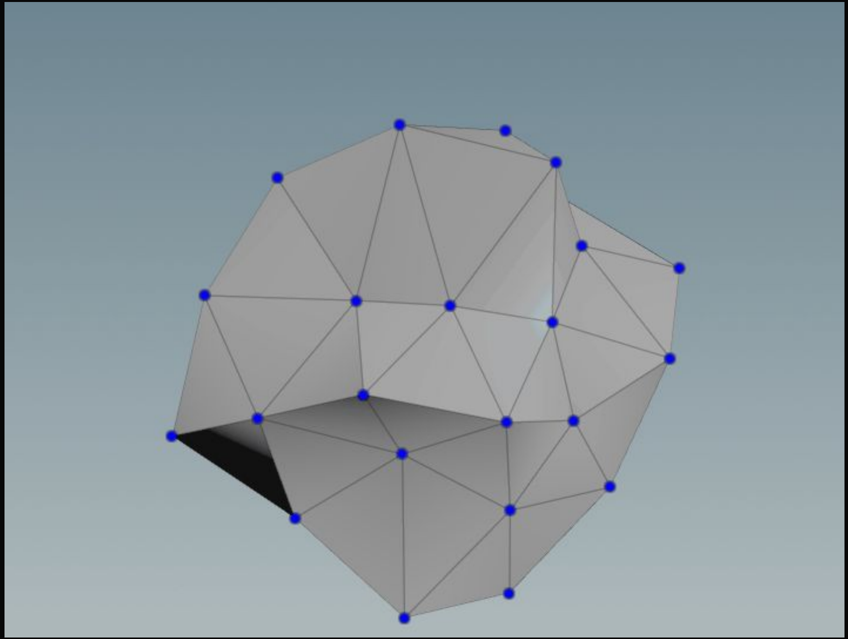
```jsx
<Canvas>
  {/* scene + camera are implicit */}
  <mesh position={[0, 0, 0]} />
</Canvas>
```

React-Three-Fiber Customization

```
<Canvas
  // Camera props
  camera={{
    position: [0, 5, 10],
    fov: 75,               // field of view (angle)
    near: 0.1,            // near clipping plane
    far: 1000,            // far clipping plane
    aspect: w / h         // auto-calculated
  }}

  // Renderer props
  gl={{
    antialias: true,      // smooth edges
    shadowMap: true,      // enable shadows
    shadowMapType: THREE.PCFShadowShadowMap,
    pixelRatio: devicePixelRatio
  }}

  // Scene settings via children
  style={{ background: '#000000' }}
>
  <fog attach="fog" args={[0x000000, 100, 1000]} />
  <ambientLight intensity={0.5} />
  <directionalLight position={[10, 10, 10]} castShadow />

</Canvas>
```

Mesh: Combining Geometry + Material

```
// What is a Mesh?
<mesh position={[0, 0, 0]} rotation={[0, 0, 0]} scale={[1, 1, 1]}>
  {/* Geometry = the shape */}
  <sphereGeometry args={[1, 32, 32]} />

  {/* Material = the appearance */}
  <meshStandardMaterial color="blue" />
</mesh>

// Mesh = Geometry + Material + Transform
// Without geometry → no shape
// Without material → invisible
// Transform (position, rotation, scale) → placement in space
```



Geometries as JSX Components with args

https://threejs.org/docs/index.html#SphereGeometry

1. JSX component = Three.js constructor call
   <sphereGeometry args={[1, 32, 32]} /> in R3F
   new THREE.SphereGeometry(1, 32, 32) in raw Three.js

2. args array = constructor parameters in order

3. Reference the docs when you need a specific geometry

4. Same parameters, declarative syntax

Animation Fundamentals

```
// Raw Three.js: Animation loop
function animate() {
  requestAnimationFrame(animate)
  mesh.rotation.x += 0.01
  mesh.rotation.y += 0.02
  renderer.render(scene, camera)
}
animate()
```

Animation Fundamentals: useFrame Hook

```
// React Three Fiber: useFrame hook
import { useFrame } from '@react-three/fiber'
import { useRef } from 'react'

function RotatingMesh() {
  const meshRef = useRef()

  useFrame(() => {
    meshRef.current.rotation.x += 0.01
    meshRef.current.rotation.y += 0.02
  })

  return (
    <mesh ref={meshRef}>
      <sphereGeometry args={[1, 32, 32]} />
      <meshStandardMaterial color="blue" />
    </mesh>
  )
}
```

60 FPS

Animation Fundamentals: Context of useFrame Hook

```
// useFrame context object
useFrame(({ clock, delta, gl, camera }) => {
  // Option 1: Use elapsed time (good for consistent speed)
  meshRef.current.rotation.y = clock.elapsedTime * speed

  // Option 2: Use delta (frame-independent, smooth)
  meshRef.current.rotation.x += 0.01 * delta * 60

  // Option 3: Direct reference, no overhead
  const mesh = meshRef.current
  mesh.position.x = Math.cos(clock.elapsedTime) * 5
  mesh.position.z = Math.sin(clock.elapsedTime) * 5
})
```

Performance: useFrame Best Practices

```
// ✓ GOOD: Minimize allocations, reuse objects
function Planet() {
  const meshRef = useRef()

  useFrame(() => {
    const mesh = meshRef.current
    mesh.rotation.y += 0.01  // Direct mutation, fast
  })

  return <mesh ref={meshRef}>...</mesh>
}

// ✗ BAD: Creating new objects every frame
function BadPlanet() {
  useFrame(() => {
    // DON'T create vectors every frame!
    const newPosition = new THREE.Vector3(1, 2, 3)
    meshRef.current.position.copy(newPosition)
  })
}
```

Performance: useFrame Best Practices

```
// ✓ GOOD: Cache vectors outside useFrame
function GoodPlanet() {
  const meshRef = useRef()
  const positionVector = useRef(new THREE.Vector3(1, 2, 3))

  useFrame(() => {
    meshRef.current.position.copy(positionVector.current)
  })
}

// ✓ GOOD: Stop animation when not needed
function OptimizedPlanet() {
  const meshRef = useRef()

  useFrame(({ clock }) => {
    if (isVisible) {  // Only animate if on screen
      meshRef.current.rotation.y = clock.elapsedTime * 0.5
    }
  })
}
```

Drei: Helper Library for R3F

```
// Load textures easily
import { useTexture } from '@react-three/drei'

function TexturedPlanet() {
  const texture = useTexture('/earth.jpg')
  return (
    <mesh>
      <sphereGeometry args={[1, 64, 64]} />
      <meshStandardMaterial map={texture} />
    </mesh>
  )
}
```

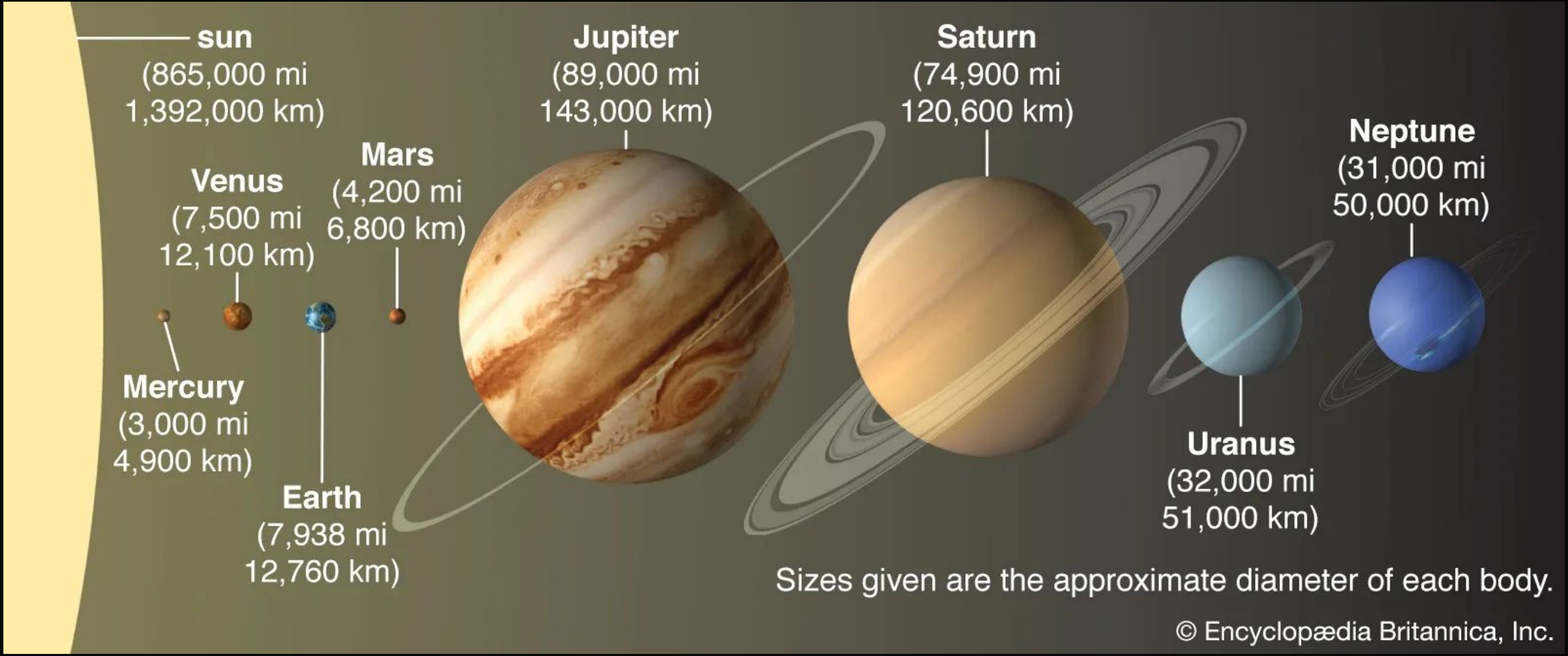
```
// Pre-built starfield
import { Stars } from '@react-three/drei'

<Stars radius={100} depth={50} count={5000} factor={4} />

// Optional: camera controls (mouse rotation)
import { OrbitControls } from '@react-three/drei'

<OrbitControls />

// Embed HTML in 3D scene
import { Html } from '@react-three/drei'

<Html position={[0, 2, 0]}>
  <h1>Planet Info</h1>
</Html>
```

Workshop Plan

What we will be doing:

- Introduction & objectives
- Initiation to Three.js and React Three Fiber (R3F)
- Solar system context and scene design
- Guided coding: from basic scene to interactive explorer
- Recap, and resources

Pedagogical Scaling: Why We Scale the Solar System



Project Structure: Component Architecture

File Organization:

```
assets/
  └── planets.json         ← Planet data (size, distance, color, etc.)

Components:
  ├── App.jsx              ← Main app, state management
  ├── Planet.jsx           ← Individual planet (mesh + animation)
  ├── OrbitPath.jsx        ← Visual orbit line
  ├── Moon.jsx             ← Planet's moon
  ├── CameraController.jsx ← Focus on selected planet
  └── PlanetSelector.jsx   ← UI for planet selection
```

Data-Driven Approach:

planets.json → App.jsx → planetData.map() → <Planet />

Workshop Plan

What we will be doing:

- Introduction & objectives
- Initiation to Three.js and React Three Fiber (R3F)
- Solar system context and scene design
- Guided coding: from basic scene to interactive explorer
- Recap, and resources

Create React App with Vite

```
mac@mymac Workspace % npm create vite@latest solar_system1 --template react

> npx
> create-vite solar_system1 react

│
◇  Select a framework:
│  React
│
◇  Select a variant:
│  JavaScript
│
◇  Use rolldown-vite (Experimental)?:
│  No
│
◇  Install with npm and start now?
│  Yes
│
◇  Scaffolding project in /Users/mac/Workspace/solar_system1...
│
◇  Installing dependencies with npm...
```

Install Dependencies

```
mac@mymac solar_system1 % npm install three @react-three/fiber @react-three/drei


added 61 packages, and audited 218 packages in 14s

36 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Planets data

Property	Purpose	Example
`name`	Planet name	"Earth"
`color`	Hex color for rendering	"#2f6a69"
`size`	Relative size (realistic ratio)	0.64
`distance`	Pedagogical distance (scaled)	30
`radius`	3D render radius	2
`speed`	Orbital speed multiplier	0.01
`rotationSpeed`	Planet spin speed	0.02
`period`	Orbital period in days (real)	365
`inclination`	Orbital inclination in degrees	0.0
`info`	Educational description	"Our home..."
`moons`	Array of moon objects	[...]

Creating the Canvas and the components

```
<Canvas camera={{ position: [0, 60, 120], fov: 45 }} style={{background: "#000"}}>
  {/* Controls */}
  <OrbitControls
    makeDefault
    minDistance={5}
    maxDistance={400}
    enablePan={true}
  />

  {/* Lighting */}
  <ambientLight intensity={0.5} />
```

Creating the Orbits

```
export default function OrbitPath({ radius, inclination = 0, color = "#EEE" }) {
  const tilt = (inclination * Math.PI) / 180 // Degrees to radians
  return (
    <mesh rotation={[-Math.PI / 2 + tilt, 0, 0]}>
      <torusGeometry args={[radius, 0.01, 16, 160]} />
      <meshBasicMaterial color={color} />
    </mesh>
  )
}
```

Creating the planets

```
<group ref={groupRef}>
  <mesh
    ref={meshRef}
    position={[planet.distance, 0, 0]}
    onClick={handleClick}
    onPointerOver={handlePointerOver}
    onPointerOut={handlePointerOut}
  >
    <sphereGeometry args={[planet.radius, 64, 64]} />
    <meshStandardMaterial
      color={planet.color}
      emissive={isActive ? planet.color : '#000000'}
      emissiveIntensity={isActive ? 2 : 0}
      wireframe={false}
    />
  </mesh>
</group>
```

Houda MOUTTALIB

Geek | XR Enthusiast | PhD Candidate | Researcher at LPRI | Professor

Casablanca, Casablanca-Settat, Morocco · Contact info

THANK YOU!