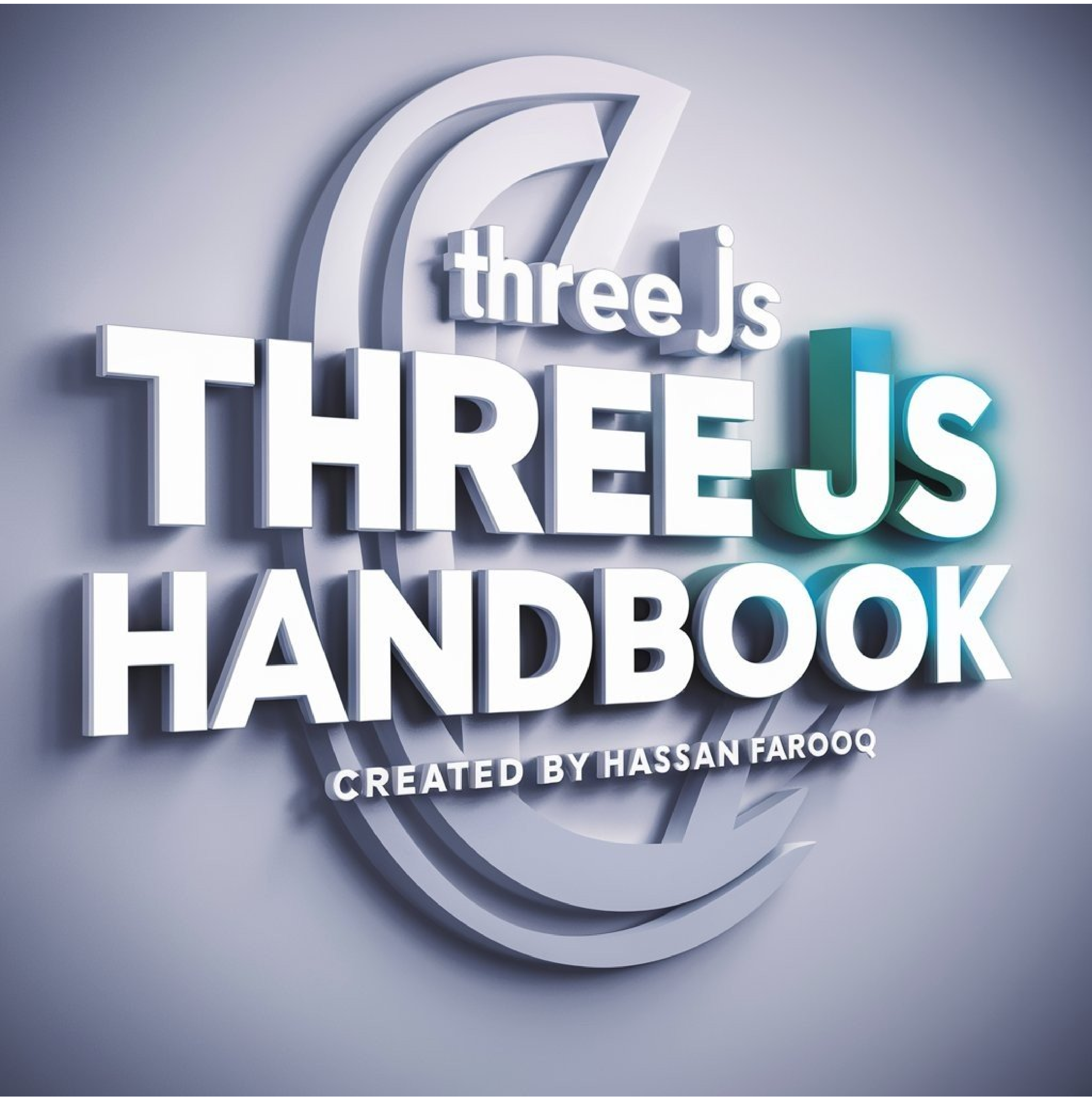

three js
THREE JS
HANDBOOK
CREATED BY HASSAN FAROOQ

What's in the guide?

Welcome to our Three.js cheat sheet! This guide provides a quick reference to the fundamental concepts and techniques used in Three.js, a powerful JavaScript library for creating 3D graphics and animations in the browser.

Whether you're a beginner or an experienced dev, this cheat sheet is designed to help you quickly find and use the most commonly used Three.js features.

In this cheat sheet, you'll find code snippets and examples for setting up a scene, creating and manipulating 3D objects, adding lighting and materials, using cameras, and more.

So, whether you're building a 3D game, a data visualization, or just experimenting with Three.js, this cheat sheet is a valuable resource to have. *Let's start...*

What is Three.js

Three.js is a powerful, open-source library for creating stunning 3D visuals on the web. It provides developers with an easy-to-use set of tools and utilities for building interactive, animated 3D graphics that can be rendered in any modern web browser.

With Three.js, developers can create a wide range of 3D objects and scenes, including complex geometries, dynamic particle systems, and realistic lighting and shadow effects. Whether you're building a game, a data visualization, or an interactive product demo, Three.js offers the flexibility and power you need to bring your ideas to life.

One of the key benefits of Three.js is its broad compatibility with different web technologies and frameworks, including HTML5, CSS, and JavaScript.

What is Three.js

This means that you can easily integrate Three.js into your existing web projects or start from scratch with a new project, using familiar web development tools and techniques.

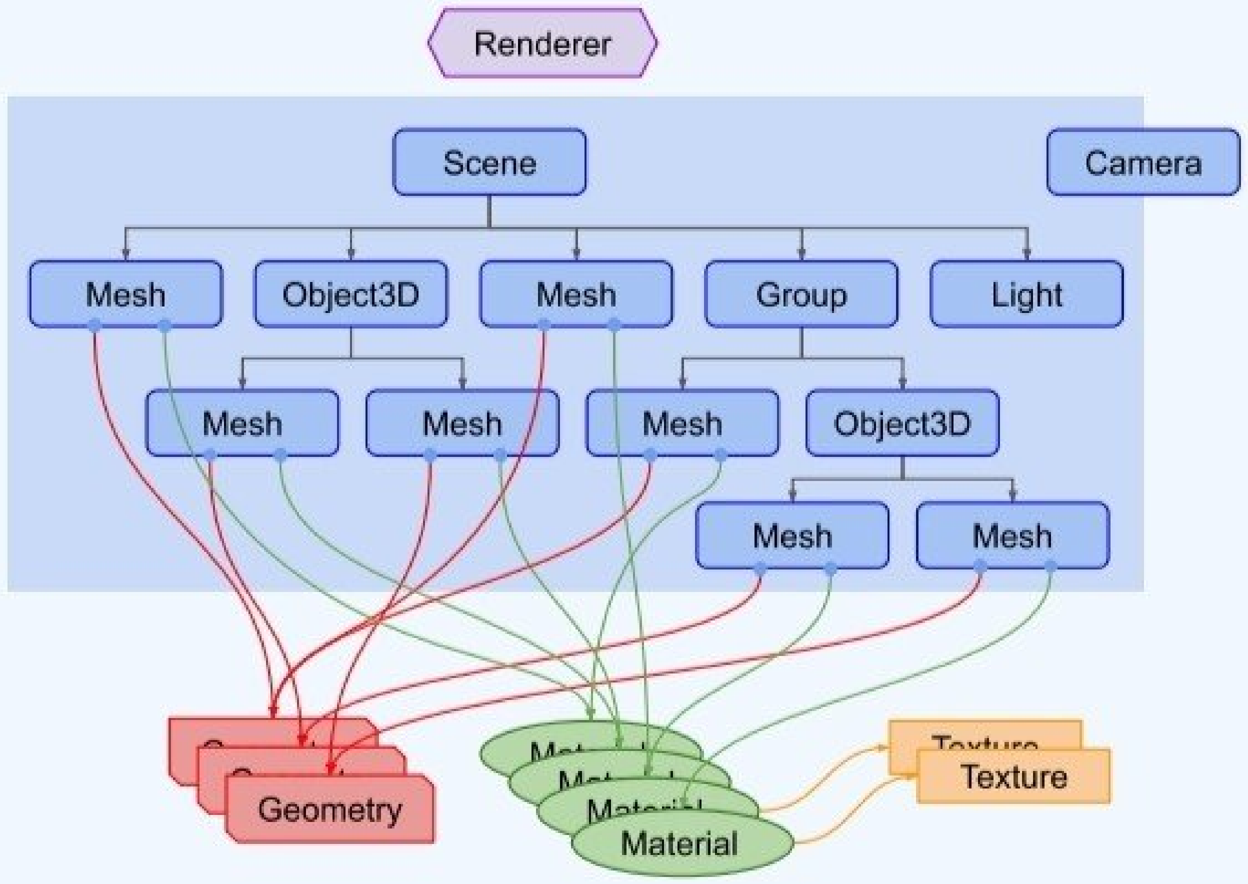
With a vibrant and supportive community of developers and designers, Three.js is constantly evolving and improving, making it an exciting and dynamic technology to explore and work with.

Whether you're a seasoned 3D programmer or just starting out, It offers a world of creative possibilities for building stunning, interactive web experiences.

Fundamentals

Before we get started let's try to give you an idea of the structure of a three.js app.

A three.js app requires you to create a bunch of objects and connect them together. Here's a diagram that represents a small three.js app



Fundamentals

Things to notice about the diagram above.

There is a Renderer. This is arguably the main object of three.js. You pass a Scene and a Camera to a Renderer and it renders (draws) the portion of the 3D scene that is inside the frustum of the camera as a 2D image to a canvas.

That was quite difficult, don't you think?

Let me break it down for you in simpler terms with a brief explanation or introduction of each term.

What is:

Renderer

A renderer is responsible for drawing the 3D scene onto the web page. In Three.js, the WebGLRenderer class is used to create a renderer. It uses WebGL, a graphics API based on OpenGL ES, to interact with the GPU and draw the scene onto the web page.

Geometry

Geometry defines the shape and structure of an object in Three.js.
It is made up of vertices (points in 3D space) and faces (triangles that connect the vertices). Three.js provides a number of built-in geometries, such as BoxGeometry, SphereGeometry, and PlaneGeometry, as well as the ability to create custom geometries.

What is:

Light

Lighting is used to simulate the way light interacts with objects in the scene. In Three.js, lights are used to illuminate the scene and create shadows. Three.js provides a number of built-in lights, such as AmbientLight, DirectionalLight, and PointLight, as well as the ability to create custom lights.

Camera

A camera determines the perspective and position of the viewer in the scene. The PerspectiveCamera and OrthographicCamera classes are used to create cameras. The PerspectiveCamera simulates a perspective view, while the OrthographicCamera simulates an isometric view.

What is:

Material

Material defines how an object appears in the scene, including its color, texture, and shading. The materials are applied to geometries to define their appearance.

It provides a number of built-in materials, such as MeshBasicMaterial, MeshLambertMaterial, and MeshPhongMaterial, as well as the ability to create custom materials using shaders.

Scene

A scene is the container that holds all of the objects, lights, and cameras in Three.js. In order to render anything in Three.js, you must create a scene and add objects, lights, and cameras to it.

What is:

Texture

A texture is an image applied to a material in Three.js.

Textures can be used to add detail to an object's surface, such as a wood grain pattern or a marble texture. In Three.js, textures are loaded using the TextureLoader class and applied to materials using the texture property.

Animation

Animation is the process of creating movement or change in a 3D scene over time. In Three.js, animation is achieved using the requestAnimationFrame method to update the position, rotation, or scale of objects in the scene.

Summary

In summary, Three.js provides a number of powerful tools for creating & rendering 3D graphics on the web.

By understanding the basics of concepts such as renderer, geometry, texture, material, camera, light, scene, and animation, you can begin creating your own 3D scenes and exploring more advanced features in Three.js.

React Three Fiber

React Three Fiber is a library that simplifies building 3D applications with Three.js and React. It provides a declarative and component-based approach to building and managing 3D scenes.

By using React Three Fiber, developers can use familiar React patterns to build and maintain complex 3D applications with less code and fewer bugs.

The library also provides performance optimizations that ensure smooth performance, even with complex scenes.

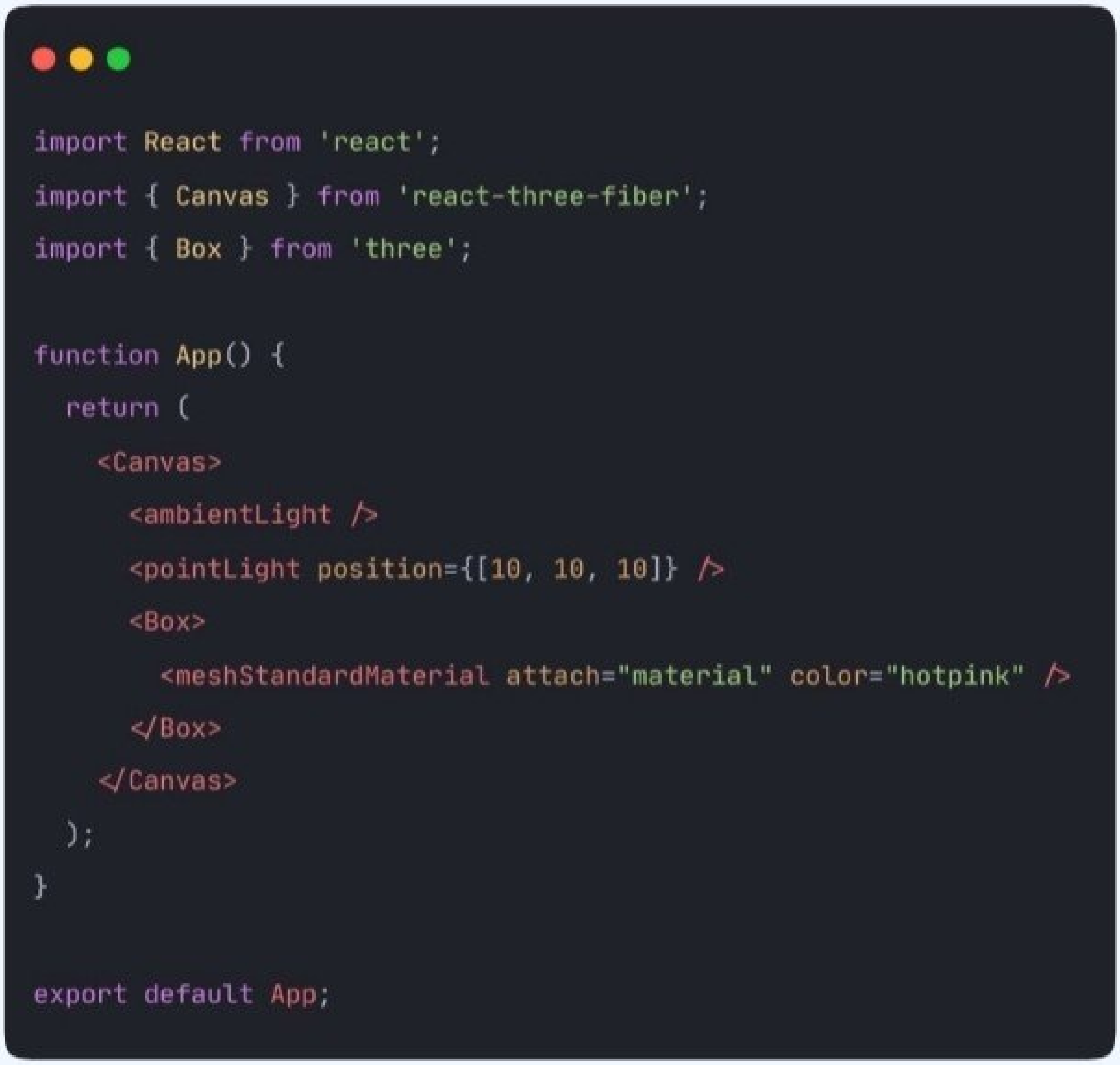
React Three Fiber's declarative and component-based approach can make it easier to reason about and manage the state and lifecycle of 3D objects in the scene, resulting in a more maintainable and scalable codebase.

React JS Setup

1. Create a new React project using a tool like Create React App.
2. Install the required dependencies using npm or yarn. These include **react-three-fiber**, three, and @types/three (if you are using TypeScript).
3. In your App.js or index.js file, import the necessary components from react-three-fiber and three.
4. Set up a basic Three.js scene using the Canvas component from react-three-fiber.
5. Add objects to the scene using Three.js primitives or custom 3D models.
6. Use react-three-fiber hooks like useFrame or useLoader to interact with the Three.js scene and update it as needed.

- *Here's an example of what your App.js file might look like →*

React JS Setup

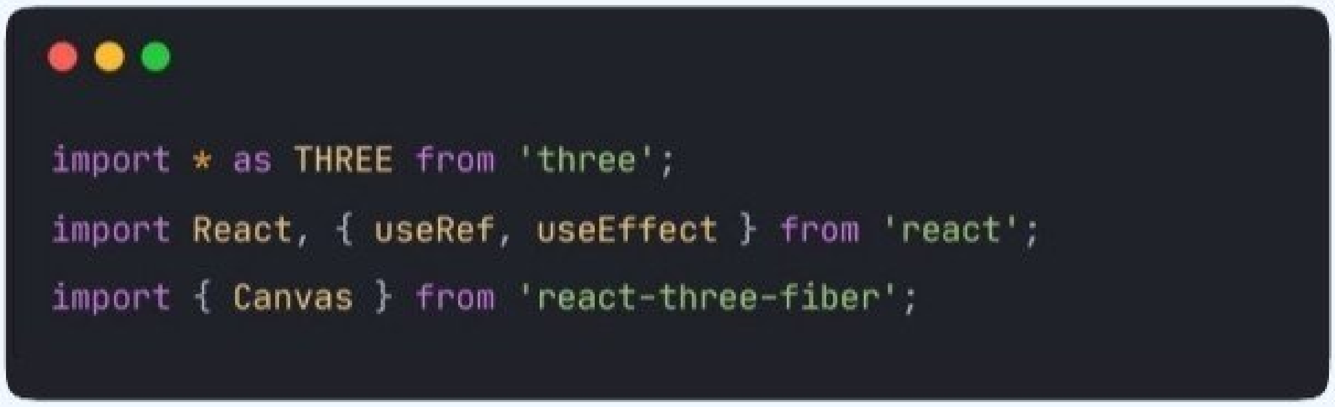


```
import React from 'react';
import { Canvas } from 'react-three-fiber';
import { Box } from 'three';

function App() {
  return (
    <Canvas>
      <ambientLight />
      <pointLight position={[10, 10, 10]} />
      <Box>
        <meshStandardMaterial attach="material" color="hotpink" />
      </Box>
    </Canvas>
  );
}

export default App;
```

This sets up a basic scene with ambient and point lighting, and a hot pink box in the center of the scene.

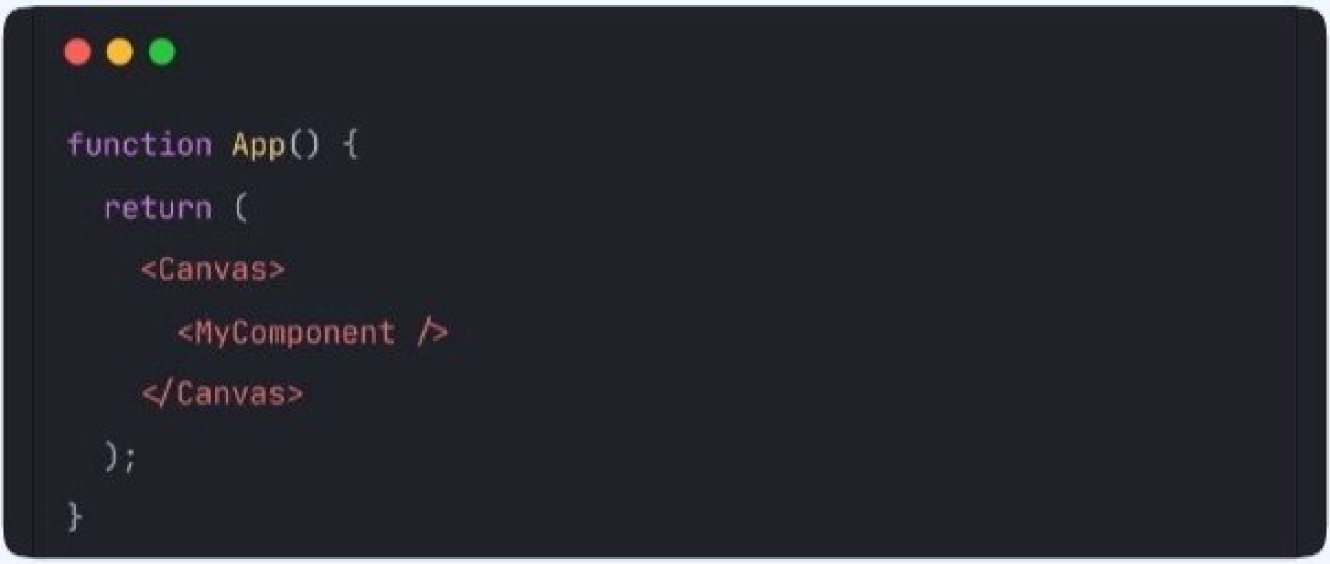
Cheat Sheet

1. Importing Three.js and React libraries



```
import * as THREE from 'three';
import React, { useRef, useEffect } from 'react';
import { Canvas } from 'react-three-fiber';
```

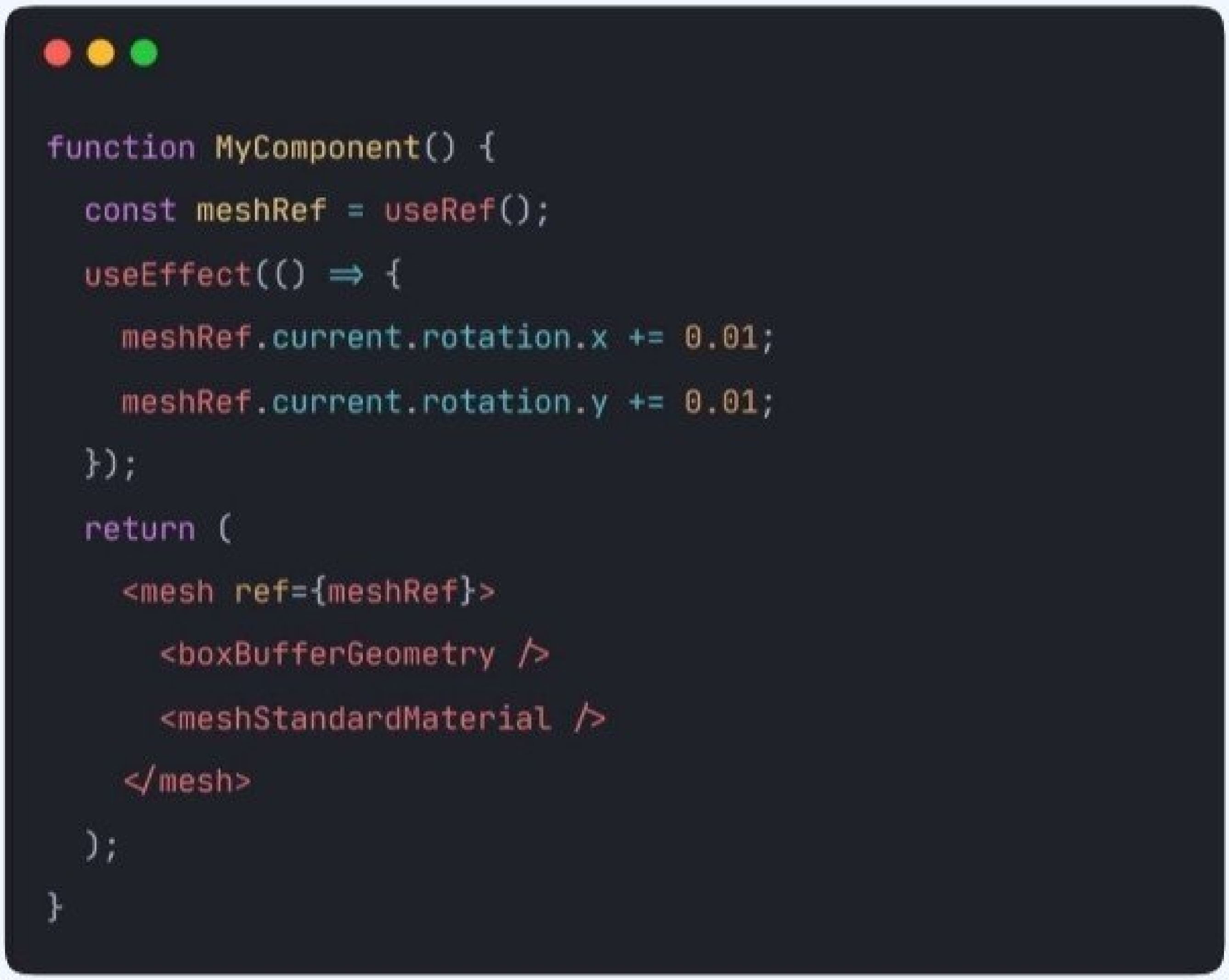
2. Rendering the Three.js component using the Canvas component from react-three-fiber



```
function App() {
  return (
    <Canvas>
      <MyComponent />
    </Canvas>
  );
}
```

Cheat Sheet

3. Creating a Three.js component using React hooks



```
function MyComponent() {
  const meshRef = useRef();
  useEffect(() => {
    meshRef.current.rotation.x += 0.01;
    meshRef.current.rotation.y += 0.01;
  });
  return (
    <mesh ref={meshRef}>
      <boxBufferGeometry />
      <meshStandardMaterial />
    </mesh>
  );
}
```

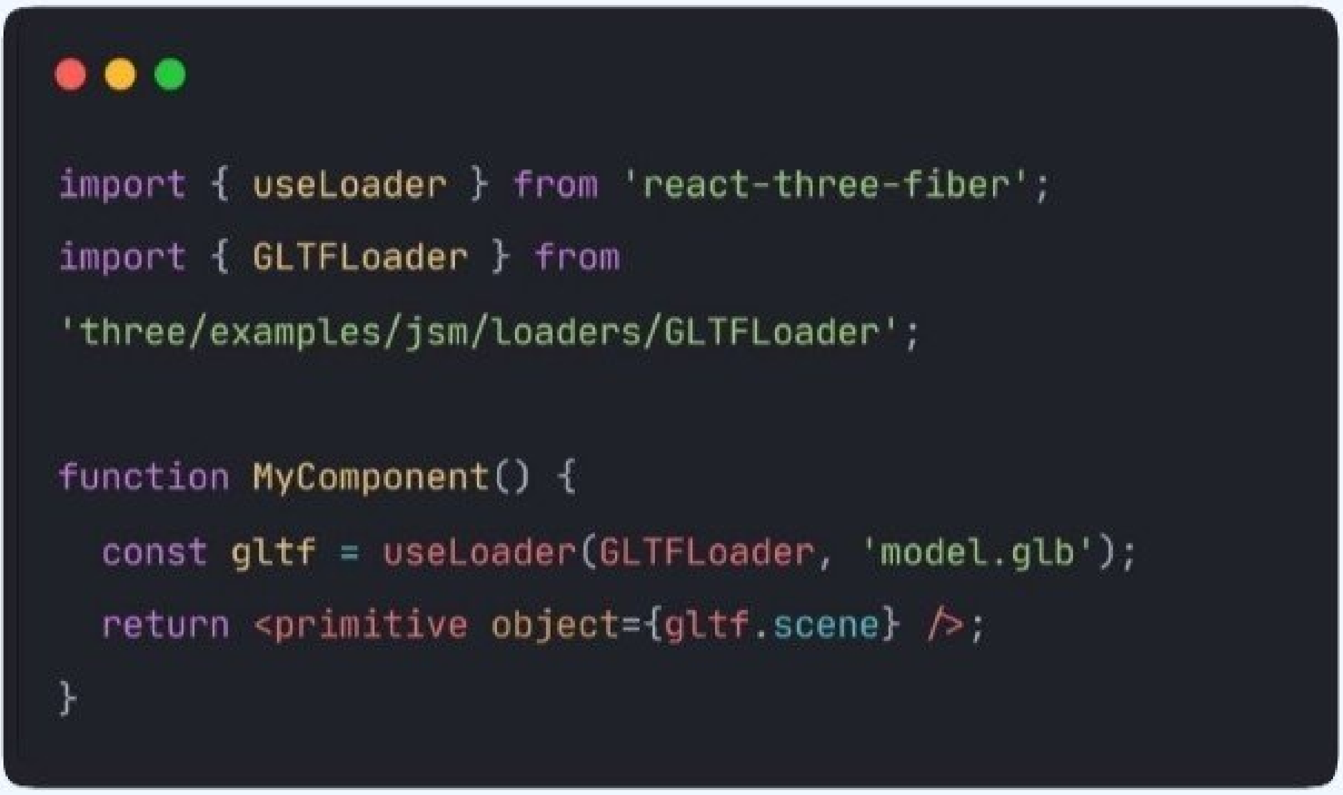
Cheat Sheet

4. Adding interactivity with the mouse using the useThree and useFrame hooks from react-three-fiber

```
import { useThree, useFrame } from 'react-three-fiber';

function MyComponent() {
  const meshRef = useRef();
  const { mouse } = useThree();
  useFrame(() => {
    meshRef.current.rotation.x = mouse.y * Math.PI;
    meshRef.current.rotation.y = mouse.x * Math.PI;
  });
  return (
    <mesh ref={meshRef}>
      <boxBufferGeometry />
      <meshStandardMaterial />
    </mesh>
  );
}
```

Cheat Sheet

5. Loading a 3D model using the useLoader hook from react-three-fiber



```
import { useLoader } from 'react-three-fiber';
import { GLTFLoader } from
'three/examples/jsm/loaders/GLTFLoader';


function MyComponent() {
  const gltf = useLoader(GLTFLoader, 'model.glb');
  return <primitive object={gltf.scene} />;
}
```

I hope this cheatsheet helps you get started with using Three.js in React!

Why react-three-fiber?

An example of a challenging aspect of building a 3D application with pure Three.js is managing the state and lifecycle of 3D objects in the scene.

Three.js provides a lot of low-level control over the 3D objects in the scene, such as the ability to add, remove, and update objects individually, but this can quickly become complex and hard to manage as the scene grows in complexity.

For example, imagine a 3D product website that allows users to customize and interact with a 3D model of a product.

The model may have multiple parts, each with its own material and texture, and the user may be able to change the color or texture of each part.

Why react-three-fiber?

Managing the state of all these objects in the scene, updating them in response to user input, and ensuring that they render correctly can be challenging.

Three.js provides a lot of low-level control over the 3D objects in the scene, such as the ability to add, remove, and update objects individually, but this can quickly become complex and hard to manage as the scene grows in complexity.

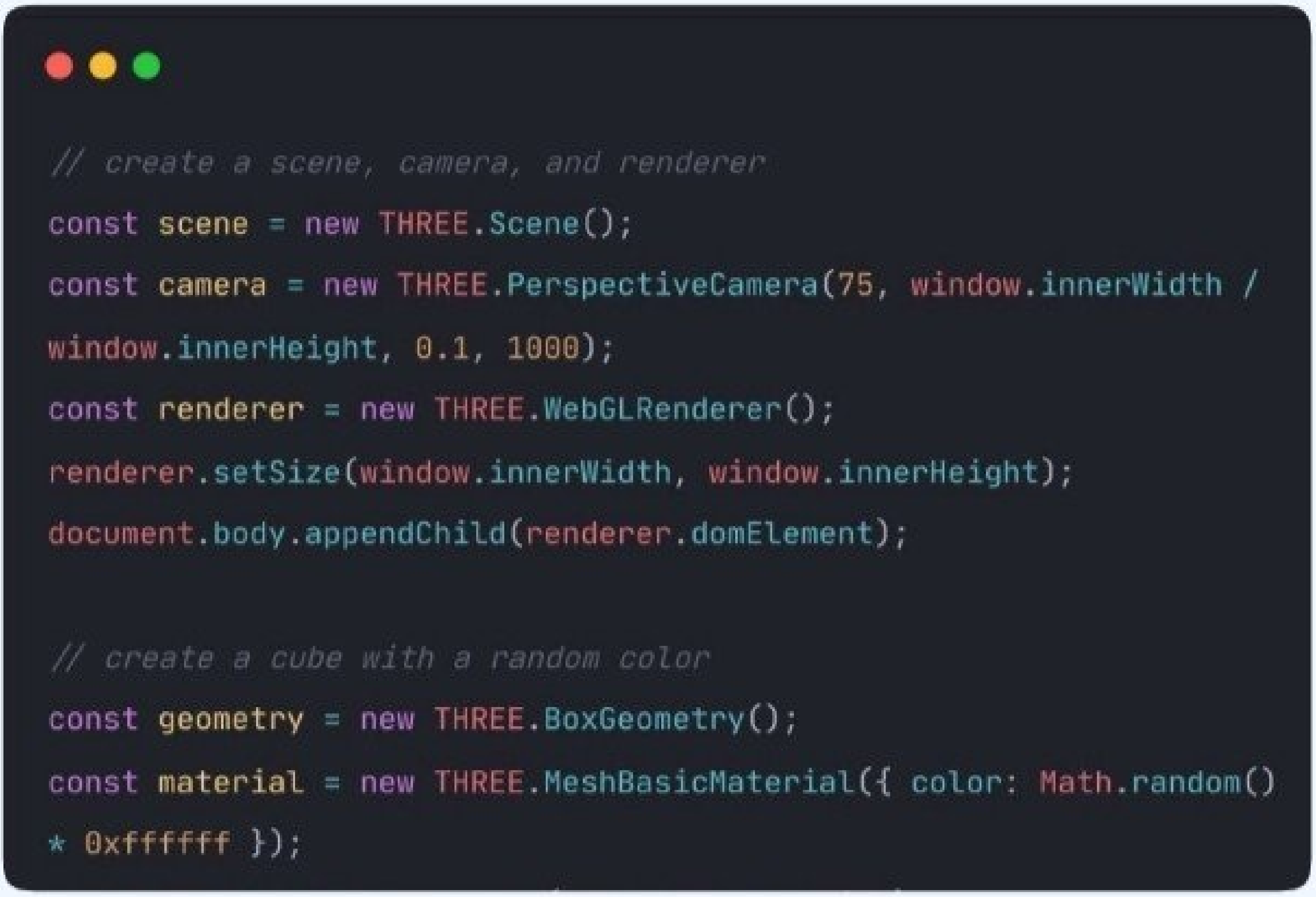
React Three Fiber was built to address this and other challenges by providing a declarative and component-based approach to building 3D apps.

With React Three Fiber, developers can use familiar React patterns to manage the state and lifecycle of 3D objects in the scene, making it easier to build and maintain complex 3D applications.

Code Example

Here's an example of how managing state in Three.js can become complex and how React Three Fiber simplifies the process:

Let's say we want to create a 3D cube that changes color when clicked on. Here's how we might do it in pure Three.js:



```
// create a scene, camera, and renderer
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

// create a cube with a random color
const geometry = new THREE.BoxGeometry();
const material = new THREE.MeshBasicMaterial({ color: Math.random()
* 0xffffff });
```

Code Example

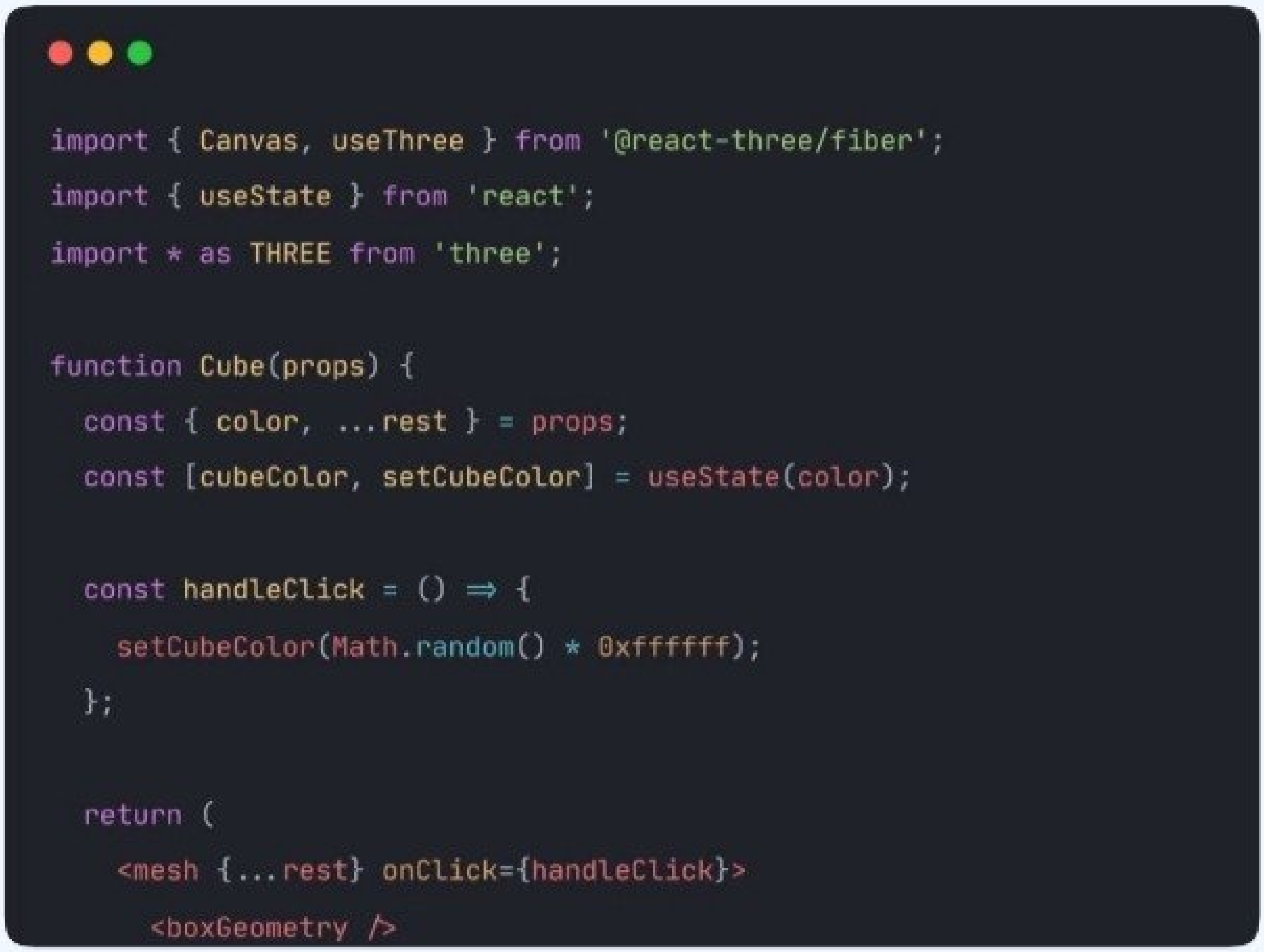
```javascript
const material = new THREE.MeshBasicMaterial({ color: Math.random()
* 0xffffff });
const cube = new THREE.Mesh(geometry, material);
scene.add(cube);

// add a click event listener to the cube to change its color
cube.addEventListener('click', () => {
  cube.material.color.setHex(Math.random() * 0xffffff);
});

// render the scene
function animate() {
  requestAnimationFrame(animate);
  renderer.render(scene, camera);
}
animate();
```

This code creates a cube with a random color and adds a click event listener to it to change its color when clicked on.

Code Example

However, as the scene grows in complexity and more objects are added, managing the state and lifecycle of these objects can become challenging.

Here's how we could accomplish the same thing with React Three Fiber:



```
import { Canvas, useThree } from '@react-three/fiber';
import { useState } from 'react';
import * as THREE from 'three';

function Cube(props) {
  const { color, ...rest } = props;
  const [cubeColor, setCubeColor] = useState(color);

  const handleClick = () => {
    setCubeColor(Math.random() * 0xffffff);
  };

  return (
    <mesh {...rest} onClick={handleClick}>
      <boxGeometry />
```

Code Example

```jsx
  const [cubeColor, setCubeColor] = useState(color);

  const handleClick = () => {
    setCubeColor(Math.random() * 0xffffff);
  };

  return (
    <mesh {...rest} onClick={handleClick}>
      <boxGeometry />
      <meshBasicMaterial color={cubeColor} />
    </mesh>
  );
}

function Scene() {
  return (
    <Canvas>
      <Cube position={[-1, 0, 0]} color={Math.random() * 0xffffff} />
      <Cube position={[1, 0, 0]} color={Math.random() * 0xffffff} />
    </Canvas>
  );
}

function App() {
  return <Scene />;
}
```

Code Example

In this code, we define a Cube component that manages its own state using the useState hook.

The Cube component renders a Three.js mesh and material, and responds to click events to change its own color.

We then use the Cube component twice in the Scene component to create two cubes with different positions and random colors.

By using React Three Fiber's declarative and component-based approach, we can easily manage the state & lifecycle of multiple 3D objects in the scene

RTF Advantages over Pure

Declarative approach:

React Three Fiber uses a declarative approach to define 3D scenes, which can make it easier to reason about and maintain the codebase over time. This is because the code describes what should be displayed, rather than how it should be displayed.

Familiar syntax:

React Three Fiber uses a syntax that is similar to that of React, which makes it easier for developers who are familiar with React to learn and use Three.js.

Component-based architecture:

It uses React's component-based architecture, which can make it easier to organize and manage the state and lifecycle of 3D objects in a scene.

RTF Advantages over Pure

Optimized for React:

React Three Fiber is optimized for React's rendering engine and lifecycle methods, which means that it can be used in large, complex applications without any performance issues.

Debugging tools:

It provides debugging tools such as a helpful error message and a built-in inspector, which can make it easier to identify and fix issues in the 3D scene.

TypeScript support:

React Three Fiber has excellent TypeScript support, which means that you can write type-safe Three.js code in React.

RTF Advantages over Pure

Hooks-based API:

React Three Fiber's hooks-based API makes it easy to manipulate Three.js objects, animate them, and add event listeners to them.

Drei library:

Drei is a collection of useful Three.js components and helpers that are built on top of React Three Fiber. Drei components make it easy to create things like 3D text, post-processing effects, and particle systems.

Overall, React Three Fiber offers a more streamlined and efficient way to work with Three.js in React, and is a great choice for developers who want to build 3D applications in React.

RTF Cheat Sheet

`<Canvas>`

A component that creates a Three.js scene and mounts it to the DOM.

`<mesh>`

A component that wraps a 3D object and its material.

`<primitive>`

A component that creates a Three.js primitive geometry, like a box or sphere.

`<group>`

A component that allows you to group objects together for easier manipulation.

RTF Cheat Sheet

`useLoader()`

A hook that loads external assets, like textures or 3D models.

`useFrame()`

A hook that runs on every frame of the animation loop, allowing you to update Three.js objects over time.

`useGraph()`

Convenience hook which creates a memoized, named object/material collection from any Object 3D.

RTF Cheat Sheet

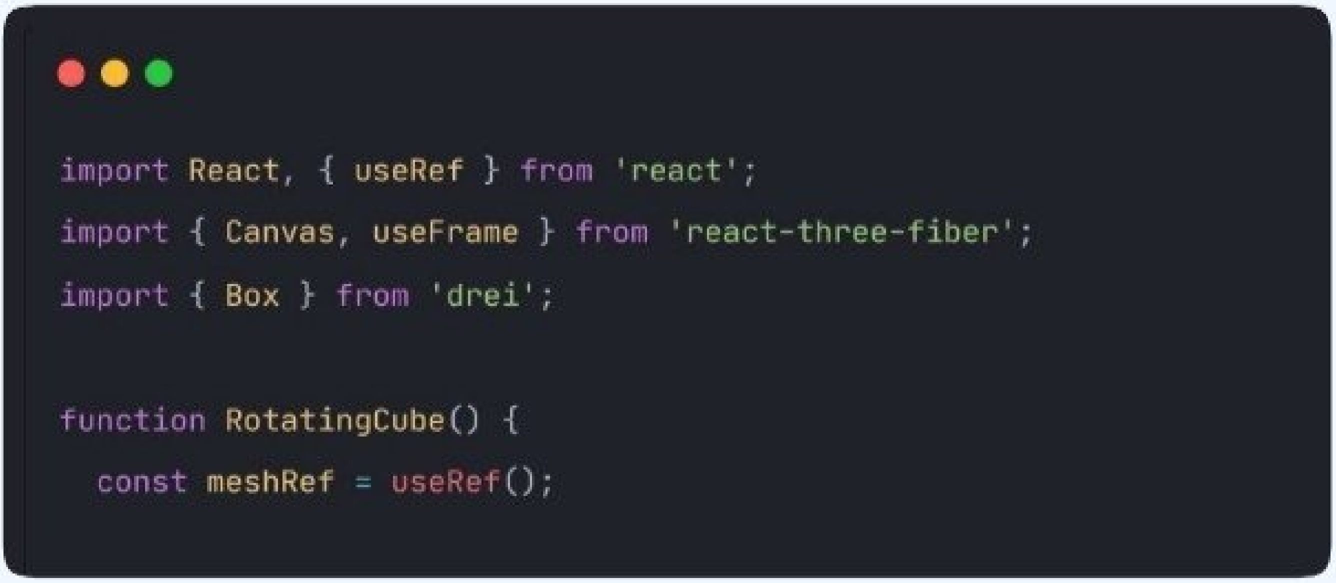
`<OrbitControls>`

A component that adds mouse and touch controls to your Three.js scene.

`<perspectiveCamera>`

A component that defines a perspective camera for your scene.

Here's an example of using some of these features together to create a rotating cube:



```
import React, { useRef } from 'react';
import { Canvas, useFrame } from 'react-three-fiber';
import { Box } from 'drei';

function RotatingCube() {
  const meshRef = useRef();
```

RTF Cheat Sheet

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

  return (
    <Box ref={meshRef}>
      <meshStandardMaterial attach="material" color="hotpink" />
    </Box>
  );
}

function App() {
  return (
    <Canvas>
      <ambientLight />
      <pointLight position={[10, 10, 10]} />
      <RotatingCube />
    </Canvas>
  );
}

export default App;
```

RTF Cheat Sheet

This creates a **`<Canvas>`** with ambient and point lighting, and a **`<RotatingCube>`** component that uses **`useFrame()`** to rotate the cube on every frame.

The cube is created using the **`<Box>`** primitive, and its reference is stored in meshRef. Finally, the **`<Box>`** is wrapped in a **`<mesh>`** with a hot pink material.

React Three Drei?

React Three Drei is a collection of useful helper components and hooks for building 3D applications with React Three Fiber.

It is built on top of React Three Fiber and provides a higher-level API for common 3D tasks such as camera controls, lighting, and loading 3D models.

React Three Drei offers a variety of pre-built components such as OrbitControls, Sky, Html, Model, shaderMaterial, Reflector and Bloom that can be easily used in a React Three Fiber scene.

These components can help streamline the process of building 3D applications by abstracting away low-level Three.js code and providing a simpler and more intuitive interface.

React Three Drei?

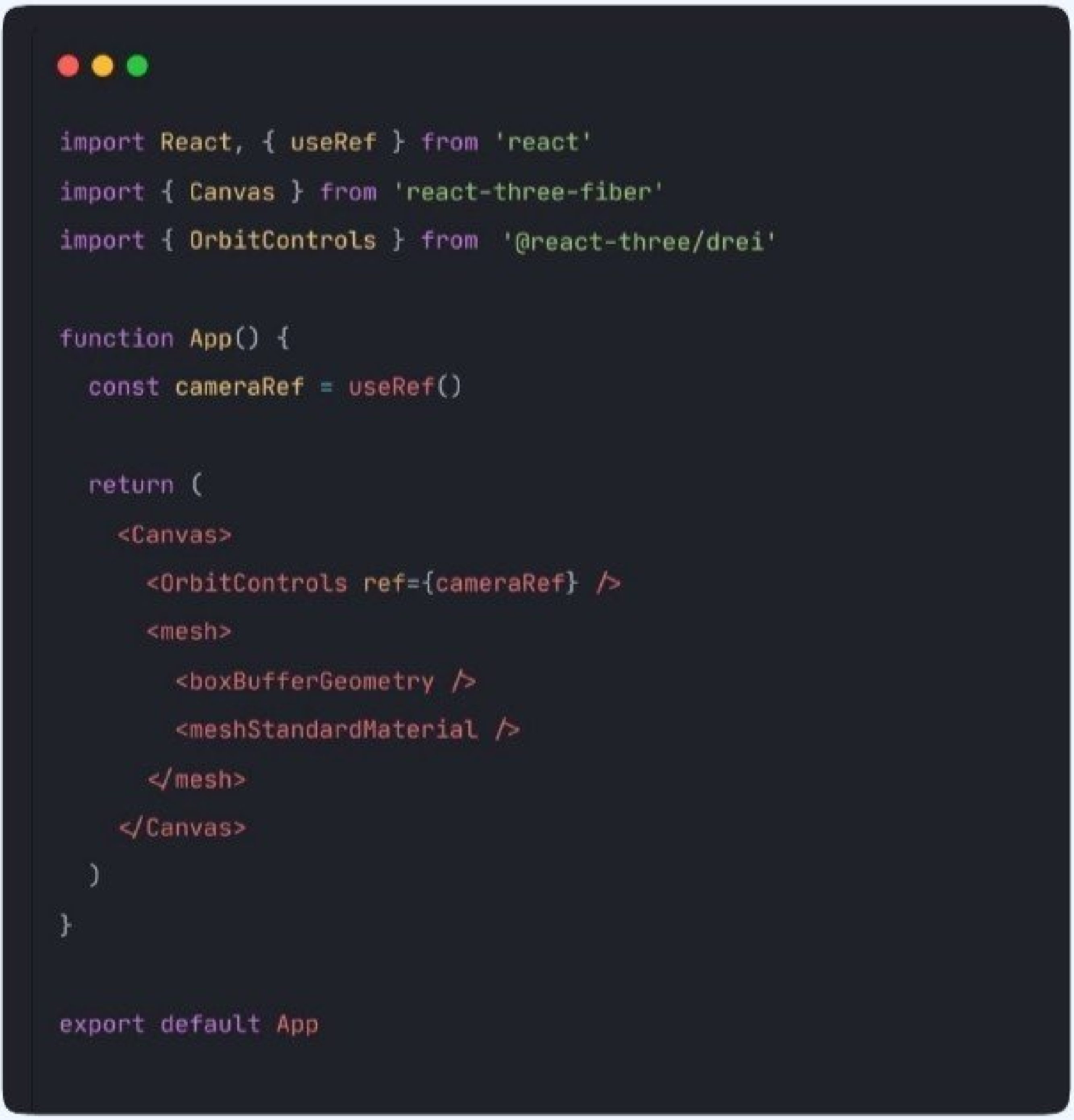
React Three Drei also includes a number of hooks, such as useTexture, useGLTF, and useAnimations, that make it easier to work with assets in a 3D scene.

Overall, React Three Drei can help developers save time and effort when building 3D applications by providing pre-built components and hooks that abstract away low-level Three.js code and simplify common 3D tasks.

Here are some code examples of React Three Drei components and hooks →

RTD Code Example

1. OrbitControls component:



```
import React, { useRef } from 'react'
import { Canvas } from 'react-three-fiber'
import { OrbitControls } from '@react-three/drei'

function App() {
  const cameraRef = useRef()

  return (
    <Canvas>
      <OrbitControls ref={cameraRef} />
      <mesh>
        <boxBufferGeometry />
        <meshStandardMaterial />
      </mesh>
    </Canvas>
  )
}

export default App
```

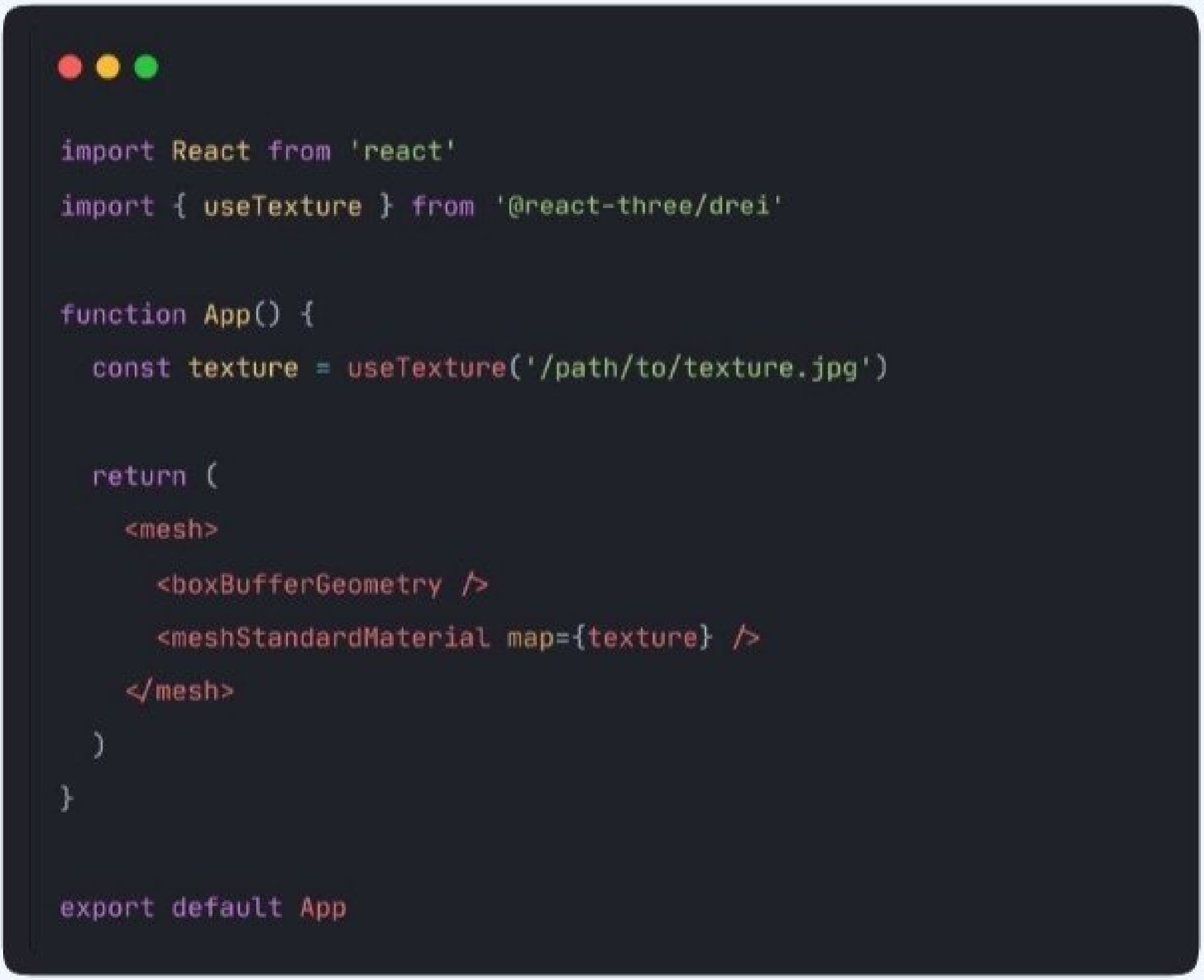
RTD Code Example

This code creates a simple 3D scene with a box mesh and an OrbitControls component for controlling the camera position and rotation.

The OrbitControls component is imported from drei (short for "three"), which is a part of React Three Drei.

RTD Code Example

2. useTexture hook:



```
import React from 'react'
import { useTexture } from '@react-three/drei'

function App() {
  const texture = useTexture('/path/to/texture.jpg')

  return (
    <mesh>
      <boxBufferGeometry />
      <meshStandardMaterial map={texture} />
    </mesh>
  )
}

export default App
```

This code uses the useTexture hook from React Three Drei to load a texture image and apply it to a mesh material.

RTD Code Example

The useTexture hook returns a Texture object, which can be used as the map property of a mesh material.

3. Html component:

```
import React from 'react'
import { Html } from '@react-three/drei'

function App() {
  return (
    <Html>
      <div style={{ color: 'white' }}>Hello, world!</div>
    </Html>
  )
}

export default App
```

RTD Code Example

This code uses the Html component from React Three Drei to render a HTML element in the 3D scene.

The Html component creates a separate HTML layer that is rendered on top of the 3D scene, allowing developers to easily create interactive and dynamic user interfaces in their 3D applications.

These are just a few examples of the many components and hooks provided by **React Three Drei**.

By using these pre-built components and hooks, developers can simplify common 3D tasks and create more complex 3D applications with less code.

Advantages

Simplified API:

React Three Drei provides a higher-level API for common 3D tasks, which can make it easier and faster to build 3D scenes compared to writing raw Three.js code.

Component-based architecture:

Drei uses React's component-based architecture, which makes it easier to organize and manage the state and lifecycle of 3D objects in a scene.

Performance optimizations:

React Three Drei includes performance optimizations such as automatic batching of meshes and pre-loading of assets, which can help improve the overall performance of a 3D application.

Advantages

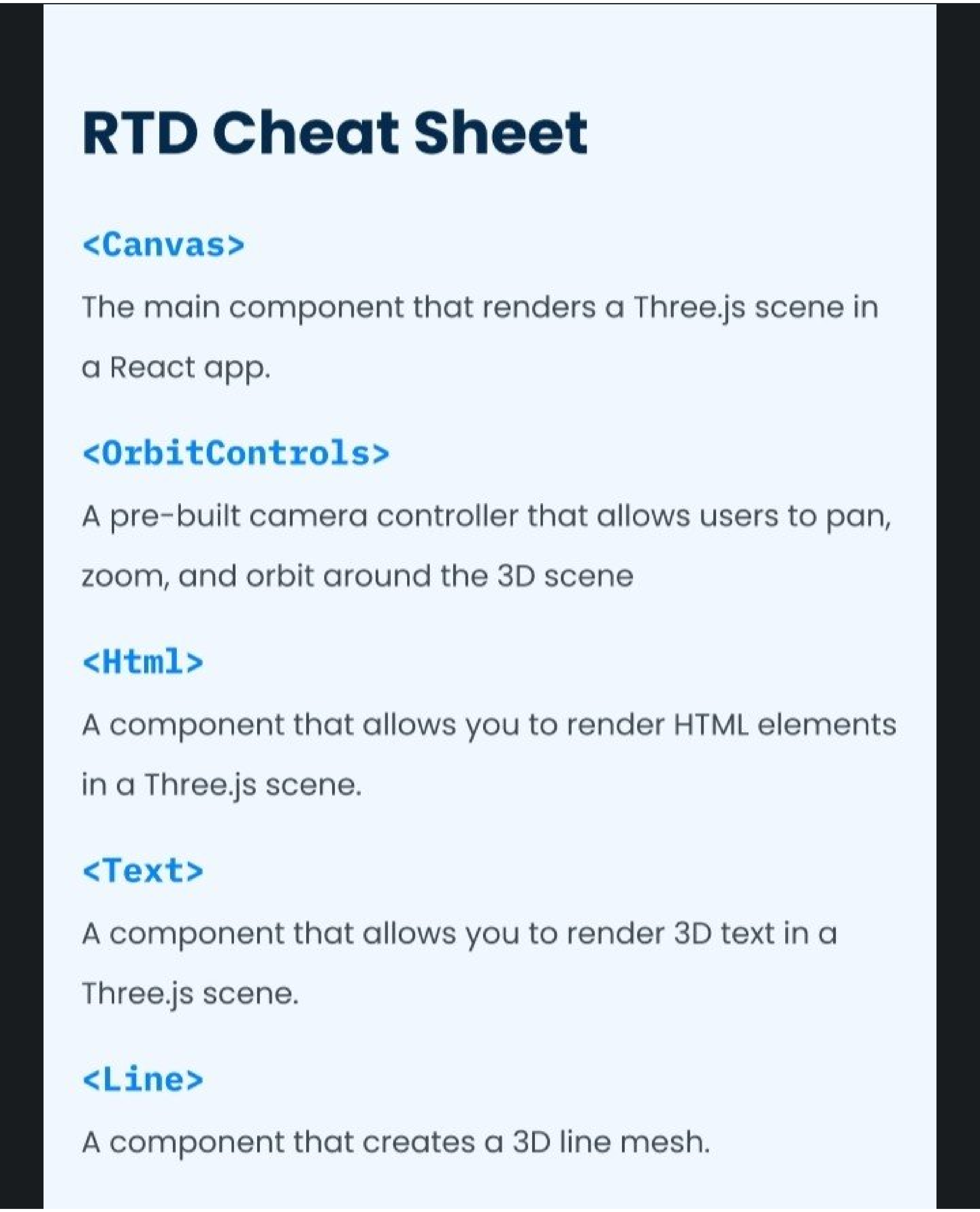
Developer-friendly:

React Three Drei can make it easier for developers to work with Three.js by providing a more familiar and developer-friendly syntax, especially for those who are already familiar with React.

Code reusability:

React Three Drei over pure Three.js is that it can help reduce code complexity & increase code reusability.

Overall, React Three Drei can make it easier and faster to build 3D applications with Three.js by providing a simpler and more intuitive interface, performance optimizations, and a component-based architecture.



RTD Cheat Sheet

`<Canvas>`

The main component that renders a Three.js scene in a React app.

`<OrbitControls>`

A pre-built camera controller that allows users to pan, zoom, and orbit around the 3D scene

`<Html>`

A component that allows you to render HTML elements in a Three.js scene.

`<Text>`

A component that allows you to render 3D text in a Three.js scene.

`<Line>`

A component that creates a 3D line mesh.

RTD Cheat Sheet

`<Canvas>`

The main component that renders a Three.js scene in a React app.

`<OrbitControls>`

A pre-built camera controller that allows users to pan, zoom, and orbit around the 3D scene

`<Html>`

A component that allows you to render HTML elements in a Three.js scene.

`<Text>`

A component that allows you to render 3D text in a Three.js scene.

`<Line>`

A component that creates a 3D line mesh.

RTD Cheat Sheet

`<Box>`

A component that creates a 3D box mesh.

`<Sphere>`

A component that creates a 3D sphere mesh.

`<Plane>`

A component that creates a 3D plane mesh.

`useTexture`

A hook that loads a texture and returns a Three.js texture object.

`useGLTF`

A hook that loads a GLTF model and returns a Three.js object.

RTD Cheat Sheet

These are just a few examples of the many components and hooks available in React Three Drei.

The library provides a wide range of pre-built components and hooks that can simplify common 3D tasks and save developers time and effort.

Beginner Project Ideas

3D Cube:

Create a 3D cube using Three.js and experiment with different materials, textures, and lighting effects to make it visually appealing.

Solar System Model:

Build a model of the solar system using Three.js and explore the different planets, moons, and other celestial bodies in a 3D environment.

3D Text:

Create 3D text using Three.js and experiment with different fonts, sizes, and colors to make it visually interesting.

Beginner Project Ideas

Interactive Gallery:

Build an interactive gallery using Three.js that allows users to navigate through different 3D objects or images.

Particle Effects:

Create particle effects using Three.js and experiment with different settings to make visually appealing effects, such as explosions, fire, or rain.

3D Terrain:

Create a 3D terrain using Three.js and experiment with different textures, heights, and shapes to create a dynamic landscape.

Beginner Project Ideas

3D Maze:

Build a 3D maze using Three.js and add interactive elements such as obstacles and rewards to make it more challenging and engaging.

3D Card Flip:

Create a simple card-flipping animation using Three.js to showcase your understanding of basic 3D transformations and animations.

3D Interactive Dice:

Build a 3D dice that users can roll and interact with using Three.js, using basic geometry and materials to create a realistic effect.

Topics you should explore

Once you have a good understanding of the fundamentals of Three.js, there are several other topics that you can explore to further develop your skills and create more complex 3D applications. Here are some examples:

Animation

Learn how to use Three.js to create animations that change the position, rotation, and scale of objects over time.

Physics

Learn how to use a physics engine such as Cannon.js or Ammo.js with Three.js to create realistic simulations of objects that interact with each other based on real-world physics principles.

Topics you should explore

Shaders

Learn how to use custom shaders to create complex visual effects and apply advanced lighting and shading techniques to your 3D objects.

Textures

Learn how to use textures to add more detail and visual interest to your 3D objects, including applying images, videos, and other media to surfaces.

Optimization

Learn how to optimize your Three.js applications for performance by reducing the number of objects, minimizing the size of textures and meshes, and using techniques such as culling and LOD (Level of Detail) to improve rendering speed.

Topics you should explore

Interactivity

Learn how to use Three.js to create interactive 3D applications that allow users to interact with objects using mouse or touch events, or even using VR/AR devices.

These are just a few examples of the topics that you can explore after learning the basics of Three.js.

The key is to keep practicing and experimenting with different techniques to improve your skills and create more advanced 3D applications.