Three Js React

npm install three @react-three/fiber

npm install @react-three/drei

npm install @react-three/postprocessing

Versions for the following example:

"@react-three/drei": "^9.19.5",

"@react-three/fiber": "^8.2.2",

"@react-three/postprocessing": "^2.5.5",

Setup:-

After installing the above three packages on the page you want 3d/2d images generated by three js to be shown , create a Canvas tag <Canvas></Canvas>. Whatever you write inside the Canvas tag the browser will know that it is to be rednered / or is a part of Three js or three fiber.

Note to remeber:- NO HTML TAG WORK INSIDE THE CANVAS TAG. IF FOR SOME REASON YOU WANT TO USE DIV OR H1 SORT OF TAGS THEY SHOULD BE PLACED INSIDE A <Html></Html> tag.

Inside the cnavas tag the first tag you need to add is a tag called Suspense. The Suspense tag waits till the body/images are loaded and till the images are loaded will show a fallback code / place using fallback tag

<Supsense fallback={null}></Suspense>

here fallback={null} states that till the images are loaded show nothing or null.

You can create your own fallback using the html tag and pass that to this null.

To create any 3D images or models a mesh is required, as mesh tells where the pollygons or vertices of the model will be placed.

<mesh></mesh>

This setup reamins same irrespective of what model you want to create. The tags are case sensetive.

Setup should look something like this:

<Canvas>

<Suspense fallback={null}>

<mesh>

{/\* 3D model code}

</mesh>

</Suspense>

</Canvas>

Making 3d Objects:

All codes will be between mesh tags

Creating a Sphere:-

Code to create a simple sphere

Create a simple sphere using the <sphereGeometry> tag. This tag can take arguments of radius, widthSegments, heightSegments, phuStart, phiLength, thetaStart and thetaLength.

WidthSegmets and heightSegments make the sphere round, as the plane we have is 2d width and height to create a round object widthsegment and height segment is required.

Philength:- Philength gives you the horizontal sweep, in other words this is what completes your sphere lower the lenght the incomplete your spere be. (can be used to make semi circles)

ThetaLength: Does the same on vertical axis, lower the value will create something like a bowl.

By default philength is pi\*2 and thetalenght = pi

Phistart and thetastart determines the start angle.

Code:-

<sphereGeometry args={[0.5,32,32]}/>

args={[radius, widthsegment, heightsegment]}

To see how these values affect the sphere: [https://threejs.org/docs/index.html?q=sphe#api/en/geometries/SphereGeometry](https://threejs.org/docs/index.html?q=sphe" \l "api/en/geometries/SphereGeometry)

by default phistart= 0 , thetastart=0 .

Whne you add this code a=the browser should show u a black circle. Lets add a color to the sphere.

For adding a color the thing need to be specified is the material of the sphere.For this example we will use <meshStandardMaterial> which is a physically based material

documentation:[https://threejs.org/docs/index.html?q=mesh#api/en/materials/MeshStandardMaterial](https://threejs.org/docs/index.html?q=mesh" \l "api/en/materials/MeshStandardMaterial)

<meshStandardMaterial color="#FF0000" />

after this your code will look something like this:

<mesh >

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" />

</mesh>

After running this you will see a black circle on the browser. This is because you have not added light to scene. An external light source needs to be added to light up the scene.

Various light sources can be added but for this example we will use and ambient light. Ambient light is basically like the sunlight spreads everywhere.

<ambientLight args={["#fffffff", 1]} />

this gives the light a white color and the intensity of 1 , play around with this to see how it changes the lightning.

Code :

<mesh >

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" />

</mesh>

<ambientLight args={["#fffffff", 1]} />

Now you should see a red color circle on the browser.

Which still looks like a 2d image. We will add more things to the scene to make it look more 3d.

We will start by adding a floor to the image. We will do this by using <planeGeometry>

documentation:[https://threejs.org/docs/index.html?q=plane#api/en/geometries/PlaneGeometry](https://threejs.org/docs/index.html?q=plane" \l "api/en/geometries/PlaneGeometry)

This will take width, height, widthsegment and heightsegment.

Here widthsegment and heightsegment will create nodes on the plane , these nodes then can be used to place objects on them. Or animate them.

**CODE:**

<planeGeometry args={[7, 7]} />

Here we have passed a width and height of 7. this we will also pass a material for rendering.And put this in a seperate mesh so that we can change it as we go along.

Code:

<mesh>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" wireframe={true} />

</mesh>

{/\* floor \*/}

<mesh >

<planeGeometry args={[7, 7]} />

<meshStandardMaterial color="#add8e6" />

</mesh>

<ambientLight args={["#fffffff", 1]} />

After this you will still see it as a 2d image as the floor is a vertical square, Next what we nedd to do is to rotate the plane so that it is horizontal or perpendicular to the sphere.We do this by passing a rotation angle which will be in radians, either you can convert online the deg to radians and pass the value or create a function in js

export const angleToRadians = (angleInDeg) => (Math.PI / 180) \* angleInDeg;

we will pass the rotation to the mesh of plane as we want to rotate the mesh that holds the plane.

Code:

import { angleToRadians } from "./angle";

*\*Wrote the function in other file called angle.js \**

<mesh >

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" wireframe={true} />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]}>

<planeGeometry args={[7, 7]} />

<meshStandardMaterial color="#add8e6" />

</mesh>

<ambientLight args={["#fffffff", 1]} />

After this you should not be able to see the plane at all, that is because the camera. The camera right now is facing the screen by default your eyballs act like the camera for this code. To change this react-three/drei have a lot of cameras that can be used . We will use a PrespectiveCamera for this example.

https://github.com/pmndrs/drei#readme

We we will place the camera above the spehre and move the sphere above the plane so it looks like it is placed on the plane.

Code:

<PerspectiveCamera makeDefault position={[0, 1, 5]} />

<mesh position={[0, 0.5, 0]}>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" wireframe={true} />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]}>

<planeGeometry args={[7, 7]} />

<meshStandardMaterial color="#add8e6" />

</mesh>

<ambientLight args={["#fffffff", 1]} />

Note: if you are getting Errors please check the imports: by this point you should have the following imports:

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

import React, { Suspense } from "react";

import { angleToRadians } from "./angle";

On thing you will see is the camera is fixed on looking straight.

Something like this

If we want to angle it we need something called ass orbit control. This will help you move the camera.

<OrbitControls />

This will take control of the defult camera which we made the prespective camera as.

Code:

<PerspectiveCamera makeDefault position={[0, 1, 8]} />

<OrbitControls />

<mesh position={[0, 0.5, 0]}>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]}>

<planeGeometry args={[7, 7]} />

<meshStandardMaterial color="#add8e6" />

</mesh>

<ambientLight args={["#fffffff", 1]} />

Click and move the mouse to see the effect! Where-ever you move the camera will be focused on the ball/sphere.Orbit control sets the polar angle. To understand orbital control imagine a moon orbiting the earth. There the rotation is based on the angle created with the earth and mood, Thus these controls wont be similar to that of prespective camera as it is fixed to a plane of x,y and z axis (0,1,8 in the example defines the x y and z axis ).

To move the image without click and drag we need a reat-three-fiber hook called useFrame. UseFrame give acess to frame you are viewing.

Useframe is to create a render loop, as we want to render each frame to get a smooth motion like a moving picture.

Whatever you write in this it will be rendered 60 times in a second. Lot of parametrs to be passed lets check one called mouse.

<https://docs.pmnd.rs/react-three-fiber/api/hooks>

useFrame((state) => {

console.log(state.mouse);

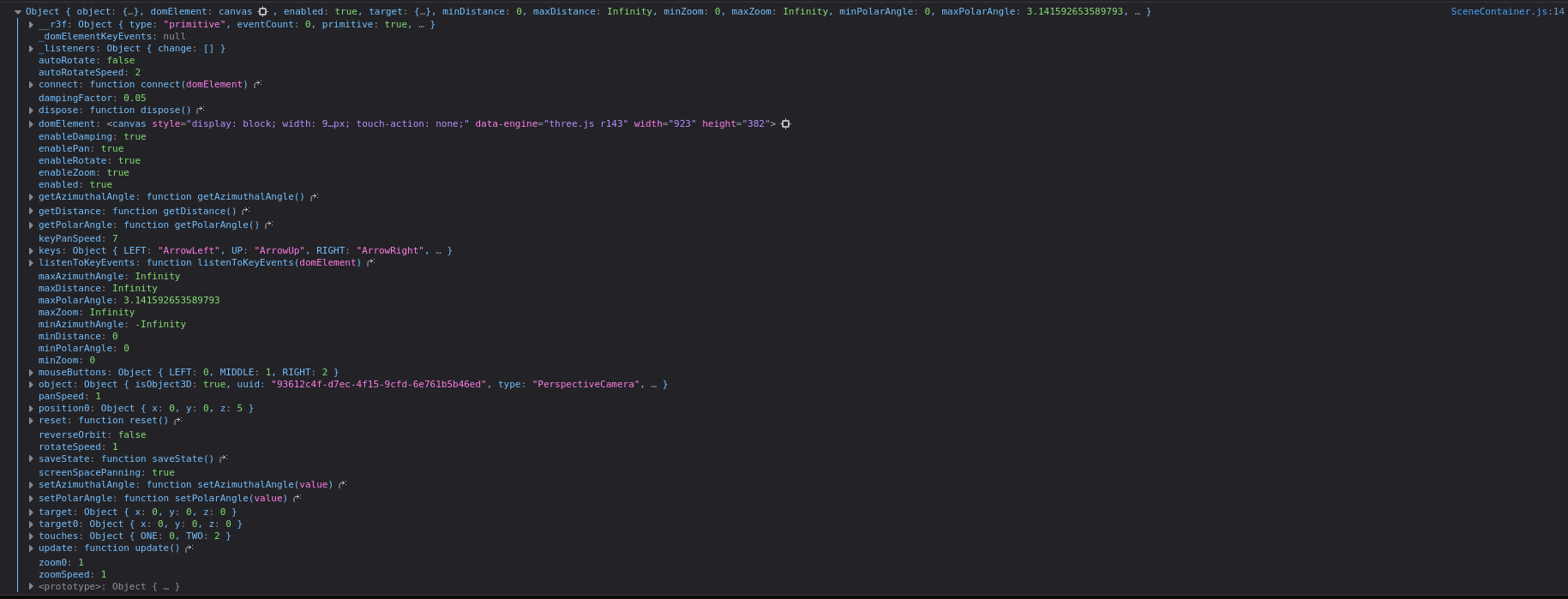
});

You will see vector on mouse move. Giving x and y cordinates these will be normalized . Normalized means values are between -1 to 1 so that value is irrespective of the screen size. This is just to understand the console.log

Lets create a ref

const orbitControlaRef=useRef(null);

Note: useRef hook allows you to keep the values between renders and is used to store a mutable value that does not cause re-render when updated.



Above image is the console .log of the orbitControls Ref. This is what we can use to set .

Press shift and drag mouse to pan the camera.

So lets add some properties of orbitcontrol, To add rotation on mouse movement we have to change setAzimuthalAngle

useFrame((state) => {

if (!!orbitControlsRef.current) {

const { x, y } = state.mouse;

orbitControlsRef.current.setAzimuthalAngle(x\*angleToRadians(45));

orbitControlsRef.current.update();

}

});

Keep it positive for inverted controls like a video game or -angleToRadians for noninverted.

Here we are taking the x vector value and multiplying it by 20 to see a move in camera. Try changing the x \*(value) to see the diffrence . This is how your code should look now:

const SceneContainer = () => {

const orbitControlsRef = useRef(null);

useFrame((state) => {

if (!!orbitControlsRef.current) {

const { x, y } = state.mouse;

orbitControlsRef.current.setAzimuthalAngle(x\*angleToRadians(45));

orbitControlsRef.current.update();

}

});

useEffect(() => {

if (!!orbitControlsRef.current) {

console.log(orbitControlsRef.current);

}

}, [orbitControlsRef.current]);

return (

<Suspense fallback={null}>

<PerspectiveCamera makeDefault position={[0, 1, 8]} />

<OrbitControls ref={orbitControlsRef} />

<mesh position={[0, 0.5, 0]}>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]}>

<planeGeometry args={[7, 7]} />

<meshStandardMaterial color="#add8e6" />

</mesh>

<ambientLight args={["#ffffff", 1]} />

</Suspense>

);

};

export default SceneContainer;

Note: app.js has <SceneContainer/> between the <Canvas> Tags. Check git for the flow of files.

Azimuthal angle starts from origin .

Polarangle starts from the top of the image.

Adding polarangle

orbitControlsRef.current.setPolarAngle((y + 1) \* angleToRadians(90 - 30));

If you just mutiply by y you will see that it wont move and the camera will be placed on top , hence we add +1 to the y . change the (y+(value)) to set the starting point of the camera on the y-axis.

After that we can add max and min polar or azimutahl angle to limit the camera move.

Code :

<OrbitControls

ref={orbitControlsRef}

minPolarAngle={angleToRadians(60)}

maxPolarAngle={angleToRadians(80)}

/>

Lights:

AmbientLight: Like sun , illuminates all objects in the scene equally.THIS CANNOT CAST A SHODOW as its not a point source

DirectionalLight: Light source in one direction shows light uniformly

PointLight: as the name suggests

HemisphereLight: 2 differnt light color in two differnt areas or hemispheres

All 3 have color and intensity with other specific properties

To cast a shodow use directional light or point light

documentation: https://threejs.org/docs/index.html#api/en/lights/PointLight

Tip: to see the effect of lights change the background color to black

In the example we used spotLight which has various properties like color, intensity, distance, angle , oenumbra and decay which all can be set.

Test it here

[https://threejs.org/examples/#webgl\_lights\_spotlight](https://threejs.org/examples/" \l "webgl_lights_spotlight)

To cast shadow enable cast shaodow in the ball mesh and directional light and enbale plane to get the shadow, also pass shodow parameter to Canvas tag to allow shadows globally.

Code:

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

import SceneContainer from "./SceneContainer";

function App() {

return (

<Canvas shadows>

<SceneContainer />

</Canvas>

);

}

export default App;

SceneContainer:

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

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

import React, { Suspense, useRef, useEffect } from "react";

import { angleToRadians } from "./angle";

const SceneContainer = () => {

const orbitControlsRef = useRef(null);

useFrame((state) => {

if (!!orbitControlsRef.current) {

const { x, y } = state.mouse;

//console.log(y \* angleToRadians(90 - 30));

//console.log(angleToRadians(x \* 24));

orbitControlsRef.current.setAzimuthalAngle(x \* angleToRadians(45));

orbitControlsRef.current.setPolarAngle((y + 1) \* angleToRadians(90 - 30));

orbitControlsRef.current.update();

}

});

useEffect(() => {

if (!!orbitControlsRef.current) {

console.log(orbitControlsRef.current);

}

}, [orbitControlsRef.current]);

return (

<Suspense fallback={null}>

<PerspectiveCamera makeDefault position={[0, 1, 5]} />

<OrbitControls

ref={orbitControlsRef}

minPolarAngle={angleToRadians(60)}

maxPolarAngle={angleToRadians(80)}

/>

<mesh position={[0, 0.5, 0]} castShadow>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#FF0000" />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]} receiveShadow>

<planeGeometry args={[7, 7]} />

<meshPhongMaterial color="#0000FF" />

</mesh>

<ambientLight args={["#ffffff", 0.25]} />

{/\* directionalLight \*/}

{/\* <directionalLight args={["#ffffe0", 1]} position={[-2, 1, 1]} /> \*/}

{/\* <pointLight args={["#ffffe0", 1]} position={[-3, 1, 0]} /> \*/}

<spotLight

args={["#ffffe0", 1.5, 10, angleToRadians(45), 0.5]}

position={[-5, 2, 0]}

castShadow

/>

</Suspense>

);

};

export default SceneContainer;

Environment:

To create dynamic environments.

import {

Environment,

OrbitControls,

PerspectiveCamera,

} from "@react-three/drei";

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

import React, { Suspense, useRef, useEffect } from "react";

import { angleToRadians } from "./angle";

import \* as THREE from "three";

const SceneContainer = () => {

const orbitControlsRef = useRef(null);

useFrame((state) => {

if (!!orbitControlsRef.current) {

const { x, y } = state.mouse;

//console.log(y \* angleToRadians(90 - 30));

//console.log(angleToRadians(x \* 24));

orbitControlsRef.current.setAzimuthalAngle(x \* angleToRadians(45));

orbitControlsRef.current.setPolarAngle((y + 1) \* angleToRadians(90 - 30));

orbitControlsRef.current.update();

}

});

useEffect(() => {

if (!!orbitControlsRef.current) {

console.log(orbitControlsRef.current);

}

}, [orbitControlsRef.current]);

return (

<Suspense fallback={null}>

<PerspectiveCamera makeDefault position={[0, 1, 5]} />

<OrbitControls

ref={orbitControlsRef}

minPolarAngle={angleToRadians(60)}

maxPolarAngle={angleToRadians(80)}

/>

<mesh position={[0, 0.5, 0]} castShadow>

{/\* args:radius,widthsegment,heightsegment \*/}

<sphereGeometry args={[0.5, 32, 32]} />

<meshStandardMaterial color="#ffffff" metalness={0.8} roughness={0.1} />

</mesh>

{/\* floor \*/}

<mesh rotation={[-angleToRadians(90), 0, 0]} receiveShadow>

<planeGeometry args={[8, 8]} />

<meshStandardMaterial color="#E0B0FF" />

</mesh>

<ambientLight args={["#ffffff", 0.25]} />

{/\* directionalLight \*/}

{/\* <directionalLight args={["#ffffe0", 1]} position={[-2, 1, 1]} /> \*/}

{/\* <pointLight args={["#ffffe0", 1]} position={[-3, 1, 0]} /> \*/}

<spotLight

args={["#ffffff", 1.5, 20, angleToRadians(45), 0.5]}

position={[-5, 2, 0]}

castShadow

/>

<Environment background>

<mesh>

<sphereGeometry args={[50, 100, 100]} />

<meshBasicMaterial side={THREE.BackSide} color="#CF9FFF" />

</mesh>

</Environment>

</Suspense>

);

};

export default SceneContainer;

Animations:

Using GSAP

Adding Textures:

Next step is adding texture to the geometry we have created

Following website to find free textures:

polyheaven

textures.com

Create a folder inside public folder named textures. Downlad .jpg images to use as textures for the images

using useTexture hook/function(i think its called a hook but its like an three-drei library which can be imported)

using this load the image in a const

const colorTexture= useTexture(

"./textures/TexturesCom\_SolarCells\_1K\_albedo.jpg"

);

Then map this to the meshStandardmaterail as:

map={colorTexture}

Simialrly for environment tag can be done giving the o/p:

