**THREE.js:**

**Web gl?**

**React 3 fiber.**

Three.js use GPL

We draw thing on canvas.

**Some repetative tasks:**

* **Scene:**

Scene is the whole area.

Area behind the camera + front of camera

All 3D world is is scene.

New THREE.Scene()

* **Camera:**

Camera is field/area in which we can see. Camera is perspective and the area in which we can see.

New THREE.perspeciveCamera(

65, 🡪 FOV (take camera new or far of the object according to value)

 window**.***innerWidth* **/** window**.***innerHeight***, 🡪 width and height ratio**

0.1, 🡪 near view( the thing at the distance of less than 0.1 will not be seen)

100 🡪 far view ( the thing at the distance of more than 100 will not be seen)

)

After creating camera we have to add camera into the scene.

Scene.add(camera)

Camera.postion(will) will decide the perspective, we can do in any of one dimension.

* **Mesh:🡪(geometry & Camera):**

Mesh.position.x/y/z=val; it tell the position in any axis

We can also use rotation in all direction. Suppose we have a pipe and a pencil. Pipe will take pencil as axis for rotation. Assume position of pencil in all axis then assume the rotation of pipe. It works like that.

For rotation we can use some mathematics, (1 pi means one rotation, we can divide in 4 parts as according to we need rotation like if we want to rotate only 90 degree then we can divide it by 2. For 45 degree we can use pi/4)

We have to know degree to radians or radians to degree so do some practice on geometry.

**Scaling can also be done in it.**

* **Renderer:**

It means print every timeframe pic, like we put camera and scene as arguments in the renderer then it shows about it one the screen.

Rendering will be according to the fps speed.

* **requestAnimationFrame:**

we use it inside a custome function, bcz we use anothers things with it.

* Window.requestAnimationFrame(animate) 🡪 it requestion the FPS speed of the screen
* Then we can render anything according to the requestion FPS, **like** renderer.render(scene,camera)
* As we are doing animation then we can rotate,scale anything by using mesh.
* As rendering speed will be according to the FPS speed of different devices so it will be different on different devices, so instead of fps speed we can set it on time, like in 1 second it do same rotaions on all te devices.

We will use

* Const clock= new THREE.clock()
* Mesh.rotation.x = clock.getElaspedTime();

**Note:** if we are using tailwind in valina js then we have to set script to convert tailwind into the normal css.

Build:css : “tailwindcss -I ./index.css -o ./style.css --watch”

We have to install threejs through npm

**Responsiveness in Animation:**

We can use event listner on resize.

* Window.addEventListener("resize”,()=>{
* Renderer.setSize(window.innerWidth,window.innerHeigth)=> set the renderer area as window size.
* Camera.aspectRatio=window.innerWidth/window.innerHeigth => help to maintain actual size of object by avoiding shrinking.
* Camera.updateProjectionMatrix()🡪 the value that we changed in aspectRation should be updae through it.

})

**Orbit Controls:** (it is addon we have to install it seperately)

Helps us to control object with keys or mouse.

Steps to use.

* Download the ornit control file
* Import orbit control by using

Import {orbitControl} from “./whateverPath/orbitControl.js”

* Then we will create control by using

Const control = new orbitControls(camera,rederer.domElement)

* Controls.update() 🡪 do it inside the animation custom function.
* Inside material object we can also set {wireframe:true}
* Controls.enableDamping=true; 🡪 create smooth animation
* Controls.autoRotate=true; make rotation without animation function
* Control.enableZoom=true;
* Controls.dampingFactor=1.23; tells for how much time it will animate by using mouse, small value long animation.

**Geometries:**

These are different geometry shapes.

Sphere Geometry:

Steps:

* New THREE.SphereGeometry(5,32,16)🡪 1st value is radius, 2nd value is width segments, 3rd value is heigthSegments,4th value is phiStart (horizontal starting angle), 5th value is phiLength (horizontal sweep angle size), 6th value is theta start (vertical tarting angle), 7th value is thetaLength (specify verticle sweep angle).
* Cylinder shape:

**Note:** three js only render front view by default, if we want to render whole object then we have to set {side:THREE.DoubleSide} inside material argument object. Useful during orbit control bcz it takes extra computation so while normal use don’t enable it.

**Mateirals:**

Materials describe the appearance of the object.

There are many type of materials like:

* MeshBasicMaterial,MeshDistanceMaterial(it is physically base rendering mean shadow can be created on it),shaderMaterial(use glsl for oxm effects) etc.

Note: Some Materials and objects need lightening so we have to Put Light on it, for this purpose we use DirectionLight(), ambientLight() and pointLight() etc.

DirectionLight:

In direction light we can set a position where from light will come.

We can set position in all axis

* const directionLight **=** **new** THREE**.**DirectionalLight(0xffffff**,** 1)**;**
* directionLight**.***position***.**set(1**,** 1**,** 1)**;**
* scene**.**add(directionLight)**;**
* it cause shadows

**ambientLight:**

Normal uniform,unfocused light around us.

const light **=** **new** THREE**.**AmbientLight(0xffffff**,**0.2)**; 🡪** 1st argument is light color, 2nd argument is light intensity

scene**.**add(light)**;**

**pointLight:**

it contains four properties. These are default properties passed as argument. We can also set position f light just like directional light

* Color: color of light
* Intensity: strength of light
* Distance: maximum range of light
* Decay: the amout of light dim along with distance

**Usage:**

**LightHelpers:**

Use to visualize direction of lights. Just use for tsting.

**Textures: (**images on obejcts**)**

We use **textureLoader()** to load the textures on the objects.

Steps:

* Download textures for object
* Const loader=new THREE.textureLoader();
* Const textLoad=loader.load(‘./path’)
* In material argument object we can pass ({map:textLoad})
* If we are inserting roughness texture then we will add {textureMap:loadedVariable}
* Similarly displacementMap,normalMap

**lil-gui:**

External Package.

When we import it and write code for it it will create a dashboard on window in which we can add custome values.

Take help of chat-gpt

**3d-Models:** (download free from the stetchfab) (🡪 learn model optimization)

* Download GLB Model (3d-model)
* Download RGBEloader from three js or npm
* Download orbit control
* Import all
* HDRI is 360 lightening (download from poly heaven)
* In RGB Loader we will load hdr (rgbLoader.load())
* After loading hdri we have to set mapping, scene environment and scene background.
* Let Rgbloader=new RGBLoader();
* rgbeLoader.load(" ./path”, (texture)=>{
* texture.mapping=Three.equieRectangularReflectionMaping()
* scene.environment=texture; 🡪 set lightening
* scene.background=texture-> set background of object according to download

})

* for model loading we will use GLTFLoader from npm or threejs adon
* create loader wit constructor:

loader=new GLTFLoader()

* load our 3D model:

loader.load("./path.model”, (gltf)=>{

* scene.add(gltf.scene())
* we can also do postprocessing like postion,rotation,scaling

})