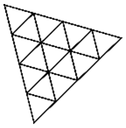
three.js

Three.js is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL



```
npm init
```

↓

```
npm i three
```

or

```
npm init vite
```

or

```
npm create vite@latest
```

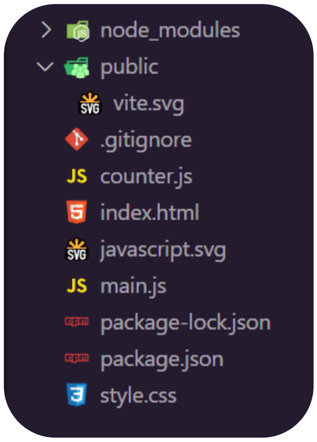
↓

```
npm i three
```

↓

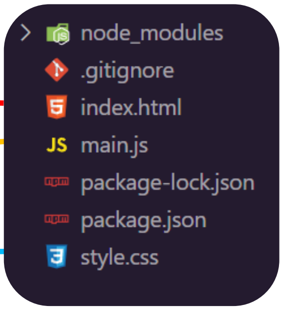
```
npm run dev
```

more information....

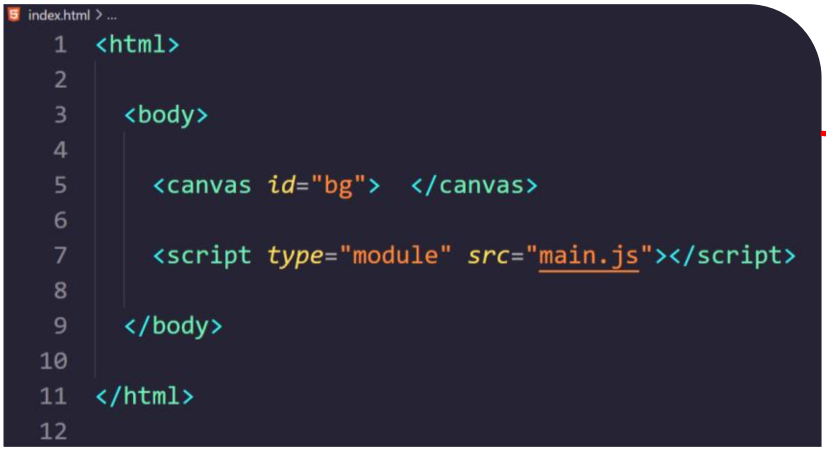


```
> node_modules
v public
    vite.svg
  .gitignore
  counter.js
  index.html
  javascript.svg
  main.js
  package-lock.json
  package.json
  style.css
```

↓

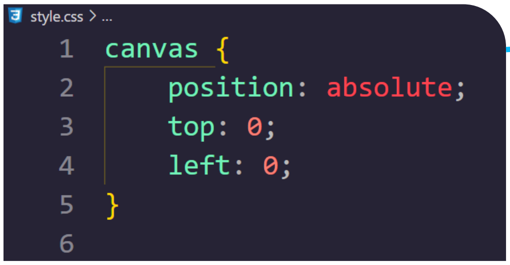


```
> node_modules
  .gitignore
  index.html
  main.js
  package-lock.json
  package.json
  style.css
```



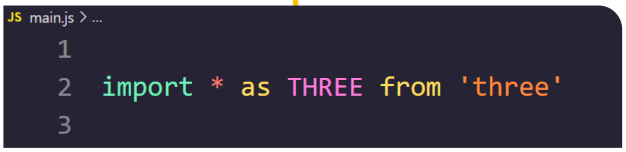
index.html > ...

```html
<html>

  <body>

    <canvas id="bg">  </canvas>

    <script type="module" src="main.js"></script>

  </body>

</html>

```



style.css > ...

```css
canvas {
    position: absolute;
    top: 0;
    left: 0;
}

```

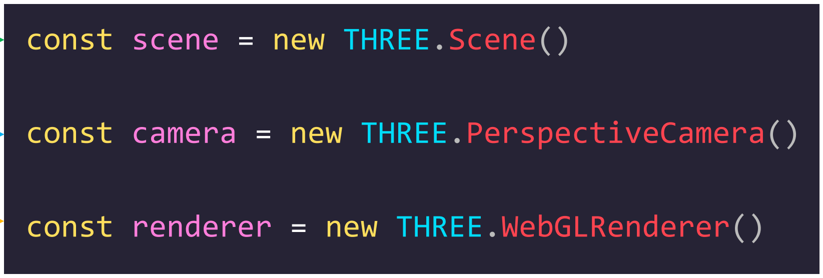


main.js > ...

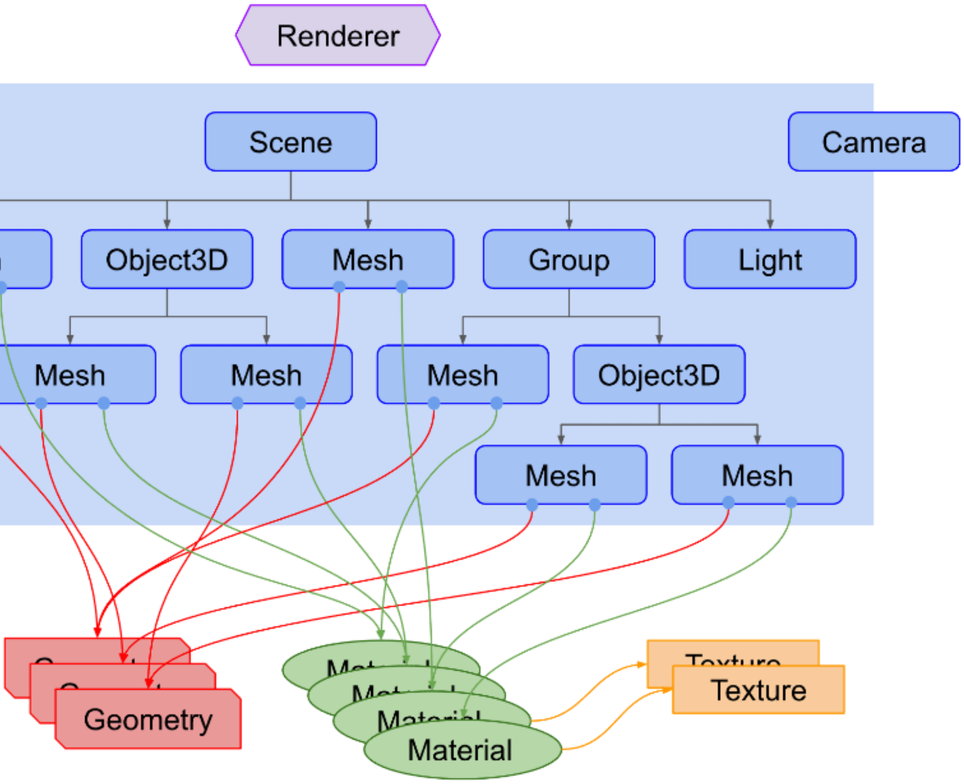
```js

import * as THREE from 'three'

```

3 Fundamental Objects

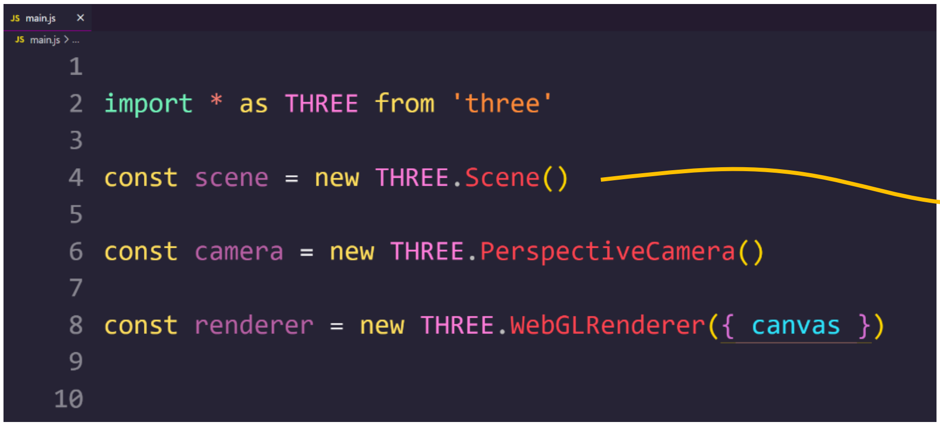
1. Scene
2. Camera
3. Renderer



```
const scene = new THREE.Scene()

const camera = new THREE.PerspectiveCamera()

const renderer = new THREE.WebGLRenderer()
```



Fundamental Steps



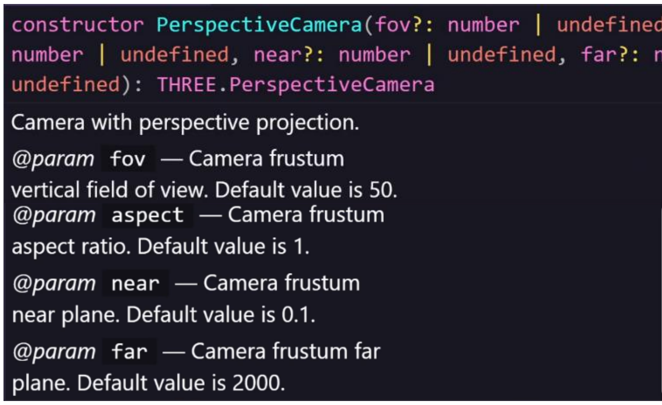
```
import * as THREE from 'three'

const scene = new THREE.Scene()

const camera = new THREE.PerspectiveCamera()

const renderer = new THREE.WebGLRenderer({ canvas })

```

scene == **CONTAINER**

camera

```
const camera = new THREE.PerspectiveCamera( 50, window.innerWidth / window.innerHeight, 0.1, 2000 )
```

camera
parameters



```
constructor PerspectiveCamera(fov?: number | undefined, aspect?: number | undefined, near?: number | undefined, far?: number | undefined): THREE.PerspectiveCamera
```

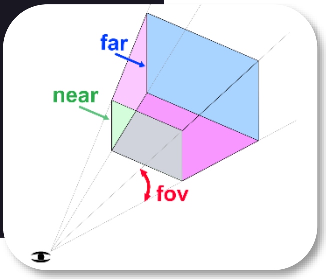
Camera with perspective projection.

@param `fov` — Camera frustum vertical field of view. Default value is 50.

@param `aspect` — Camera frustum aspect ratio. Default value is 1.

@param `near` — Camera frustum near plane. Default value is 0.1.

@param `far` — Camera frustum far plane. Default value is 2000.



Field of view

Aspect ratio

Viewing distance

renderer

```
const renderer = new THREE.WebGLRenderer({ canvas })
```

render == **DRAW**

or

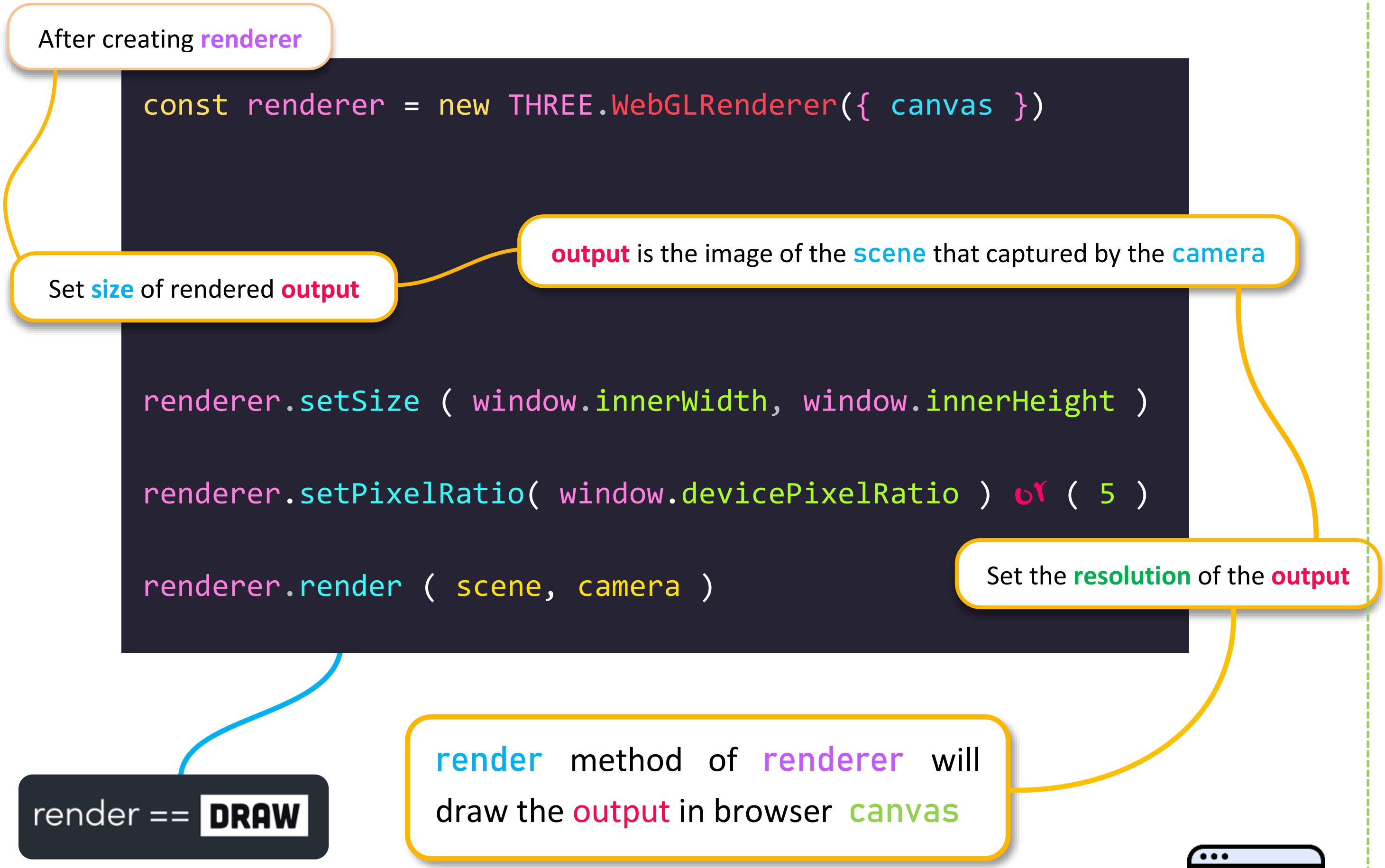
```
<canvas id="bg"> </canvas>
```

if the canvas has id , then

```
const renderer = new THREE.WebGLRenderer({
    canvas: document.querySelector('#bg')
})
```

Need to draw on something

Rendering the output in browser



After creating renderer

```
const renderer = new THREE.WebGLRenderer({ canvas })
```

Set size of rendered output

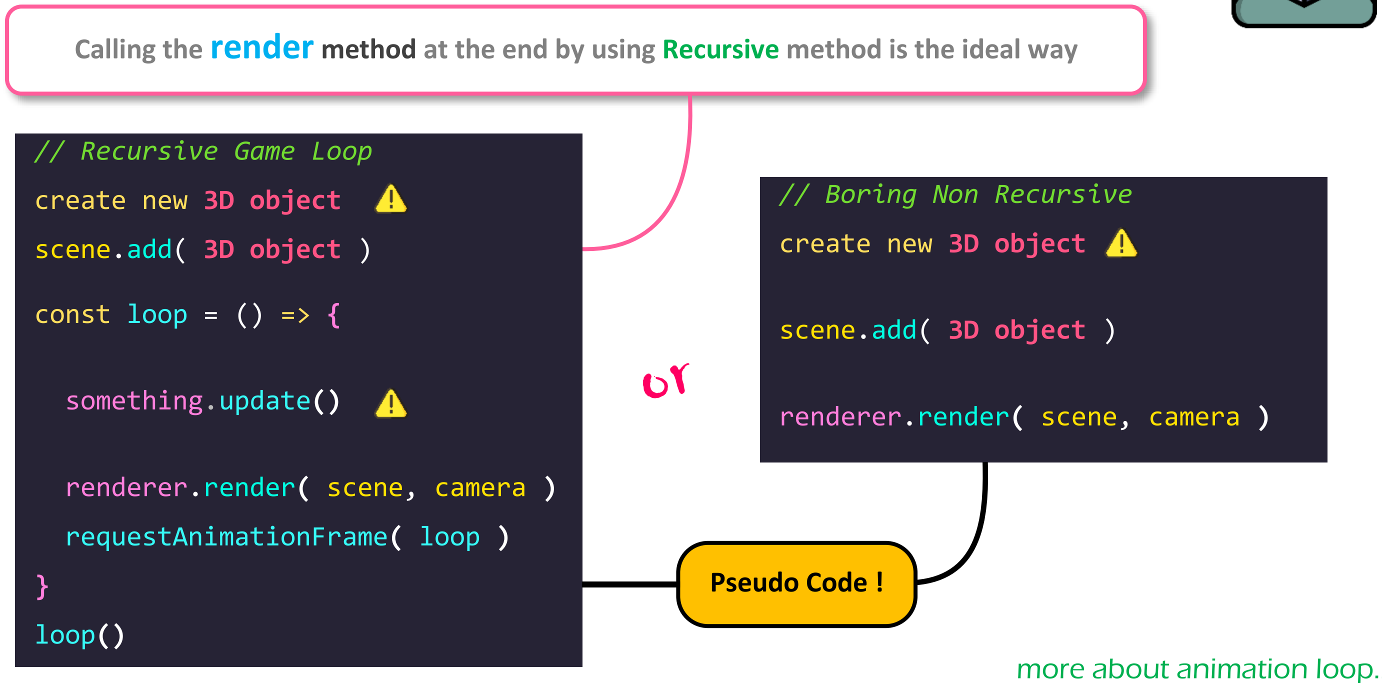
output is the image of the scene that captured by the camera

```
renderer.setSize ( window.innerWidth, window.innerHeight )

renderer.setPixelRatio( window.devicePixelRatio ) or ( 5 )

renderer.render ( scene, camera )
```

Set the resolution of the output

render == DRAW

render method of renderer will draw the output in browser canvas



Calling the **render** method at the end by using **Recursive** method is the ideal way

```
// Recursive Game Loop
create new 3D object ⚠
scene.add( 3D object )

const loop = () => {

  something.update() ⚠

  renderer.render( scene, camera )
  requestAnimationFrame( loop )
}
loop()
```

or

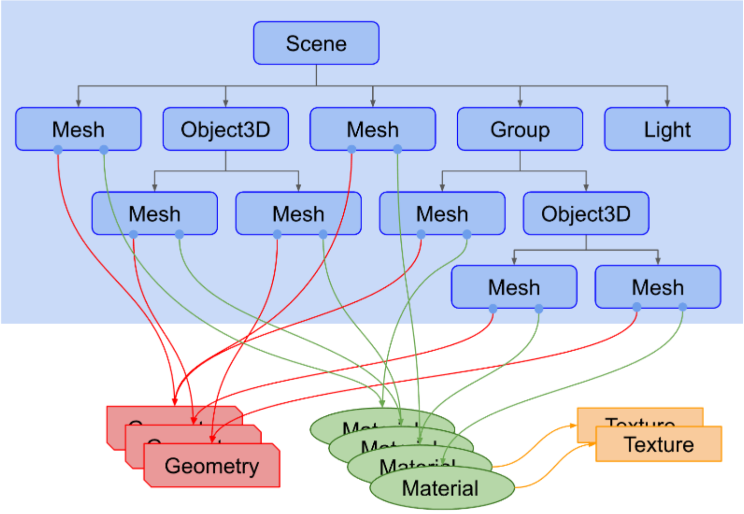
```
// Boring Non Recursive
create new 3D object ⚠

scene.add( 3D object )

renderer.render( scene, camera )
```

Pseudo Code !

more about animation loop...

Lights Camera Action

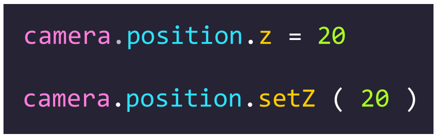
Major steps are done, basic setup completed, now it's time to add **Lights** & **3D** objects to the **scene**



after creating an object, always add it to the scene

```
scene.add ( object-name )
```

Set the camera to a better position

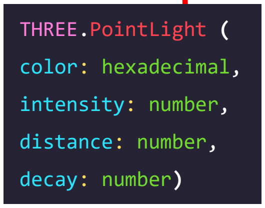


```
camera.position.z = 20

camera.position.setZ ( 20 )
```

or

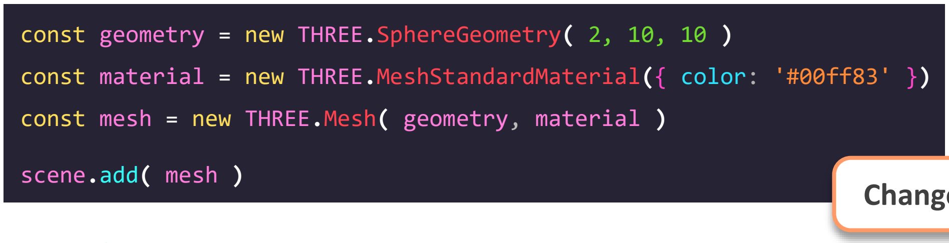
Add a light source to the scene

```
const light = new THREE.PointLight( 0xffffff )

light.position.set( 10,10,10 )

scene.add( light )
```



```
THREE.PointLight (
color: hexadecimal,
intensity: number,
distance: number,
decay: number)
```

Add a mesh to the scene

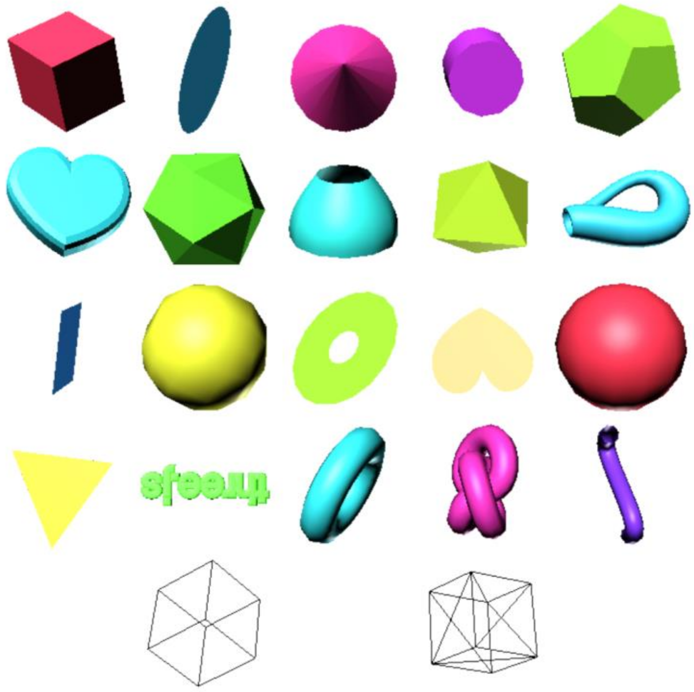


```
const geometry = new THREE.SphereGeometry( 2, 10, 10 )
const material = new THREE.MeshStandardMaterial({ color: '#00ff83' })
const mesh = new THREE.Mesh( geometry, material )

scene.add( mesh )
```

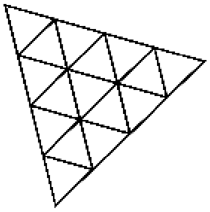
Change the mesh color

more information....

Primitive Geometries



Geometries
BoxGeometry
CapsuleGeometry
CircleGeometry
ConeGeometry
CylinderGeometry
DodecahedronGeometry
EdgesGeometry
ExtrudeGeometry
IcosahedronGeometry
LatheGeometry
OctahedronGeometry
PlaneGeometry
PolyhedronGeometry
RingGeometry
ShapeGeometry
SphereGeometry
TetrahedronGeometry
TorusGeometry
TorusKnotGeometry
TubeGeometry
WireframeGeometry



Lights

Lights
AmbientLight
AmbientLightProbe
DirectionalLight
HemisphereLight
HemisphereLightProbe
Light
LightProbe
PointLight
RectAreaLight
SpotLight

Cameras

Cameras
ArrayCamera
Camera
CubeCamera
OrthographicCamera
PerspectiveCamera
StereoCamera