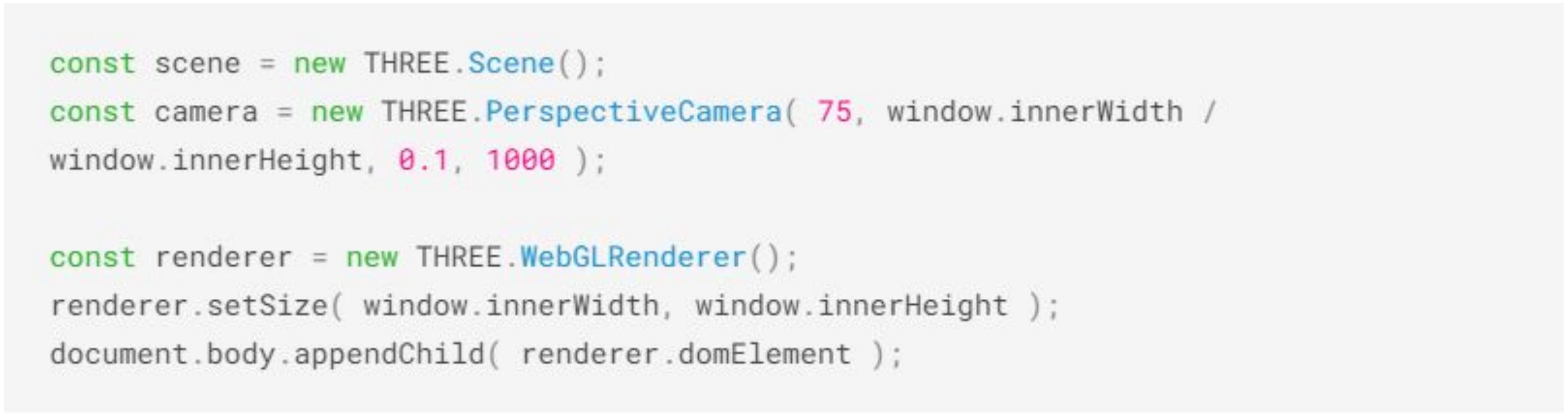
WebGL+ThreeJS

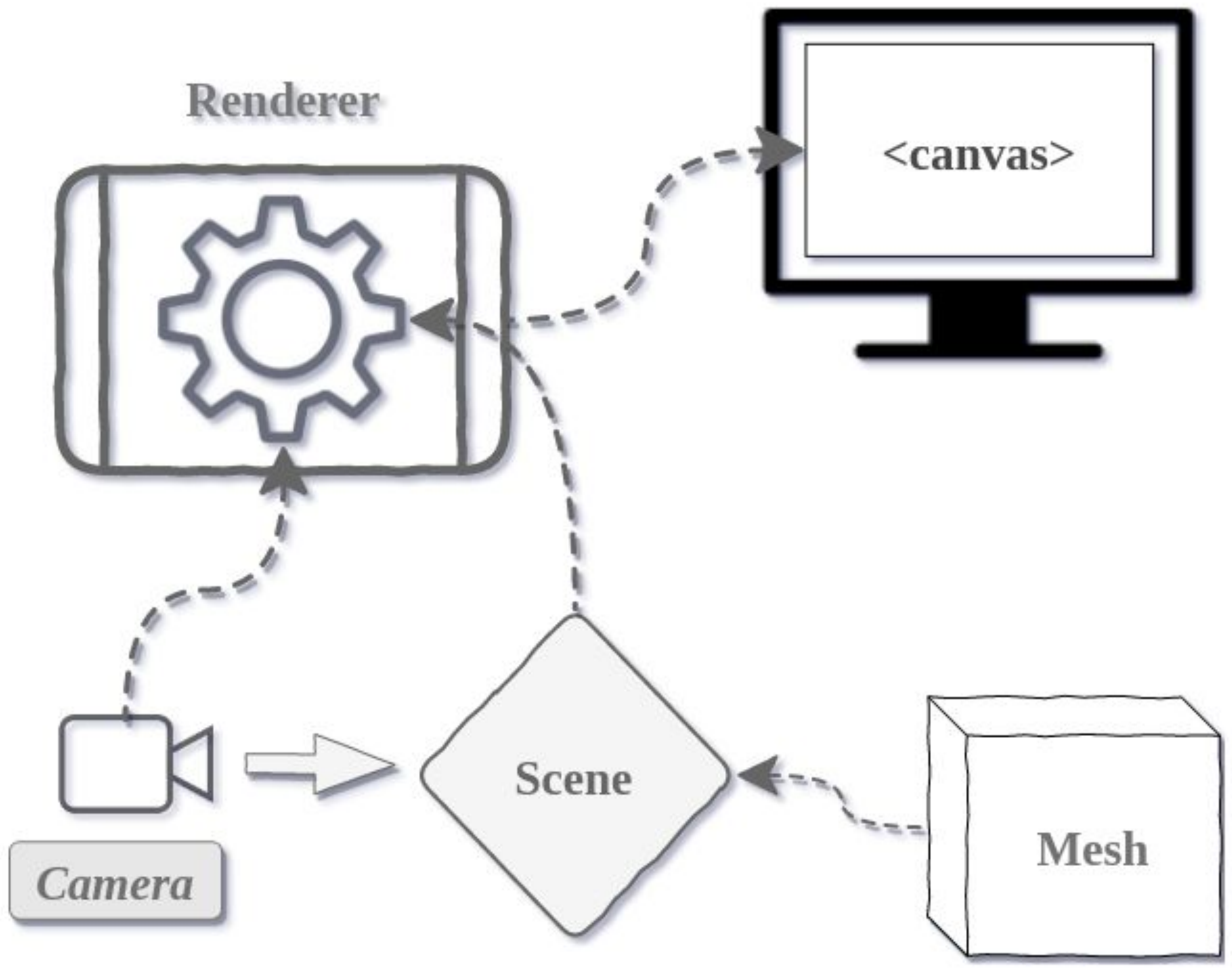
Creating the scene

- To actually be able to display anything with three.js, we need three things: scene, camera and renderer, so that we can render the scene with camera.



```
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 );
```


Renderer
<canvas>
Scene
Mesh
Camera

Before we start

- Before you can use three.js, you need somewhere to display it. Save the following HTML to a file on your computer and open it in your browser.

  https://threejs.org/build/three.js

- https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

File MSSV.html

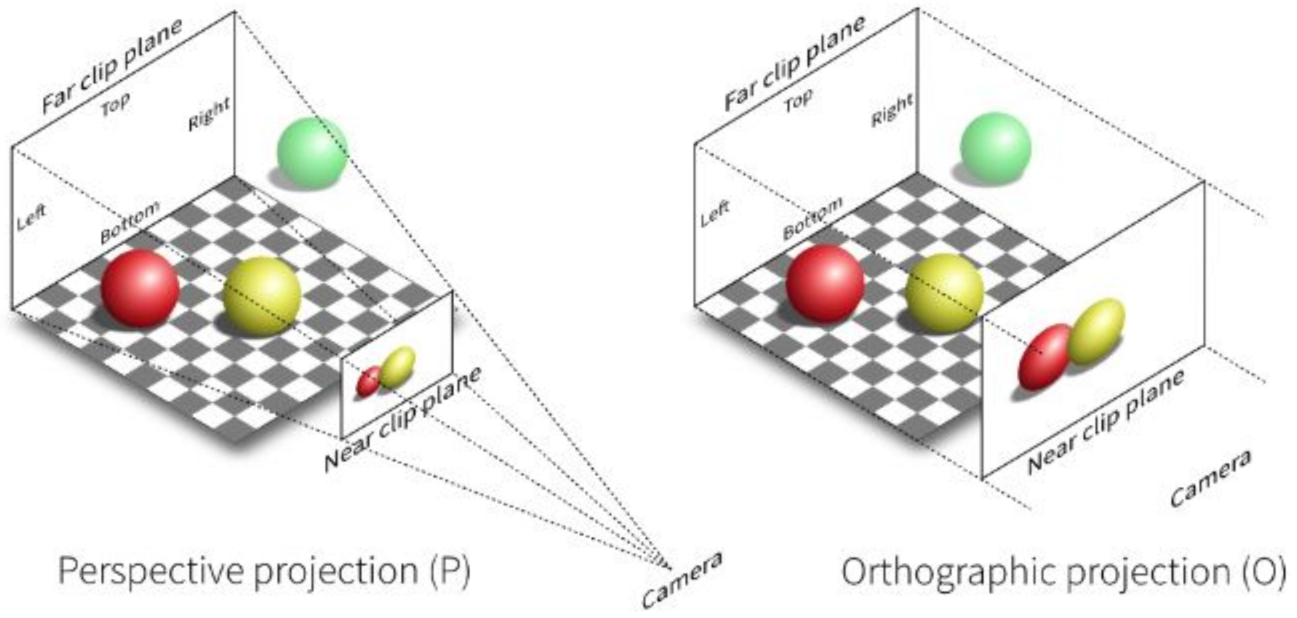
```
<!DOCTYPE html>
<html>
	<head>
	<title>MSSV</title>
	</head>
	<body>

		<script src="https://threejs.org/build/three.js"></script>

		<script>
			// Nội dung code
		</script>
	</body>
 </html>
```

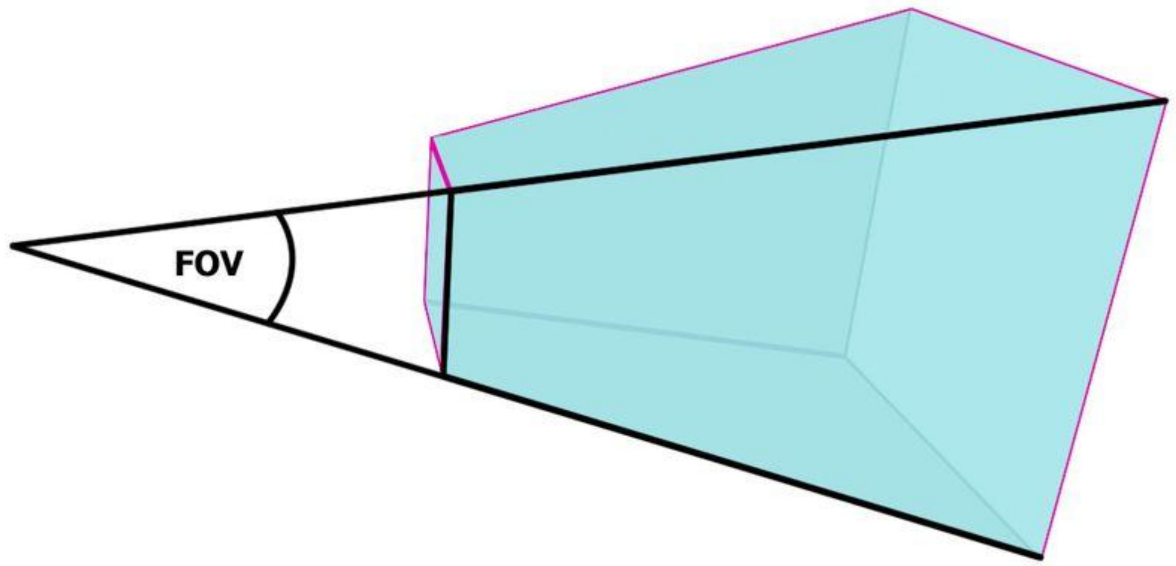
Camera

- PerspectiveCamera
- OrthographicCamera

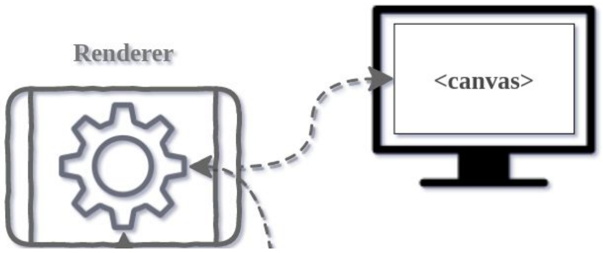


Camera

- PerspectiveCamera(fov : Number, aspect : Number, near : Number, far : Number)



Renderer

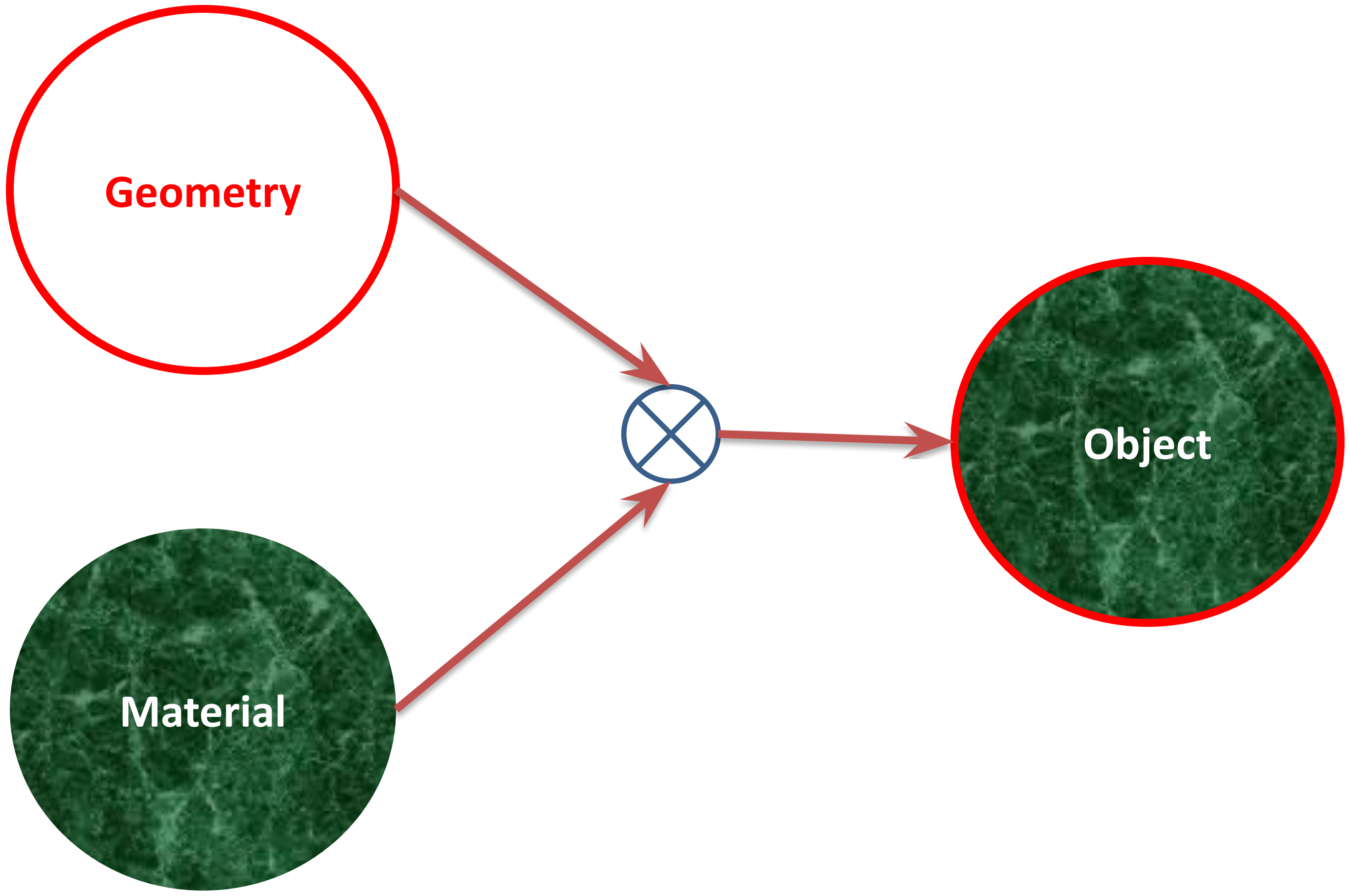


- var renderer = new THREE.WebGLRenderer();
- renderer.setSize(window.innerWidth, window.innerHeight);

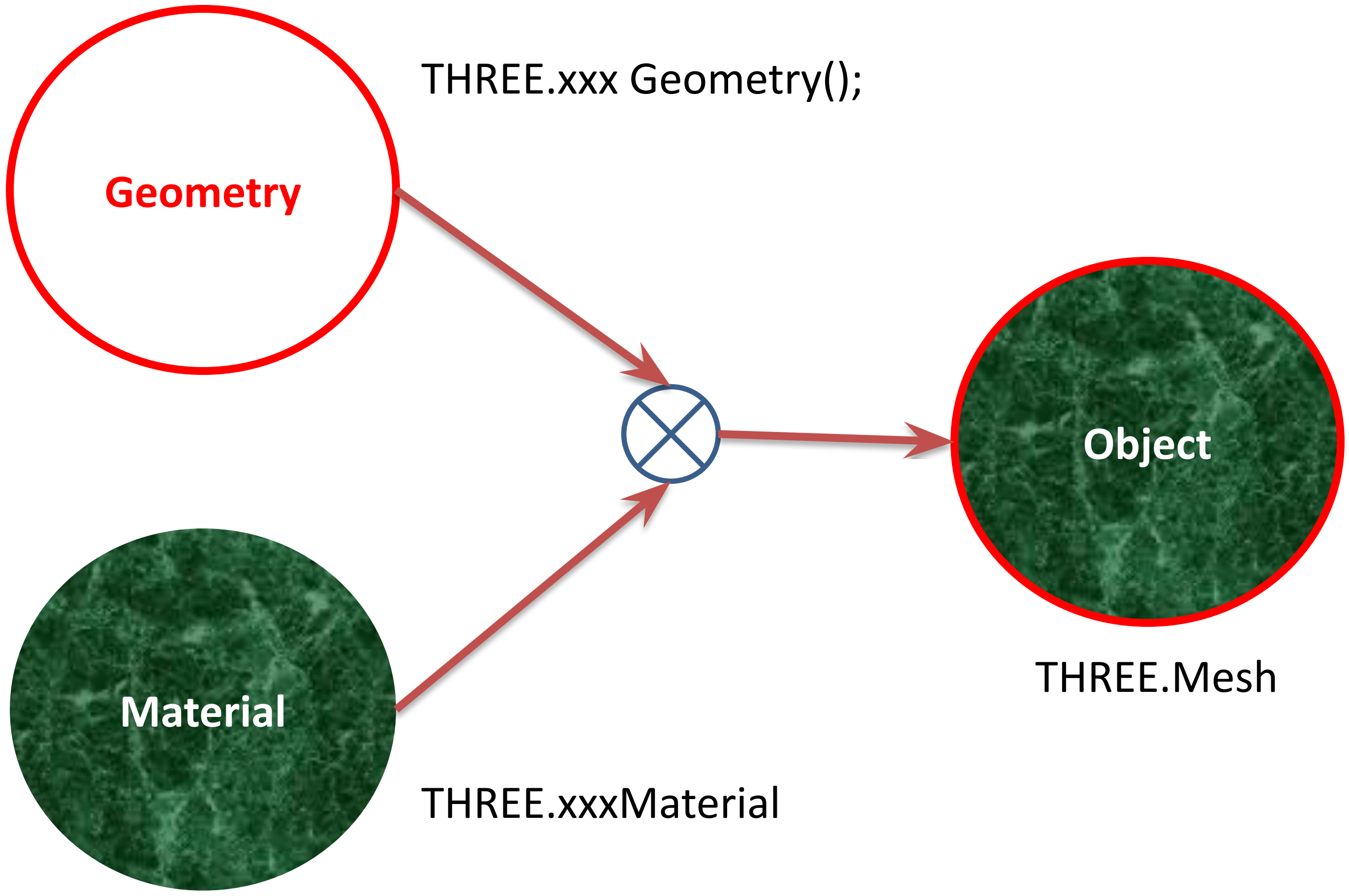
we add the **renderer** element to our HTML document. This is a <canvas> element the renderer uses to display the scene to us.

- document.body.appendChild(renderer.domElement);

Add Objects



Add Objects


THREE.xxx Geometry();
Geometry
Object
THREE.Mesh
Material
THREE.xxxMaterial

Add Objects

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

Add Objects

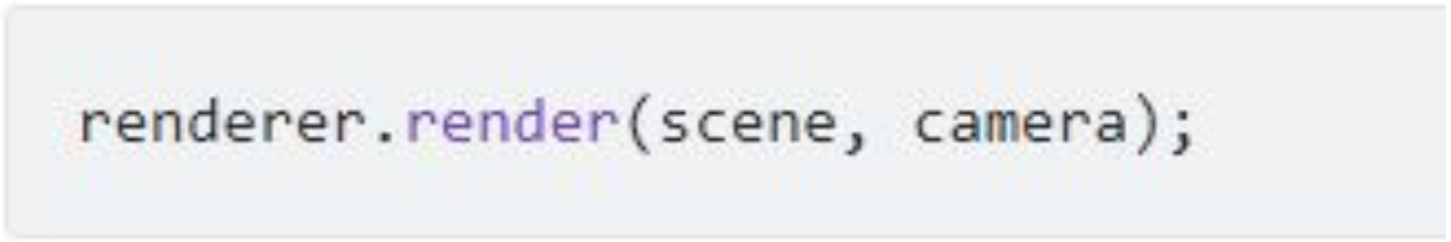
- Create an object

```
const geometry = new THREE.BoxGeometry();
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
```

- Add into scene

```
scene.add( cube );
```

Rendering the scene



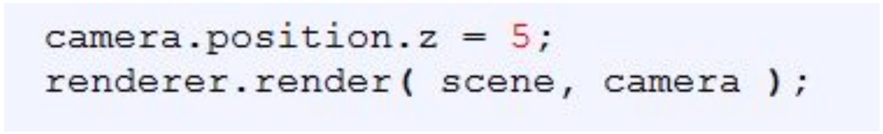
```
renderer.render(scene, camera);
```

Camera position

- By default, when we call **scene.add()**, the thing we add will be added to the coordinates **(0,0,0)**. This would cause both the camera and the cube to be inside each other. To avoid this, we simply move the camera out a bit.

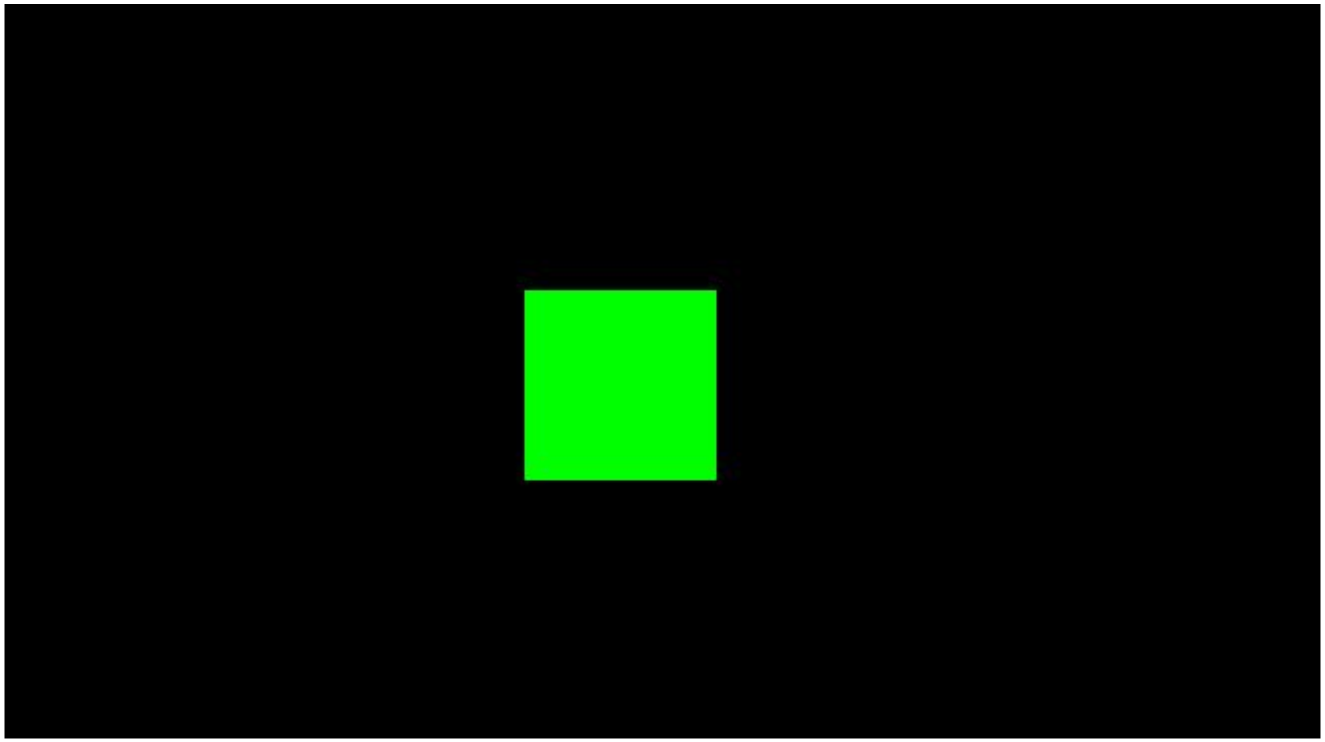
- camera.position.set(0, 0, 5);

- camera.position.x = 0;
- camera.position.y = 0;
- camera.position.z = 5;



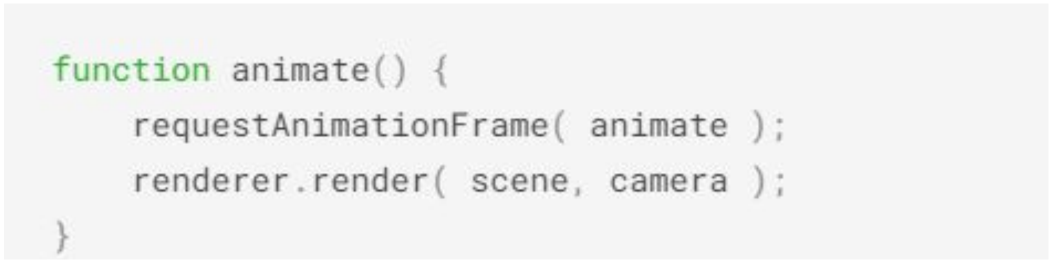
```
camera.position.z = 5;
renderer.render( scene, camera );
```

The result



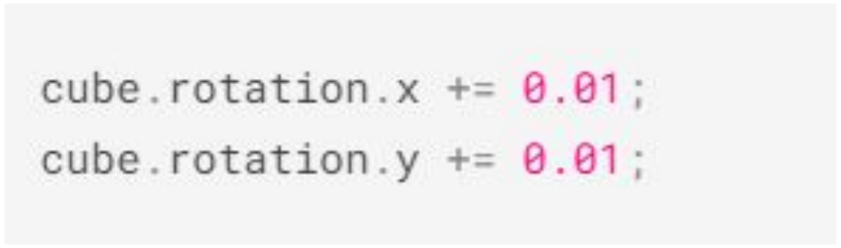
Animating the cube

- we need what's called a **render or animate loop**.



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

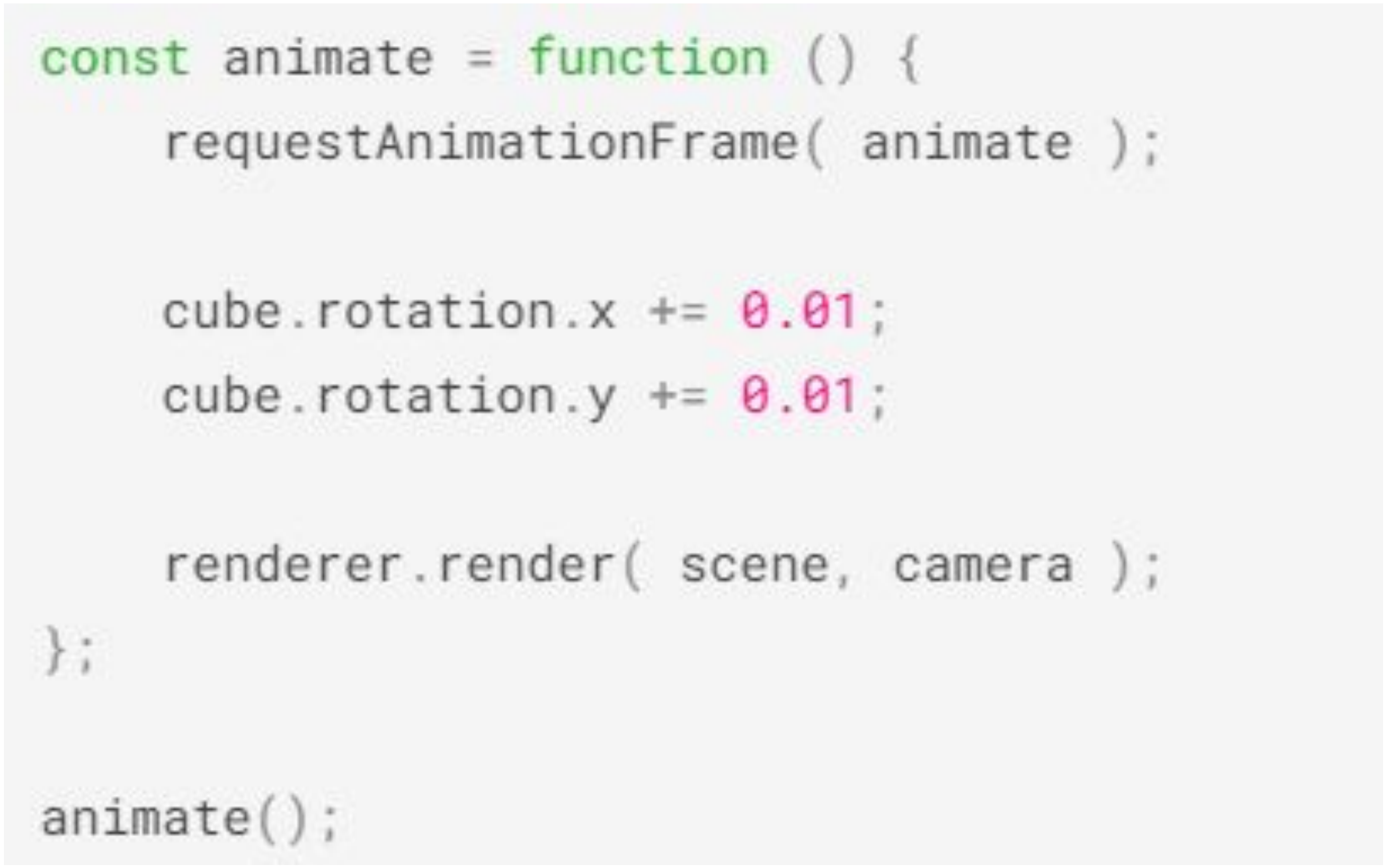
- make it all a little more interesting by rotating it



```
cube.rotation.x += 0.01;
cube.rotation.y += 0.01;
```

□ This will be run every frame (normally 60 times per second), and give the cube a nice rotation animation.

Animating the cube



```
const animate = function () {
    requestAnimationFrame( animate );

    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;

    renderer.render( scene, camera );
};

animate();
```

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>MSSV</title>
       <style> body { margin: 0; } </style>
   </head>



   <body>

      <script
src=“https://threejs.org/build/three.js"></script>
      <script>
         // Our Javascript will go here.
      </script>
   </body>
 </html>
```

```
    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 );

    const geometry = new THREE.BoxGeometry();
    const material = new THREE.MeshBasicMaterial( { color:
0x00ff00 } );
    const cube = new THREE.Mesh( geometry, material );
    scene.add( cube );

    camera.position.z = 5;
    const animate = function ()
     {
       requestAnimationFrame( animate );
       cube.rotation.x += 0.01;
       cube.rotation.y += 0.01;
       renderer.render( scene, camera );
    };
    animate();
```

Responsive canvas

Responsive canvas

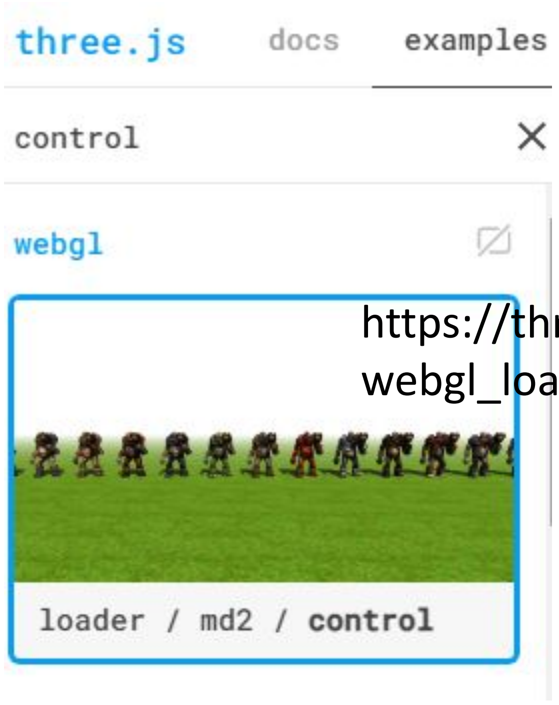
- Window is resized ?

```
window.addEventListener('resize', function() {
        var width = window.innerWidth;
        var height = window.innerHeight;
renderer.setSize(width, height);
        camera.aspect = width / height;
camera.updateProjectionMatrix();
});
```

Updates the camera projection matrix. Must be called after any change of parameters.

Responsive canvas

- OrbitControls
  - https://threejs.org/examples/js/controls/OrbitControls.js

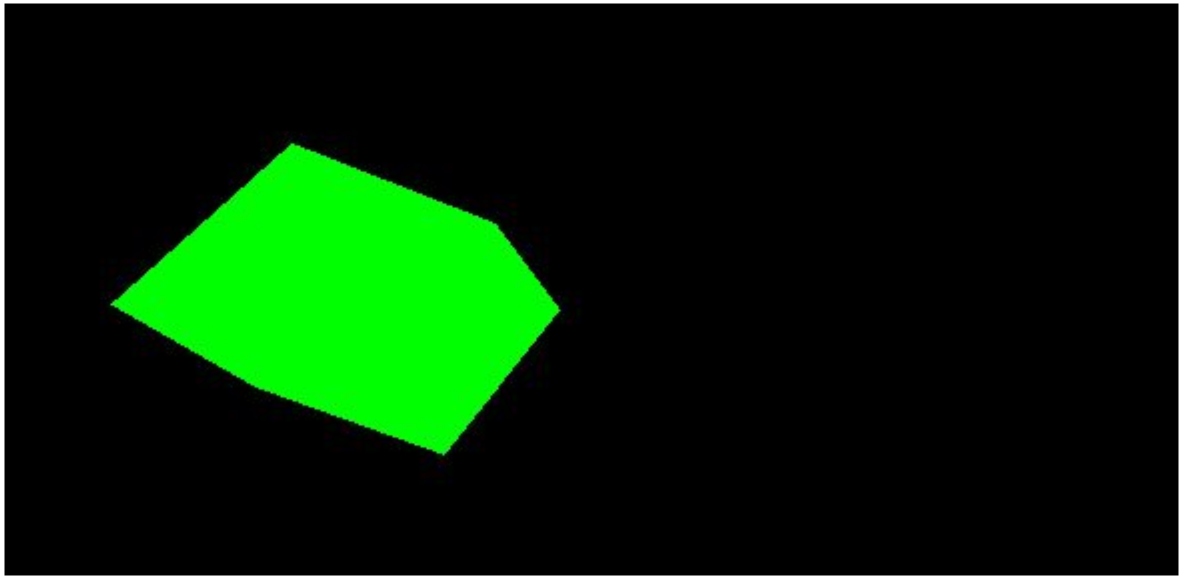


https://threejs.org/examples/?q=control#
webgl_loader_md2_control

Responsive canvas

- OrbitControls

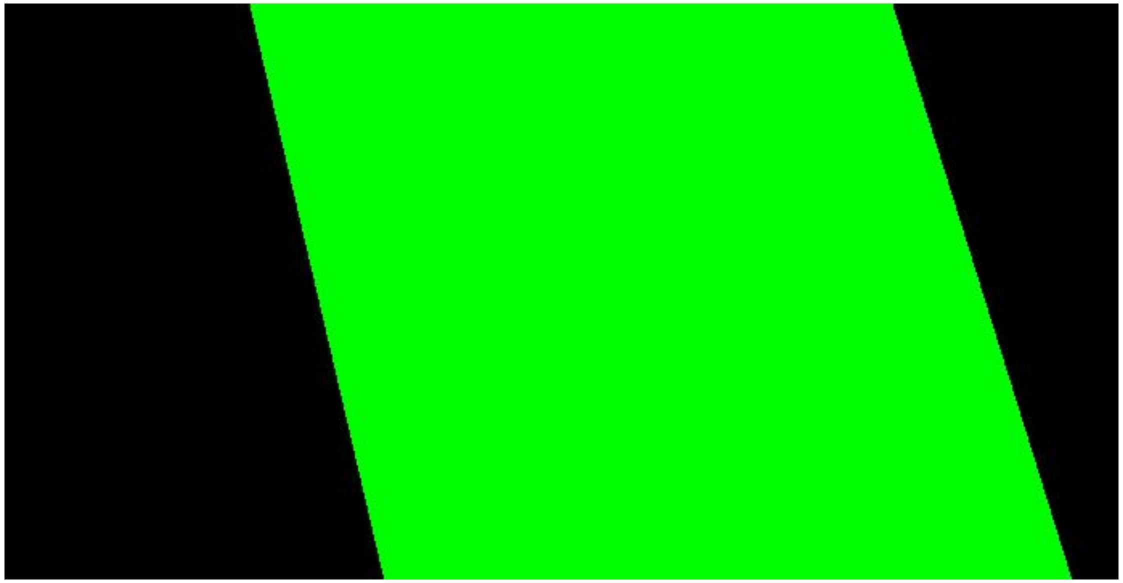
```
controls = new THREE.OrbitControls(camera, renderer.domElement);
```



Responsive canvas

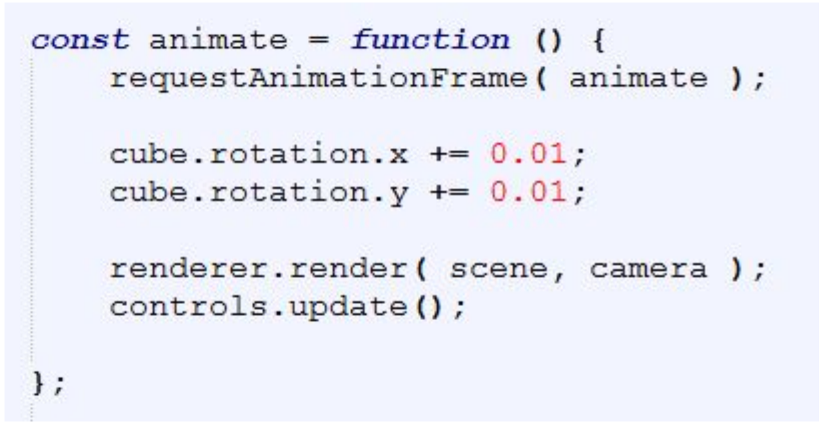
- DragControls
  - https://threejs.org/examples/js/controls/DragControls.js

```
controls = new THREE.DragControls([cube], camera, renderer.domElen
```



Responsive canvas

- TrackballControls
  - https://threejs.org/examples/js/controls/TrackballControls.js

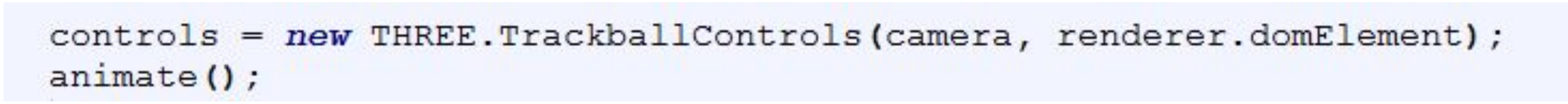


```
const animate = function () {
    requestAnimationFrame( animate );

    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;

    renderer.render( scene, camera );
    controls.update();

};
```



```
controls = new THREE.TrackballControls(camera, renderer.domElement);
animate();
```