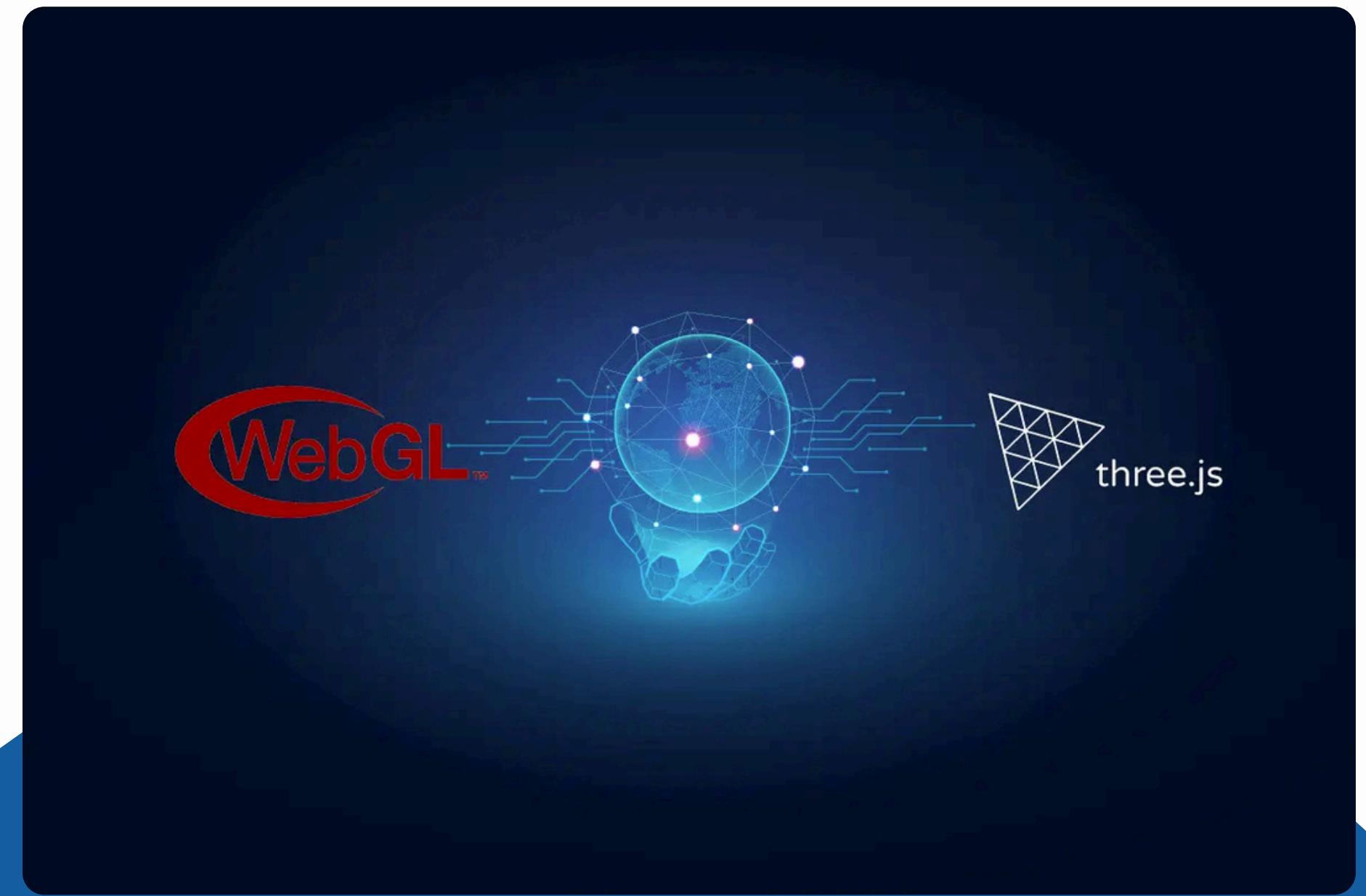


WebGL and Three.js



Team



samuel.montoya.diaz.24@ull.edu.es



roberto.padron.31@ull.edu.es

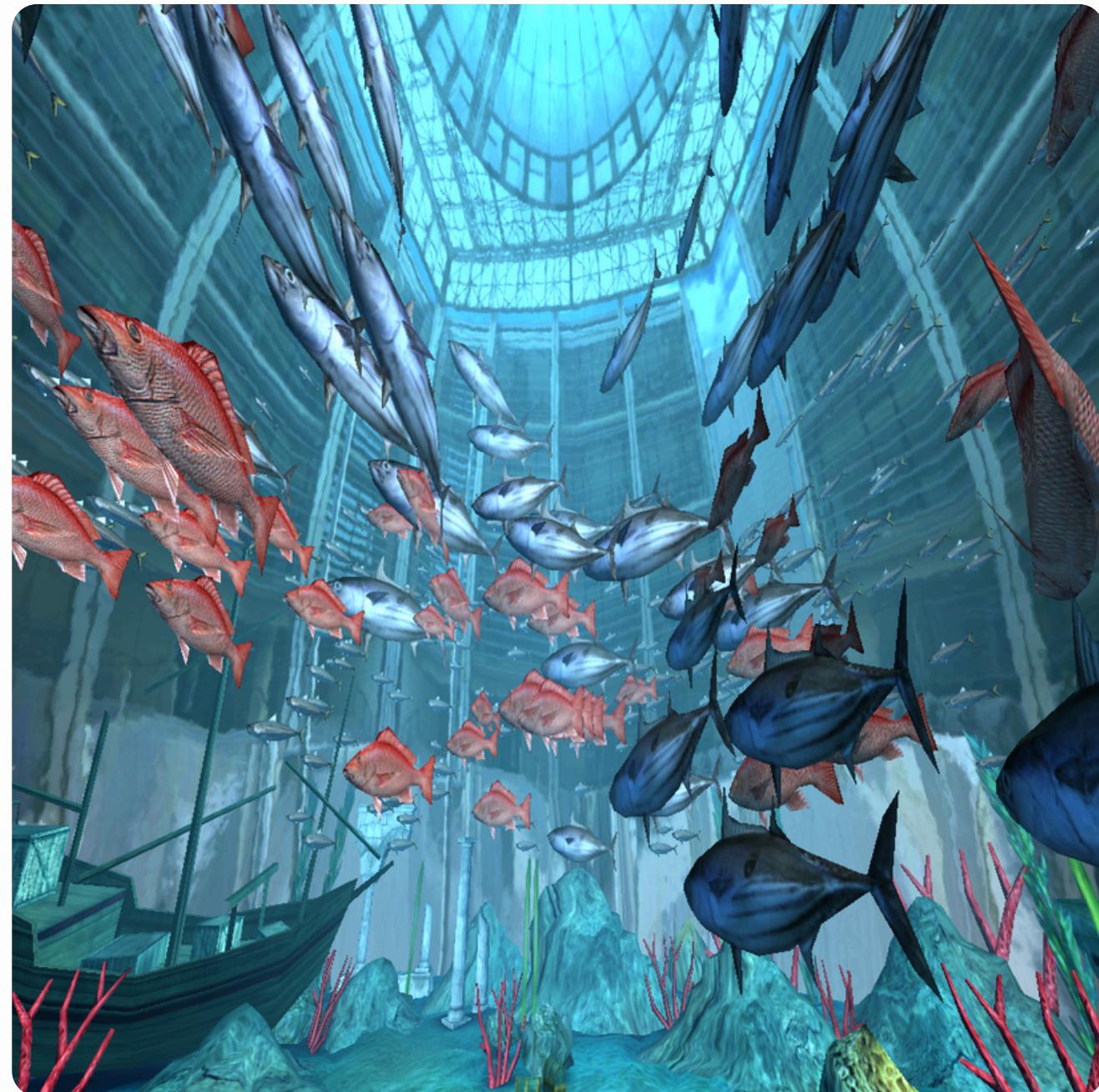


alu0101551395@ull.edu.es

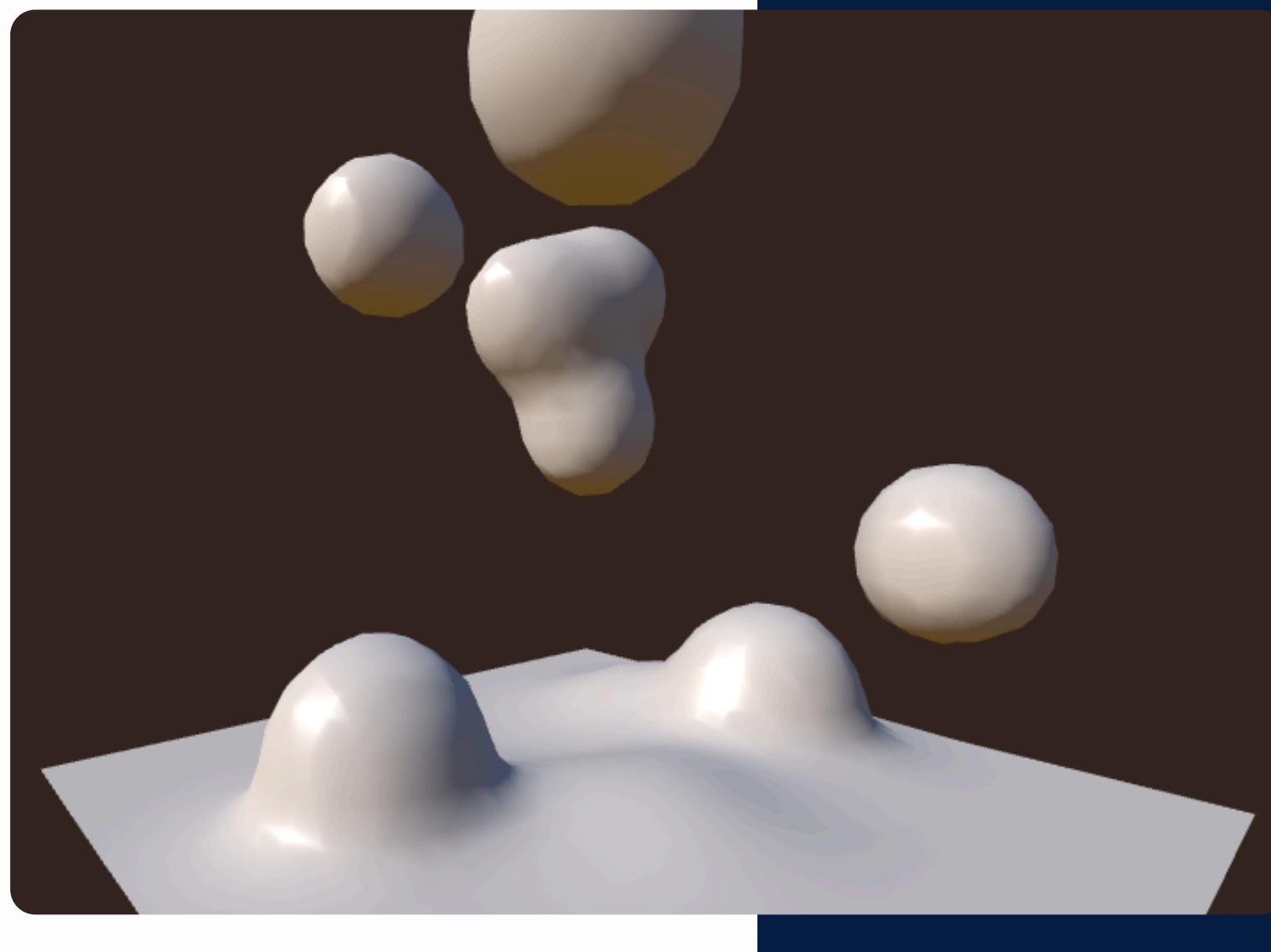
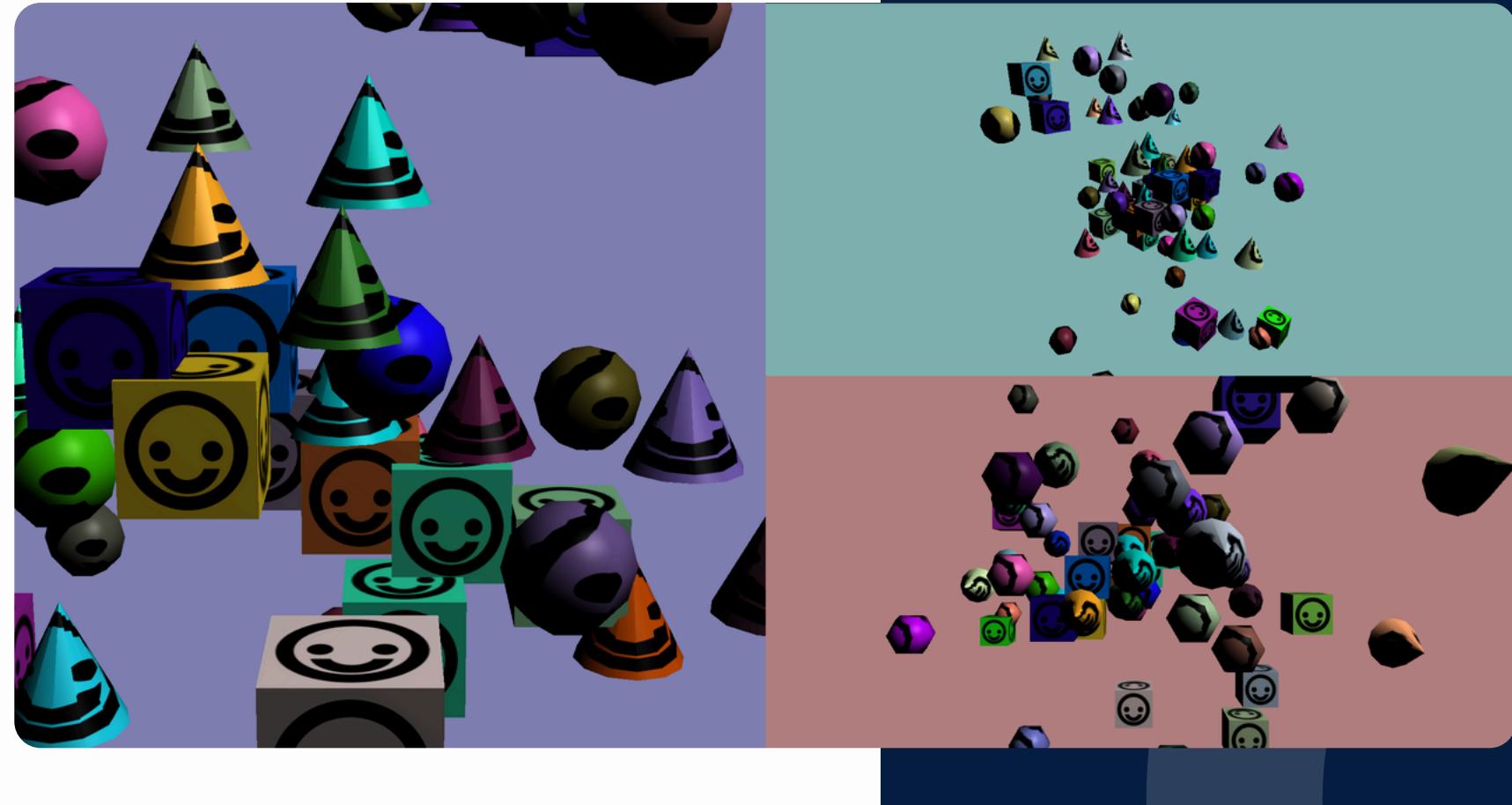
Overview:

▶ WebGL Introduction	01	▶ Materials	09
▶ WebGL Basics	02	▶ Textures	10
▶ WebGL 2D and 3D	03	▶ Cameras	11
▶ Three.js introduction	04	▶ Lights	12
▶ Installation and setup	05	▶ Shadows	13
▶ Three.js structure	06	▶ Obj and Fbx Load	14
▶ First scene	07	▶ Clock on three js	15
▶ Primitive figures	08	▶ Bibliography	16

WebGL Motivation



WebGL examples



1) WebGL Introduction



- Standard for web graphics.
- Interactive 2D and 3D rendering.
- JavaScript API compatible with HTML5.
- No plugins or add-ons required.

Advantages of WebGL

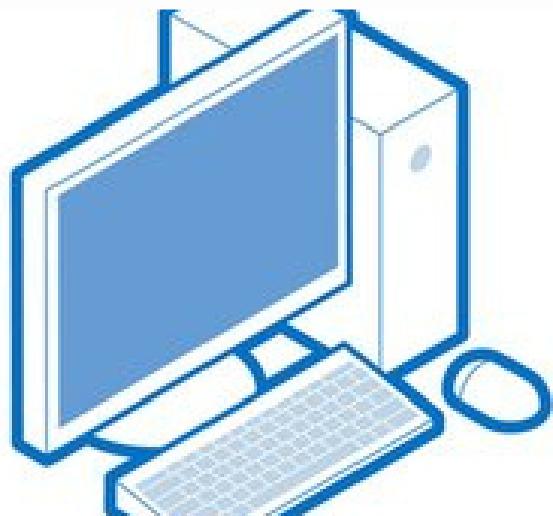


JavaScript
programming

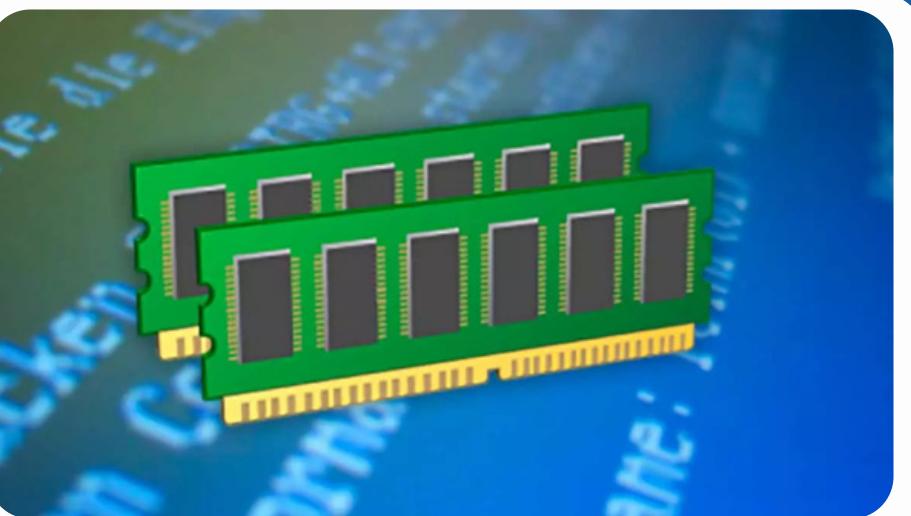


Better Mobile Browser
Compatibility

Open Source



No Need for Compilation



Automatic Memory
Management

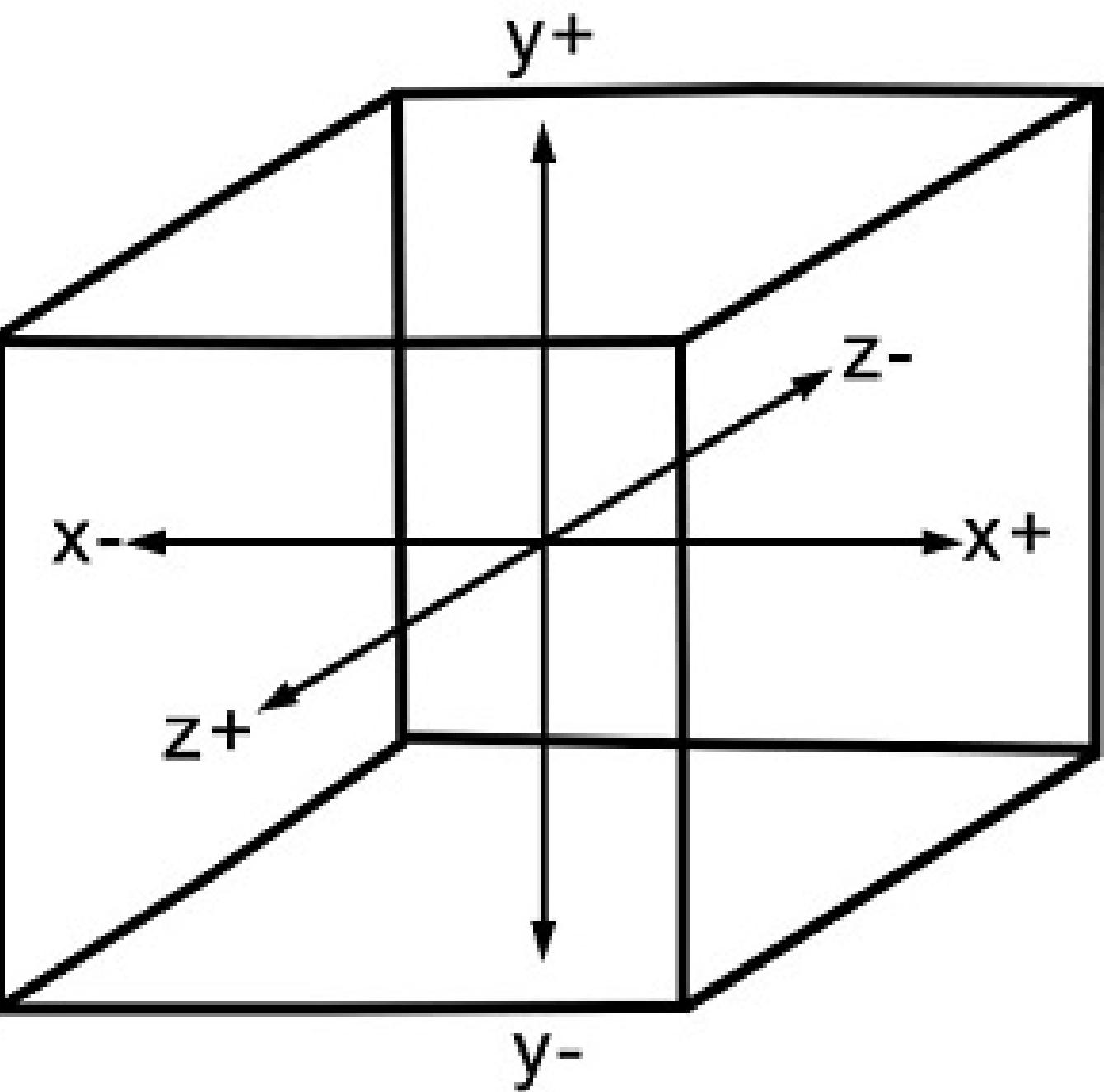


Easy to Set Up

2) WebGL Basics



- Coordinates:

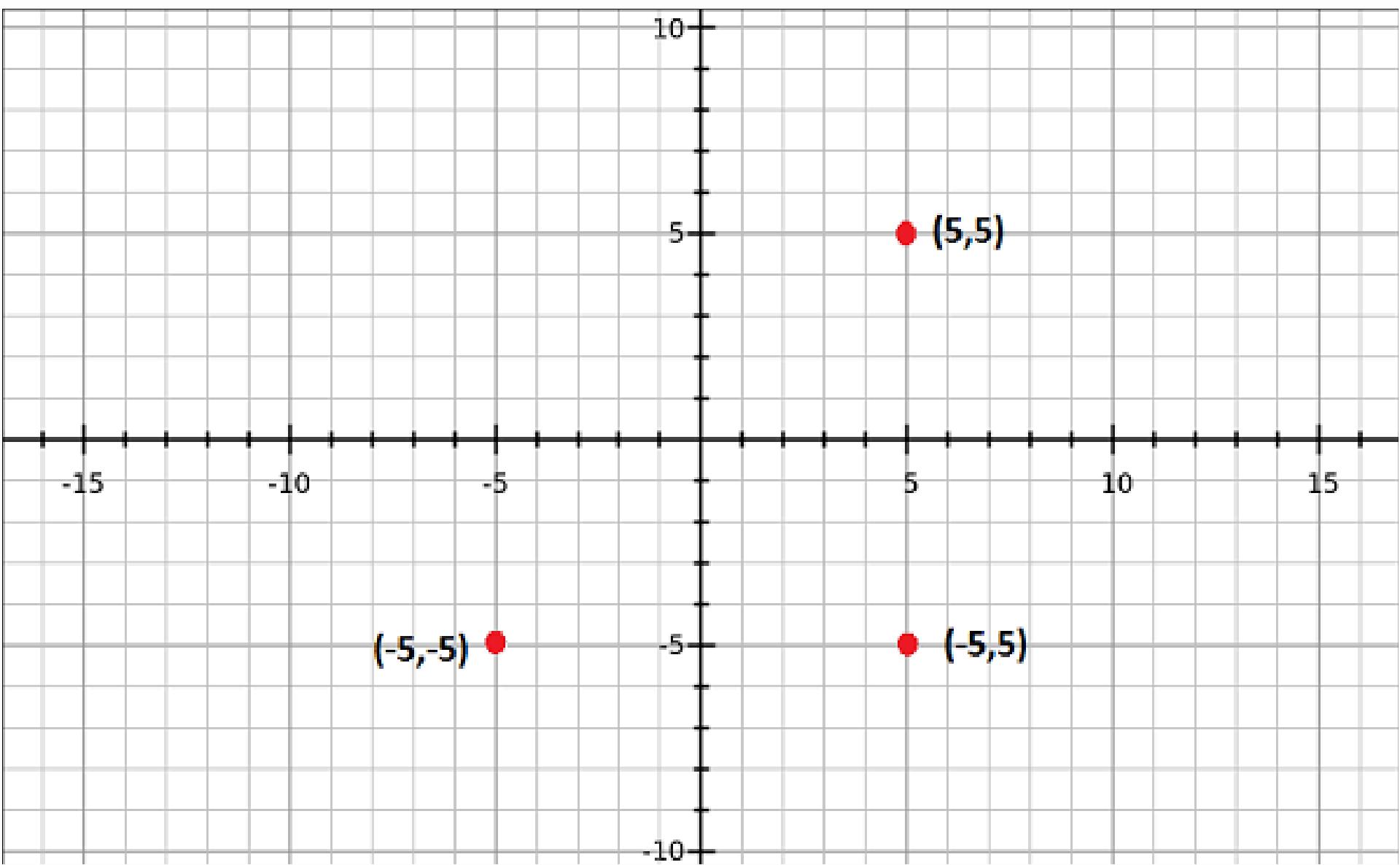


2) WebGL Basics



- Graphics: A small icon of a paintbrush with a yellow handle and a white bristle tip, positioned next to the word "Graphics".
- Vertex or Index.
- **Buffer object.**

(0.5, 0.5), (-0.5, 0.5), (-0.5, -0.5)

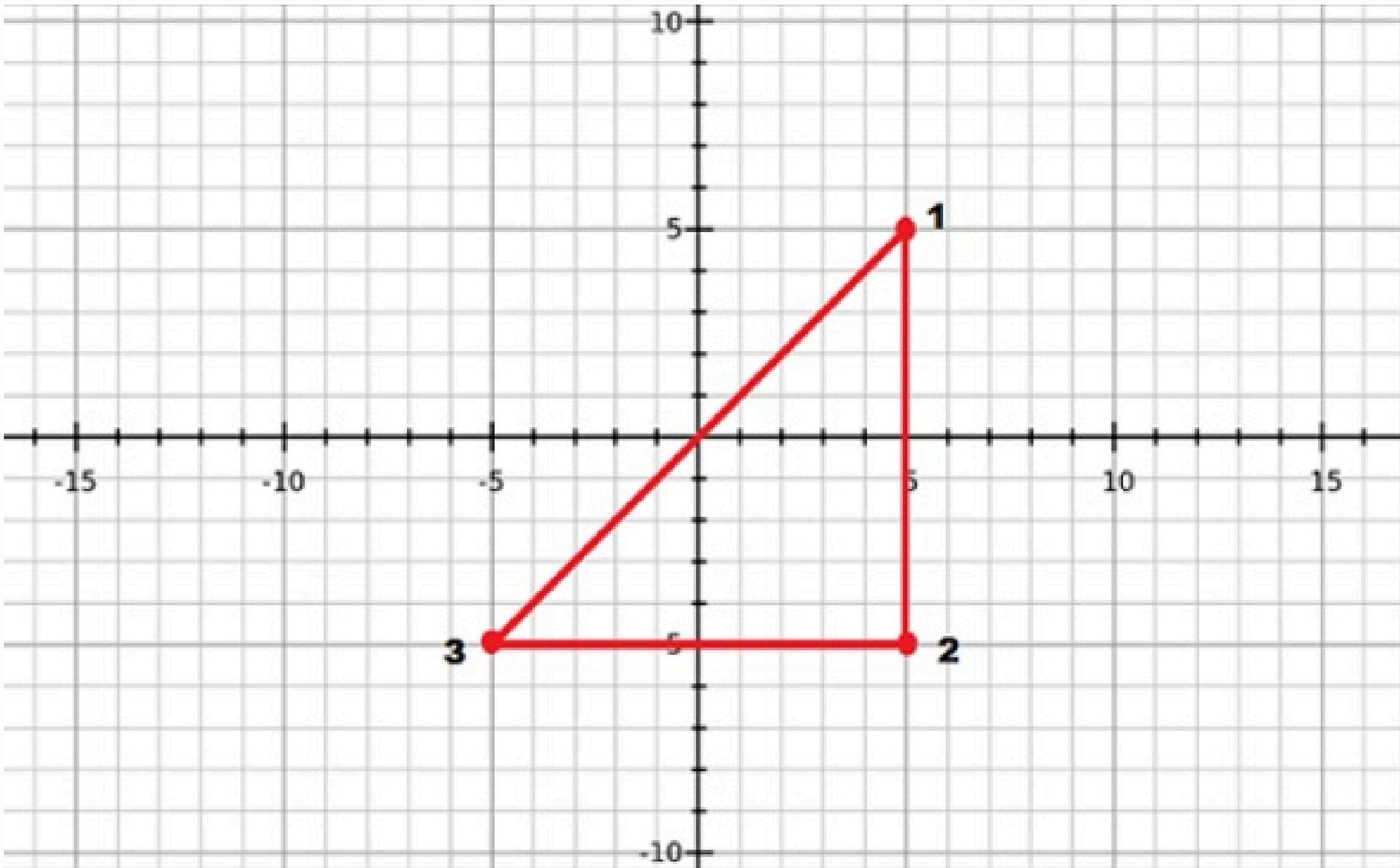


2) WebGL Basics



- Graphics:

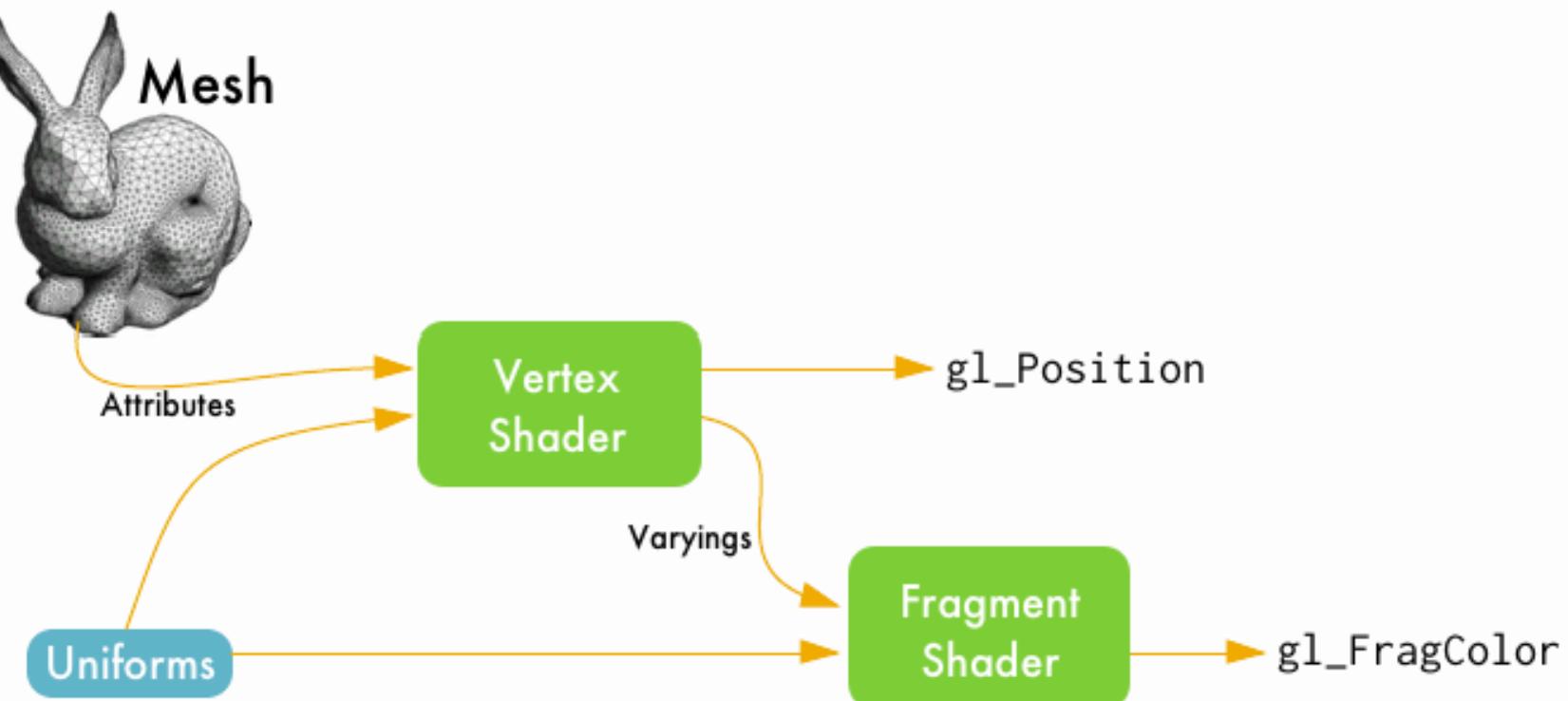
(0.5, 0.5), (-0.5, 0.5), (-0.5, -0.5)



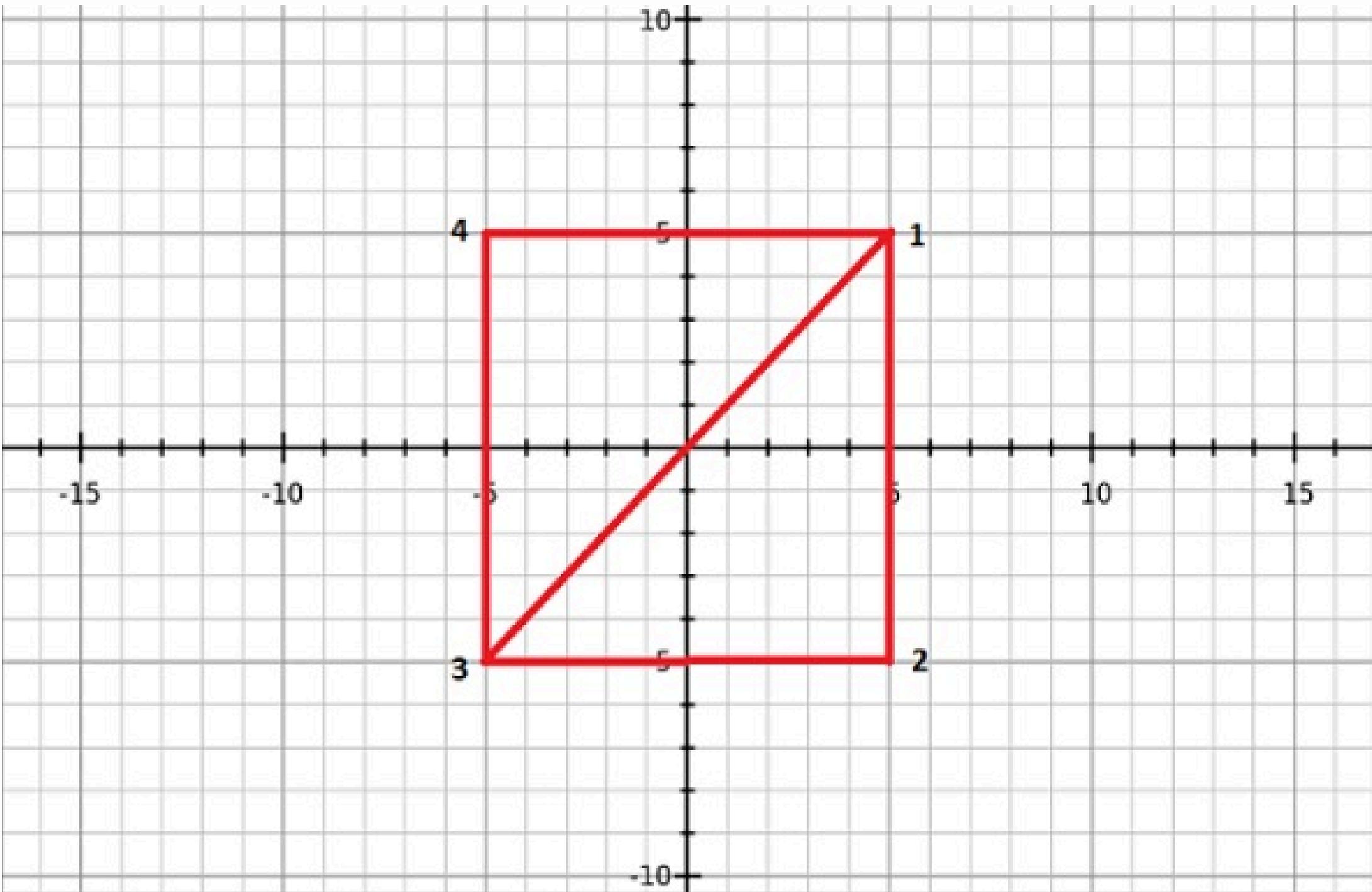
2) WebGL Basics



- Buffers: WebGL data storage areas.
- Meshes: **drawArray()** and **drawElement()**.
- **Mode.**



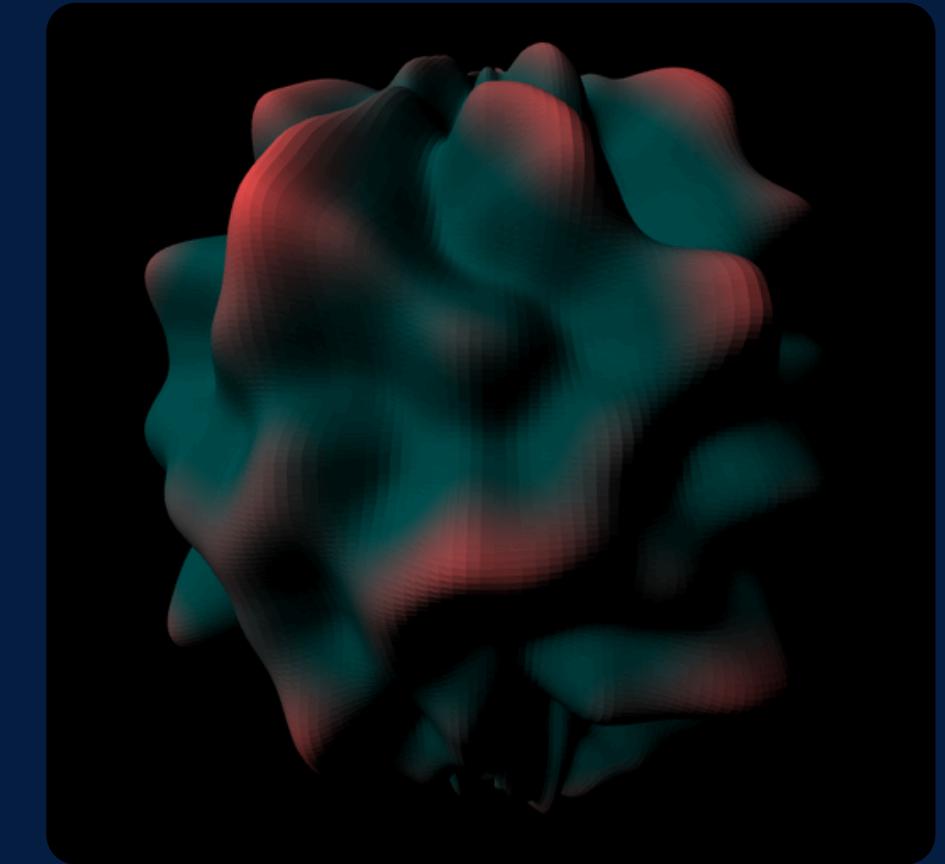
2) WebGL Basics



2) WebGL Basics

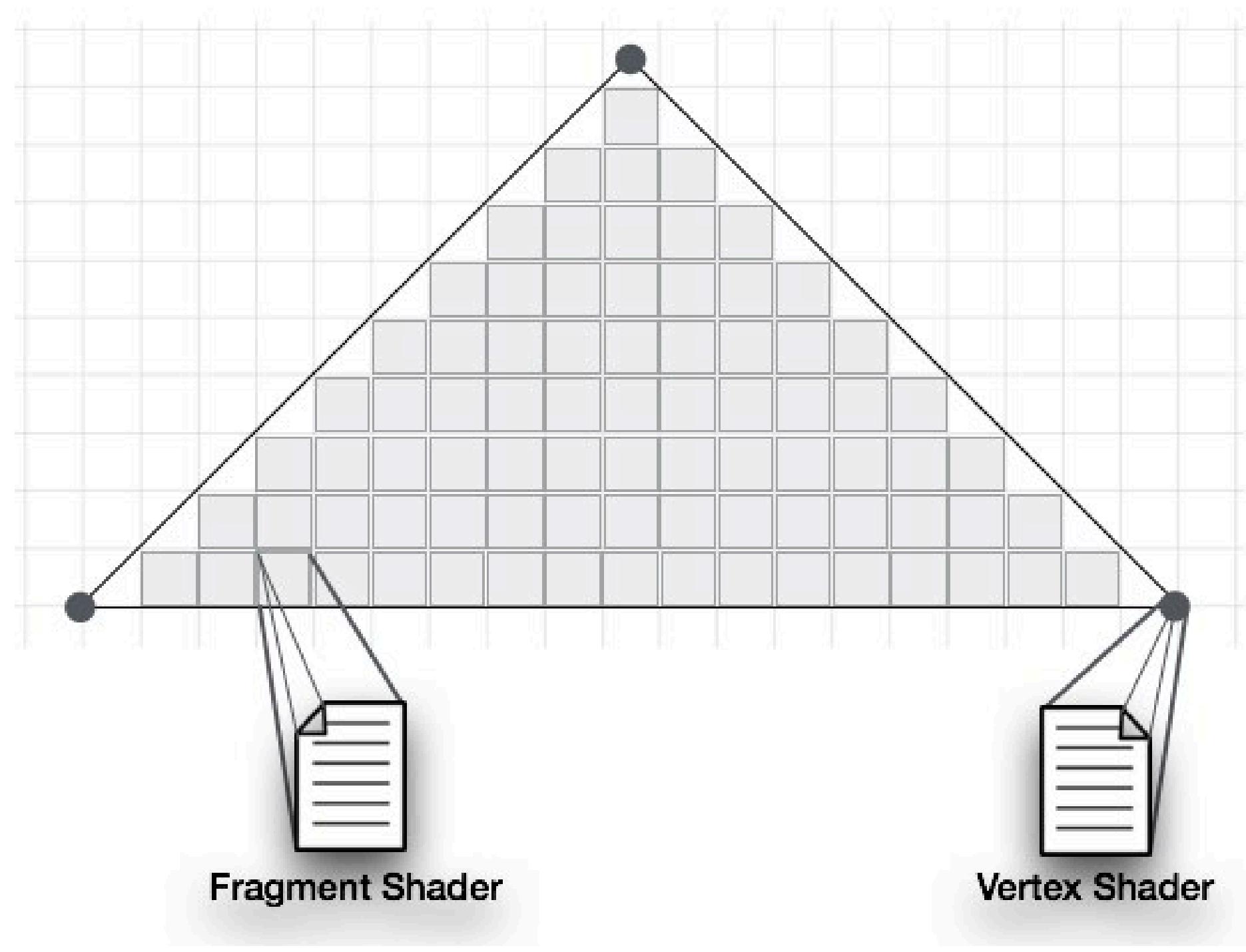


- Shaders: A small icon depicting a grid of colored squares, representing a shader program.
 - GPU acceleration.
 - GLSL.
 - Interaction of components.
 - **Vertex Shader** and **Fragment Shader**.



Shader examples

2) WebGL Basics



3) WebGL 2D and 3D



- Prepare the canvas and get WebGL rendering context.

```
this.canvas = document.getElementById(canvasID) as HTMLCanvasElement;  
this.context = this.canvas.getContext('webgl')!;
```

3) WebGL 2D and 3D

- Define the geometry and store it in buffer objects.

```
const vertices = [-0.5, 0.5, -0.5, -0.5, 0.0, -0.5];
const vertexBuffer = this.context.createBuffer();
this.context.bindBuffer(this.context.ARRAY_BUFFER, vertexBuffer);
this.context.bufferData(this.context.ARRAY_BUFFER, new Float32Array(vertices), this.context.STATIC_DRAW);
this.context.bindBuffer(this.context.ARRAY_BUFFER, null);
```

3) WebGL 2D and 3D

- Create and compile Shader programs.

```
const vertCode =`  
attribute vec2 coordinates;  
void main(void) {  
    gl_Position = vec4(coordinates, 0.0, 1.0);  
}  
  
const vertShader = this.context.createShader(this.context.VERTEX_SHADER)!;  
this.context.shaderSource(vertShader, vertCode);  
this.context.compileShader(vertShader);  
const fragCode =`  
void main(void) {  
    gl_FragColor = vec4(0.0, 0.0, 0.0, 0.1);  
}  
  
const fragShader = this.context.createShader(this.context.FRAGMENT_SHADER)!;  
this.context.shaderSource(fragShader, fragCode);  
this.context.compileShader(fragShader);
```

3) WebGL 2D and 3D

- Create and compile Shader programs.

```
const shaderProgram = this.context.createProgram()!;  
this.context.attachShader(shaderProgram, vertShader);  
this.context.attachShader(shaderProgram, fragShader);  
this.context.linkProgram(shaderProgram);  
  
this.context.useProgram(shaderProgram);
```

3) WebGL 2D and 3D

- Associate the shader programs with buffer objects.

```
this.context.bindBuffer(this.context.ARRAY_BUFFER, vertexBuffer);
const coord = this.context.getAttribLocation(shaderProgram, "coordinates");
this.context.vertexAttribPointer(coord, 2, this.context.FLOAT, false, 0, 0);
this.context.enableVertexAttribArray(coord);
```

3) WebGL 2D and 3D



- Drawing the required object.

```
this.context.clearColor(0.5, 0.5, 0.5, 0.9);
this.context.enable(this.context.DEPTH_TEST);
this.context.clear(this.context.COLOR_BUFFER_BIT);
this.context.viewport(0, 0, this.canvas.width, this.canvas.height);
this.context.drawArrays(this.context.TRIANGLES, 0, 3);
```

3) WebGL 2D and 3D



- Result:



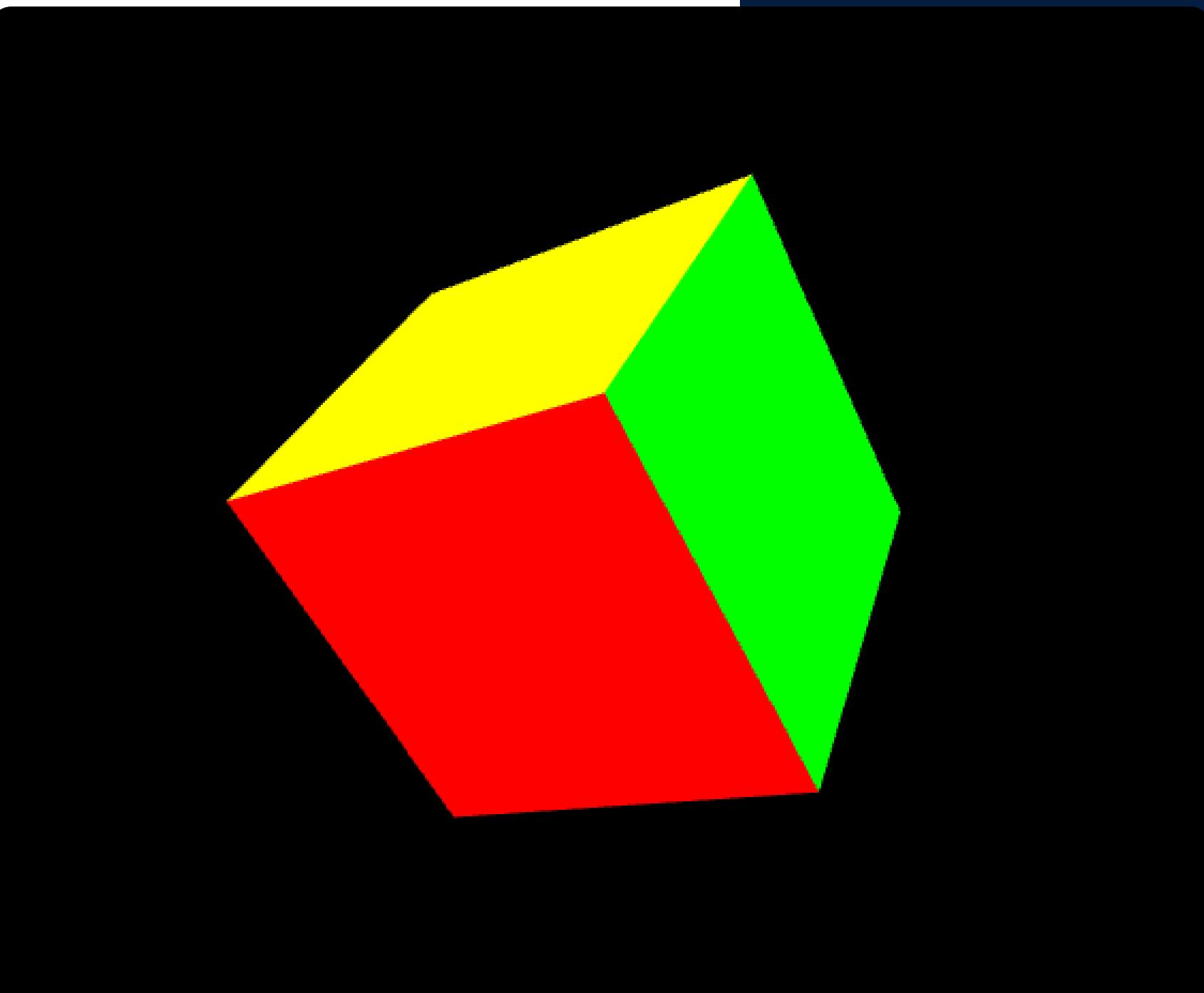
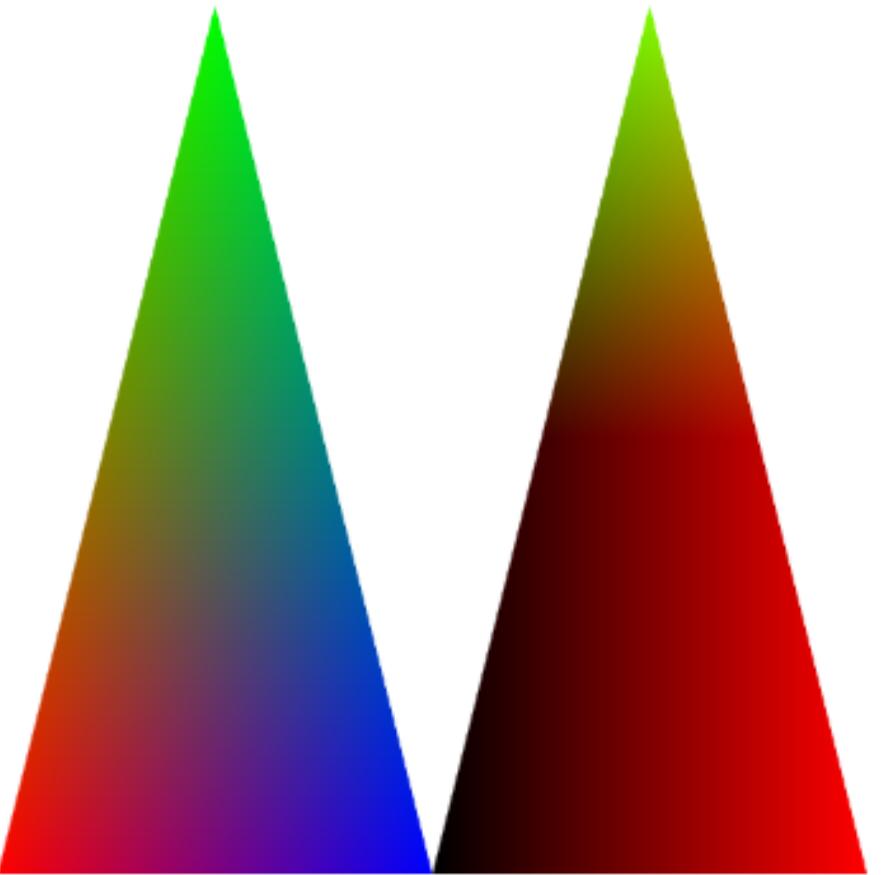
3) WebGL 2D and 3D

```
const positions = [
    // Front face
    -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0,
    // Back face
    -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0,
    // Top face
    -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0,
    // Bottom face
    -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0,
    // Right face
    1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0,
    // Left face
    -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0,
];
```

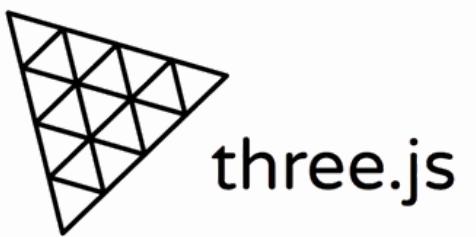
3) WebGL 2D and 3D



```
const colors = new Float32Array([
  1.0, 0.0, 0.0, // Red
  0.0, 1.0, 0.0, // Green
  0.0, 0.0, 1.0, // Blue
])
```

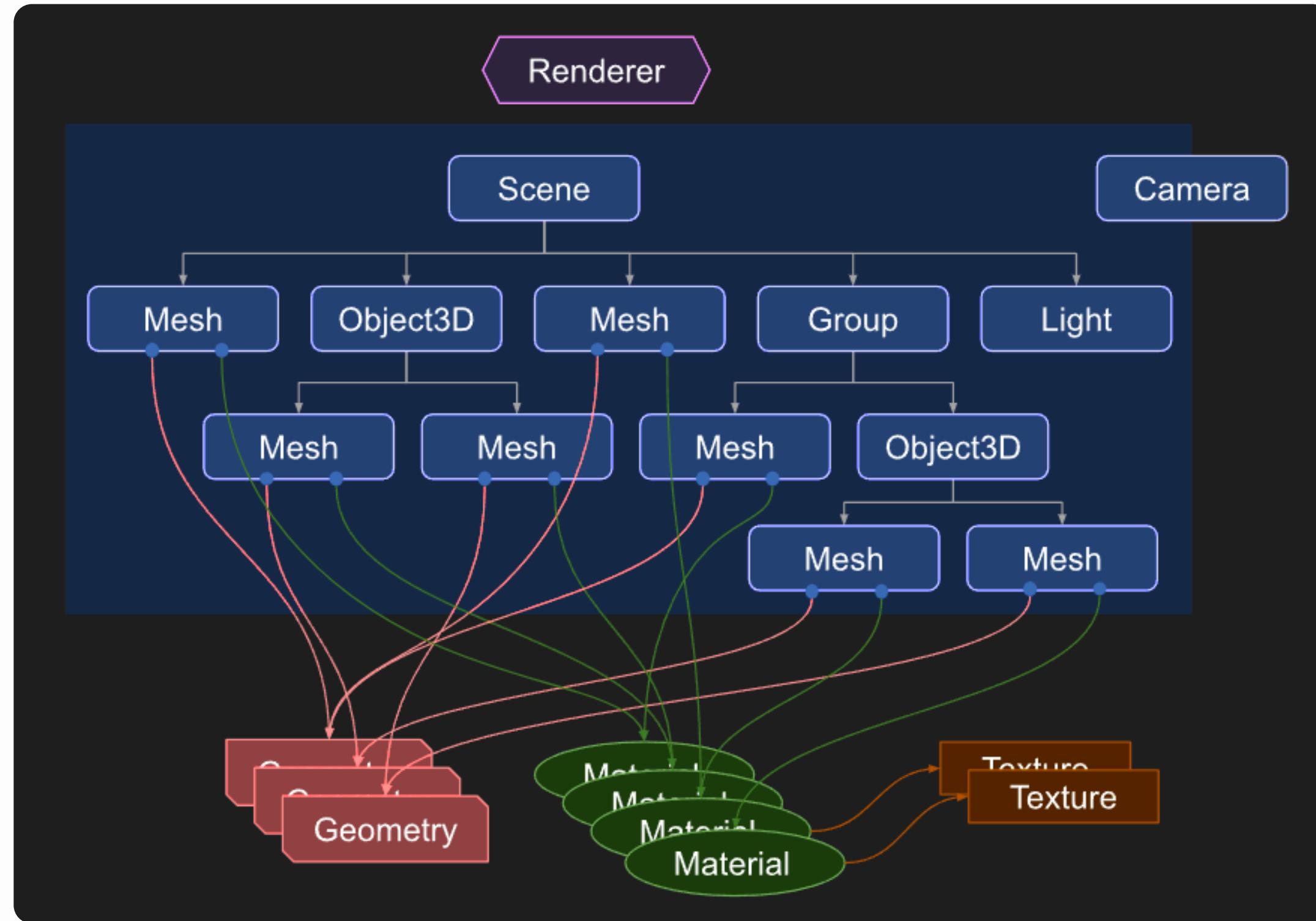


4) Three.js introduction



What is Three.js?

6) Three.js structure



7) First scene



- 1) • import * as THREE from "three";
• const scene = new THREE.Scene();
- 2) • const camera = new THREE.PerspectiveCamera(75,
window.innerWidth / window.innerHeight, 0.1, 1000);

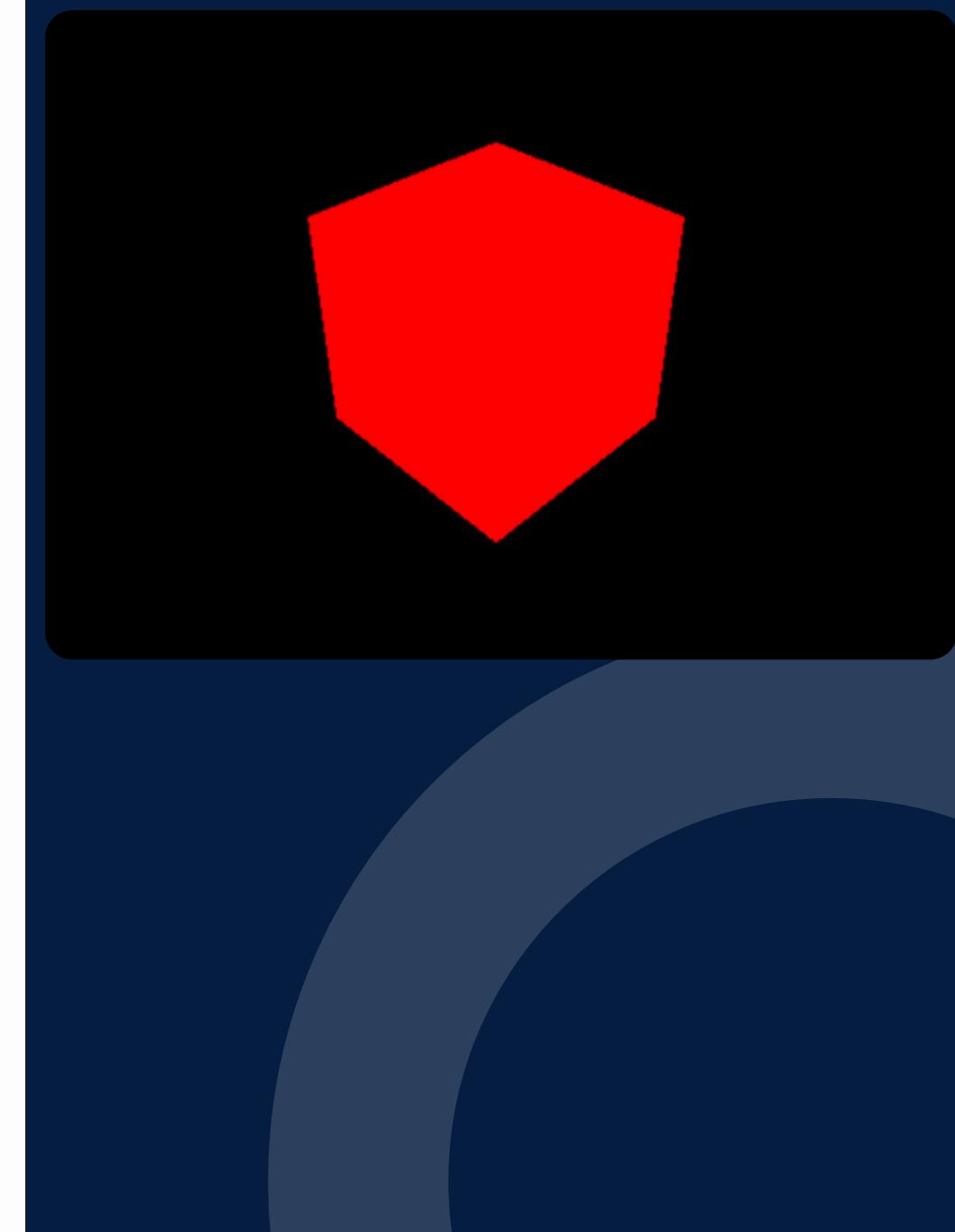
• const renderer = new THREE.WebGLRenderer();
• renderer.setSize(window.innerWidth,
window.innerHeight);
• document.body.appendChild(renderer.domElement);
• renderer.render(scene, camera);
- 3)

7) First scene



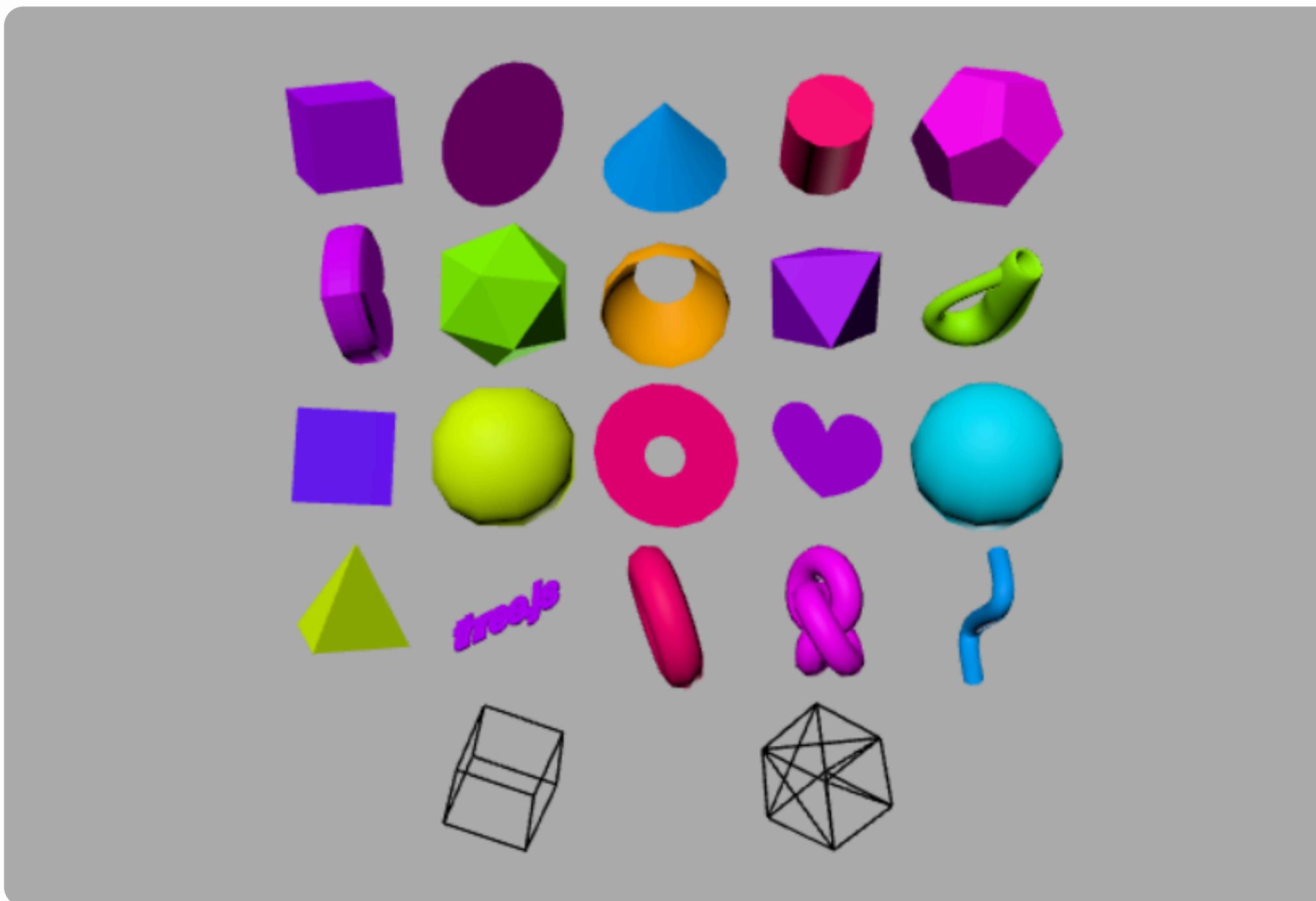
- 4) • const geometry = new THREE.BoxGeometry(1, 1, 1);
- 5) • const material = new THREE.MeshBasicMaterial({
color: "red"});
- 6) • const cube = new THREE.Mesh(geometry, material);
• scene.add(cube);

• camera.position.set(5, 5, 5); // Posición de la cámara
- 7) (x, y, z)
• camera.lookAt(0, 0, 0); // Enfocar la cámara en el
cubo



8) Primitive figures

What are primitive figures?



8) Primitive figures

Some primitives:

```
new THREE.BoxGeometry(1, 1, 1),  
new THREE.SphereGeometry(0.5, 32, 32),  
new THREE.ConeGeometry(0.5, 1, 32),  
new THREE.CylinderGeometry(0.5, 0.5, 1, 32),  
new THREE.TorusGeometry(0.5, 0.2, 16, 100),  
new THREE.TetrahedronGeometry(0.5),  
new THREE.OctahedronGeometry(0.5),  
new THREE.IcosahedronGeometry(0.5),  
new THREE.DodecahedronGeometry(0.5),  
new THREE.PlaneGeometry(1, 1),  
new THREE.CircleGeometry(0.5, 32),
```

8) Primitive figures

- **Extrusion:**

```
const shape = new THREE.Shape();
shape.moveTo(0, 0);
shape.lineTo(1, 0);
shape.lineTo(1, 1);
shape.lineTo(0, 1);
shape.lineTo(0, 0);
```

```
const extrudeSettings = { depth: 1, bevelEnabled: true,
  bevelThickness: 0.2, bevelSize: 0.2, bevelSegments: 1 };
let extrudeGeometry = new THREE.ExtrudeGeometry(shape, extrudeSettings);
```

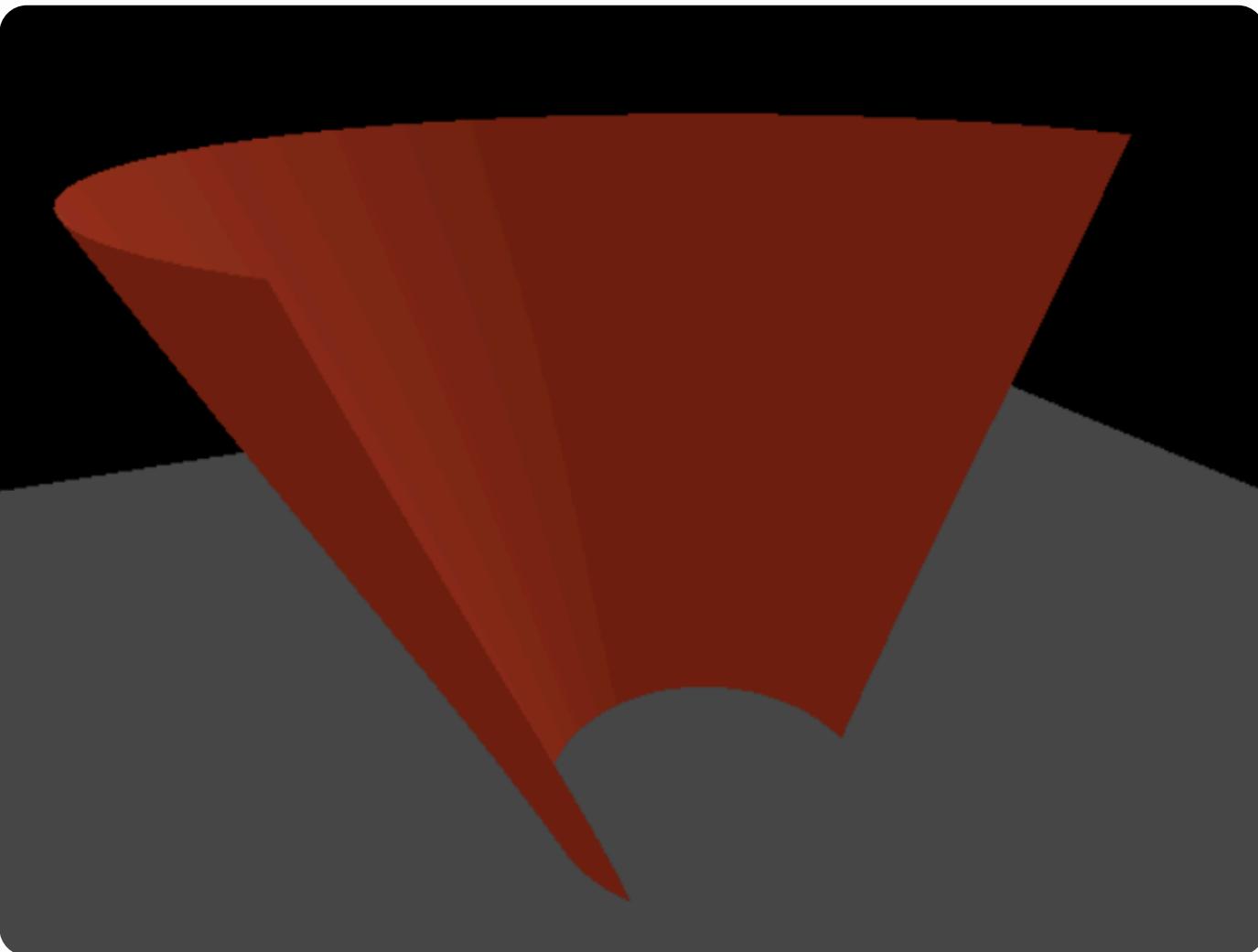


8) Primitive figures



- **Lathe:**

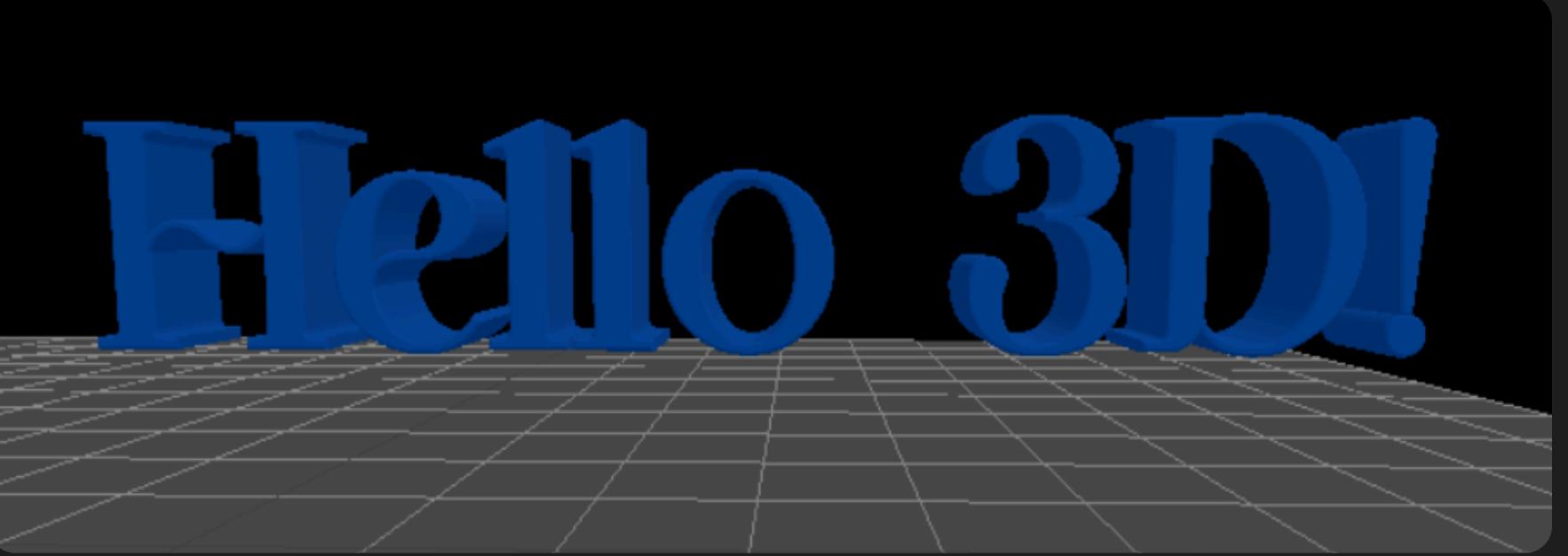
```
const radialPoints: THREE.Vector2[] = [];
let latheGeometry = new THREE.LatheGeometry(radialPoints, 32);
```



8) Primitive figures

- Text:

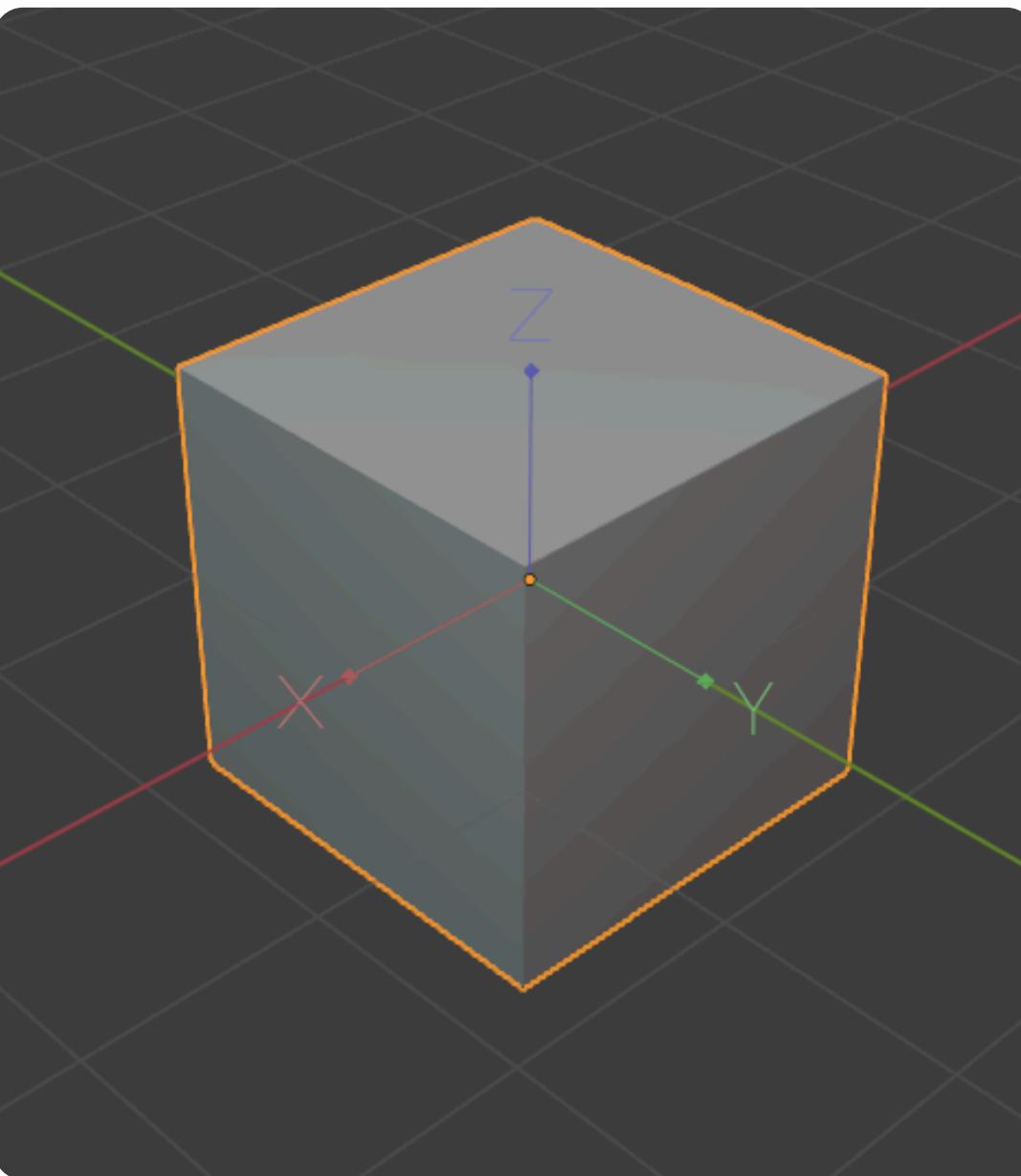
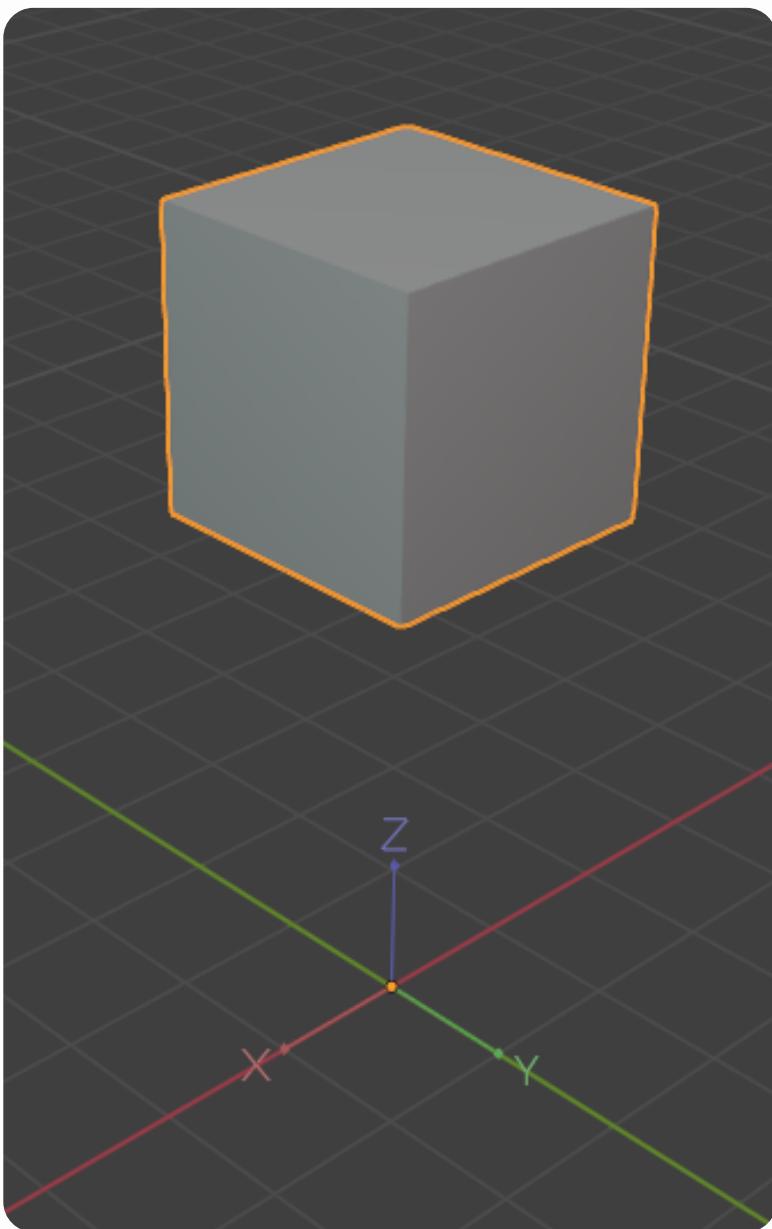
```
const loader = new FontLoader();
loader.load('http://localhost:3000/src/figures/Hefana_Regular.json', (font) => {
  const textGeometry = new TextGeometry('Hello 3D!', {
    font: font,
    size: 1,
    depth: 0.5,
    curveSegments: 12,
    bevelEnabled: true,
    bevelThickness: 0.03,
    bevelSize: 0.02,
    bevelSegments: 5,
  });
});
```



8) Primitive figures

Important property:

- The center of geometry.



9) Materials

- What are materials?



9) Materials

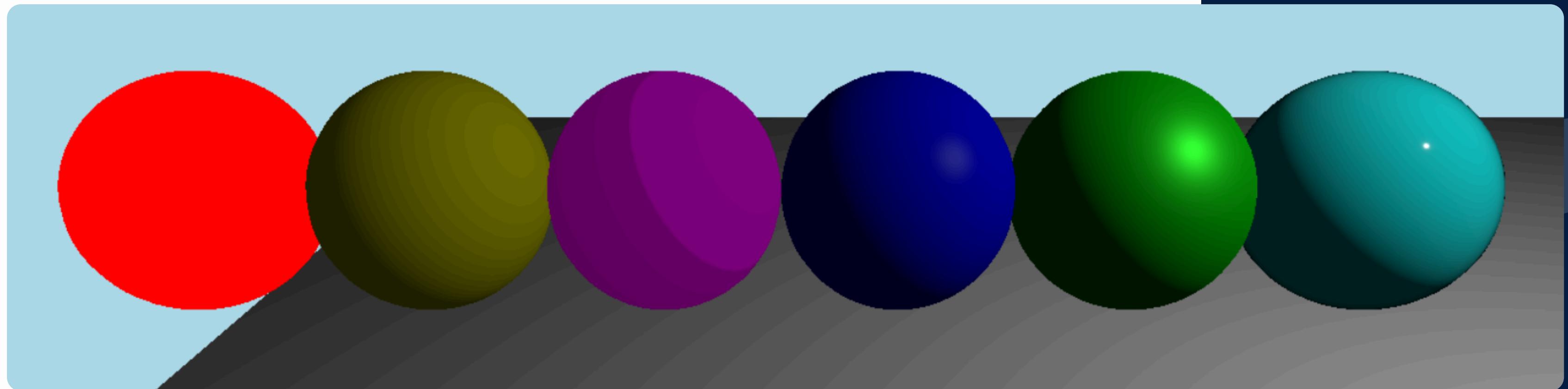
- Material declaration:

```
const material = new THREE.MeshBasicMaterial({  
    color: 'brown',  
    wireframe: true,  
});  
  
material.side = THREE.DoubleSide;  
material.color = new THREE.Color('lightblue');
```

9) Materials

- Principal materials:

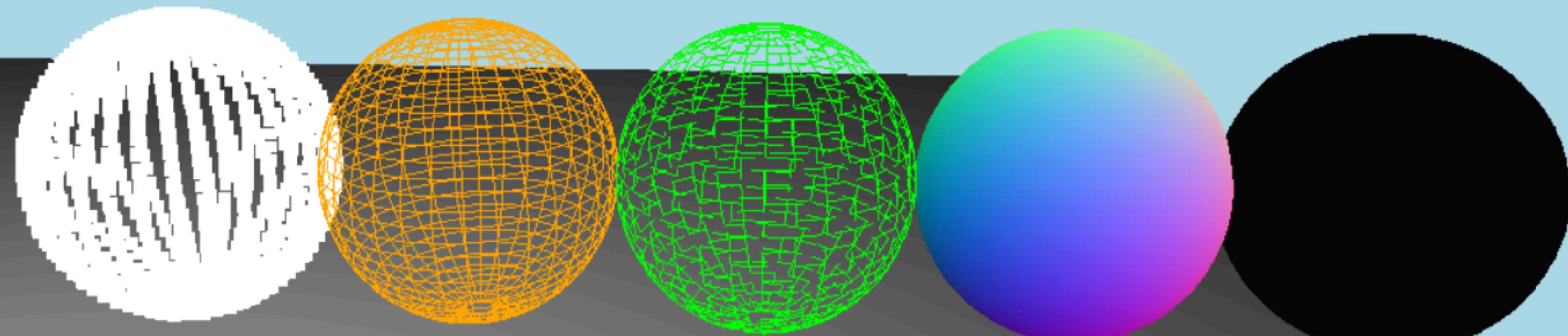
```
new THREE.MeshBasicMaterial({  
new THREE.MeshLambertMaterial({  
new THREE.MeshToonMaterial({  
new THREE.MeshPhongMaterial({  
new THREE.MeshStandardMaterial({  
new THREE.MeshPhysicalMaterial({
```



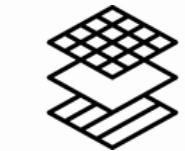
9) Materials

- Other materials:

```
new THREE.ShadowMaterial({ opacity: 0.5 });
new THREE.PointsMaterial({ color: 0x0000ff, size: 20, sizeAttenuation: true });
new THREE.LineBasicMaterial({ color: 0x0000ff });
new THREE.LineDashedMaterial({ color: 0x0000ff, dash: [5, 5] });
new THREE.MeshNormalMaterial(),
new THREE.MeshDepthMaterial(),
```

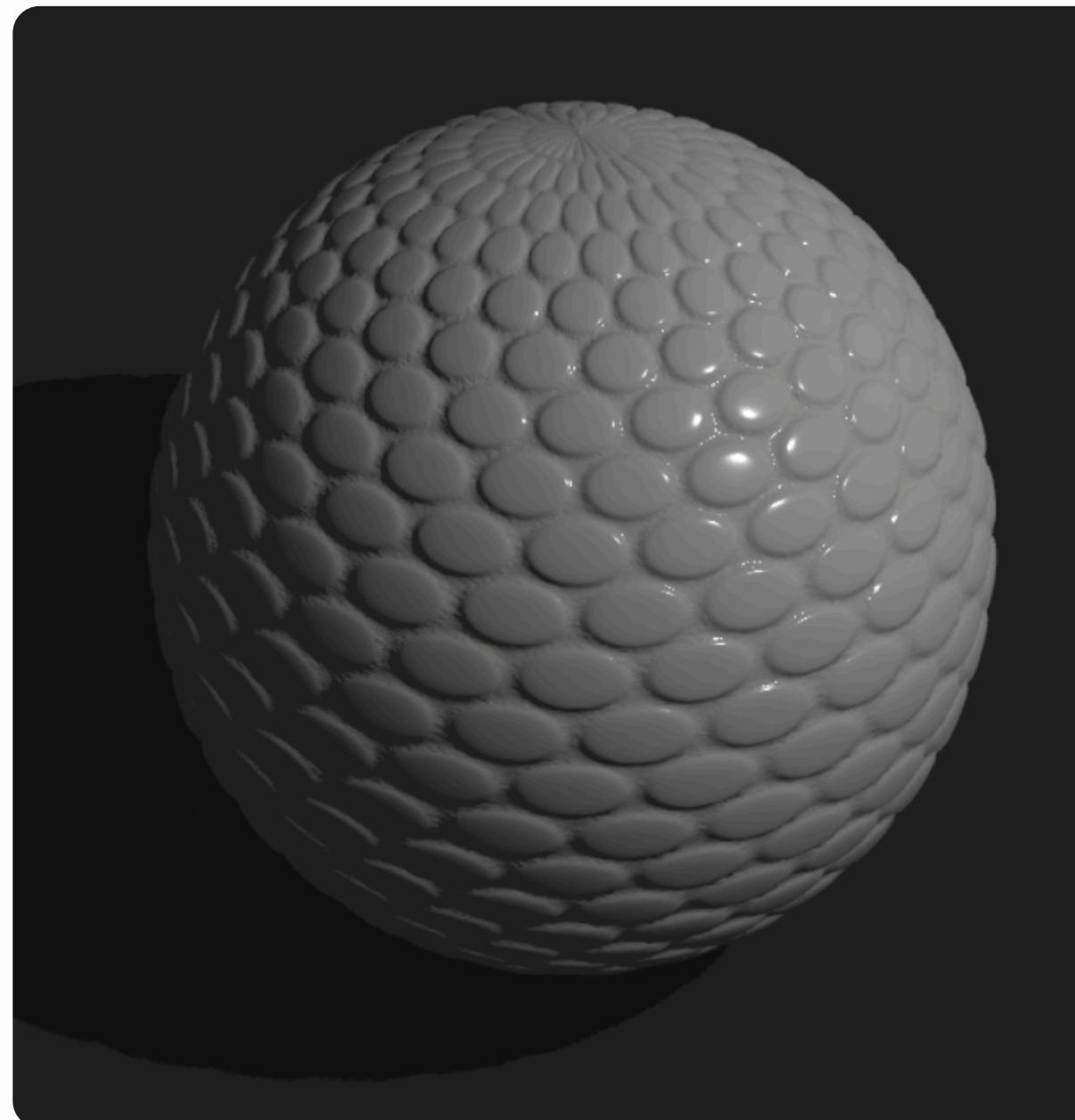


9) Textures



```
const textureLoader = new THREE.TextureLoader();
```

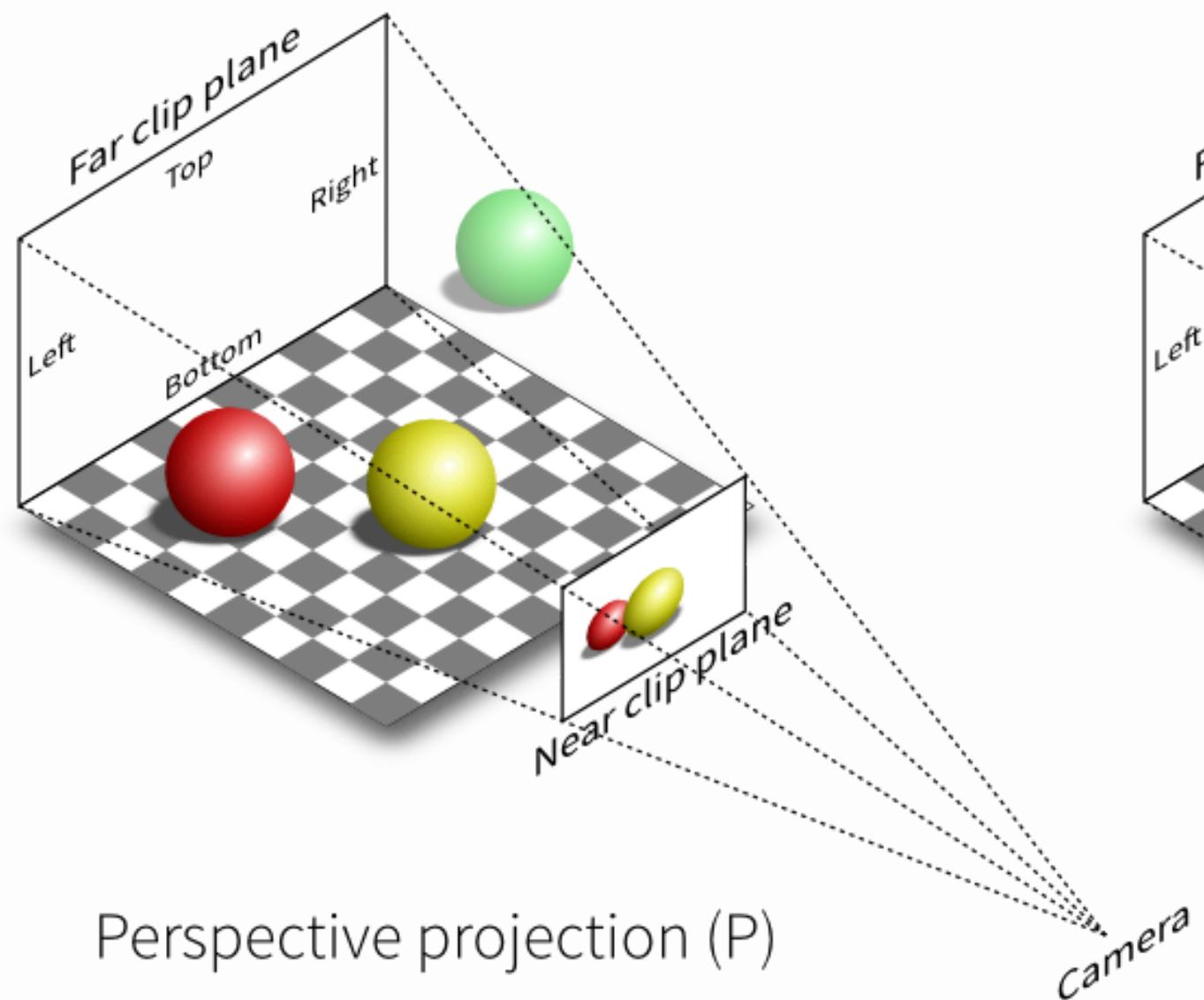
```
const colorTexture: THREE.Texture = textureLoader.load('file.png');
```



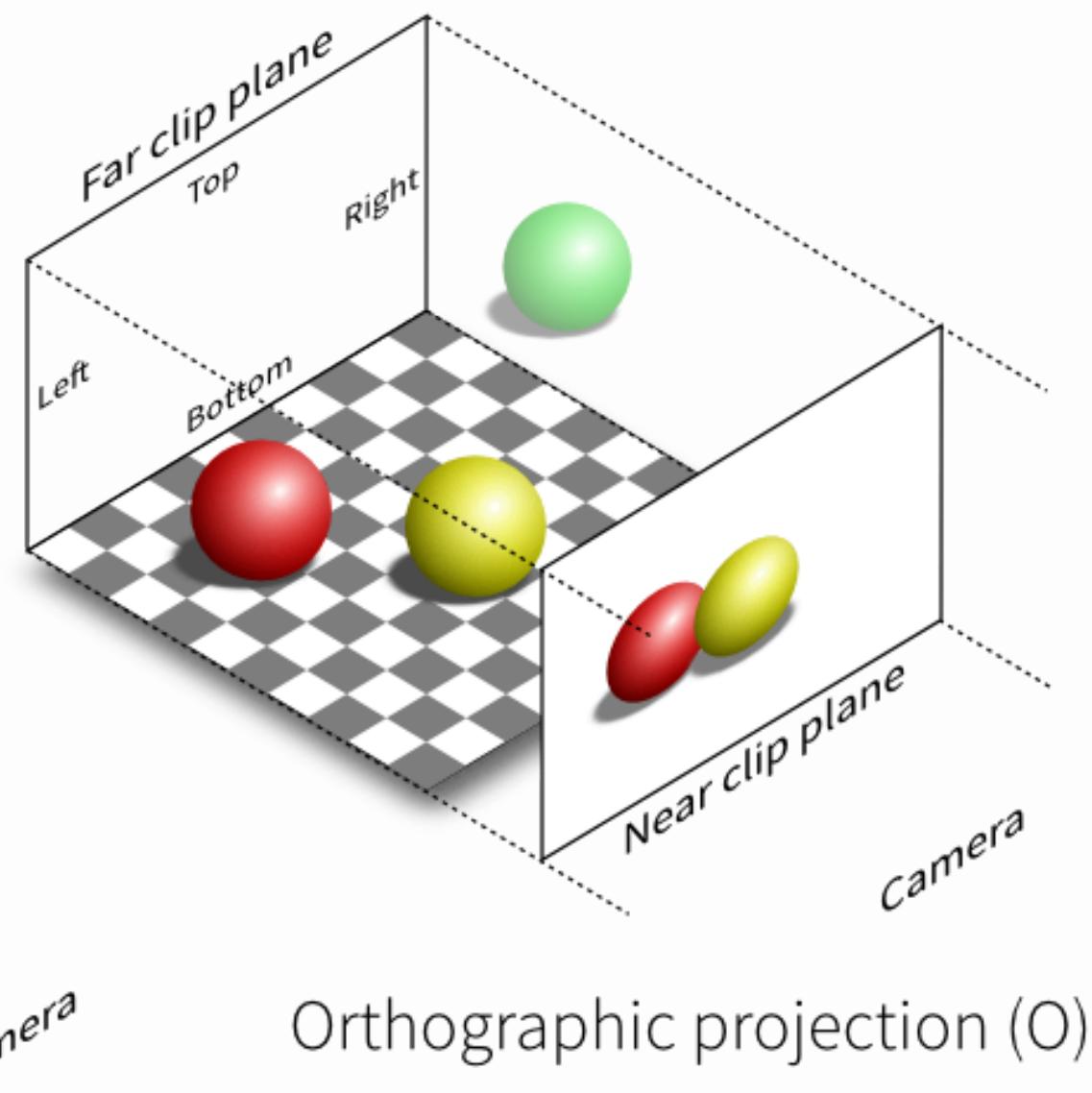
```
material.map = colorTexture;  
material.normalMap = normalTexture;  
material.roughnessMap = roughnessTexture;  
material.displacementMap = displacementTexture;
```

10) Cameras

The view of the scene



Perspective projection (P)

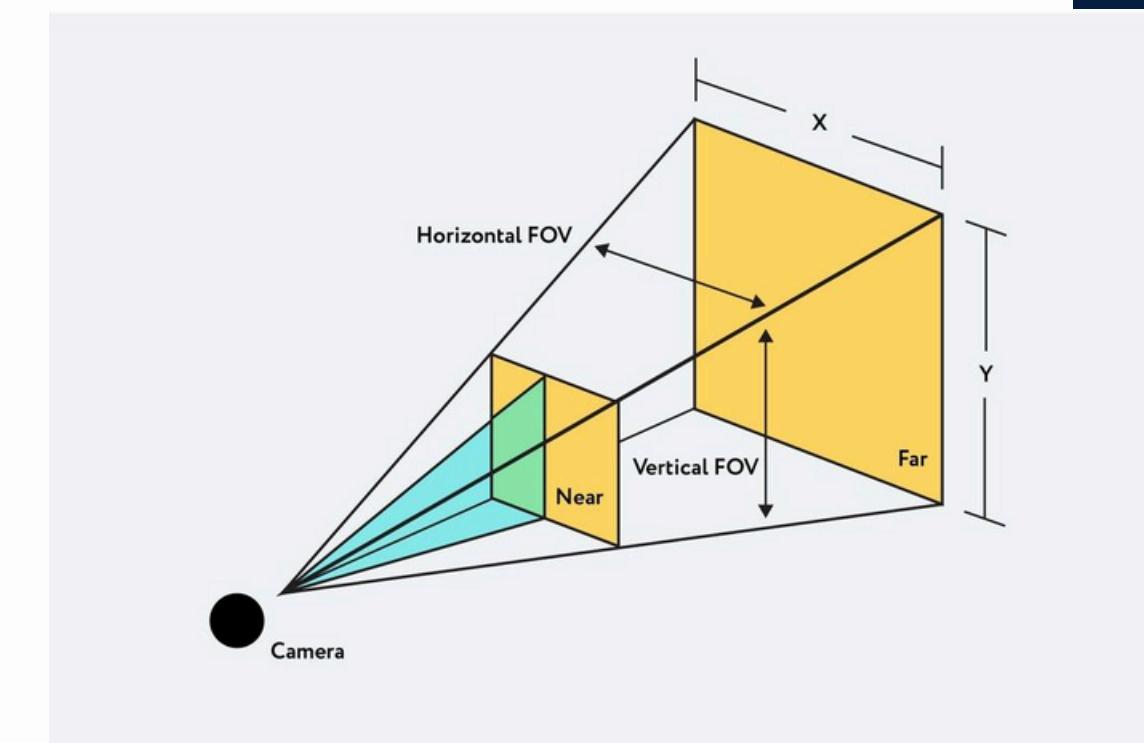
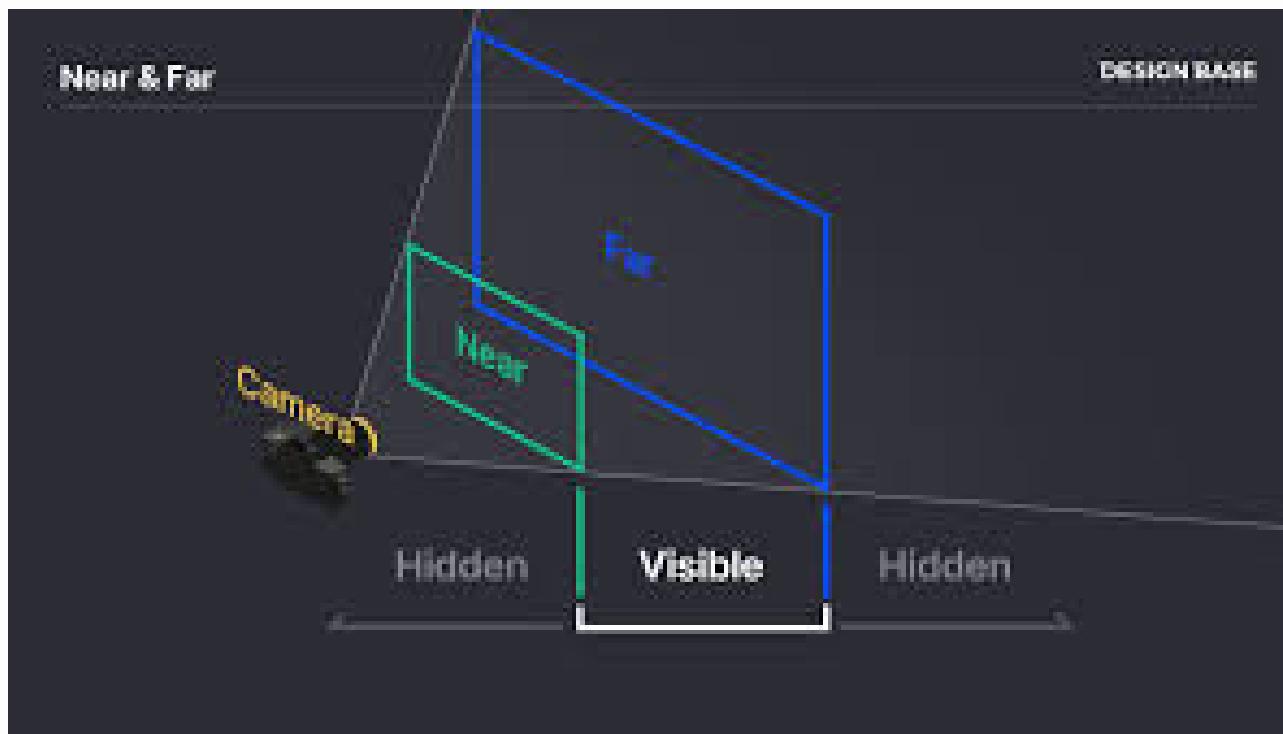


Orthographic projection (O)

10) Cameras

Main properties of cameras:

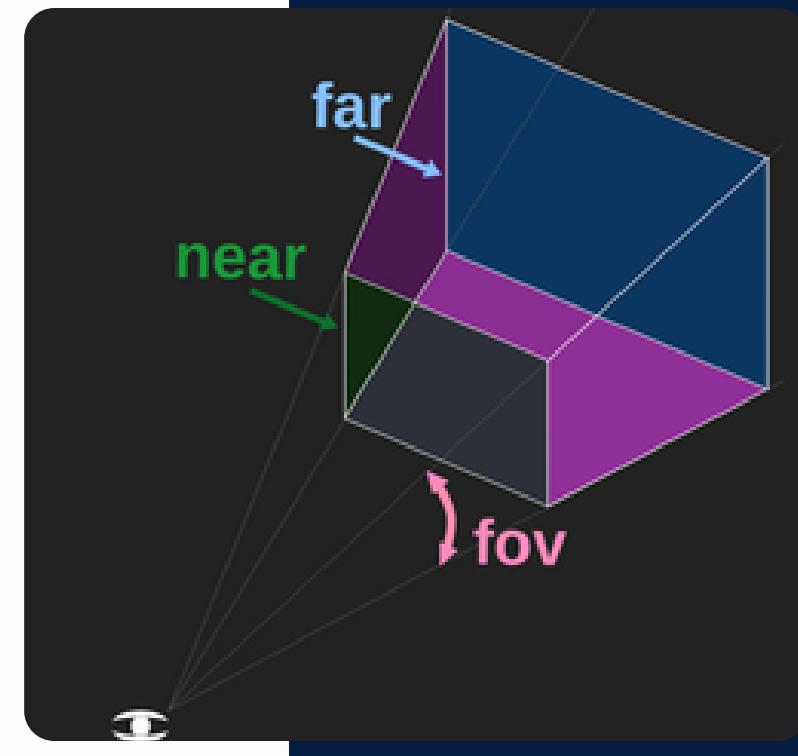
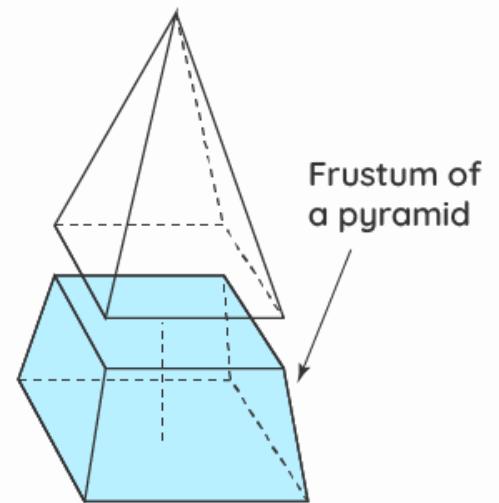
- Near
- Far



10) Cameras

Perspective camera:

- Defined with a frustum (depth).
- Fov: field of view in degrees.
- Aspect : width / height relationship.



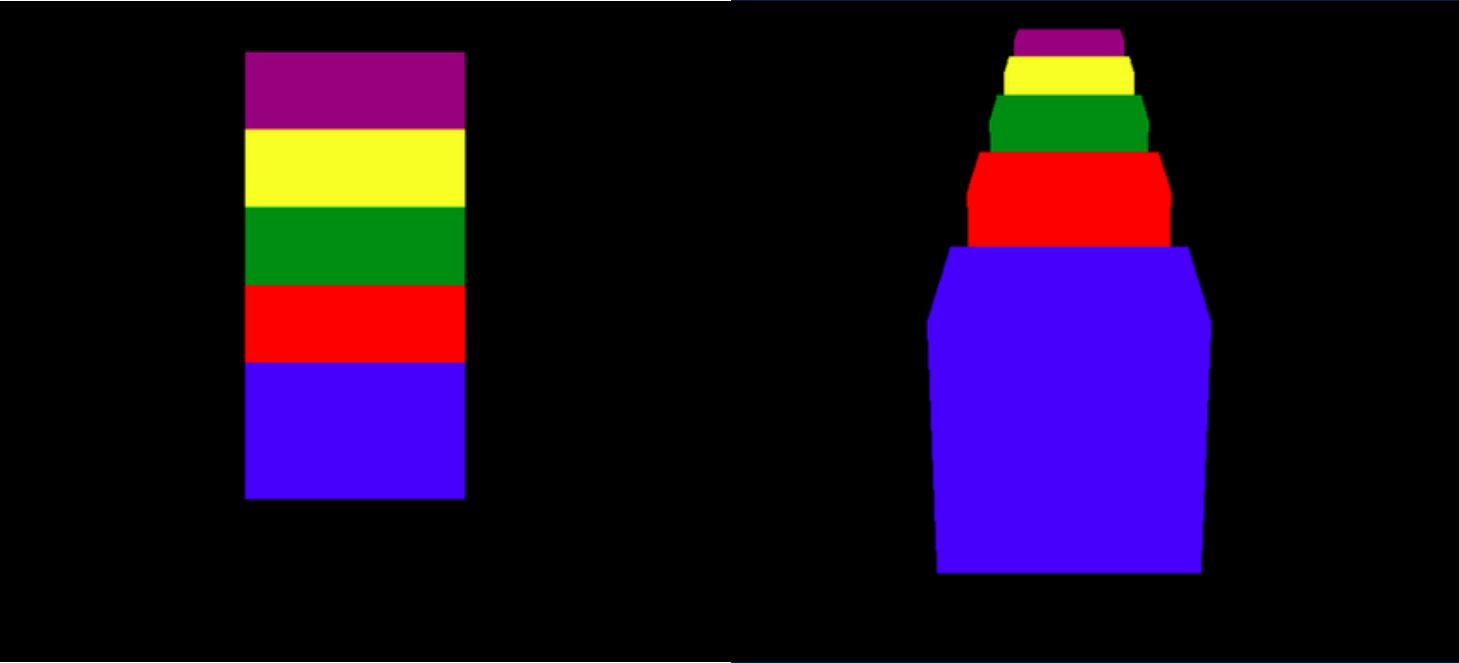
```
const fov = 75;
const aspect = window.innerWidth / window.innerHeight;
const near = 0.1;
const far = 500;
const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
```

Interactive application of perspective camera

10) Cameras

Orthographic camera:

- Defined with a box, (2D view).
- Left, right, top, bottom: Defines visible limits of the box.



```
const near = 0.1;
const far = 500;
const left = -1;
const right = 1;
const top = 1;
const bottom = -1;
const camera = new THREE.OrthographicCamera(left, right, top, bottom, near, far);
```

[Interactive application of ortographic camera](#)

10) Camera



Orbit controls:

- Allow you to manipulate the camera for the main types of movements:
- Rotate: rotate the camera around a focus point.
- Zoom: zoom in and out .

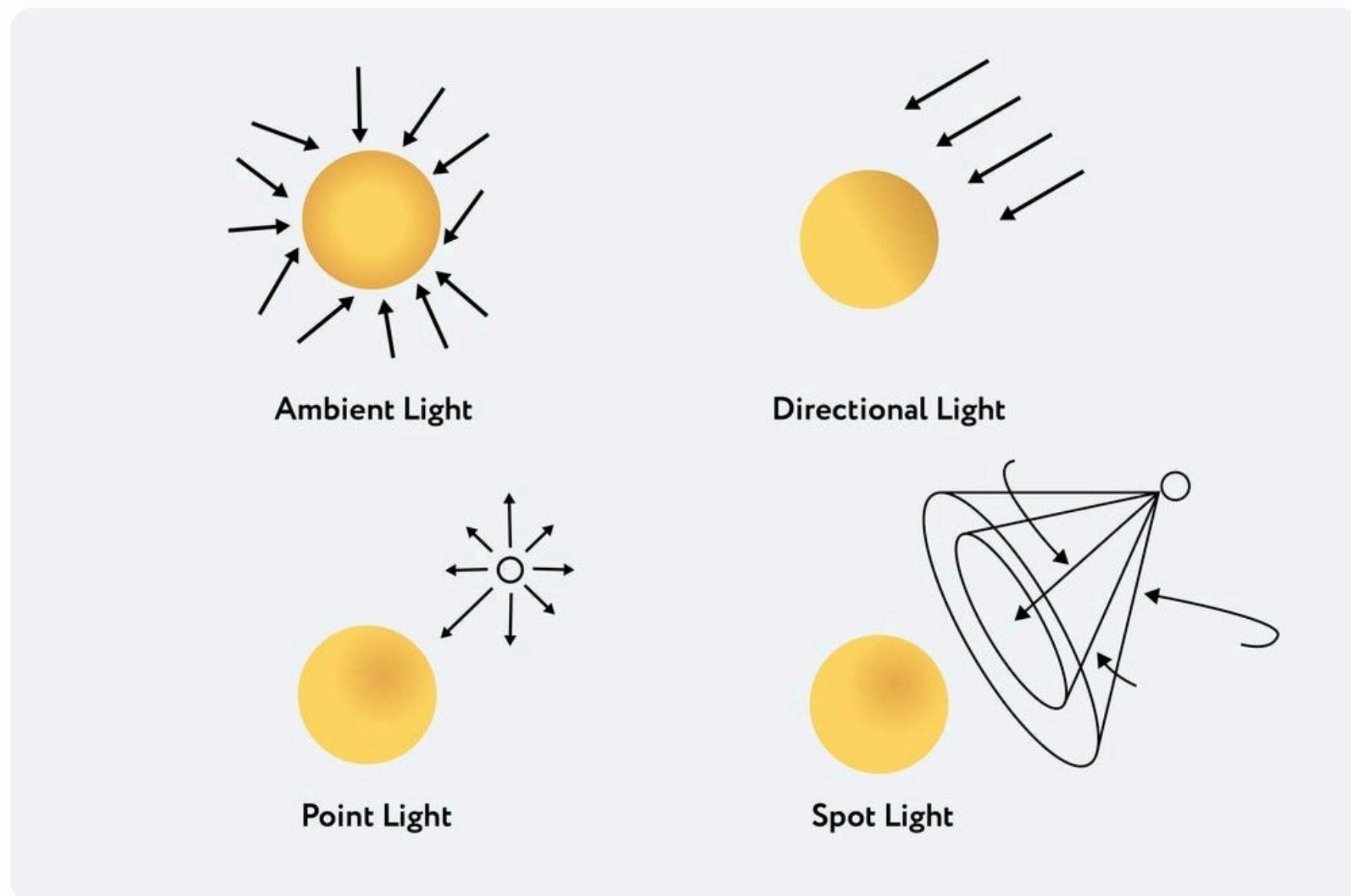
```
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js';
```

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

11) Lights



Responsible for lighting the scene

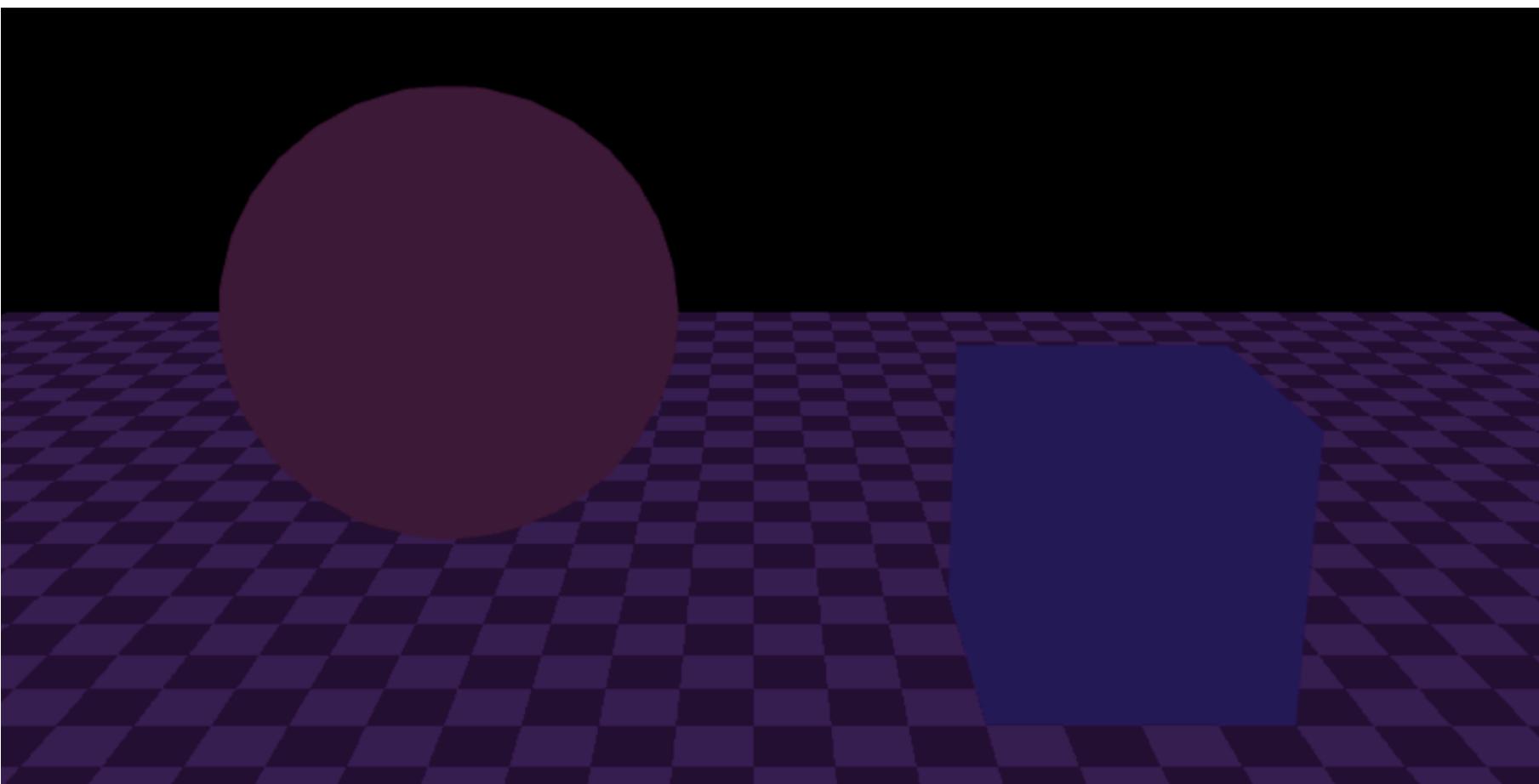


11) Lights



Main properties of lights:

- color
- intensity

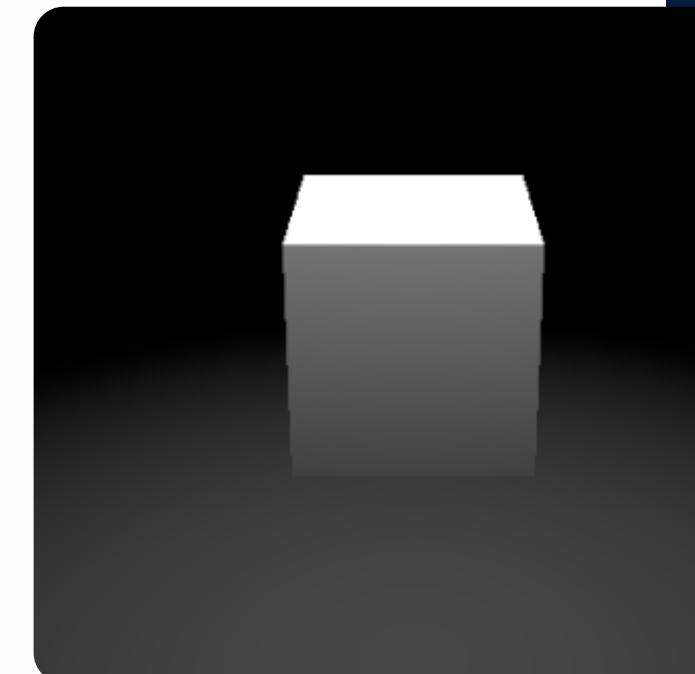
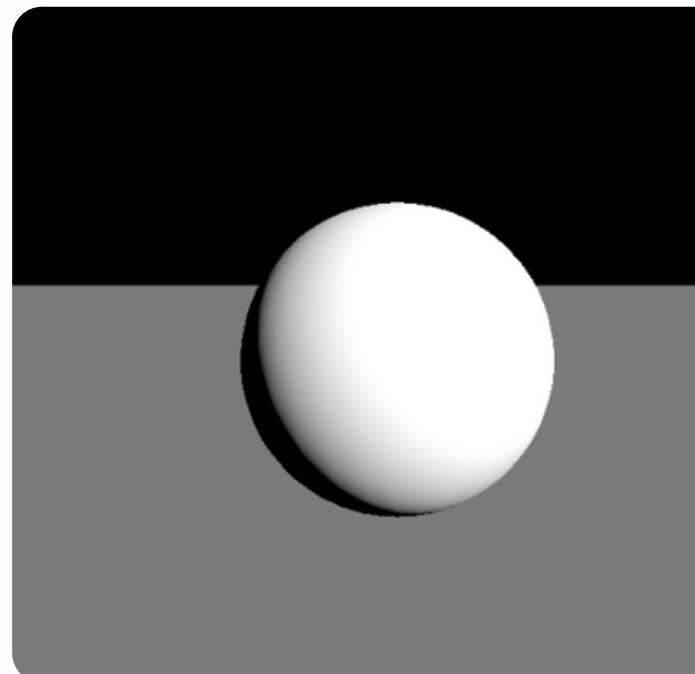


11) Lights



Some type of lights:

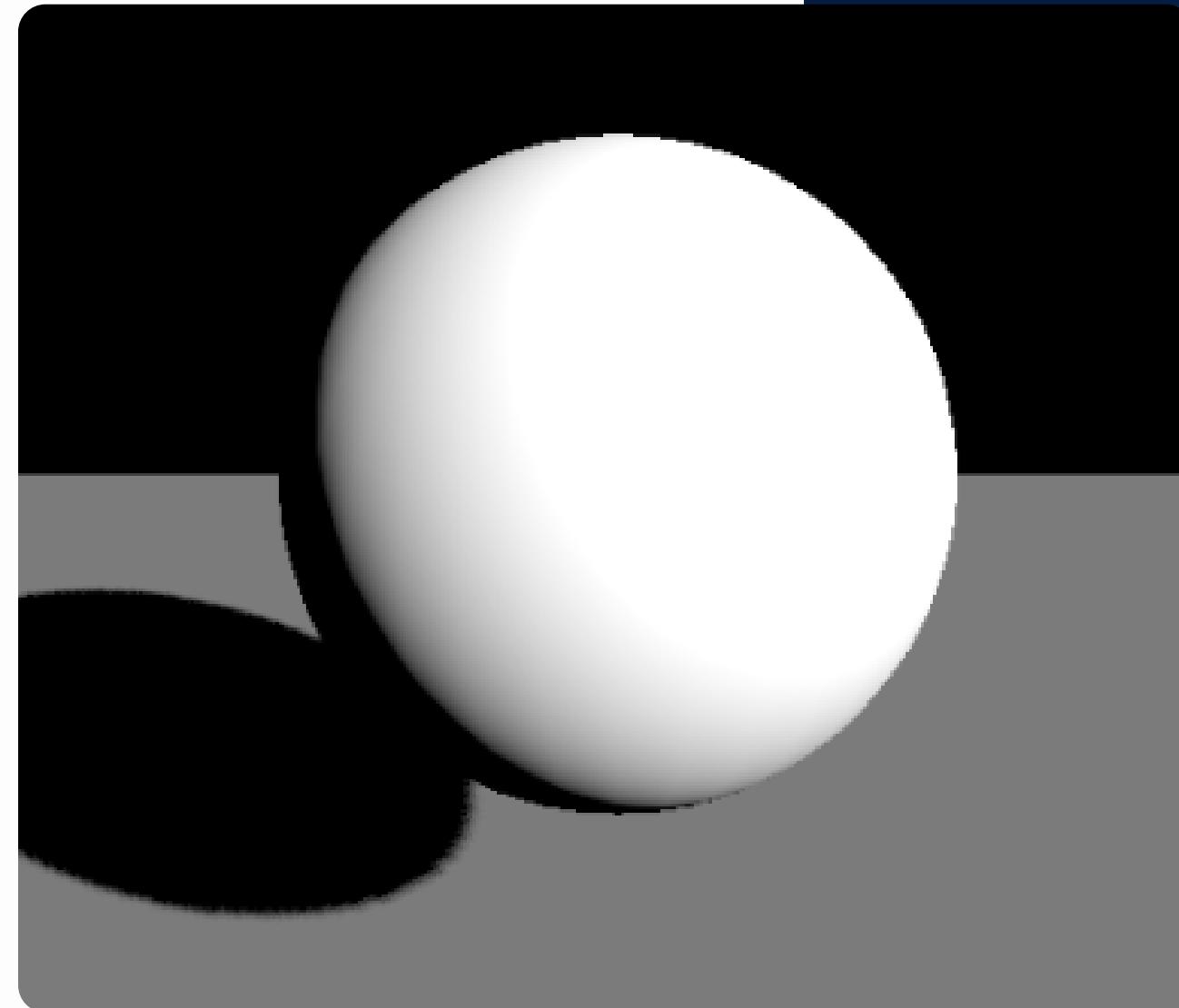
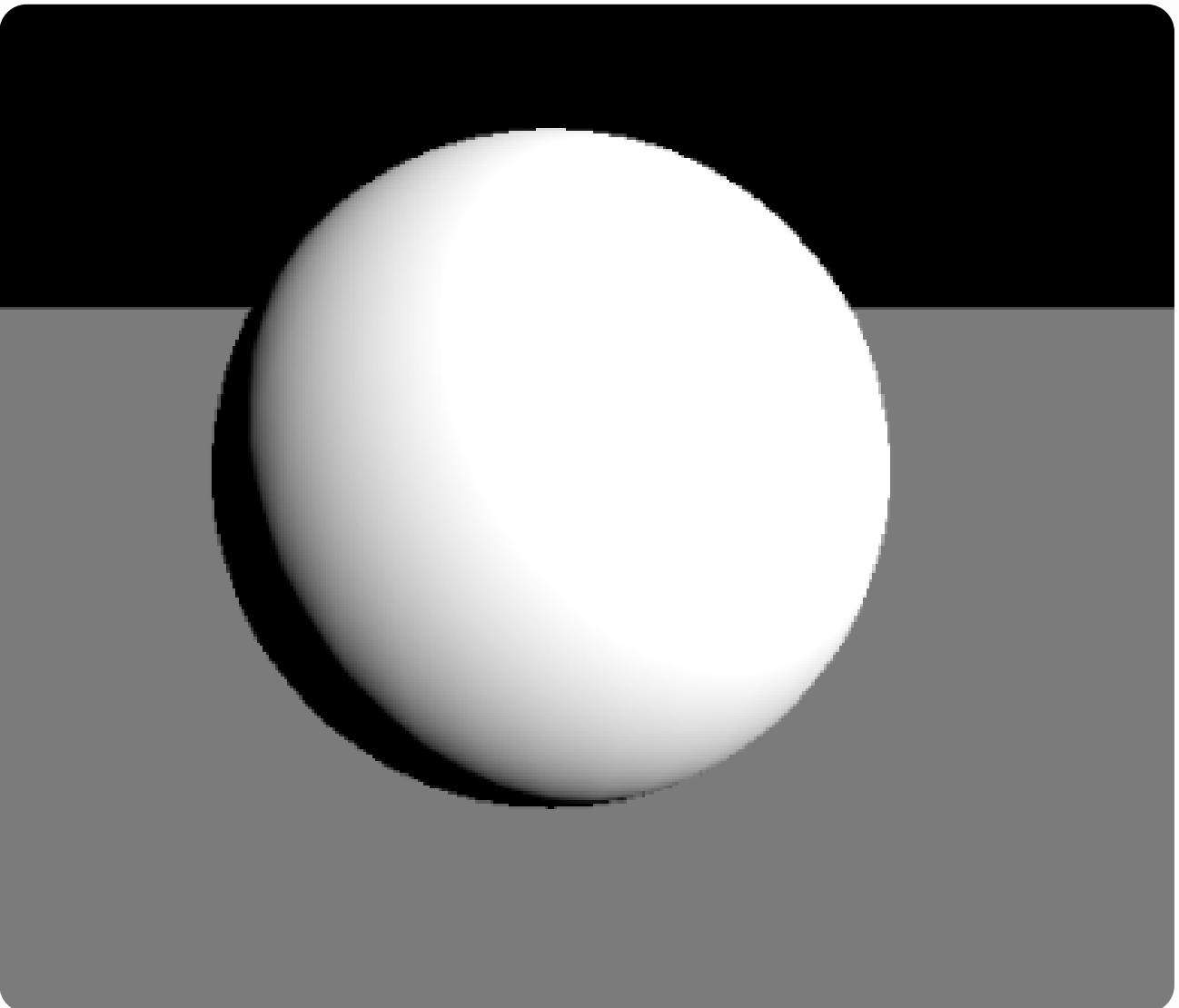
- Ambient. [Interactive application of ambient light](#)
- Directional. [Interactive application of directional light](#)
- Hemispheric. [Interactive application of hemispherical light](#)
- Point. [Interactive application of point light](#)
- Spot. [Interactive application of spot light](#)



12) Shadows



Difference between a lighting effect and a shadow in Three.js.



12) Shadows



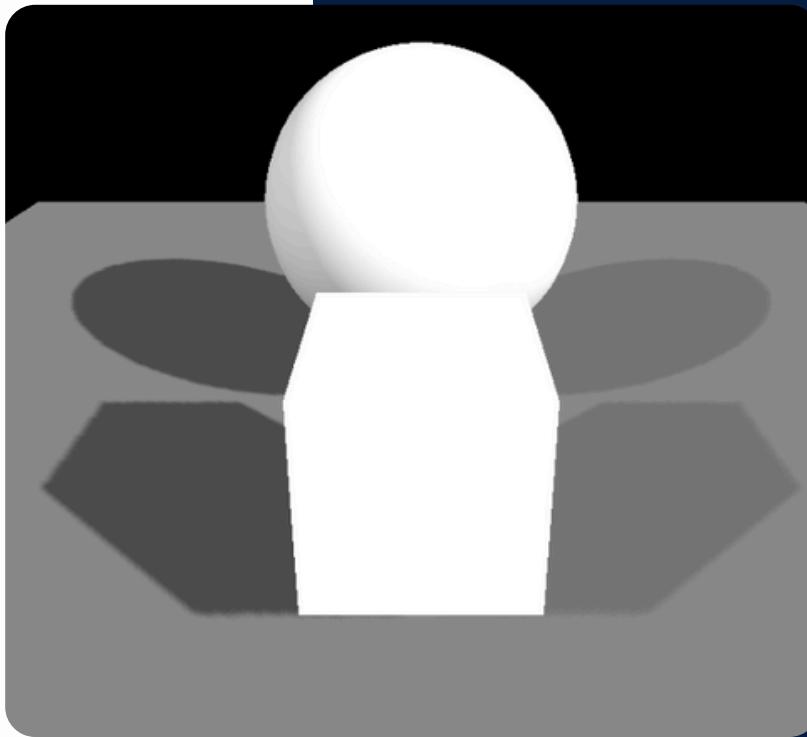
Steps for enable shadows:

1. Enable shadowMap at renderer.
2. Enable light cast shadows.
3. Enable Objects cast or receive shadows

Some type of shadows (Set at thre renderer):

- BasicShadowMap.
- PCFShadowMap.
- PCFSOFTShadowMap

Interactive application of Shadows

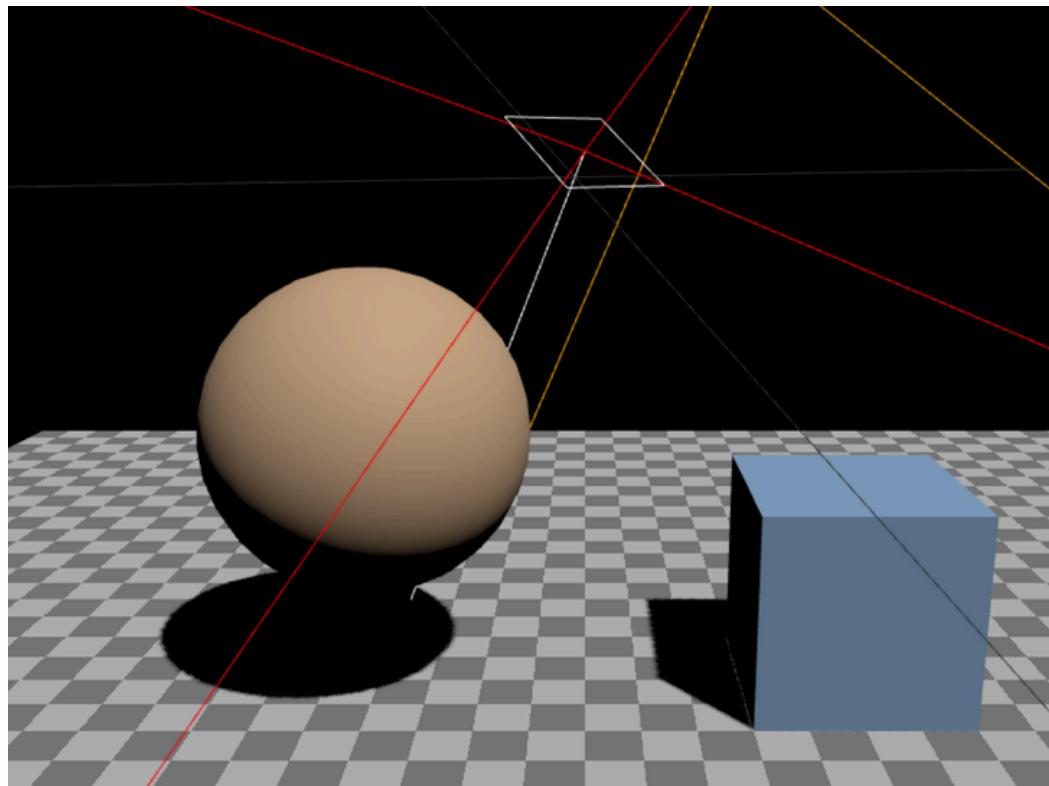
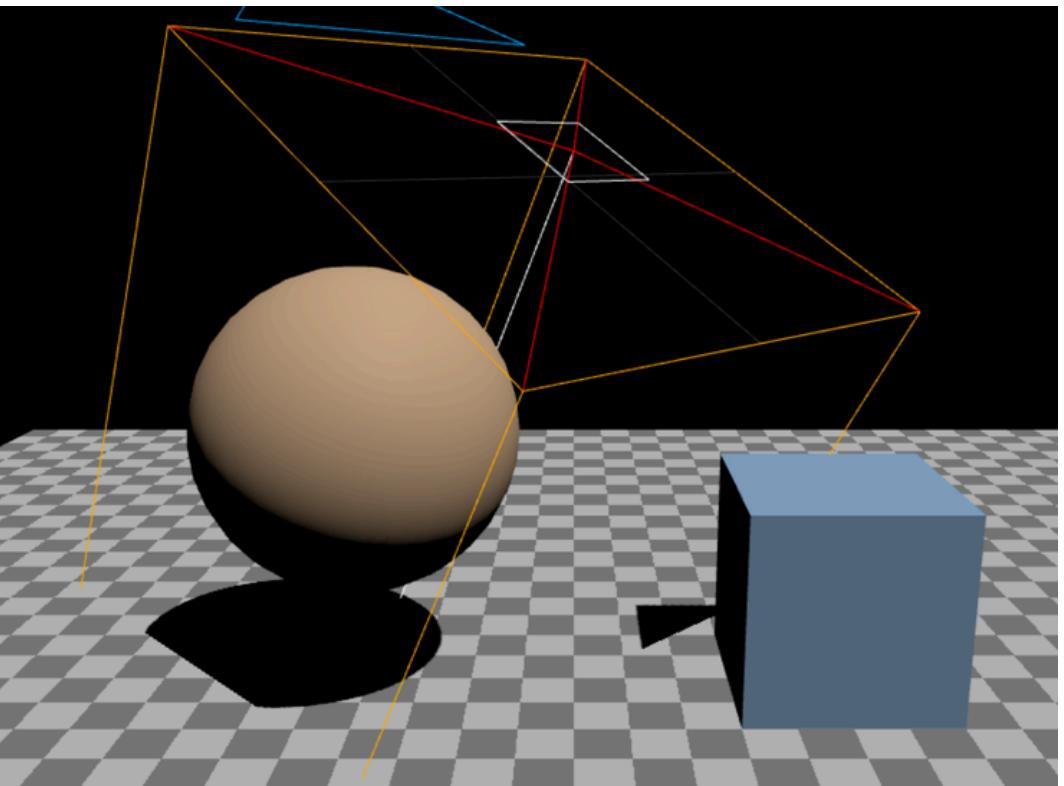


12) Shadows



Light shadow camera:

Is the shadow range generated by the light.



Interactive application of shadows camera

12) Shadows

Not all lights and materials generates shadows:

Light	Cast Shadows
Ambient	✗
Directional	✓
Hemispheric	✗
Spot	✓
Point	✓

Material	Cast Shadows
Basic	✗
Phong	✓
Lambert	✓
Standard	✓
Physical	✓

13) Loading OBJ and FBX



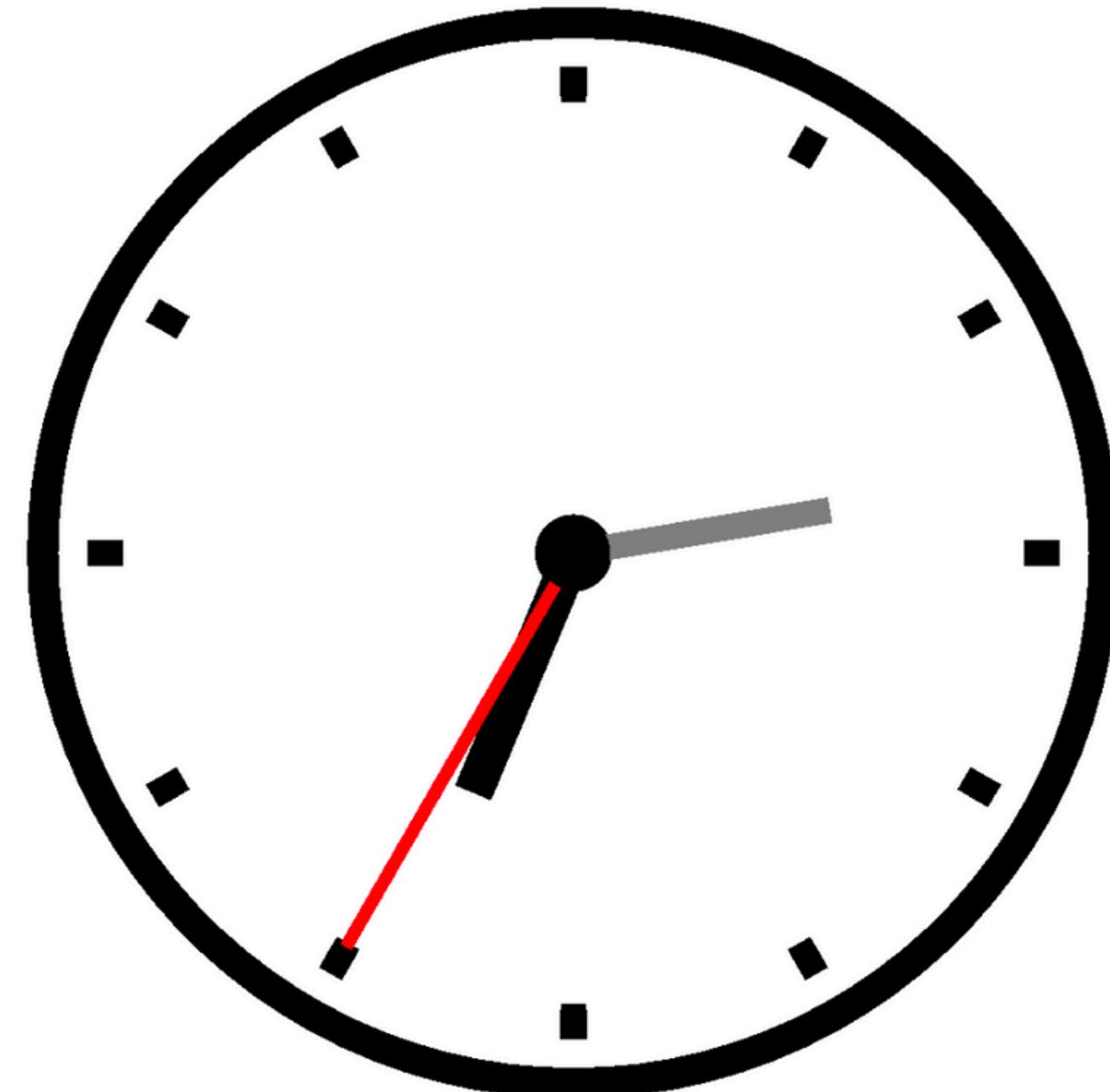
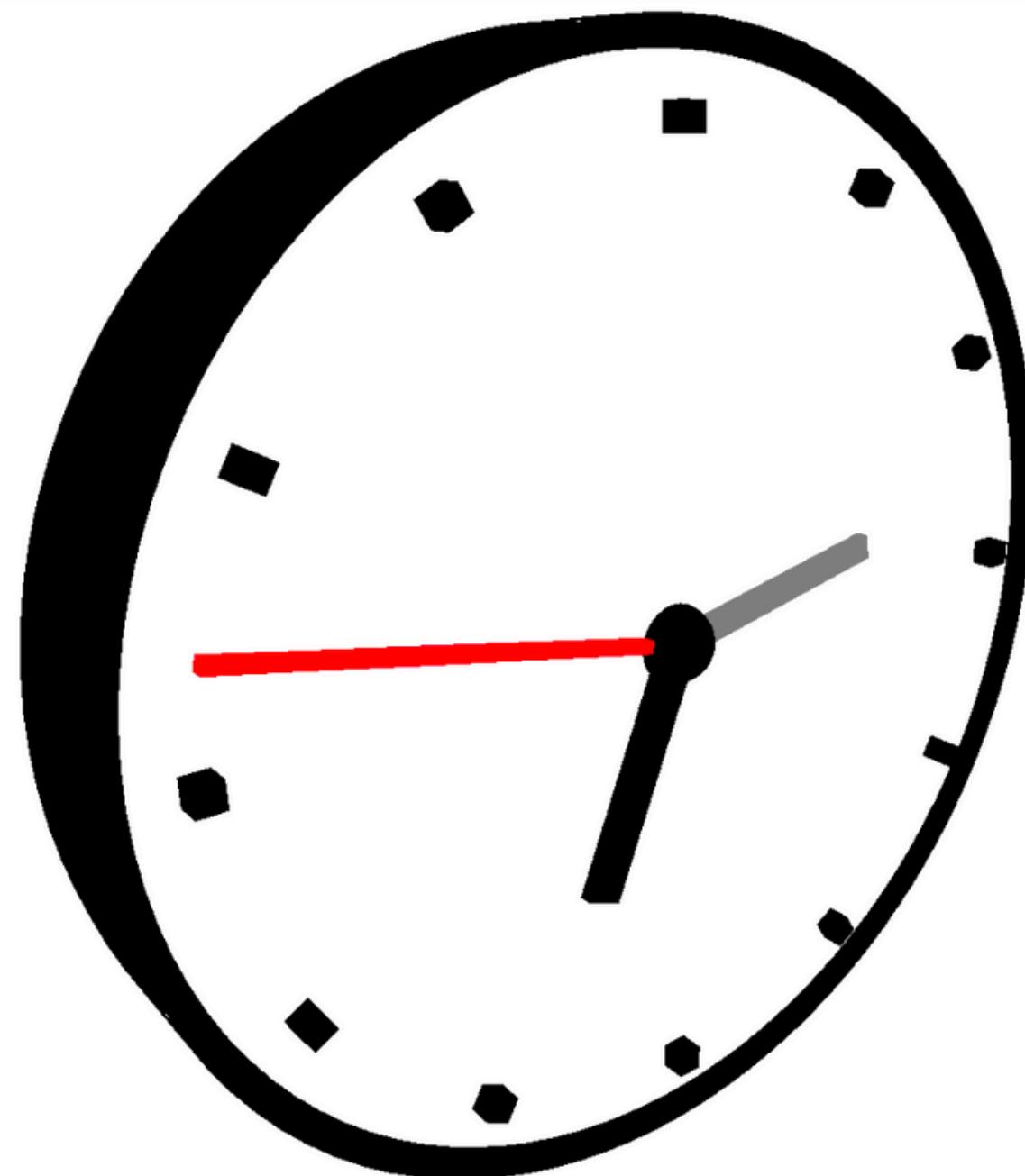
```
import { OBJLoader } from 'three/examples/jsm/loaders/OBJLoader.js';
import { FBXLoader } from 'three/examples/jsm/loaders/FBXLoader.js';

const loader = new FBXLoader();
loader.load('Mercedes-model/Mercedes.fbx', (fbx) => {
  fbx.scale.set(0.05, 0.05, 0.05);
  fbx.position.set(-7, 0, 0);
  scene.add(fbx);
});

const objLoader = new OBJLoader();
objLoader.load('./Mercedes-model/Mercedes.obj', (obj) => {
  scene.add(obj);
});
```



14) Clock at Three.js



Clock at three js

REFERENCES:

- Wikipedia WebGL: <https://en.wikipedia.org/wiki/WebGL>
- WebGL tutorial: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial
- 3D Graphics: A WebGL Tutorial: <https://www.toptal.com/javascript/3d-graphics-a-webgl-tutorial>
- WebGL Sample Application: https://www.tutorialspoint.com/webgl/webgl_sample_application.htm
- WebGL - Shaders: https://www.tutorialspoint.com/webgl/webgl_shaders.htm

REFERENCES:

- Rotating Cube in WebGL: <https://github.com/mdn/dom-examples/tree/main/webgl-examples/tutorial/sample5>
- Animating Textures in WebGL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Animating_textures_in_WebGL
- Three JS documentation: <https://threejs.org/docs/>
- Three JS video tutorial: <https://www.youtube.com/watch?v=Y4PvekQUIqk&list=PLDIIzmccetSPVF3JN6OFazp39N00yHE3A&index=3>

thank you