WebGL and Three.js

Team

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

roberto.padron.31@ull.edu.es

alu0101551395@ull.edu.es

Overview

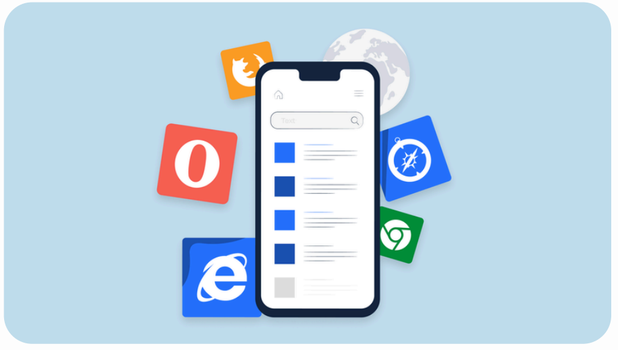
1) WebGL Introduction

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

Advantages of WebGL

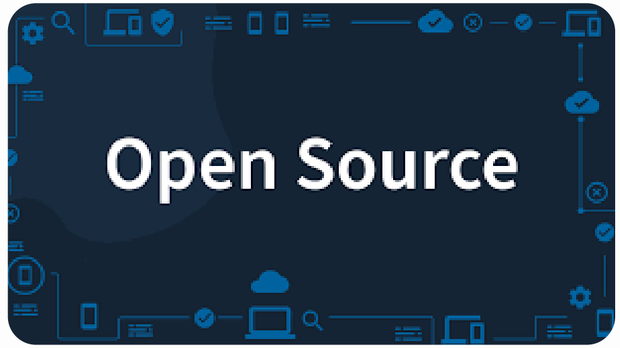
JavaScript programming

Write applications in JavaScript, interact with HTML elements, and use JavaScript libraries.



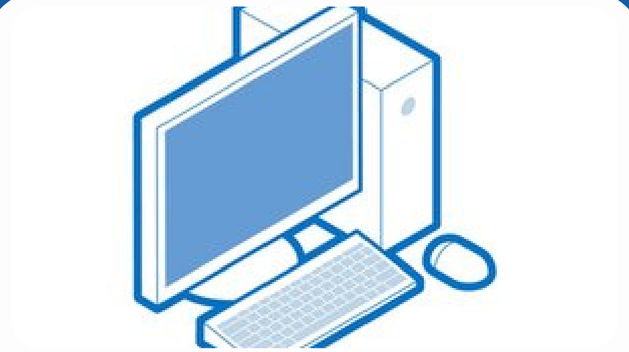
Better Mobile Browser Compatibility

WebGL works on mobile browsers like Safari and Chrome.



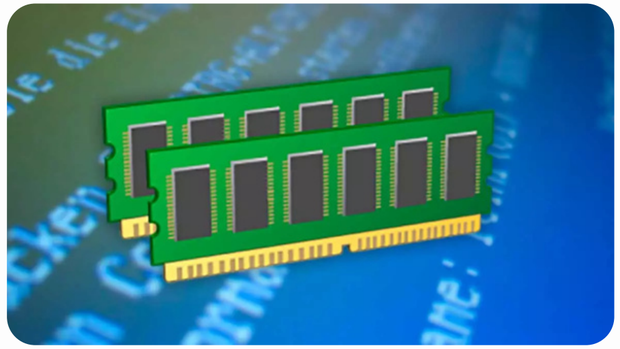
Open Source

WebGL is open-source, allowing access to its source code.



No Need for Compilation

WebGL apps run directly in the browser without compilation.



Automatic Memory Management

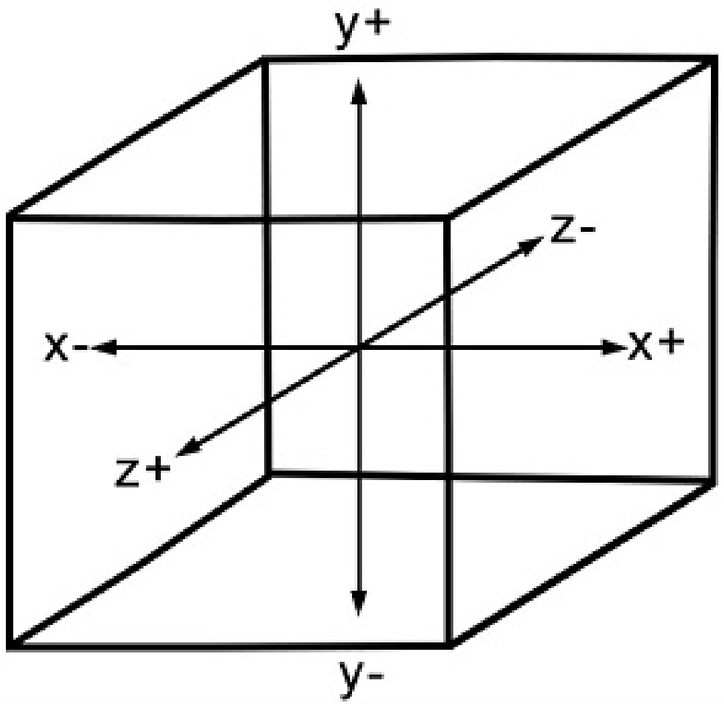
JavaScript handles memory management, inherited by WebGL.

Easy to Set Up

WebGL works with HTML5, needing only a text editor and browser.

2) WebGL Basics

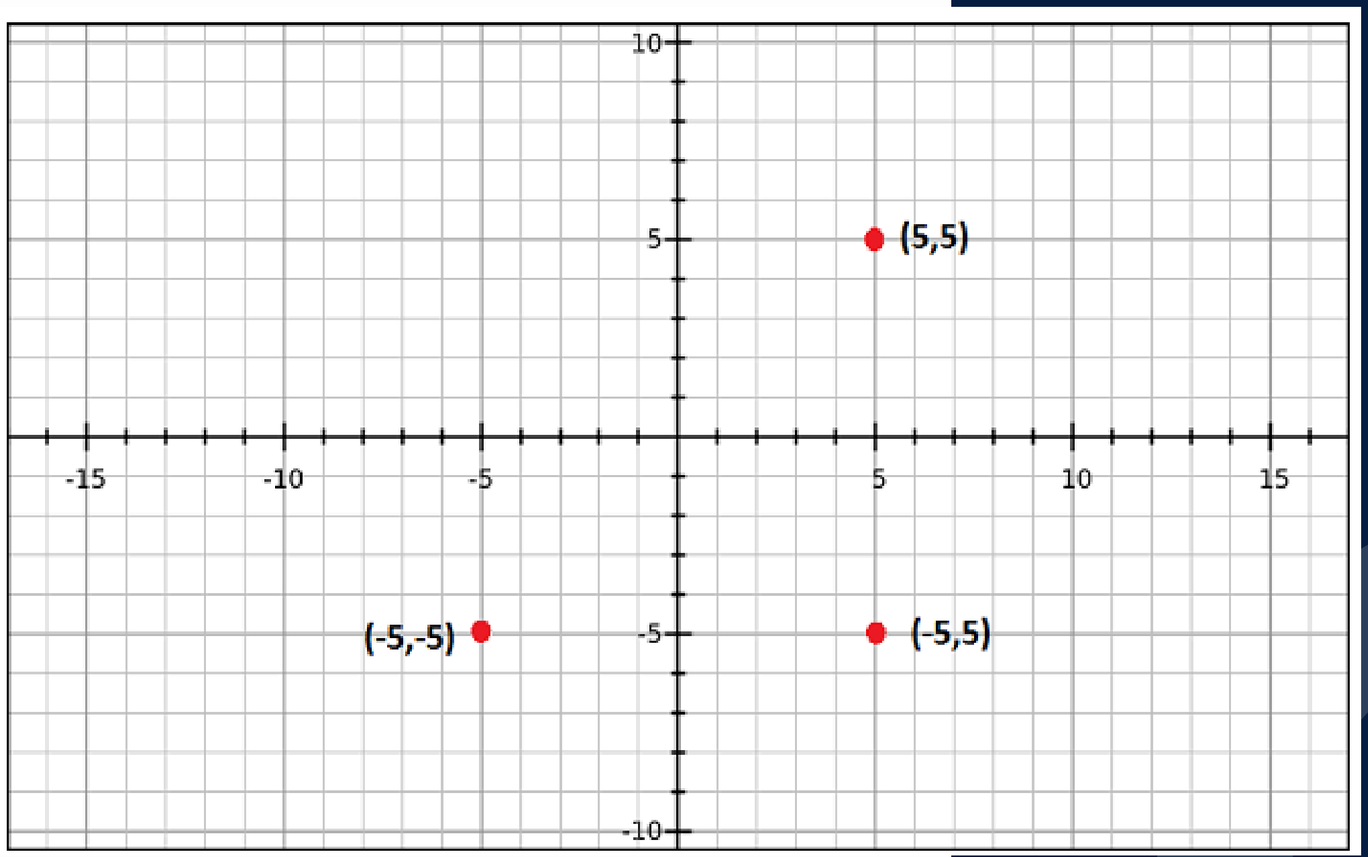
- Coordinates
  - x, y, z.
  - Restricted (1, 1, 1)-(-1, -1, -1).



2) WebGL Basics

- Graphics
  - Vertex or Index.
  - Save values manualy on a array.
  - Save into **buffer object.**

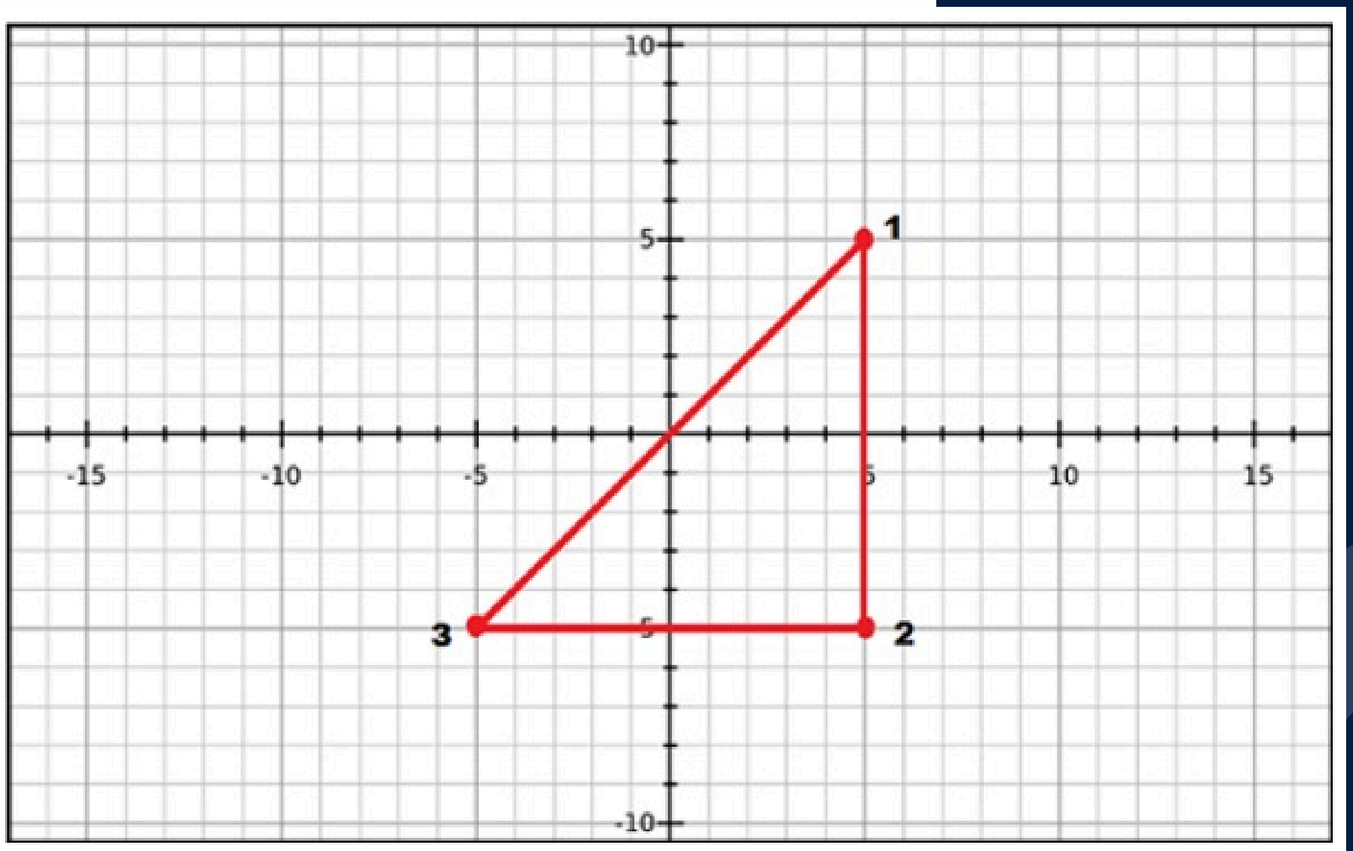
(0.5, 0.5), (-0.5, 0.5), (-0.5, -0.5)



2) WebGL Basics

- Graphics
  - Index for drawing meshes.

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



2) WebGL Basics

- Buffers
  - WebGL stores data in buffers drawing, frame, vertex, and index buffers.
  - Vertex and index buffers handle geometry data.
- Meshes
  - **drawArray()** and **drawElement().**
  - Mode → object to draw (primitives only).

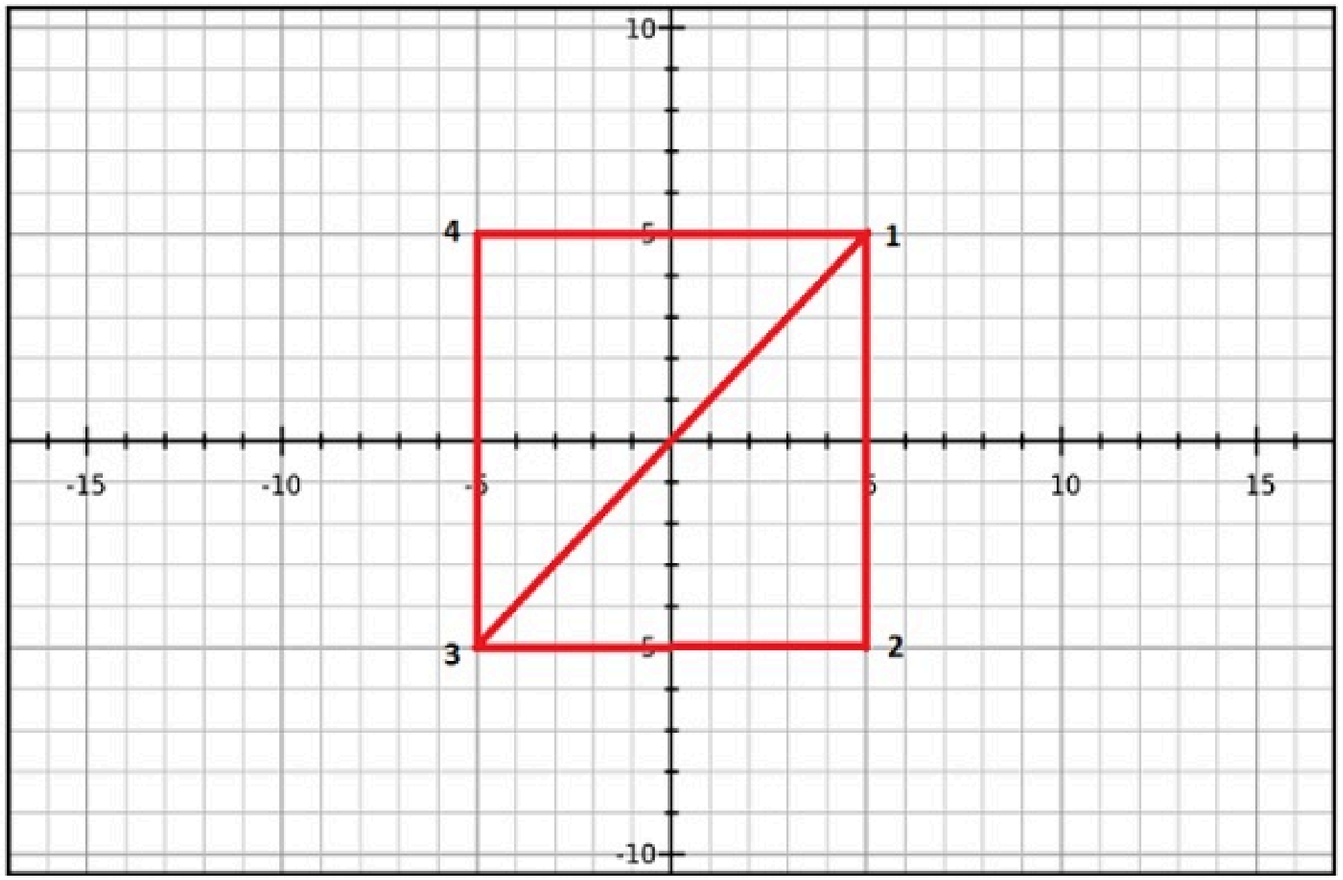
drawArray()

- Sequential vertex arrays.
- Does not reuse vertices.
- Best for simple geometries.

drawElement()

- index buffer.
- Does reuse vertices.
- Ideal for complex meshes.

2) WebGL Basics



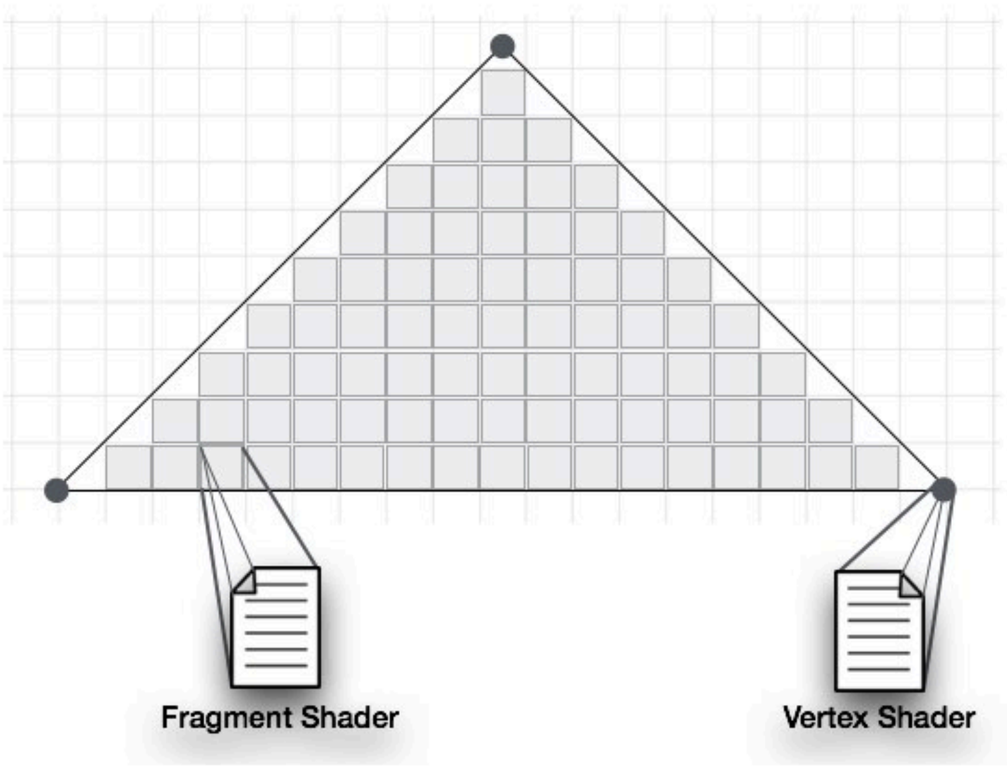
2) WebGL Basics

- Shaders
  - Uses triangles to build meshes.
  - Data must be transfered from CPU to GPU (GPU acceleration).
  - Shaders written in GLSL, which run directly on the GPU.
  - Define how vertices, transformations, materials, lights, and the camera interact to generate an image.
  - Two main shaders: **Vertex Shader** and **Fragment Shader.**

2) WebGL Basics

- Vertex Shader
  - Runs on each vertex to transform geometry.
  - Handle coordinates, normals, colors, and textures.
  - Performs vertex transformations.

- Fragment Shader
  - Runs on each fragment (triangle surface).
  - Calculates the color of each pixel.
  - Access and apply textures, effects...



1) WebGL 2D and 3D

- Prepare the canvas and get WebGL rendering context

```
/* Step1: Prepare the canvas and get WebGL context */

let canvas = document.getElementById('my_Canvas');
let gl = canvas.getContext('webgl');
```

1) WebGL 2D and 3D

- Define the geometry and store it in buffer objects

```
/* Step2: Define the geometry and store it in buffer objects */

const vertices = [-0.5, 0.5, -0.5, -0.5, 0.0, -0.5,];

// Create a new buffer object
const vertex_buffer = gl.createBuffer();

// Bind an empty array buffer to it
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);

// Pass the vertices data to the buffer
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

// Unbind the buffer
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

1) WebGL 2D and 3D

- Create and compile Shader programs

```
/* Step3: Create and compile Shader programs */

// Vertex shader source code
const vertCode =
  'attribute vec2 coordinates;' +
  'void main(void) {' + ' gl_Position = vec4(coordinates,0.0, 1.0);' + '}';

//Create a vertex shader object
const vertShader = gl.createShader(gl.VERTEX_SHADER);

//Attach vertex shader source code
gl.shaderSource(vertShader, vertCode);

//Compile the vertex shader
gl.compileShader(vertShader);
```

1) WebGL 2D and 3D

- Create and compile Shader programs

```
//Fragment shader source code
const fragCode = 'void main(void) {' + 'gl_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' + '}';

// Create fragment shader object
const fragShader = gl.createShader(gl.FRAGMENT_SHADER);

// Attach fragment shader source code
gl.shaderSource(fragShader, fragCode);

// Compile the fragment shader
gl.compileShader(fragShader);
```

1) WebGL 2D and 3D

- Create and compile Shader programs

```
// Create a shader program object to store combined shader program
const shaderProgram = gl.createProgram();

// Attach a vertex shader
gl.attachShader(shaderProgram, vertShader);

// Attach a fragment shader
gl.attachShader(shaderProgram, fragShader);

// Link both programs
gl.linkProgram(shaderProgram);

// Use the combined shader program object
gl.useProgram(shaderProgram);
```

1) WebGL 2D and 3D

- Associate the shader programs with buffer objects

```
/* Step 4: Associate the shader programs to buffer objects */

//Bind vertex buffer object
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);

//Get the attribute location
const coord = gl.getAttribLocation(shaderProgram, "coordinates");

//point an attribute to the currently bound VBO
gl.vertexAttribPointer(coord, 2, gl.FLOAT, false, 0, 0);

//Enable the attribute
gl.enableVertexAttribArray(coord);
```

1) WebGL 2D and 3D

- Drawing the required object

```
/* Step5: Drawing the required object (triangle) */

// Clear the canvas
gl.clearColor(0.5, 0.5, 0.5, 0.9);

// Enable the depth test
gl.enable(gl.DEPTH_TEST);

// Clear the color buffer bit
gl.clear(gl.COLOR_BUFFER_BIT);

// Set the view port
gl.viewport(0,0,canvas.width,canvas.height);

// Draw the triangle
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

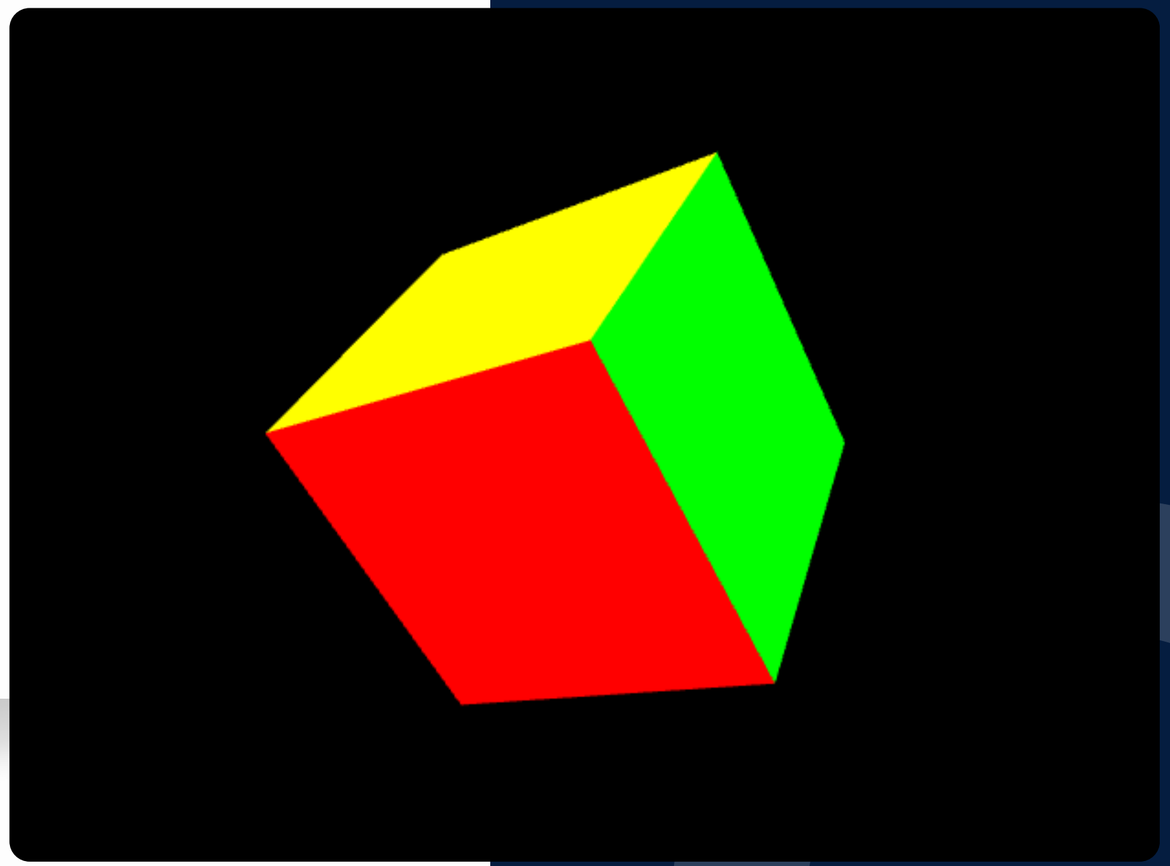
1) WebGL 2D and 3D

1) WebGL 2D and 3D

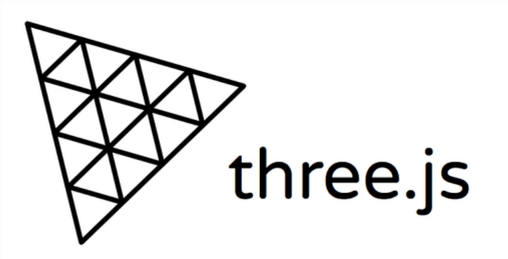
```
const positions = [
  // Front face
  -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0,

  // 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,
];
```

1) WebGL 2D and 3D

```
const colors = new Float32Array([
  1.0, 0.0, 0.0,  // Rojo
  0.0, 1.0, 0.0,  // Verde
  0.0, 0.0, 1.0   // Azul
]);
```



4) Three.js introduction



What is Three.js?

5) Instalation and setup

- Using npm and a builder ← prefered option
- CDN(Content Delivery Network)

1. Installing vite and three.js:
    a. npm install --save three
    b. npm install --save-dev vite

5) Instalation and setup

```
import { defineConfig } from 'vite';

export default defineConfig({
  // You can add extra options here
  server: {
    open: './public/index.html',
  },
});
```

```
.
├── node_modules
├── package.json
├── package-lock.json
├── public
├── src
├── tsconfig.json
├── vite.config.ts
└── web-server
```

Execute in root directory(important):

- npx vite

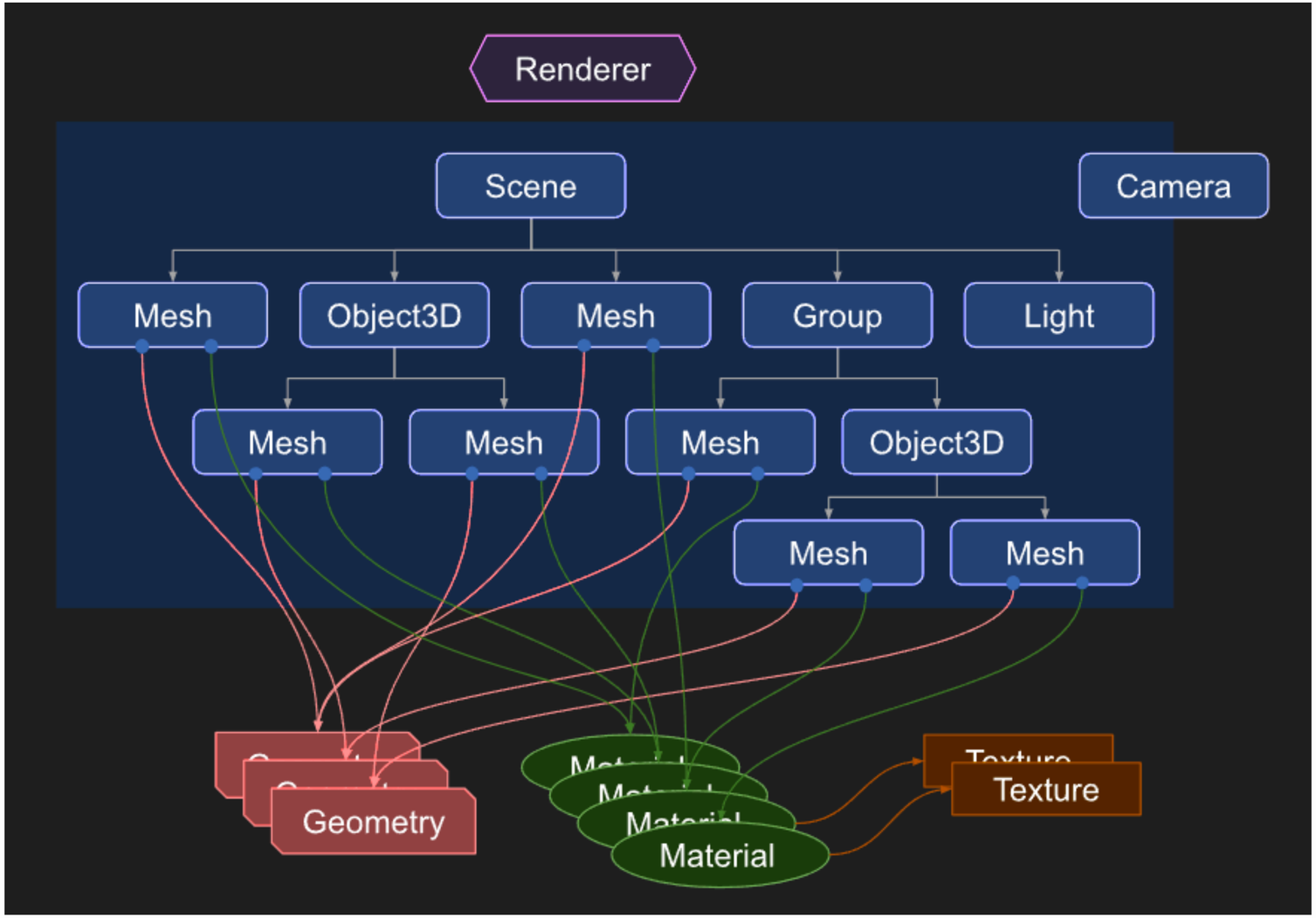
5) Instalation and setup

```
import { defineConfig } from 'vite';

export default defineConfig({
  build: {
    rollupOptions: {
      input: {
        main: '/public/index.html',          // Main page
        about: '/public/about.html',         // "About" page
        contact: '/public/contact.html',     // "Contact" page
        services: '/public/services.html',   // New "Services" page
        blog: '/public/blog.html',           // New "Blog" page
      },
    },
  },
});
```

Execute in root directory(important):

- npx vite build

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);

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

7) First scene

4)
- const geometry = new THREE.BoxGeometry(1, 1, 1);

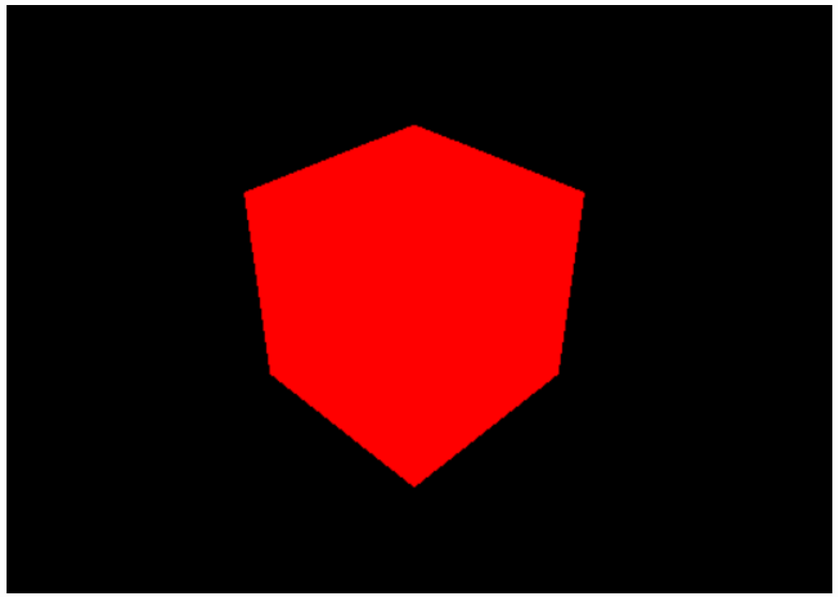
5)
- const material = new THREE.MeshBasicMaterial({ color: "red"});

6)
- const cube = new THREE.Mesh(geometry, material);
- scene.add(cube);

7) First scene

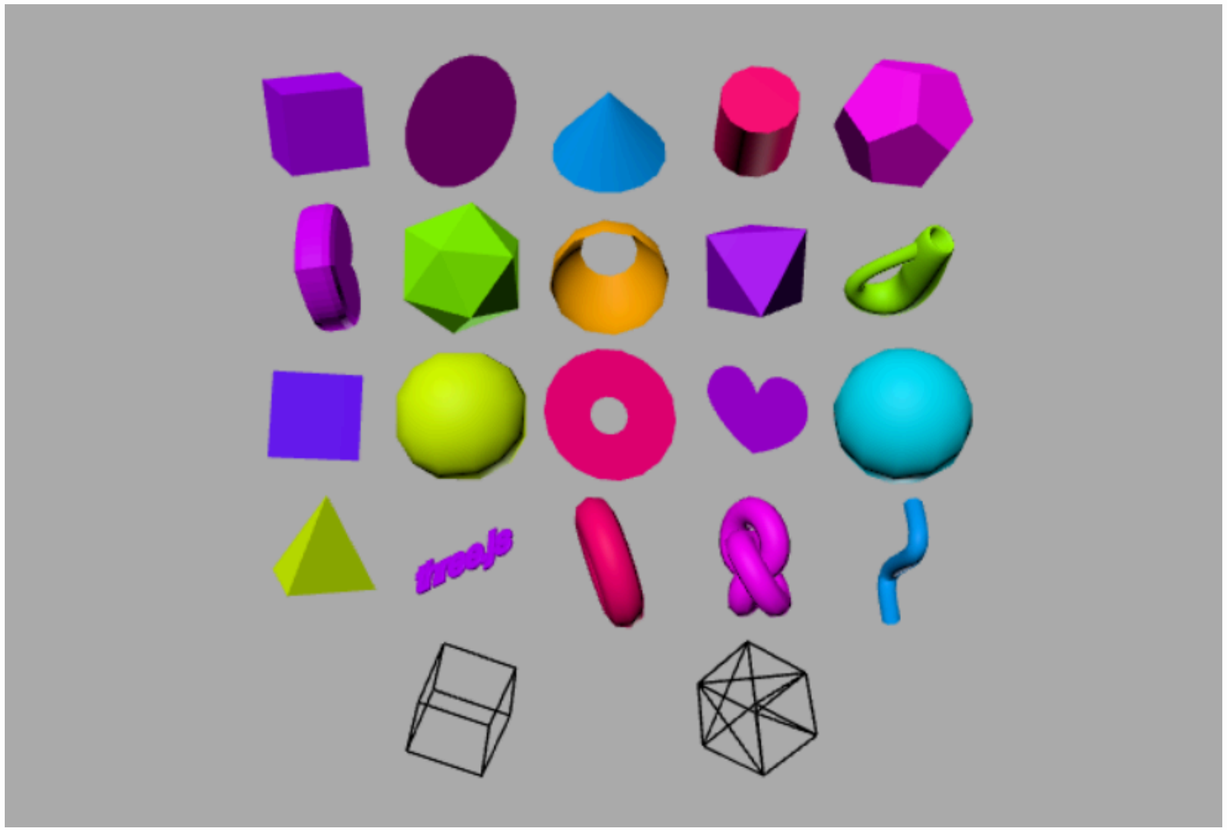
7)

- camera.position.set(5, 5, 5); // Posición de la cámara (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

Some more primitives:

-Extrusion.

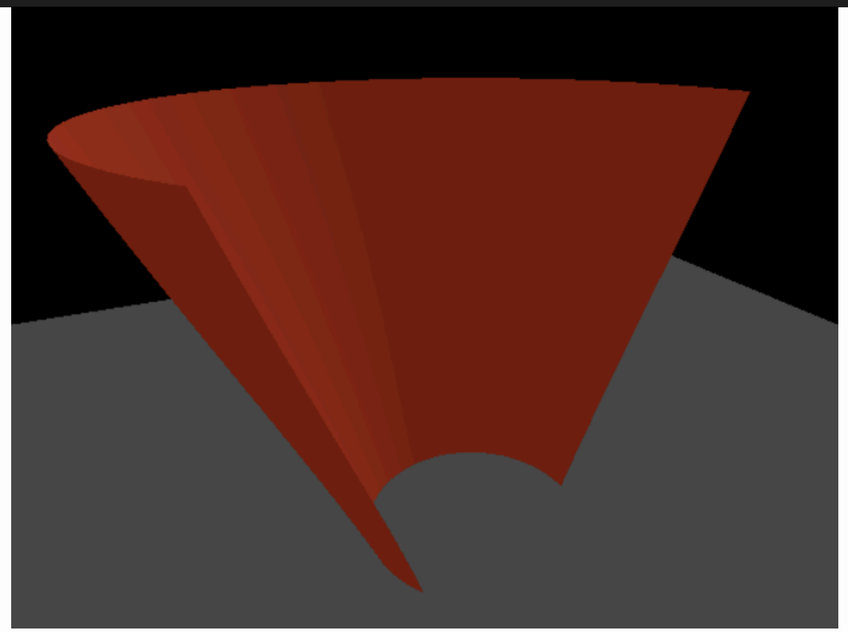
```
const shape = new THREE.Shape();
shape.moveTo(0, 0);
shape.lineTo(1, 0);
shape.lineTo(1, 1);
shape.lineTo(0, 1);
shape.lineTo(0, 0);

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

8) Primitive figures

Some more primitives:

-Lathe.

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

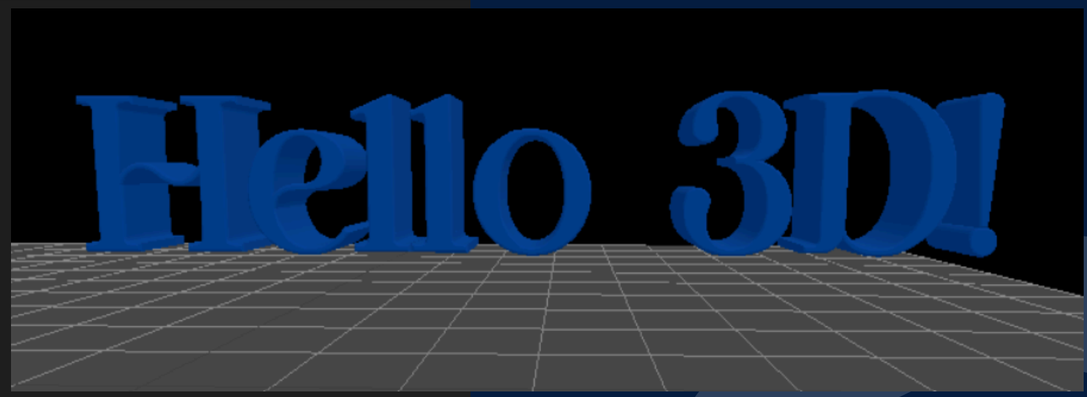


8) Primitive figures

Some more primitives:

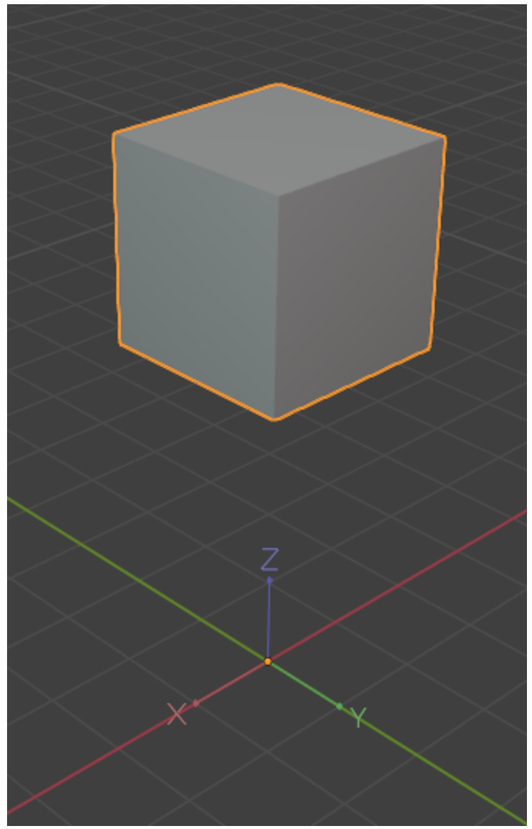
-Text

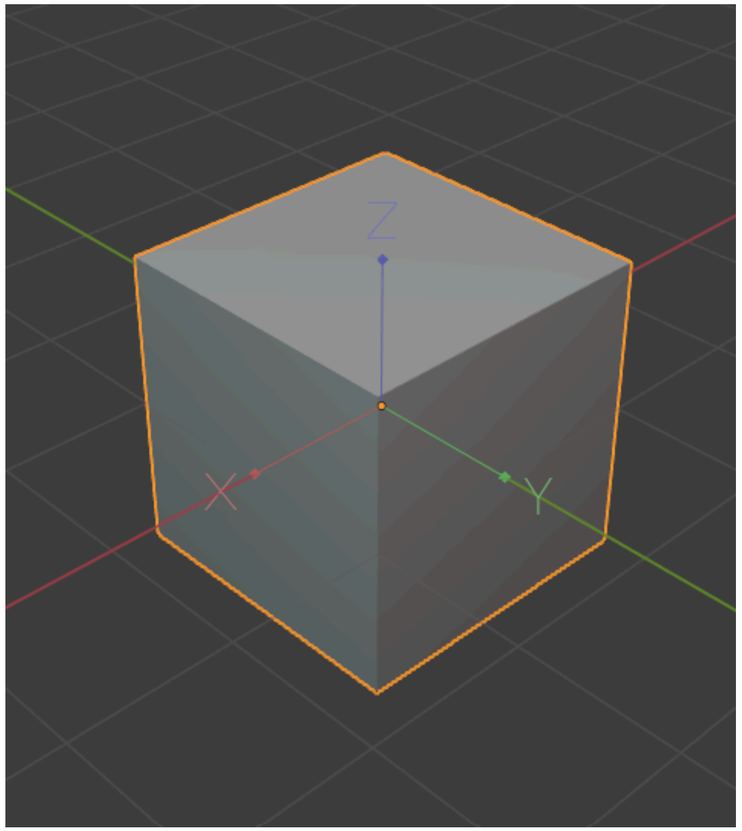
```
const loader = new FontLoader();
loader.load('http://localhost:3000/src/figures/Hefana_Regular.json', (font) => {
  const textGeometry = new TextGeometry('Hello 3D!', {
    font: font,
    size: 1,
    depth: 0.5,
    curveSegments: 12,
    bevelEnabled: true,
    bevelThickness: 0.03,
    bevelSize: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

  const textMesh = new THREE.Mesh(textGeometry, material);
  textMesh.position.set(0, 1, 0);
  scene.add(textMesh);
});
```



8) Primitive figures

Important property:
-The center of geometry





9) Materials

-What are materials?

9) Materials

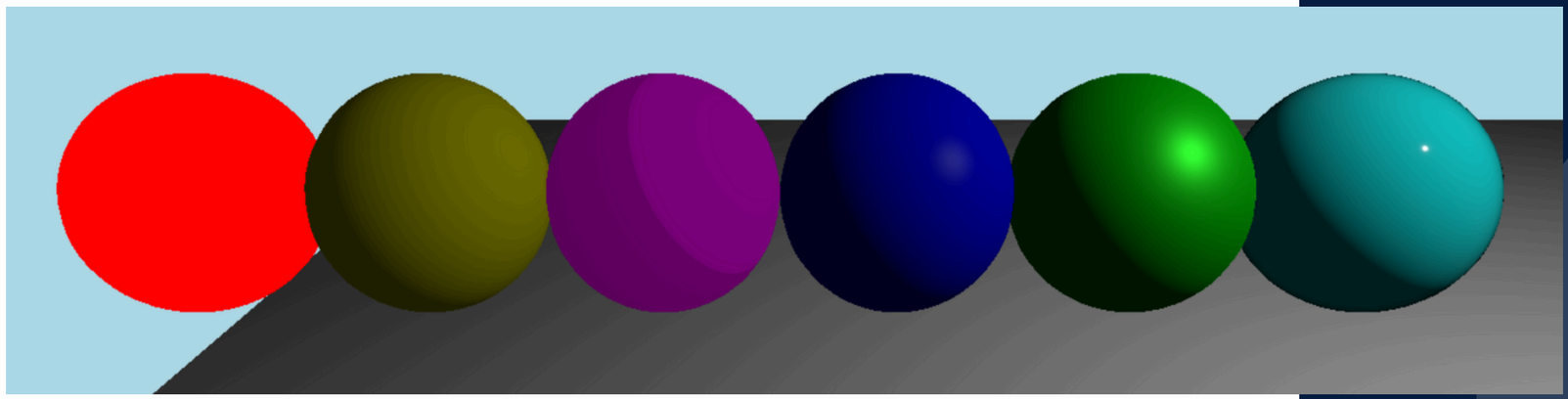
-Material declaration:

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

material.side = THREE.DoubleSide;
material.color = new THREE.Color('lightblue');
```

9) Materials

-Principal materials

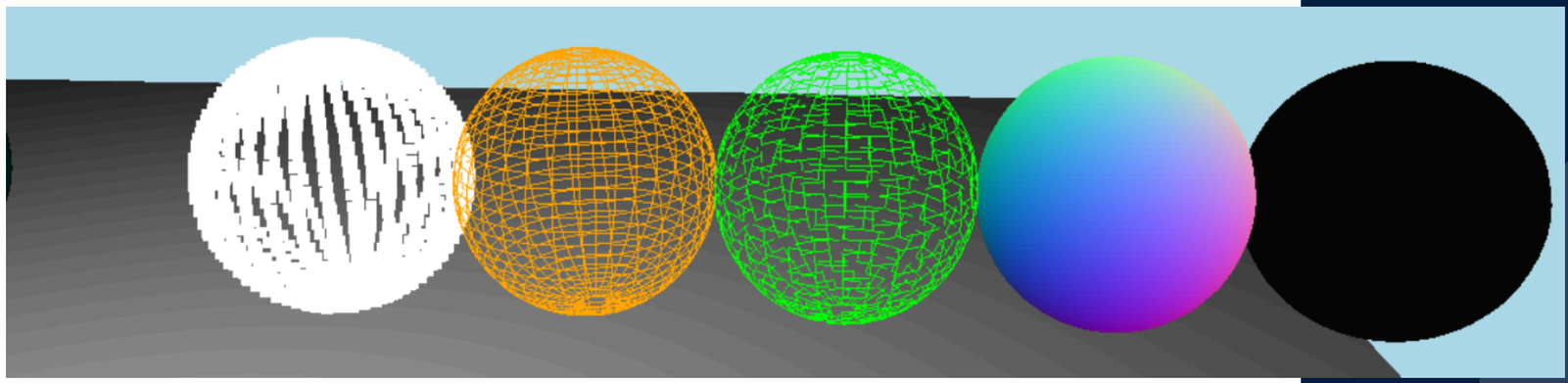
```
new THREE.MeshBasicMaterial({
new THREE.MeshLambertMaterial(
new THREE.MeshToonMaterial({ c
new THREE.MeshPhongMaterial({
new THREE.MeshStandardMaterial
new THREE.MeshPhysicalMaterial
```



9) Materials

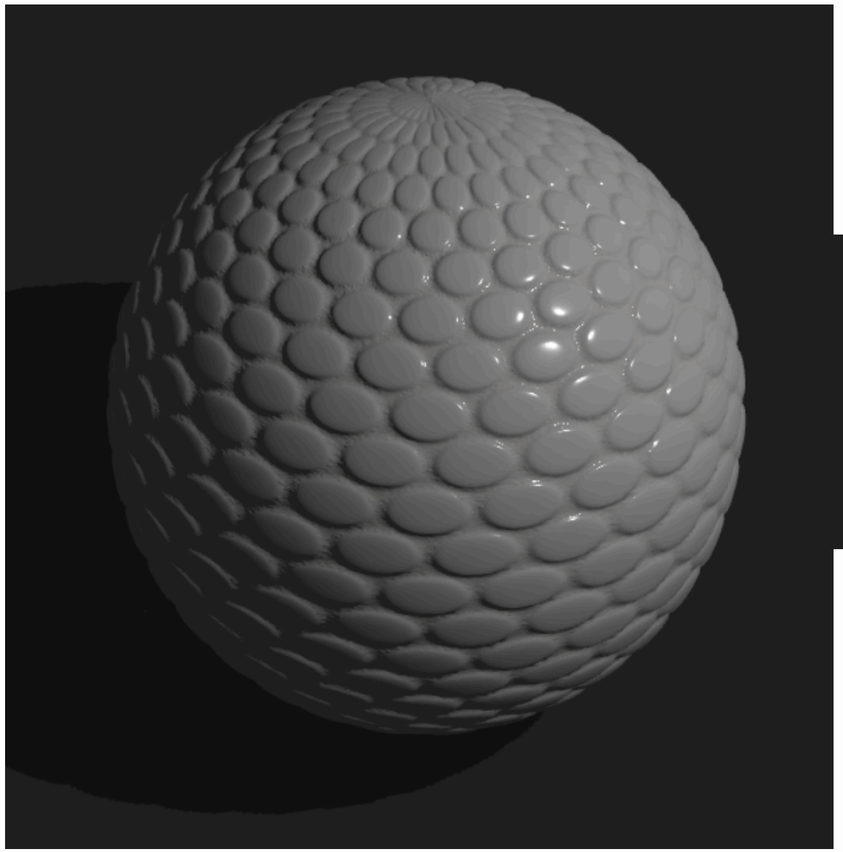
-Other materials

```
new THREE.ShadowMaterial({ opac
new THREE.PointsMaterial({ colo
new THREE.LineBasicMaterial({ c
new THREE.LineDashedMaterial({
new THREE.MeshNormalMaterial(),
new THREE.MeshDepthMaterial(),
```



9) Textures

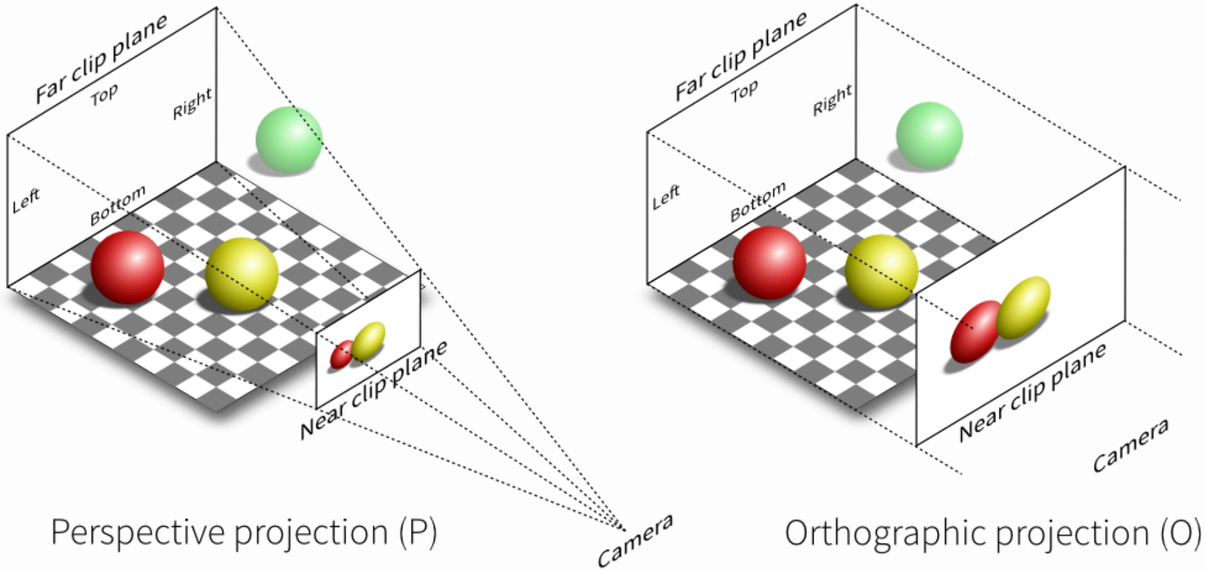
```
const textureLoader = new THREE.TextureLoader();
const colorTexture: THREE.Texture = textureLoader.load('ruta.png');
```



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

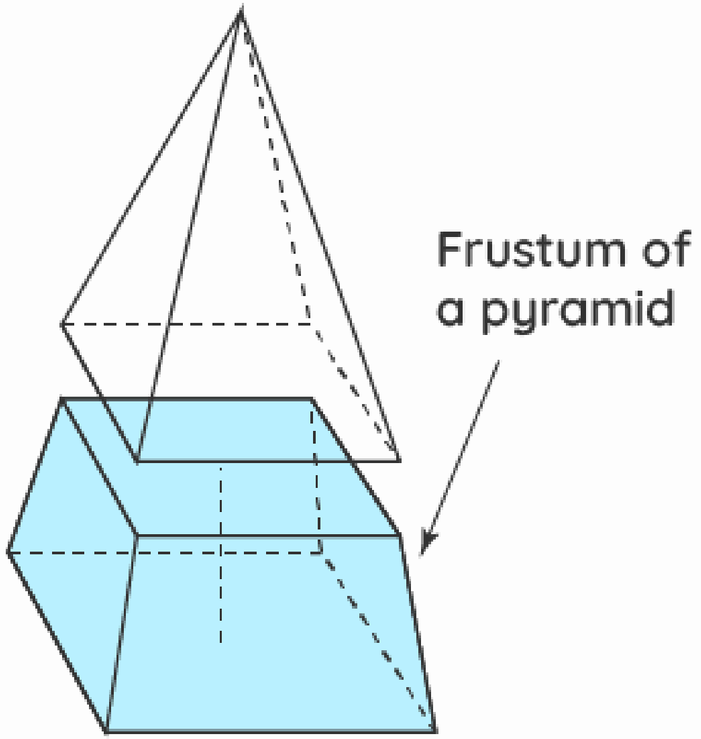
10) Cameras

They act as a view of the scene, allowing you to see it from one or more perspectives
The following cameras are the most used:



10) Prespective Camera

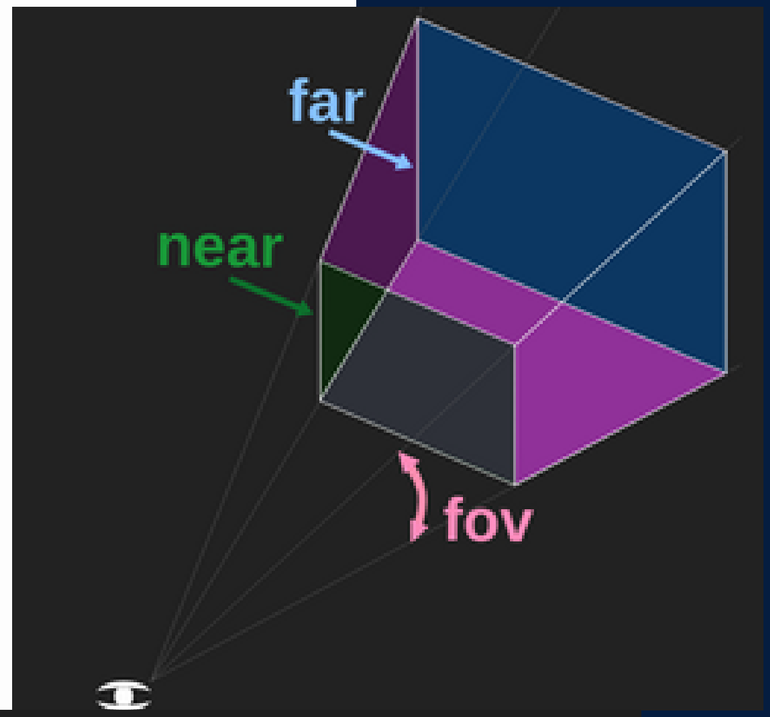
The operation of this camera is similar to the behavior of the human eye and is defined with a frustrum



10) Prespective Camera

4 properties:

- Fov(field of view) in degrees
- Near: minimun of distance objects are visible
- Far: maximun of distance objects are visible
- aspect (width / height), for example: aspect = 2 => whidth = height * 2




```
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 aplication of prespective camera

10) Ortographic Camera

Defined with a box, doesn't . Objects will appear the same size regardless of their distance. This is often used for 2D design.

6 properties:

- Near and Far
- 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 aplication of ortographic camera

10) Orbit controls

Allow you to manipulate the camera teo main types of movements:

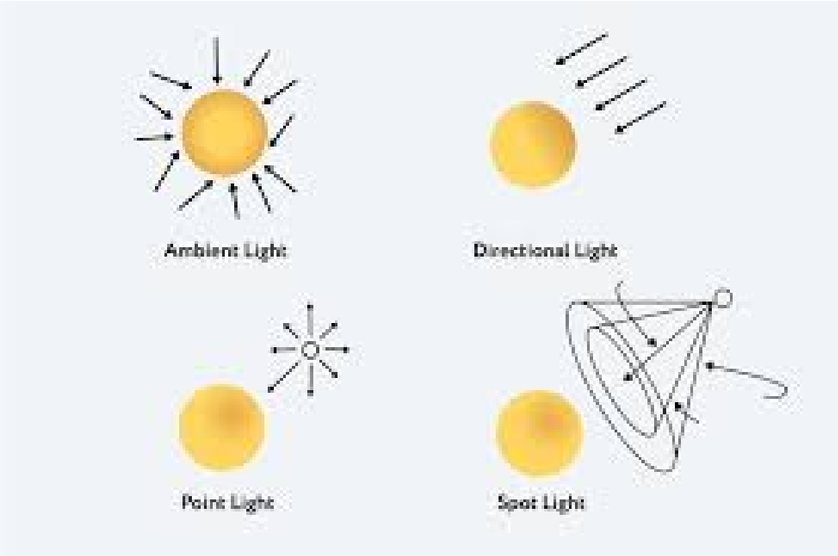
- Rotate: rotate the camera around a focus point.
- Zoom: Zoom in and out

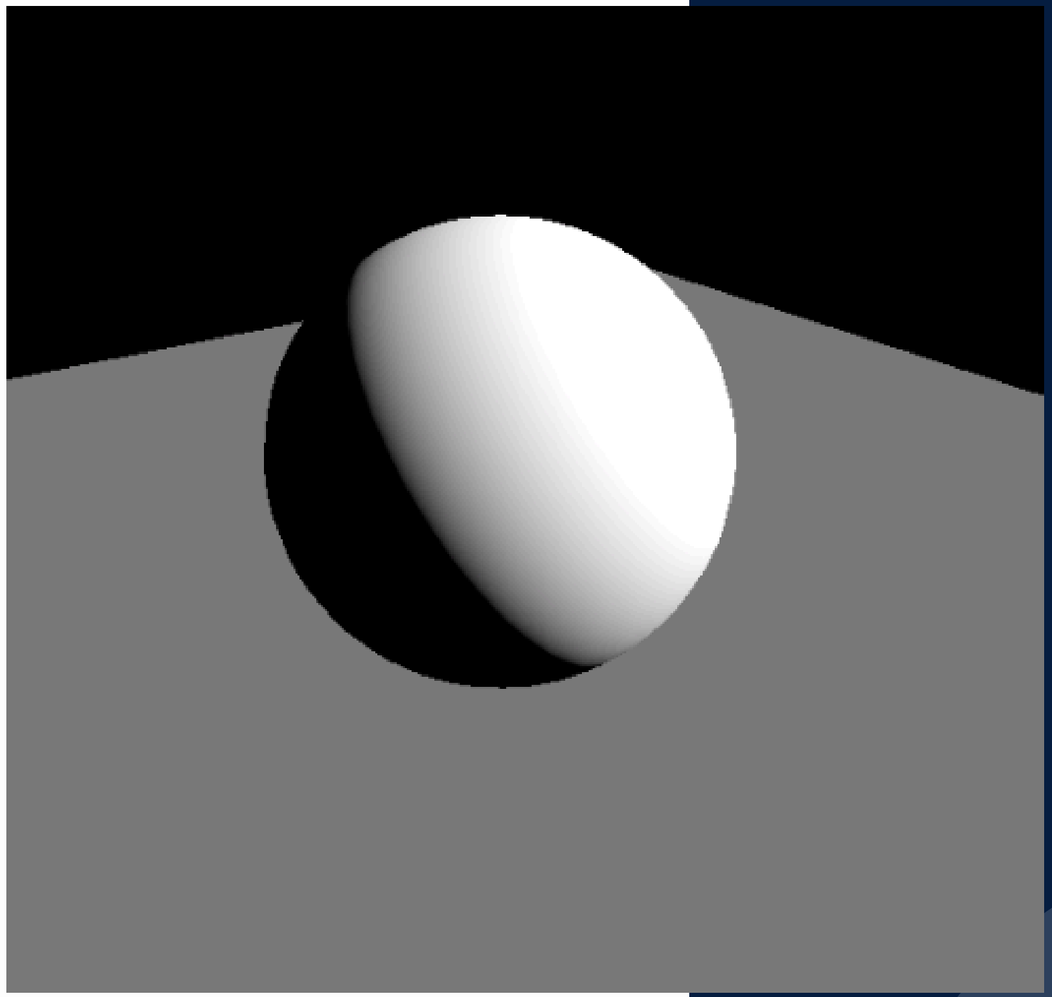
```
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js';
```

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

11) Lights

- Ambient linght
- Directional light
- Point light





11) Ambient Light

illuminates the entire scene equally

Have 2 properties

- Light color
- Intensity

```
const ambientLight = new THREE.AmbientLight('blue', 5);
```

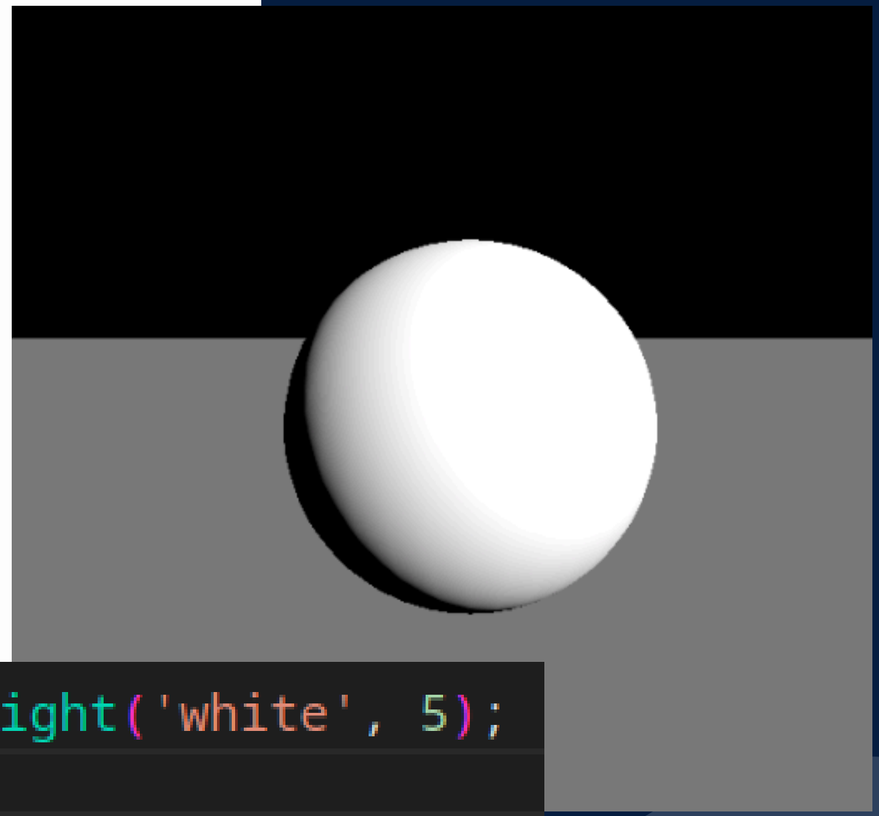
Interactive aplication of ambient light

11) Directional Light

illuminates the entire scene in one direction
Have 2 properties

- Light color
- Intensity

Positon by default (0,1,0) looking at (0,0,0)



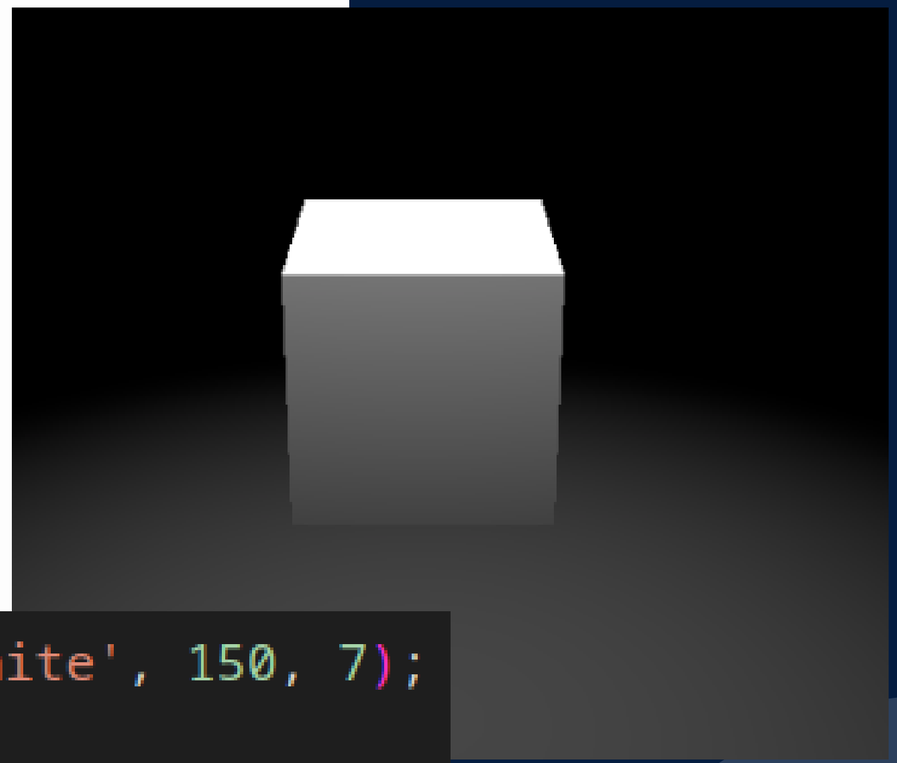
```
const directionalLight = new THREE.DirectionalLight('white', 5);
directionalLight.position.set(5, 5, 5);
```

Interactive aplication of directional light

11) Point Light

illuminates the entire scene in one range
Have 3 properties:

- Light color
- Iluminated area
- Intensity

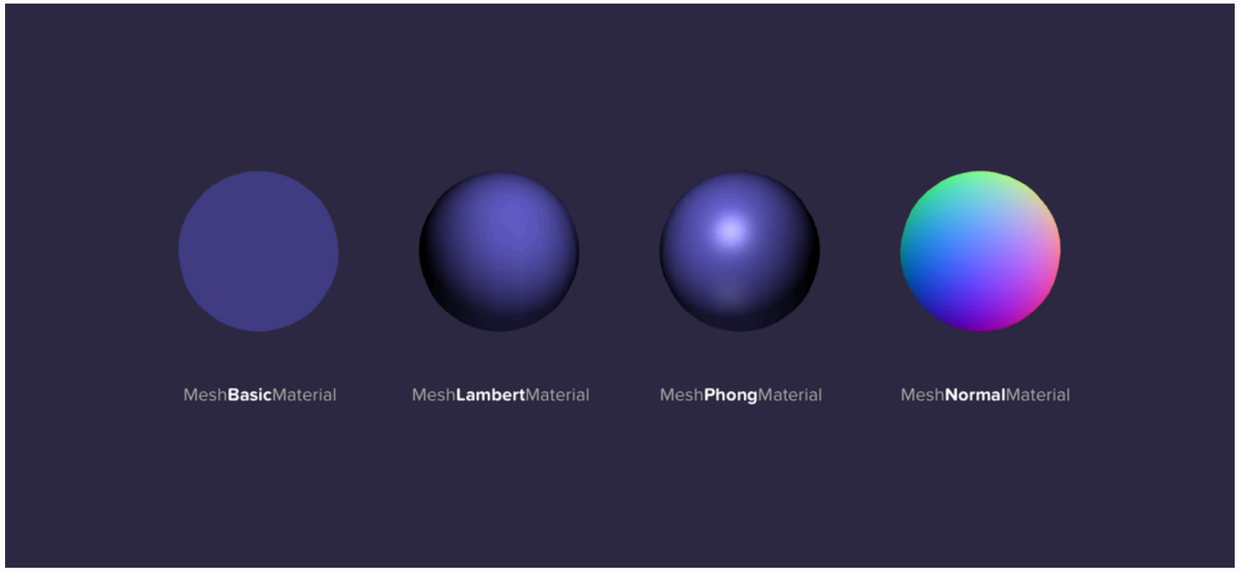


```
const pointLight = new THREE.PointLight('white', 150, 7);
pointLight.position.set(0, 5, 1);
```

Interactive aplication of point light

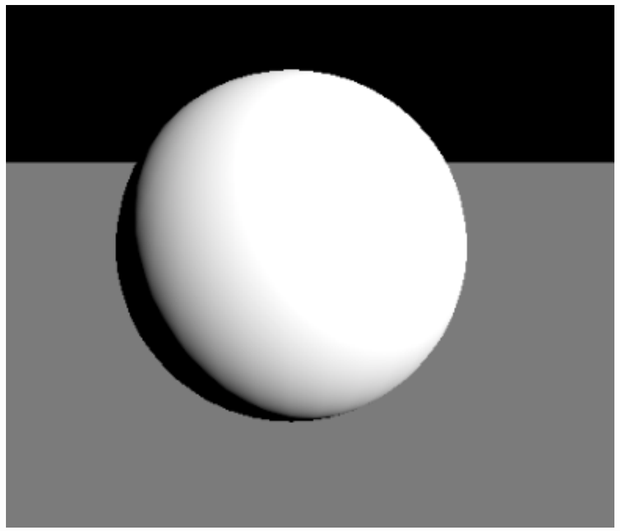
11) Lights Why not MeshBasic material

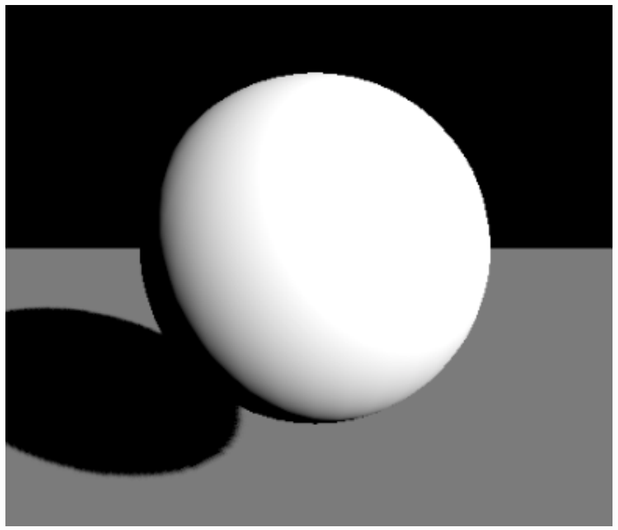
Not all materials react to lights, Basic Matereal does not, is always al same colour, some materials for lights are PhongMatereal, LambertMaterial, etc.



12) Shadows

Difference between a lighting effect and a shadow in three js





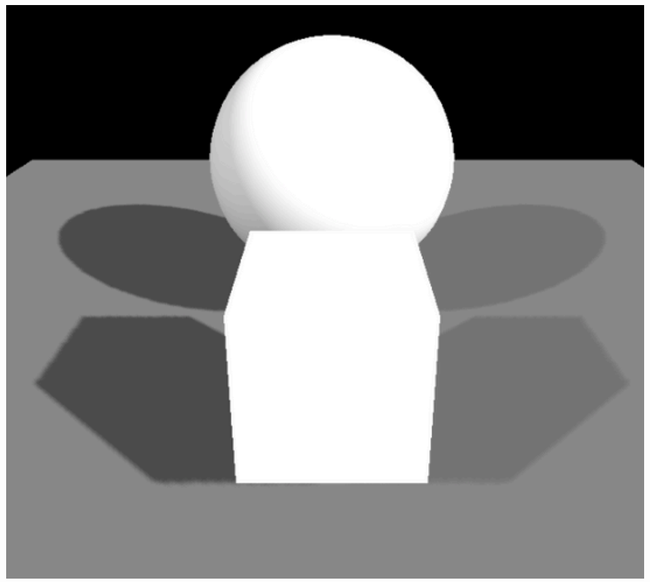
12) Shadows

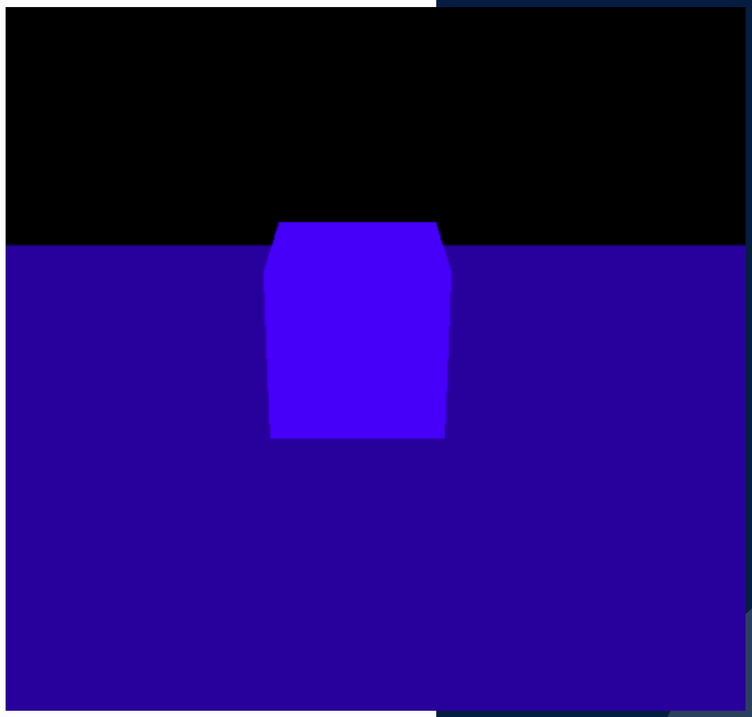
Enable shadows at renderer, at lights and at objects

```
// Sombras para el renderizador
renderer.shadowMap.enabled = true;

// Sombras para la luz direccional
directionalLight.castShadow = true;

// Sombras para la esfera
sphere.getSphere().castShadow = true;

// Sombras para el plano
plane.getPlane().receiveShadow = true;
```

12) Shadows

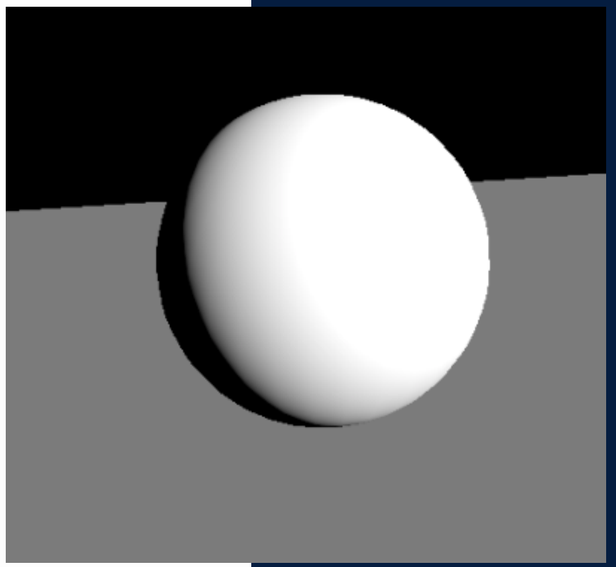
Not all lights generates Shadows, for example Ambiental Light, directional and point lights generates shadows





12) Shadows

Light shadow camera, is the shadow range generated by the light.



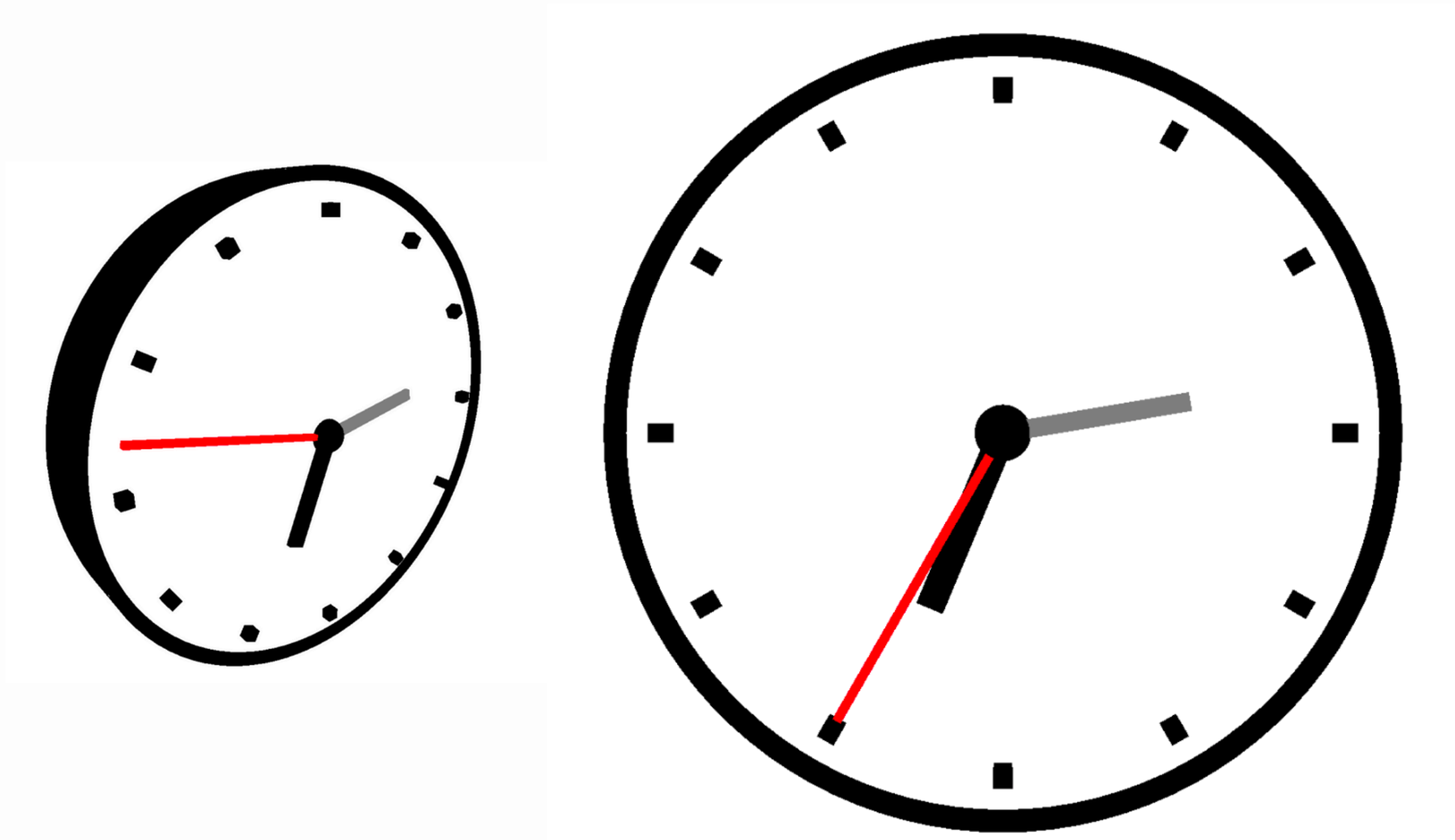
```
directionalLight.shadow.camera.near = 0.5;
directionalLight.shadow.camera.far = 10;
directionalLight.shadow.camera.left = -5;
directionalLight.shadow.camera.right = 5;
directionalLight.shadow.camera.top = 5;
directionalLight.shadow.camera.bottom = -5;
```

Interactive aplication of shadows camera

13) Loading Obj and Fbx

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

// Fbx loader, mercedes con texturas
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); // Ajusta la posición del modelo
  scene.add(fbx);
});

// OBJ loader, mercedes sin texturas
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
- 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=PLDIlzmccetSPVF3JN6OFazp39N00yHE3A&index=3

Thank you