

Gráficos y visualización 3D

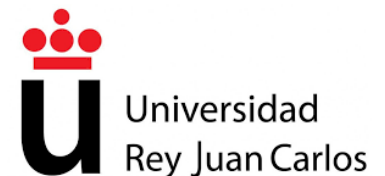
8. Creación de gráficos 3D con Three.js

Boni García

Web: <http://bonigarcia.github.io/>

Email: boni.garcia@urjc.es

Dept. Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación (GSyC)
Escuela Superior De Ingeniería De Telecomunicación (ETSIT)
Universidad Rey Juan Carlos (URJC)



Índice de contenidos

1. Introducción
2. Conceptos básicos
3. Geometrías
4. Texturas
5. Cámaras
6. Iluminación
7. Animación

Índice de contenidos

1. Introducción
 - I. Librerías gráficas
 - II. Three.js
2. Conceptos básicos
3. Geometrías
4. Texturas
5. Cámaras
6. Iluminación
7. Animación

1. Introducción – Librerías gráficas

- Hasta el momento hemos aprendido a generar gráficos (interactivos) usando directamente la API WebGL (JavaScript)
- Este procedimiento se puede considerar de bajo nivel, ya que tenemos que trabajar directamente con los shaders en GLSL
- En la actualidad existen multitud de librerías JavaScript que proporcionan mecanismos de alto nivel para la creación de gráficos 3D

1. Introducción – Librerías gráficas

- Algunas de estas librerías son:

- Three.js
- Babylon.js
- Cannon.js
- CopperLicht
- Phoria.js
- Scene.js
- D3.js
- LightGL.js
- Seen.js

three.js



Cannon.js

CopperLicht



sceneJS
3D Engine for the Web



seen.js

1. Introducción – Three.js

- Three.js es una librería de código abierto (licencia MIT) que permite crear gráficos WebGL de manera sencilla
- Fue creada inicialmente en 2010 por Ricardo Cabello (Mr.Doob)

<https://threejs.org/>

<https://github.com/mrdoob/three.js/>

Índice de contenidos

1. Introducción
- 2. Conceptos básicos**
 - I. Carga de librería**
 - II. Terminología**
 - III. Coordenadas**
 - IV. Hola mundo**
3. Geometrías
4. Texturas
5. Cámaras
6. Iluminación
7. Animación

2. Conceptos básicos – Carga de la librería

- El primer paso para poder usar Three.js consiste en incorporar la librería en nuestra página web. Podemos enlazar la librería:
 - De forma local, descargada desde el repositorio GitHub o mediante NPM (Node.js Package Manager):

```
<script src="js/three.min.js"></script>
```

- De forma remota, usando una CDN (*content delivery network*):

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/three.min.js"></script>
```


2. Conceptos básicos – Terminología

- El elemento de alto nivel que engloba nuestro gráfico 3D realizado con Three.js se denomina **escena**:

```
var scene = new THREE.Scene();
```

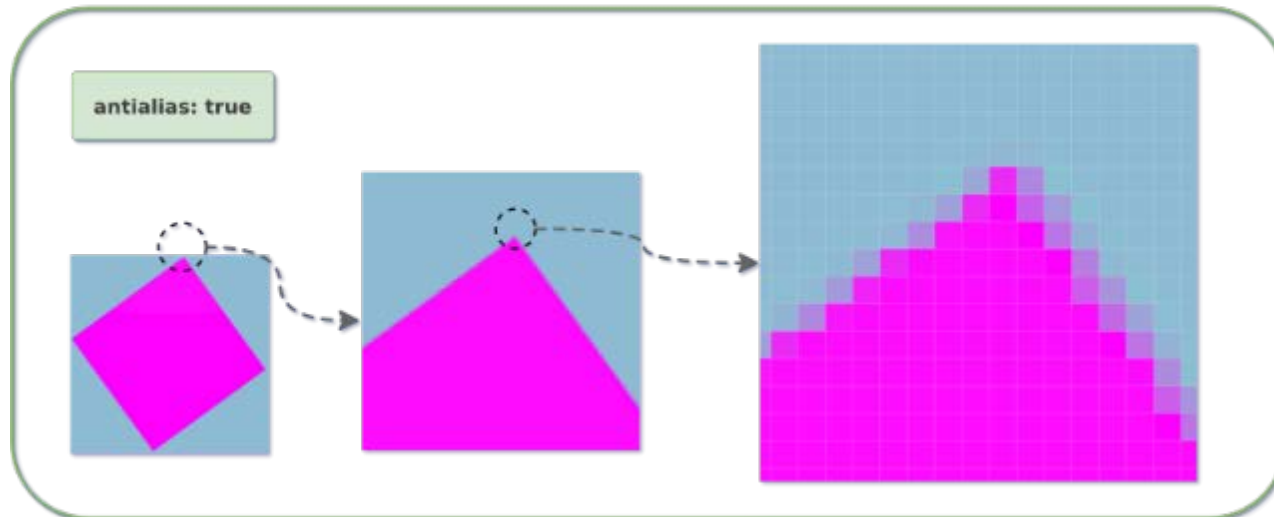
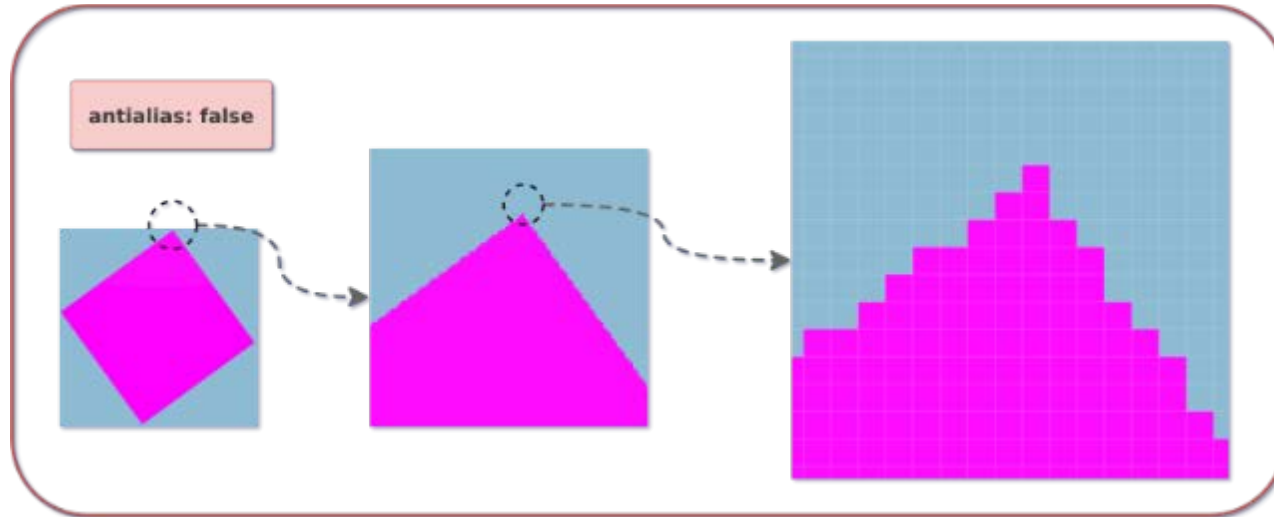
Toda la API Three.js es accesible a través del objeto THREE

- El elemento que permite generar gráficos dentro de la escena se conoce como **renderizador** (*renderer*):

```
var renderer = new THREE.WebGLRenderer({  
  antialias : true  
});
```

Una opción común al crear el renderizador es activar el suavizado de bordes (*antialiasing*)

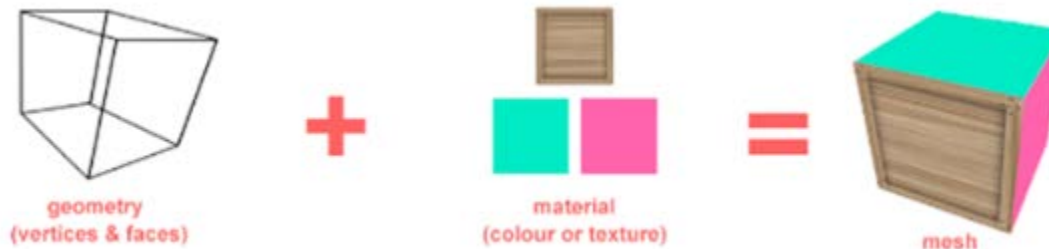
2. Conceptos básicos – Terminología



2. Conceptos básicos – Terminología

- Una escena estará formada por un conjunto de **mallas poligonales** (mesh)
- Una malla es un elemento formado a su vez por un conjunto de vértices llamado **geometría** más un **material** (color o textura)

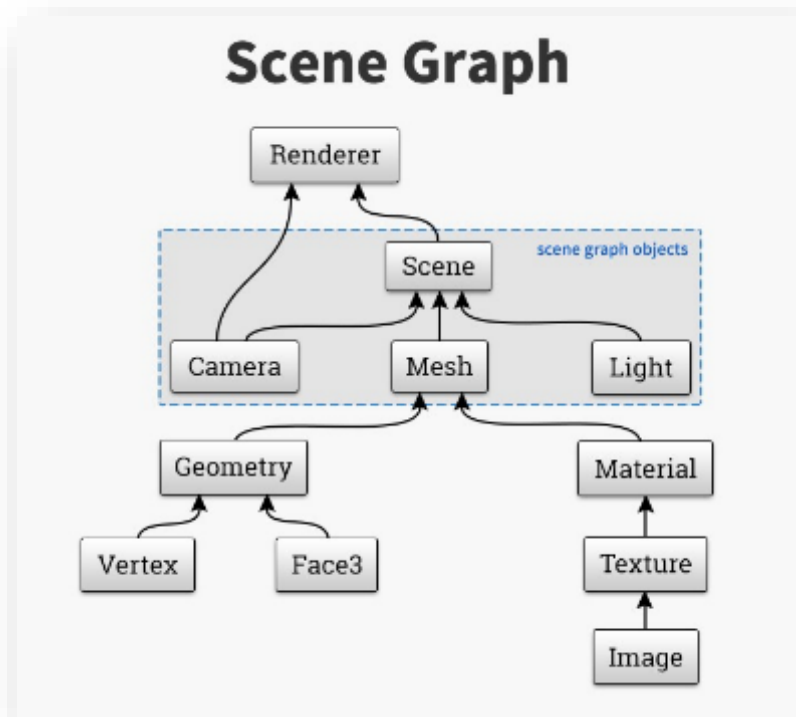
```
var mesh = new THREE.Mesh(geometry, material);  
scene.add(mesh);
```



2. Conceptos básicos – Terminología

- Las objetos de escena en Three.js (mallas, se implementan mediante objetos de tipo Object3D
- Algunas propiedades/métodos importantes son:

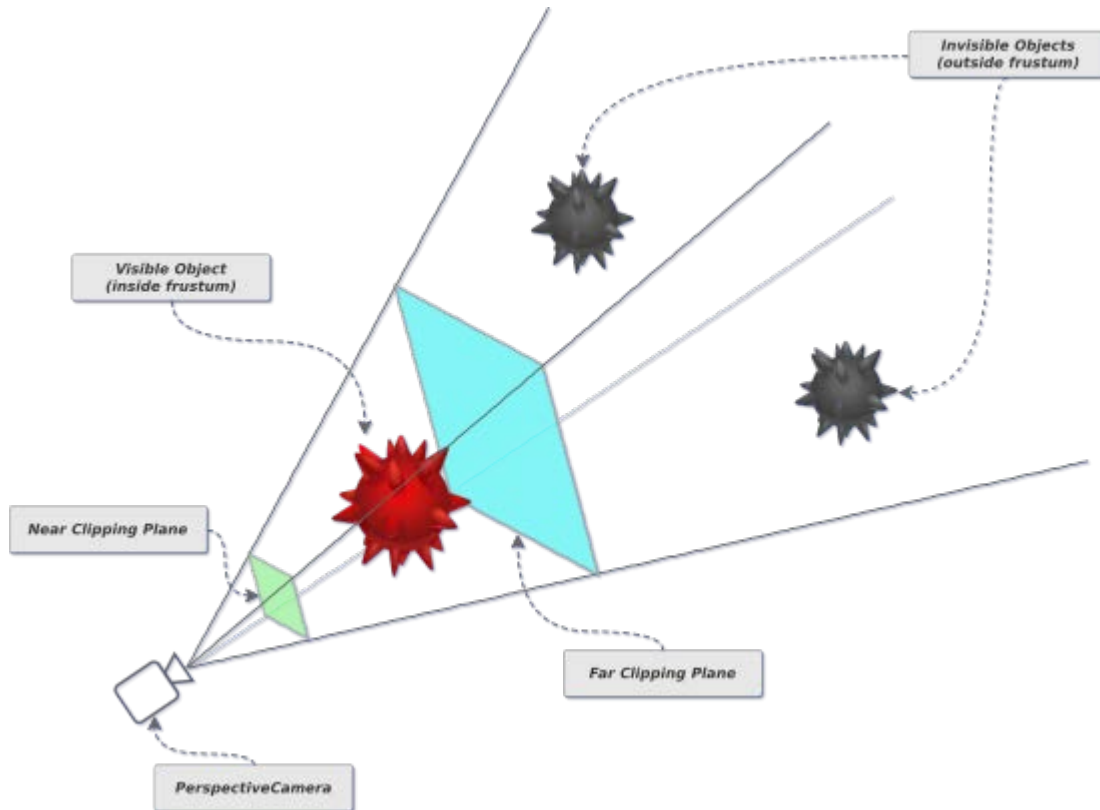
Método	Descripción
position	Posición (x, y, z)
rotation	Giro (x, y, z)
scale	Escala (x, y, z)
Método	Descripción
add	Añadir
clone	Clonar
lookAt	Mirar (rotar para encarar)
remove	Eliminar



2. Conceptos básicos – Terminología

- Además, será necesario especificar el tipo de proyección (denominada **cámara**):

```
var camera = new THREE.PerspectiveCamera(fov, ratio, near, far);
```



Los parámetros del campo de visión en perspectiva (frustum, tronco piramidal) son:

- fov: Ángulo en grados de la cámara
- ratio: Relación de aspecto (ancho / alto)
- near: Plano de recorte cercano
- far : Plano de recorte lejano

2. Conceptos básicos – Terminología

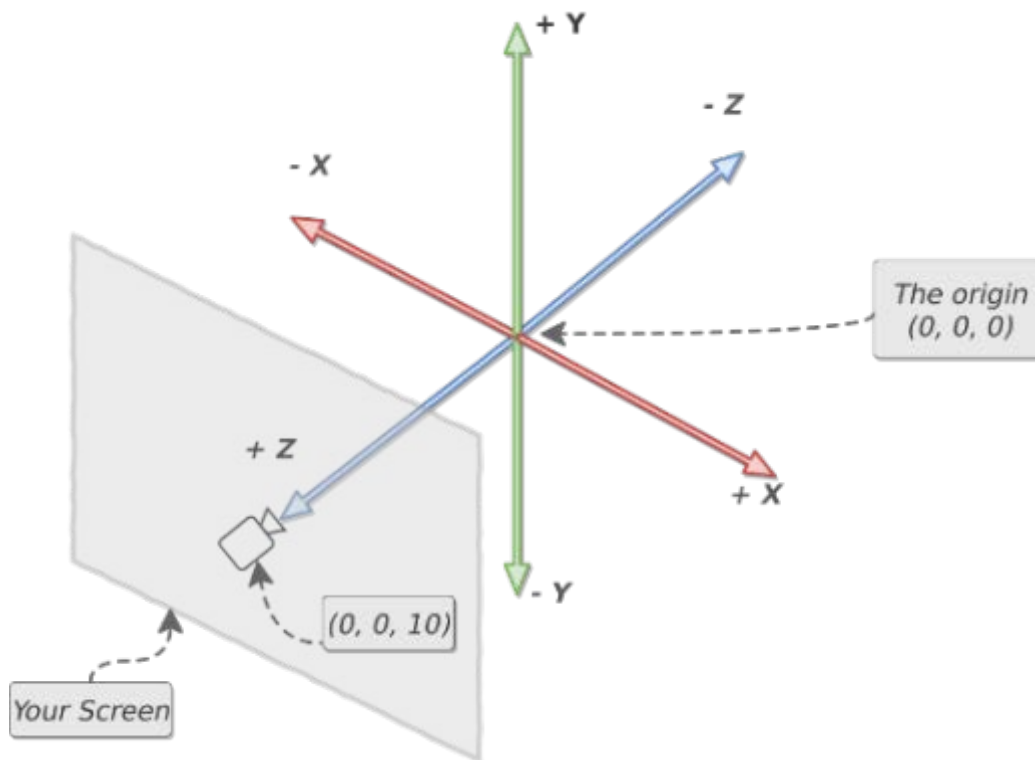
- Finalmente, el método render del objeto renderizador nos permite realizar la operación de dibujado de la escena:

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

<https://threejs.org/docs/#api/en/renderers/WebGLRenderer>

2. Conceptos básicos – Coordenadas

- En Three.js se usa un sistema de coordenadas equivalente al de WebGL



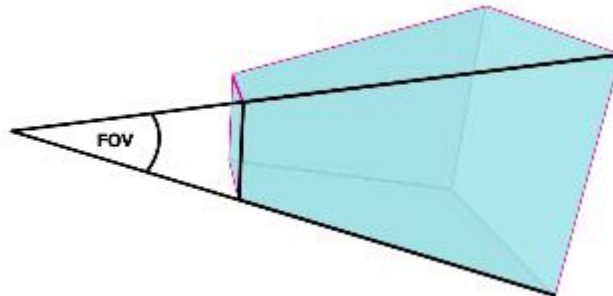
Cada objeto en Three.js tiene sus propias coordenadas locales, y en la escena se habla de coordenadas globales (*world coordinates*)

2. Conceptos básicos – Coordenadas

- Al crear una geometría tendremos que especificar sus coordenadas globales x , y , z
- El tamaño con el que visualizamos las geometrías dentro de la escena va a depender de:
 - La posición de la cámara
 - La proyección de la cámara (perspectiva, ortogonal)

2. Conceptos básicos – Coordenadas

- El ángulo de la proyección en perspectiva (**FOV**, *Field of View*) define el ángulo del área de visualización
- Su valor va a ser determinante a la hora del tamaño real en el que se perciben las geometrías renderizadas



2. Conceptos básicos – Coordenadas

- El FOV de los humanos es de unos 120°
- Los juegos de videoconsola se suelen diseñan con un FOV de $40-60^\circ$
- Los juegos de PC se suelen diseñar con un FOV de **90°** ya que la pantalla está más cercana al usuario
 - En los siguientes ejemplos vamos a usar este valor para el FOV de la cámara en perspectiva
- Por convención, 1 unidad de las coordenadas en Three.js se entiende como **1 metro**

2. Conceptos básicos – Hola mundo

```
<!DOCTYPE html>
<html>

<head>
<title>Three.js: Hello World</title>
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/three.min.js"></script>
<script>
  function init() {
    var scene = new THREE.Scene();
    var sceneWidth = window.innerWidth;
    var sceneHeight = window.innerHeight;

    var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
    camera.position.z = 5;

    var geometry = new THREE.BoxGeometry(2, 2, 2);
    var material = new THREE.MeshNormalMaterial();
    var mesh = new THREE.Mesh(geometry, material);
    scene.add(mesh);

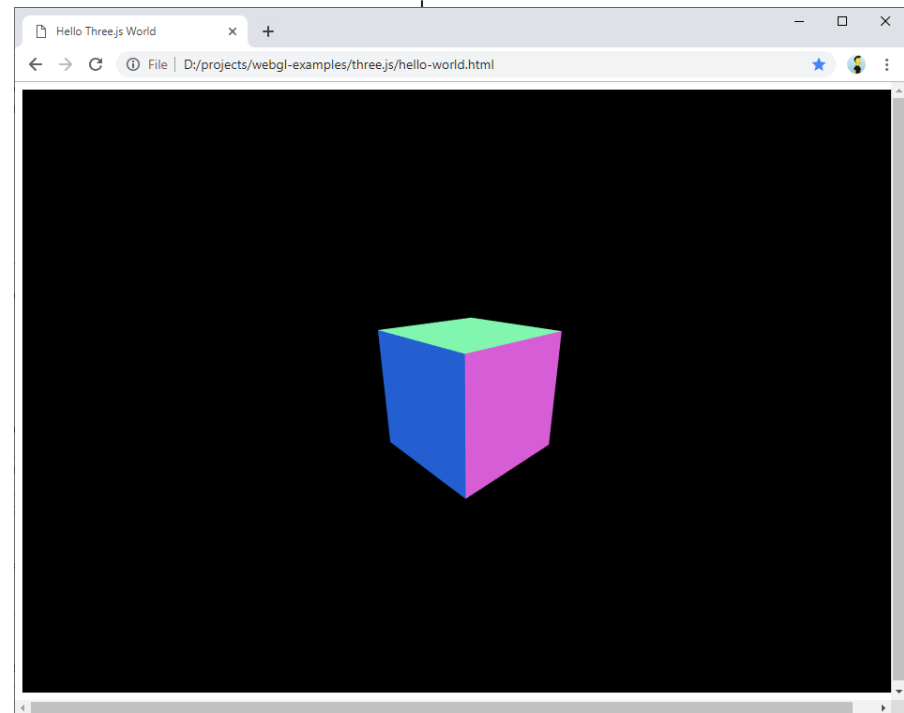
    var renderer = new THREE.WebGLRenderer({
      antialias : true
    });

    renderer.setSize(sceneWidth, sceneHeight);
    document.body.appendChild(renderer.domElement);

    animate(mesh, renderer, scene, camera);
  }

  // ...
</script>
</head>

<body onload="init()">
</body>
</html>
```



2. Conceptos básicos – Hola mundo

```
<script>

  function animate(mesh, renderer, scene, camera) {
    mesh.rotation.x += 0.01;
    mesh.rotation.y += 0.02;

    renderer.render(scene, camera);

    requestAnimationFrame(function() {
      animate(mesh, renderer, scene, camera);
    });
  }

</script>
</head>
```

Como hacemos habitualmente, usamos la función `requestAnimationFrame` para redibujar nuestra escena

2. Conceptos básicos – Hola mundo

```
<!DOCTYPE html>
<html>

<head>
<title>Three.js: Hello World with canvas</title>
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/
<script>
function init() {
  var scene = new THREE.Scene();
  var myCanvas = document.getElementById("myCanvas");
  var sceneWidth = myCanvas.width;
  var sceneHeight = myCanvas.height;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.z = 5;

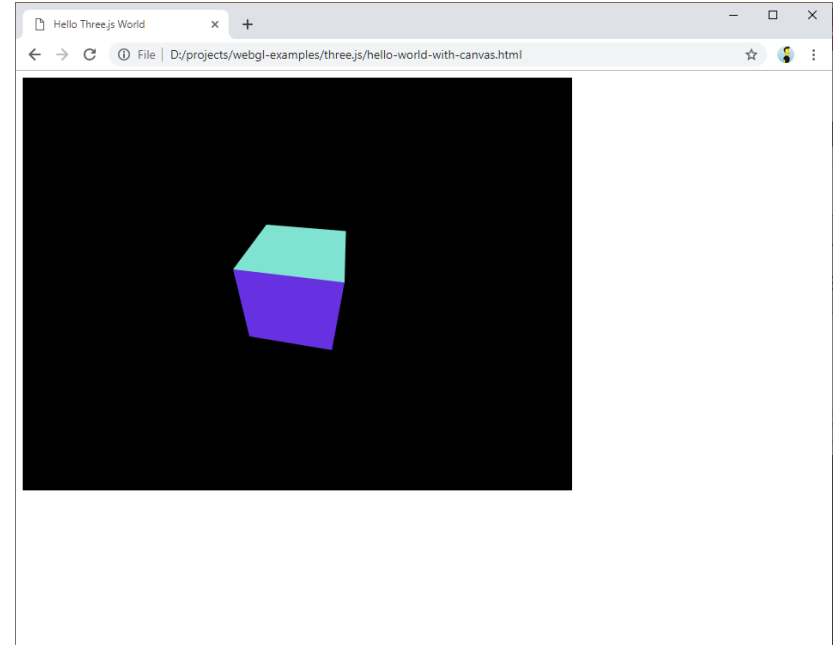
  var geometry = new THREE.BoxGeometry(2, 2, 2);
  var material = new THREE.MeshNormalMaterial();
  var mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);

  var renderer = new THREE.WebGLRenderer({
    antialias : true,
    canvas : myCanvas
  });

  animate(mesh, renderer, scene, camera);
}
// ...
</script>
</head>

<body onload="init()">
  <canvas id="myCanvas" width="640" height="480"></canvas>
</body>
</html>
```

También podemos partir de un canvas ya existente en nuestra página web

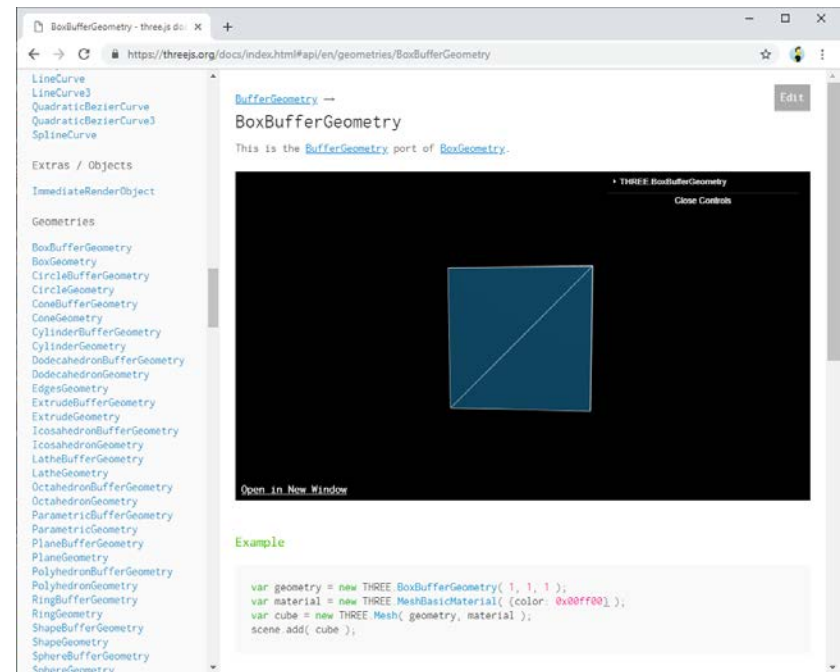


Índice de contenidos

1. Introducción
2. Conceptos básicos
- 3. Geometrías**
4. Texturas
5. Cámaras
6. Iluminación
7. Animación

3. Geometrías

- Three.js ofrece una gran variedad de geometrías listas para ser usadas: esferas, cubos, cilindros, etc
- La documentación de Three.js es una buena referencia para las geometrías



<https://threejs.org/docs/index.html#api/en/geometries/BoxBufferGeometry>

3. Geometrías






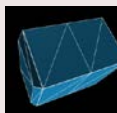

- Hay 2 tipos de geometrías (desde un punto de vista funcional, ambas son iguales):

	<Name>BufferGeometry	<Name>Geometry
Descripción	Geometría implementada con arrays JavaScript de tipo Float32Array	Geometría implementada con objetos propios de Three.js (Vec3, Color)
Ventaja	Más eficiente	Más sencilla de modificar (incluir nuevos vértices, etc.)
Inconveniente	Más compleja de modificar	Menos eficiente

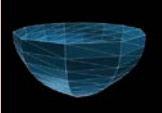

<https://threejs.org/docs/#api/en/core/Geometry>

<https://threejs.org/docs/#api/en/core/BufferGeometry>

3. Geometrías

Geometría	Descripción	Ejemplo
<u>BoxBufferGeometry</u> <u>BoxGeometry</u>	Cubo	
<u>CircleBufferGeometry</u> <u>CircleGeometry</u>	Círculo	
<u>ConeBufferGeometry</u> <u>ConeGeometry</u>	Cono	
<u>CylinderBufferGeometry</u> <u>CylinderGeometry</u>	Cilindro	
<u>DodecahedronBufferGeometry</u> <u>DodecahedronGeometry</u>	Dodecaedro	
<u>ExtrudeBufferGeometry</u> <u>ExtrudeGeometry</u>	Ortoedro con bordes extruidos	
<u>IcosahedronBufferGeometry</u> <u>IcosahedronGeometry</u>	Icosaedro	

3. Geometrías

Geometría	Descripción	Ejemplo
<u>LatheBufferGeometry</u> <u>LatheGeometry</u>	Vaso	
<u>OctahedronBufferGeometry</u> <u>OctahedronGeometry</u>	Octaedro	
<u>ParametricBufferGeometry</u> <u>ParametricGeometry</u>	Función paramétrica (una <u>botella de Klein</u> en el ejemplo)	
<u>PlaneBufferGeometry</u> <u>PlaneGeometry</u>	Plano	
<u>RingBufferGeometry</u> <u>RingGeometry</u>	Anillo	
<u>ShapeBufferGeometry</u> <u>ShapeGeometry</u>	Línea poligonal (un corazón en el ejemplo)	

3. Geometrías

Geometría	Descripción	Ejemplo
<u>SphereBufferGeometry</u> <u>SphereGeometry</u>	Esfera	
<u>TetrahedronBufferGeometry</u> <u>TetrahedronGeometry</u>	Tetraedro	
<u>TextBufferGeometry</u> <u>TextGeometry</u>	Texto	
<u>TorusBufferGeometry</u> <u>TorusGeometry</u>	Toro	
<u>TorusKnotBufferGeometry</u> <u>TorusKnotGeometry</u>	Nudo	
<u>TubeBufferGeometry</u> <u>TubeGeometry</u>	Tubeo	

3. Geometrías

- Las geometrías `WireframeGeometry` y `EdgesGeometry` se usan como auxiliares (*helpers*)

```
<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(3, 3, 3);
  camera.lookAt(scene.position);

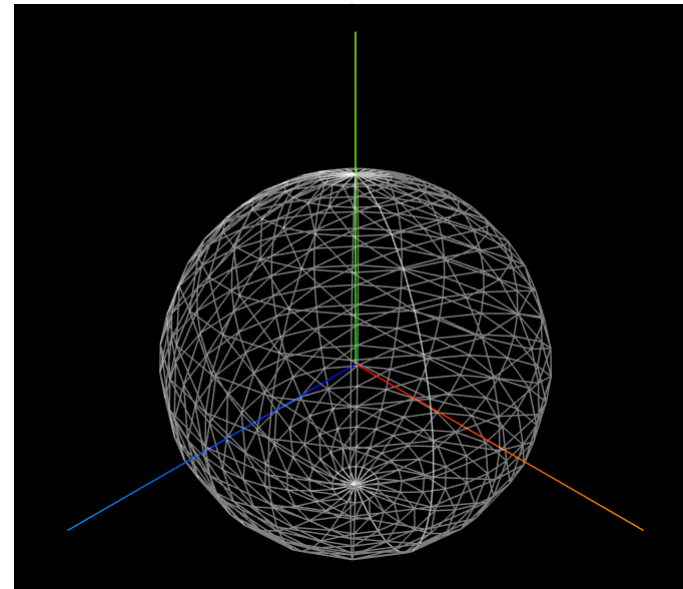
  var geometry = new THREE.SphereBufferGeometry(2, 20, 20);
  var wireframe = new THREE.WireframeGeometry(geometry);
  var sphere = new THREE.LineSegments(wireframe);
  sphere.material.color = { r:1, g:1, b:1 };
  sphere.material.transparent = true;
  sphere.material.opacity = 0.5;
  scene.add(sphere);

  var axes = new THREE.AxesHelper(3);
  scene.add(axes);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });

  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  animate(sphere, renderer, scene, camera);
}
</script>
```



3. Geometrías

• Ejemplo con varias geometrías:

```
<script>
function init() {
  // Scene
  var scene = new THREE.Scene();

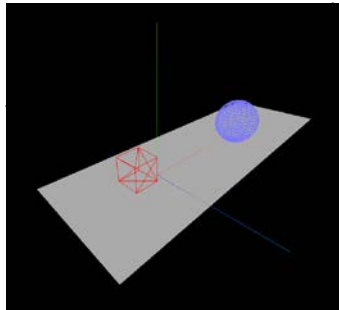
  // Renderer
  var renderer = new THREE.WebGLRenderer();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  // Axes helper
  var axes = new THREE.AxesHelper(20);
  scene.add(axes);

  // Plane
  var planeGeometry = new THREE.PlaneGeometry(60, 20);
  var planeMaterial = new THREE.MeshBasicMaterial({
    color : 0xA8AAAA
  });
  var plane = new THREE.Mesh(planeGeometry, planeMaterial);

  plane.rotation.x = -0.5 * Math.PI;
  plane.position.set(15, 0, 0);
  scene.add(plane);

```



```

// Cube
var cubeGeometry = new THREE.BoxGeometry(4, 4, 4);
var cubeMaterial = new THREE.MeshBasicMaterial({
  color : 0xFF0000,
  wireframe : true
});
var cube = new THREE.Mesh(cubeGeometry, cubeMaterial);

cube.position.set(-4, 3, 0);
scene.add(cube);

// Sphere
var sphereGeometry = new THREE.SphereGeometry(4, 20, 20);
var sphereMaterial = new THREE.MeshBasicMaterial({
  color : 0x7777FF,
  wireframe : true
});
var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);

sphere.position.set(20, 4, 2);
scene.add(sphere);

// Camera
var camera = new THREE.PerspectiveCamera(90,
  sceneWidth / sceneHeight, 0.01, 100);
camera.position.set(-20, 20, 20);
camera.lookAt(scene.position);

// Render
renderer.render(scene, camera);
}
</script>

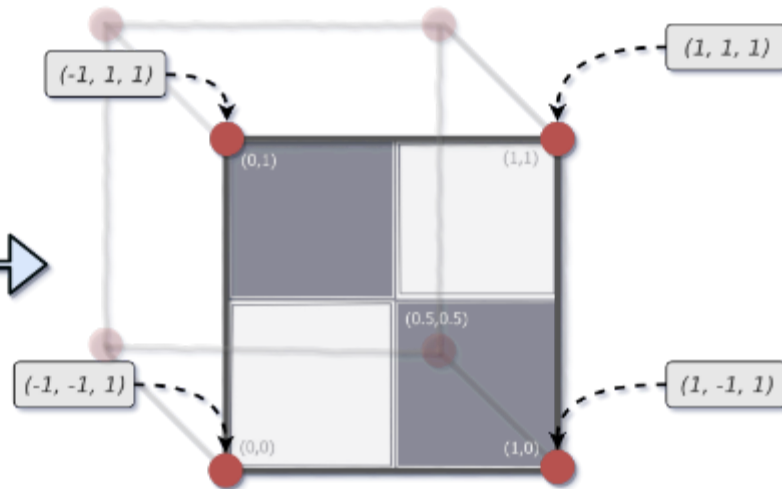
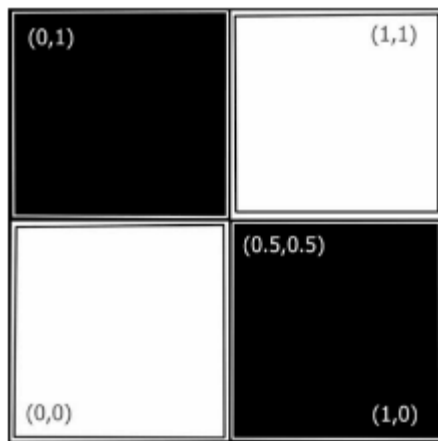
```

Índice de contenidos

1. Introducción
2. Conceptos básicos
3. Geometrías
- 4. Texturas**
5. Cámaras
6. Iluminación
7. Animación

4. Texturas

- El proceso de texturización interno de Three.js es el que hemos visto en WebGL (las coordenadas u,v de una imagen se mapean en un conjunto de vértices)



4. Texturas

- Three.js proporciona una colección de objetos textura con diferentes propiedades
- Algunas de las más importantes son:

Textura	Descripción	Afectada por luz
<u>MeshBasicMaterial</u>	Textura básica	No
<u>MeshLambertMaterial</u>	Textura para superficies no brillantes	Sí
<u>MeshPhongMaterial</u>	Textura para superficies brillantes	Sí
<u>MeshStandardMaterial</u>	Textura para superficies con brillo metálico	Sí
<u>MeshPhysicalMaterial</u>	Textura para superficies con brillo metálico con mayor control de la reflexividad	Sí

4. Texturas

```
<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.z = 5;

  var geometry = new THREE.BoxGeometry(2, 2, 2);

  var texture = new THREE.TextureLoader().load("texture.png")
  var material = new THREE.MeshBasicMaterial({
    map : texture
  });

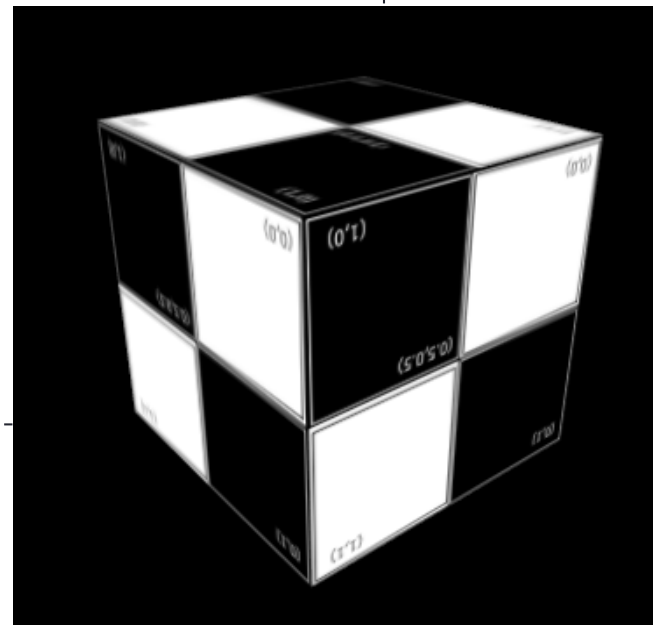
  var mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });

  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  animate(mesh, renderer, scene, camera);
}
</script>
```

En la textura MeshBasicMaterial
no es necesario incluir un punto
de luz en la escena



4. Texturas

```
<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.z = 5;

  var geometry = new THREE.BoxGeometry(2, 2, 2);

  var texture = new THREE.TextureLoader().load("texture.png")
  var material = new THREE.MeshStandardMaterial({
    map : texture
  });

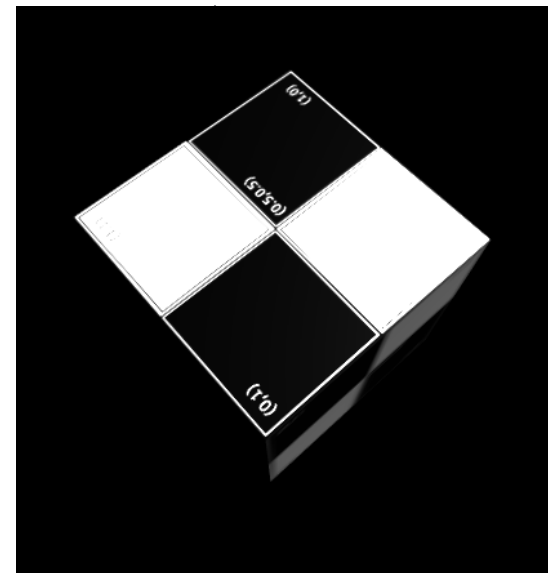
  var mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);

  var light = new THREE.DirectionalLight(0xffffff, 3.0);
  light.position.set(10, 10, 10);
  scene.add(light);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });

  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  animate(mesh, renderer, scene, camera);
}
</script>
```



Para el resto de texturas sí que es necesario incluir un punto de luz en la escena

4. Texturas

```

<!DOCTYPE html>
<html>
<head>
<title>Three.js: Repeated textures (basic material)</title>
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/three.min.js"></script>

<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.z = 5;

  var geometry = new THREE.BoxGeometry(2, 2, 2);

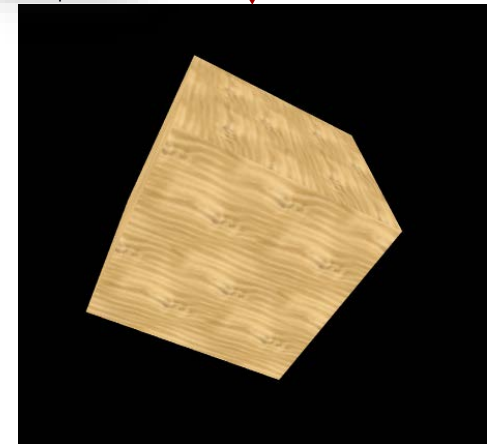
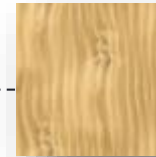
  var texture = new THREE.TextureLoader().load("wood.png")
  var material = new THREE.MeshBasicMaterial({
    map : texture
  });
  material.map.repeat.set(2, 2);
  material.map.wrapS = THREE.RepeatWrapping;
  material.map.wrapT = THREE.RepeatWrapping;
  material.side = THREE.DoubleSide;
  var mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  animate(mesh, renderer, scene, camera);
}
// ...

```

wood.png



En este ejemplo se realizan configuraciones adicionales en la textura:

- Repetición (2, 2): Método repeat
- WrapS (RepeatWrapping): Método wrapS
- WrapT (RepeatWrapping): Método wrapT
- Cara visible (DoubleSide): Método side

4. Texturas

- Al igual que ocurría en WebGL, la carga de imágenes en Three.js es **asíncrona**
- El método load de TextureLoader acepta tres funciones callback

```
.load ( url : String, onLoad : Function, onProgress : Function, onError : Function ) : Texture
```

Callback que se ejecuta cuando termina la carga de la imagen

Callback que se ejecuta durante la carga de la imagen

Callback que se ejecuta en caso de error

- Las texturas en los ejemplos que hemos visto hasta ahora se visualizan correctamente porque se hace uso de requestAnimationFrame

<https://threejs.org/docs/index.html#api/en/loaders/TextureLoader>

Índice de contenidos

1. Introducción
2. Conceptos básicos
3. Geometrías
4. Texturas
- 5. Cámaras**
 - I. Perspectiva**
 - II. Ortogonal**
6. Iluminación
7. Animación

5. Cámaras - Perspectiva

- Ejemplo con cámara en perspectiva y controles:

```
<!DOCTYPE html>
<html>

<head>
<title>Three.js: Perspective camera with controls</title>

<style>
* {
  margin: 0px;
}
</style>

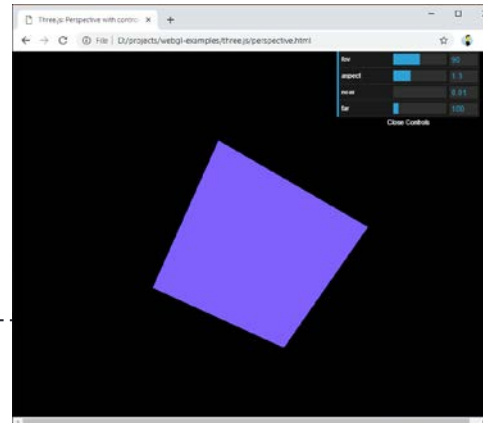
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/three.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/dat-gui/0.7.6/dat.gui.min.js"></script>

<script>
  // ...
</script>
</head>

<body onload="init()">
</body>

</html>
```

En este ejemplo añadimos un cuadro de controles usando la librería **dat.gui** (<https://github.com/dataarts/dat.gui>)



5. Cámaras - Perspectiva

- Ejemplo con cámara en perspectiva y controles:

```
// Control
var control = new function() {
  this.fov = camera.fov;
  this.aspect = camera.aspect;
  this.far = camera.far;
  this.near = camera.near;

  this.update = function() {
    camera.fov = control.fov;
    camera.aspect = control.aspect;
    camera.near = control.near;
    camera.far = control.far;
    camera.updateProjectionMatrix();
  }
};

var gui = new dat.GUI();
gui.add(control, 'fov', 0, 180).onChange(control.update);
gui.add(control, 'aspect', 0, 4).onChange(control.update);
gui.add(control, 'near', 0, 40).onChange(control.update);
gui.add(control, 'far', 0, 1000).onChange(control.update);
```

Creamos un objeto JavaScript con las propiedades que queremos que sean modificables de la interfaz de usuario

5. Cámaras - Ortogonal

- Ejemplo con cámara ortogonal:

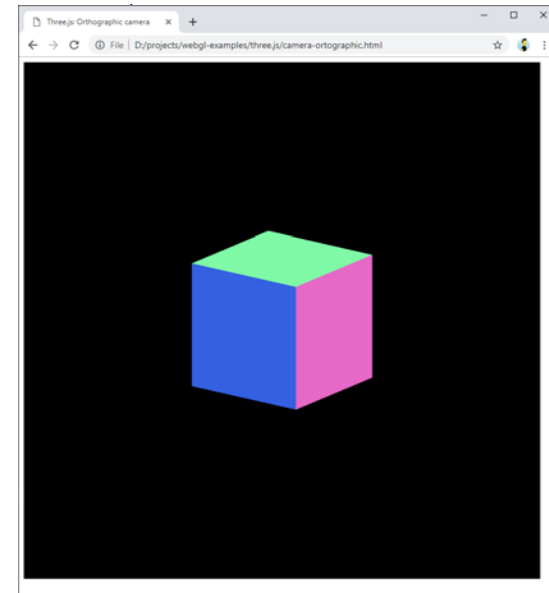
```
<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = 800;
  var sceneHeight = 800;

  var camera = new THREE.OrthographicCamera();
  camera.left = sceneWidth / -2;
  camera.right = sceneWidth / 2;
  camera.top = sceneHeight / 2;
  camera.bottom = sceneHeight / -2;
  camera.near = 0.1;
  camera.far = 300;
  camera.position.z = 10;

  var geometry = new THREE.BoxGeometry(0.5, 0.5, 0.5);
  var material = new THREE.MeshNormalMaterial();
  var mesh = new THREE.Mesh(geometry, material);
  scene.add(mesh);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  animate(mesh, renderer, scene, camera);
}
</script>
```



Índice de contenidos

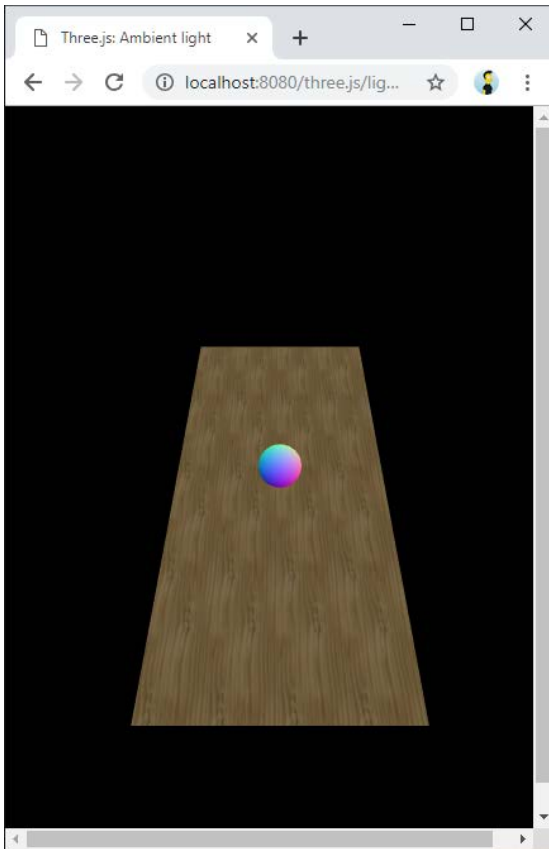
1. Introducción
2. Conceptos básicos
3. Geometrías
4. Texturas
5. Cámaras
- 6. Iluminación**
 - I. Luz ambiental**
 - II. Luz direccional**
 - III. Luz puntal**
7. Animación

6. Iluminación

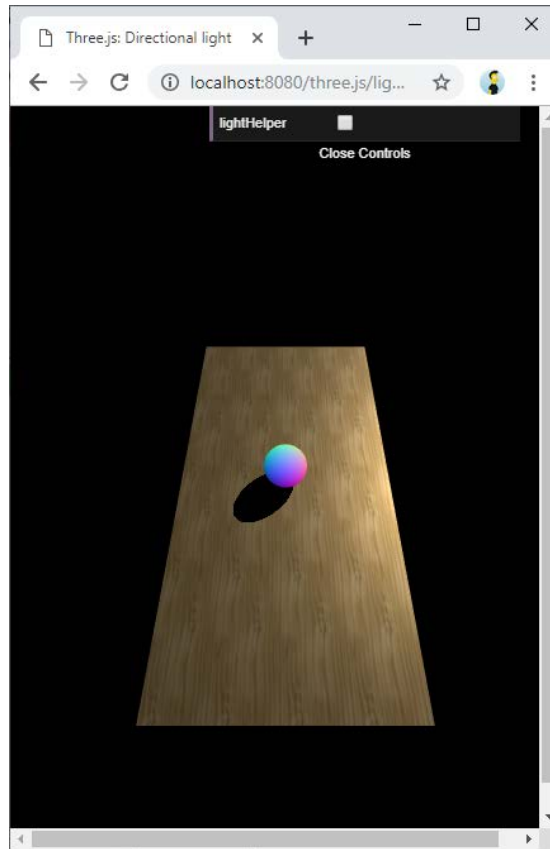
- Three.js proporciona una colección de objetos para crear puntos de luz con diferentes propiedades
- Algunas de las más importantes son:

Textura	Descripción	Produce sombra
<u>AmbientLight</u>	Luz ambiental	No
<u>DirectionalLight</u>	Luz direccional	Sí
<u>PointLight</u>	Luz puntual	Sí

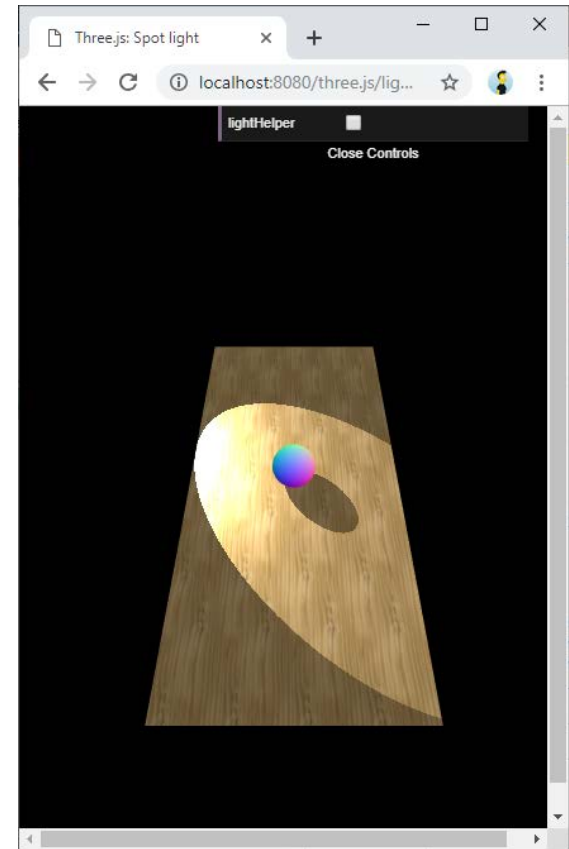
6. Iluminación



AmbientLight



DirectionalLight



PointLight

6. Iluminación - Luz ambiental

```
<!DOCTYPE html>
<html>

<head>
<title>Three.js: Ambient light</title>

<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/103/three.min.js"></script>

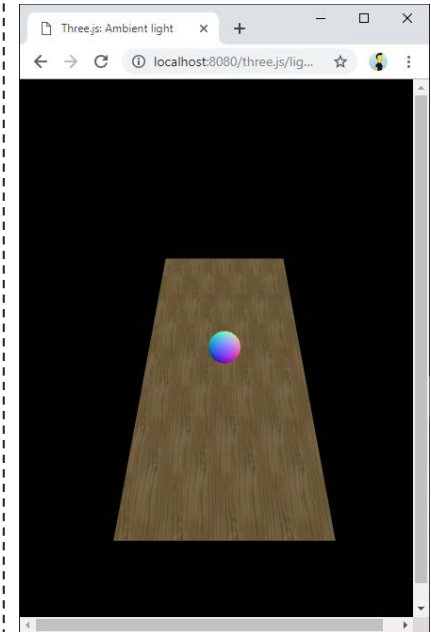
<script>
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(0, -10, 15);
  camera.lookAt(scene.position);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);
  var render = function() {
    renderer.render(scene, camera);
  };

  addLight(scene);
  addSphere(scene);
  addFloor(scene, render);
}

// ...
```



Creamos una variable que contendrá la función para renderizar la escena (se usará como callback en la carga de la textura)

6. Iluminación - Luz ambiental

```
// ...  
  
function addLight(scene) {  
  var light = new THREE.AmbientLight(0xFFFFFF);  
  scene.add(light);  
}  
  
function addSphere(scene) {  
  var geometry = new THREE.SphereGeometry(1, 20, 20);  
  var material = new THREE.MeshNormalMaterial();  
  var mesh = new THREE.Mesh(geometry, material);  
  mesh.position.z = 1;  
  scene.add(mesh);  
}  
  
function addFloor(scene, render) {  
  var geometry = new THREE.PlaneGeometry(10, 20);  
  var texture = new THREE.TextureLoader().load("wood.png", render);  
  var material = new THREE.MeshPhysicalMaterial({  
    map : texture  
  });  
  material.map.wrapS = material.map.wrapT = THREE.RepeatWrapping;  
  material.map.repeat.set(4, 4);  
  material.side = THREE.DoubleSide;  
  var mesh = new THREE.Mesh(geometry, material);  
  scene.add(mesh);  
}  
  
</script>  
</head>  
  
<body onload="init()">  
</body>  
  
</html>
```

Añadimos luz ambiental
blanca (0xFFFFFF)

Añadimos una esfera

Añadimos un plano

6. Iluminación - Luz direccional

```
function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(0, -10, 15);
  camera.lookAt(scene.position);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.shadowMap.enabled = true;
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);
  var render = function() {
    renderer.render(scene, camera);
  };

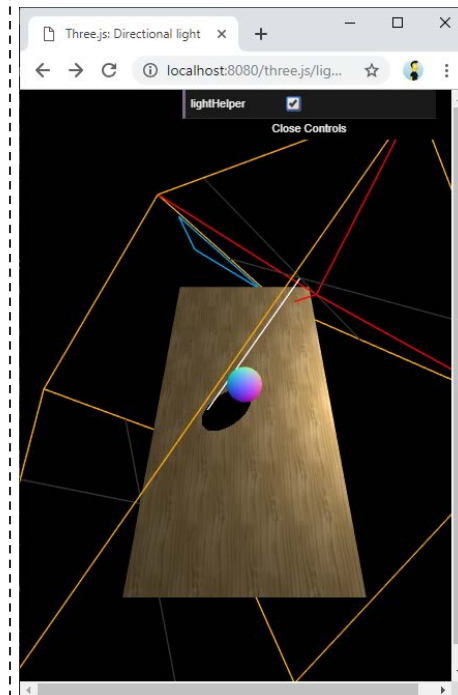
  var helper = addLight(scene);
  addSphere(scene);
  addFloor(scene, render);

  // Control
  var control = new function() {
    this.lightHelper = false;

    this.update = function() {
      if (control.lightHelper) {
        scene.add(helper);
      } else {
        scene.remove(helper);
      }
      render();
    }
  };
  var gui = new dat.GUI();
  gui.add(control, 'lightHelper', true, false).onChange(control.update);
}
```

Habilitamos el efecto de sombra en el renderer (shadowMap.enabled = true)

Añadimos un control para activar/desactivar el *helper* para la zona de sombra



6. Iluminación - Luz direccional

```
function addLight(scene) {  
  var light = new THREE.DirectionalLight();  
  light.position.set(4, 4, 4);  
  light.castShadow = true;  
  light.shadow.camera.near = 0;  
  light.shadow.camera.far = 16;  
  light.shadow.camera.left = -8;  
  light.shadow.camera.right = 5;  
  light.shadow.camera.top = 10;  
  light.shadow.camera.bottom = -10;  
  light.shadow.mapSize.width = 4096;  
  light.shadow.mapSize.height = 4096;  
  scene.add(light);  
  var helper = new THREE.CameraHelper(light.shadow.camera);  
  return helper;  
}
```

Añadimos luz direccional que produce sombra (`castShadow = true`). El área de sombra con mismos parámetros usados en la proyección ortogonal (`near`, `far`, `left`, `right`, `top`, `bottom`)

```
function addSphere(scene) {  
  var geometry = new THREE.SphereGeometry(1, 20, 20);  
  var material = new THREE.MeshNormalMaterial();  
  var mesh = new THREE.Mesh(geometry, material);  
  mesh.position.z = 1;  
  mesh.castShadow = true;  
  scene.add(mesh);  
}
```

Añadimos una esfera que produce sombra (`castShadow = true`)

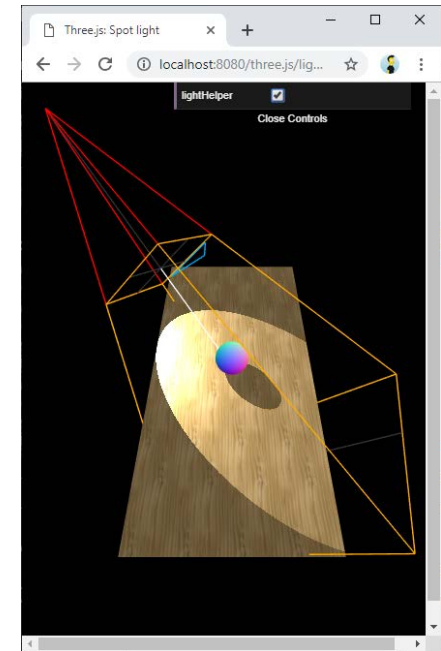
```
function addFloor(scene, render) {  
  var geometry = new THREE.PlaneGeometry(10, 20);  
  var texture = new THREE.TextureLoader().load("wood.png", render);  
  var material = new THREE.MeshPhysicalMaterial({  
    map : texture  
  });  
  material.map.wrapS = material.map.wrapT = THREE.RepeatWrapping;  
  material.map.repeat.set(4, 4);  
  material.side = THREE.DoubleSide;  
  var mesh = new THREE.Mesh(geometry, material);  
  mesh.receiveShadow = true;  
  scene.add(mesh);  
}
```

Añadimos un plano que recibe sombra (`receiveShadow = true`)

6. Iluminación - Luz puntual

```
function addLight(scene) {  
  var spotLight = new THREE.SpotLight();  
  spotLight.position.set(-10, 10, 10);  
  spotLight.angle = Math.PI / 12;  
  spotLight.castShadow = true;  
  spotLight.shadow.camera.visible = true;  
  spotLight.shadow.camera.near = 10;  
  spotLight.shadow.camera.far = 25;  
  spotLight.shadow.camera.fov = 30;  
  spotLight.shadow.camera.aspect = 1;  
  spotLight.shadow.mapSize.width = 4096;  
  spotLight.shadow.mapSize.height = 4096;  
  scene.add(spotLight);  
  
  var ambientLight = new THREE.AmbientLight(0xFFFFFF);  
  scene.add(ambientLight);  
  
  var helper = new THREE.CameraHelper(spotLight.shadow.camera);  
  return helper;  
}
```

Añadimos luz puntual que produce sombra (`castShadow = true`). El área de sombra con mismos parámetros usados en la proyección en perspectiva (`near`, `far`, `fov`, `aspect`). En este ejemplo además añadimos luz ambiental (para poder visualizar fuera del punto de luz)



Índice de contenidos

1. Introducción
2. Conceptos básicos
3. Geometrías
4. Texturas
5. Cámaras
6. Iluminación
- 7. Animación**
 - I. Objeto en movimiento**
 - II. Luz en movimiento**
 - III. Detección de colisiones**
 - IV. Motor físico**

7. Animación

- Para animar una escena usamos la función JavaScript `requestAnimationFrame()`
- Cuando el navegador puede redibujar, ejecutaremos una función
- Típicamente modificaremos alguno de los atributos (posición, color, etc.) de uno o varios elementos de escena (mallas, luz, etc.)
- La interactividad se consigue capturando eventos de usuario y usándolos para caracterizar la animación

7. Animación - Objeto en movimiento

```
var counter = 0;

function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(0, -10, 15);
  camera.lookAt(scene.position);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.shadowMap.enabled = true;
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

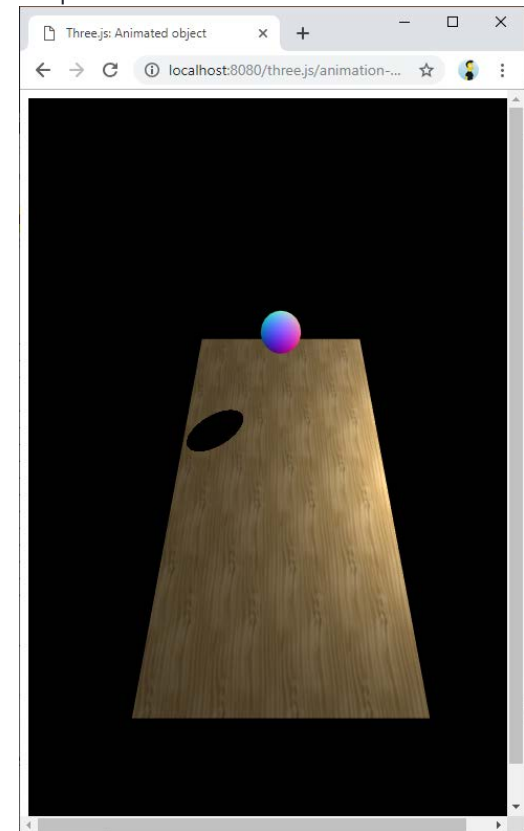
  var light = getLight();
  var sphere = getSphere();
  var floor = getFloor();
  scene.add(light);
  scene.add(sphere);
  scene.add(floor);

  animate(sphere, renderer, scene, camera);
}

function animate(sphere, renderer, scene, camera) {
  sphere.position.y = 10 * Math.cos(counter);
  sphere.position.z = 1 + 3 * Math.abs(Math.sin(counter))

  renderer.render(scene, camera);
  counter += 0.02;

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



Creamos una animación estacionaria usando funciones sinusoidales de las coordenadas y, z de la esfera

7. Animación - Luz en movimiento

```
var speed = 0.2;

function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(0, -10, 15);
  camera.lookAt(scene.position);

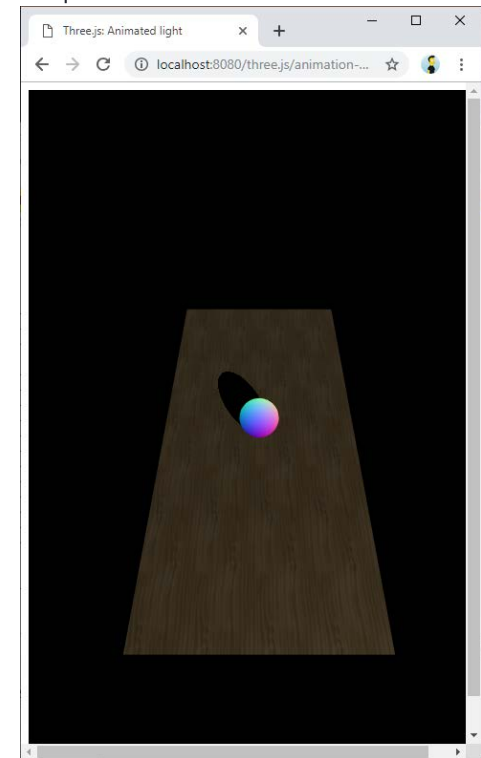
  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.shadowMap.enabled = true;
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

  var light = getLight();
  var sphere = getSphere();
  var floor = getFloor();
  scene.add(light);
  scene.add(sphere);
  scene.add(floor);

  animate(light, renderer, scene, camera);
}

function animate(light, renderer, scene, camera) {
  if (Math.abs(light.position.y) > 8) {
    speed *= -1;
  }
  light.position.y += speed;
  renderer.render(scene, camera);

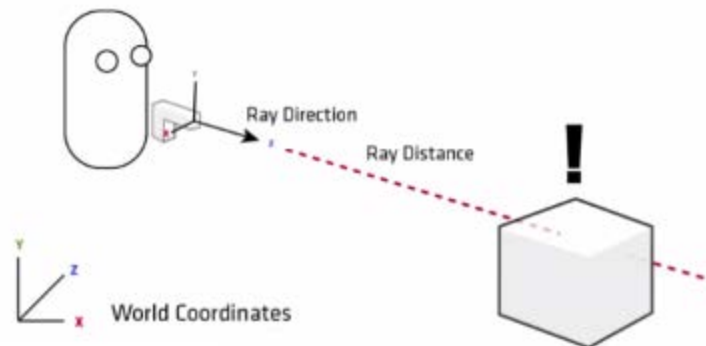
  requestAnimationFrame(function() {
    animate(light, renderer, scene, camera);
  });
}
```



Variamos la coordenada y de la dirección de la luz dentro de un rango y a una velocidad determinada

7. Animación - Detección de colisión

- La detección de colisiones en Three.js se puede conseguir mediante **ray casting**
- Esta técnica se basa en el cálculo de la intersección rayo-superficie
- Three.js proporciona la función Raycaster



7. Animación - Detección de colisión

```

var stepX = 0.15;
var stepY = 0.25;

function init() {
  var scene = new THREE.Scene();
  var sceneWidth = window.innerWidth;
  var sceneHeight = window.innerHeight;

  var camera = new THREE.PerspectiveCamera(90, sceneWidth / sceneHeight, 0.01, 100);
  camera.position.set(0, -10, 15);
  camera.lookAt(scene.position);

  var renderer = new THREE.WebGLRenderer({
    antialias : true
  });
  renderer.shadowMap.enabled = true;
  renderer.setSize(sceneWidth, sceneHeight);
  document.body.appendChild(renderer.domElement);

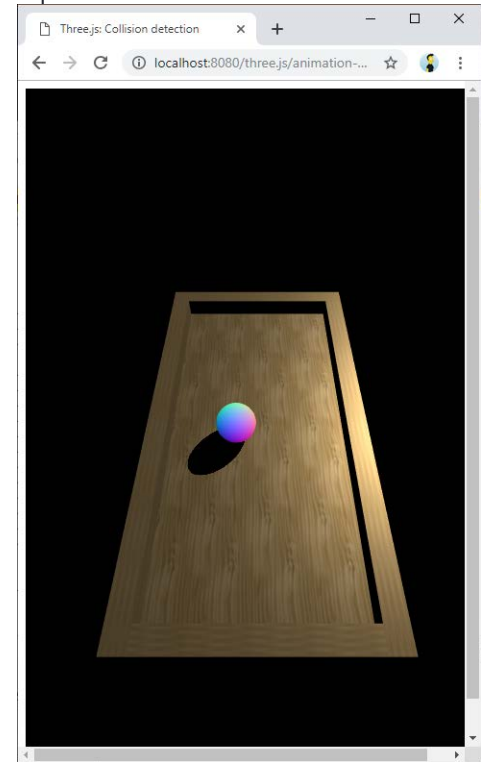
  var light = getLight();
  var leftBorder = getBorder("left", 1, 20, 2, -5, 0, 0);
  var rightBorder = getBorder("right", 1, 20, 2, 5, 0, 0);
  var topBorder = getBorder("top", 11, 1, 2, 0, 10, 0);
  var downBorder = getBorder("down", 9, 1, 2, 0, -9.5, 0);
  var sphere = getSphere();
  var floor = getFloor();

  scene.add(light);
  scene.add(leftBorder);
  scene.add(rightBorder);
  scene.add(topBorder);
  scene.add(downBorder);
  scene.add(sphere);
  scene.add(floor);

  var borders = [ leftBorder, rightBorder, topBorder, downBorder ];

  animate(sphere, borders, renderer, scene, camera);
}

```



En este ejemplo añadimos bordes al plano original. Se guardan estos 4 bordes un array para realizar el ray casting

7. Animación - Detección de colisión

```
function animate(sphere, borders, renderer, scene, camera) {
  checkCollision(sphere, borders);

  sphere.position.x += stepX;
  sphere.position.y += stepY;

  renderer.render(scene, camera);

  requestAnimationFrame(function() {
    animate(sphere, borders, renderer, scene, camera);
  });
}

function checkCollision(sphere, borders) {
  var originPosition = sphere.position.clone();

  for (var i = 0; i < sphere.geometry.vertices.length; i++) {
    var localVertex = sphere.geometry.vertices[i].clone();
    var globalVertex = localVertex.applyMatrix4(sphere.matrix);
    var directionVector = globalVertex.sub(sphere.position);
    var ray = new THREE.Raycaster(originPosition, directionVector.clone().normalize());
    var collisionResults = ray.intersectObjects(borders);
    if (collisionResults.length > 0 && collisionResults[0].distance < directionVector.length()) {
      // Collision detected
      if (collisionResults[0].object.name == "left" || collisionResults[0].object.name == "right") {
        stepX *= -1;
      }
      if (collisionResults[0].object.name == "down" || collisionResults[0].object.name == "top") {
        stepY *= -1;
      }
      break;
    }
  }
}
```

Antes de realizar la animación, llamamos a la función `checkCollision()`

En primer lugar calculamos un vector en la dirección de cada vértice de la malla que queremos detectar colisión

Creamos un objeto `THREE.Raycaster` y calculamos la intersección con la lista de objetos que puede colisionar

7. Animación - Motor físico

- Un **motor físico** es un software capaz de realizar simulaciones de leyes físicas tales como la dinámica, movimiento, colisión, etc.
- Three.js no proporciona de forma nativa ningún motor física
- Existen diferentes librerías que pueden ser utilizadas como motor físico de Three.js, por ejemplo:
 - [Cannon.js](#)
 - [Physijs](#) (basado en [ammo.js](#)). Vamos a ver un ejemplo de esta librería

7. Animación - Motor físico

```
<!DOCTYPE html>
<html>

<head>
<title>Three.js: Using a physics engine</title>

<style>
* {
margin: 0px;
}
</style>

<script src="https://cdn.jsdelivr.net/ajax/libs/three.js/103/three.min.js"></script>
<script src="https://cdn.jsdelivr.net/ajax/libs/dat-gui/0.7.6/dat.gui.min.js"></script>
<script src="js/physi.js"></script>

<script>
  Physijs.scripts.worker = 'js/physijs_worker.js';

  // ...

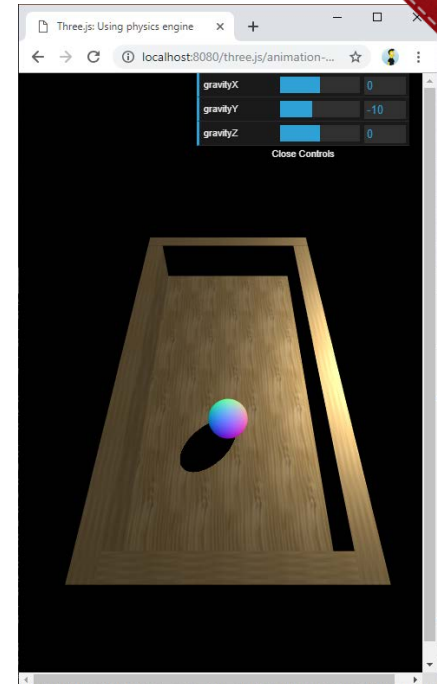
</script>
</head>

<body onload="init()">
</body>

</html>
```

Hemos incorporado las librerías `physi.js`, `physijs_worker.js` y `ammo.js` en nuestro proyecto

Los cálculos realizados por `physi.js` se realizan en un hilo secundario (*web worker*)



7. Animación - Motor físico

```
function init() {  
  var scene = new Physijs.Scene;  
  scene.setGravity(new THREE.Vector3(0, -10, 0));  
  
  var sceneWidth = window.innerWidth;  
  var sceneHeight = window.innerHeight;  
  
  var camera = new THREE.PerspectiveCamera(90, sceneWidth, sceneHeight, 1);  
  camera.position.set(0, -10, 15);  
  camera.lookAt(scene.position);  
  
  var renderer = new THREE.WebGLRenderer({  
    antialias : true  
  });  
  renderer.shadowMap.enabled = true;  
  renderer.setSize(sceneWidth, sceneHeight);  
  document.body.appendChild(renderer.domElement);  
  
  var light = getLight();  
  var leftBorder = getBorder("left", 1, 20, 5, -5, 0, 0);  
  var rightBorder = getBorder("right", 1, 20, 5, 5, 0, 0);  
  var topBorder = getBorder("top", 11, 1, 5, 0, 10, 0);  
  var downBorder = getBorder("down", 9, 1, 5, 0, -9.5, 0);  
  var sphere = getSphere();  
  var floor = getFloor();  
  
  scene.add(light);  
  scene.add(leftBorder);  
  scene.add(rightBorder);  
  scene.add(topBorder);  
  scene.add(downBorder);  
  scene.add(sphere);  
  scene.add(floor);  
  
  // Control
```

Nuestra escena será ahora de tipo `Physijs.Scene`

Se simula la fuerza de la gravedad con un vector en la escena

Añadimos un control para modificar la gravedad desde la interfaz de usuario

7. Animación - Motor físico

```
function getSphere() {
  var geometry = new THREE.SphereGeometry(1, 20, 20);
  var material = new THREE.MeshNormalMaterial();
  var mesh = new Physijs.BoxMesh(geometry, material);
  mesh.position.z = 1;
  mesh.castShadow = true;
  mesh.name = "sphere";
  mesh.addEventListener('collision', function(otherObject) {
    console.log(otherObject.name);
  });

  return mesh;
}

function getFloor() {
  var geometry = new THREE.PlaneGeometry(10, 20);
  var mesh = new Physijs.BoxMesh(geometry, getWoodMaterial(), 0);
  mesh.receiveShadow = true;
  mesh.name = "floor";
  return mesh;
}

function getBorder(name, x, y, z, posX, posY, posZ) {
  var geometry = new THREE.BoxGeometry(x, y, z);
  var mesh = new Physijs.BoxMesh(geometry, getWoodMaterial(), 0);
  mesh.receiveShadow = true;
  mesh.position.set(posX, posY, posZ);
  mesh.name = name;
  return mesh;
}
```

Las mallas que estarán controladas por el motor físico serán de tipo `Physijs.BoxMesh`

La colisión en los objetos `Physijs.BoxMesh` se consigue registrando un *eventListener*

Los parámetros de `Physijs.BoxMesh` son:

- Geometría
- Material
- Masa (0 si queremos que no le afecte la gravedad ni las colisiones)