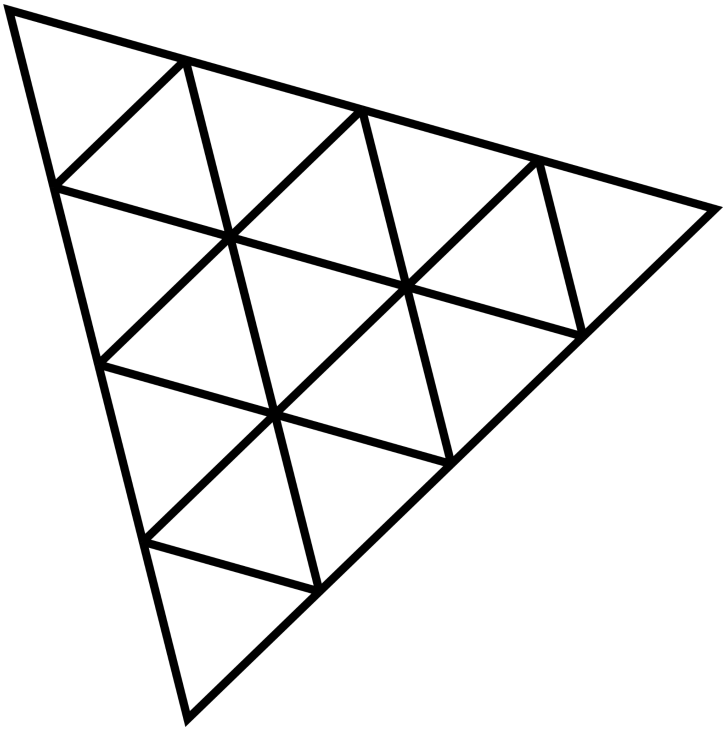


ThreeJS

Unos Apuntes

Instalación

Estructura del proyecto

Todo proyecto three.js necesita al menos un archivo HTML para definir la página web y un archivo JavaScript para ejecutar el código three.js. Las opciones de estructura y nombres que se indican a continuación no son obligatorias, pero se usarán a lo largo de esta guía para mantener la coherencia.

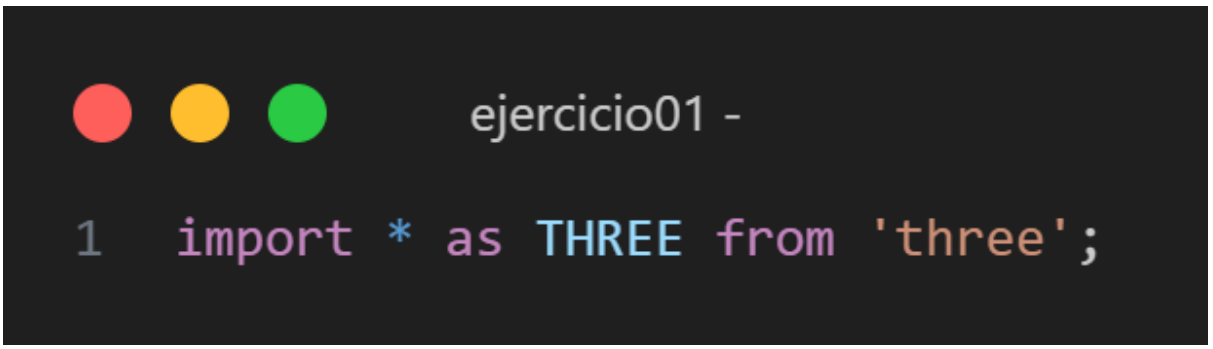
index.html

ejercicio01 - index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

creamos el archivo

main.js



Importar desde una CDN

Desarrollo

La instalación sin herramientas de compilación requerirá algunos cambios en la estructura del proyecto indicada anteriormente.

1. Importamos el código de 'three' (un paquete npm) en *main.js* , y los navegadores web no lo entienden. En *index.html,* necesitaremos agregar un mapa de importación que defina dónde obtener el paquete. Coloque el código a continuación dentro de la etiqueta *<head></head>* , después de los estilos.

ejercicio01 - index.html

```
  <script type="importmap">
{
  "imports": {
    "three": "https://cdn.jsdelivr.net/npm/three@<version>/build/three.module.js",
    "three/addons/": "https://cdn.jsdelivr.net/npm/three@<version>/examples/jsm/"
  }
}
</script>
```

No olvides reemplazar *<version>* con la versión actual de three.js, como *"v0.149.0"* . La versión más reciente se encuentra en la lista de versiones de npm .

Addons

three.js incluye de fábrica los fundamentos de un motor 3D. Otros componentes de three.js, como controles, cargadores y efectos de posprocesamiento, se encuentran en el directorio addons/ . No es necesario instalar los complementos *por* separado, pero sí importarlos *por* separado.

El siguiente ejemplo muestra cómo importar three.js con los complementos `OrbitControls` y `GLTFLoader`. Si es necesario, esto también se mencionará en la documentación o los ejemplos de cada complemento.

ejercicio01 - main.js

```
import * as THREE from 'three';
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js';

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

index completo

ejercicio01 - index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="./css/styles.css">
  <title>Cubo en ThreeJS</title>

  <script type="importmap">
    {
      "imports": {
        "three": "https://cdn.jsdelivr.net/npm/three@0.179.1/build/three.module.js",
        "three/addons/": "https://cdn.jsdelivr.net/npm/three@0.179.1/examples/jsm/"
      }
    }
  </script>

</head>
<body>
  <script type="module" src="./js/main.js"></script>
</body>
</html>
```

main completo

ejercicio01 - main.js

```
import * as THREE from 'three';

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 );
renderer.setAnimationLoop( animate );
document.body.appendChild( renderer.domElement );

const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;

function animate() {

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

  renderer.render( scene, camera );

}
```

Creando una escena

El objetivo de esta sección es ofrecer una breve introducción a three.js. Comenzaremos configurando una escena con un cubo giratorio. Al final de la página encontrará un ejemplo práctico por si tiene alguna dificultad o necesita ayuda.

Antes de empezar

Si aún no lo has hecho, consulta la guía de `Instalación`. Damos por hecho que ya configuraste la misma estructura de proyecto (incluyendo *index.html* y *main.js*), instalaste three.js y estás ejecutando una herramienta de compilación o usando un servidor local con una CDN y mapas de importación.

Creando la escena

Para poder mostrar cualquier cosa con three.js, necesitamos tres cosas: escena, cámara y renderizador, para que podamos renderizar la escena con la cámara.

main.js —

```
import * as THREE from 'three';

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
);

cument.body.appendChild( renderer.domElement );
```

Tomemos un momento para explicar qué sucede aquí. Ya hemos configurado la escena, la cámara y el renderizador.

Hay varias cámaras diferentes en three.js. Por ahora, usemos una `PerspectiveCamera`.

El primer atributo es el `field of view` FOV (campo de visión). Es la extensión de la escena que se ve en la pantalla en un momento dado. El valor se expresa en grados.

El segundo es el `aspect ratio`. Casi siempre conviene usar el ancho del elemento dividido por la altura; de lo contrario, se obtendrá el mismo resultado que al reproducir películas antiguas en un televisor de pantalla ancha: la imagen se ve comprimida.

Los siguientes dos atributos son el plano de recorte `near` y `far` el plano de recorte. Esto significa que los objetos que estén más lejos de la cámara que el valor de [Aquí falta información `far`] o más cerca que [`near` Aquí falta información] no se renderizarán. No tienes que preocuparte por esto ahora, pero quizás quieras usar otros valores en tus aplicaciones para obtener un mejor rendimiento.

A continuación, está el renderizador. Además de crear la instancia del renderizador, también debemos definir el tamaño con el que queremos que

renderice nuestra aplicación. Es recomendable usar el ancho y el alto del área que queremos rellenar con nuestra aplicación; en este caso, el ancho y el alto de la ventana del navegador. Para aplicaciones con un alto rendimiento, también puedes asignar `setSize` valores más pequeños, como `window.innerWidth/2` y `window.innerHeight/2`, lo que hará que la aplicación se renderice a un cuarto de su tamaño.

Si desea mantener el tamaño de su aplicación, pero renderizarla a una resolución menor, puede hacerlo llamando `setSize` con `false` `updateStyle` (el tercer argumento). Por ejemplo, `` `setSize(window.innerWidth/2, window.innerHeight/2, false)` renderizará su aplicación a la mitad de la resolución, dado que su <canvas> tiene el 100% de ancho y alto.

Por último, añadimos el `renderer` elemento a nuestro documento HTML. Se trata de un elemento <canvas> que el renderizador utiliza para mostrarnos la escena.

"Está bien, pero ¿dónde está el cubo que prometiste?" Añádalo ahora.

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color:
0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;
```

Para crear un cubo, necesitamos un objeto `BoxGeometry`. Este es un objeto que contiene todos los puntos (`vertices`) y el relleno (`faces`) del cubo. Analizaremos esto con más detalle más adelante.

Además de la geometría, necesitamos un material para colorearla. Three.js incluye varios materiales, pero `MeshBasicMaterial` por ahora nos centraremos en [texto incoherente]. Todos los materiales toman un objeto de propiedades que se les aplicará. Para simplificar, solo proporcionamos un atributo de color de [texto incoherente] `0x00ff00`, que es verde. Esto

funciona de la misma forma que los colores en CSS o Photoshop ([`hex colors` texto incoherente].

El tercer elemento que necesitamos es un `Mesh`. Una malla es un objeto que toma una geometría y le aplica un material, que luego podemos insertar en nuestra escena y mover libremente.

Por defecto, al llamar a `scene.add()`, el elemento que añadimos se añadirá a las coordenadas `(0,0,0)`. Esto haría que la cámara y el cubo estuvieran uno dentro del otro. Para evitarlo, simplemente desplazamos la cámara ligeramente hacia afuera.

Renderizando la escena

Si copiaras el código anterior al archivo main.js que creamos antes, no verías nada. Esto se debe a que aún no estamos renderizando nada. Para ello, necesitamos un bucle de renderizado o animación.

```
function animate() {

  renderer.render( scene, camera );

}

renderer.setAnimationLoop( animate );
```

Esto creará un bucle que hace que el renderizador dibuje la escena cada vez que se refresca la pantalla (en una pantalla típica, esto significa 60 veces por segundo). Si eres nuevo en la programación de juegos en el navegador, podrías preguntarte *"¿por qué no creamos un setInterval?".* La cuestión es que podríamos, pero `requestAnimationFrame` que se usa internamente `WebGLRenderer` tiene varias ventajas. Quizás la más importante es que se pausa cuando el usuario navega a otra pestaña del

navegador, evitando así el consumo de batería y la potencia de procesamiento.

Animando el cubo

Si insertas todo el código anterior en el archivo que creaste antes de empezar, deberías ver un recuadro verde. Para hacerlo más interesante, girándolo.

Agregue el siguiente código justo encima de la `renderer.render` llamada en su `animate` función:

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

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

Esto se ejecutará en cada fotograma (normalmente 60 veces por segundo) y le dará al cubo una atractiva animación de rotación. Básicamente, cualquier movimiento o cambio que desees realizar mientras la aplicación se ejecuta debe pasar por el bucle de animación. Por supuesto, puedes llamar a otras funciones desde ahí, para evitar que termines con una `animate` función de cientos de líneas.

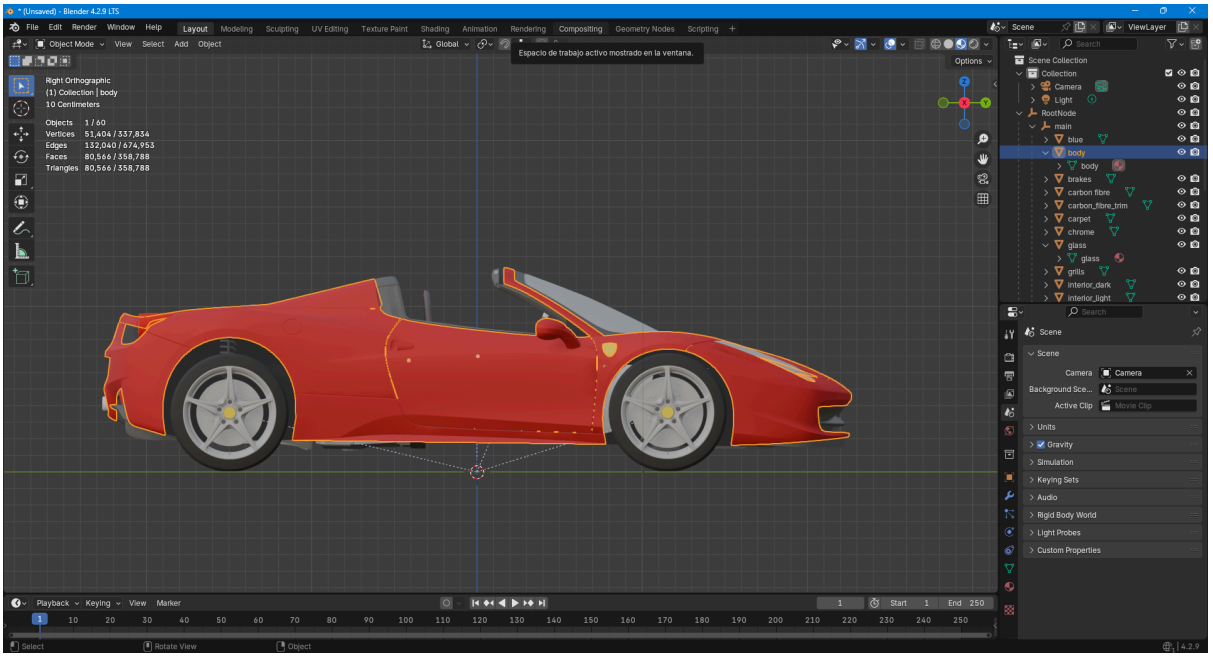
El resultado

¡Felicitaciones! Has completado tu primera aplicación three.js. Es simple, pero hay que empezar por algún lado.

El código completo está disponible a continuación y como ejemplo en vivo editable . Experimente con él para comprender mejor su funcionamiento.

Car Material

información del ferrari



índice.html —

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My first three.js app</title>
    <style>
      body { margin: 0; }
```

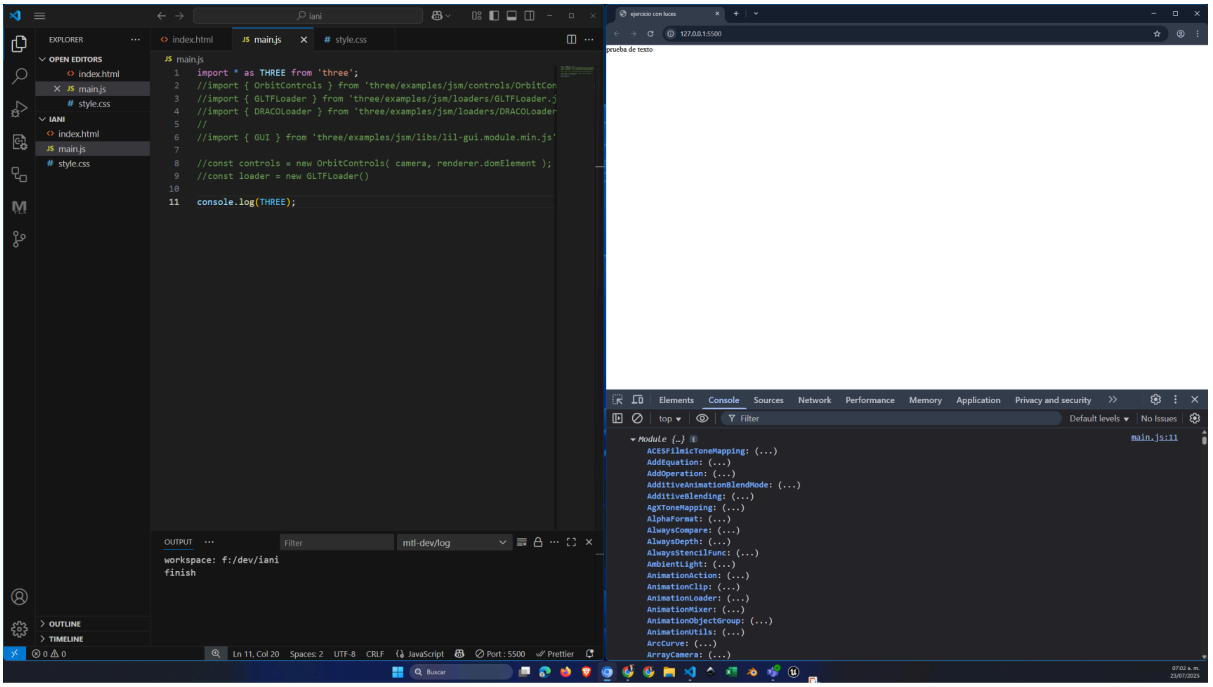
```
    </style>
  </head>
  <body>
    <script type="module" src="/main.js"></script>
  </body>
</html>
```

main.js —

```
import * as THREE from 'three';

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
);

renderer.setAnimationLoop( animate );

document.body.appendChild( renderer.domElement );

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

const material = new THREE.MeshBasicMaterial( { color:
0x00ff00 } );

const cube = new THREE.Mesh( geometry, material );

scene.add( cube );

camera.position.z = 5;

function animate() {

  cube.rotation.x += 0.01;

  cube.rotation.y += 0.01;

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

Ejercicio luces

haciendo lo necesario debemos conseguir esto



Lightmap Baking in Blender for Three.js

https://youtu.be/DpBhfd-LNVk?si=5Uu6Sc65z0gZmx8s