

Introducción Three.js

Inicio

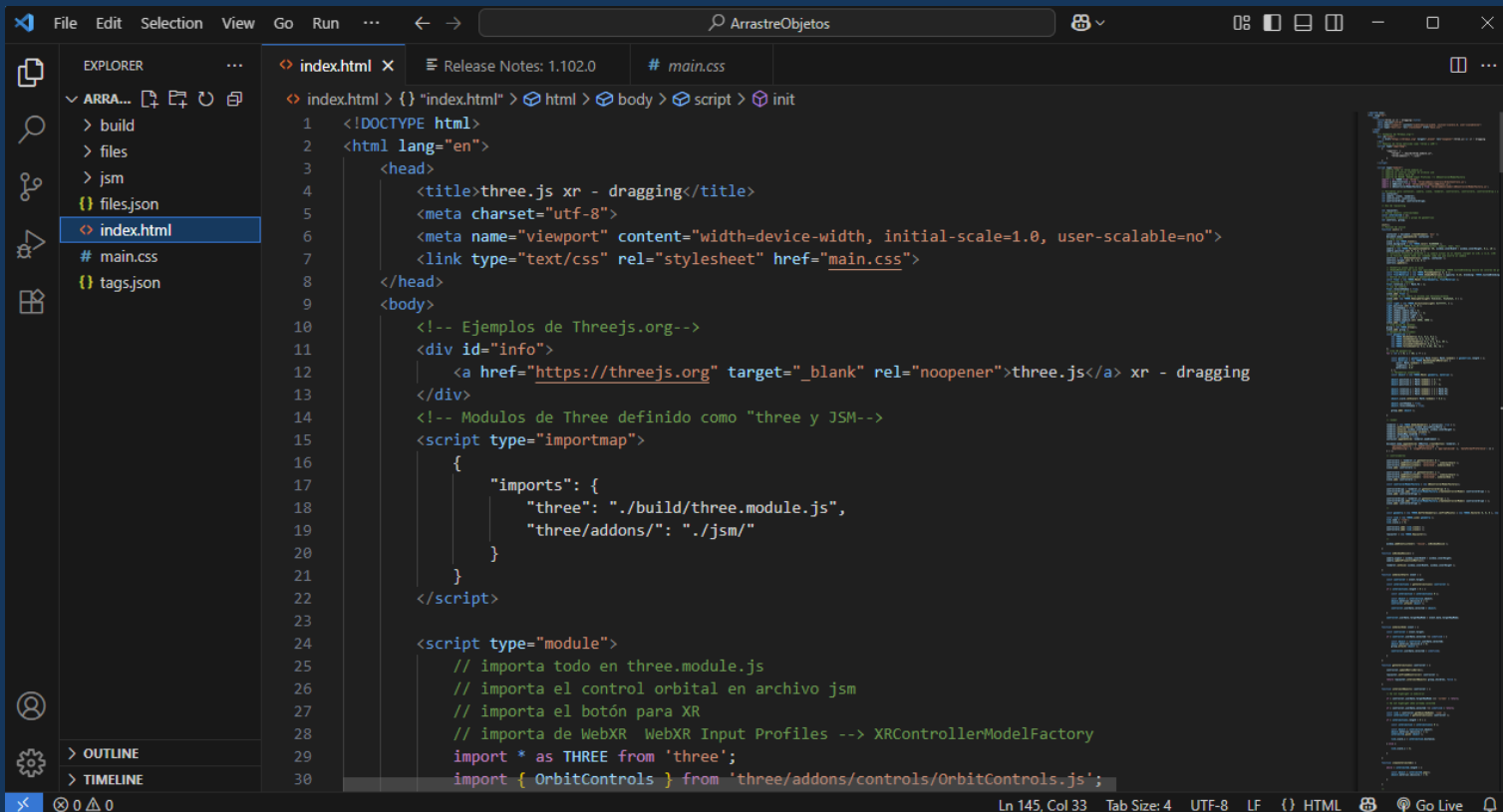
José Luis Carreño Arteaga
jcarreno53@yahoo.com.mx

Distribución local del primer proyecto → https://threejs.org/examples/#webxr_xr_dragging



Nombre	Fecha de modificación	Tipo	Tamaño
build	30/06/2025 06:35 a. m.	Carpeta de archivos	
files	30/06/2025 06:35 a. m.	Carpeta de archivos	
jsm	30/06/2025 06:35 a. m.	Carpeta de archivos	
files.json	30/06/2025 06:35 a. m.	Archivo JSON	16 KB
index	12/07/2025 11:44 a. m.	Chrome HTML Do...	9 KB
main.css	30/06/2025 06:35 a. m.	CSSfile	2 KB
tags.json	30/06/2025 06:35 a. m.	Archivo JSON	9 KB

Con algunos pequeños cambios tenemos el mapa de importaciones



The screenshot shows the Visual Studio Code editor with the 'index.html' file open. The Explorer sidebar on the left shows the project structure with files like 'index.html', 'main.css', and 'tags.json'. The main editor area displays the HTML code, which includes a script tag for an import map (lines 15-22) and a module script (lines 24-30). The import map defines the 'three' module as './build/three.module.js' and the 'three/addons/' folder as './jsm/'. The module script then imports 'THREE' from 'three' and 'OrbitControls' from 'three/addons/controls/OrbitControls.js'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>three.js xr - dragging</title>
5     <meta charset="utf-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
7     <link type="text/css" rel="stylesheet" href="main.css">
8   </head>
9   <body>
10    <!-- Ejemplos de Threejs.org-->
11    <div id="info">
12      <a href="https://threejs.org" target="_blank" rel="noopener">three.js</a> xr - dragging
13    </div>
14    <!-- Modulos de Three definido como "three y JSM-->
15    <script type="importmap">
16      {
17        "imports": {
18          "three": "../build/three.module.js",
19          "three/addons/": "../jsm/"
20        }
21      }
22    </script>
23
24    <script type="module">
25      // importa todo en three.module.js
26      // importa el control orbital en archivo jsm
27      // importa el botón para XR
28      // importa de WebXR WebXR Input Profiles --> XRControllerModelFactory
29      import * as THREE from 'three';
30      import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
```

- Líneas 15-22 Mapa de importación: three en el folder build "three.module.js" y es llamado "three/addons" el folder jsm

Importaciones

```
24 <script type="module">
25   // importa todo en three.module.js
26   // importa el control orbital en archivo jsm
27   // importa el botón para XR
28   // importa de WebXR WebXR Input Profiles --> XRControllerModelFactory
29   import * as THREE from 'three';
30   import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
31   import { XRButton } from 'three/addons/webxr/XRButton.js';
32   import { XRControllerModelFactory } from 'three/addons/webxr/XRControllerModelFactory.js';
33
```

- Líneas 24-33 importamos three.module.js de manera completa.
- Controles orbitales
- Botón para XR
- Se importa WebXR Input Profiles usando XRControllerModelFactory

Variables

```
34 // Variables para container, camera, scene, renderer, controller1, controller2, controllerGrip 1 y 2
35 let container;
36 let camera, scene, renderer;
37 let controller1, controller2;
38 let controllerGrip1, controllerGrip2;
39
40 // Uso de raycasting
41
42 let raycaster;
43 // Guarda objetos intersectados
44 const intersected = [];
45 // Controles orbitales y grupo de geometrias
46 let controls, group;
47
```

☒ Líneas 34 a 47 declaración de variables

☐ Uso de raycaster

Escena, Cámara y Controles

```
54 // Crea escena
55 scene = new THREE.Scene();
56 scene.background = new THREE.Color( 0x808080 );
57 // Crea camara THREE.PerspectiveCamera(fov, aspect, near, far)
58 camera = new THREE.PerspectiveCamera( 50, window.innerWidth / window.innerHeight, 0.1, 10 );
59 camera.position.set( 0, 1.6, 3 );
60 // Controles Orbitales permiten a la camara actuar en el objeto (target en x=0, y =1.6, z=0)
61 // la función update debe ser llamada cada vez que ocurre un cambio
62 controls = new OrbitControls( camera, container );
63 controls.target.set( 0, 1.6, 0 );
64 controls.update();
65
```



Líneas 54 a 65 Escena, Cámara y controles

Piso, Escena y luz

```
101 // Crea 50 geometrías
102 for ( let i = 0; i < 50; i ++ ) {
103
104     const geometry = geometries[ Math.floor( Math.random() * geometries.length ) ];
105     const material = new THREE.MeshStandardMaterial( {
106         color: Math.random() * 0xffffff,
107         roughness: 0.7,
108         metalness: 0.0
109     } );
110     // Parametros aleatorios
111     const object = new THREE.Mesh( geometry, material );
112
113     object.position.x = Math.random() * 4 - 2;
114     object.position.y = Math.random() * 2;
115     object.position.z = Math.random() * 4 - 2;
116
117     object.rotation.x = Math.random() * 2 * Math.PI;
118     object.rotation.y = Math.random() * 2 * Math.PI;
119     object.rotation.z = Math.random() * 2 * Math.PI;
120
121     object.scale.setScalar( Math.random() + 0.5 );
122
123     object.castShadow = true;
124     object.receiveShadow = true;
125
126     group.add( object );
127
128 }
129
```



Líneas 101 a 129 Crea objetos con geometría aleatoria, material aleatorio, posición rotación y escala aleatoria. Luz lejana con incidencia paralela. Y recibe sombras

Renderer

```
130 // render
131
132 renderer = new THREE.WebGLRenderer( { antialias: true } );
133 renderer.setPixelRatio( window.devicePixelRatio );
134 renderer.setSize( window.innerWidth, window.innerHeight );
135 renderer.setAnimationLoop( animate );
136 renderer.shadowMap.enabled = true;
137 renderer.xr.enabled = true;
138 container.appendChild( renderer.domElement );
139
140 document.body.appendChild( XRButton.createButton( renderer, {
141     'optionalFeatures': [ 'depth-sensing' ],
142     'depthSensing': { 'usagePreference': [ 'gpu-optimized' ], 'dataFormatPreference': [] }
143 } ) );
144
```



Líneas 130 a 144 CreaRenderer con XR habilitado

Controladores

```
145 // controladores
146
147 controller1 = renderer.xr.getController( 0 );
148 controller1.addEventListener( 'selectstart', onSelectStart );
149 controller1.addEventListener( 'selectend', onSelectEnd );
150 scene.add( controller1 );
151
152 controller2 = renderer.xr.getController( 1 );
153 controller2.addEventListener( 'selectstart', onSelectStart );
154 controller2.addEventListener( 'selectend', onSelectEnd );
155 scene.add( controller2 );
156
157 const controllerModelFactory = new XRControllerModelFactory();
158
159 controllerGrip1 = renderer.xr.getControllerGrip( 0 );
160 controllerGrip1.add( controllerModelFactory.createControllerModel( controllerGrip1 ) );
161 scene.add( controllerGrip1 );
162
163 controllerGrip2 = renderer.xr.getControllerGrip( 1 );
164 controllerGrip2.add( controllerModelFactory.createControllerModel( controllerGrip2 ) );
165 scene.add( controllerGrip2 );
166
```

- Líneas 145-166
- `xr.getController`: Devuelve un grupo que representa el espacio de rayos objetivo del controlador XR. Úsalo para visualizar objetos 3D que ayudan al usuario en tareas de apuntado, como la interacción con la interfaz de usuario.
- `XRControllerModelFactory`: Crea modelo del controlador
- `GetControllerGrip`: Espacio de sujeción del objeto

Controladores

```
145 // controladores
146
147 controller1 = renderer.xr.getController( 0 );
148 controller1.addEventListener( 'selectstart', onSelectStart );
149 controller1.addEventListener( 'selectend', onSelectEnd );
150 scene.add( controller1 );
151
152 controller2 = renderer.xr.getController( 1 );
153 controller2.addEventListener( 'selectstart', onSelectStart );
154 controller2.addEventListener( 'selectend', onSelectEnd );
155 scene.add( controller2 );
156
157 const controllerModelFactory = new XRControllerModelFactory();
158
159 controllerGrip1 = renderer.xr.getControllerGrip( 0 );
160 controllerGrip1.add( controllerModelFactory.createControllerModel( controllerGrip1 ) );
161 scene.add( controllerGrip1 );
162
163 controllerGrip2 = renderer.xr.getControllerGrip( 1 );
164 controllerGrip2.add( controllerModelFactory.createControllerModel( controllerGrip2 ) );
165 scene.add( controllerGrip2 );
166
```

- Líneas 145-166
- `xr.getController`: Devuelve un grupo que representa el espacio de rayos objetivo del controlador XR. Úsalo para visualizar objetos 3D que ayudan al usuario en tareas de apuntado, como la interacción con la interfaz de usuario.
- `XRControllerModelFactory`: Crea modelo del controlador
- `GetControllerGrip`: Espacio de sujeción del objeto

Linea para para utilizar en los controladores

```
167 //
168
169 const geometry = new THREE.BufferGeometry().setFromPoints( [ new THREE.Vector3( 0, 0, 0 ), new THREE.Vector3( 0, 0, - 1 ) ] );
170
171 const line = new THREE.Line( geometry );
172 line.name = 'line';
173 line.scale.z = 5;
174
175 controller1.add( line.clone() );
176 controller2.add( line.clone() );
177
178 raycaster = new THREE.Raycaster();
179
180 //
181
```

- Líneas 167-181
- Linea apuntadora 169 - 176
- Linea 178 declara raycaster

Cambia de tamaño el canvas

```
179  
180 //  
181  
182 window.addEventListener( 'resize', onWindowResize );  
183  
184 }  
185
```



Referencias Bibliográficas