Universidad de La Laguna

Development of a Web Application based on ThreeJS

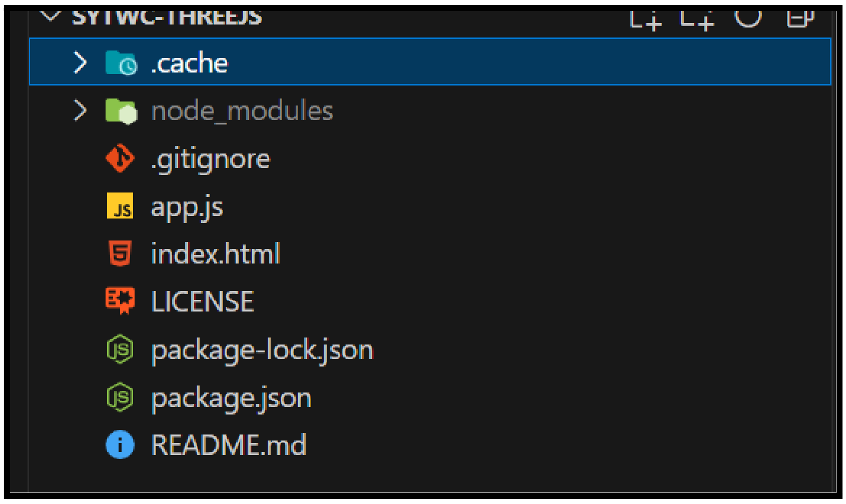
Master's Degree in Computer Engineering - Systems and Web Technologies: Client

Yago Pérez Molanes

In this practice, the technical characteristics and operation of a web application for creating scenes with 3D objects based on the threejs graphics development framework are presented.

Technical features of the application

Firstly, for the development of the practice, a Github repository has been created to host all the code of the application, the structure of the project follows the following scheme:

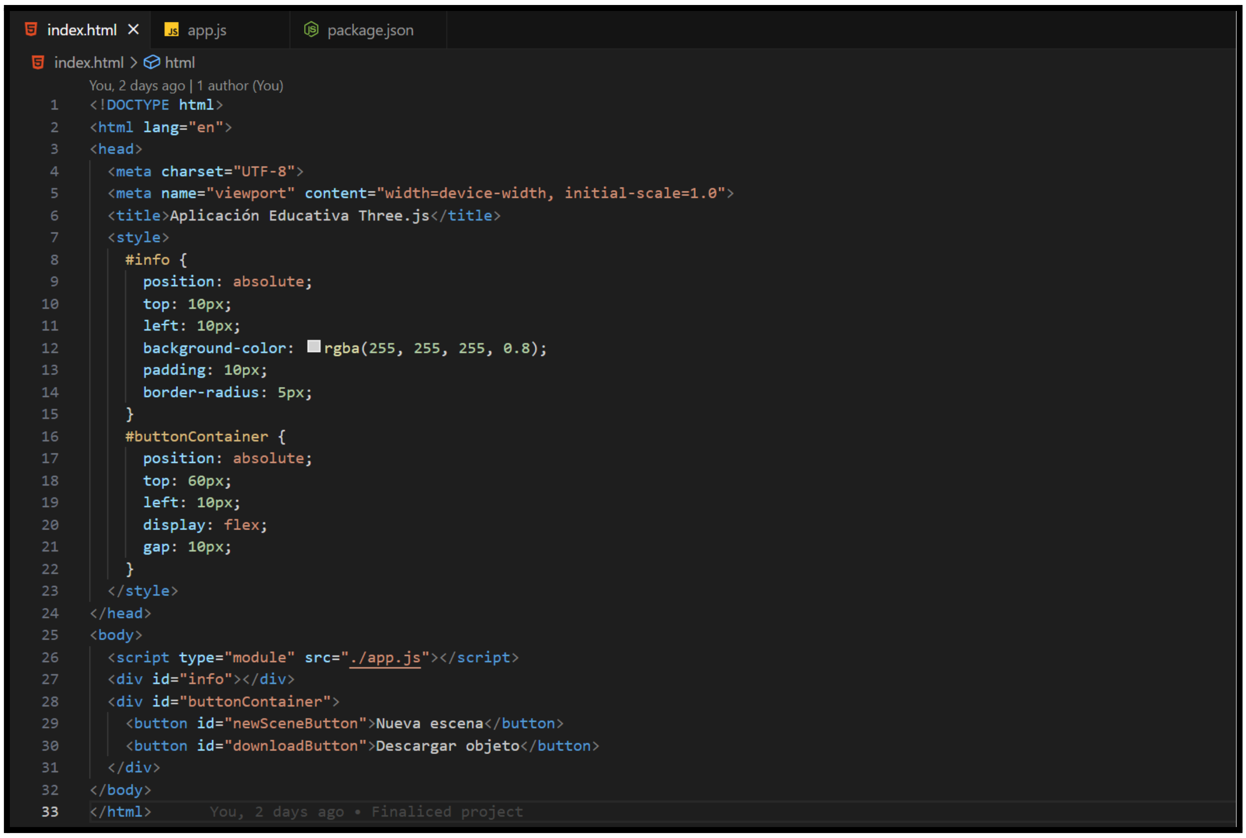


A classic NodeJS project (with its package.json, package-lock.json and node_modules directory) has been created from JavaScript, and other related files from the Github repository (.gitignore, README.md, LICENSE).

Apart from that, we focus on the source code (index.html and app.js).

index.html

This is a simple web page, with the following content:



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Aplicación Educativa Three.js</title>
  <style>
    #info {
      position: absolute;
      top: 10px;
      left: 10px;
      background-color: rgba(255, 255, 255, 0.8);
      padding: 10px;
      border-radius: 5px;
    }
    #buttonContainer {
      position: absolute;
      top: 60px;
      left: 10px;
      display: flex;
      gap: 10px;
    }
  </style>
</head>
<body>
  <script type="module" src="./app.js"></script>
  <div id="info"></div>
  <div id="buttonContainer">
    <button id="newSceneButton">Nueva escena</button>
    <button id="downloadButton">Descargar objeto</button>
  </div>
</body>
</html>     You, 2 days ago • Finaliced project
```

As can be seen, the styles are included in the structure of the web page itself, although this is not a good practice, but it is done because the web is simple.

As we can see, two elements are created, one to show information about the object, and the other is a container for the buttons to be used.

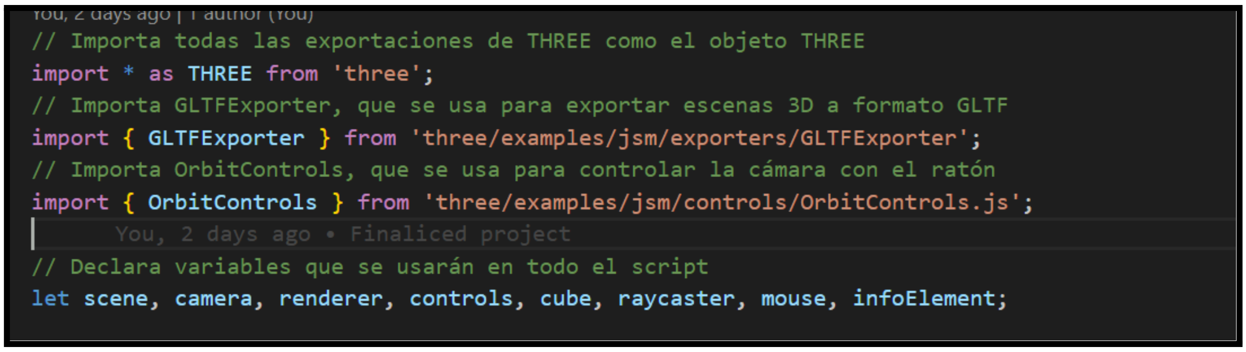
Also, in the body of the html, the javascript module is imported where all our developed functions are hosted, and two divs are created where ids identifiers are created for the elements to be linked to the previous styles, one for the information and the other for the buttons.

In the html a window with the ThreeJS animated scene will be rendered.

app.js

In this file is where all the JavaScript code is hosted, let's go step by step:

First you import ThreeJS, and GLTFExporter and OrbitControls, which are used to export 3D scenes to GLTF format and to control the camera with the mouse, respectively. It also declares a number of variables that will be used throughout the script.

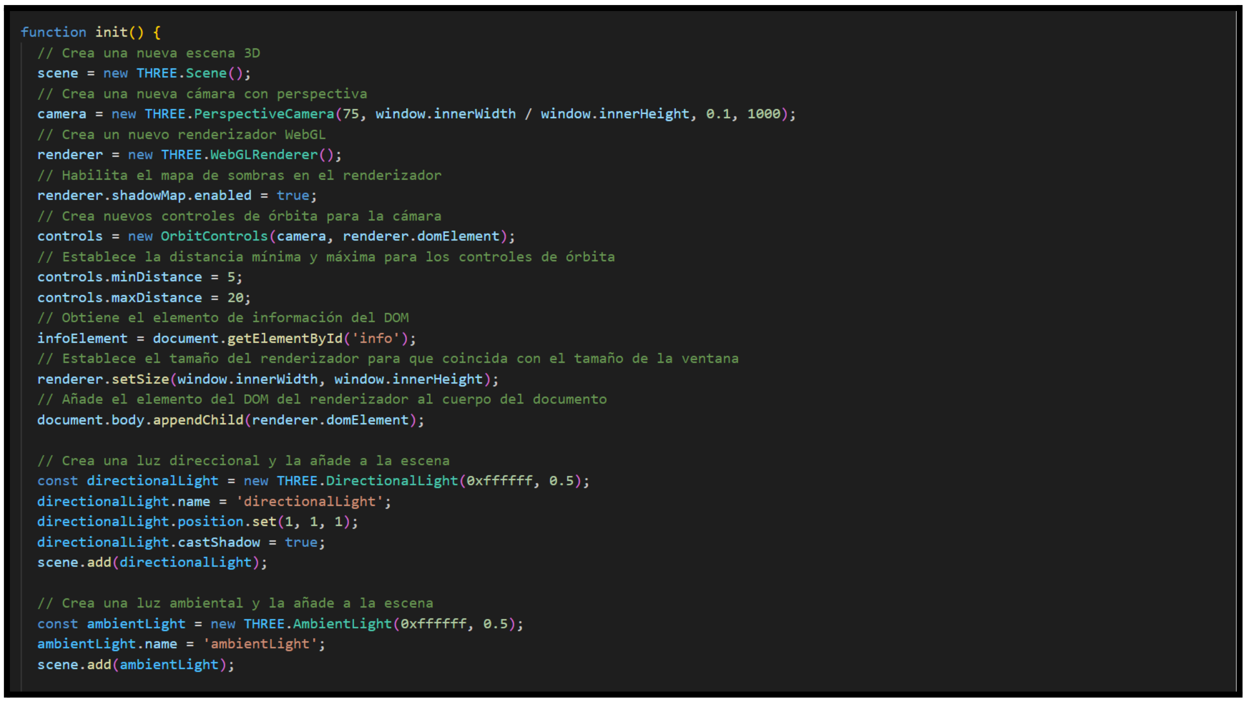


```
You, 2 days ago | 1 author (You)
// Importa todas las exportaciones de THREE como el objeto THREE
import * as THREE from 'three';
// Importa GLTFExporter, que se usa para exportar escenas 3D a formato GLTF
import { GLTFExporter } from 'three/examples/jsm/exporters/GLTFExporter';
// Importa OrbitControls, que se usa para controlar la cámara con el ratón
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js';
      You, 2 days ago • Finaliced project
// Declara variables que se usarán en todo el script
let scene, camera, renderer, controls, cube, raycaster, mouse, infoElement;
```

Following chronologically, we find a listener event that is executed when the document is loaded, which initialises the scene, starts the animation and adds the event handlers for the buttons, to detect when the user wants to request a new scene, or unload the mesh of an object.

We then turn to the function that initialises the scene:

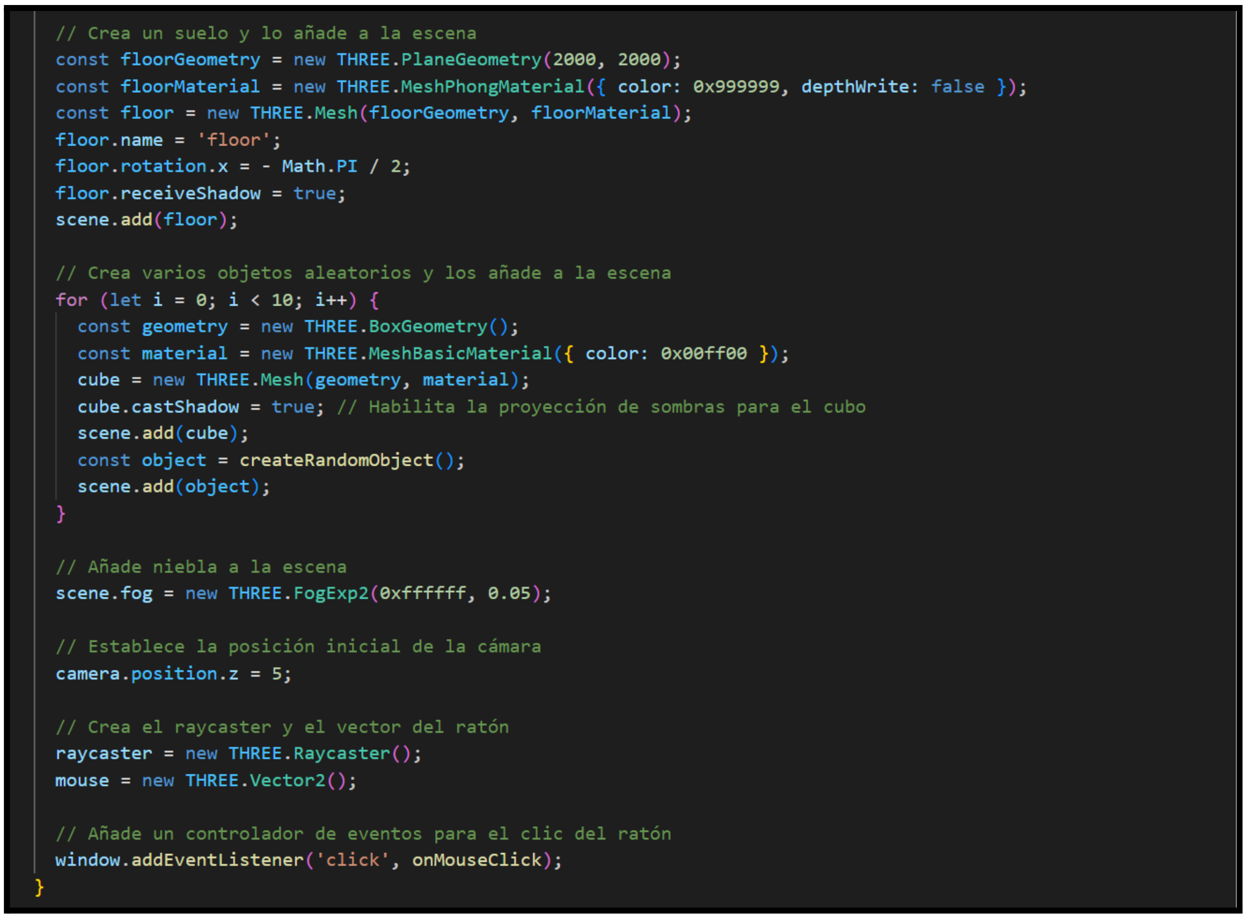
init()



```
function init() {
  // Crea una nueva escena 3D
  scene = new THREE.Scene();
  // Crea una nueva cámara con perspectiva
  camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
  // Crea un nuevo renderizador WebGL
  renderer = new THREE.WebGLRenderer();
  // Habilita el mapa de sombras en el renderizador
  renderer.shadowMap.enabled = true;
  // Crea nuevos controles de órbita para la cámara
  controls = new OrbitControls(camera, renderer.domElement);
  // Establece la distancia mínima y máxima para los controles de órbita
  controls.minDistance = 5;
  controls.maxDistance = 20;
  // Obtiene el elemento de información del DOM
  infoElement = document.getElementById('info');
  // Establece el tamaño del renderizador para que coincida con el tamaño de la ventana
  renderer.setSize(window.innerWidth, window.innerHeight);
  // Añade el elemento del DOM del renderizador al cuerpo del documento
  document.body.appendChild(renderer.domElement);

  // Crea una luz direccional y la añade a la escena
  const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
  directionalLight.name = 'directionalLight';
  directionalLight.position.set(1, 1, 1);
  directionalLight.castShadow = true;
  scene.add(directionalLight);

  // Crea una luz ambiental y la añade a la escena
  const ambientLight = new THREE.AmbientLight(0xffffff, 0.5);
  ambientLight.name = 'ambientLight';
  scene.add(ambientLight);
```

First create a 3D scene, then create a camera with perspective, create a WebGL renderer, enable shadow mapping in the renderer, create new orbit controls for the camera, set the minimum and maximum distance for the orbit controls (you can modify these values if you wish), get the DOM information element, set the renderer size to match the window size, and add the DOM element of the renderer to the document curp.

We then create a directional light and add it to the scene, the same with the ambient light that is added to the scene.

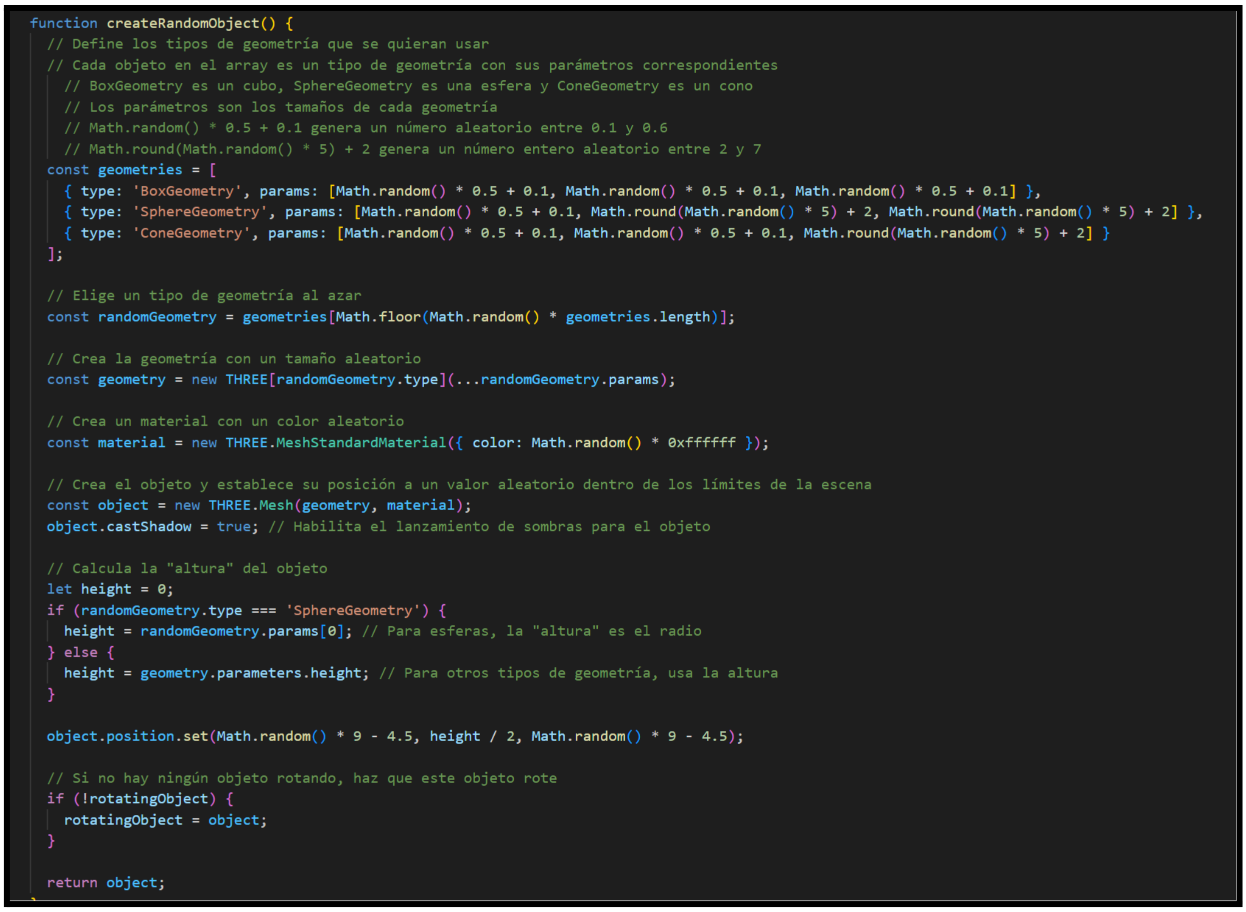


```
  // Crea un suelo y lo añade a la escena
  const floorGeometry = new THREE.PlaneGeometry(2000, 2000);
  const floorMaterial = new THREE.MeshPhongMaterial({ color: 0x999999, depthWrite: false });
  const floor = new THREE.Mesh(floorGeometry, floorMaterial);
  floor.name = 'floor';
  floor.rotation.x = - Math.PI / 2;
  floor.receiveShadow = true;
  scene.add(floor);

  // Crea varios objetos aleatorios y los añade a la escena
  for (let i = 0; i < 10; i++) {
    const geometry = new THREE.BoxGeometry();
    const material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
    cube = new THREE.Mesh(geometry, material);
    cube.castShadow = true; // Habilita la proyección de sombras para el cubo
    scene.add(cube);
    const object = createRandomObject();
    scene.add(object);
  }

  // Añade niebla a la escena
  scene.fog = new THREE.FogExp2(0xffffff, 0.05);

  // Establece la posición inicial de la cámara
  camera.position.z = 5;

  // Crea el raycaster y el vector del ratón
  raycaster = new THREE.Raycaster();
  mouse = new THREE.Vector2();

  // Añade un controlador de eventos para el clic del ratón
  window.addEventListener('click', onMouseClick);
}
```

Now you create a floor and add it to the scene, and in the first instance you create several random objects and add them to the scene (with the help of another function that we will explain later, which is used to create a random object), then you add fog to the scene, set the initial position of the camera, create the raycaster and the mouse vector (this is to detect when you want to click on an object) and then add an event handler to the mouse click.

Let's go with the createrandomobject function:

createrandomObject()



```javascript
function createRandomObject() {
  // Define los tipos de geometría que se quieran usar
  // Cada objeto en el array es un tipo de geometría con sus parámetros correspondientes
    // BoxGeometry es un cubo, SphereGeometry es una esfera y ConeGeometry es un cono
    // Los parámetros son los tamaños de cada geometría
    // Math.random() * 0.5 + 0.1 genera un número aleatorio entre 0.1 y 0.6
    // Math.round(Math.random() * 5) + 2 genera un número entero aleatorio entre 2 y 7
  const geometries = [
    { type: 'BoxGeometry', params: [Math.random() * 0.5 + 0.1, Math.random() * 0.5 + 0.1, Math.random() * 0.5 + 0.1] },
    { type: 'SphereGeometry', params: [Math.random() * 0.5 + 0.1, Math.round(Math.random() * 5) + 2, Math.round(Math.random() * 5) + 2] },
    { type: 'ConeGeometry', params: [Math.random() * 0.5 + 0.1, Math.random() * 0.5 + 0.1, Math.round(Math.random() * 5) + 2] }
  ];

  // Elige un tipo de geometría al azar
  const randomGeometry = geometries[Math.floor(Math.random() * geometries.length)];

  // Crea la geometría con un tamaño aleatorio
  const geometry = new THREE[randomGeometry.type](...randomGeometry.params);

  // Crea un material con un color aleatorio
  const material = new THREE.MeshStandardMaterial({ color: Math.random() * 0xffffff });

  // Crea el objeto y establece su posición a un valor aleatorio dentro de los límites de la escena
  const object = new THREE.Mesh(geometry, material);
  object.castShadow = true; // Habilita el lanzamiento de sombras para el objeto

  // Calcula la "altura" del objeto
  let height = 0;
  if (randomGeometry.type === 'SphereGeometry') {
    height = randomGeometry.params[0]; // Para esferas, la "altura" es el radio
  } else {
    height = geometry.parameters.height; // Para otros tipos de geometría, usa la altura
  }

  object.position.set(Math.random() * 9 - 4.5, height / 2, Math.random() * 9 - 4.5);

  // Si no hay ningún objeto rotando, haz que este objeto rote
  if (!rotatingObject) {
    rotatingObject = object;
  }

  return object;
```

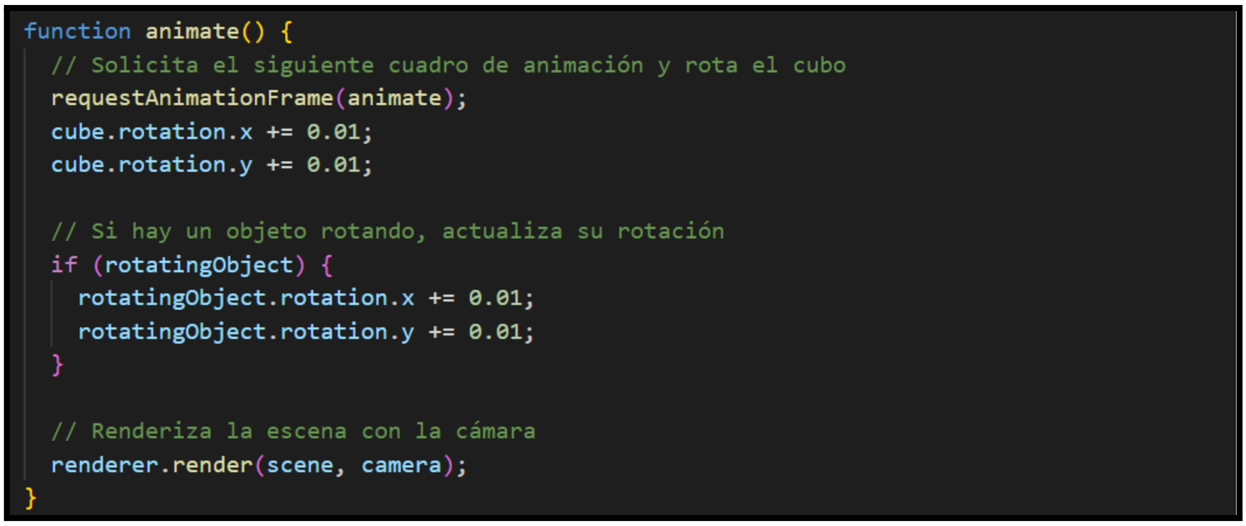
Its purpose is to generate a random object, in the init function a loop is used to generate several elements.

As you can see, you define the types of geometry you want to use, which, in this case, are going to be cubes, spheres and cones, with their random parameters, then you choose a type of geometry at random, create the geometry with a random size, and the material with a random colour, then proceed to create the object and set its position to a random value within the limits of the scene.

Then shadow casting is enabled for the object, the height for the object is calculated and one of the objects is randomly rotated.

After initialising the scene and generating random objects we move on to the animate function, which is executed next:

animate()



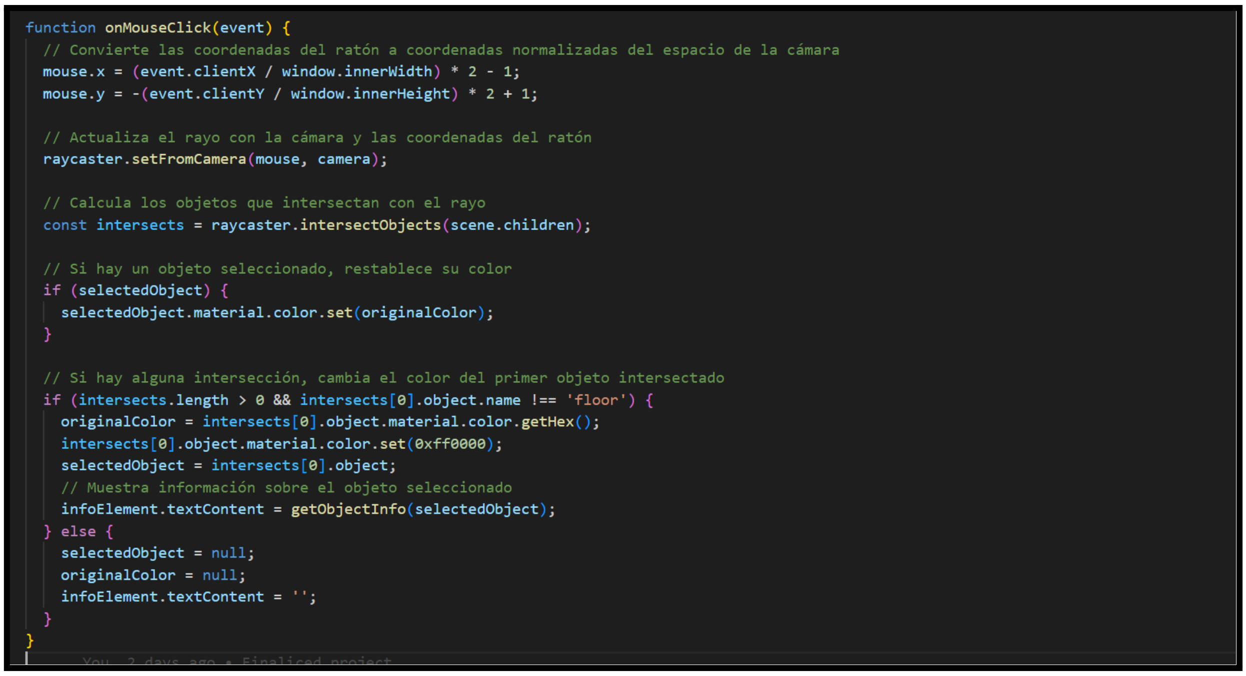
```javascript
function animate() {
  // Solicita el siguiente cuadro de animación y rota el cubo
  requestAnimationFrame(animate);
  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;

  // Si hay un objeto rotando, actualiza su rotación
  if (rotatingObject) {
    rotatingObject.rotation.x += 0.01;
    rotatingObject.rotation.y += 0.01;
  }

  // Renderiza la escena con la cámara
  renderer.render(scene, camera);
}
```

The purpose of this function is to generate new frames, that is to say, to animate the scene so that the rotations of the objects are produced. It does this by requesting the next animation frame, rotating a cube that we had created at the beginning, and rotating the other object that also had an animation, and finally rendering the scene with the camera.

Now if we remember in the init function we had added an event handler to control the user's click on the scene, so when a user clicks on an object the event is fired and the onmouseclick function is executed, which can be seen in the screenshot on the next page and it does the following:

onmouseClick(event)

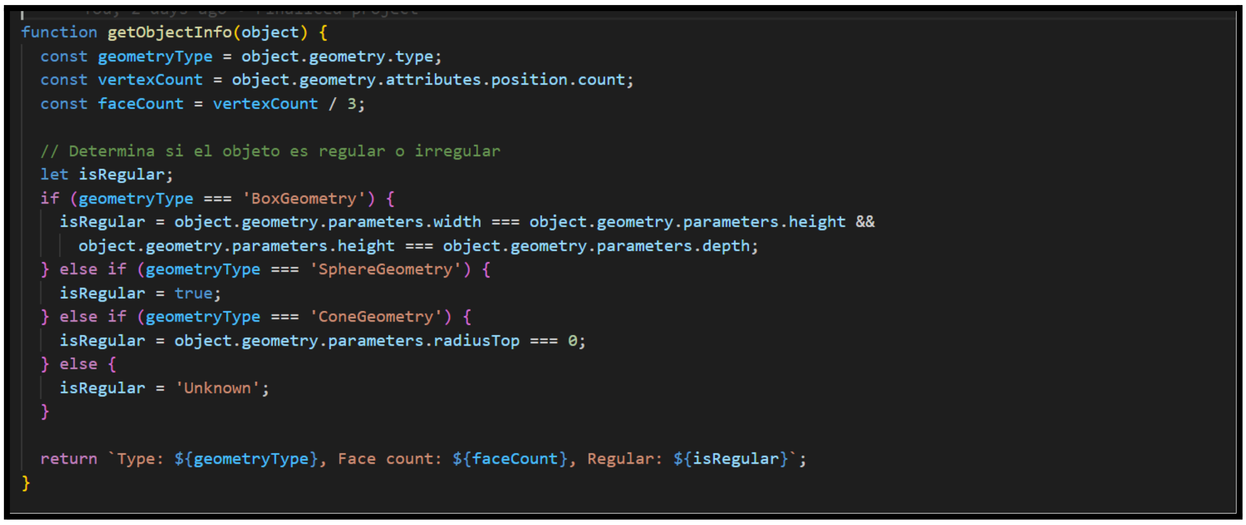
It converts the mouse coordinates to normalised coordinates in camera space, updates the ray with the camera and mouse coordinates, calculates the objects intersecting the ray (in case there is a selected object it resets its colour) and in case there is an intersection it changes the colour of the first intersected object, it also displays information about the selected object by accessing the html element (to display information it makes use of another auxiliary function).



```
function onMouseClick(event) {
  // Convierte las coordenadas del ratón a coordenadas normalizadas del espacio de la cámara
  mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

  // Actualiza el rayo con la cámara y las coordenadas del ratón
  raycaster.setFromCamera(mouse, camera);

  // Calcula los objetos que intersectan con el rayo
  const intersects = raycaster.intersectObjects(scene.children);

  // Si hay un objeto seleccionado, restablece su color
  if (selectedObject) {
    selectedObject.material.color.set(originalColor);
  }

  // Si hay alguna intersección, cambia el color del primer objeto intersectado
  if (intersects.length > 0 && intersects[0].object.name !== 'floor') {
    originalColor = intersects[0].object.material.color.getHex();
    intersects[0].object.material.color.set(0xff0000);
    selectedObject = intersects[0].object;
    // Muestra información sobre el objeto seleccionado
    infoElement.textContent = getObjectInfo(selectedObject);
  } else {
    selectedObject = null;
    originalColor = null;
    infoElement.textContent = '';
  }
}
      You, 2 days ago • Finalized project
```

The auxiliary function we were discussing is as follows:

getObjectInfo(object)



```
function getObjectInfo(object) {
  const geometryType = object.geometry.type;
  const vertexCount = object.geometry.attributes.position.count;
  const faceCount = vertexCount / 3;

  // Determina si el objeto es regular o irregular
  let isRegular;
  if (geometryType === 'BoxGeometry') {
    isRegular = object.geometry.parameters.width === object.geometry.parameters.height &&
      object.geometry.parameters.height === object.geometry.parameters.depth;
  } else if (geometryType === 'SphereGeometry') {
    isRegular = true;
  } else if (geometryType === 'ConeGeometry') {
    isRegular = object.geometry.parameters.radiusTop === 0;
  } else {
    isRegular = 'Unknown';
  }

  return `Type: ${geometryType}, Face count: ${faceCount}, Regular: ${isRegular}`;
}
```

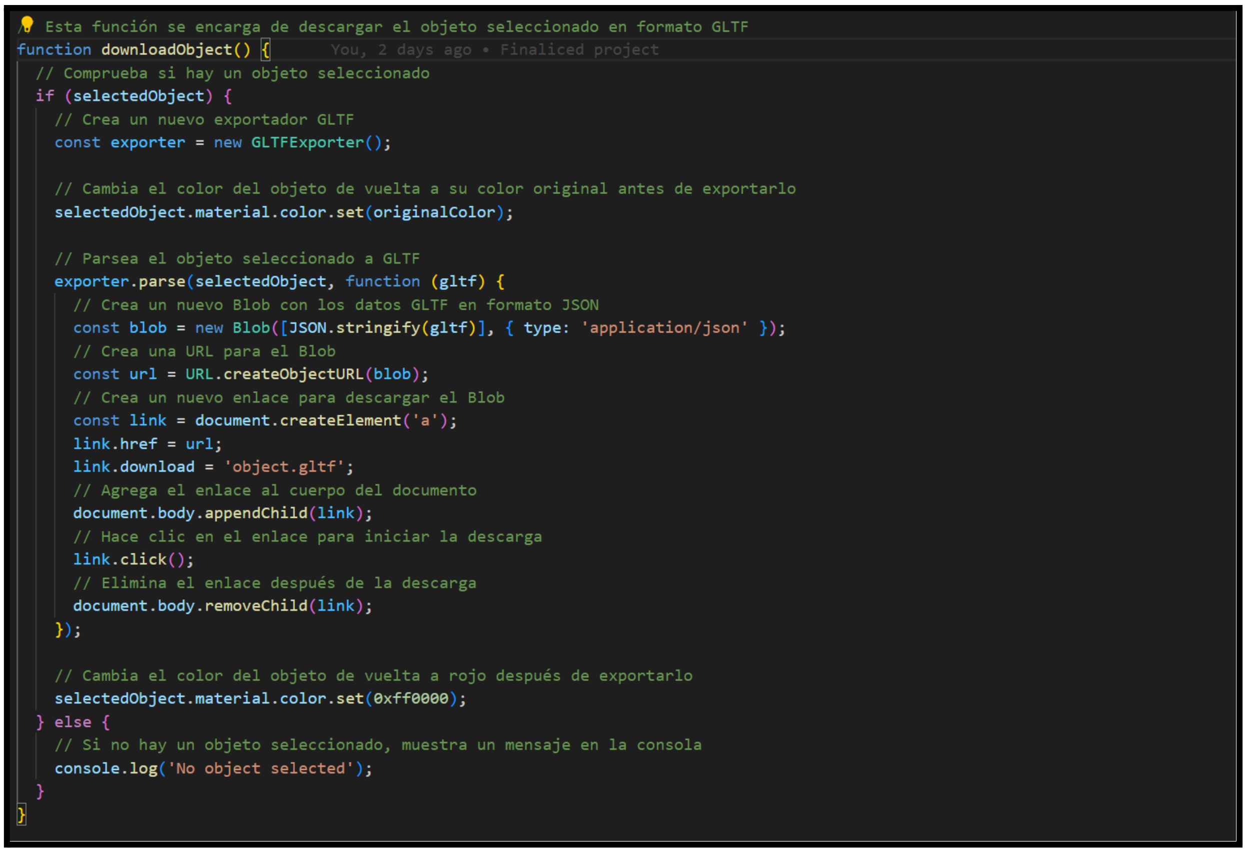
Specifically, the information returned consists of the type of geometry of the object, the number of faces of the object and whether it is a regular or irregular object.

When the user clicks on an object it is considered selected, and to show it to the user, its colour is changed to red and information about it is displayed.

Now let's go to the functions that are executed once the "Download Object" and "New Scene" buttons are clicked, (since two listen events have been added to these buttons when the document is loaded).

The downloadObject function is responsible for downloading the selected object in GLTF format.

downloadObject()



```
// Esta función se encarga de descargar el objeto seleccionado en formato GLTF
function downloadObject() {        You, 2 days ago • Finaliced project
  // Comprueba si hay un objeto seleccionado
  if (selectedObject) {
    // Crea un nuevo exportador GLTF
    const exporter = new GLTFExporter();

    // Cambia el color del objeto de vuelta a su color original antes de exportarlo
    selectedObject.material.color.set(originalColor);

    // Parsea el objeto seleccionado a GLTF
    exporter.parse(selectedObject, function (gltf) {
      // Crea un nuevo Blob con los datos GLTF en formato JSON
      const blob = new Blob([JSON.stringify(gltf)], { type: 'application/json' });
      // Crea una URL para el Blob
      const url = URL.createObjectURL(blob);
      // Crea un nuevo enlace para descargar el Blob
      const link = document.createElement('a');
      link.href = url;
      link.download = 'object.gltf';
      // Agrega el enlace al cuerpo del documento
      document.body.appendChild(link);
      // Hace clic en el enlace para iniciar la descarga
      link.click();
      // Elimina el enlace después de la descarga
      document.body.removeChild(link);
    });

    // Cambia el color del objeto de vuelta a rojo después de exportarlo
    selectedObject.material.color.set(0xff0000);
  } else {
    // Si no hay un objeto seleccionado, muestra un mensaje en la consola
    console.log('No object selected');
  }
}
```

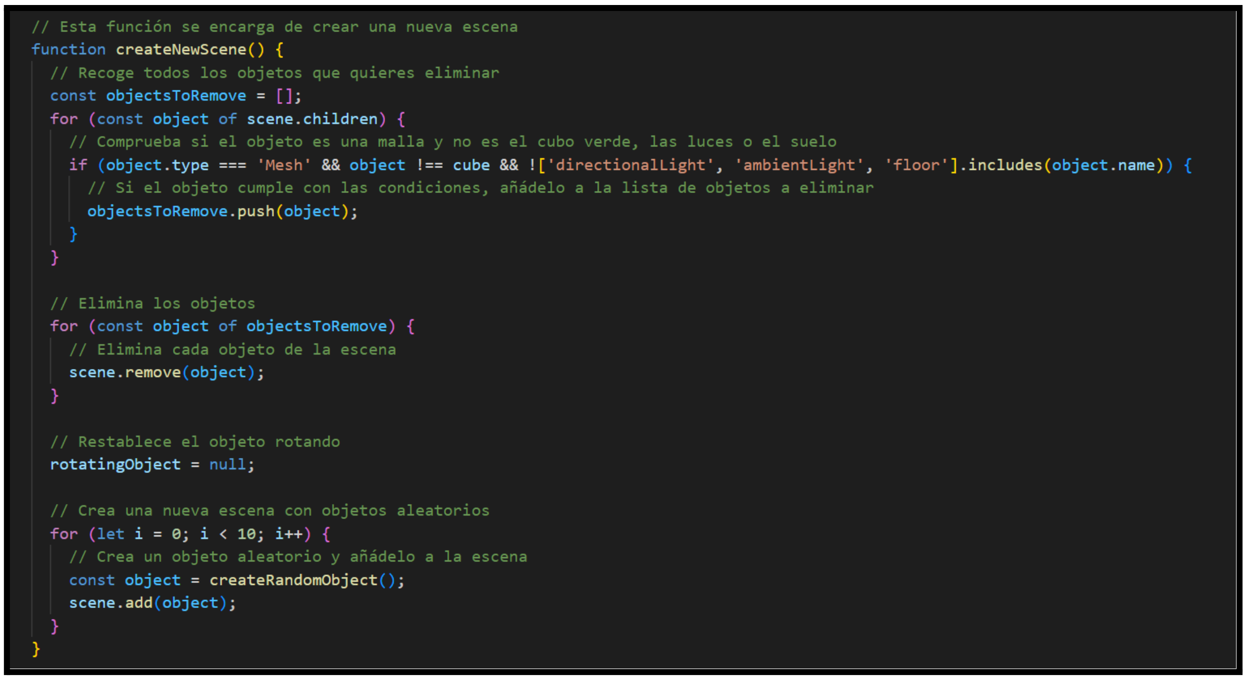
First create a new GLTF exporter, change the colour of the object back to its original colour before exporting, and then parse the selected object to GLTF.

When parsed, it creates a new Blob with the GLTF data in JSON format, creates a URL for the Blob, creates a new link to download the Blob, adds the link to the body of the document, clicks on the link to start the download, and deletes the link after the download.

Then change the colour of the object back to red after export.

We now move on to the last function, createNewScene, which is responsible for creating a new scene:

createNewScene()



```javascript
// Esta función se encarga de crear una nueva escena
function createNewScene() {
  // Recoge todos los objetos que quieres eliminar
  const objectsToRemove = [];
  for (const object of scene.children) {
    // Comprueba si el objeto es una malla y no es el cubo verde, las luces o el suelo
    if (object.type === 'Mesh' && object !== cube && !['directionalLight', 'ambientLight', 'floor'].includes(object.name)) {
      // Si el objeto cumple con las condiciones, añádelo a la lista de objetos a eliminar
      objectsToRemove.push(object);
    }
  }

  // Elimina los objetos
  for (const object of objectsToRemove) {
    // Elimina cada objeto de la escena
    scene.remove(object);
  }

  // Restablece el objeto rotando
  rotatingObject = null;

  // Crea una nueva escena con objetos aleatorios
  for (let i = 0; i < 10; i++) {
    // Crea un objeto aleatorio y añádelo a la escena
    const object = createRandomObject();
    scene.add(object);
  }
}
```

What it does is to collect the objects you want to remove, check if the object is a mesh and is not the green cube, (you want to keep the cube created at the beginning) nor the lights or the floor, you just want to remove the objects.

It then proceeds to delete the objects, removes each object from the scene, restores the rotating object, and creates a new scene with random objects, i.e. within a loop it creates random objects and adds them to the scene.

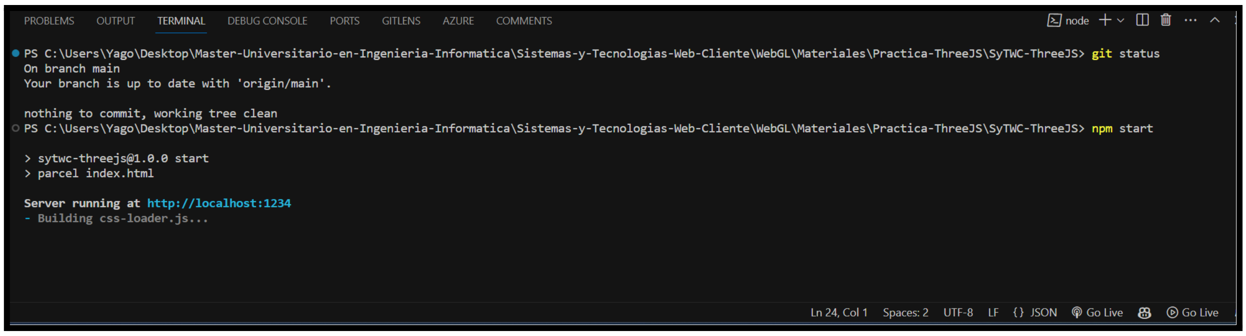
Application operation

The application code can be found in the following repository:

- https://github.com/alu0101254678/SyTWC-ThreeJS

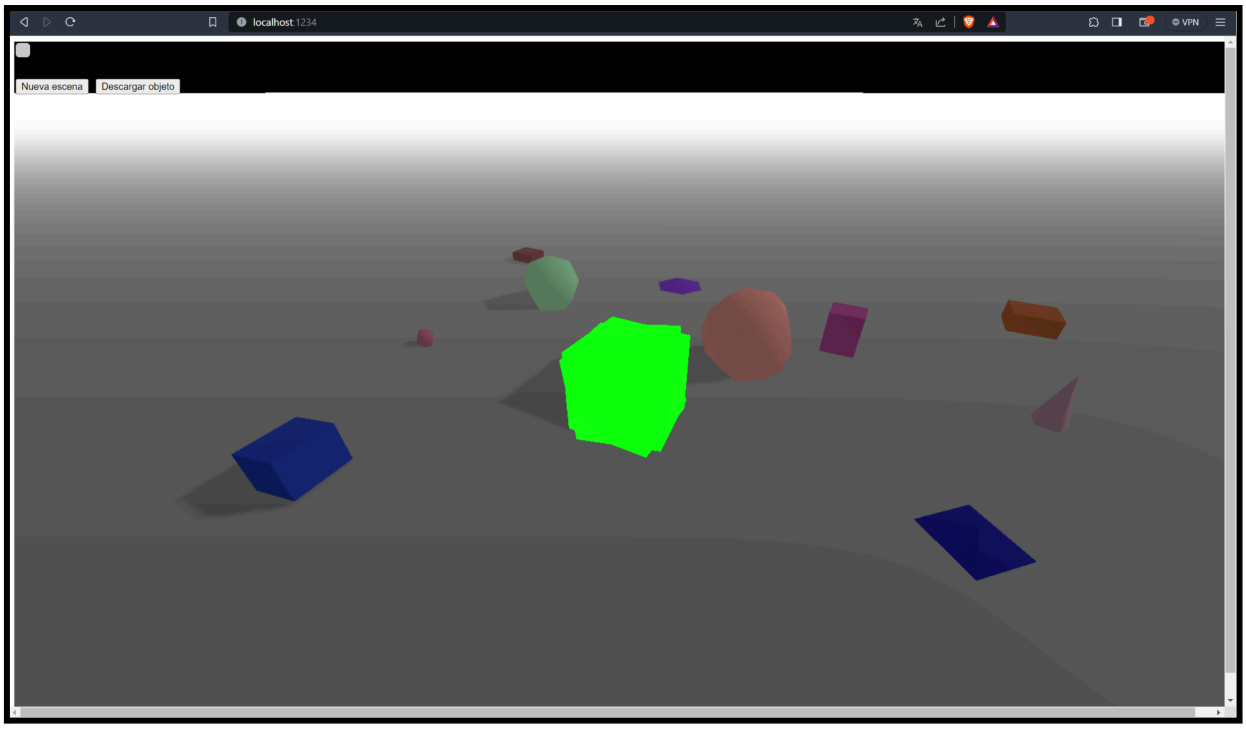
We have two ways to test the application, one of them would be locally, for this, we clone the repository on the machine and install the dependencies with npm install, and then with the command npm start we use parcel to serve the static files (parcel index.html).

Then we would see something like this:



```
PS C:\Users\Yago\Desktop\Master-Universitario-en-Ingenieria-Informatica\Sistemas-y-Tecnologias-Web-Cliente\WebGL\Materiales\Practica-ThreeJS\SyTWC-ThreeJS> git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
PS C:\Users\Yago\Desktop\Master-Universitario-en-Ingenieria-Informatica\Sistemas-y-Tecnologias-Web-Cliente\WebGL\Materiales\Practica-ThreeJS\SyTWC-ThreeJS> npm start

> sytwc-threejs@1.0.0 start
> parcel index.html

Server running at http://localhost:1234
- Building css-loader.js...
```

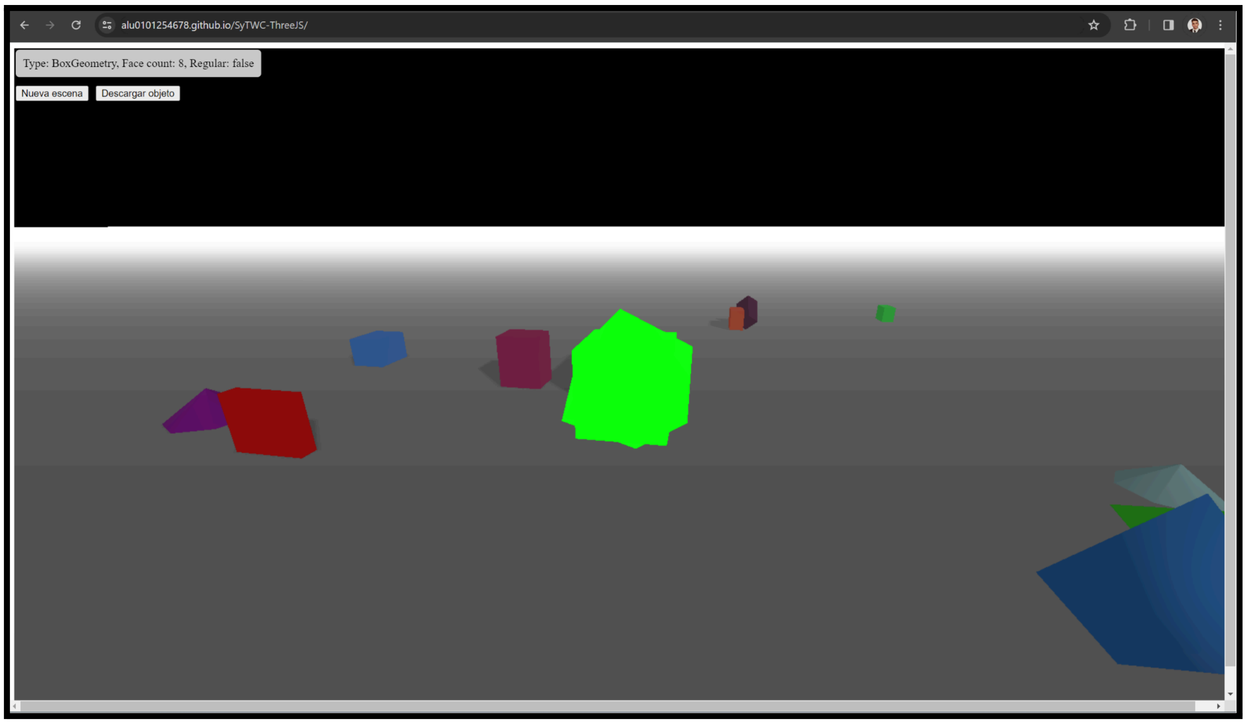
If we access the address http://localhost:1234 in a browser, the following screen will appear:



The application is also deployed on Github Pages, at the following link:

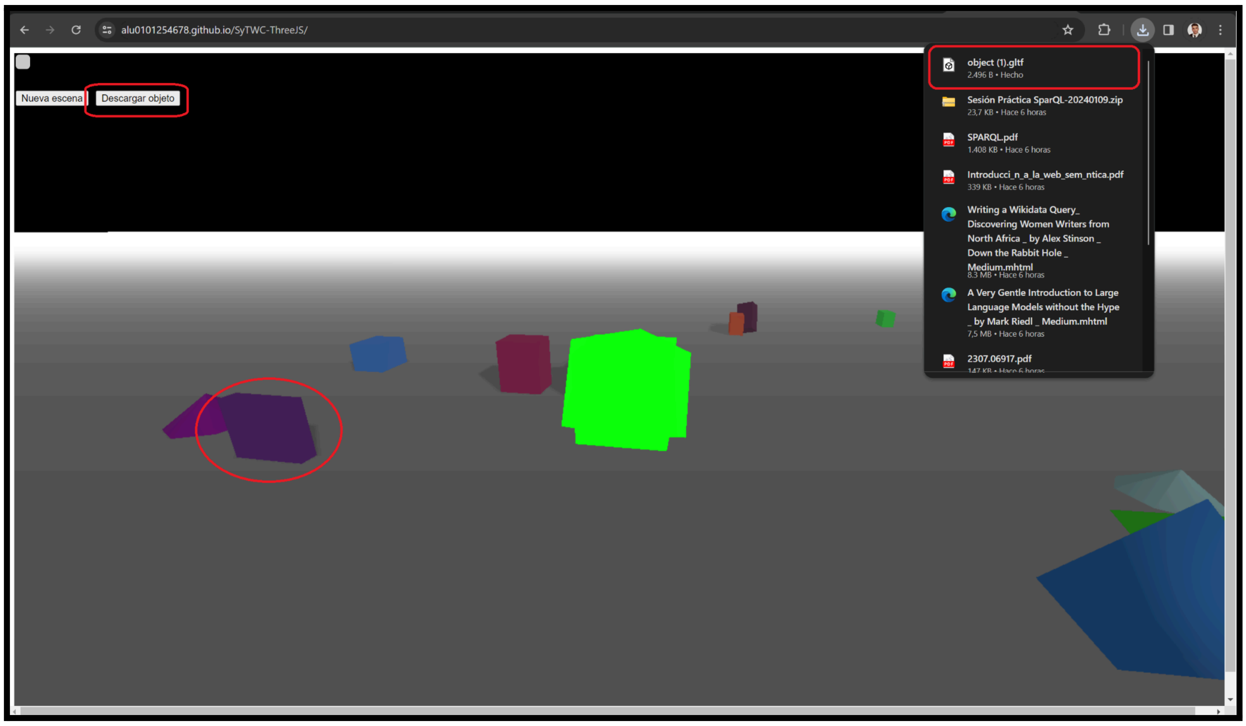
- https://alu0101254678.github.io/SyTWC-ThreeJS/

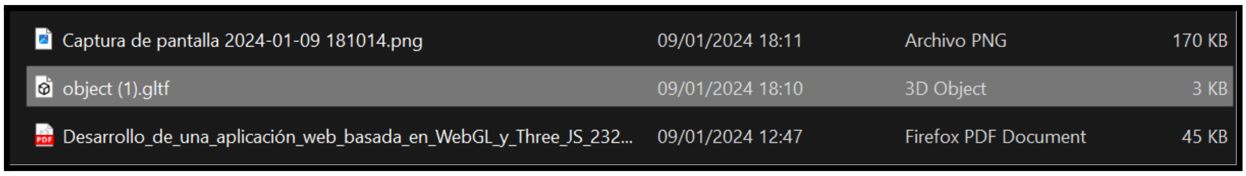
In both cases the home screen of the page should be the same, with the mouse we can move the camera, and, if we click on an object:


alu0101254678.github.io/SyTWC-ThreeJS/
Type: BoxGeometry, Face count: 8, Regular: false
Nueva escena
Descargar objeto

We see that in the top left hand corner there is an information box showing the type of object, the number of faces and whether it is regular or irregular.

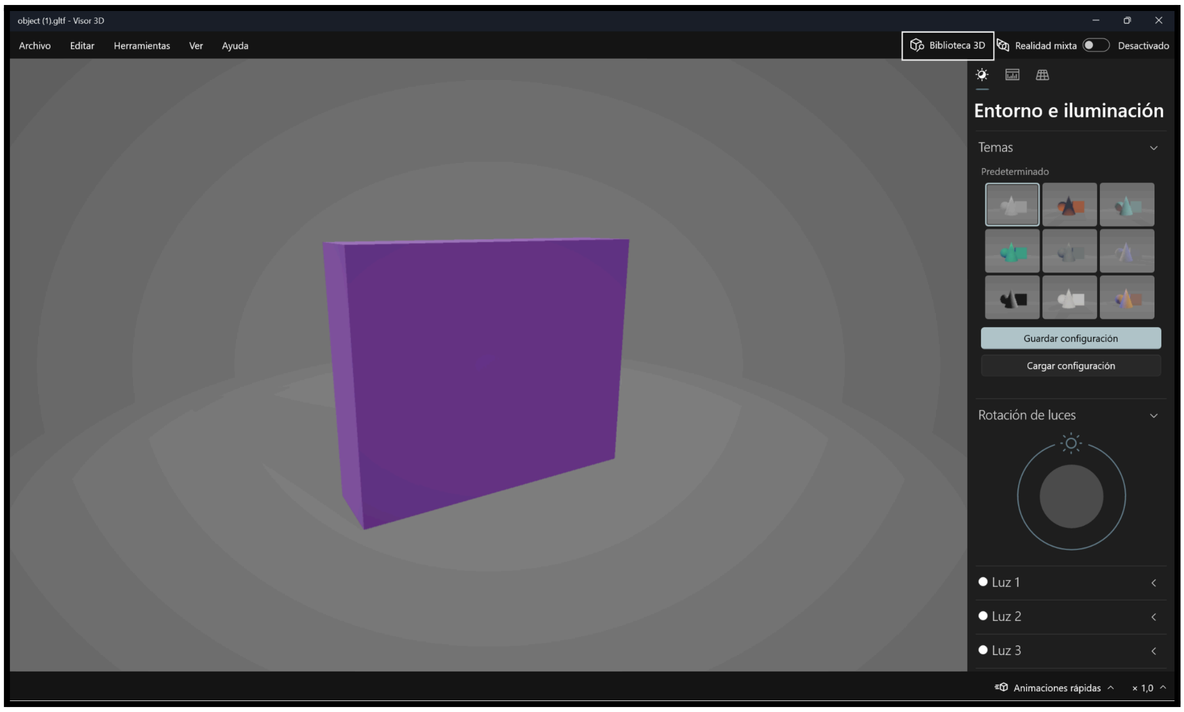
In addition, it is also possible to download the mesh of the object by clicking on the Download Object button:


alu0101254678.github.io/SyTWC-ThreeJS/
Nueva escena
Descargar objeto
object (1).gltf
2.496 B • Hecho
Sesión Práctica SparQL-20240109.zip
23,7 KB • Hace 6 horas
SPARQL.pdf
1.408 KB • Hace 6 horas
Introducci_n_a_la_web_sem_ntica.pdf
339 KB • Hace 6 horas
Writing a Wikidata Query_ Discovering Women Writers from North Africa _ by Alex Stinson _ Down the Rabbit Hole _ Medium.mhtml
8,3 MB • Hace 6 horas
A Very Gentle Introduction to Large Language Models without the Hype _ by Mark Riedl _ Medium.mhtml
7,5 MB • Hace 6 horas
2307.06917.pdf
147 KB • Hace 6 horas

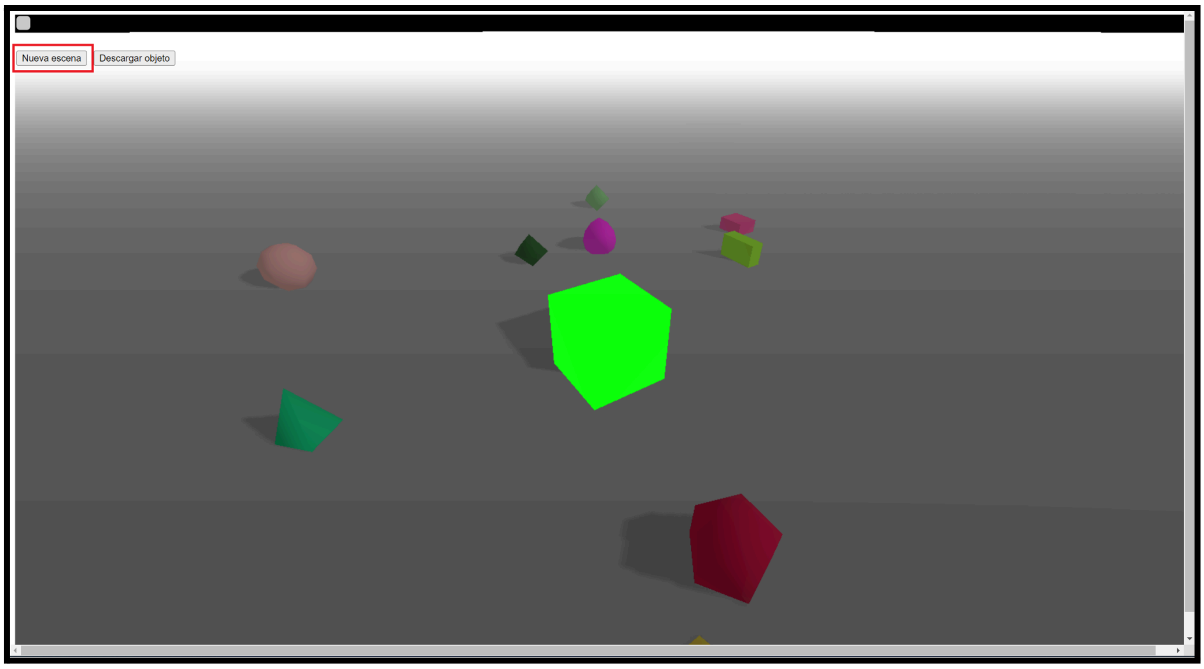


Captura de pantalla 2024-01-09 181014.png	09/01/2024 18:11	Archivo PNG	170 KB
object (1).gltf	09/01/2024 18:10	3D Object	3 KB
Desarrollo_de_una_aplicación_web_basada_en_WebGL_y_Three_JS_232...	09/01/2024 12:47	Firefox PDF Document	45 KB

It is possible to open the downloaded file:



Finally, it is also possible to create a new scene by clicking on the New Scene button:



This effectively creates a new scene, as you can see the objects are illuminated and have shadows, there is a floor and the objects are placed on top of it.

Bibliography

- JavaScript 3D Library ThreeJS
- ThreeJS's repository