

Instituto Tecnológico Superior de Tierra Blanca

Semestre Agosto 2021 Enero 2022

Ingeniería en Sistemas Computacionales

ISIC 2010-224

SCC-1010 Graficación

**Unidad III y IV Graficación 3D, Relleno, iluminación
y sombreado.**

**Reporte de practica | Insertando personajes al
escenario**

Blanca Ramírez Cruz 198N0530

Sergio Jared Valencia Cortaza 198N0068

Tierra Blanca Veracruz



Introducción

Los modelos 3D sin duda han tenido un gran impacto en la sociedad. Tanto así que el conocimiento para la creación de estos modelos se ha vuelto en la actualidad un requerimiento para diseñadores gráficos, programadores, etc.

El diseño de los modelos 3D además de permitirnos crear objetos, también ir más allá, llevando nuestra imaginación y creatividad a otro nivel completamente diferente y avanzado.

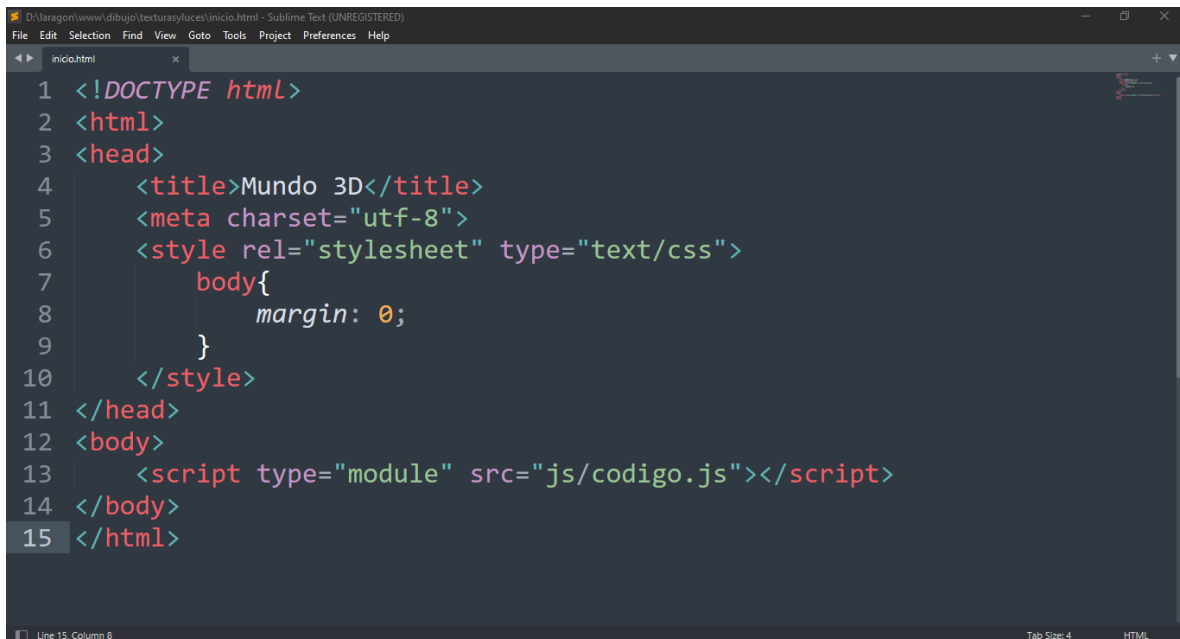
Actualmente gracias a los avances en la tecnología podemos incluso importar y exportar dichos modelos para ser usados externamente o compartirlos a donde deseemos.

Desarrollo

Para poder realizar la carga de un modelo a Three.js lo que emplearemos en esta ocasión es el método de importación GLTFLoader, el cual nos ayudará a cargar un modelo gltf para posteriormente poder visualizarlo. Método que se localiza en la ruta `“./three.js-master/examples/jsm/loaders/GLTFLoader.js”`.

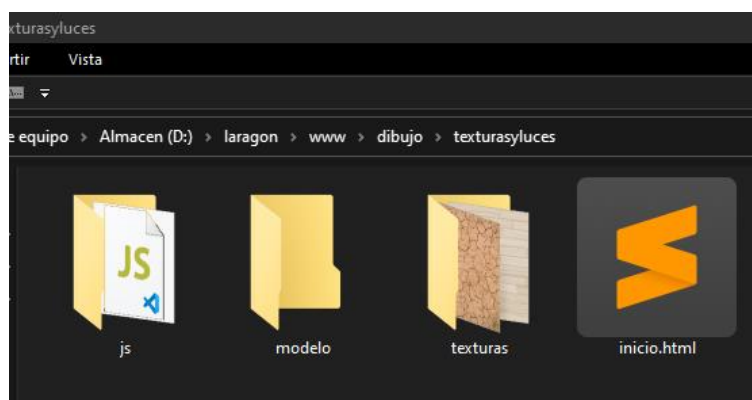
Creación del proyecto

Uno de los principales requisitos es generar la página web de HTML en donde se hace referencia para cargar el código JS.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Mundo 3D</title>
5   <meta charset="utf-8">
6   <style rel="stylesheet" type="text/css">
7     body{
8       margin: 0;
9     }
10  </style>
11 </head>
12 <body>
13   <script type="module" src="js/codigo.js"></script>
14 </body>
15 </html>
```

Como ya sabemos al lado de la página en la misma ruta de carpeta deben existir distintas carpetas para guardar los códigos, las texturas y los modelos tridimensionales.



Ahora el proyecto luciría mas o menos de esta forma en el sentido de que usaremos carpetas para modelos tridimensionales y texturas aparte de las posibles imágenes y archivos de la página.

```
File Edit Selection View Go Run Terminal Help
* codigo.js - Visual Studio Code

D:\> laraion > www > dibujo > texturasy luces > js > JS codigo.js > Main > initAritubutos

1
2 import * as THREE from './three.js-master/build/three.module.js'
3 //import { GLTFExporter } from './three.js-master/examples/jsm/exporters/GLTFExporter'
4 //import { GLTFLoader } from './three.js-master/examples/jsm/loaders/GLTFLoader.js'
5 import { OrbitControls } from './three.js-master/examples/jsm/controls/OrbitControls'
6
```

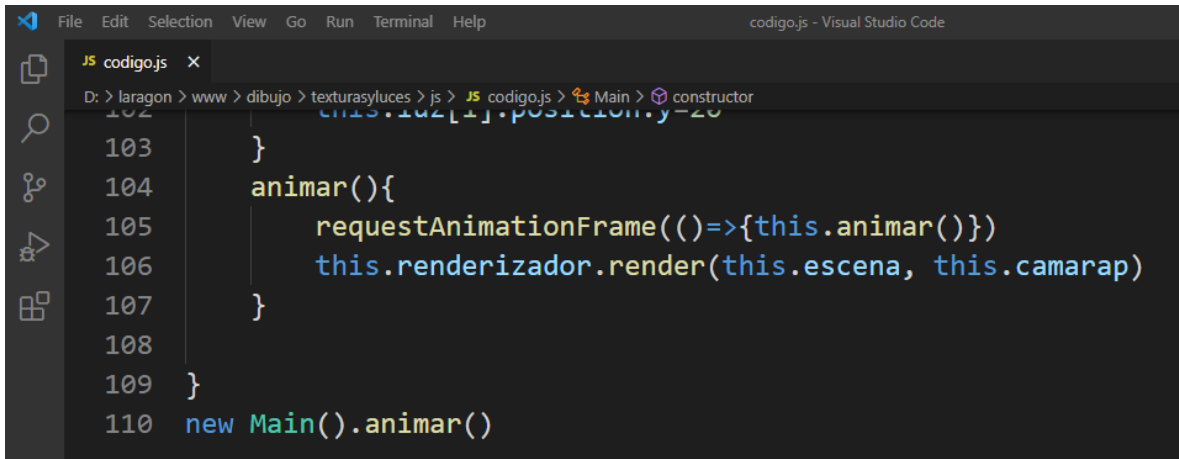
Nuestra página hace referencia solo a nuestro código, pero el utiliza en varias ocasiones las clases de Three.js y como la página no hace referencia a él en la ejecución se verá un error en la consola del navegador diciendo que no está definido, lo que se soluciona con una importación.

```
6
7 class Main {
8   constructor(){
9     this.initAritubutos()
10    this.initMateriales()
11    this.initEscena()
12    this.initCamara()
13    this.initObjetos()
14  }
15  initMateriales() {
```

Como antes, utilizaremos POO para dividir el código en distintas funciones, una para todas las variables, para los materiales que incluyen: geometrías, mallas de polígonos, texturas, bastantes MeshBasicMaterial y el modelo 3D.

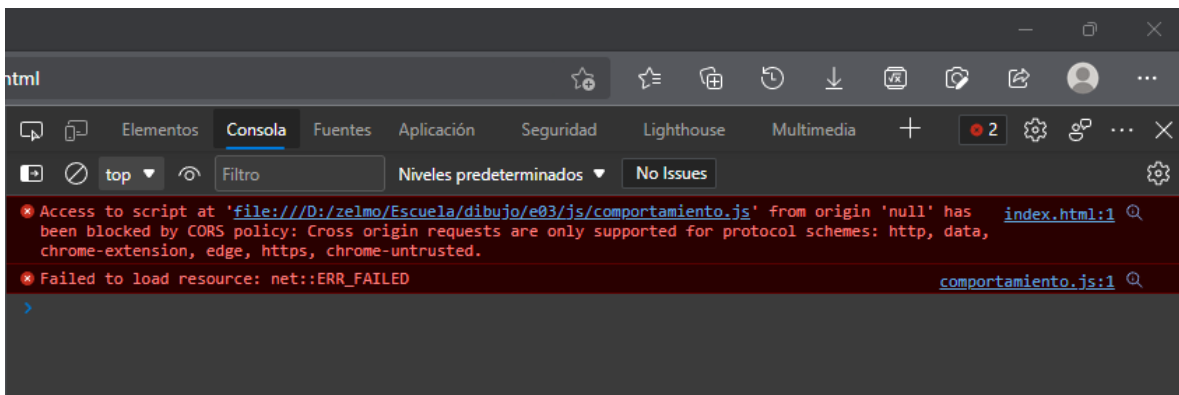
```
69
70 initEscena(){
71   this.escena=new THREE.Scene()
72 }
73 initCamara(){
74   this.camarap=new THREE.PerspectiveCamera(
75     100, this.aspecto, 0.1, 1000
76   );
77   this.camarao=new THREE.OrthographicCamera(
78     -window.innerWidth/2 ,window.innerWidth/2 ,
79     window.innerHeight/2, -window.innerHeight/2,
80     1,1000
81   );
82   this.renderizador=new THREE.WebGLRenderer()
83   this.renderizador.setSize(window.innerWidth, window.innerHeight)
84   document.body.appendChild(this.renderizador.domElement)
```

Por practicidad manejamos una cámara ortogonal y una perspectiva, se ponen los códigos de añadir el canvas al body de la página de una vez

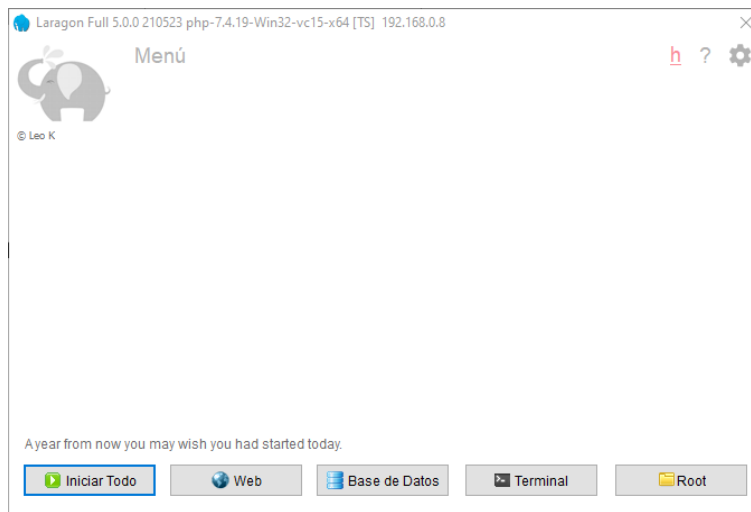


```
102     this.tuZ[1].position.y -= 20
103   }
104   animar(){
105     requestAnimationFrame(()=>{this.animar()})
106     this.renderizador.render(this.escena, this.camarap)
107   }
108
109 }
110 new Main().animar()
```

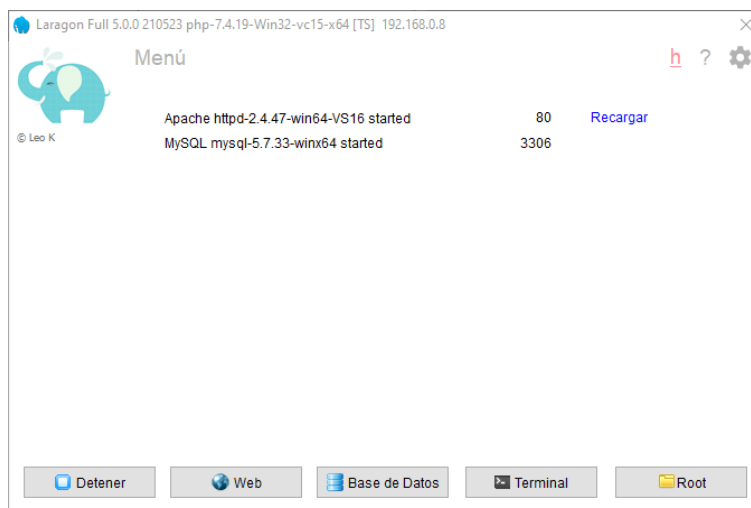
Estrictamente el método que vamos a ejecutar es el encargado de las animaciones que sigue incluyendo la renderización.



Algo nuevo es que los navegadores web están protegidos con políticas provocando que un script no como quiera puede tomar archivos del equipo donde se ejecuta para mostrarlos en el navegador. De modo que tal vez la mayoría de Loaders no funcionen para visualizar imágenes, texturas ni modelos tridimensionales.



Necesitamos cargar primero dichos recursos en un servidor, pero no nos conviene ninguno en la web porque principalmente cuando no haya conexión a internet dejen de funcionar.



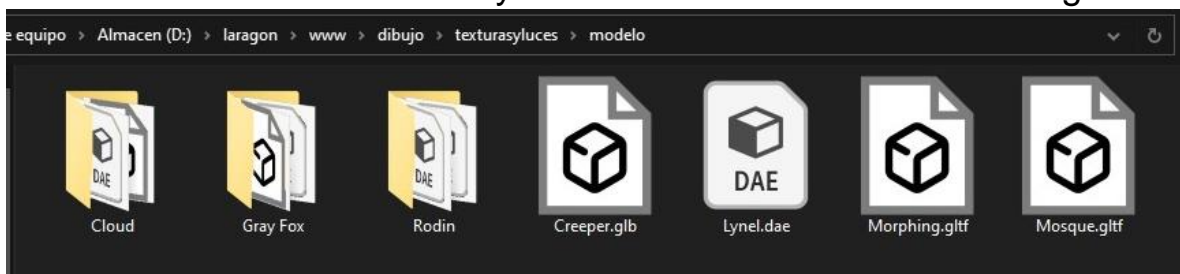
Una forma de solucionarlo es usar una aplicación que utilice WAMP en este caso Laragon, para volver nuestro equipo en un pequeño servidor y así evitaremos el error del navegador, y cuando subamos en alguna página nuestro proyecto funcionara correctamente en teoría.

Ejecución

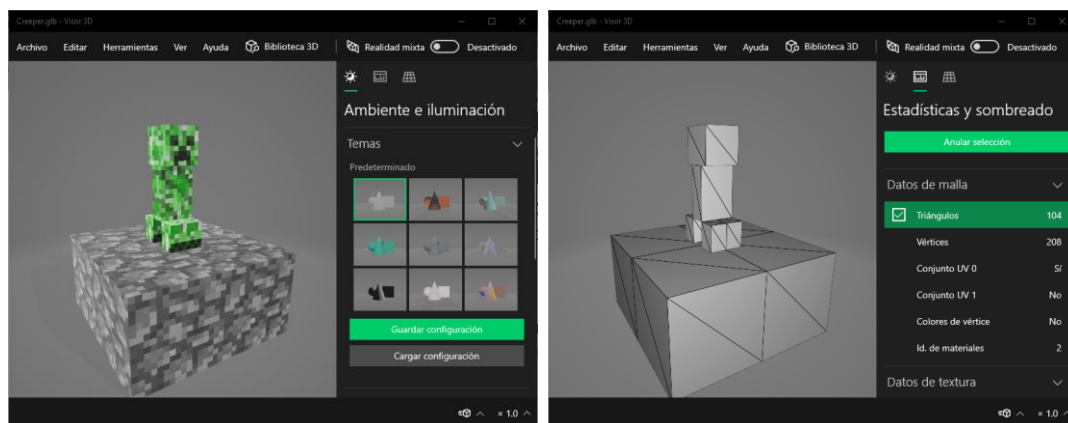
Ahora nos dirigimos a nuestra página enlazada al navegador web y recargamos para poder visualizar el ya conocido escenario oscuro de Three.js.

Importando modelo

Después de ya tener listo el modelo que queramos cargar, podemos proceder a realizar nuestra carga. Es recomendable tratar de visualizarlos de una vez y enterarnos de si les hace falta algo



El visor de Windows no reconoce los archivos del tipo DAE, mientras que con los OBJ nos los muestra en escala de grises, pero gltf y sus derivados se visualizan más o menos bien.



Empezando por lo más importante lo primero que debemos hacer es importar el cargador GLTFLoader antes mencionado.

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

Una vez que se haya importado el cargador, ya estaría listo para poder agregar un modelo a la escena. La sintaxis varía entre los distintos cargadores; y el que corresponde a GLTFLoader es:

```
const loader = new GLTFLoader();

loader.load( 'path/to/model.glb', function ( gltf ) {

    scene.add( gltf.scene );

}, undefined, function ( error ) {

    console.error( error );

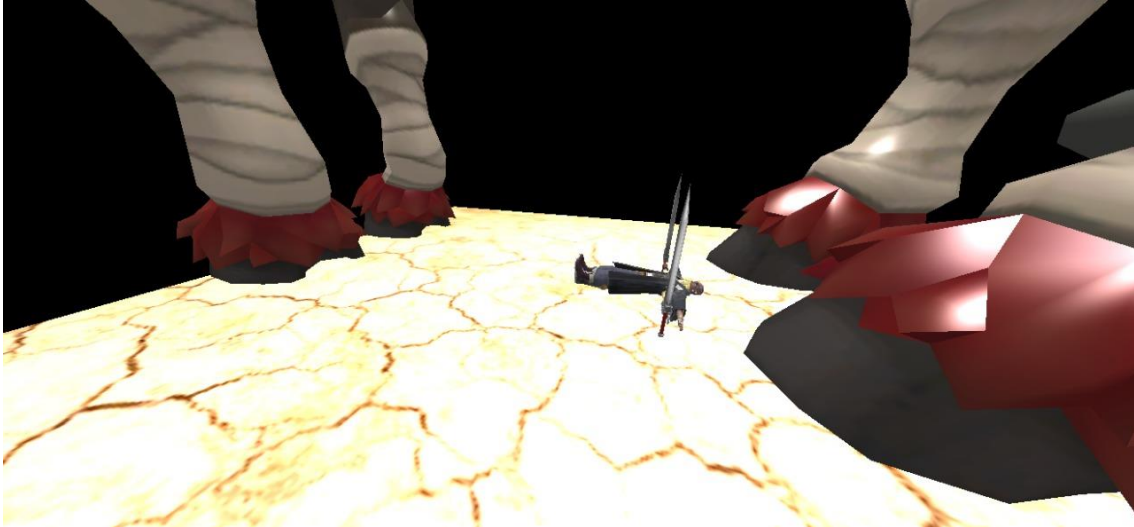
} );
```

Remplazamos la ubicación poniendo en su lugar la de nuestro archivo

```
const loader = new GLTFLoader();
loader.load('ubicación de nuestro modelo.glb', function(gltf){
    scene.add(gltf.scene);
}, undefined, function (error){
    console.error(error);
});
```

En la práctica hay varios posibles errores que pueden dispararse y no ser necesariamente nuestra culpa, desde las versiones en que los modelos fueron exportados hasta las versiones de los cargadores.

En eso es útil la consola que te ayuda a indagar cuál fue el origen que disparó dichos errores, nosotros constantemente vimos errores en el three-module.js diciendo que el modelo tenía position indefinida entre otros.



Si tus modelos son muy pequeños y quieres escalarlos te llevas la sorpresa de que no lo puedes hacer directamente y primero debes juntarlos en grupos, tiene lógica porque en las aplicaciones de modelado siempre que se puede se modela parte por parte del personaje.

```
File Edit Selection View Go Run Terminal Help codigojs - js - Visual Studio Code Área de trabajo de Windows Ink
JS codigojs x JS comportamiento.js
JS codigojs > Main > initMateriales
27 this.luz={
28   this.luz={
29   this.leon=new THREE.Group()
30   this.rubio=new THREE.Group()
31   this.brujo=new THREE.Group()
32   this.alien=new THREE.Group()
33   this.cyborg=new THREE.Group()
34   this.criper=new THREE.Group()
35   this.aspecto=window.innerWidth / window.innerHeight
36 }
37 initMateriales() {
```

En los atributos de necesito usar un atributo para cada uno de los modelos insertados y así poder trabajar con ellos su rotación, escala y posición.

```
File Edit Selection View Go Run Terminal Help
codigojs - js - Visual Studio Code

JS codigojs x
JS codigojs > Main > initMateriales

76 //modelos prediseñados
77 this.malla["Creeper"] = new GLTFLoader().load('modelo/Creeper.glb', (glb2)=>{
78   console.log(glb2)
79   this.criper.add(glb2.scene)
80   this.escena.add(this.criper)
81 });
82 this.malla["Alien"] = new GLTFLoader().load('modelo/Morphing.gltf', (gltf)=>{
83   console.log(gltf)
84   this.alien.add(gltf.scene)
85   this.escena.add(this.alien)
86 });
87 this.malla["leon"] = new ColladaLoader().load('modelo/Lynel.dae', (dae1)=>{
88   console.log(dae1)
89   this.leon.add(dae1.scene)
90   this.escena.add(this.leon)
91 });
92 this.malla["Rodin"] = new ColladaLoader().load('modelo/Rodin/rodin.dae', (dae2)=>{
93   console.log(dae2)
94   this.brujo.add(dae2.scene)
95   this.escena.add(this.brujo)
96 });
```

De modo que en la parte de insertar a escena el modelo, primero lo pasamos a nuestro grupo y lo que se sube es el grupo.

Tras mucha prueba y error fue evidente que algunos modelos gratuitos de la web estan incompletos por no decir dañados o a la hora de verlos con el visor 3D de Windows son muy diferentes a la previa de la página de origen





```

JS codigo.js x
JS codigo.js > Main > InitObjetos
137 this.malla[ "plano" ].rotateX(3*Math.PI/2)
138 this.escena.add(this.luz[0])
139 this.escena.add(this.luz[1])
140 this.alien.position.set(0,3,10)
141 this.alien.rotateY(3*Math.PI/2)
142 this.leon.position.set(-4,0,-7)
143 this.escalar(this.rubio,10,10,10)
144 this.rubio.rotateX(Math.PI/2)
145 this.rubio.position.set(4,0,-7)
146 this.escalar(this.brujo,10,10,10)
147 this.brujo.rotateX(Math.PI/2)
148 this.brujo.position.set(4,0,7)
149 this.escalar(this.cyborg,10,10,10)
150 this.cyborg.rotateX(Math.PI/2)
151 this.cyborg.position.set(-4,0,7)
152 this.criper.position.set(0,4,0)
153 this.escalar(this.criper,1.5,1.5,1.5)
154 this.luz[0].position.y=20
155 this.luz[1].position.y=20
156

```

Como se menciona previamente para trabajar con los modelos y transformarlos se hace con los grupos que creamos en `initAtributos`.

```

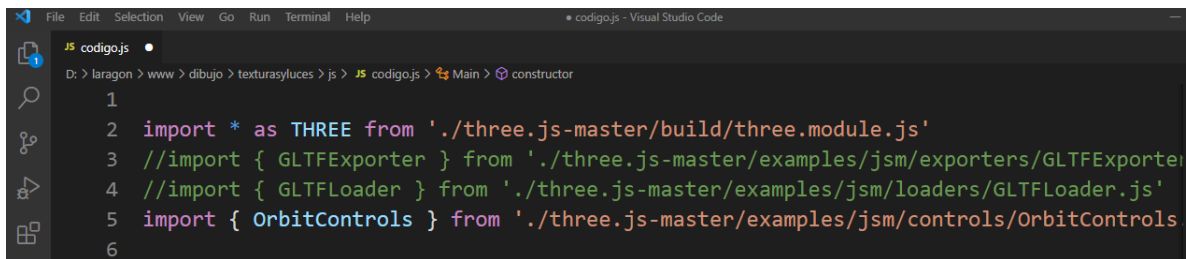
156 }
157 escalar(mllldplgns,x,y,z){
158     mllldplgns.scale.x=x
159     mllldplgns.scale.y=y
160     mllldplgns.scale.z=z
161 }

```

Para repetir menos veces estas 3 líneas de código decidí hacer un método adicional.

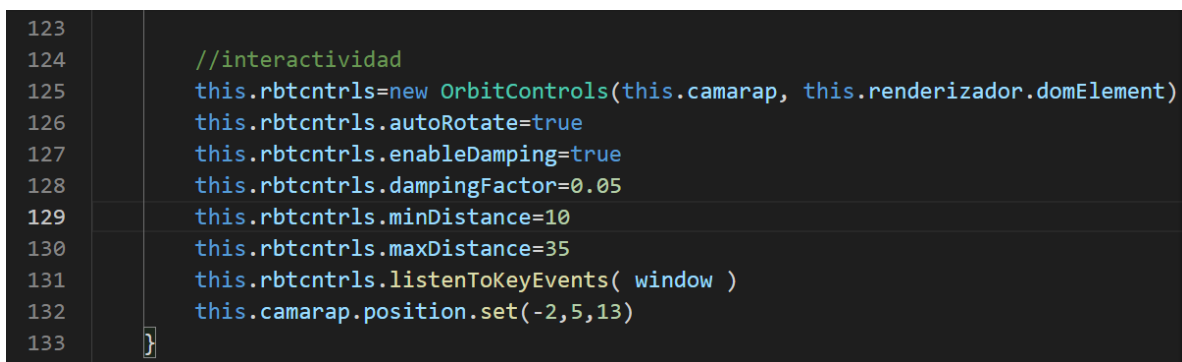
Controles de órbita

Sucede que a pesar de tener todo bonito carecerá de interactividad, es más cómodo pulsar en la pantalla y mover la cámara en distintos ángulos, pero siempre mirando el personaje, sin irnos a otro lado.



```
1
2 import * as THREE from './three.js-master/build/three.module.js'
3 //import { GLTFExporter } from './three.js-master/examples/jsm/exporters/GLTFExporter'
4 //import { GLTFLoader } from './three.js-master/examples/jsm/loaders/GLTFLoader.js'
5 import { OrbitControls } from './three.js-master/examples/jsm/controls/OrbitControls'
6
```

Queremos que la cámara siga una órbita como la de una luna alrededor de un centro y si es posible que mire siempre hacia un punto deseado algo que proporciona la clase de controles de órbita que un grupo de 8 usuarios hicieron con Three.js.



```
123
124 //interactividad
125 this.rbtcntrls=new OrbitControls(this.camarap, this.renderizador.domElement)
126 this.rbtcntrls.autoRotate=true
127 this.rbtcntrls.enableDamping=true
128 this.rbtcntrls.dampingFactor=0.05
129 this.rbtcntrls.minDistance=10
130 this.rbtcntrls.maxDistance=35
131 this.rbtcntrls.listenToKeyEvents( window )
132 this.camarap.position.set(-2,5,13)
133
```

Debajo de la carga del canvas creamos los controles de órbita, para darle una sensación de videojuego tendrá autorrotación a parte de la manipulación con las pulsaciones que es un ajuste por defecto.

Damping es la inercia de movimiento a modo de animación, de manera que al moverla esta tendrá un efecto más realista de ser una cámara, distancias mínimas y máximas para que las personas que visualicen el modelo no se acercan ni alejen demasiado, por defecto la órbita tendrá un radio de 15 a menos que el modelo que consigamos resulte ser muy grande.

```

JS codigo.js
JS codigo.js > Main > InitMateriales
112         this.luz[1].position.y=20
113     }
114     animar(){
115         requestAnimationFrame(()=>{this.animar()})
116         this.renderizador.render(this.escena, this.camarap)
117         this.rbtctrls.update()
118     }
119
120 }
121 new Main().animar()

```

Como la inercia y la autorrotación de la cámara son animaciones debemos actualizar los controles de orbita en el método de anímate. Con esto el proyecto final quedara sumamente atractivo para todo aquel que lo visualice.

Conclusiones

En resumen three.js es una excelente herramienta para aprender graficación tridimensional por computadora con la desventaja de que sería muy difícil hacer un modelado mas avanzado como el de un cuerpo humano.

Es muy recomendable hacer un diseño en un programa que te ofrezca un entorno de trabajo más cómodo con muchas herramientas destinadas a la creación y una vez que termines podrías exportarlo hacia Three.js para colocarlo en una página web.

Esto es tan conveniente para todos que ya hay decenas de paginas que contienen modelos enteros para descargarlos incluso de forma gratuita y si sabes su formato no es difícil encontrar el importador que necesitas. Una vez que tienes el objeto tridimensional en tu escenario debes cuidar de arreglar detalles, colocarle la animación en caso de que sea OBJ, colocando luces ambientales en caso de que use materiales sensibles a la luz, escalarlo, entre otros.

Es increíble la poderosa utilidad de unas pocas líneas de código, teniendo conocimiento de la cantidad de operaciones de geometría analítica y calculo vectorial, mientras que nosotros los usuarios de Three.js en unos pocos minutos con menos de 100 líneas de código podemos generar algo que hace años hubiera parecido ciencia ficción, parece que la mayor barrera por romper es la de la imaginación