

Instituto Tecnológico Superior de Tierra Blanca

Semestre Agosto 2021 Enero 2022

Ingeniería en Sistemas Computacionales

ISIC 2010-224

SCC-1010 Graficación

Unidad II Graficación 2D

Practica | Voxelarts con JavaScript

Blanca Ramírez Cruz 198N0530

Marian Araceli Uriarte Sarmiento 198N0830

Sergio Jared Valencia Cortaza 198N0068

Tierra Blanca Veracruz



Índice

Introducción	1
Desarrollo	2
Creación del proyecto	2
Ejecución	5
Three js	6
Figuras en un escenario	8
El nodo	10
Transformaciones básicas	13
Traslación	13
Escalamiento.....	14
Rotación	15
Conclusión	17

Introducción

Para crear elementos visuales dinámicos fue creado el lenguaje de programación interpretado JavaScript que es considerado como lenguaje estructurado, aunque hoy en día ya se pueden trabajar objetos con él.

Para utilizarlo necesitas un documento creado con hipertexto como por ejemplo una página tipo HTML y después enlazar ambos archivos, para poder observar correctamente un documento de este tipo podemos usar simplemente un navegador actualizado.

Desarrollo

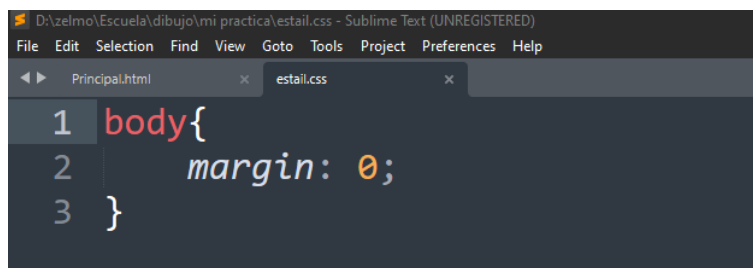
Con ayuda de una librería de código abierto vamos a crear sobre un documento un escenario tridimensional con una cámara que verá los objetos renderizados, sin embargo, visualizamos los voxels con una cámara ortogonal y parecerá solo una imagen bidimensional.

Creación del proyecto

Primero comenzamos creando el sitio web, una carpeta con recursos multimedia, otra para los códigos de JS, un archivo que le dará estilos bonitos a los elementos de la página en caso de tenerlos y de ahí todas las páginas html que necesitemos.



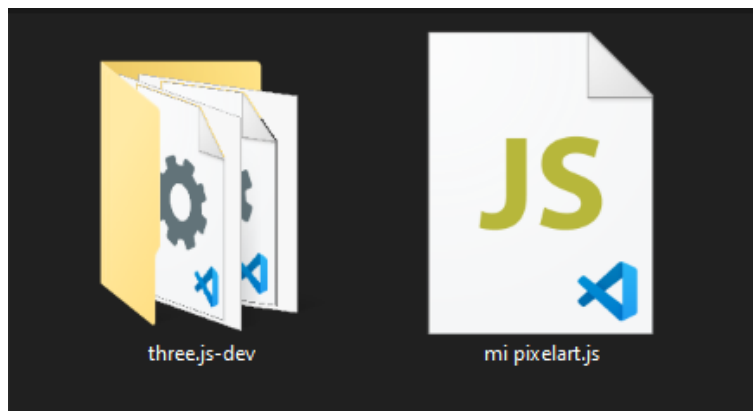
El sistema operativo en el que trabajamos es Windows 10, editor de texto plano es Sublime text en su versión gratuita, aunque para la programación en JS usamos Visual Studio Code y por último el navegador usado es Microsoft Edge junto con Google Chrome



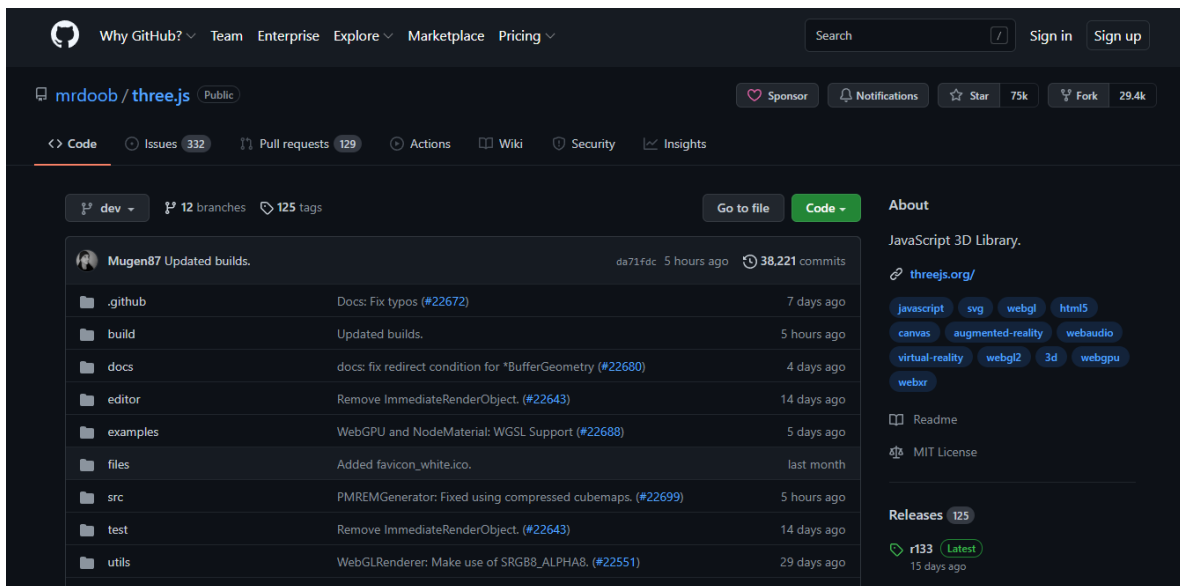
En los estilos simplemente quitamos el margen que tienen por defecto todas las páginas, esto porque no pusimos ni texto ni imágenes en la práctica.

```
D:\zeimo\Escuela\ dibujo\mi practica\Principal.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
Principal.html x estail.css x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Intento</title>
5     <meta charset="utf-8">
6     <link rel="stylesheet" type="text/css" href="estail.css">
7   </head>
8   <body>
9     <script type="text/javascript" src="jvscrp/three.js-dev/
10    build/three.js"></script>
11    <script type="text/javascript" src="jvscrp/mi pixelart.js">
12    </script>
13  </body>
14 </html>
```

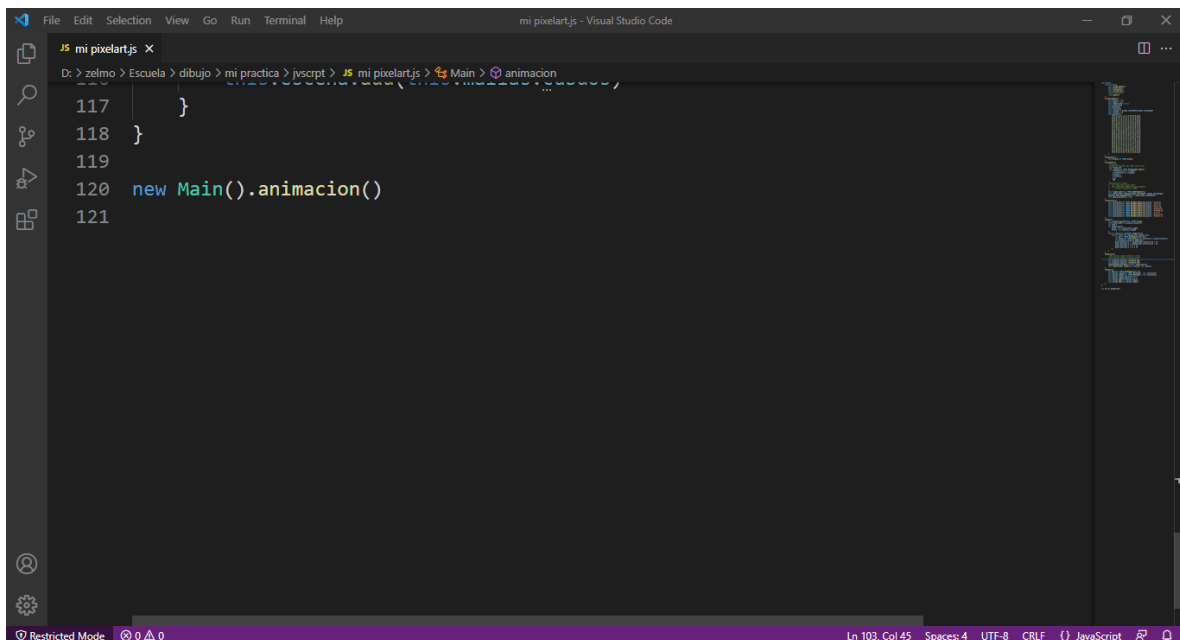
En la página principal enlazamos a los estilos en la cabecera y en el cuerpo a los dos archivos de JavaScript que están ubicados en la respectiva carpeta de códigos.



El archivo JS es el que escribimos completamente nosotros con ayuda del paquete que se encuentra al lado descargado de una página oficial.



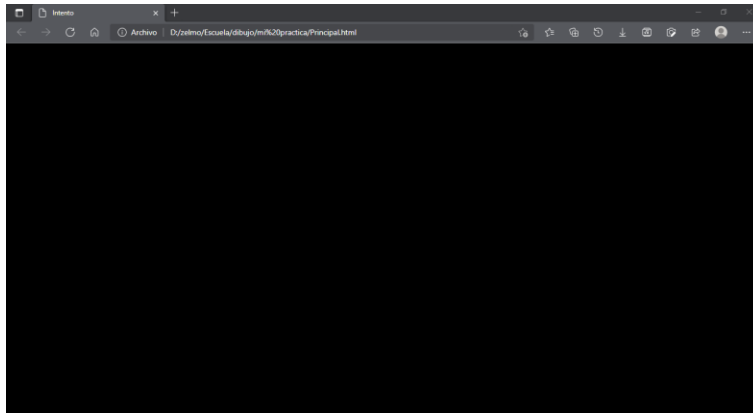
Este es un proyecto de Github que ofrece precisamente lo que nosotros necesitábamos, un entorno con cámaras donde creas componentes y trabajas con sus propiedades mediante JS, solo necesitas buscar los métodos en su documentación.



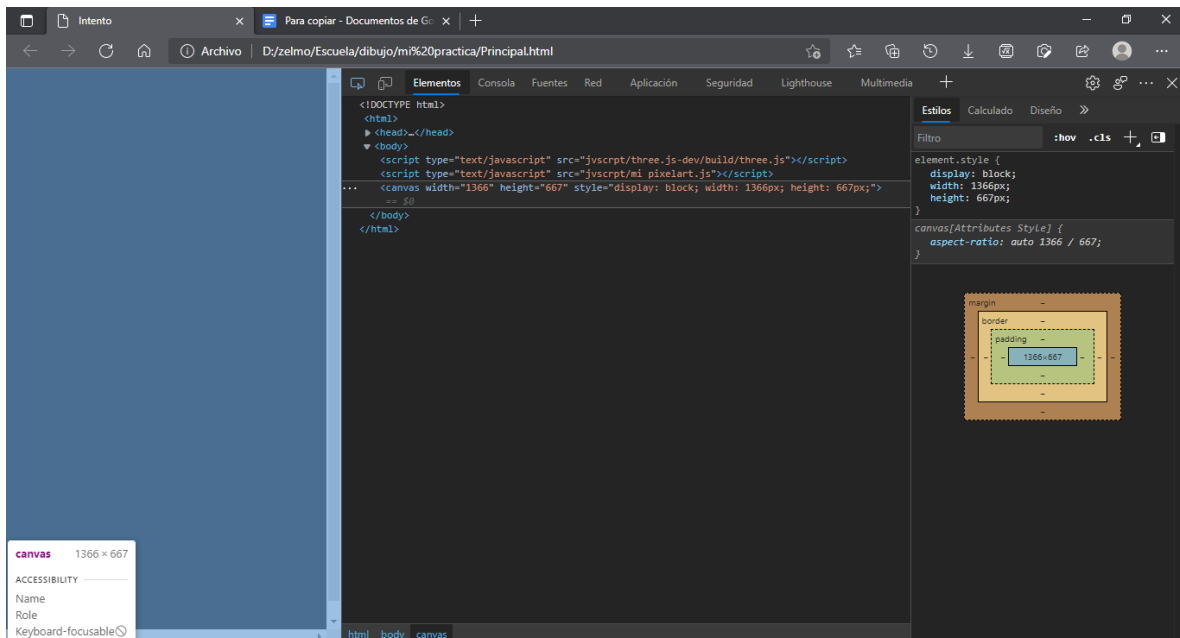
Al abrir nuestro código en realidad el único método que es invocado es el `animacion()` y antes de él se crea una clase `Main()` en la cual solo hay un constructor que no necesita parámetros.

Ejecución

Solo es necesario abrir nuestra página enlazada en cualquier navegador web y lo primero que veremos en caso de no tener errores será un simple color negro.



En caso de no verlo la mejor explicación es la existencia de errores en el código ya sea en la página o en el JS, es recomendable ver la consola del navegador donde te indica los fallos.



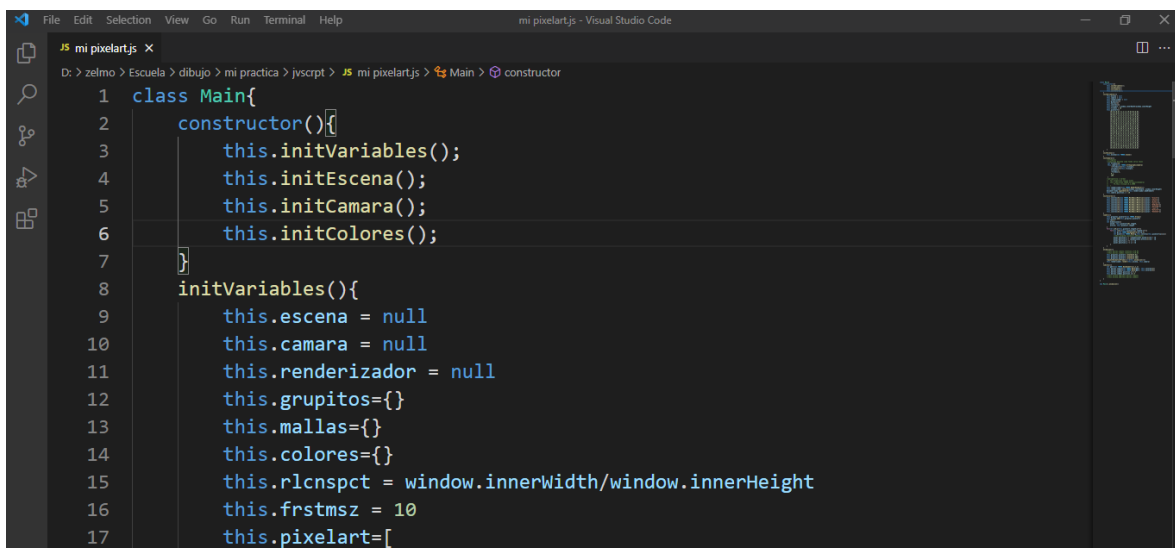
Siendo más curiosos vemos que el navegador interpreta una nueva etiqueta que no escribimos, es un canva con el ancho y alto de la ventana en ese momento además de un estilo de bloque.

Esta fue insertada al momento de renderizar la escena con la cámara como veremos próximamente.

Si vemos esto se debe a que seguramente así se ve por defecto el `THREE.Scene()` que estamos aprovechando.

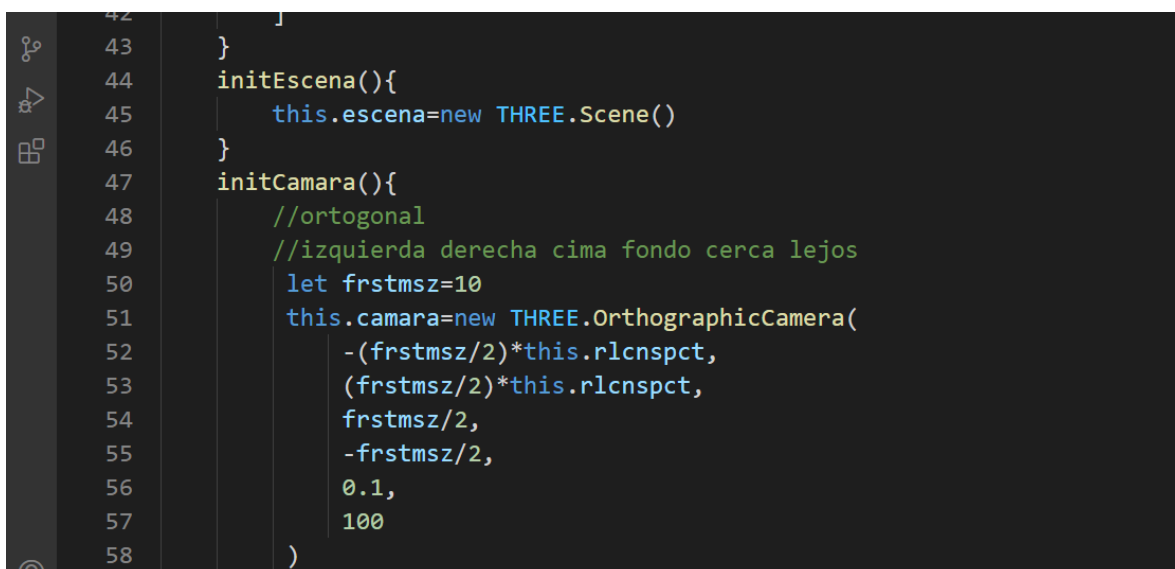
Three js

Como buena práctica tratamos de organizar todo en distintos lugares y es por eso que esta práctica ha sido dividida en distintas funciones.



```
1 class Main{
2   constructor(){
3     this.initVariables();
4     this.initEscena();
5     this.initCamara();
6     this.initColores();
7   }
8   initVariables(){
9     this.escena = null
10    this.camara = null
11    this.renderizador = null
12    this.grupitos={}
13    this.mallas={}
14    this.colores={}
15    this.rlcnspt = window.innerWidth/window.innerHeight
16    this.frstmsz = 10
17    this.pixelart=[
```

Resumidamente vamos a renderizar un mundo tridimensional y para ello necesitamos el mundo y una cámara para ver que hay en él.



```
43   }
44   initEscena(){
45     this.escena=new THREE.Scene()
46   }
47   initCamara(){
48     //ortogonal
49     //izquierda derecha cima fondo cerca lejos
50     let frstmsz=10
51     this.camara=new THREE.OrthographicCamera(
52       -(frstmsz/2)*this.rlcnspt,
53       (frstmsz/2)*this.rlcnspt,
54       frstmsz/2,
55       -frstmsz/2,
56       0.1,
57       100
58     )
```

```
59 //perspectiva o normal
60 // fov proporcion inicio final
61 // this.camara=new THREE.PerspectiveCamera(
62 //     75,this.rlcnspect,0.1,1000
63 // )
64 this.renderizador=new THREE.WebGLRenderer();
65 this.renderizador.setSize(window.innerWidth, window.innerHeight)
66 document.body.appendChild(this.renderizador.domElement)
67 this.camara.position.z = 10
68 }
```

Es muy fácil crear el mundo, pero en cuanto a la cámara existen dos posibilidades y tenemos preparadas ambas para cada ocasión, no obstante, del tipo de cámara aprovechamos a crear un renderizador y lo añadimos al cuerpo del documento.

```
97     }
98   }
99 }
100 animacion(){
101     requestAnimationFrame(=>{this.animacion()})
102     this.renderizador.render(this.escena, this.camara)
103 }
```

El método de animación aparte de renderizar nuestro proyecto pone un añadido necesario para ver movimientos en los componentes cuando los tengamos.

Justo ahora si ejecutamos el escenario no tendrá cambio alguno visualmente, esto se debe a que aún no añadimos ningún componente tridimensional, solo tenemos el cuarto oscuro y vacío.

Figuras en un escenario

Los demás métodos son complementarios a los necesarios para el mundo tridimensional.

```
68     }
69     initColores(){
70         this.colores[0]=new THREE.MeshBasicMaterial({color: "black"})
71         this.colores[1]=new THREE.MeshBasicMaterial({color: "white"})
72         this.colores[2]=new THREE.MeshBasicMaterial({color: "green"})
73         this.colores[3]=new THREE.MeshBasicMaterial({color: "yellow"})
74         this.colores[4]=new THREE.MeshBasicMaterial({color: "#f0cb7d"})
75         this.colores[5]=new THREE.MeshBasicMaterial({color: "#4f3706"})
76         this.colores[6]=new THREE.MeshBasicMaterial({color: "cyan"})
77         this.colores[7]=new THREE.MeshBasicMaterial({color: "#1f1f1f"})
78         this.colores[8]=new THREE.MeshBasicMaterial({color: "#333333"})
79     }
```

Como el de los materiales básicos que por el momento únicamente contiene colores en RGB hexadecimal.

```
10     initVariables(){
11         this.escena = null
12         this.camara = null
13         this.renderizador = null
14         this.grupitos={}
15         this.mallas={}
16         this.colores={}
17         this.rlcnspt = window.innerWidth/window.innerHeight
18         this.frstmsz = 10
19         this.pixelart=[
```

Y el de nuestros valores que siempre es más organizado instanciar desde la parte más alta de la clase, deberíamos de colocar todos nuestros componentes en esta parte.

```
100     cubitos(){
101         let gmtr=new THREE.BoxGeometry(2,2,2)
102         this.mallas.cubuno=new THREE.Mesh(gmtr, this.colores[2])
103         this.mallas.cubdos=new THREE.Mesh(gmtr, this.colores[3])
104         this.mallas.cubuno.position.x=-1.5
105         this.mallas.cubdos.position.x=1.5
106         this.escena.add(this.mallas.cubuno)
107         this.escena.add(this.mallas.cubdos)
108     }
```

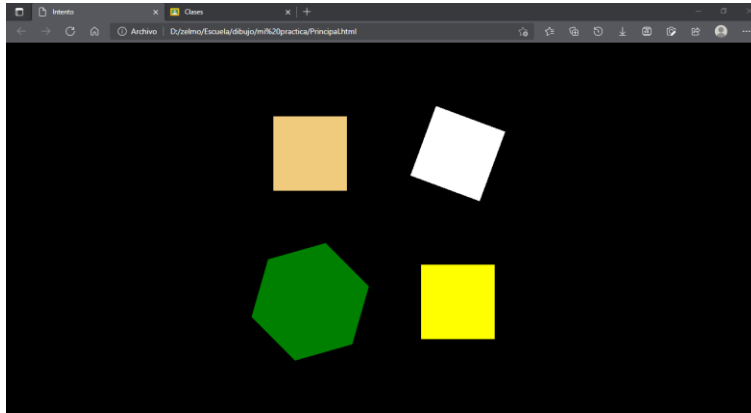
Resulta que todos los componentes que vemos son considerados como una malla de polígonos, antes de crear dichas mallas es necesario determinar su geometría incluyendo cuánto mide cada lado.

Como es una malla todos los componentes que deseemos los añadimos en el arreglo de mallas determinando un color para todos los lados.

Como es una malla todos los componentes que deseemos los añadimos en el arreglo de mallas determinando un color para todos los lados.

Sin olvidar indicar dónde estará este componente con coordenadas, tuvimos en cuenta que siempre se colocaban en el centro de la ventana así que ese debía ser el origen.

Por último, a pesar de haber creado los objetos es obligatorio añadirlos a escena, de modo que es posible tener muchos componentes y ver tan solo unos de ellos.



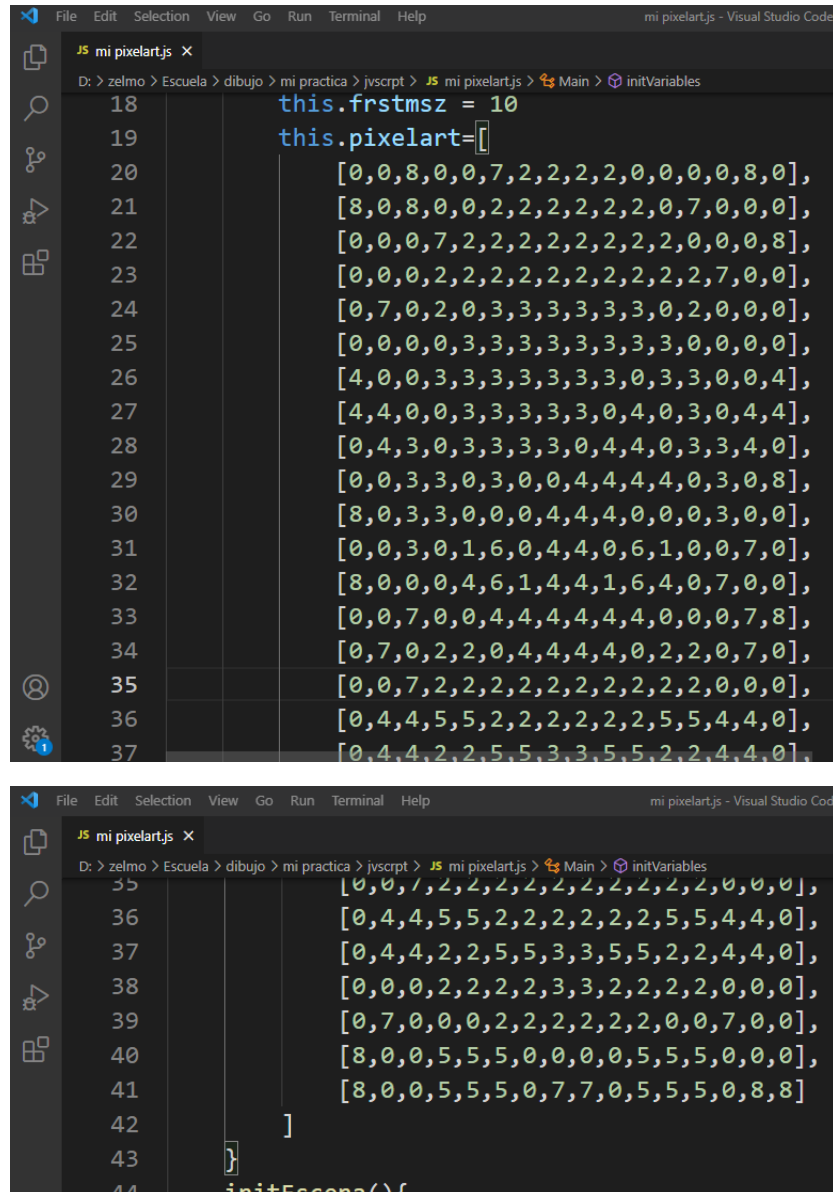
El nodo

Hay una desventaja a la hora de trabajar con múltiples componentes y es para el caso de que quieras hacer movimientos o redimensiones con ellos, puedes ahorrarte algo de tiempo utilizando un `THREE.Group()` que visualmente hace cambios a todos como si fueran un solo componente.

```
File Edit Selection View Go Run Terminal Help
mi pixelart.js - Visual Studio Code
D:\zemo\Escuela > dibujo > mi practica > jsrpt > JS mi pixelart.js > Main > cuadro

80 cuadro(){
81   this.grupitos.pixelart=new THREE.Group()
82   this.escena.add(this.grupitos.pixelart)
83   let ld=0.3
84   let dimensiones={
85     ancho: this.pixelart[0].length,
86     altura: this.pixelart.length
87   }
88   for(let i=0;i<this.pixelart.length;i++){
89     for(let j=0;j<this.pixelart[i].length;j++){
90       let g=new THREE.BoxGeometry(ld,ld,ld)
91       let pixel=new THREE.Mesh(g,this.colores[this.pixelart[i][j]])
92       this.grupitos.pixelart.add(pixel)
93       pixel.position.x = -(dimensiones.ancho/2)*ld
94       pixel.position.y = (dimensiones.altura/2)*ld
95       pixel.position.x += j * ld
96       pixel.position.y -= i * ld
97     }
98   }
99 }
```

Lo añadimos inmediatamente a escena y rápidamente comenzamos a trabajar con la parte del Pixel Art, en realidad son decenas de cubos colocados en un rectángulo vistos con una cámara ortogonal.



```

18     this.frstmsz = 10
19     this.pixelart=[
20         [0,0,8,0,0,7,2,2,2,2,0,0,0,0,8,0],
21         [8,0,8,0,0,2,2,2,2,2,2,0,7,0,0,0],
22         [0,0,0,7,2,2,2,2,2,2,2,2,0,0,0,8],
23         [0,0,0,2,2,2,2,2,2,2,2,2,2,7,0,0],
24         [0,7,0,2,0,3,3,3,3,3,3,0,2,0,0,0],
25         [0,0,0,0,3,3,3,3,3,3,3,3,0,0,0,0],
26         [4,0,0,3,3,3,3,3,3,0,3,3,0,0,4],
27         [4,4,0,0,3,3,3,3,3,0,4,0,3,0,4,4],
28         [0,4,3,0,3,3,3,3,0,4,4,0,3,3,4,0],
29         [0,0,3,3,0,3,0,0,4,4,4,0,3,0,8],
30         [8,0,3,3,0,0,0,4,4,4,0,0,0,3,0,0],
31         [0,0,3,0,1,6,0,4,4,0,6,1,0,0,7,0],
32         [8,0,0,0,4,6,1,4,4,1,6,4,0,7,0,0],
33         [0,0,7,0,0,4,4,4,4,4,0,0,0,7,8],
34         [0,7,0,2,2,0,4,4,4,4,0,2,2,0,7,0],
35         [0,0,7,2,2,2,2,2,2,2,2,2,2,0,0,0],
36         [0,4,4,5,5,2,2,2,2,2,2,5,5,4,4,0],
37         [0,4,4,2,2,5,5,3,3,5,5,2,2,4,4,0],
38         [0,0,0,2,2,2,2,3,3,2,2,2,2,0,0,0],
39         [0,7,0,0,0,2,2,2,2,2,2,0,0,7,0,0],
40         [8,0,0,5,5,5,0,0,0,0,5,5,5,0,0,0],
41         [8,0,0,5,5,5,0,7,7,0,5,5,5,0,8,8]
42     ]
43 }
44 initEscena(){

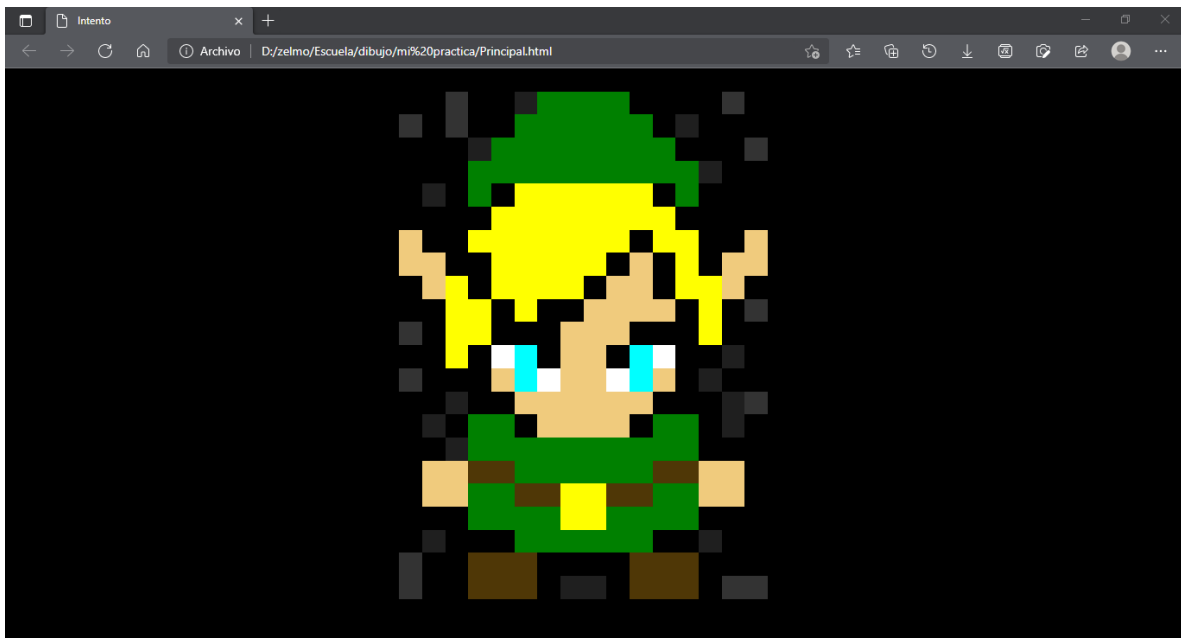
```

Por lo tanto, recorrimos una matriz con 2 ciclos for tomar las dimensiones totales de nuestra variable pixelart y en cada casilla de la matriz creamos una geometría que será idéntica para todos.

La matriz determina qué color tendrá cada casilla, el número en su interior llama al índice del arreglo de materiales básicos, una vez creado podemos añadirlo al grupo sin esta línea no se ve nada.

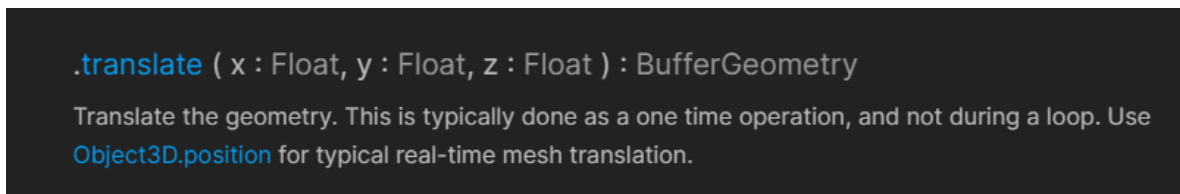
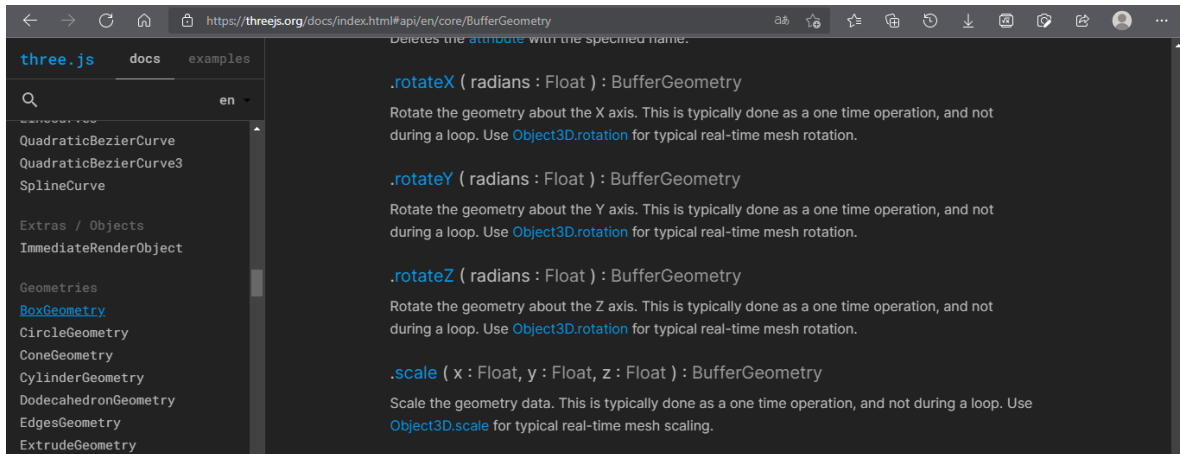
Lo siguiente es colocarlos correctamente porque si no se hace veremos algo extraño en lugar del efecto de imagen que buscamos, la posición de los colores en la matriz debe coincidir con ese escenario.

A todos los pixeles los llevaremos primero a la esquina superior izquierda y una vez ahí los empujamos a la derecha y hacia abajo dependiendo de las iteraciones en el ciclo.



Transformaciones básicas

En la documentación de Threejs vemos las funciones básicas que queremos, todas tienen como parámetros unas coordenadas o un valor en radianes así que fueron fáciles de recordar.

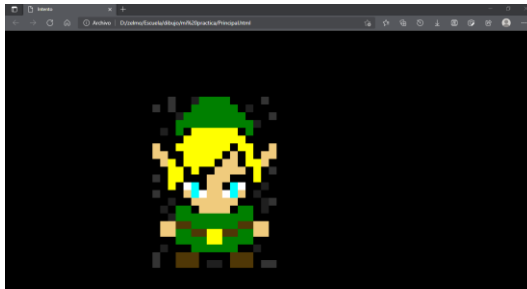
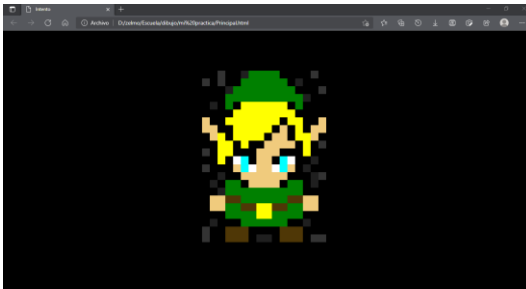


Traslación

Al grupo entero le movimos del origen a una posición determinada y bastante aleatoria, esto es después de su creación así que no es necesario cambiar las líneas en el respectivo método.

```
97         }
98     }
99     this.grupitos.pixelart.translateX(0)
100    this.grupitos.pixelart.translateY(0)
101    this.grupitos.pixelart.translateZ(0)
102    }
103    cubitos(){
```

```
97         }
98     }
99     this.grupitos.pixelart.translateX(-2)
100    this.grupitos.pixelart.translateY(-1)
101    this.grupitos.pixelart.translateZ(-10)
102    }
103    cubitos(){
```

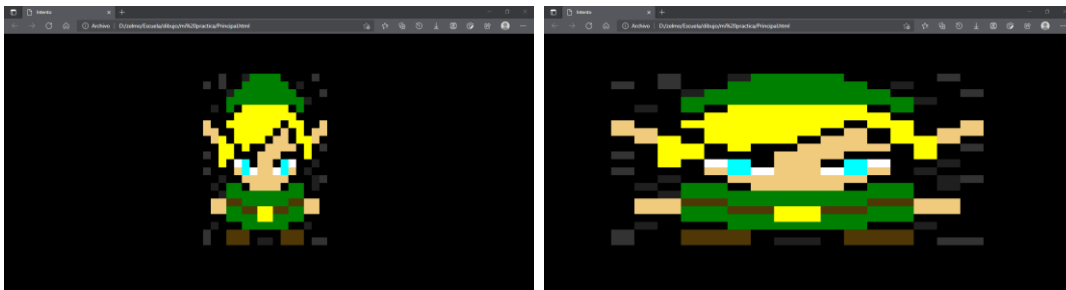


Escalamiento

En este caso un valor de 1 en cada eje equivale a la misma figura, así que a modo de porcentaje se fue realizando una variación a su tamaño, recordando que escalar por igual en todos los ejes no deforma el componente y caso contrario lo deforma notablemente.

```
119    }
120    animacion(){
121        this.grupitos.pixelart.scale.x=1
122        this.grupitos.pixelart.scale.y=1
123        this.grupitos.pixelart.scale.z=1
124        requestAnimationFrame(()=>{this.animacion()})
125        this.renderizador.render(this.escena, this.camara)
126    }
127
128 }
```

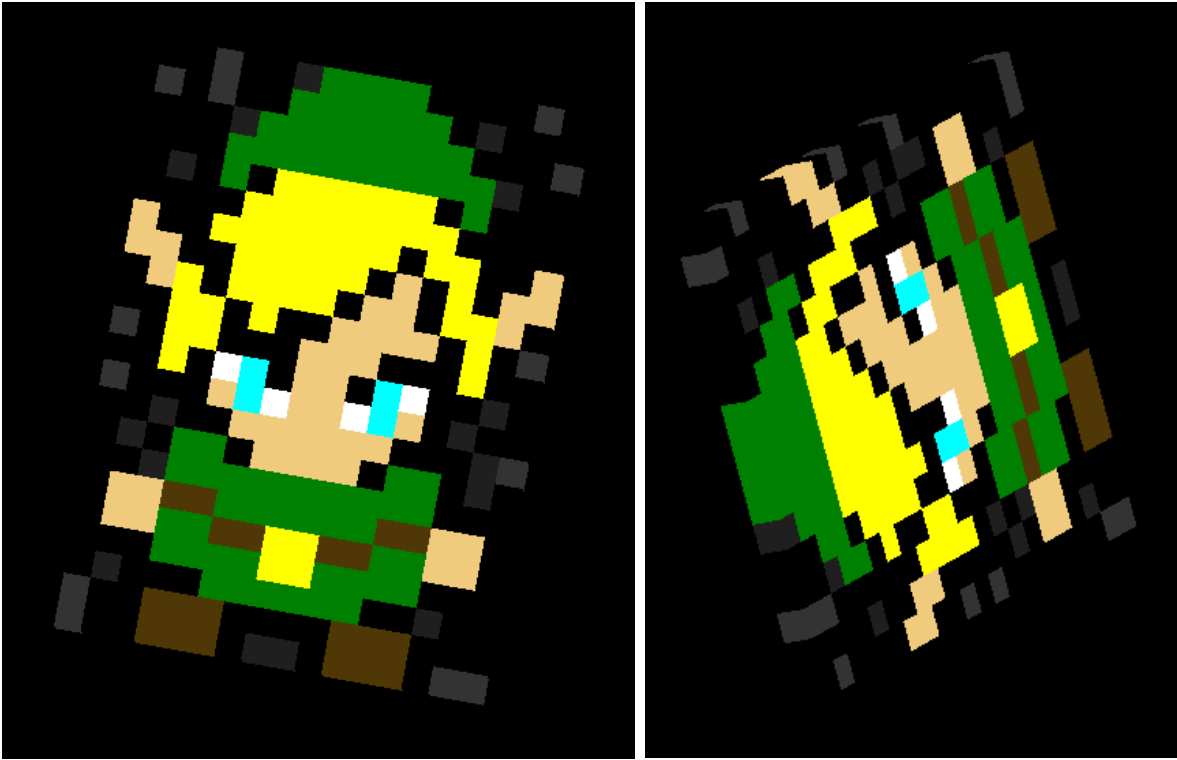
```
119 }
120 animacion(){
121     this.grupitos.pixelart.scale.x=3
122     this.grupitos.pixelart.scale.y=1
123     this.grupitos.pixelart.scale.z=2
124     requestAnimationFrame(()=>{this.animacion()})
125     this.renderizador.render(this.escena, this.camara)
126 }
127
128 }
```



Rotación

Podríamos previamente crear una función secundaria con el propósito de convertir unidades de grados a radianes y de esa manera colocar valores de 0 a 360 pero nosotros lo dejamos así poniendo directamente valores bajos.

```
119 animacion(){
120     this.grupitos.pixelart.rotateZ(0.01)
121     this.grupitos.pixelart.rotateY(0.005)
122     this.grupitos.pixelart.rotateX(0.005)
123     requestAnimationFrame(()=>{this.animacion()})
124     this.renderizador.render(this.escena, this.camara)
125 }
126
127 }
```

Conclusión

El propósito era hacer un trabajo bidimensional pero mejor que eso empezamos a manejar un escenario tridimensional con una cámara e incluso un proceso de renderizado muchas veces por segundo.

Son contados los casos reales donde diseñadores profesionales usaron cámaras ortogonales y texturas planas que combinaron de forma estratégica para provocar en la vista un efecto de imagen bidimensional, sobre todo en el mundo de los videojuegos.

Todo sin mencionar el gran mérito de la librería de código abierto Threejs ya que con él en cuestión de unos minutos podemos hacer efectos visuales bonitos y llamativos, nos ahorra una gran parte del trabajo con sus objetos y métodos que cada vez ofrece más características.

A ciencia cierta para realizar este tipo de proyecto de programación que representa objetos que nuestros sentidos pueden entender solo es necesaria una conexión a internet y unos cuantos editores de texto.