[[1]](#footnote-1)

Segundo parcial Computación Gráfica

Daniela Alvarado Arboleda -1201924

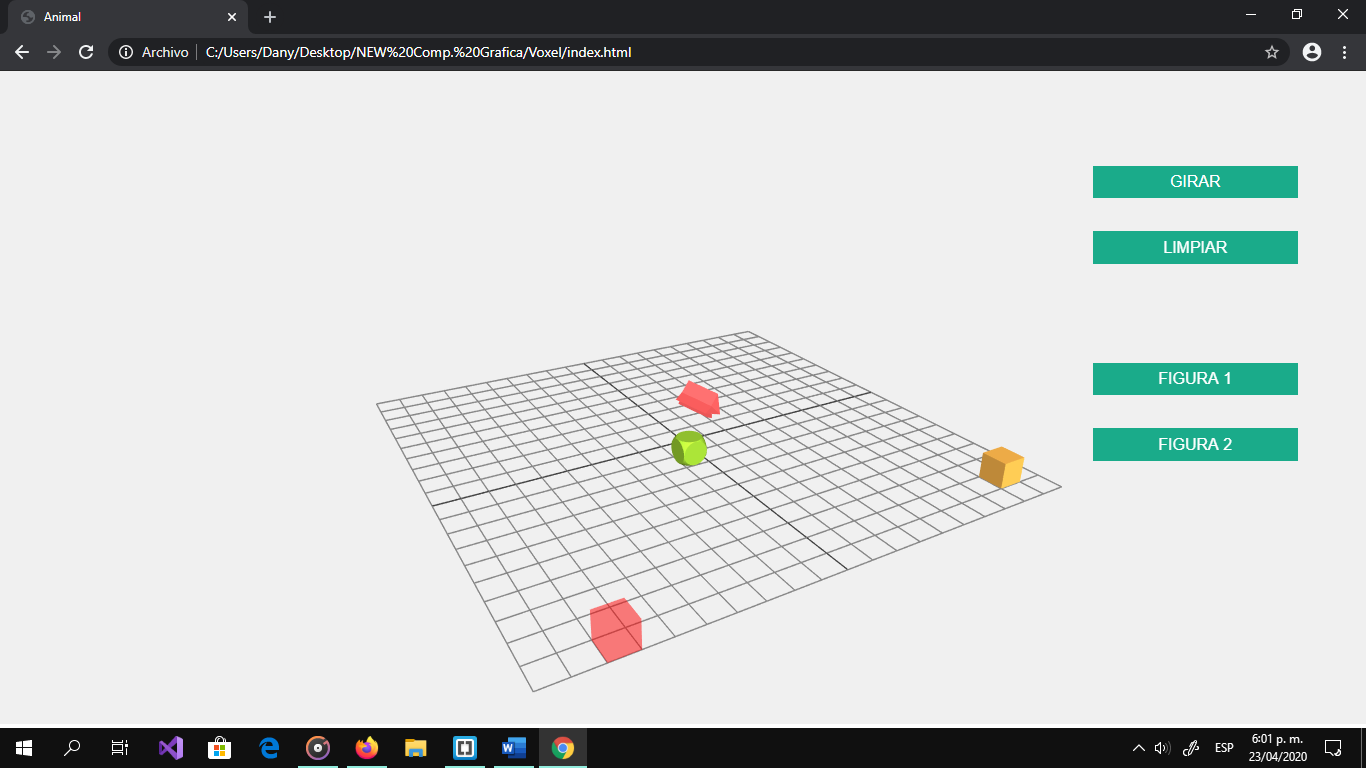
I.OBJETVOS

Utilizando el Voxel\_painter que aparece en Three.js, mejorar el programa y asi´mismo agregar nuevas funcionalidades

II.DESCRIPCIÓN

El siguiente documento tiene el registro de los pasos y operaciones para realizar figuras utilizando las diferentes formas de modelado: Fan/Strip, geometría constructiva, Extrude y Spline y primitivas.

Lo primero que se hace, luego de montar el código base es definir los botones que van a aparecer en la interfaz, los dos primeros siendo: GIRAR y LIMPIAR, que consisten en girar la cámara y eliminar la escena respectivamente.



Img1. Gráfico base para la interacción entre el usuario y la escena.

A continuación, se muestra el código empleado. Iniciando por la creación de estos.

**<button id='rot' type="button" style="left:80%; top:15%" onClick="animateClick()">GIRAR</button>**

**<button id='reset' type="button" style="left:80%; top:25%" onClick="stopClick()">LIMPIAR</button>**

Después se hacen las funciones, una que permite el giro de la cámara cada que se oprime el botón y otra que borra todos los elementos de la escena.

Como problema que surgían fueron los siguientes, el primero es que, al tener la animación de la cámara continua, es decir ya definida y que no requería de un botón, por lo mismo antes mencionado la existencia del botón era inútil y a eso agregarle que la interacción con la escena era difícil, por no decir incómoda ya que se movía el escenario.

**function animate() {**

**var time = -performance.now() \* 0.0003;**

**camera.position.x = 1000 \* Math.cos(time);**

**camera.position.z = 1000 \* Math.sin(time);**

**camera.lookAt(scene.position);**

**render();**

**stats.update();**

**}**

La solución a este inconveniente fue eliminar el “requestAnimationFrame(animate);” y agregar la siguiente línea en la función init():

**document.getElementById('rot').addEventListener('click', animate);**

De este modo así se llamaba la función animar únicamente cuando se oprimía el botón “GIRAR”.

Para el botón de limpiar se siguió el mismo procedimiento, con otra función y así mismo otro llamado en el init() y aunque, funciona, no es el objetivo que se esperaba ya que borra toda la escena.

**function resetScene() {**

**while (scene.children.length > 0) {**

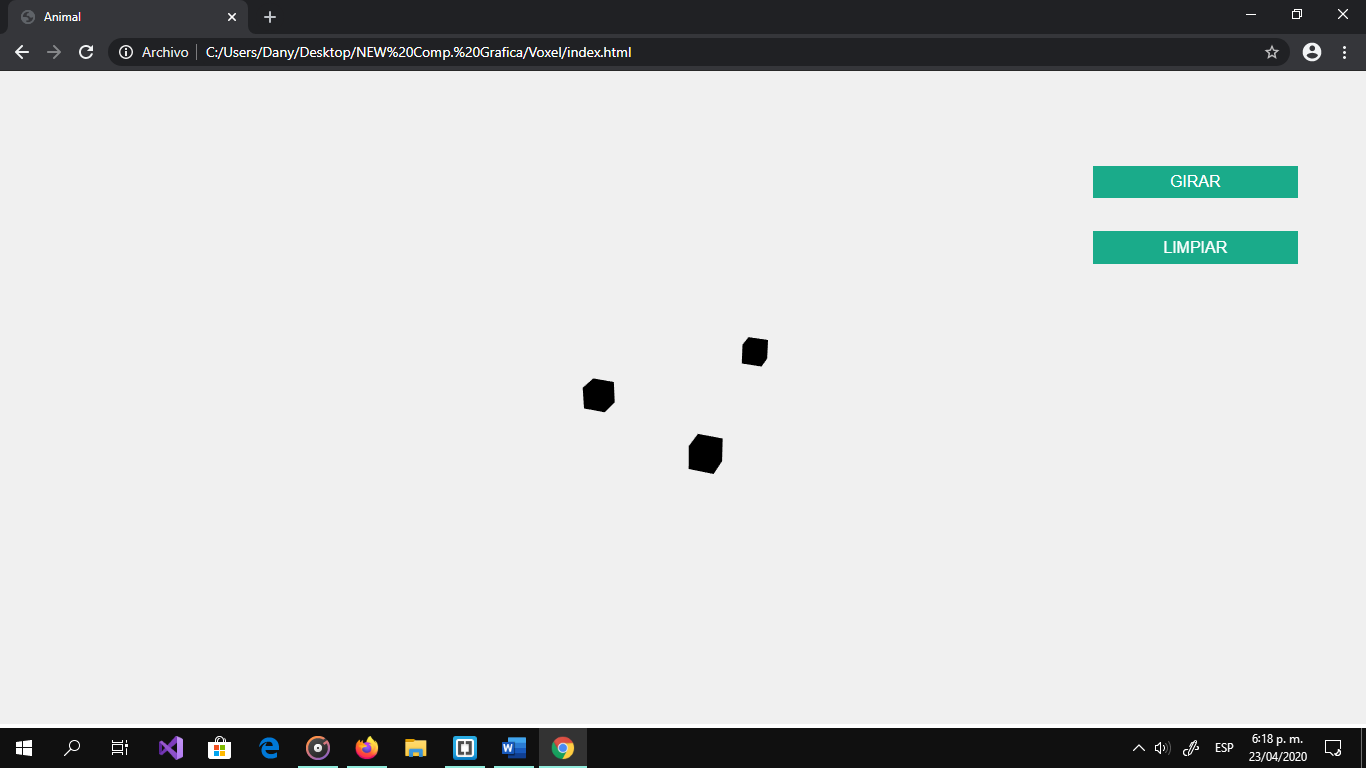
**scene.remove(scene.children[0]);**

**}**

**}**

Y se llama la función:

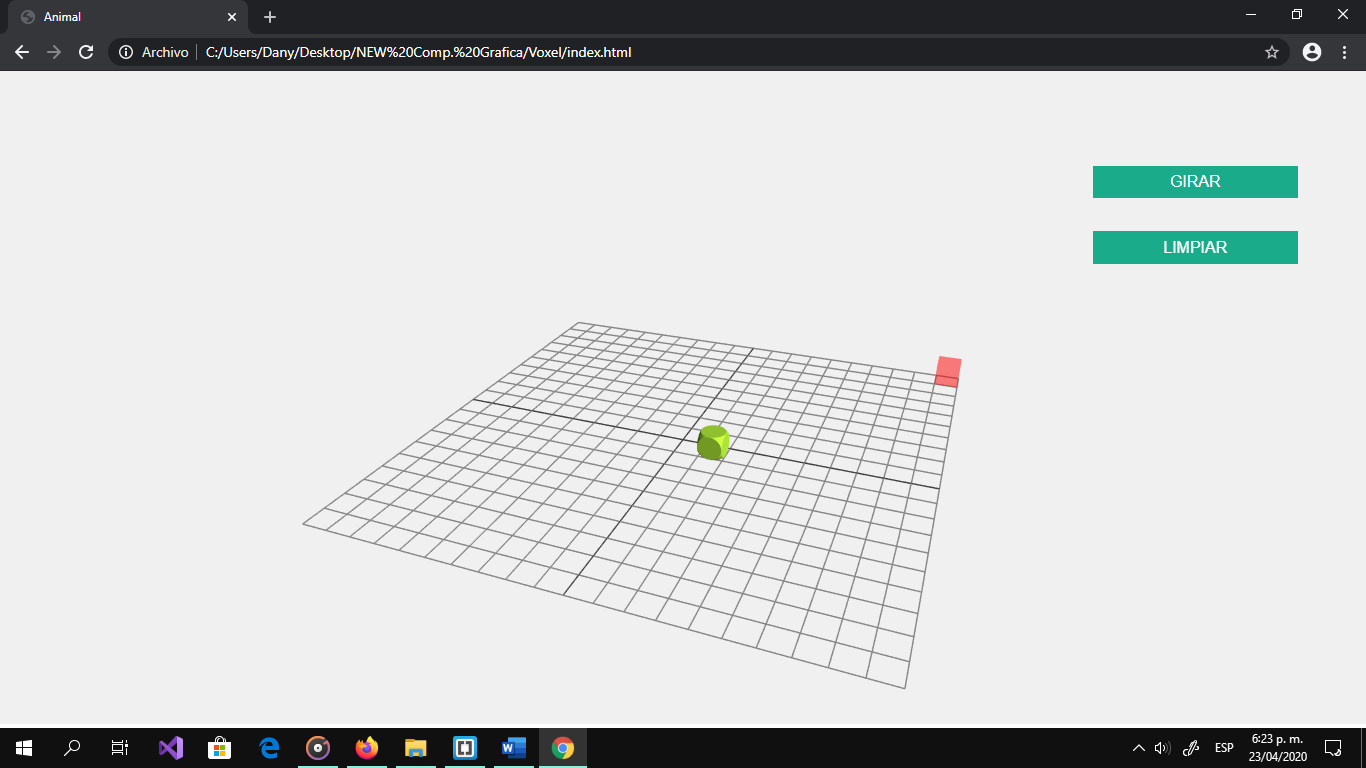
**document.getElementById('reset').addEventListener('click', resetScene);**



Img2. Inconveniente al borrar la escena.

Como se ve en la imagen, se eliminan tanto las figuras creadas, tanto como la grilla, pero aun se puede interactuar con el escenario.

Posteriormente, y para la creación de otra figura por medio de CSG, se reutilizó código de un ejercicio anterior y se adaptó a este para que el objeto calzara con la escena.



Img3.Figura con forma de dado CSG.

Se crean las primitivas con todo lo que esto conlleva, geometría y material. En este caso el material es Lambert para que los colores queden más vibrantes con respecto a la iluminación.

**var boxGeometry = new THREE.BoxGeometry(50, 50, 50);**

**var sphereGeometry = new THREE.SphereGeometry(35, 35, 50);**

**var colorR = new THREE.Color(0x9ACD32);**

**material1 = new THREE.MeshLambertMaterial({color: colorR});**

**cube = new THREE.Mesh(boxGeometry, material1);**

**var sphere = new THREE.Mesh(sphereGeometry, material1);**

Luego esas mismas primitivas se pasan a CSG como se muestra en el código.

**var boxCSG = THREE.CSG.fromMesh(cube);**

**var sphereCSG = THREE.CSG.fromMesh(sphere);**

Se hacen las respectivas operaciones entre las primitivas transformadas y el resultado se convierte nuevamente a Mesh.

**var result = boxCSG.intersect(sphereCSG);**

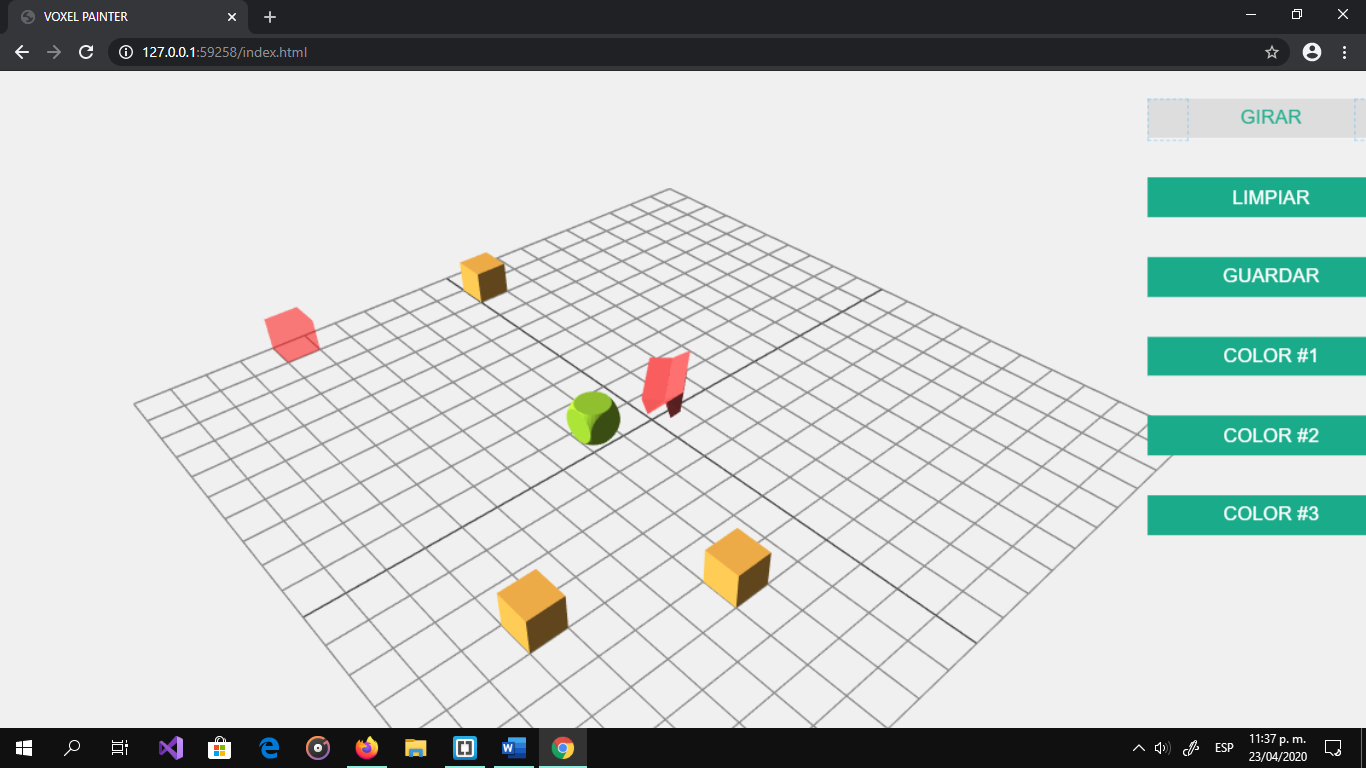
**cube = THREE.CSG.toMesh(result);**

**cube.position.set(75, 25, 30);**

**cube.material = material1;**

**scene.add(cube);**

Para realizar el avión de papel, se hizo uso de las spline para formar la curva que va a recorrer el extrude, para esto se realizó un gráfico en XY para dibujar la curvatura y obtener los puntos necesarios antes de pasar los datos al programa pero la línea no forma en sí sola la, por eso mismo esto se utilizó un extrude para definir las características, tales como el biselado o si la curva se cierra, en este caso si, ya que de lo contrario solo se mostrara una figura recta.



Img3.Figura con forma de avión de papel usando Spline y Extrude.

A continuación, se mostrara el código que permite su creación:

Se crea una curva cerrada Spline y se insertan los puntos que va a seguir la curva por medio de vectores de 3 dimensiones, después se crean las configuraciones del extrude que seguirá la curva

**var closedSpline = new THREE.CatmullRomCurve3**

**[**

**new THREE.Vector3(-25, 14, 13),**

**new THREE.Vector3(25, 50, -13),**

**]**

**);**

**closedSpline.curveType = 'catmullrom';**

**closedSpline.closed = true;**

**var extrudeSettings = {**

**steps: 100,**

**bevelEnabled: false,**

**bevelOffset: 100,**

**extrudePath: closedSpline**

**};**

Posteriormente se pasa a crear una arreglo que puntos el cual permite la creación de la superficie generada.

**var pts = [],**

**count = 2;**

**for (var i = 0; i < count; i++) {**

**var l = 30;**

**var a = 3 \* i / count \* Math.PI;**

**pts.push(new THREE.Vector2(Math.cos(a) \* l, Math.sin(a) \* l));**

**for (var j = 0; j < 3; j++) {**

**pts.push(new THREE.Vector2(0, j - 0.2, 0));**

**}**

**}**

**var shape = new THREE.Shape(pts);**

**var geometry2 = new THREE.ExtrudeBufferGeometry(shape, extrudeSettings);**

**var material2 = new THREE.MeshLambertMaterial({**

**color: 0xe25454,**

**metalness: 0.5,**

**roughness: 0.1**

**});**

**var F3 = new THREE.Mesh(geometry2, material2);**

**F3.position.set(0, 80, 0);**

**F3.rotation.x = 30;**

**scene.add(F3);**

Una vez tenemos las figuras ubicadas en el escenario, pasamos con cambiar el color del voxel que interactúa con la grilla; para esto se usaron tres colores en formato hexadecimal, estos colores son los que adquirida el voxel siempre y cuando se pulse su botón correspondiente, una vez esto está listo se crea la interacción de la siguiente manera:

**var color = document.getElementById("color").addEventListener('click',**

**function() {**

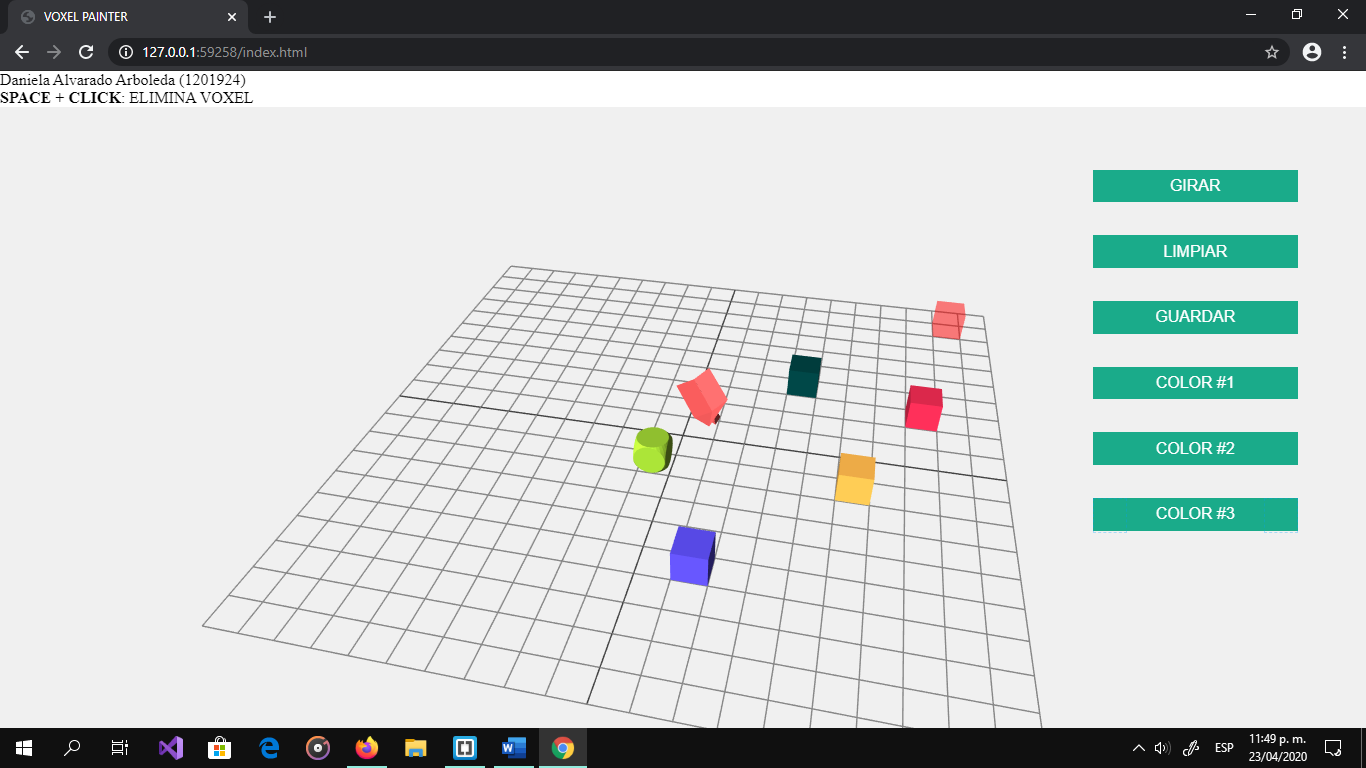
**cubeMaterial = new THREE.MeshLambertMaterial({**

**color: (Color correspondiente)**

**});**

**});**

Se repitió el mismo formato para los otros dos colores restantes y su resultado es el siguiente:



Img4. Voxel con diferentes colores.

Para el último botón la idea era que se guardara una captura del canvas en formato png que este se descargara automáticamente en otra ventana

**function save() {**

**var w = window.open('', '');**

**w.document.title = "Screenshot Canvas";**

**var img = new Image();**

**img.src = renderer.domElement.toDataURL();**

**w.document.body.appendChild(img);**

**var a = document.createElement('a');**

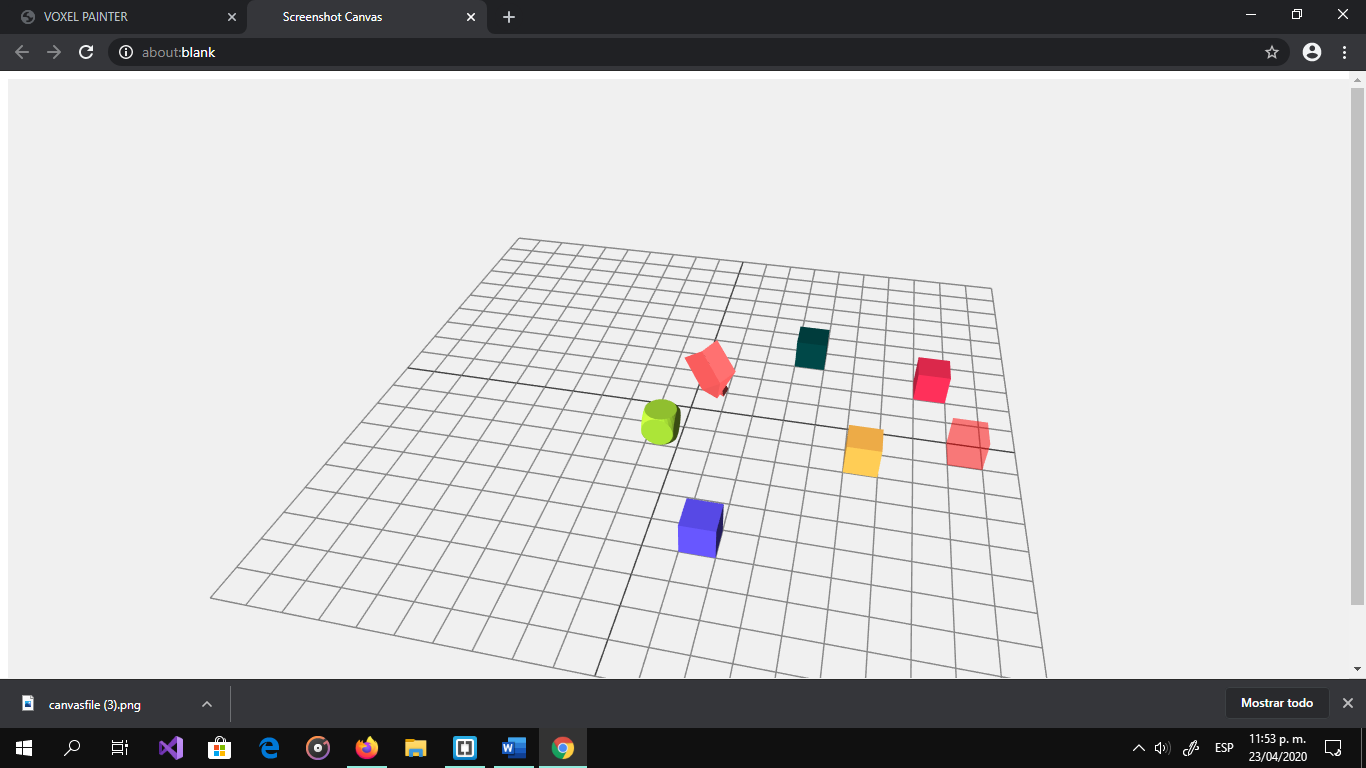
**a.href = renderer.domElement.toDataURL().replace("image/png", "image/octet-stream");**

**a.download = 'canvasfile.png'**

**a.click();**

**}**

De tal manera que:



Img5. Screenshot y descarga del canvas.

VI.CONCLUSIONES

A pesar de implementar otras funcionas de las que nos disponía el ejemplo voxel\_painter, tales como el cambio de color, guardar capturas y girar la cámara, otras tantas como que se pintaran los objetos, tales como la figura CSG o el Extrude como un voxel, no lograron concretarse.

No obstante, se implementaron los conceptos vistos en clase

1. [↑](#footnote-ref-1)