

Primer Examen Parcial

Implementación de la estructura multidimensional OcTree

Integrantes: Mita Yagua Lesly Yaneth
Pinto Medina Brian
Ccari Quispe José
Gonza Condori, Gabriel
Flores Herrera, Jefferson
Profesor: MSc. Vicente Machaca Arceda

Fecha de entrega: 15 de octubre de 2020
Arequipa, Perú

Índice de Contenidos

1. Introducción	1
1.1. Presentación y objetivos	1
1.2. Estructura del documento	1
2. Implementación	2
2.1. Estructura de OcTree	2
2.1.1. Clases y Funciones	3
2.1.2. Visualización del OcTree	4
2.2. Renderización de Gráficos 3D	5
2.2.1. Shader	5
2.2.2. Mesh	5
2.2.3. Engine	5
2.3. Función Principal	6
2.4. Funcionalidades	6
3. Evaluación y Pruebas	8
3.1. Query (Búsqueda)	8
3.2. Query (Búsqueda) dinámica	9
3.3. Inserción Dinámica y aleatoria de puntos	10
3.4. Transparencia	11
3.5. Capacidad de Datos	12
4. Conclusión	13
Anexo A. Código fuente	14
Anexo B. Capturas de pantalla	36
Anexo C. Repositorios	41

Índice de Figuras

1. Herramientas utilizadas	2
2. Estructura multidimensional OcTree	2
3. Visualización de OcTree	4
4. Visualización de OcTree con 8 puntos	4
5. OcTree con 10 puntos	8
6. OcTree con 100 puntos	8
7. Query inicial con 77 puntos dentro del cubo query	9
8. Query con un punto	9
9. Inserción de puntos dinámicamente	10
10. Inserción de puntos aleatoriamente	10
11. Captura antes de aplicada la transparencia	11

12.	Captura después de aplicada la transparencia	11
13.	Prueba con 500 puntos.	12
14.	Prueba con 5000 puntos	12
B.1.	Captura de Pantalla de la ejecucion de main.html	36
B.2.	Captura de Pantalla de main.js	36
B.3.	Captura de Pantalla de engine.js	37
B.4.	Captura de Pantalla de main.html	37
B.5.	Captura de Pantalla de mesh.js	38
B.6.	Captura de Pantalla de octree.js	38
B.7.	Captura de Pantalla de render.js	39
B.8.	Captura de Pantalla de shader.js	39
B.9.	Captura de Pantalla de testing.js	40

Índice de Códigos

A.1.	main.html	14
A.2.	main.js	14
A.3.	octree.js	14
A.4.	shader.js	20
A.5.	mesh.js	21
A.6.	engine.js	22
A.7.	render.js	28
A.8.	testing.html	32

1. Introducción

1.1. Presentación y objetivos

Se implemento la estructura de datos espacial OcTree con una visualización de la estructura en tres dimensiones. Para ello se utilizo el lenguaje Javascript y la visualización de la estructura se utilizo WebGL basado en OpenGL que define una API para la renderización de gráficos en 3D dentro de cualquier navegador web.

1.2. Estructura del documento

El presente documento está dividido en una serie de secciones que conforman el proceso de desarrollo del proyecto, detalladas a continuación:

- **Implementación:** Se detalla el todo el desarrollo de la implementación, su estructura y las funcionalidades que se agregaron tanto para el OcTree como para la visualización de la estructura en tres dimensiones.
- **Evaluación y Pruebas:** Se detalla los resultados de pruebas en el OcTree luego de la implementación con diferentes datos y simulaciones para la comprobación del correcto funcionamiento mediante una serie de pruebas.
- **Conclusión:** Se detallan las observaciones sobre la estructura multidimensional OcTree, en función a la implementación y capacidades de este.

2. Implementación

Para la implementación como lenguaje de programación se utilizó Javascript para la implementación de la estructura y para su visualización en 3D se utilizó WebGL , esto debido a que es muy práctico en el sentido de que no es necesario descargar plugins ni paquetes de dependencias , sino el navegador por defecto lo soporta .



(a) Javascript

(b) WebGL

Figura 1: Herramientas utilizadas

2.1. Estructura de OcTree

OcTree es una estructura de datos de árbol en la que cada nodo interno puede tener como máximo 8 hijos. Al igual que el árbol binario que divide el espacio en dos segmentos, OcTree divide el espacio en un máximo de ocho partes, lo que se denomina octanos. Se utiliza para almacenar el punto 3-D que ocupa una gran cantidad de espacio. Si todo el nodo interno del OcTree contiene exactamente 8 hijos, se llama OcTree completo. Si S es el número de puntos en cada dimensión, entonces el número de nodos que se forman en OcTree viene dado por esta fórmula $(S^3 - 1)/7$

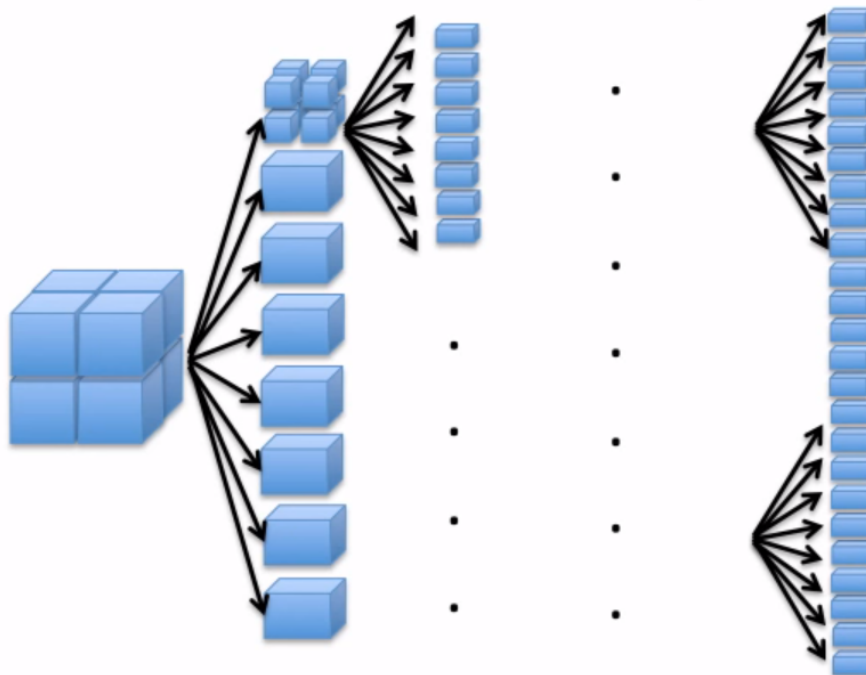


Figura 2: Estructura multidimensional OcTree

2.1.1. Clases y Funciones

El código de la **ocTree.js** se encuentra en el Anexo A. como **Código A.3** y su captura de pantalla en el Anexo B. como **Figura B.6**

Nuestra estructura Octree esta basada en la anterior realizada en clases QuadTree por lo mismo contiene los mismos métodos , a diferencia que ahora es una estructura tridimensional.

Implementamos las funciones:

- **numberToBinaryString:** Método que convierte un número a un String binario , esta función es utilizada en la inicialización del cubo.
- **pad** Función utilizada por numberToBinaryString para convertir un número a un String binario

Seguidamente implementamos las siguientes clases:

- **Color** Define los colores a utilizar.
- **Point:** Define un punto con las coordenadas x,y,z .
- **Cube** Posee los siguientes métodos:
 - **init:** Método que inicializa el cubo.
 - **getMinCoords:** Método que devuelve las coordenadas (x,y,z) mínimas de un cubo.
 - **getMaxCoords:** Método que devuelve las coordenadas (x,y,z) máximas de un cubo.
 - **contains:** Método que recibe un punto y devuelve True si el punto esta contenido en el cubo y False si no está contenido.
 - **intersects:** Método que recibe un cubo y devuelve verdadero si existe una intersección con el otro cubo y falso si no hay intersección.
 - **move:** Método que realiza un movimiento de un cubo en una dirección determinada.
- **OcTree** Posee los siguientes métodos:
 - **subdivide:** Método que subdivide un cubo en ocho partes iguales éstas son: topLeft-Front , topRightFront, bottomRightFront , bottomLeftFront , topLeftBack ,topRight-Back , bottomRightBack , bottomLeftBack y a la vez cambia el estado inicial del cubo de dividido a True .
 - **insert:** Método inserta un determinado punto para ello verifica si el punto insertado está dentro del cubo y que aún tenga la capacidad de albergar uno más , si el cubo excede la capacidad , este usa el método subdividir y se parte en otras 8 partes.
 - **query:** Método que tiene como parámetros un cubo y un array de puntos ,simplemente agrega al array todos los puntos contenidos en el cubo.

Una vez implementada las clases base , ahora si comenzamos con la estructura Octree como tal:
Aquí es donde se implementan los métodos: Subdivide, Insert y Query

2.1.2. Visualización del OcTree

Estructura multidimensional OcTree



Figura 3: Visualización de OcTree

Inicialmente se inserto 8 puntos dentro de la estructura OcTree, como podemos ver en la consola se muestra el primer punto que está contenido en el query, más abajo los 8 puntos repartidos en los diferentes cuadrantes.

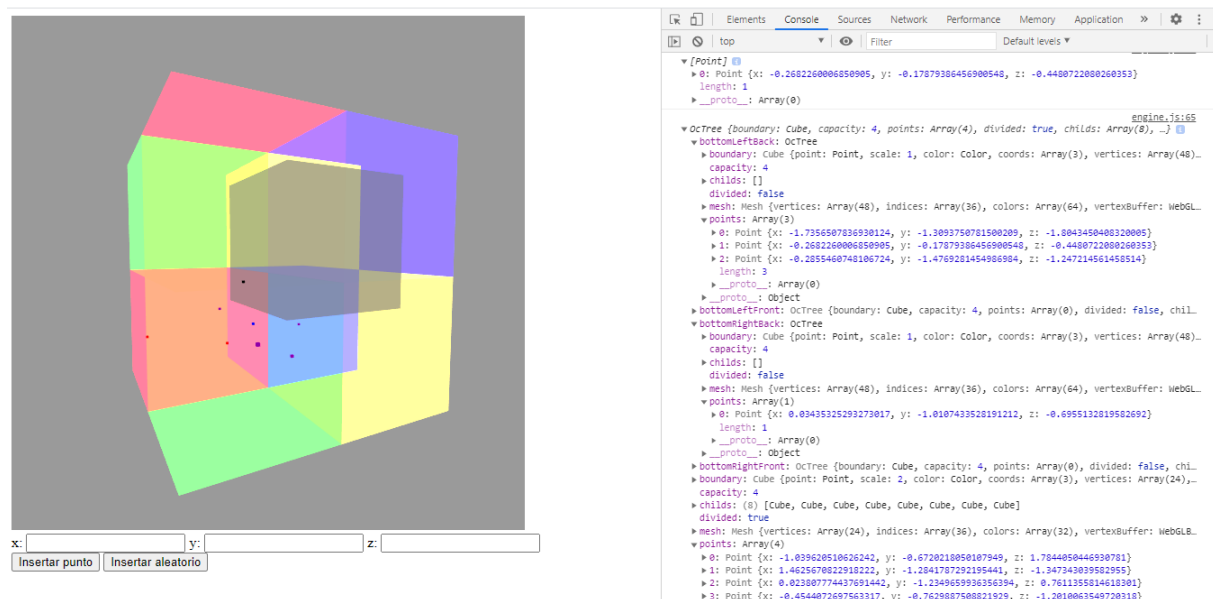


Figura 4: Visualización de OcTree con 8 puntos

Nota: en la sección de pruebas veremos más detalles de las funcionalidades.

2.2. Renderizacion de Gráficos 3D

WebGL está basado en OpenGL y proporciona una API para gráficos 3D. Se utiliza el elemento canvas HTML5 al que se accede mediante interfaces Document Object Model (DOM) que permite el acceso dinámico a través de la programación para acceder, añadir y cambiar dinámicamente contenido estructurado

2.2.1. Shader

El código de la **shader.js** se encuentra en el Anexo A. como **Código A.4** y su captura de pantalla en el Anexo B. como **Figura B.8**. Un Shader es un programa definido por el usuario diseñado para ejecutarse en alguna etapa de un procesador gráfico. Para la implementación se utilizó:

- **Vertex Shader** es la programable Shader etapa en el canal de renderizado que maneja el procesamiento de vértices individuales. Los sombreadores de vértices se alimentan de datos de atributos de vértice, como se especifica desde un objeto de matriz de vértices mediante un comando de dibujo. Un sombreador de vértices recibe un único vértice del flujo de vértices y genera un único vértice en el flujo de vértices de salida.
- **Fragment Shader** es la etapa de Shader que procesará un Fragmento generado por la Rasterización en un conjunto de colores y un valor de profundidad único. Para cada muestra de píxeles cubiertos por una primitiva, se genera un "fragmento". Cada fragmento tiene una posición de espacio de ventana, algunos otros valores y contiene todos los valores de salida por vértice interpolados de la última etapa de procesamiento de vértices.

2.2.2. Mesh

El código de la **mesh.js** se encuentra en el Anexo A. como **Código A.5** y su captura de pantalla en el Anexo B. como **Figura B.5**. Es un tipo de geometría 3D general del lado del cliente compuesto por vértices con atributos. Los vértices incluyen la posición geográfica, las normales que afectan la iluminación, sombreado y las coordenadas uv que se pueden usar para mapear imágenes. Se implementaron los siguientes métodos:

- **initBuffers** Inicializa los buffers en GPU para el envío del shader
- **associateShadersBuffer** Asocia un Shader al buffer correspondiente, tanto para la posición como para su color.

2.2.3. Engine

El código de la **engine.js** se encuentra en el Anexo A. como **Código A.6** y su captura de pantalla en el Anexo B. como **Figura B.3**. Este archivo es el motor para la renderización de gráficos 3D de la estructura OcTree, el cual posee los siguientes métodos que se detallarán a continuación.

- **init** Inicializa los métodos **initMatrix**, **initShaders**, **initModels** y **initUniforms**.
- **initMatrix** Inicializa las matrices necesarias para la renderización de la estructura OcTree con su posición y matrices de modelo y vista.
- **initShaders** Inicializa un Shader

- **initModels** Crea a la estructura multidimensional OcTree y a la representación del cubo, incluyendo los puntos dentro de él y una consulta de los mismos.
- **initUniforms** Inicializa las tres matrices Pmatrix, Vmatrix y Mmatrix.
- **render** Realiza un bucle que se encargara de dibujar constantemente, incluyendo las funcionalidades del mouse para mover la estructura. También se encarga de dibujar query (búsqueda) dentro del OcTree.

2.3. Función Principal

El código de la **main.js** se encuentra en el Anexo A. como **Código A.2** y su captura de pantalla en el Anexo B. como **Figura B.2**.

En este archivo se creará Canvas a partir de engine que definimos anteriormente, luego se inicializan los mesh y puntos también una constante de movimiento para la visualización en 3D. Por último, de inicializan los shaders, modelos, uniforms y matrices.

2.4. Funcionalidades

- **Inserción de Datos** Primero comprobamos si existe un nodo o no si existe un nodo y luego regresamos de lo contrario vamos de forma recursiva, comenzamos con el nodo raíz y lo marcamos como actual, luego encontramos el nodo hijo en el que podemos almacenar el punto. Si el nodo está vacío, reemplázelo con el nodo que queremos insertar y conviértalo en un nodo hoja, si el nodo es el nodo hoja, conviértalo en un nodo interno y, si es un nodo interno, vaya al nodo secundario. haga este proceso de forma recursiva hasta que no se encuentre un nodo vacío.
- **Query (Búsqueda)** Esta funcionalidad se utiliza para buscar el punto si existe el árbol o no, se comienza con el nodo raíz y busque de forma recursiva si se encuentra el nodo con el punto dado, luego devuelva verdadero si se encuentra un nodo vacío o un punto límite o un punto vacío y luego devuelva falso, si se encuentra un nodo interno, se va a ese nodo. Gráficamente consiste en crear un nuevo cubo de determinado tamaño y posición dentro del cubo padre y consultar cuántos puntos este almacena este.
- **Query (Búsqueda) dinámico** Esta funcionalidad permite mover el cubo Query en el espacio es decir, el cubo puede estar dentro y fuera de la estructura multidimensional OcTree haciendo posible la consulta de los puntos que están dentro de él. Para mover el query es posible presionando las siguientes teclas
 - **Tecla Q** Movimiento a la izquierda
 - **Tecla E** Movimiento a la derecha
 - **Tecla W** Movimiento hacia arriba
 - **Tecla S** Movimiento hacia abajo
- **Inserción de datos dinámicamente** El algoritmo de inserción dentro del OcTree es el mismo, sin embargo la diferencia es que para agregar un dato se podrá realizar de dos maneras por medio de las **coordenadas** del punto y también de manera **aleatoria**

- **Transparencia** Permite la visualización de solo los puntos y el cubo query dentro de OcTree, es decir la estructura multidimensional OcTree se volverá transparente. Esta funcionalidad nos ayuda a la visualización de los puntos que no perderán su color según al sector que pertenezca tanto en el OcTree como dentro de query.

3. Evaluación y Pruebas

3.1. Query (Búsqueda)

Se realizó una prueba de inserción dentro del OcTree con 10 puntos, como se puede observar en consola tenemos 2 puntos dentro del cubo query que está de color negro transparente.

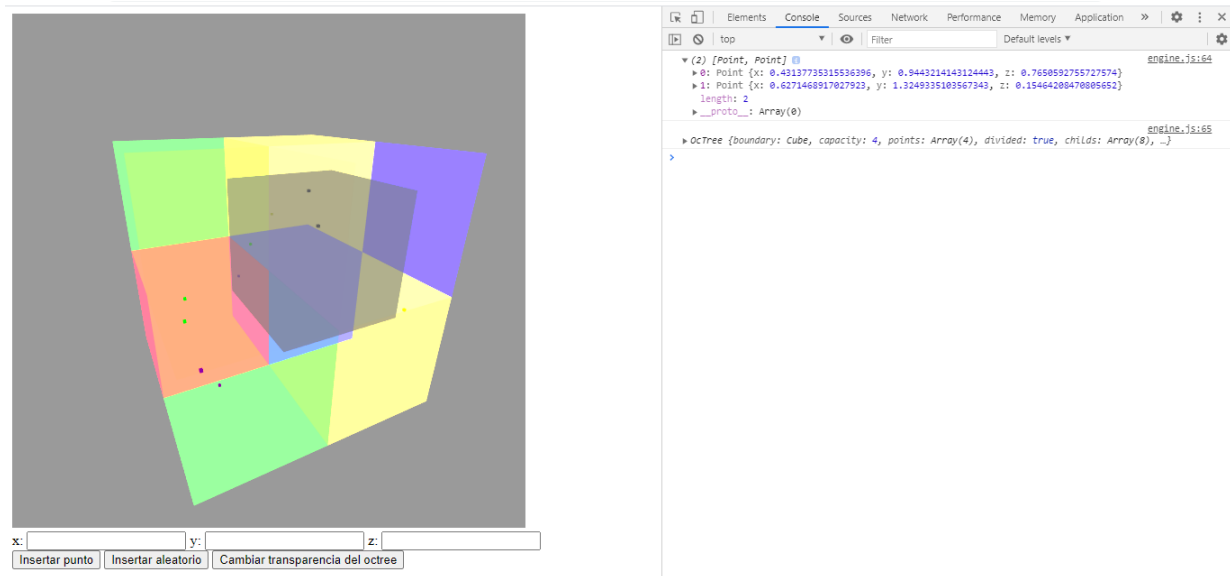


Figura 5: OcTree con 10 puntos

Se realizó una prueba de inserción dentro del OcTree con 100 puntos, de los cuales 11 están en el cubo query.

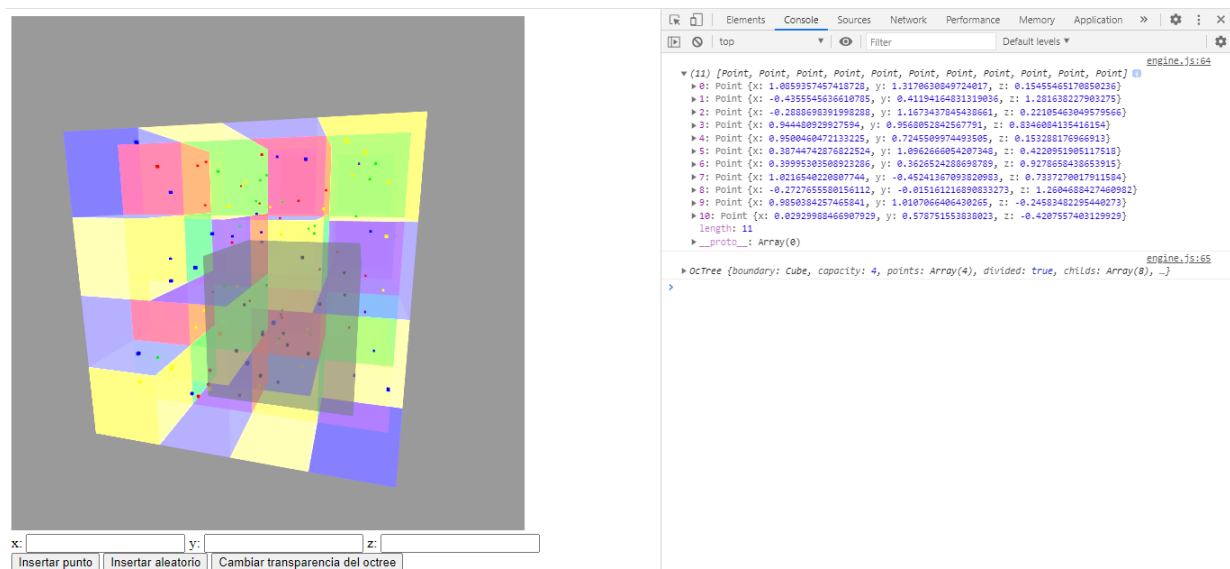


Figura 6: OcTree con 100 puntos

3.2. Query (Búsqueda) dinámica

Esta funcionalidad permite mover el cubo Query en el espacio, se realizaron pruebas de las posiciones del cubo que se verán en las siguientes Figuras.

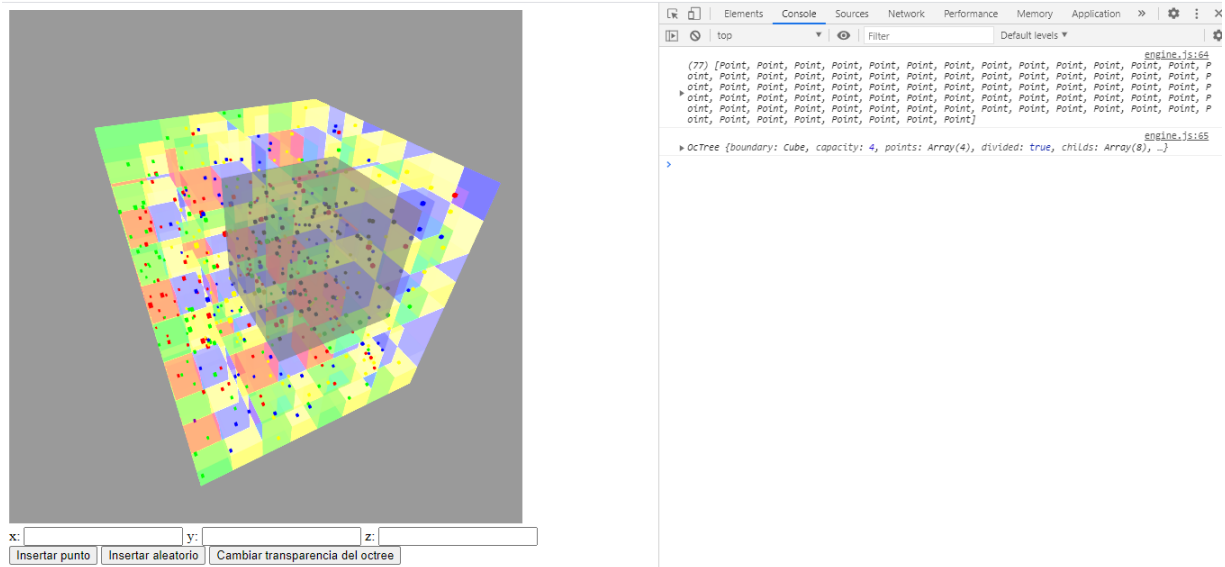


Figura 7: Query inicial con 77 puntos dentro del cubo query

Llevamos el cubo query a un posición donde solo incluya un punto, como se puede observar en la consola hay un punto dentro de la consulta

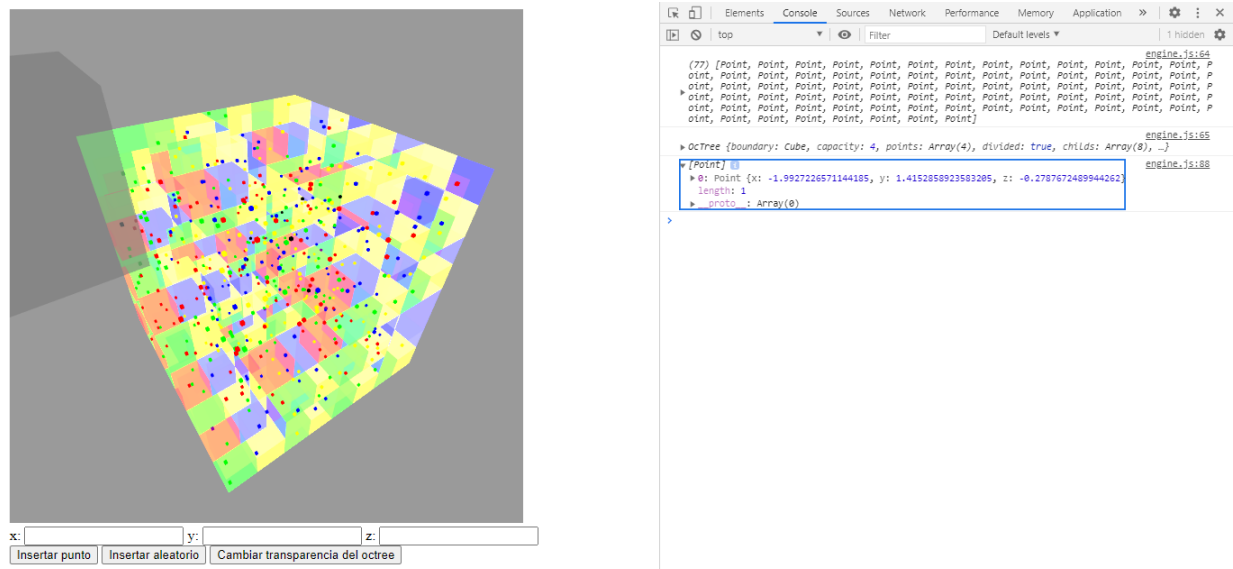


Figura 8: Query con un punto

3.3. Inserción Dinámica y aleatoria de puntos

También se implementó la inserción tanto dinámica como aleatoria de puntos, a continuación se vera dos ejemplo de ellos en las siguientes Figuras.

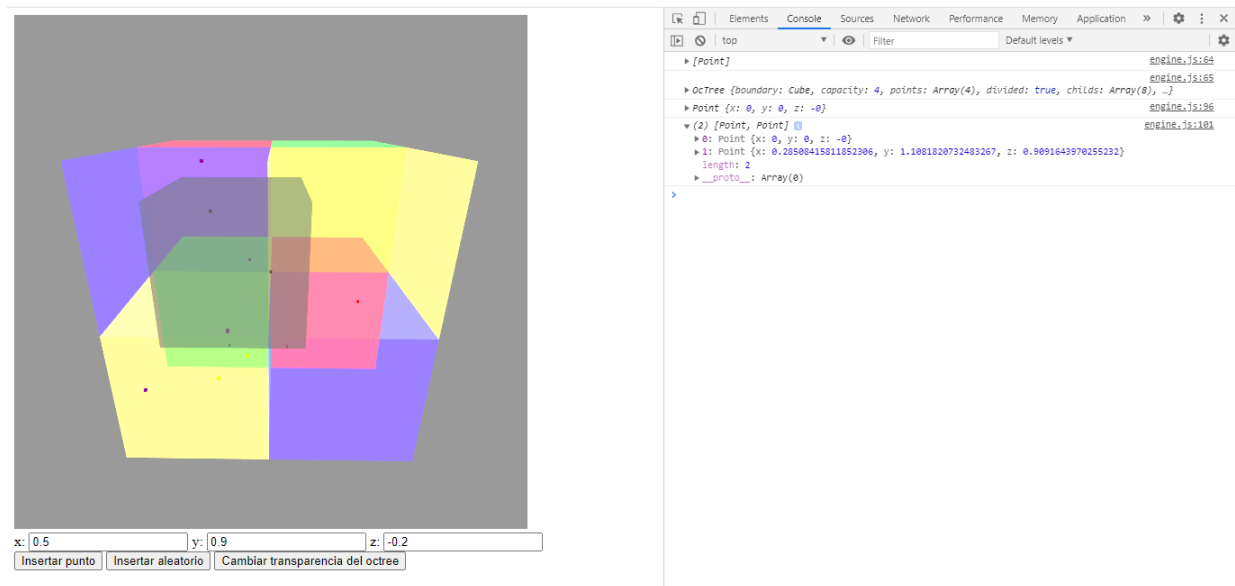


Figura 9: Inserción de puntos dinámicamente

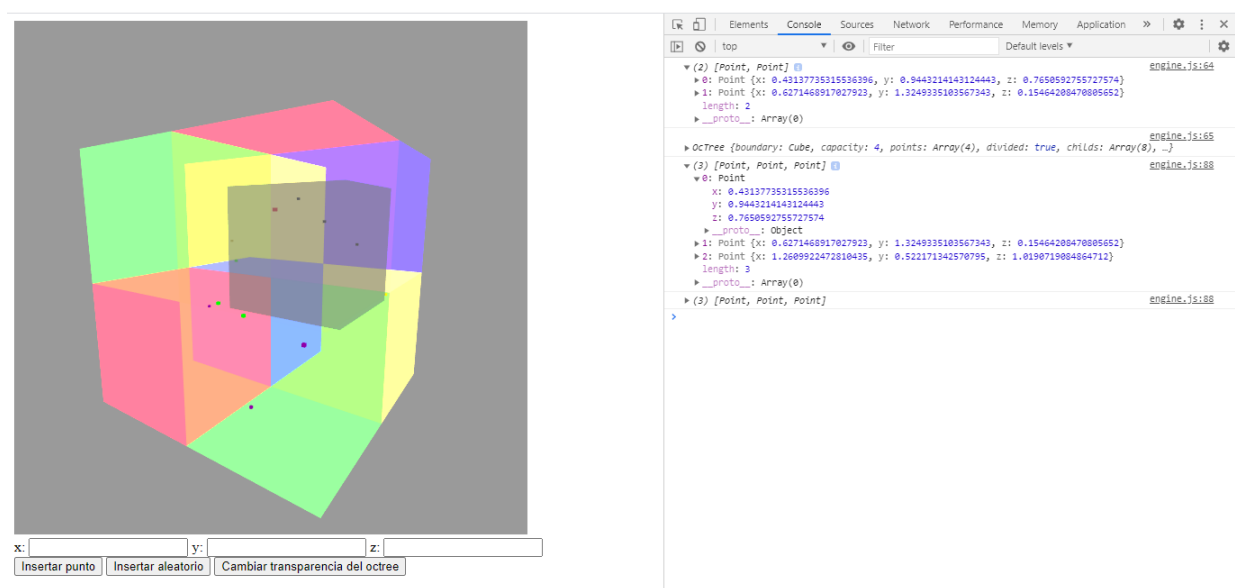


Figura 10: Inserción de puntos aleatoriamente

3.4. Transparencia

Esta es una funcionalidad que decidimos implementarla para poder visualizar mejor a los puntos esparcidos en el Octree.

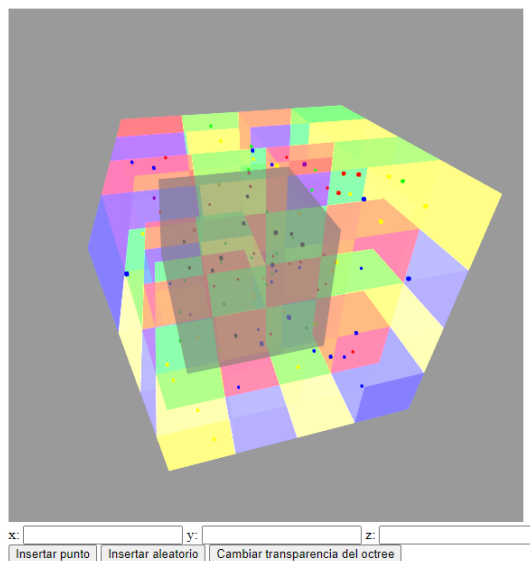


Figura 11: Captura antes de aplicada la transparencia

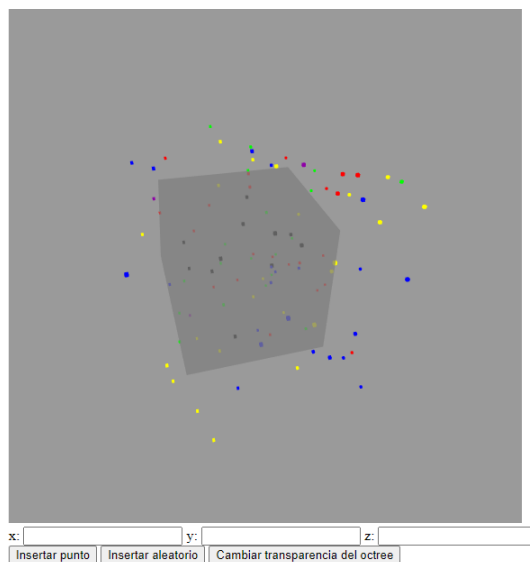


Figura 12: Captura después de aplicada la transparencia

3.5. Capacidad de Datos

Probaremos insertando tantos datos como sea posible , en primera instancia probaremos con 500 puntos y posteriormente con 5000 puntos:

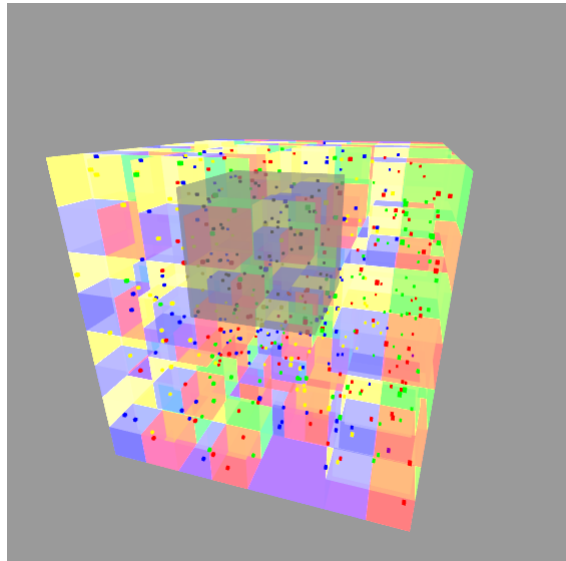


Figura 13: Prueba con 500 puntos.

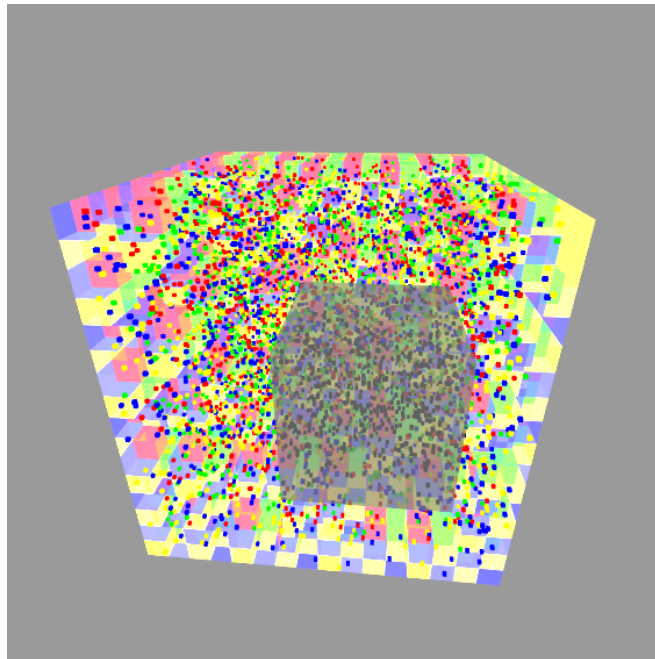


Figura 14: Prueba con 5000 puntos .

Nota: al insertar 5000 puntos ya notamos que demora bastante y le cuesta a nuestra pc , así que lo dejaremos aquí.

4. Conclusión

- También la idea de utilizar un OcTree para encontrar los nodos intransitables es para aprovechar la estructura y definición para hacer un menor número de comparaciones al evaluar si un cubo o nodo en la malla esta ocupado o colisiona con un objeto de la escena, los nodos del OcTree que envuelvan a los objetos de la escena serán los nodos no transitables.
- La importancia de las operaciones de conjunto dentro del OcTree ya que fue lo ocurrido con la función query que intersectaba elementos dentro del mismo espacio. Como operaciones geométricas que se hacen sobre un OcTree y calcular un nuevo OcTree como resultado de una traslación, rotación o escalonamiento como sucedió en la implementación de la visualización gráfica en 3D.
- Para finalizar podemos concluir que usamos la estructura de datos Octree para trabajar con cualquier tipo de datos tridimensionales entre los usos más comunes que se le pueden dar al Octree están:
 - Detección de colisiones eficiente en tres dimensiones
 - Determinación de cara oculta
 - Métodos no estructurados
 - Método de los elementos finitos
 - Octree de vóxeles escasos
 - Teorema de Bayes

Anexo A. Código fuente

Código A.1: main.html

```

1 <!doctype html>
2 <html>
3   <head>
4     <script src="shader.js" defer></script>
5     <script src="mesh.js" defer></script>
6     <script src="octree.js" defer></script>
7     <script src="engine.js" defer></script>
8     <script src="main.js" defer></script>
9   </head>
10  <body>
11    <canvas width = "570" height = "570" id = 'my_Canvas' tabindex='1'></canvas>
12    <div>
13      <label>x: </label>
14      <input type="text" id="coordX">
15      <label>y: </label>
16      <input type="text" id="coordY">
17      <label>z: </label>
18      <input type="text" id="coordZ">
19      <br>
20      <button type="submit" id="addPoint" onclick="engine.addPoint()">Insertar punto</
    ↪ button>
21      <button type="submit" id="addRandomPoint" onclick="engine.addRandomPoint()"
    ↪ >Insertar aleatorio</button>
22      <button type="submit" id="changeOctreeTransparency" onclick="
    ↪ engine.changeOctreeTransparency()">Cambiar transparencia del octree</button>
23    </div>
24  </body>
25 </html>

```

Código A.2: main.js

```

1 /*===== Creating a canvas =====*/
2 var engine = new Engine("my_Canvas");
3 var gl = engine.canvas.getContext('experimental-webgl');
4 /*===== Defining and storing the geometry =====*/
5 var meshes = [];
6 var meshPoints = [];
7 var queryPoints = [];
8 var CONSTANTE_MOVIMIENTO = 0.2;
9 engine.initShaders();
10 engine.initModels();
11 engine.initUniforms();
12 engine.initMatrix();
13 engine.updateEvents();
14 engine.render();

```

Código A.3: octree.js

```
1 function pad(n, width, z){
2   z = z || '0';
3   n = n + '';
4   return n.length >= width ? n : new Array(width - n.length + 1).join(z) + n;
5 }
6
7 function numberToBinaryString(n){
8   let numberInString = Number((n >>> 0).toString(2));
9   numberInString = pad(numberInString, 3);
10  return numberInString;
11 }
12
13 class Color{
14   constructor(r, g, b, a){
15     this.red = r;
16     this.green = g;
17     this.blue = b;
18     this.alfa = a;
19   }
20 }
21 class Point{
22   constructor(x, y, z){
23     this.x = x;
24     this.y = y;
25     this.z = z;
26   }
27 }
28 class Cube{
29   constructor(inPoint, scale, color){
30     this.point = inPoint;
31     this.scale = scale;
32     this.color = color;
33     this.coords = [[],[],[]];
34     this.vertices = [];
35     this.colors = [];
36     this.indices = [];
37   }
38
39   init(){
40     let x = this.point.x;
41     let y = this.point.y;
42     let z = this.point.z;
43
44     for(let i = 7; i >= 0; i--){
45       let pointInBinary = numberToBinaryString(i);
46       let tempPoint = [x,y,z];
47       let indexToChange = 2;
48       for(let j = 2 ; j >= 0; j--){
49         if(pointInBinary[j] == '1'){
50           tempPoint[indexToChange] -= this.scale;
51         }else if(pointInBinary[j] == '0')
```

```
52         tempPoint[indexToChange] += this.scale;
53         indexToChange--;
54     }
55
56     let newPoint = new Point(tempPoint[0], tempPoint[1], tempPoint[2]);
57     this.coords[0].push(newPoint.x);
58     this.coords[1].push(newPoint.y);
59     this.coords[2].push(newPoint.z);
60
61     this.vertices.push(newPoint.x);
62     this.vertices.push(newPoint.y);
63     this.vertices.push(newPoint.z);
64 }
65
66 this.indices = [
67     0,4,6, 0,6,2,
68     1,5,7, 1,7,3,
69     0,2,3, 0,3,1,
70     4,6,7, 4,7,5,
71     0,1,5, 0,5,4,
72     2,3,7, 2,7,6
73 ];
74
75 for(let i = 0; i < 8; i++){
76     this.colors.push(this.color.red);
77     this.colors.push(this.color.green);
78     this.colors.push(this.color.blue);
79     this.colors.push(this.color.alfa);
80 }
81
82 }
83
84 getMinCoords() {
85     let minCoordX = Math.min.apply(null,this.coords[0]);
86     let minCoordY = Math.min.apply(null,this.coords[1]);
87     let minCoordZ = Math.min.apply(null,this.coords[2]);
88     return new Point(minCoordX, minCoordY, minCoordZ);
89 }
90
91 getMaxCoords() {
92     let maxCoordX = Math.max.apply(null,this.coords[0]);
93     let maxCoordY = Math.max.apply(null,this.coords[1]);
94     let maxCoordZ = Math.max.apply(null,this.coords[2]);
95     return new Point(maxCoordX, maxCoordY, maxCoordZ);
96 }
97
98 contains( point ) {
99     let minCoords = this.getMinCoords();
100     let maxCoords = this.getMaxCoords();
101
102     if(minCoords.x <= point.x && point.x <= maxCoords.x &&
103         minCoords.y <= point.y && point.y <= maxCoords.y &&
```

```
104     minCoords.z <= point.z && point.z <= maxCoords.z )
105     return true;
106   else
107     return false ;
108 }
109
110 intersects ( range ) {
111   let minCoordsOwn = this.getMinCoords();
112   let maxCoordsOwn = this.getMaxCoords();
113   let minCoordsRange = range.getMinCoords();
114   let maxCoordsRange = range.getMaxCoords();
115
116   if (minCoordsOwn.x <= maxCoordsRange.x && maxCoordsOwn.x >= minCoordsRange.x &&
117       minCoordsOwn.y <= maxCoordsRange.y && maxCoordsOwn.y >= minCoordsRange.y &&
118       minCoordsOwn.z <= maxCoordsRange.z && maxCoordsOwn.z >= minCoordsRange.z )
119     return true;
120   else
121     return false ;
122 }
123
124 move( direccion ){
125   for (let i = 0; i < this.coords[0].length; i++){
126     this.coords[0][i] += direccion.x * CONSTANTE_MOVIMIENTO;
127     this.coords[1][i] += direccion.y * CONSTANTE_MOVIMIENTO;
128     this.coords[2][i] += direccion.z * CONSTANTE_MOVIMIENTO;
129   }
130   //TODO add mesh
131 }
132 }
133
134 class OcTree {
135   constructor( boundary, capacity ) {
136     this.boundary = boundary;
137     this.boundary.init();
138     this.capacity = capacity;
139     this.points = [];
140     this.divided = false;
141     this.childs = [];
142     this.meshModels = []; //puntos en forma de cubos
143
144     this.mesh = new Mesh(boundary.point, boundary.vertices, boundary.colors, boundary.indices);
145     this.mesh.initBuffers();
146
147     meshes.push(this.mesh);
148   }
149
150   subdivide() {
151     let x = this.boundary.point.x;
152     let y = this.boundary.point.y;
153     let z = this.boundary.point.z;
154     let dimension = this.boundary.scale;
155
```

```

156     var colorRed = new Color(1, 0, 0, 0.5);
157     var colorGreen = new Color(0, 1, 0, 0.5);
158     var colorBlue = new Color(0, 0, 1, 0.5);
159     var colorYellow = new Color(1, 1, 0, 0.5);
160
161     let otTopLeftFront = new Cube( new Point( x - dimension / 2, y + dimension / 2, z +
↪ dimension / 2 ), this.boundary.scale / 2, colorRed);
162     otTopLeftFront.init();
163     this . childs . push(otTopLeftFront);
164
165     let otTopRightFront = new Cube( new Point( x + dimension / 2, y + dimension / 2, z +
↪ dimension / 2 ), this.boundary.scale / 2, colorBlue);
166     otTopRightFront.init();
167     this . childs . push(otTopRightFront);
168
169     let otBottomRightFront = new Cube( new Point( x + dimension / 2, y - dimension / 2, z +
↪ dimension / 2 ), this.boundary.scale / 2, colorYellow);
170     otBottomRightFront.init();
171     this . childs . push(otBottomRightFront);
172
173     let otBottomLeftFront = new Cube( new Point( x - dimension / 2, y - dimension / 2, z +
↪ dimension / 2 ), this.boundary.scale / 2, colorGreen);
174     otBottomLeftFront.init();
175     this . childs . push(otBottomLeftFront);
176
177     let otTopLeftBack = new Cube( new Point( x - dimension / 2, y + dimension / 2, z -
↪ dimension / 2 ), this.boundary.scale / 2, colorGreen);
178     otTopLeftBack.init();
179     this . childs . push(otTopLeftBack);
180
181     let otTopRightBack = new Cube( new Point( x + dimension / 2, y + dimension / 2, z -
↪ dimension / 2 ), this.boundary.scale / 2, colorYellow);
182     otTopRightBack.init();
183     this . childs . push(otTopRightBack);
184
185     let otBottomRightBack = new Cube( new Point( x + dimension / 2, y - dimension / 2, z -
↪ dimension / 2 ), this.boundary.scale / 2, colorBlue);
186     otBottomRightBack.init();
187     this . childs . push(otBottomRightBack);
188
189     let otBottomLeftBack = new Cube( new Point( x - dimension / 2, y - dimension / 2, z -
↪ dimension / 2 ), this.boundary.scale / 2, colorRed);
190     otBottomLeftBack.init();
191     this . childs . push(otBottomLeftBack);
192
193     this . topLeftFront = new OcTree( otTopLeftFront, this.capacity );
194     this . topRightFront = new OcTree( otTopRightFront, this.capacity );
195     this . bottomRightFront = new OcTree( otBottomRightFront, this.capacity );
196     this . bottomLeftFront = new OcTree( otBottomLeftFront, this.capacity );
197     this . topLeftBack = new OcTree( otTopLeftBack, this.capacity );
198     this . topRightBack = new OcTree( otTopRightBack, this.capacity );
199     this . bottomRightBack = new OcTree( otBottomRightBack, this.capacity );

```

```

200     this.bottomLeftBack = new OcTree( otBottomLeftBack, this.capacity );
201     this.divided = true;
202 }
203
204 insert(point) {
205     if (!this.boundary.contains(point)) {
206         return false;
207     }
208
209     if (this.points.length < this.capacity) {
210         var color = new Color(this.boundary.color.red, this.boundary.color.green, this.boundary.
↪ color.blue, 1 );
211         var randomPoint = new Cube(point, 0.02, color);
212         randomPoint.init();
213
214         var meshPoint = new Mesh(point, randomPoint.vertices, randomPoint.colors, randomPoint.
↪ indices );
215         meshPoint.initBuffers();
216         meshPoints.push(meshPoint);
217         this.points.push(point);
218         this.meshModels.push(meshPoint);
219
220         return true;
221     } else {
222         if (!this.divided)
223             this.subdivide();
224
225         if (this.topLeftFront.insert(point))
226             return true;
227         else if (this.topRightFront.insert(point))
228             return true;
229         else if (this.bottomRightFront.insert(point))
230             return true;
231         else if (this.bottomLeftFront.insert(point))
232             return true;
233         else if (this.topLeftBack.insert(point))
234             return true;
235         else if (this.topRightBack.insert(point))
236             return true;
237         else if (this.bottomRightBack.insert(point))
238             return true;
239         else if (this.bottomLeftBack.insert(point))
240             return true;
241     }
242 }
243
244 query(range, found) {
245
246     if (!this.boundary.intersects(range)) {
247         return;
248     } else {
249         for ( let mesh of this.meshModels) {

```

```

250         if( range.contains(mesh.point)) {
251             var color = new Color(range.color.red, range.color.green, range.color.blue, 1);
252             mesh.pintar(color);
253             found.push(mesh.point);
254         }else{
255             var parentColor = new Color(this.boundary.color.red, this.boundary.color.green, this
↪ .boundary.color.blue, 1);
256             mesh.pintar(parentColor);
257         }
258     }
259     if( this.divided ) {
260         this.topLeftFront.query(range, found);
261         this.topRightFront.query(range, found);
262         this.bottomRightFront.query(range, found);
263         this.bottomLeftFront.query(range, found);
264         this.topLeftBack.query(range, found);
265         this.topRightBack.query(range, found);
266         this.bottomRightBack.query(range, found);
267         this.bottomLeftBack.query(range, found);
268     }
269 }
270 }
271 }

```

Código A.4: shader.js

```

1  class VertexShaderSource{
2      constructor(){
3          this.source = 'attribute vec3 position;' +
4              'uniform mat4 Pmatrix;' +
5              'uniform mat4 Vmatrix;' +
6              'uniform mat4 Mmatrix;' +
7              'attribute vec4 color;' + //the color of the point
8              'varying vec4 vColor;' +
9              'void main(void) { ' + //pre-built function
10                 'gl_Position = Pmatrix*Vmatrix*Mmatrix*vec4(position, 1.);' +
11                 'vColor = color;' +
12                 '}' ;
13      }
14  }
15
16  class FragmentShaderSource{
17      constructor(){
18          this.source =
19              'precision mediump float;' +
20              'varying vec4 vColor;' +
21              'void main(void) {' +
22                 'gl_FragColor = vec4(vColor);' +
23                 '}' ;
24      }
25  }
26

```

```

27 class Shader{
28     constructor(){
29         this.loadShaderSource();
30         this.loadShader();
31         this.linkProgram();
32         this.useProgram();
33     }
34     loadShaderSource(){
35         var vertCodeSource = new VertexShaderSource();
36         this.vertCode = vertCodeSource.source;
37         var fragCodeSource = new FragmentShaderSource();
38         this.fragCode = fragCodeSource.source;
39     }
40     loadShader(){
41         this.vertShader = gl.createShader(gl.VERTEX_SHADER);
42         gl.shaderSource(this.vertShader, this.vertCode);
43         gl.compileShader(this.vertShader);
44
45         this.fragShader = gl.createShader(gl.FRAGMENT_SHADER);
46         gl.shaderSource(this.fragShader, this.fragCode);
47         gl.compileShader(this.fragShader);
48     }
49     linkProgram(){
50         this.shaderProgramId = gl.createProgram();
51         gl.attachShader(this.shaderProgramId, this.vertShader);
52         gl.attachShader(this.shaderProgramId, this.fragShader);
53         gl.linkProgram(this.shaderProgramId);
54         gl.useProgram(this.shaderProgramId);
55     }
56     useProgram(){
57         gl.useProgram(this.shaderProgramId);
58     }
59     quitProgram(){
60         gl.useProgram(0);
61     }
62 }
63 }

```

Código A.5: mesh.js

```

1 class Mesh{
2     constructor(point, vertices, colors, indices){
3         this.point = point;
4         this.vertices = vertices;
5         this.indices = indices;
6         this.colors = colors;
7     }
8     move(direccion){
9         for(let i = 0; i < this.vertices.length; i+=3){
10             this.vertices[i] += direccion.x * CONSTANTE_MOVIMIENTO;
11             this.vertices[i + 1] += direccion.y * CONSTANTE_MOVIMIENTO;
12             this.vertices[i + 2] += direccion.z * CONSTANTE_MOVIMIENTO;

```



```
13     }
14     gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexBuffer);
15     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.vertices), gl.STATIC_DRAW);
16     gl.bindBuffer(gl.ARRAY_BUFFER, null);
17 }
18 pintar(color){
19     for(let i = 0; i < this.colors.length; i+=4){
20         this.colors[i] = color.red;
21         this.colors[i + 1] = color.green;
22         this.colors[i + 2] = color.blue;
23         this.colors[i + 3] = color.alfa;
24     }
25     gl.bindBuffer(gl.ARRAY_BUFFER, this.colorBuffer);
26     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.colors), gl.STATIC_DRAW);
27     gl.bindBuffer(gl.ARRAY_BUFFER, null);
28 }
29 initBuffers(){
30     this.vertexBuffer = gl.createBuffer();
31     gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexBuffer);
32     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.vertices), gl.STATIC_DRAW);
33     gl.bindBuffer(gl.ARRAY_BUFFER, null);
34
35     this.colorBuffer = gl.createBuffer();
36     gl.bindBuffer(gl.ARRAY_BUFFER, this.colorBuffer);
37     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(this.colors), gl.STATIC_DRAW);
38     gl.bindBuffer(gl.ARRAY_BUFFER, null);
39
40
41     this.indexBuffer = gl.createBuffer();
42     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.indexBuffer);
43     gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(this.indices), gl.
↪ STATIC_DRAW);
44     gl.bindBuffer(gl.ARRAY_BUFFER, null);
45
46 }
47
48 associateShadersBuffer(shaderProgram){
49     // Position
50     gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexBuffer);
51     var position = gl.getAttribLocation(shaderProgram, "position");
52     gl.vertexAttribPointer(position, 3, gl.FLOAT, false, 0, 0);
53     gl.enableVertexAttribArray(position);
54     // Color
55     gl.bindBuffer(gl.ARRAY_BUFFER, this.colorBuffer);
56     var color = gl.getAttribLocation(shaderProgram, "color");
57     gl.vertexAttribPointer(color, 4, gl.FLOAT, false, 0, 0);
58     gl.enableVertexAttribArray(color);
59 }
60 }
```

Código A.6: engine.js

```
1 class Engine{
2
3   constructor(canvasId){
4     this.canvas = document.getElementById(canvasId);
5     this.THETA = 0,
6     this.PHI = 0;
7     this.time_old = 0;
8     this.drag = false;
9     this.AMORTIZATION = 0.95;
10    this.old_x;
11    this.old_y;
12    this.dX = 0;
13    this.dY = 0;
14  }
15  init(){
16    this.initMatrix();
17    this.initShaders();
18    this.initModels();
19    this.initUniforms();
20  }
21  initMatrix(){
22    this.getProjection = function(angle, a, zMin, zMax) {
23      var ang = Math.tan((angle*.5)*Math.PI/180);//angle*.5
24      return [
25        0.5/ang, 0 , 0, 0,
26        0, 0.5*a/ang, 0, 0,
27        0, 0, -(zMax+zMin)/(zMax-zMin), -1,
28        0, 0, (-2*zMax*zMin)/(zMax-zMin), 0
29      ];
30    }
31
32    this.projectionMatrix = this.getProjection(40, this.canvas.width/this.canvas.height, 1,
↪ 100);
33    this.modelMatrix = [ 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 ];
34    this.viewMatrix = [ 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 ];
35
36    this.viewMatrix[14] = this.viewMatrix[14]-6;
37
38  }
39
40  initShaders(){
41    this.shaderProgram = new Shader();
42  }
43
44  initModels(){
45    var point = new Point(0, 0, 0);
46    var colorPurple = new Color(0.56, 0, 0.66, 0.5);
47    this.cube = new Cube(point, 2, colorPurple);
48    this.ot = new OcTree(this.cube, 4);
49
50    for( let i = 0; i < 0; ++i ) {
51      var p = new Point(this.randomNumber(-2, 2), this.randomNumber(-2,2), this.
```

```

↪ randomNumber(-2,2));
52     this.ot.insert(p);
53 }
54
55 var colorQuery = new Color(0,0,0,0.5);
56 var pQuery = new Point(0.5,0.5,0.5);
57 this.consulta = new Cube(pQuery, 1, colorQuery);
58 this.consulta.init();
59 this.meshQuery = new Mesh(pQuery, this.consulta.vertices, this.consulta.colors, this.consulta
↪ .indices);
60 this.meshQuery.initBuffers();
61
62 var puntos = [];
63 this.ot.query(this.consulta, puntos);
64 }
65
66 changeOctreeTransparency(){
67     if(meshes[0].colors[3] === 0.5){
68         for(let i = 0; i < meshes.length; ++i){
69             let colorTransparente = new Color(meshes[i].colors[0], meshes[i].colors[1], meshes[i].
↪ colors[2], 0);
70             meshes[i].pintar(colorTransparente);
71         }
72     }else{
73         for(let i = 0; i < meshes.length; ++i){
74             let colorTransparente = new Color(meshes[i].colors[0], meshes[i].colors[1], meshes[i].
↪ colors[2], 0.5);
75             meshes[i].pintar(colorTransparente);
76         }
77     }
78 }
79 }
80
81 addRandomPoint() {
82     var point = new Point(this.randomNumber(-2, 2), this.randomNumber(-2,2), this.
↪ randomNumber(-2,2));
83     var puntos = [];
84     this.ot.insert(point);
85     this.ot.query(this.consulta, puntos);
86     console.log(puntos);
87 }
88
89 addPoint() {
90     var x = parseInt(document.getElementById('coordX').value);
91     var y = parseInt(document.getElementById('coordY').value);
92     var z = parseInt(document.getElementById('coordZ').value);
93     var point = new Point(x, y, z);
94     console.log(point);
95     var puntos = [];
96     this.ot.insert(point);
97     var cont = 0;
98     this.ot.query(this.consulta, puntos);

```

```
99     console.log(puntos);
100 }
101
102 randomNumber(minimo, maximo){
103     return (Math.random() * (maximo - minimo) + minimo );
104 }
105
106 initUniforms(){
107     this.projectionMatrixUniformLocation = gl.getUniformLocation(this.shaderProgram.
↵ shaderProgramId, "Pmatrix");
108     this.viewMatrixUniformLocation = gl.getUniformLocation(this.shaderProgram.
↵ shaderProgramId, "Vmatrix");
109     this.modelMatrixUniformLocation = gl.getUniformLocation(this.shaderProgram.
↵ shaderProgramId, "Mmatrix");
110 }
111
112 updateMouseEvent(){
113
114     this.mouseDown = function(e) {
115         engine.drag = true;
116         engine.old_x = e.pageX, engine.old_y = e.pageY;
117         e.preventDefault();
118         return false ;
119     };
120
121     this.mouseUp = function(e){
122         engine.drag = false ;
123     };
124
125     this.mouseMove = function(e) {
126         if (!engine.drag) return false ;
127         engine.dX = (e.pageX - engine.old_x)*2*Math.PI/engine.canvas.width,
128         engine.dY = (e.pageY - engine.old_y)*2*Math.PI/engine.canvas.height;
129         engine.THETA += engine.dX;
130         engine.PHI += engine.dY;
131         engine.old_x = e.pageX, engine.old_y = e.pageY;
132         e.preventDefault();
133     };
134
135     this.canvas.addEventListener("mousedown", this.mouseDown, false);
136     this.canvas.addEventListener("mouseup", this.mouseUp, false);
137     this.canvas.addEventListener("mouseout", this.mouseUp, false);
138     this.canvas.addEventListener("mousemove", this.mouseMove, false);
139
140 }
141
142 updateModelMovementEvent(mesh, model){
143     this.keyDown = function(e){
144         let aKey = 'A', dKey = 'D', wKey = 'W', sKey = 'S', qKey = 'Q', eKey = 'E';
145         let direccion = new Point(0,0,0);
146         var puntos=[];
147
```

```
148     switch(e.keyCode){
149         case aKey.charCodeAt():
150             direccion.x = -1;
151             break;
152         case dKey.charCodeAt():
153             direccion.x = 1;
154             break;
155         case wKey.charCodeAt():
156             direccion.y = 1;
157             break;
158         case sKey.charCodeAt():
159             direccion.y = -1;
160             break;
161         case qKey.charCodeAt():
162             direccion.z = 1;
163             break;
164         case eKey.charCodeAt():
165             direccion.z = -1;
166             break;
167         default :
168             break;
169     }
170     mesh.move(direccion);
171     model.move(direccion);
172     var cont = 0;
173     engine.ot.query(engine.consulta, puntos);
174 }
175 document.addEventListener("keydown", this.keyDown);
176 }
177
178 updateEvents(){
179     this.updateMouseEvent();
180     this.updateModelMovementEvent(this.meshQuery, this.consulta);
181 }
182
183 render(){
184     this.animate = function(time) {
185         if (!engine.drag) {
186             engine.dX *= engine.AMORTIZATION, engine.dY *= engine.AMORTIZATION;
187             engine.THETA += engine.dX, engine.PHI += engine.dY;
188         }
189
190         //set model matrix to I4
191
192         engine.modelMatrix[0] = 1, engine.modelMatrix[1] = 0, engine.modelMatrix[2] = 0,
193         engine.modelMatrix[3] = 0,
194
195         engine.modelMatrix[4] = 0, engine.modelMatrix[5] = 1, engine.modelMatrix[6] = 0,
196         engine.modelMatrix[7] = 0,
197
198         engine.modelMatrix[8] = 0, engine.modelMatrix[9] = 0, engine.modelMatrix[10] = 1,
199         engine.modelMatrix[11] = 0,
```

```
200     engine.modelMatrix[12] = 0, engine.modelMatrix[13] = 0, engine.modelMatrix[14] = 0,
201     engine.modelMatrix[15] = 1;
202
203
204     engine.rotateX = function(m, angle) {
205         var c = Math.cos(angle);
206         var s = Math.sin(angle);
207         var mv1 = m[1], mv5 = m[5], mv9 = m[9];
208
209         m[1] = m[1]*c-m[2]*s;
210         m[5] = m[5]*c-m[6]*s;
211         m[9] = m[9]*c-m[10]*s;
212
213         m[2] = m[2]*c+mv1*s;
214         m[6] = m[6]*c+mv5*s;
215         m[10] = m[10]*c+mv9*s;
216     }
217
218     engine.rotateY = function(m, angle) {
219         var c = Math.cos(angle);
220         var s = Math.sin(angle);
221         var mv0 = m[0], mv4 = m[4], mv8 = m[8];
222
223         m[0] = c*m[0]+s*m[2];
224         m[4] = c*m[4]+s*m[6];
225         m[8] = c*m[8]+s*m[10];
226
227         m[2] = c*m[2]-s*mv0;
228         m[6] = c*m[6]-s*mv4;
229         m[10] = c*m[10]-s*mv8;
230     }
231
232     engine.rotateY(engine.modelMatrix, engine.THETA);
233     engine.rotateX(engine.modelMatrix, engine.PHI);
234
235     engine.time_old = time;
236
237     // gl.depthFunc(gl.LEQUAL);
238
239     gl.clearColor(0.5, 0.5, 0.5, 0.0);
240     gl.disable(gl.DEPTH_TEST);
241     gl.depthFunc(gl.LEQUAL)
242     gl.depthMask(false);
243     gl.enable(gl.BLEND);
244     gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
245
246
247     gl.clearColor(0.5, 0.5, 0.5, 0.9);
248     gl.clearDepth(1.0);
249     gl.viewport(0.0, 0.0, engine.canvas.width, engine.canvas.height);
250     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
251
```

```

252         gl.uniformMatrix4fv(engine.projectionMatrixUniformLocation, false, engine.
253 ↪ projectionMatrix);
254         gl.uniformMatrix4fv(engine.viewMatrixUniformLocation, false, engine.viewMatrix);
255         gl.uniformMatrix4fv(engine.modelMatrixUniformLocation, false, engine.modelMatrix);
256
257         for(let i=0;i<meshes.length;i++){
258             meshes[i].associateShadersBuffer(engine.shaderProgram.shaderProgramId);
259             gl.drawElements(gl.TRIANGLES, meshes[i].indices.length, gl.UNSIGNED_SHORT,
260 ↪ 0);
261         }
262
263         for(let i=0;i<meshPoints.length;i++){
264             meshPoints[i].associateShadersBuffer(engine.shaderProgram.shaderProgramId);
265             gl.drawElements(gl.TRIANGLES, meshPoints[i].indices.length, gl.
266 ↪ UNSIGNED_SHORT, 0);
267         }
268
269         engine.meshQuery.associateShadersBuffer(engine.shaderProgram.shaderProgramId);
270         gl.drawElements(gl.TRIANGLES, engine.meshQuery.indices.length, gl.
271 ↪ UNSIGNED_SHORT, 0);
272
273         for(let i=0;i<queryPoints.length;i++){
274             queryPoints[i].associateShadersBuffer(engine.shaderProgram.shaderProgramId);
275             gl.drawElements(gl.TRIANGLES, queryPoints[i].indices.length, gl.
276 ↪ UNSIGNED_SHORT, 0);
277         }
278
279         window.requestAnimationFrame(engine.animate);
280     }
281     this.animate(0);
282 }
283 }

```

Código A.7: render.js

```

1  var cubeRotation = 0.0;
2
3  class Canvas{
4      constructor(htmlId){
5          let canvas = document.querySelector(htmlId);
6          this.gl = canvas.getContext('webgl') || canvas.getContext('experimental-webgl');
7      }
8      init(){
9          if(!this.gl){
10             alert('No se pudo inicializar WebGL');
11             return;
12         }
13         var shader = new Shader();
14         this.shaderProgram = shader.initShaderProgram(this.gl);
15         this.programInfo = {
16             program: this.shaderProgram,

```

```
17     attribLocations: {
18         vertexPosition: this.gl.getAttribLocation(this.shaderProgram, 'aVertexPosition'),
19         vertexColor: this.gl.getAttribLocation(this.shaderProgram, 'aVertexColor'),
20     },
21     uniformLocations: {
22         projectionMatrix: this.gl.getUniformLocation(this.shaderProgram, 'uProjectionMatrix'),
23         modelViewMatrix: this.gl.getUniformLocation(this.shaderProgram, 'uModelViewMatrix'),
24     }
25 };
26 this.buffers = this.initBuffers();
27 var then = 0;
28
29 this.render = function(now){
30     now *= 0.001;
31     const deltaTime = now - then;
32     then = now;
33     this.drawScene(deltaTime);
34     requestAnimationFrame(this.render);
35 }
36 requestAnimationFrame(this.render);
37 }
38
39 initBuffers(){
40     var positionBuffer = this.gl.createBuffer();
41     this.gl.bindBuffer(this.gl.ARRAY_BUFFER, positionBuffer);
42     //TODO add cubes positions
43     var positions = [
44         // Front face
45         -1.0, -1.0,  1.0,
46         1.0,  -1.0,  1.0,
47         1.0,  1.0,   1.0,
48         -1.0,  1.0,  1.0,
49
50         // Back face
51         -1.0, -1.0, -1.0,
52         -1.0,  1.0, -1.0,
53         1.0,  1.0, -1.0,
54         1.0, -1.0, -1.0,
55
56         // Top face
57         -1.0,  1.0, -1.0,
58         -1.0,  1.0,  1.0,
59         1.0,  1.0,  1.0,
60         1.0,  1.0, -1.0,
61
62         // Bottom face
63         -1.0, -1.0, -1.0,
64         1.0,  -1.0, -1.0,
65         1.0,  -1.0,  1.0,
66         -1.0, -1.0,  1.0,
67
68         // Right face
```



```
69         1.0, -1.0, -1.0,
70         1.0,  1.0, -1.0,
71         1.0,  1.0,  1.0,
72         1.0, -1.0,  1.0,
73
74         // Left face
75         -1.0, -1.0, -1.0,
76         -1.0, -1.0,  1.0,
77         -1.0,  1.0,  1.0,
78         -1.0,  1.0, -1.0,
79     ];
80     this.gl.bufferData(this.gl.ARRAY_BUFFER, new Float32Array(positions), this.gl.
↪ STATIC_DRAW);
81     //TODO add face colors
82     var faceColors = [
83         [1.0, 1.0, 1.0, 1.0],
84         [1.0, 0.0, 0.0, 1.0],
85         [0.0, 1.0, 0.0, 1.0],
86         [0.0, 0.0, 1.0, 1.0],
87         [1.0, 1.0, 0.0, 1.0],
88         [1.0, 0.0, 1.0, 1.0],
89     ];
90     var colors = [];
91
92     for(let i=0; i< faceColors.length; ++i){
93         let c = faceColors[i];
94         colors = colors.concat(c, c, c, c);
95     }
96
97     var colorBuffer = this.gl.createBuffer();
98     this.gl.bindBuffer(this.gl.ARRAY_BUFFER, colorBuffer);
99     this.gl.bufferData(this.gl.ARRAY_BUFFER, new Float32Array(colors), this.gl.
↪ STATIC_DRAW);
100
101     var indexBuffer = this.gl.createBuffer();
102     this.gl.bindBuffer(this.gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
103
104     //TODO add indices
105     var indices =[
106         0, 1, 2, 0, 2, 3,
107         4, 5, 6, 4, 6, 7,
108         8, 9, 10, 8, 10, 11,
109         12, 13, 14, 12, 14, 15,
110         16, 17, 18, 16, 18, 19,
111         20, 21, 22, 20, 22, 23,
112     ];
113
114     this.gl.bufferData(this.gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), this.gl.
↪ STATIC_DRAW);
115
116     return{
117         position: positionBuffer,
```

```
118     color: colorBuffer ,
119     indices: indexBuffer,
120   };
121 }
122
123 drawScene(deltaTime){
124   this.gl.clearColor(0.0, 0.0, 0.0, 1.0);
125   this.gl.clearDepth(1.0);
126   this.gl.enable(this.gl.DEPTH_TEST);
127   this.gl.depthFunc(this.gl.LEQUAL);
128
129   this.gl.clear(this.gl.COLOR_BUFFER_BIT | this.gl.DEPTH_BUFFER_BIT);
130
131   const fieldOfView = 45 * Math.PI / 180;
132   const aspect = this.gl.canvas.clientWidth / this.gl.canvas.clientHeight;
133   const zNear = 0.1;
134   const zFar = 100.0;
135   const projectionMatrix = mat4.create();
136
137   mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
138   const modelViewMatrix = mat4.create();
139
140   mat4.translate(modelViewMatrix, modelViewMatrix, [-0.0, 0.0, -6.0]);
141   mat4.rotate(modelViewMatrix, modelViewMatrix, cubeRotation, [0, 0, 1]);
142   mat4.rotate(modelViewMatrix, modelViewMatrix, cubeRotation * .7, [0, 1, 0]);
143   {
144     const numComponents = 3;
145     const type = this.gl.FLOAT;
146     const normalize = false;
147     const stride = 0;
148     const offset = 0;
149     this.gl.bindBuffer(this.gl.ARRAY_BUFFER, this.buffers.position);
150     this.gl.vertexAttribPointer(
151       this.programInfo.attribLocations.vertexPosition,
152       numComponents,
153       type,
154       normalize,
155       stride,
156       offset
157     );
158     this.gl.enableVertexAttribArray(this.programInfo.attribLocations.vertexPosition);
159   }
160   {
161     const numComponents = 4;
162     const type = this.gl.FLOAT;
163     const normalize = false;
164     const stride = 0;
165     const offset = 0;
166     this.gl.bindBuffer(this.gl.ARRAY_BUFFER, this.buffers.color);
167     this.gl.vertexAttribPointer(
168       this.programInfo.attribLocations.vertexColor,
169     numComponents,
```

```

170         type,
171         normalize,
172         stride ,
173         offset
174     );
175     this.gl.enableVertexAttribArray(this.programInfo.attribLocations.vertexColor);
176 }
177 this.gl.bindBuffer(this.gl.ELEMENT_ARRAY_BUFFER, this.buffers.indices);
178 this.useProgram(this.programInfo.program);
179
180 this.gl.uniformMatrix4fv(
181     this.programInfo.uniformLocations.projectionMatrix,
182     false ,
183     projectionMatrix);
184
185 this.gl.uniformMatrix4fv(
186     this.programInfo.uniformLocations.modelViewMatrix,
187     false ,
188     modelViewMatrix);
189 {
190     const vertexCount = 36;
191     const type = this.gl.UNSIGNED_SHORT;
192     const offset = 0;
193     this.gl.drawElements(this.gl.TRIANGLES, vertexCount, type, offset);
194 }
195
196 cubeRotation += deltaTime;
197 }
198 }

```

Código A.8: testing.html

```

1 <!doctype html>
2 <html>
3   <body>
4     <canvas width = "300" height = "300" id = "my_Canvas"></canvas>
5
6     <script>
7       /*===== Creating a canvas =====*/
8
9       var canvas = document.getElementById('my_Canvas');
10      var gl = canvas.getContext('experimental-webgl');
11
12      /*===== Defining and storing the geometry =====*/
13
14      var vertices = [
15        -0.7, -0.1, 0,
16        -0.3, 0.6, 0,
17        -0.3, -0.10, 0,
18        0.2, 0.6, 10,
19        0.3, -0.3, 10,
20        0.7, 0.6, 0

```

```
21     ]
22
23
24     var vertices2 = [
25         -0.9,-0.1,0,
26         -0.3,0.6,0,
27         -0.10,-0.3,10,
28         0.2,0.6,0,
29         0.3,-0.3,0,
30         0.7,0.6,0
31     ]
32
33     // Create an empty buffer object
34     var vertex_buffer = gl.createBuffer();
35
36     // Bind appropriate array buffer to it
37     gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
38
39     // Pass the vertex data to the buffer
40     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
41
42     // Unbind the buffer
43     gl.bindBuffer(gl.ARRAY_BUFFER, null);
44
45     var vertex_buffer2 = gl.createBuffer();
46
47     gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer2);
48
49     // Pass the vertex data to the buffer
50     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices2), gl.STATIC_DRAW);
51
52     // Unbind the buffer
53     gl.bindBuffer(gl.ARRAY_BUFFER, null);
54
55     /*===== Shaders =====*/
56
57     // Vertex shader source code
58     var vertCode =
59         'attribute vec3 coordinates;' +
60         'void main(void) {' +
61         '    gl_Position = vec4(coordinates, 1.0);' +
62         '}'
63
64     // Create a vertex shader object
65     var vertShader = gl.createShader(gl.VERTEX_SHADER);
66
67     // Attach vertex shader source code
68     gl.shaderSource(vertShader, vertCode);
69
70     // Compile the vertex shader
71     gl.compileShader(vertShader);
72
```

```
73 // Fragment shader source code
74 var fragCode =
75     'void main(void) {' +
76     '    gl_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +
77     '  }';
78
79 // Create fragment shader object
80 var fragShader = gl.createShader(gl.FRAGMENT_SHADER);
81
82 // Attach fragment shader source code
83 gl.shaderSource(fragShader, fragCode);
84
85 // Compile the fragment shader
86 gl.compileShader(fragShader);
87
88 // Create a shader program object to store
89 // the combined shader program
90 var shaderProgram = gl.createProgram();
91
92 // Attach a vertex shader
93 gl.attachShader(shaderProgram, vertShader);
94
95 // Attach a fragment shader
96 gl.attachShader(shaderProgram, fragShader);
97
98 // Link both the programs
99 gl.linkProgram(shaderProgram);
100
101 // Use the combined shader program object
102 gl.useProgram(shaderProgram);
103
104 /*===== Drawing the triangle =====*/
105
106 // Clear the canvas
107 gl.clearColor (0.5, 0.5, 0.5, 0.9);
108
109 // Enable the depth test
110 gl.enable(gl.DEPTH_TEST);
111
112 // Clear the color and depth buffer
113 gl.clear (gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
114
115 // Set the view port
116 gl.viewport (0,0, canvas.width, canvas.height);
117
118
119 /*===== Associating shaders to buffer objects =====*/
120
121 // Bind vertex buffer object
122 gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);
123
124 // Get the attribute location
```

```
125     var coord = gl.getAttribLocation(shaderProgram, "coordinates");
126
127     // Point an attribute to the currently bound VBO
128     gl.vertexAttribPointer(coord, 3, gl.FLOAT, false, 0, 0);
129
130     // Enable the attribute
131     gl.enableVertexAttribArray(coord);
132
133     // Draw the triangle
134     gl.drawArrays(gl.TRIANGLES, 0, 6);
135
136     // Bind vertex buffer object
137     gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer2);
138
139     // Get the attribute location
140     var coord2 = gl.getAttribLocation(shaderProgram, "coordinates");
141
142     // Point an attribute to the currently bound VBO
143     gl.vertexAttribPointer(coord2, 3, gl.FLOAT, false, 0, 0);
144
145     // Enable the attribute
146     gl.enableVertexAttribArray(coord2);
147
148     // Draw the triangle
149     gl.drawArrays(gl.TRIANGLES, 0, 6);
150
151     // POINTS, LINE_STRIP, LINE_LOOP, LINES,
152     // TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES
153     </script>
154 </body>
155 </html>
```

Anexo B. Capturas de pantalla

Estructura multidimensional OcTree



Figura B.1: Captura de Pantalla de la ejecucion de main.html

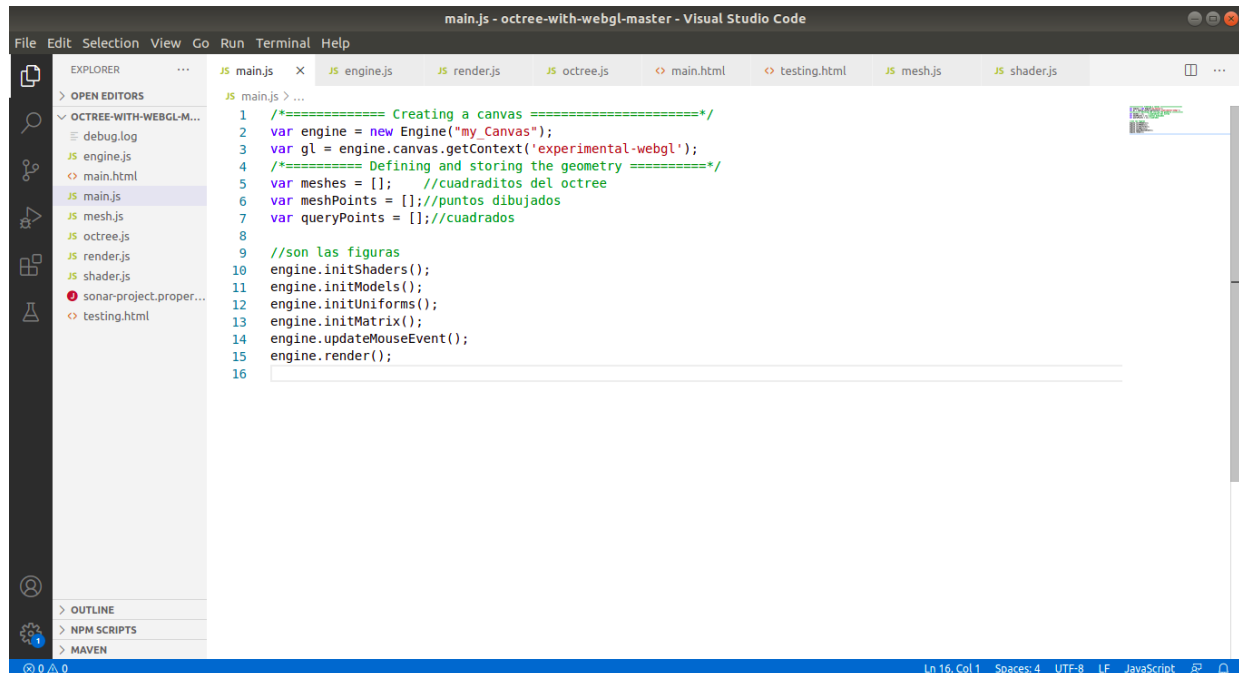


Figura B.2: Captura de Pantalla de main.js

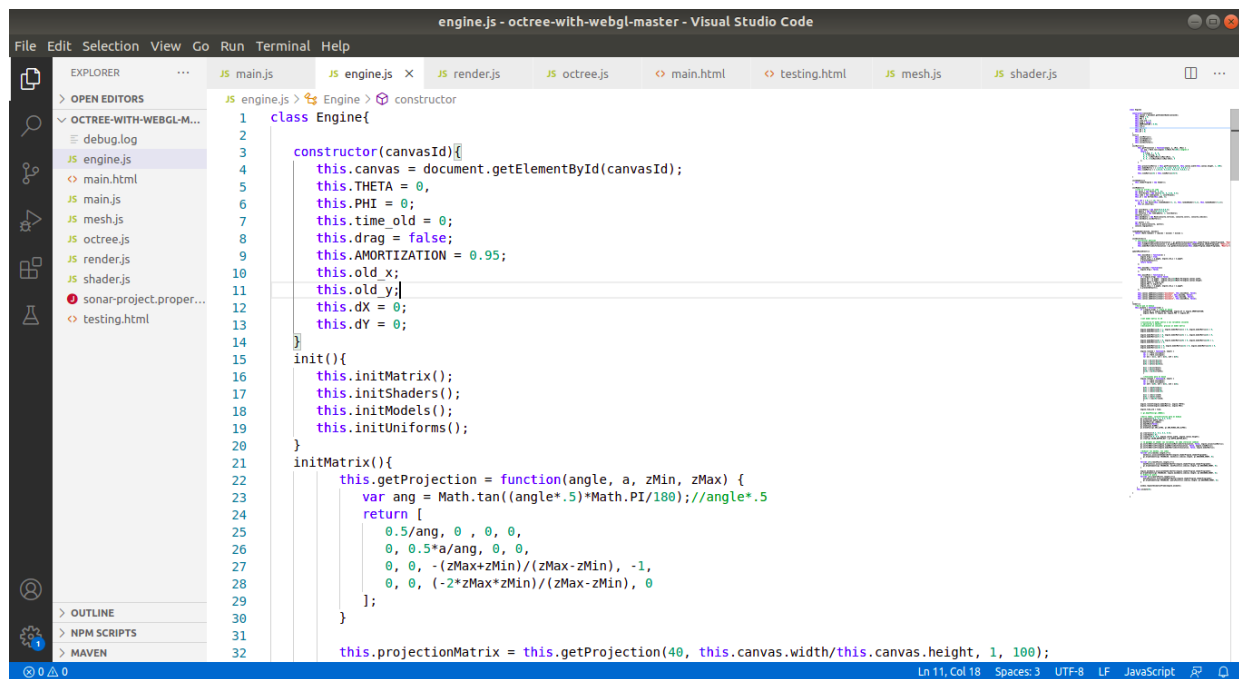


Figura B.3: Captura de Pantalla de engine.js

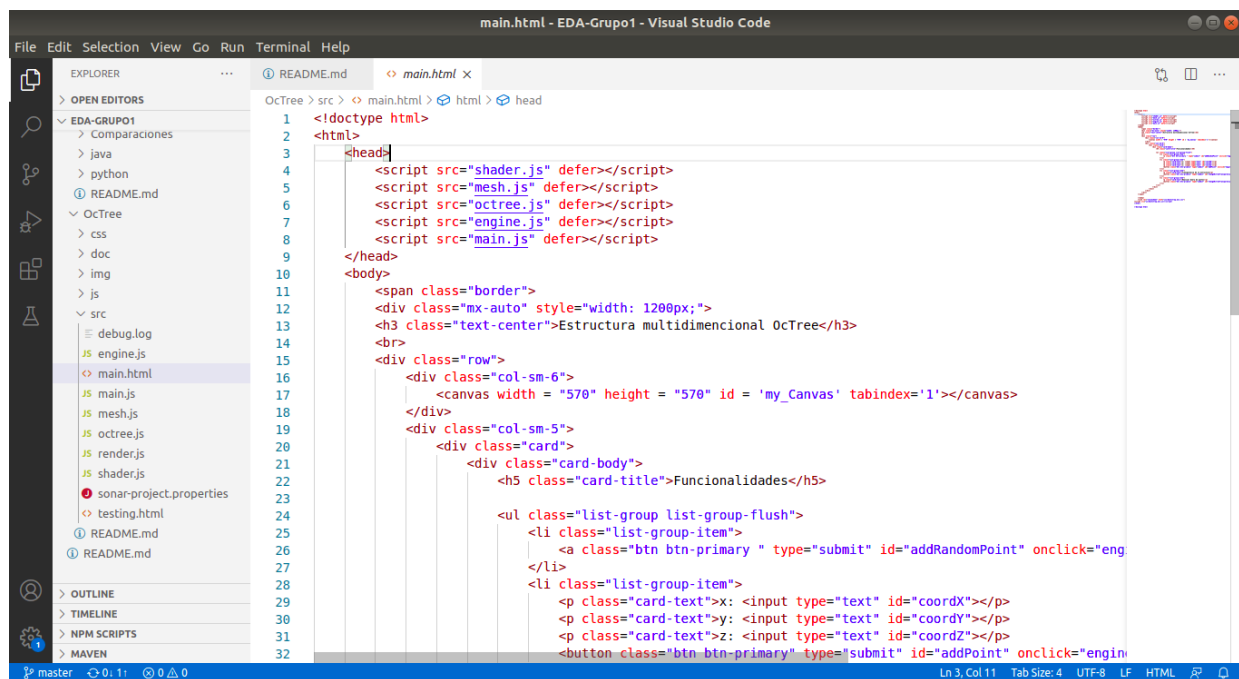


Figura B.4: Captura de Pantalla de main.html

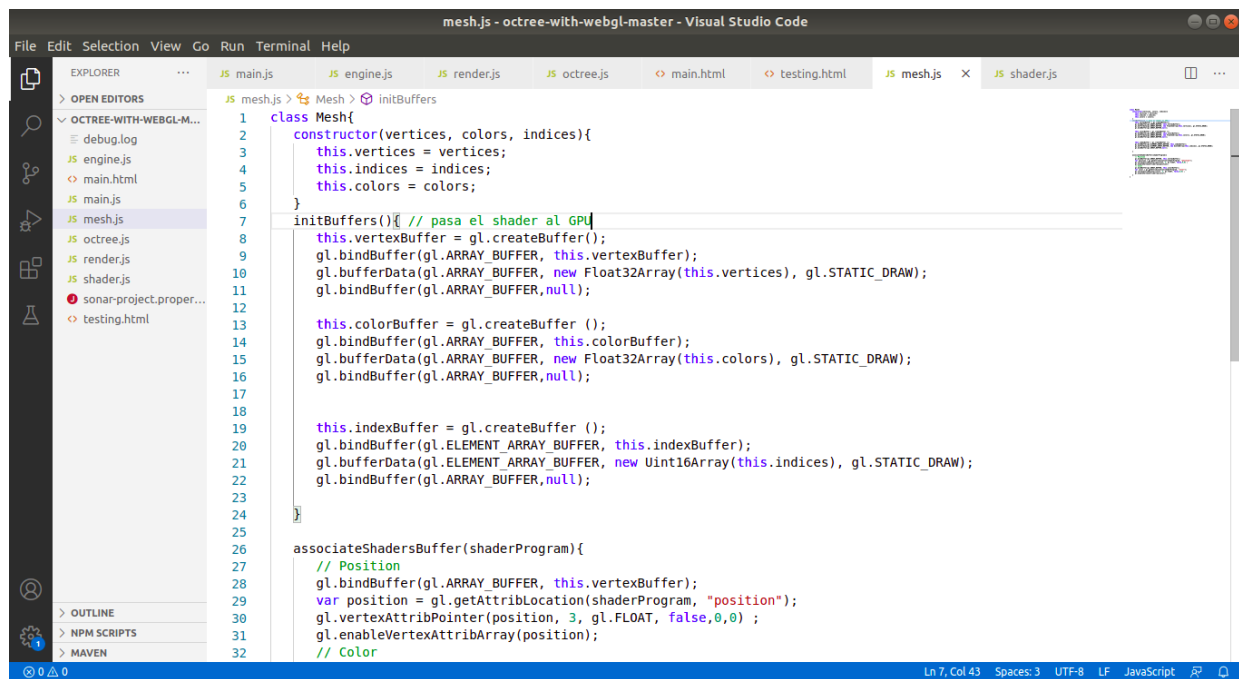


Figura B.5: Captura de Pantalla de mesh.js

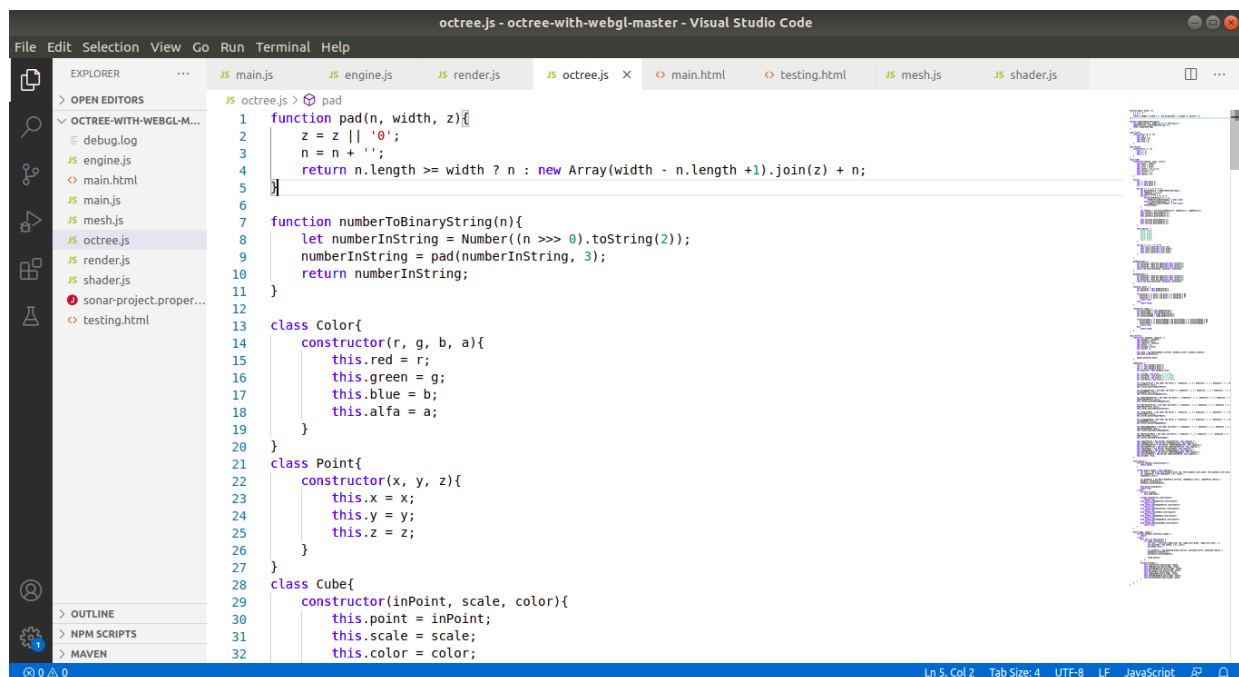
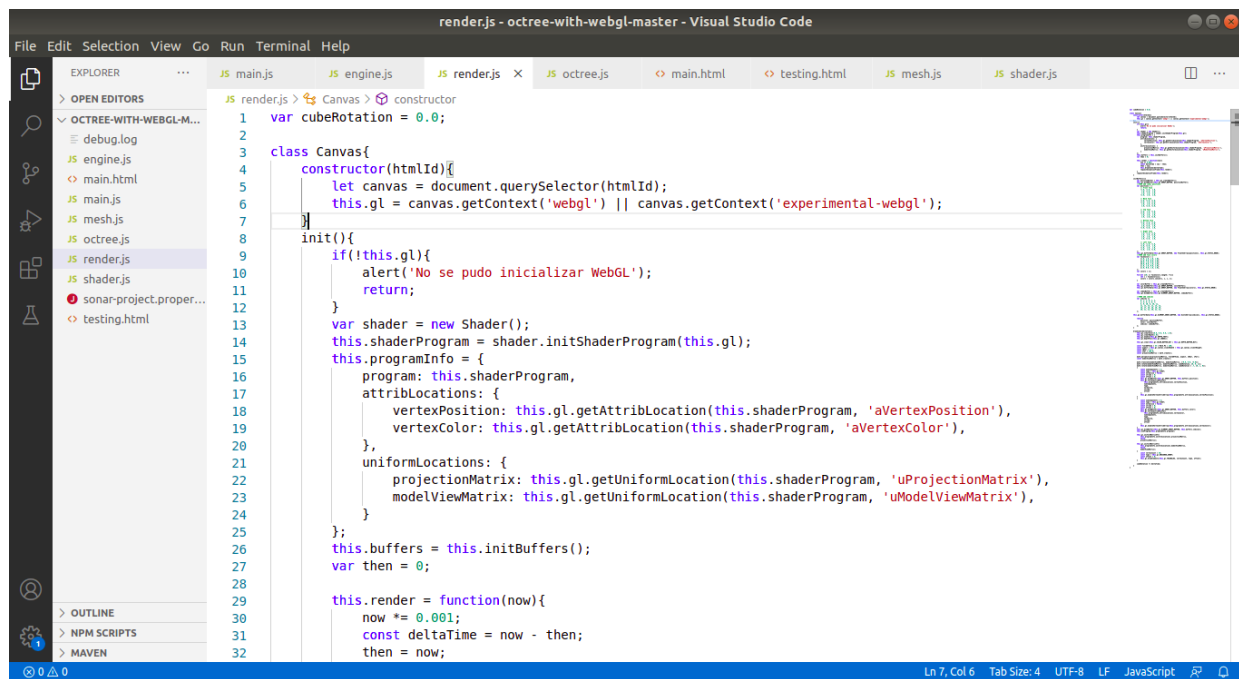


Figura B.6: Captura de Pantalla de octree.js

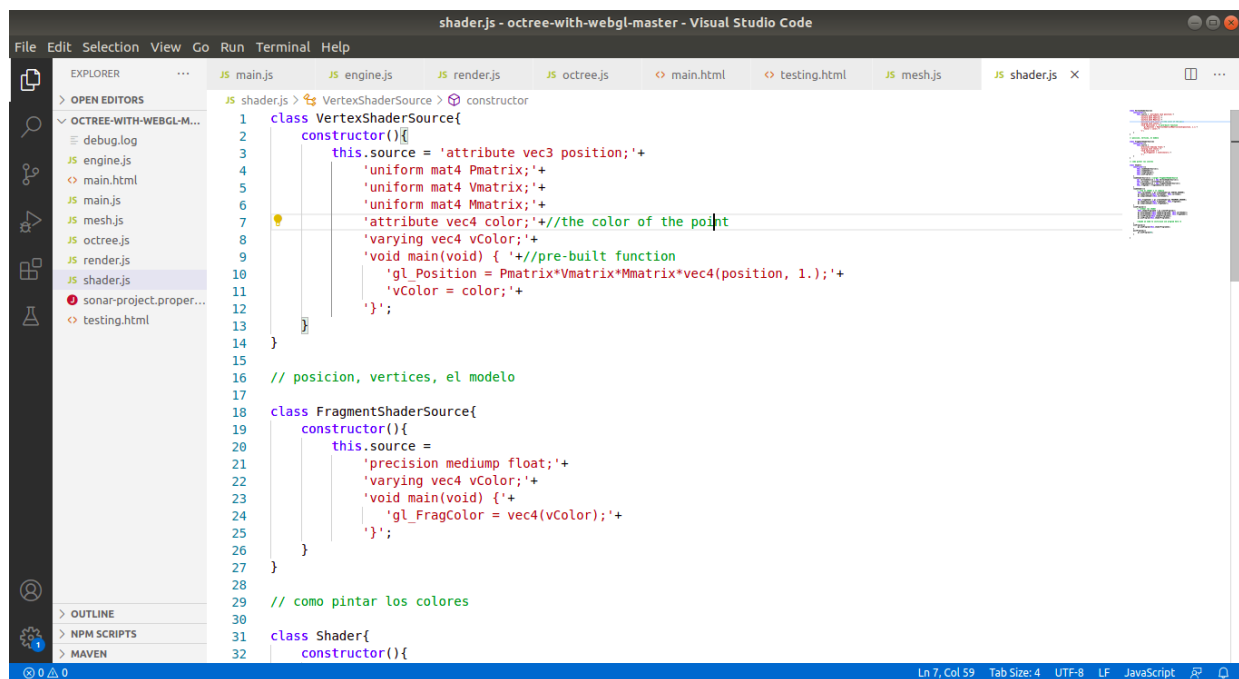


```

render.js - octree-with-webgl-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
OCTREE-WITH-WEBGL-M...
  debug.log
  engine.js
  main.html
  main.js
  mesh.js
  octree.js
  render.js
  shader.js
  sonar-project.proper...
  testing.html
OUTLINE
NPM SCRIPTS
MAVEN
1  var cubeRotation = 0.0;
2
3  class Canvas{
4    constructor(htmlId){
5      let canvas = document.querySelector(htmlId);
6      this.gl = canvas.getContext('webgl') || canvas.getContext('experimental-webgl');
7    }
8    init(){
9      if(!this.gl){
10        alert('No se pudo inicializar WebGL');
11        return;
12      }
13      var shader = new Shader();
14      this.shaderProgram = shader.initShaderProgram(this.gl);
15      this.programInfo = {
16        program: this.shaderProgram,
17        attribLocations: {
18          vertexPosition: this.gl.getAttribLocation(this.shaderProgram, 'aVertexPosition'),
19          vertexColor: this.gl.getAttribLocation(this.shaderProgram, 'aVertexColor'),
20        },
21        uniformLocations: {
22          projectionMatrix: this.gl.getUniformLocation(this.shaderProgram, 'uProjectionMatrix'),
23          modelViewMatrix: this.gl.getUniformLocation(this.shaderProgram, 'uModelViewMatrix'),
24        }
25      };
26      this.buffers = this.initBuffers();
27      var then = 0;
28
29      this.render = function(now){
30        now *= 0.001;
31        const deltaTime = now - then;
32        then = now;

```

Figura B.7: Captura de Pantalla de render.js

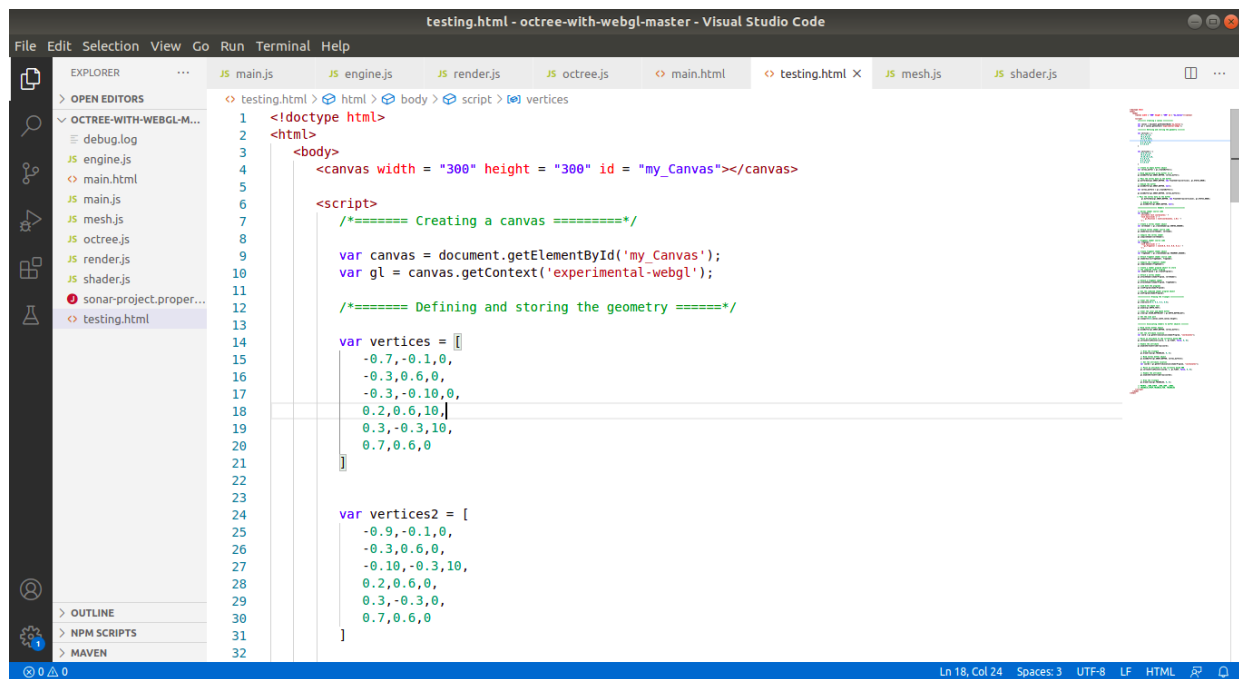


```

shader.js - octree-with-webgl-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> OPEN EDITORS
OCTREE-WITH-WEBGL-M...
  debug.log
  engine.js
  main.html
  main.js
  mesh.js
  octree.js
  render.js
  shader.js
  sonar-project.proper...
  testing.html
OUTLINE
NPM SCRIPTS
MAVEN
1  class VertexShaderSource{
2    constructor(){
3      this.source = 'attribute vec3 position;'+
4        'uniform mat4 Pmatrix;'+
5        'uniform mat4 Vmatrix;'+
6        'uniform mat4 Mmatrix;'+
7        'attribute vec4 color;'+//the color of the point
8        'varying vec4 vColor;'+
9        'void main(void) { '+//pre-built function
10          'gl_Position = Pmatrix*Vmatrix*Mmatrix*vec4(position, 1.);'+
11          'vColor = color;'+
12        '}';
13    }
14  }
15
16  // posicion, vertices, el modelo
17
18  class FragmentShaderSource{
19    constructor(){
20      this.source =
21        'precision mediump float;'+
22        'varying vec4 vColor;'+
23        'void main(void) {'+
24        'gl_FragColor = vec4(vColor);'+
25        '}';
26    }
27  }
28
29  // como pintar los colores
30
31  class Shader{
32    constructor(){

```

Figura B.8: Captura de Pantalla de shader.js



```
1 <!doctype html>
2 <html>
3 <body>
4   <canvas width = "300" height = "300" id = "my_Canvas"></canvas>
5
6   <script>
7     /*===== Creating a canvas =====*/
8
9     var canvas = document.getElementById('my_Canvas');
10    var gl = canvas.getContext('experimental-webgl');
11
12    /*===== Defining and storing the geometry =====*/
13
14    var vertices = [
15      -0.7, -0.1, 0,
16      -0.3, 0.6, 0,
17      -0.3, -0.10, 0,
18      0.2, 0.6, 10,
19      0.3, -0.3, 10,
20      0.7, 0.6, 0
21    ]
22
23
24    var vertices2 = [
25      -0.9, -0.1, 0,
26      -0.3, 0.6, 0,
27      -0.10, -0.3, 10,
28      0.2, 0.6, 0,
29      0.3, -0.3, 0,
30      0.7, 0.6, 0
31    ]
32  ]
```

Figura B.9: Captura de Pantalla de testing.js

Anexo C. Repositorios

Toda la implementación de la estructura OcTree, el código, documento, imágenes y un README, se encuentran en los siguientes repositorios

- GitHub: <https://github.com/Leslym03/EDA-Grupo1/tree/master/OcTree>
- GitLab: <https://gitlab.com/gabrieldgc1999/octree-with-webgl>