Laboratorio 02: Octree

Pablo Luis Carazas Barrios

Setember 2023

Abstract

En la creación de imágenes por computadora, siempre ha existido el desafío de gestionar eficientemente el espacio de memoria. A medida que la industria evolucionó hacia los gráficos en 3D, este desafío se amplificó con una dimensión adicional. Sin embargo, surgió una solución ingeniosa que no reinventó la rueda, sino que modificó una solución ya existente para adaptarse perfectamente a este nuevo espacio tridimensional.

1 Introduction

Como fue mencionado, la solucion para este problema fue la creacion del Octree, una estructura que almacena datos tridimensionales en octantes, de modo que los espacios sin puntos quedan vacios, en el siguiente informe voy a presentar mi version del Octree programada en c++.

2 GitHub

La carpeta Laboratorio-02 contiene dos versiones: una es el programa completo Octree.cpp , que incluye la versión original del programa con la estructura de octree, y la otra versión es Octree_With_Graphics.cpp, que contiene la versión del programa con gráficos. Sin embargo, nos enfocaremos principalmente en el archivo de la estructura original.

3 Codigo

A partir de ahora, comenzaremos a analizar e explicar la implementación.

3.1 Includes

Para la implementación de las estructuras, excluyendo la parte gráfica, utilizaremos las siguientes bibliotecas:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <sstream>
using namespace std;
```

Estas bibliotecas se utilizan para posibles entradas y salidas, para la apertura de archivos, para el uso de vectores y también para cálculos como la raíz.

3.2 Variable global

La forma base del octree permite la existencia de un punto por octante, pero también existe la posibilidad de permitir más de un punto por octante. Por lo tanto, utilizaremos esta variable para modificar nuestra estructura en el futuro:

```
int PO = 1;
```

3.3 Struct Point

Nuestra estructura 'Point' se utiliza para almacenar puntos tridimensionales. Los puntos se guardan como números de punto flotante (doubles) para facilitar cálculos al dividir los octantes. Además, esta estructura cuenta con un método 'print' que permite mostrar el punto por consola.

```
struct Point {
    double x;
    double y;
    double z;

    Point(double _x = 0, double _y = 0, double _z = 0) {
        x = _x;
        y = _y;
        z = _z;
    }

    void print() {
        cout << "x=" << x << " y=" << y << " z=" << z;
    }
};
```

3.4 Función ReadCSV

La función 'ReadCSV' tiene la finalidad de leer archivos CSV. Para cada fila del archivo, separa las tres columnas en tokens y luego crea un nuevo objeto 'Point' con los valores de las tres columnas, que se utilizan como coordenadas. Estos puntos se agregan a un vector, y al final de la lectura del archivo, la función devuelve un vector que contiene todos los puntos extraídos del archivo CSV.

```
vector<Point> ReadCSV(const string& filename) {
    vector<Point> points;
    ifstream file(filename);

    if (!file.is_open()) {
        cerr << "Error: No se pudo abrir el archivo " << filename << endl;
        return points;
    }

    string line;
    while (getline(file, line)) {
        stringstream ss(line);
        string token;
        vector<int> tokens;
        while (getline(ss, token, ';')) {
            int value = stoi(token);
            tokens.push_back(value);
        }

        // Verificar si se obtuvieron suficientes tokens (x, y, z)
        if (tokens.size() == 3) {
            Point p(tokens[0], tokens[1], tokens[2]);
            points.push_back(p);
        }
        else {
            cerr ;
        }
    }
```

```
    file.close();
    return points;
}
```

3.5 Struct Octal

A partir de acá comenzaremos a desmenuzar la estructura para poder entederla bien, a sus tributos y metodos

```
struct Octal {
        Point bottomLeft;
        double h = 0;
        vector<Octal*> Childrens;
        vector<Point> hojas;
```

El atributo `Point bottomLeft;` desde donde comenzaremos a contar nuestro octal, `double h=0;` es donde guardaremos el tamaño de nuestro octal, ahora el `vector≤Octal*≥ Childres` van a ser los hijos de nuestro octal y `vector≤Point≥ hojas` es donde guardaremos los puntos que contrendra nuestro octal, eso es un vector para los casos donde `PO` sea mayor que 1.

```
        Octal(Point bl, double _h) {
                Childrens.resize(8, nullptr);
                bottomLeft = bl;
                h = _h;
        }
```

Es un constructor simple, se pasa como parametro un Point que sera nuestro `bottomLeft`, tambien se reasigna el tamaño de `Childres` a 8 y se hace que todos los valores sean punteros nulos, y `_h` se le iguala a

```
        void add\_point(Point p) {
                hojas.push_back(p);
        }
```

Esta funcion solo agrega un nuevo `Point p` a nuestra lista de hojas

```
        void newOctans() {
                double mid = h / 2;
                double x = bottomLeft.x;
                double y = bottomLeft.y;
                double z = bottomLeft.z;
                Childrens[0] = new Octal(bottomLeft, mid);
                Childrens[1] = new Octal(Point(x + (mid), y, z), mid);
                Childrens[2] = new Octal(Point(x, y, z + (mid)), mid);
                Childrens[3] = new Octal(Point(x + (mid), y, z + (mid)), mid);
                Childrens[4] = new Octal(Point(x, y + (mid), z), mid);
                Childrens[5] = new Octal(Point(x + (mid), y + (mid), z), mid);
                Childrens[6] = new Octal(Point(x, y + (mid), z + (mid)), mid);
                Childrens[7] = new Octal(Point(x + (mid), y + (mid), z + (mid)), mid);
        }
```

Esta funcion aunque parece un poco larga es simple, se encarga de inicializar todos los hijos de el Octal en el que se este, la logica es simple, se crean combinando todas las posbiles combinaciones de coordenada entre x,y,z siendo sumadas por mid.

```
        void asignOctan() {
        if (hojas.size() > PO or Childrens[0]) {
                if (!Childrens[0]) {
                        newOctans();
                }
                double midx = bottomLeft.x + (h / 2);
```

```
                double midy = bottomLeft.y + (h / 2);
                double midz = bottomLeft.z + (h / 2);
                for (auto& it : hojas) {
                        if (it.x <= midx and it.y <= midy and it.z <= midz) {
                                Childrens[0]->hojas.push_back(it);
                                Childrens[0]->asignOctan();
                        }
                        else if (it.x > midx and it.y <= midy and it.z <= midz) {
                                Childrens[1]->hojas.push_back(it);
                                Childrens[1]->asignOctan();
                        }
                        else if (it.x <= midx and it.y <= midy and it.z > midz) {
                                Childrens[2]->hojas.push_back(it);
                                Childrens[2]->asignOctan();
                        }
                        else if (it.x > midx and it.y <= midy and it.z > midz) {
                                Childrens[3]->hojas.push_back(it);
                                Childrens[3]->asignOctan();
                        }
                        else if (it.x <= midx and it.y > midy and it.z <= midz) {
                                Childrens[4]->hojas.push_back(it);
                                Childrens[4]->asignOctan();
                        }
                        else if (it.x > midx and it.y > midy and it.z <= midz) {
                                Childrens[5]->hojas.push_back(it);
                                Childrens[5]->asignOctan();
                        }
                        else if (it.x <= midx and it.y > midy and it.z > midz) {
                                Childrens[6]->hojas.push_back(it);
                                Childrens[6]->asignOctan();
                        }
                        else if (it.x > midx and it.y > midy and it.z > midz) {
                                Childrens[7]->hojas.push_back(it);
                                Childrens[7]->asignOctan();
                        }
                }
                        hojas.clear();
                }
                else {
                        return;
                }
        }

};
```

Esta funcion es la encargada princpipal del funcionamiento del Octree, `if (hojas.size() ≤ PO or Childrens[0]` verifica que se cumpla que haya la cantidad de puntos por octante, en este caso agregamos Point (20,30,10)

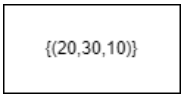

Figure 1: Cantidad igual a PO

luego queremos añadir un Point (10,10,10), entonces se vuelve a llamar a la funcion `asignOctan()` (más adelante se ve cuando ocurre eso) a ese Octante, al añadir ocurre esto

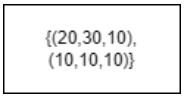

Figure 2: Cantidad supera a PO

pero en esta ocacion se supera nuestro PO ya que `hojas.size()` es 2, entonces se crean sus nuevos hijos

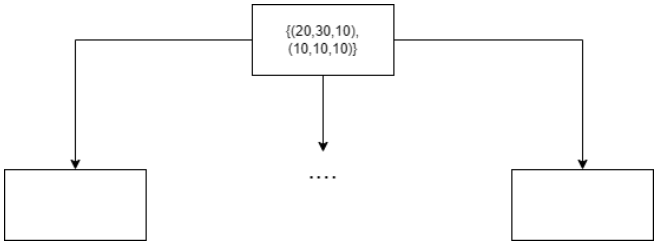

Figure 3: Se crean los Childrens

y de ahi se entra dentro del `for` se comienza a revisar hoja por hoja, luego se busca al Children que corresponde como la imagen

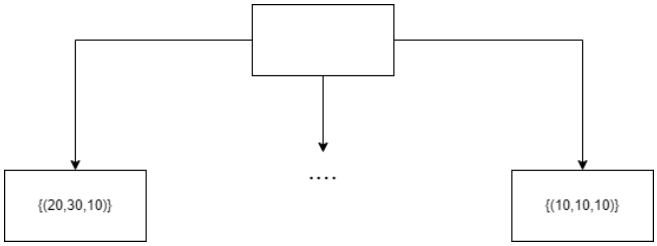

Figure 4: Se reasigna

y se hace un `push` en el vector de hojas de ese hijo y se llama de nuevo dentro de ese hijo a la funcion `asignOctan()`, si es que se cumple el primer `if` acaba la recursion y se limpia las hojas que se usaron y no fueron finanles, en caso no se cumple se hace el mismo proceso hasta que es cumpla la condicion, en caso que `PO` sea mayor normal podrá seguir almacenando puntos dentro

3.6 Struct Octree

La estructura solo contiene como atributos un Octal la cual será la raiz, un vector donde se almacenarán todos los puntos para procesar el cuadrante contenedor de todos y una variable auxiliar dif

```
struct OcTree {
        Octal* root = nullptr;
        vector<Point> puntos;
        double dif = 0;

        void setpuntos(vector<Point> p) {
                puntos = p;
        }
```

también se cuenta con una funcion `void setpuntos(vector<Point> p)` donde se guardaran todos los puntos por procesar.

```
        Point maxmin() {
                int maxx = INT_MIN;
                int maxy = INT_MIN;
                int maxz = INT_MIN;
```

```
        int minx = INT_MAX;
        int miny = INT_MAX;
        int minz = INT_MAX;
        for (auto& it : puntos) {
                if (it.x < minx) minx = it.x;
                if (it.y < miny) miny = it.y;
                if (it.z < minz) minz = it.z;

                if (it.x > maxx) maxx = it.x;
                else if (it.y > maxy) maxy = it.y;
                else if (it.z > maxz) maxz = it.z;

        }
        if (abs(minx - maxx) > dif) dif = abs(minx - maxx);
        if (abs(miny - maxy) > dif) dif = abs(miny - maxy);
        if (abs(minz - maxz) > dif) dif = abs(minz - maxz);
        Point min(minx, miny, minz);
        return min;
}
```

Esta funcion `Point maxmin()` se encarga de obtener el punto minimo en la coordenada *x,y,z*, esas coordenadas minimas las guarda en un punto temporal `Point min(minx,miny,minz)` y tambien se obtiene la mayor entre todas las coordenadas y se guarda en auxiliares, luego se compara con los minimos de todos, y el que contenga la mayoy diferencia sera dif, el cual sera h dentro del primer Octal raiz, y para finalizar la funcion se retorna `min`.

```
Octal* Find(Point p) {
        Octal* tmp = root;
        while (tmp) {
        for (int i = 0; i < tmp->hojas.size(); i++) {
                if (tmp->hojas[i].x == p.x and
                        tmp->hojas[i].y == p.y and
                        tmp->hojas[i].z == p.z) {
                        return tmp;
                }
        }
        double midx = tmp->bottomLeft.x + (tmp->h / 2);
        double midy = tmp->bottomLeft.y + (tmp->h / 2);
        double midz = tmp->bottomLeft.z + (tmp->h / 2);
        if (tmp->Childrens[0] and p.x <= midx and p.y <= midy and p.z <= midz) {
                tmp = tmp->Childrens[0]; }
        else if (tmp->Childrens[1] and p.x > midx and p.y <= midy and p.z <= midz) {
                tmp = tmp->Childrens[1]; }
        else if (tmp->Childrens[2] and p.x <= midx and p.y <= midy and p.z > midz) {
                tmp = tmp->Childrens[2]; }
        else if (tmp->Childrens[3] and p.x > midx and p.y <= midy and p.z > midz) {
                tmp = tmp->Childrens[3]; }
        else if (tmp->Childrens[4] and p.x <= midx and p.y > midy and p.z <= midz) {
                tmp = tmp->Childrens[4]; }
        else if (tmp->Childrens[5] and p.x > midx and p.y > midy and p.z <= midz) {
                tmp = tmp->Childrens[5]; }
        else if (tmp->Childrens[6] and p.x <= midx and p.y > midy and p.z > midz) {
                tmp = tmp->Childrens[6]; }
        else if (tmp->Childrens[7] and p.x > midx and p.y > midy and p.z > midz) {
                tmp = tmp->Childrens[7]; }
        else {
                return NULL;
```

```
                }
        }
        return NULL;
}
```

La funcion `Find` recibe un `Point p` y devuelve un Octal, se crea un Octal temporal al cual se le asigna el valor de la raiz, luego inicia un bucle while que la condicion es que siempre que exista tmp seguirá, dentro del while hay un `for (int i = 0; i < tmp->hojas.size(); i++)`, allí se revisa dentro de todas las posibles hojas, ya que en casos que `PO` sea mayor que uno se tiene que revisar entro los posibles casos,con el `if (tmp->hojas[i].x == p.x and tmp->hojas[i].y == p.y and tmp->hojas[i].z == p.z) return tmp;` se verifica si es que algunos de los posibles puntos es el que se busca, si es se retorna el octal que lo contiene al punto y `p` se le asigna ese punto que se busca. En caso que ninguno de las hojas sea el que se busca se saca un calculo al igual que la funcion `asingOctant` y se iguala nuestro Octante temporal a ese Children y vuelve a iniciar el ciclo del while para buscar, en caso ninguno de los hijos tenga algun Children, al tmp ser igualado a ese puntero Octante faltante será null, y al ser null acaba el ciclo informando que no existe el Point.

```
Octal* OctalContainer(Point p, double radius) {
Octal* tmp = root;
while (tmp and tmp->h > radius) {

        double midx = tmp->bottomLeft.x + (tmp->h / 2);
        double midy = tmp->bottomLeft.y + (tmp->h / 2);
        double midz = tmp->bottomLeft.z + (tmp->h / 2);
        if (tmp->Childrens[0] and p.x <= midx and p.y <= midy and p.z <= midz) {
                tmp = tmp->Childrens[0];
        }
        else if (tmp->Childrens[1] and p.x > midx and p.y <= midy and p.z <= midz) {
                tmp = tmp->Childrens[1];
        }
        else if (tmp->Childrens[2] and p.x <= midx and p.y <= midy and p.z > midz) {
                tmp = tmp->Childrens[2];
        }
        else if (tmp->Childrens[3] and p.x > midx and p.y <= midy and p.z > midz) {
                tmp = tmp->Childrens[3];
        }
        else if (tmp->Childrens[4] and p.x <= midx and p.y > midy and p.z <= midz) {
                tmp = tmp->Childrens[4];
        }
        else if (tmp->Childrens[5] and p.x > midx and p.y > midy and p.z <= midz) {
                tmp = tmp->Childrens[5];
        }
        else if (tmp->Childrens[6] and p.x <= midx and p.y > midy and p.z > midz) {
                tmp = tmp->Childrens[6];
        }
        else if (tmp->Childrens[7] and p.x > midx and p.y > midy and p.z > midz) {
                tmp = tmp->Childrens[7];
        }
        else {
                return nullptr;
        }
}
return tmp;
}
```

La funcion `OctalContainer` usa la misma logica de busqueda que el Find, solo que el While se detiene cuando `tmp->h > radius`, esto significando que mientras que el octal sea mas grande que el

radio de busqueda podemos ir reduciendo el espacio donde queremos buscar hasta que se cumpla la condicion y ahí iniciar una busqueda y detenerse, solo que seta funcion su importancia es que nos devuelve el octal donde está contenido el Octal que contiene el punto, de modo que `[Octal que devuelve]->contiene->[Octal]->'punto que se busca'`.

```
void Leafs(Octal* p, vector<Point>& hojas) {
        if (!p) {
                return;
        }
        if (!p->hojas.empty()) {
                for (const Point& hoja : p->hojas) {
                        hojas.push_back(hoja);
                }
        }
        for (Octal* suboctal : p->Childrens) {
                Leafs(suboctal, hojas);
        }
}
```

La funcion `Leafs(Octal *p,vector<Point>&hojas)` recibe como parametros un Octal donde se comenzara la recursion y un puntero a un vector de Puntos llamado hojas, la logica es la siguiente. El caso donde termina el codigo es si el Octal `p` no existe, si es que si existe verifica si tiene almacenados Points, si es que si se itera dentro de su vector de puntos `p->hojas` y se comienza a añadir, una vez acabado el for se vuelve a llamar a la funcion con cada hijo en el segundo `for (Octal* suboctal : p->Childrens) Leafs(suboctal, hojas);` con la finalidad de almacenar todos los puntos que se contiene en ese Octal que contiene octales.
El motivo por el cual se toma un cubo más grande es para poder tener un espectro de busqueda mayor, el cual puede tener un menor desempeño computacionalmente hablando pero es más optimo que

```
Point find_closest(Point p, double radius) {
    Octal* tmp = OctalContainer(p, radius);
    vector<Point> Candidatos;
    Leafs(tmp, Candidatos);
    double min = INT_MAX;
    Point Closest;
    for (auto& it : Candidatos) {
        double calc = calc = distance(p, it);
        if (calc < min and calc != 0 and calc < radius) {
            Closest = it;
            min = calc;
        }
    }
    return Closest;
}
```

Ahora la funcion `find_closest` se le pasa un Point que se desea buscar y el radio de busqueda, se llama la funcion que nos devuelve el Octal que contiene el Octal del punto el motivo por el cual se toma un cubo más grande es para poder tener un espectro de busqueda mayor ya que pueden haber puntos que estén limitando en el borde del octal, esto puede tener un menor desempeño computacionalmente hablando pero obtiene siempre la respuesta real

```
void insert(Point p) {
        if (!root) {
                root = new Octal(maxmin(), dif);
                root->add_point(p);
        }
        else if (!Find(p)) {
```

```
                root->hojas.push_back(p);
                root->asignOctan();
            }
        }
```

Esta funcion es la mas simple ya que solo llama a las funciones previamente declaradas, si no existe ningun punto aun se crea la raiz y usando el contructor de Octal y pasando como parametros `maxmin(),dif` de modo que el `botomLeft` es dado por maxmin y dif es el tamaño del bloque, en caso que ya exista la raiz se verifica que el punto que se quiere insertar no exita previamente, si no existe se pushea a la raiz y luego se usa la funcion `asignOctan()` para descender hasta la hoja que le corresponde.

```
        void make() {
            int iteracion = 0;
            for (int i = 0; i < puntos.size(); i++, iteracion++) {
                insert(puntos[i]);
            }
        }
```

Esta funcion solo se encarga de insertar todos los puntos en el Octree

3.7 Main

```
int main() {
        OcTree octree;
        string filename = "Puntos1.csv"; // Reemplaza con el nombre de tu archivo CSV
        vector<Point> points = ReadCSV(filename);

        octree.setpuntos(points);
        octree.make();
        vector<Octal*> octales; //esto se usara para graficar
        octree.OctalLeafs(octales); //son todos los octales que contienen puntos
}
```

Crea la estructura OcTree, luego se da el nombre del archivo que se quiere añadir los puntos, se crea un vector de puntos donde se iguala la funcion `ReadCSV`. Luego se usa `setpuntos(points)` para setear los puntos, se llama al make y queda todo el arbol construido.

4 Graficos

Como fue mencionado previamente el codigo implementado brinda la capacidad de poder decidir cuantos puntos por octal y veremos las 2 figuras con varios casos diferentes de PO *(puntos por octal)* y tambien probé 2 funciones para graficar, las 2 funciones no son directamente mias ya que no pude instalar el vtk, la primera version de grafico fue una dada por Gpt y la otra fue dada por la ayuda de un comáñero

4.1 Gatito

El gato contiene 1136 puntos en total, y varios no siendo procesados ya que son repetidos, una cantidad relativamente pequeña la cual puede ser facil de ejecutar para Vtk

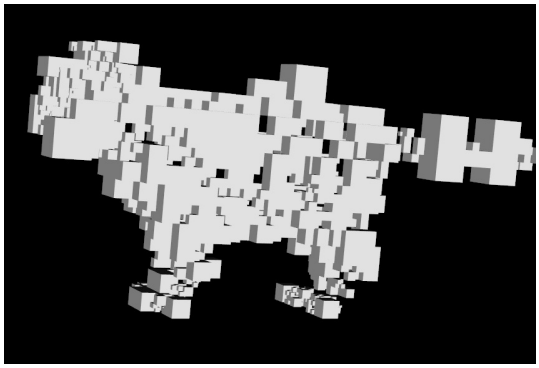

Figure 5: PO=1

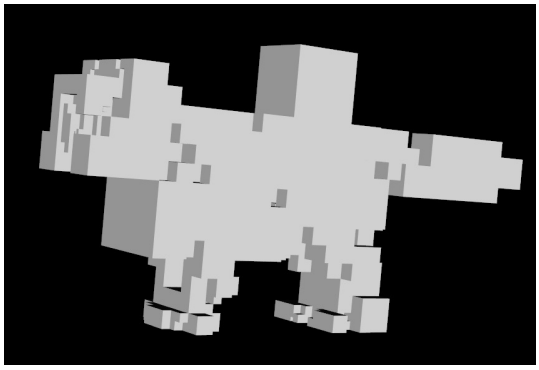

Figure 6: PO=4

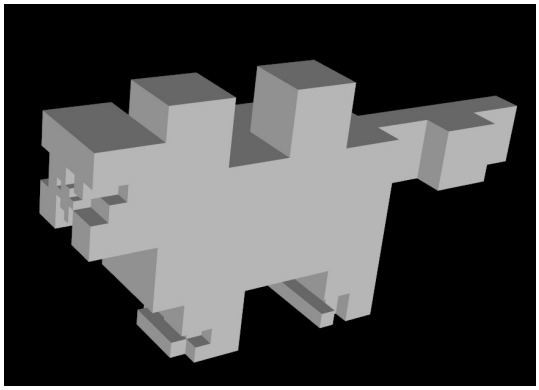

Figure 7: PO=15

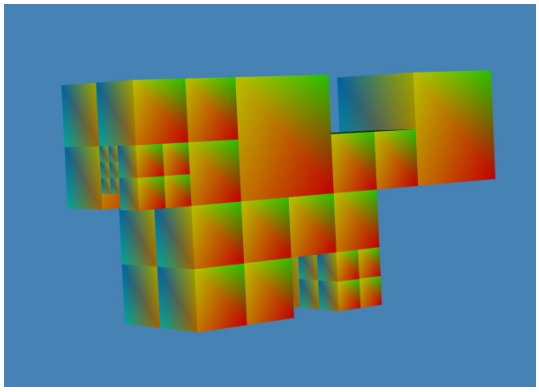

Figure 8: PO=70 con diferente funcion graficadora

4.2 Dragon

El dragon si fue un poco más pesado de procesar para la maquina pero si pudo

Figure 9: PO=1

Figure 10: PO=4

Figure 11: PO=15

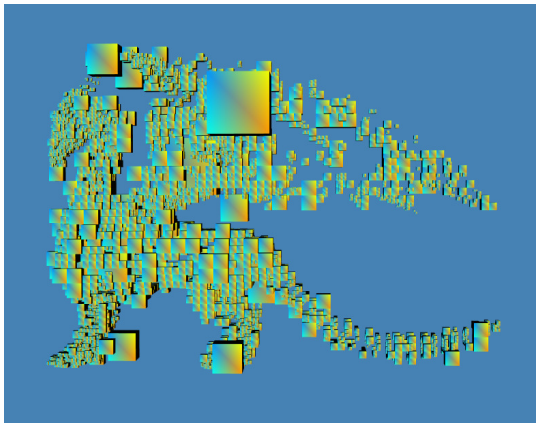

Figure 12: PO=1 y otra funcion de graficadora

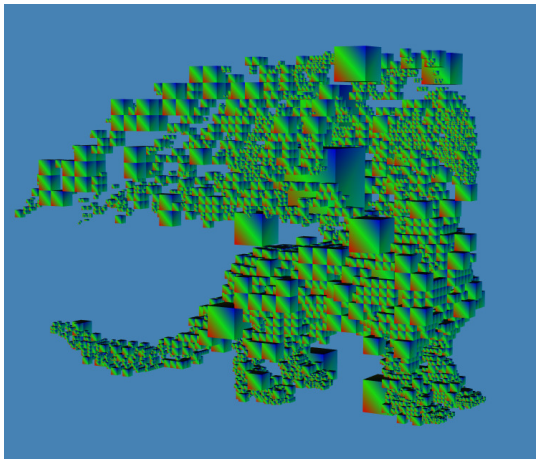

Figure 13: PO=4 y otra funcion de graficadora

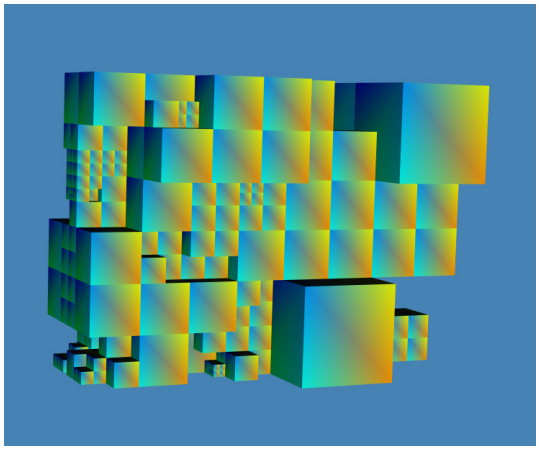

Figure 14: PO=70 y otra funcion de graficadora

5 Conclusiones

Se puede concluir y se demostró que el octree es una estructura bastante buena a la hora de almacenar valores, se vio que a medida que la cantidad de puntos por Octales aumentaba comenzo a mejorar bastante bien el rendimiento ya que reduce de manera logarimica la profundidad y la distribucion de valores.
Como se pudo apreciar en las figuras al aumentar la cantidad de puntos por octal la resolucion de nuestras modelos 3d se redujo bastante ya que las subdivisiones ocurrian con menos frecuencia. Todo esto es una gran herramienta de optimizacion de recursos en muchos casos siendo ejemplo de esto un juego, si en un juego hacemos una funcion que a medida que un personaje se acerque a una figura la cantidad de puntos por Octante disminuya ya que es algo que se verá muy de cerca y se apreciaran detalles, pero si es que se aleja tambien aumenta la cantidad de puntos por octal de modo que se ahorren recursos ya que el jugador por la distancia no sera capaz de ver los detalles.
En resumen, se logro implementar el Octree y demostrar su potencia de una manera visual y eficiente con la posibilidad de reducir y aumentar la cantidad de puntos por octal

6 Agradecimientos

Como mencioné no pude instalar VTK y algunos compañeros me hicieron de favor de compilar mis codigos en sus maquinas con VtK y tomar los screens desde allí. Gracias Mark y Agenl `<3`

7 Enlaces

Github con el codigo fuente e informe