UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

PRIMER EXAMEN

Curso : Estructura de Datos Avanzados

Docente:

Vicente Enrique Machaca Arceda

Integrantes:

Renato Gonzalo Cespedes Fuentes

Josnick Chayña Batallanes

Angelo Perez Rodriguez

AREQUIPA - PERÚ

2020

Índice

1. Introducción

En el presente documento se presenta la estructura del octree con su respectiva visualización en 3d.

2. Octree

Un octárbol es una estructura de datos de árbol en la que cada nodo interno tiene exactamente ocho hijos. Los octárboles se utilizan con mayor frecuencia para dividir un espacio tridimensional subdividiéndolo recursivamente en ocho octantes y su análogo son los quadtrees que son usados para la representación en 2D.

La idea principal es un algoritmo de divide y venceras. En el quadtree el plano se divide en cuadrantes. Si se usa el cuadrante completo se describe como 'full ', y si no se considera 'empty'. Si es parcialmente usado se subdivide, hasta una profundidad determinada. Un octree es semejante solo que cuenta con tres dimensiones, subdivididos en octantes de manera recursiva. El número de nodos es proporcional al perimetro en un quadtree o superficie del objeto en un octree.

3. Historia

El uso de octrees para gráficos por computadora en 3D fue iniciado por Donald Meagher en el Instituto Politécnico de Rensselaer, descrito en un informe de 1980 .ºctree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer", para el cual él posee una patente de 1995 (con una fecha de prioridad de 1984) "High-speed image generation of complex solid objects using octree encoding".

En los últimos años el realismo conseguido en los gráficos por ordenador se acerca visualmente a la realidad hasta l´ımites insospechados hace pocos lustros. Las industrias del cine y los videojuegos así como los grupos de investigación de las universidades de todo el mundo, se han afanado en conseguir métodos de iluminación cada vez mas cercanos al comportamiento físico de la luz, en simular las propiedades de los materiales mas complejos, y desarrollar técnicas de digitalización que consiguen replicas exactas de objetos reales con precisión microscópica. Este aumento en la calidad de los gráficos obtenidos lleva aparejado un aumento en las prestaciones requeridas en cuanto a computo, siendo prácticamente imprescindible el uso de procesamiento paralelo para ejecutar los algoritmos de renderizado y el tratamiento de modelos cada vez mas complejos en un tiempo razonable. Estas arquitecturas paralelas ya no son un privilegio de las grandes empresas, sino que cada usuario dispone de procesadores vectoriales en su tarjeta gráfica. Podríamos decir que estamos llegando al lımite de la perfección en cuanto a la percepción visual de los graficos por ordenador, pero esta misma perfección lleva aparejados otros problemas que son los que en parte intentamos resolver en el trabajo que sustenta la presente memoria.

Es en el marco de esta problemática donde se inicia el trabajo de investigación que ha dado lugar a la presente memoria, y en el cual se abordan las cuestiones relativas a la simplificación de grandes modelos poligonales de forma que se puedan realizar de forma eficiente la detección de colisiones entre objetos as´ı como la visualización de estos grandes modelos a través de una red de comunicaciones de forma progresiva y/o adaptativa (OCTREE).

4. Aplicaciones del Octree

4.1. Videojuegos

La estructura es bastante óptima para detectar puntos en un espacio determinado , es por ello que la utilización en Shooters con una pseudo implementación de ray casting para el seguimiento de algunas partículas que flotan y tienen interacción con el entorno .

4.2. Color Quantization

la cuantificación del color o la cuantificación de la imagen del color es la cuantificación aplicada a los espacios de color ; es un proceso que reduce la cantidad de colores distintos utilizados en una imagen , generalmente con la intención de que la nueva imagen sea lo más similar posible visualmente a la imagen original.

4.3. Finite element analysis

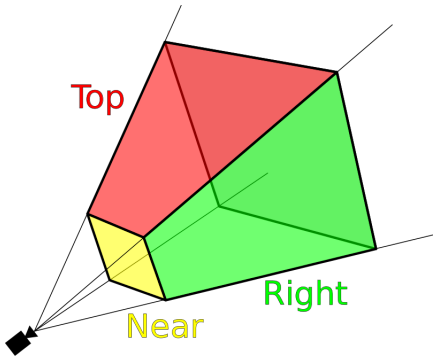
el objeto o sistema se representa por un modelo geométricamente similar que consta de múltiples regiones discretas simplificadas y conectadas. Ecuaciones de equilibrio, junto con consideraciones físicas aplicables así como relaciones constitutivas, se aplican a cada elemento, y se construye un sistema de varias ecuaciones. El sistema de ecuaciones se resuelve para los valores desconocidos usando técnicas de álgebra lineal o esquemas no lineales, dependiendo del problema. Siendo un método aproximado, la precisión de los métodos FEA puede ser mejorada refinando la discretización en el modelo, usando más elementos y nodos.

4.4. View frustum culling

"Tronco piramidal.[es] la región cerrada del espacio que delimita los objetos que aparecen representados en la pantalla. De acuerdo con la geometría de la cámara virtual que sirve para visualizar las imágenes de objetos cuyas coordenadas se conocen, delimita la zona visible que aparece en las imágenes generadas estas pueden ser:

- **VPN:** Vision normal al plano
- **VUV:** Vector del plano que indica el sentido adhiasente

- **VRP:** Punto de referencia de la vista, localizada en el plano de visión y origen VRC
- **PRP:** Punto de referencia de proyección, que es donde la imagen se proyecta
- **VRC:** Sistemas de coordenadas de la referencia de la vista



5. Algoritmo

Listing 1: .Algoritmo Octree con vtk"

```
from vtk import *
from vtk.wx.wxVTKRenderWindowInteractor import wxVTKRenderWindowInteractor
from vtk.wx.wxVTKRenderWindow import wxVTKRenderWindow
import wx
from random import*


class Rectangulo:
    def __init__(self, x, y, z, a, b, c):
        self.x = x
        self.y = y
        self.z = z
        self.w = a
        self.h = b
        self.p = c

    def contains(self, point):
        return (point.x <= self.x+self.w and point.x >= self.x-self.w and self.y+self.h >=
            point.y and point.y >= self.y-self.h and self.z+self.p >= point.z and point.z >=
            self.z-self.p)


    def intersect(self, range): # range es otro cubo
        return (range.x + range.w > self.x - self.w or range.x - range.w < self.x + self.w or
            range.y + range.h > self.y - self.h or range.y - range.h < self.y + self.h or
            range.z + range.p > self.z - self.p or range.z - range.p < self.z + self.p)




class Point:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

class Octree:
    def __init__(self, cub, tam, color, color_punto):
        self.cubo = cub
        self.capacidadMaxima = tam
        self.points = []
        self.divided = False
        self.hijos = []
        self.color = color
        self.color_punto = color_punto
```

```
def insert(self, point, puntosG, estado, colores):
    #PuntosG almacena todos los nuevos arboles, para ser dibujados
    if not self.cubo.contains(point):
        return

    if len(self.points) < self.capacidadMaxima:
        self.points.append(point)
        estado[0] = True
        colores.append(self.color_punto) # asignacion del color
        return
    else:#division del octree en caso de que alcance
         #el maximo de puntos
        if not self.divided:
            self.subdivide(puntosG)

    self.hijos[0].insert(point, puntosG, estado, colores)
    self.hijos[1].insert(point, puntosG, estado, colores)
    self.hijos[2].insert(point, puntosG, estado, colores)
    self.hijos[3].insert(point, puntosG, estado, colores)
    self.hijos[4].insert(point, puntosG, estado, colores)
    self.hijos[5].insert(point, puntosG, estado, colores)
    self.hijos[6].insert(point, puntosG, estado, colores)
    self.hijos[7].insert(point, puntosG, estado, colores)

#Funcion para subdividir el octree
def subdivide(self, puntosG):
    # puntosG almacenar  todos los nuevos arboles creados, para ser dibujados en el vtk
    x = self.cubo.x
    y = self.cubo.y
    z = self.cubo.z
    w = self.cubo.w / 2
    h = self.cubo.h / 2
    p = self.cubo.p / 2
    #Division del octree en 8 cuadrantes

    #cuadrantes superiores
    neup = Rectangulo(x + w, y + h, z + p, w, h, p)
    noup = Rectangulo(x - w, y + h, z + p, w, h, p)
    seup = Rectangulo(x + w, y + h, z - p, w, h, p)
    soup = Rectangulo(x - w, y + h, z - p, w, h, p)

    #cuadrantes inferiores
    nedown = Rectangulo(x + w, y - h, z + p, w, h, p)
    nodown = Rectangulo(x - w, y - h, z + p, w, h, p)
    sedown = Rectangulo(x + w, y - h, z - p, w, h, p)
    sodown = Rectangulo(x - w, y - h, z - p, w, h, p)

    self.hijos.append(Octree(neup, self.capacidadMaxima, self.color.next_color(),
        self.color_punto.next_color_punto()))
    self.hijos.append(Octree(nedown, self.capacidadMaxima,
        self.hijos[0].color.next_color(), self.hijos[0].color_punto.next_color_punto()))
    self.hijos.append(Octree(noup, self.capacidadMaxima, self.hijos[1].color.next_color(),
        self.hijos[1].color_punto.next_color_punto()))
    self.hijos.append(Octree(nodown, self.capacidadMaxima,
        self.hijos[2].color.next_color(), self.hijos[2].color_punto.next_color_punto()))
    self.hijos.append(Octree(seup, self.capacidadMaxima, self.hijos[3].color.next_color(),
        self.hijos[3].color_punto.next_color_punto()))
    self.hijos.append(Octree(sedown, self.capacidadMaxima,
        self.hijos[4].color.next_color(), self.hijos[4].color_punto.next_color_punto()))
    self.hijos.append(Octree(soup, self.capacidadMaxima, self.hijos[5].color.next_color(),
        self.hijos[5].color_punto.next_color_punto()))
    self.hijos.append(Octree(sodown, self.capacidadMaxima,
        self.hijos[6].color.next_color(), self.hijos[6].color_punto.next_color_punto()))

    puntosG.append(Octree(neup, self.capacidadMaxima, self.color.next_color(),
        self.color_punto.next_color_punto()))
    puntosG.append(Octree(nedown, self.capacidadMaxima, self.hijos[0].color.next_color(),
        self.hijos[0].color_punto.next_color_punto()))
    puntosG.append(Octree(noup, self.capacidadMaxima,
        self.hijos[1].color.next_color(),self.hijos[1].color_punto.next_color_punto()))
    puntosG.append(Octree(nodown, self.capacidadMaxima,
        self.hijos[2].color.next_color(),self.hijos[2].color_punto.next_color_punto()))
    puntosG.append(Octree(seup, self.capacidadMaxima,
        self.hijos[3].color.next_color(),self.hijos[3].color_punto.next_color_punto()))
    puntosG.append(Octree(sedown, self.capacidadMaxima,
        self.hijos[4].color.next_color(),self.hijos[4].color_punto.next_color_punto()))
    puntosG.append(Octree(soup, self.capacidadMaxima,
        self.hijos[5].color.next_color(),self.hijos[5].color_punto.next_color_punto()))
    puntosG.append(Octree(sodown, self.capacidadMaxima,
        self.hijos[6].color.next_color(),self.hijos[6].color_punto.next_color_punto()))
    self.divided = True


#Funcion para hacer consultas al octree
#sobre cuantos puntos hay en una area
def query(self, cub, found):

    if not self.cubo.intersect(cub):
        return

    for punt in self.points:
```

```
            if cub.contains(punt):
                found.append(punt)

        if self.divided:
            self.hijos[0].query(cub, found)
            self.hijos[1].query(cub, found)
            self.hijos[2].query(cub, found)
            self.hijos[3].query(cub, found)
            self.hijos[4].query(cub, found)
            self.hijos[5].query(cub, found)
            self.hijos[6].query(cub, found)
            self.hijos[7].query(cub, found)



#Funcion usada para la asignaci n de colores para el octree
class AsignarColor:
    def __init__(self, a, b, c, d, e, f):
        self.a = a
        self.b = b
        self.c = c
        self.inicial_a = d
        self.inicial_b = e
        self.inicial_c = f

    def next_color(self):
        extra = None
        if self.a - 0.02 >= 0:
            extra = AsignarColor(self.a - 0.02, self.b, self.c, self.inicial_a, self.inicial_b,
                self.inicial_c)
        elif self.b - 0.02 >= 0:
            extra = AsignarColor(self.a, self.b-0.02, self.c, self.inicial_a, self.inicial_b,
                self.inicial_c)
        else:
            if self.c - 0.02 >= 0:
                extra = AsignarColor(self.a, self.b, self.c-0.02, self.inicial_a,
                    self.inicial_b, self.inicial_c)
            else:
                extra = AsignarColor(self.inicial_a - 0.05, self.inicial_b - 0.05,
                    self.inicial_c - 0.05, self.inicial_a - 0.05, self.inicial_b - 0.05,
                    self.inicial_c - 0.05)
        return extra


    def next_color_punto(self):
        extra = None
        if self.a + 0.15 < 1:
            extra = AsignarColor(self.a + 0.15, self.b, self.c, self.inicial_a, self.inicial_b,
                self.inicial_c)
        elif self.b + 0.15 < 1:
            extra = AsignarColor(self.a, self.b+0.15, self.c, self.inicial_a, self.inicial_b,
                self.inicial_c)
        else:
            if self.c + 0.15 < 1:
                extra = AsignarColor(self.a, self.b, self.c+0.15, self.inicial_a,
                    self.inicial_b, self.inicial_c)
            else:
                extra = AsignarColor(self.inicial_a + 0.05, self.inicial_b + 0.05,
                    self.inicial_c + 0.05, self.inicial_a + 0.05, self.inicial_b + 0.05,
                    self.inicial_c + 0.05)
        return extra


class MostrarOctree(wx.Frame):
    def __init__(self, parent, id):
        #Incializaci n del size de pantalla
        wx.Frame.__init__(self, parent, id, 'Examen_Estructura de Datos Avanzados', size=(1024,
            720))

        self.Val_X = None
        self.Val_Y = None
        self.Val_Z = None
        self.Tamanho_Cubo = None

        self.actor1=None # usado en las consultas


        #Generacion de la barra de menu
        menuBar = wx.MenuBar()

        menuFile = wx.Menu()
        itemAdd = menuFile.Append(-1, "&A adir_Punto", "Agregar punto al cubo")
        itemQuery = menuFile.Append(-1, "&Consultar", "Consutar en un rango")
        itemQuit = menuFile.Append(-1, "&Quit", "Quit application")
        self.Bind(wx.EVT_MENU, self.Salir, itemQuit)
        #Asignacion de botones para la sub barra de opciones
        self.Bind(wx.EVT_MENU, self.Agregar, itemAdd)
        self.Bind(wx.EVT_MENU, self.Consultar, itemQuery)
        self.Bind(wx.EVT_MENU, self.Salir, itemQuit)
        #Creacion de la barra de opciones
```

```
        menuBar.Append(menuFile, "&Opciones")
        self.SetMenuBar(menuBar)
        self.statusBar = self.CreateStatusBar()
        self.statusBar.SetFieldsCount(2)
        self.statusBar.SetStatusWidths([-4, -1])

        self.ren = vtk.vtkRenderer()
        self.ren.SetBackground(1, 1, 1)

        #Creacion del octree
        # Y genereacion de puntos aleatorios en el octree
        self.arbol = Octree(Rectangulo(80, 80, 80, 80, 80, 80), 4, AsignarColor(1, 1, 1, 1, 1,
            1), AsignarColor(0, 0, 0, 0, 0, 0))
        self.Create_Cubo(80, 80, 80, 80, 80, 80, 1, 1, 1,0.1)
        for i in range(0,20):
            punto_color = []
            puntosG=[]
            val=[False]
            val1=randrange(150)
            val2=randrange(150)
            val3=randrange(150)
            p=Point(val1,val2,val3)
            self.arbol.insert(p, puntosG, val, punto_color)
            for occt in puntosG:
                    self.Create_Cubo(occt.cubo.x, occt.cubo.y, occt.cubo.z, occt.cubo.w,
                        occt.cubo.h, occt.cubo.p, occt.color.a, occt.color.b, occt.color.c,0.1)
            self.Create_Punto(float(val1), float(val2), float(val3), punto_color[0].a,
                punto_color[0].b, punto_color[0].c)

        self.rwi = wxVTKRenderWindow(self, -1)
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.rwi, 1, wx.EXPAND)
        self.SetSizer(sizer)
        self.Layout()
        self.rwi.Enable(1)
        self.rwi.GetRenderWindow().AddRenderer(self.ren)

    #Funcion usada para la generaci n del grafico del octree
    def Create_Cubo(self, x1, y1, z1, x, y, z, a, b, c, d):

        cubooo = vtkCubeSource()
        cubooo.SetCenter(x1, y1, z1)
        cubooo.SetXLength(x*2)
        cubooo.SetYLength(y*2)
        cubooo.SetZLength(z*2)
        CubeMapper = vtkPolyDataMapper()
        CubeMapper.SetInputConnection(cubooo.GetOutputPort())
        CubeActor = vtkActor()

        CubeActor.GetProperty().SetColor(a, b, c)
        CubeActor.GetProperty().SetOpacity(d)
        CubeActor.SetMapper(CubeMapper)
        self.ren.AddActor(CubeActor)


        return CubeActor

    """Funcion usada para eliminar el actor cubo
    al momento de hacer consultas"""
    def delete_Cubo(self, actor):
        self.ren.RemoveActor(actor)


    #Creacion del punto en el grafico o imagen generada
    def Create_Punto(self, a, b, c, color_1, color_2, color_3):

        source = vtk.vtkSphereSource()
        source.SetCenter(a, b, c)
        source.SetRadius(2)
        source.SetPhiResolution(10)
        source.SetThetaResolution(10)
        # mapper
        mapper = vtk.vtkPolyDataMapper()
        if vtk.VTK_MAJOR_VERSION <= 5:
            mapper.SetInput(source.GetOutput())
        else:
            mapper.SetInputConnection(source.GetOutputPort())

        # actor
        actor = vtk.vtkActor()
        actor.SetMapper(mapper)
        actor.GetProperty().SetColor(color_1, color_2, color_3)
        actor.GetProperty().SetOpacity(0.8)
        # the render is a 3D Scene
        self.ren.AddActor(actor)


    #Funcion usada para el agregado durante el tiempo de ejecuci n
    def Agregar(self, event):
        dlg = wx.TextEntryDialog(self, 'Ingrese Posicion X', 'Posicion X')
```

```
        if dlg.ShowModal() == wx.ID_OK:
            self.Val_X = dlg.GetValue()
            dlg1 = wx.TextEntryDialog(self, 'Ingrese Posicion Y', 'Posicion Y')
            if dlg1.ShowModal() == wx.ID_OK:
                self.Val_Y = dlg1.GetValue()
                dlg2 = wx.TextEntryDialog(self, 'Ingrese Posicion Z', 'Posicion Z')
                if dlg2.ShowModal() == wx.ID_OK:
                    self.Val_Z = dlg2.GetValue()
                else:
                    return
                dlg2.Destroy()
            else:
                return
            dlg1.Destroy()
        else:
            return
        dlg.Destroy()

        p = Point(float(self.Val_X), float(self.Val_Y), float(self.Val_Z))
        puntosG = []
        val = [False]
        punto_color = []
        self.arbol.insert(p, puntosG, val, punto_color)
        if val[0]:
            if len(puntosG) != 0:
                for occt in puntosG:
                    self.Create_Cubo(occt.cubo.x, occt.cubo.y, occt.cubo.z, occt.cubo.w,
                        occt.cubo.h, occt.cubo.p, occt.color.a, occt.color.b, occt.color.c, 0.1)
            self.Create_Punto(float(self.Val_X), float(self.Val_Y), float(self.Val_Z),
                punto_color[0].a, punto_color[0].b, punto_color[0].c)
            #print(str(punto_color[0].a) + " " + str(punto_color[0].b) + " " +
                str(punto_color[0].c))


    #Funcion usada para hacer consultas en el octree en tiempo de ejecucion
    def Consultar(self, event):
        dlg = wx.TextEntryDialog(self, 'Posicion X del Centro del rango a consultar: ',
            'Posicion X')
        if dlg.ShowModal() == wx.ID_OK:
            self.Val_X = dlg.GetValue()
            dlg1 = wx.TextEntryDialog(self, 'Posicion Y del Centro del rango a consultar:',
                'Posicion Y')
            if dlg1.ShowModal() == wx.ID_OK:
                self.Val_Y = dlg1.GetValue()
                dlg2 = wx.TextEntryDialog(self, 'Posicion Z del Centro del rango a consultar:',
                    'Posicion Z')
                if dlg2.ShowModal() == wx.ID_OK:
                    self.Val_Z = dlg2.GetValue()
                    dlg4 = wx.TextEntryDialog(self, 'Longitud del rango a preguntar',
                        'Perimetro')
                    if dlg4.ShowModal() == wx.ID_OK:
                        self.Tamanho_Cubo = dlg4.GetValue()
                    else:
                        return
                else:
                    return
                dlg2.Destroy()
            else:
                return
            dlg1.Destroy()
        else:
            return
        dlg.Destroy()


        answer = []
        if self.actor1==None:
            self.actor1=self.Create_Cubo(float(self.Val_X), float(self.Val_Y),
                float(self.Val_Z), float(self.Tamanho_Cubo)/2, float(self.Tamanho_Cubo)/2,
                float(self.Tamanho_Cubo)/2, 0.0, 0.085, 0.532,0.5)
            newP=Rectangulo(float(self.Val_X), float(self.Val_Y), float(self.Val_Z),
                float(self.Tamanho_Cubo)/2, float(self.Tamanho_Cubo)/2,
                float(self.Tamanho_Cubo)/2)
            self.arbol.query(newP, answer)
            print(len(answer))
            for p in answer:
                print("p("+str(p.x)+" , "+ str(p.y)+" , "+str(p.z)+")")
            respuesta = "Cantidad de puntos en el rango indicado: " + str(len(answer))
            mostrar_respuesta = wx.MessageBox(respuesta, "Respuesta", wx.OK)
        else:
            self.delete_Cubo(self.actor1)
            self.actor1=None
            self.actor1=self.Create_Cubo(float(self.Val_X), float(self.Val_Y),
                float(self.Val_Z), float(self.Tamanho_Cubo)/2, float(self.Tamanho_Cubo)/2,
                float(self.Tamanho_Cubo)/2, 0.0, 0.085, 0.532,0.5)
            newP=Rectangulo(float(self.Val_X), float(self.Val_Y), float(self.Val_Z),
                float(self.Tamanho_Cubo)/2, float(self.Tamanho_Cubo)/2,
                float(self.Tamanho_Cubo)/2)
            self.arbol.query(newP, answer)
            print(len(answer))
            for p in answer:
```

```
            print("p("+str(p.x)+" , "+ str(p.y)+" , "+str(p.z)+")")
        respuesta = "Cantidad de puntos en el rango indicado: " + str(len(answer))
        mostrar_respuesta = wx.MessageBox(respuesta, "Respuesta", wx.OK)



    def Salir(self, event):
        self.Close()

# Inicializacion del programa con la interfaz grafica
app = wx.App()
frame = MostrarOctree(None, -1)
frame.Show()
app.MainLoop()
```

6. Compilación y ejecución del programa

6.1. Requisitos

- python v3.6
- libreria wxpython
- vtk v8.1.2

6.2. Ejecución

Para la ejecucion del programa ejecutar la siguiente linea de comando:

Listing 2: linea de comando

```
python octree.py
```

6.3. Resultados

Al momento de ejecutar el programa puede que se redimensione ocasionando algunos problemas en la generacion del gráfico como se muestra en la Figura 1

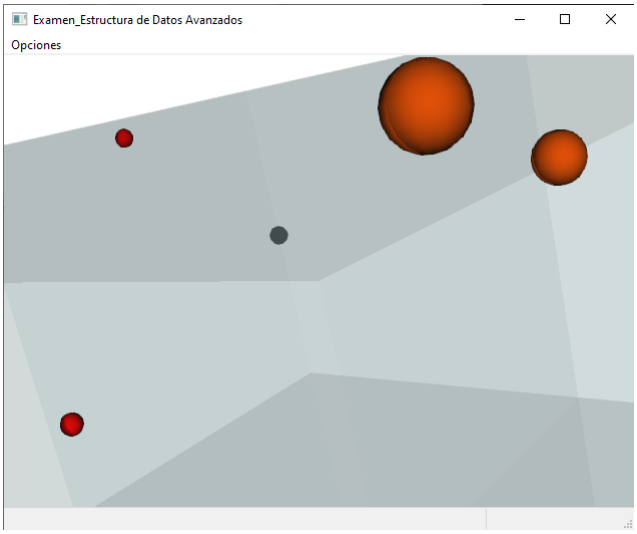


Figura 1: muesta del error de redimension

Para poder solucionar ello, lo recomendable es ejecutar la linea de comandos listing 2 una cierta cantidad de veces hasta que genere correctamente como en la Figura 2

Figura 2: Octree generado con 20 elementos

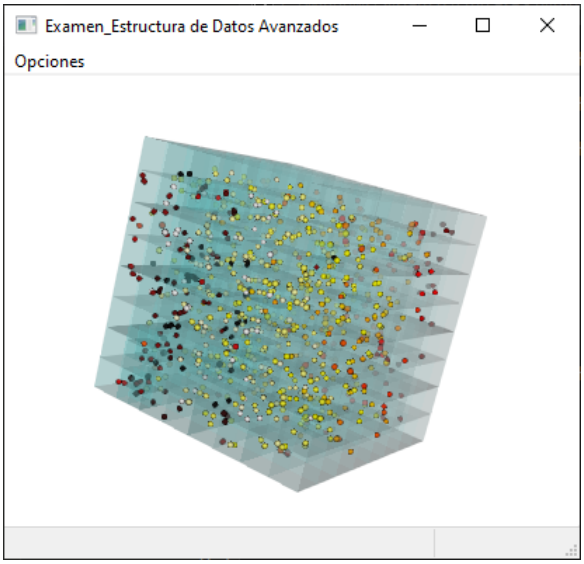


Figura 3: Octree generado con 1000 puntos

Después de ello nosotros podemos rotar el octree para ver si se ha graficado las divisiones correctamente como se muestra en Figura 4,Figura 5.

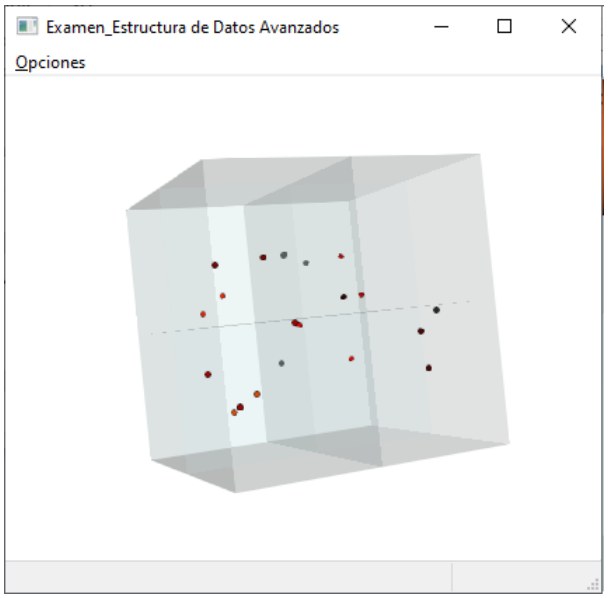


Figura 4: Octree al momento de rotar

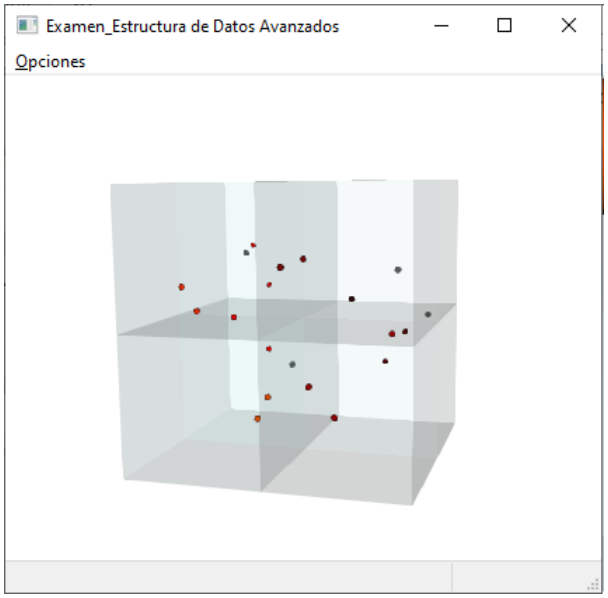


Figura 5: perspectiva octree

Y una vez visto que funciona correctamente procedemos a las opciones puestas en la barra de menu Figura 6. Estas operaciones constan de:

- agregar: Usado para la inserción de puntos
- consultar: Usado para saber cuantos puntos hay en el minicubo creado dentro del octree
- salir: Usado para cerrar el programa

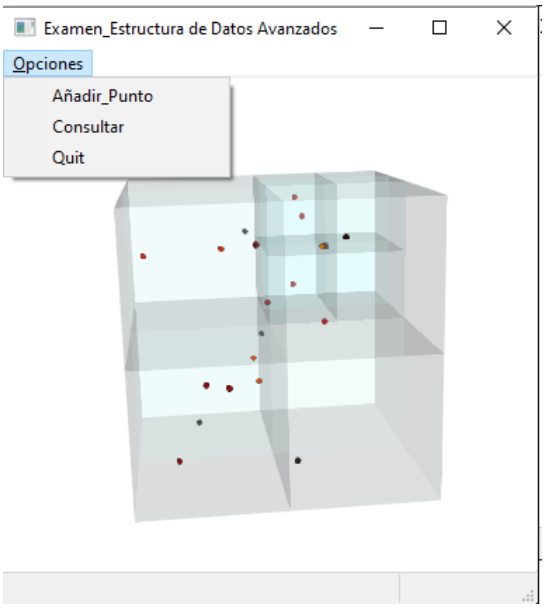


Figura 6: Barra de opciones

La insercion de un punto y la visualización en tiempo de ejecución, esto se puede aprecion en Figura 7, Figura 8, Figura 9, Figura 10

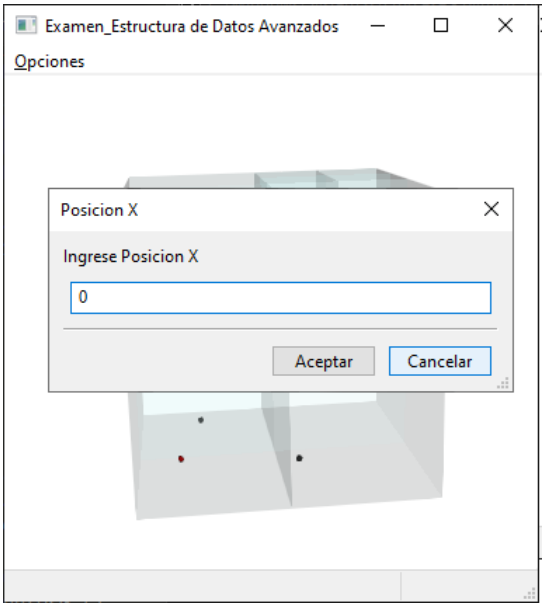


Figura 7: Insercion de la coordenada x de un punto

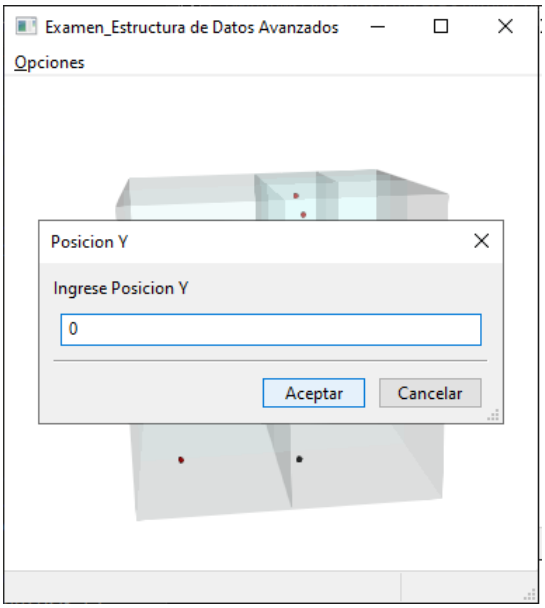


Figura 8: Insercion de la coordenada y de un punto

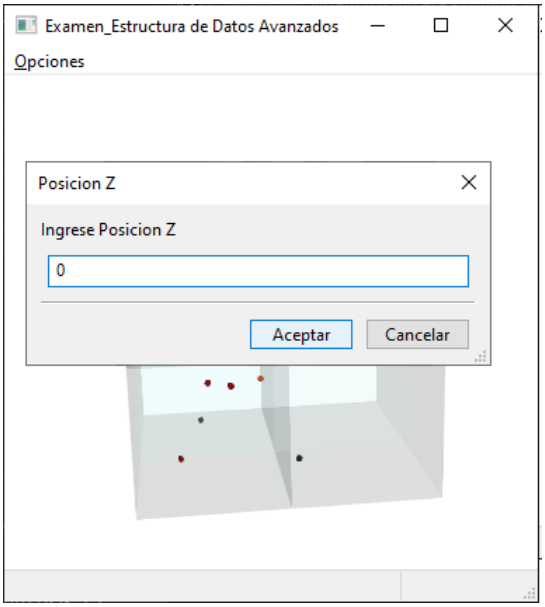


Figura 9: Insercion de la coordenada z de un punto

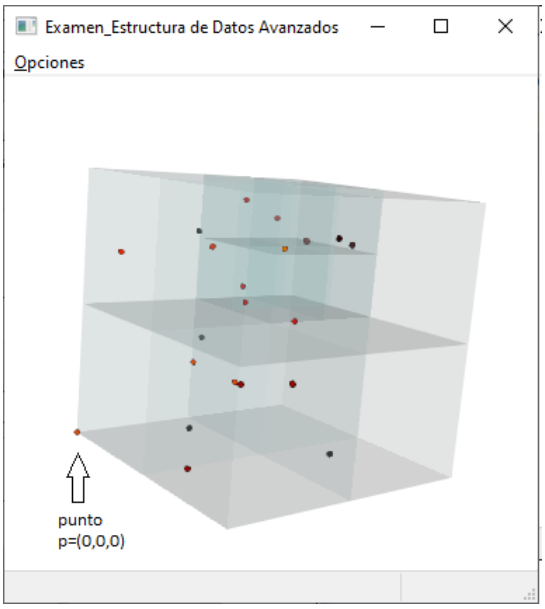


Figura 10: Punto (0,0,0) ingresado

Una vez realizado la insercion procedemos a hacer la consulta en tiempo real como se muestra en Figura 11, Figura 13

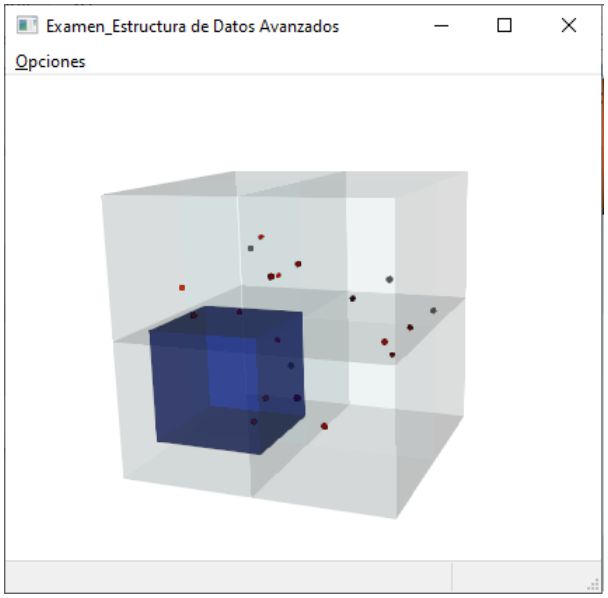


Figura 11: Consulta en tiempo real con 6 puntos en el rango del minicubo

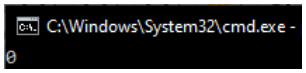


Figura 12: 0 puntos obtenidos

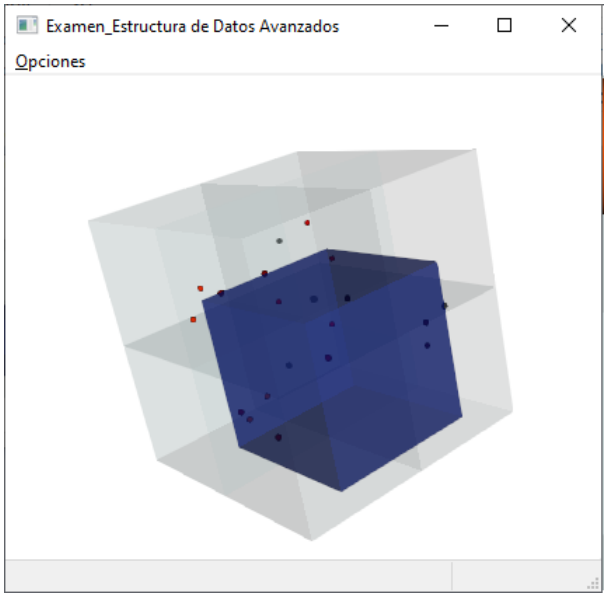


Figura 13: Consulta en tiempo real con 6 puntos en el rango del minicubo

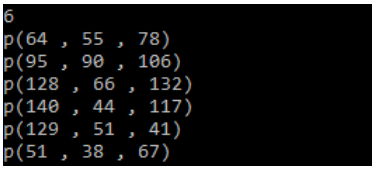


Figura 14: 6 puntos obtenido

7. Repositorio

El codigo se encuentra en el siguiente repositorio github donde esta todo el codigo, mas la capturas presentes en el documento