Universidad Nacional de San Agustín
Facultad de Producción y servicios
Escuela profesional de Ciencia de la Computación

OcTree

Informe del primer examen parcial

Integrantes: Angelo Perez Rodriguez
Renato Cespedes Fuentes
Josnick Chayña Batallanes
Rodrigo Mamani Sucacahua
Shirley Oxa Cacya
Profesor: VICENTE ENRIQUE MACHACA ARCEDA
Fecha de realización: 15 de octubre de 2020
Fecha de entrega: 15 de octubre de 2020
Arequipa, Perú

Índice de Contenidos

Lista de Figuras

Lista de Códigos

1. Introducción

El siguiente informe tiene la finalidad de presentar la estructura de datos OcTree, su implementación, así como sus resultados.Partiremos detallando que es la estructura de datos para tener un mejor entendimiento, se usara el lenguaje de programación Python y la librería VTK.

1.1. Requerimientos

- **Python:**es un lenguaje de programación de código abierto, orientado a objetos, muy simple y fácil de entender. Tiene una sintaxis sencilla que cuenta con una vasta biblioteca de herramientas, que hacen de Python un lenguaje de programación único.
- **VTK:** El Kit de herramientas de visualización es un sistema de software libre, libremente disponible para la realización de gráficos 3D por computadora, procesamiento de imagen y visualización. VTK consiste en una biblioteca de clases de C++ y varias capas de interfaz interpretadas como Tcl/Tk, Java, y Python.

1.2. Octree

Un octree es una estructura de datos en árbol en la que cada nodo interno tiene exactamente ocho hijos. Los octrees se usan con mayor frecuencia para dividir un espacio tridimensional subdividiéndolo recursivamente en ocho octantes. Los octrees son el análogo tridimensional de los quadtree. Los octrees a menudo se usan en gráficos 3D y motores de juegos 3D.

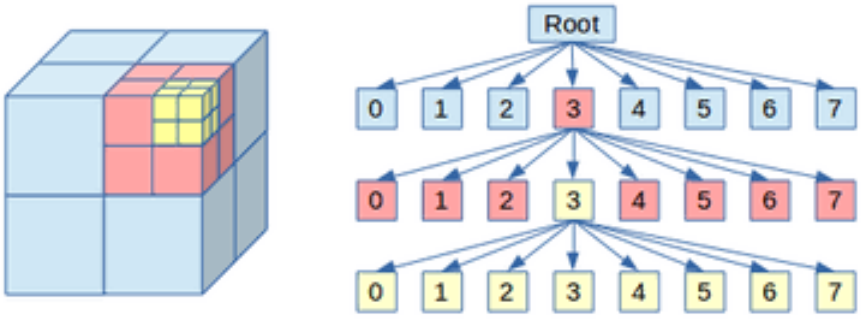

Figura 1: Representación gráfica de un Octree.

2. Implementación de Octree

Primero implementaremos las estructuras básicas de un Octree que son los puntos y los rectángulos o cubos.
Para poder implementar un punto, tenemos que tener en cuenta las dimensiones x,y,z, estas serán las coordenadas que nos permitirán posicionar un punto en nuestro Octree.

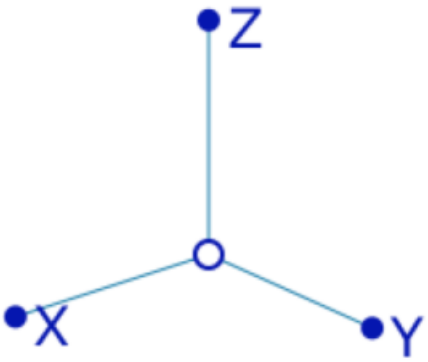

Figura 2: Punto en un plano tridimensional.

Por lo tanto para realizar los rectángulos se adicionara el atributo de profundidad, ancho y altura.

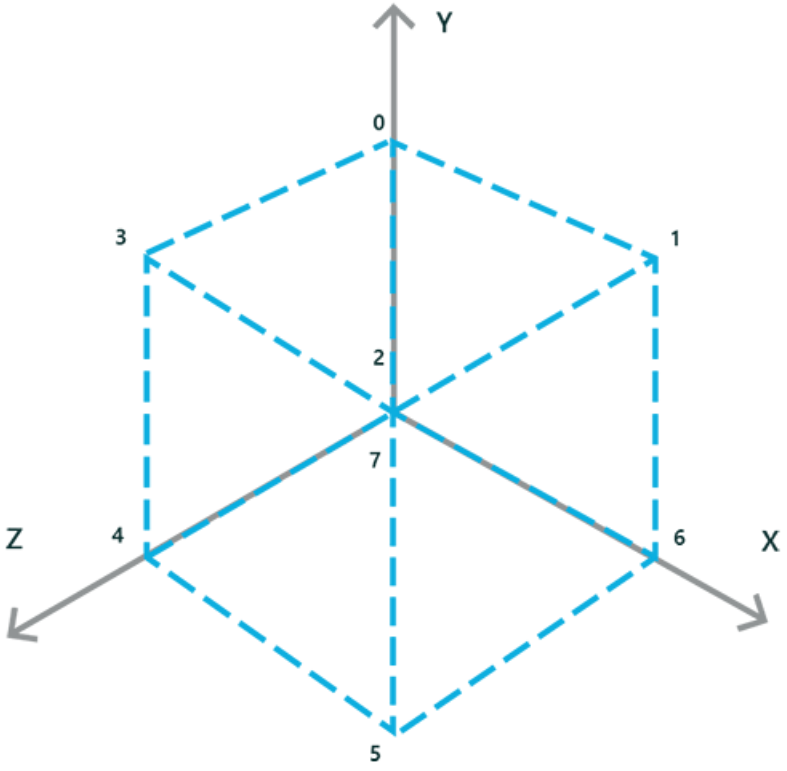

Figura 3: Cubo en un plano tridimensional.

2.1. Class Point y class Rect

Se muestra la implementación de los puntos con los que se trabajara y la clase rectángulo que funcionara de manera visual como un cubo. Notamos además que ahora a diferencia de nuestro Quadtree, trabajmos con 3 dimensiones, x,y,z , y en nuestro cubo se trabajara con w,h,f (weight,high,front) que serán la forma de división continua de nuestro Octree.
El siguiente trozo de código muestra la implementación de las clases **class Point** y **class Rectangle**, en el código1):

Código 1: *octree.py*

```
class Point:
  def ___init___ (self, x, y, z, userData=None): #Creacion del punto con eje x y z
    self.x = x
    self.y = y
    self.z = z
    self.flag = False      #
    self.userData = userData #Dato

class Rectangle:
  def ___init___ (self ,x,y,z,w,h,f):
    self.x = x  #center
    self.y = y
    self.z = z
    self.w = w  #half width
    self.h = h  #half height
    self.f = f  #frente

# verifica si este objeto contiene un objeto Punto
  def contains (self,point):
    if (point.x <= self.x+self.w and point.x >= self.x-self.w and point.y <= self.y+self.h  and
        point.y >= self.y-self.h and point.z <= self.z + self.f and point.z >= self.z-self.f):
        #point.flag=True
      print(point.x)
      print(point.y)
      print(point.z)
      return True
    return False

# verifica si este objeto se intersecta con otro objeto Rectangle
  def intersects (self,range):
    if(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.f >self.z + self.f or range.z + range.f < self.z - self.f):
      return False
    return True


```

2.2. Class Octree

Se muestra la implementación del Octree definiendo el perímetro o margen del Octree principal como paso base, también se define la capacidad máxima de cada cubo o sub-Octree construido, seguido de esto se define un array o vector de puntos el cual almacenara los puntos ingresados en el Octree, finalmente nuestro Octree tendra una especie de flag llamado divided el cual cambiara a verdadero si y solo si, se comprueba que el Octree fue dividido.

Código 2: Implementación del archivo *octree.py*

```
class OcTree:
  def ___init___(self,boundary , n ):
    self.boundary = boundary #Rectangle
    self.capacity = n #capacidad máxima de cada cuadrante
    self.points = [] #vector, almacena los puntos a almacenar
    self.divided = False
    self.hijos = []
```

2.2.1. Class Subdivide

Nuestra función de subdivide no es otra que la misma del quadtre, con estas diferencias:

- Se crea una tercera dimensión y también una variable que definirá el tamaño de la mitad del octree actual
- Se asigna los 8 sub Octree en las diferentes posiciones correspondientes, noreste , noroeste,sureste y suroeste.
- Una vez dividido el Octree en 8 , este cambia su flag o bandera de dividido a verdadero , para poder corroborar que este cubo ya fue divido.

Código 3: Implementación del archivo *octree.py*

```
#divide nuestro quadtree en 4 quadtrees
  def subdivide (self):
    x = self.boundary.x
    y = self.boundary.y
    z = self.boundary.z
    w = self.boundary.w/2
    h = self.boundary.h/2
    f = self.boundary.f/2

    no = Rectangle(x-w,y-h,z-f,w,h,f)
    ne = Rectangle(x+w,y-h,z-f,w,h,f)
    so = Rectangle(x-w,y+h,z-f,w,h,f)
    se = Rectangle(x+w,y+h,z-f,w,h,f)

    fno = Rectangle(x-w,y-h,z+f,w,h,f)
    fne = Rectangle(x+w,y-h,z+f,w,h,f)
    fso = Rectangle(x-w,y+h,z+f,w,h,f)
```

```
18      fse = Rectangle(x+w,y+h,z+f,w,h,f)
19
20      sonNO = OcTree(no, self.capacity)
21      sonNE = OcTree(ne, self.capacity)
22      sonSO = OcTree(so, self.capacity)
23      sonSE = OcTree(se, self.capacity)
24      fsonNO = OcTree(fno, self.capacity)
25      fsonNE = OcTree(fne, self.capacity)
26      fsonSO = OcTree(fso, self.capacity)
27      fsonSE = OcTree(fse, self.capacity)
28      self.hijos.append(sonNO)
29      self.hijos.append(sonNE)
30      self.hijos.append(sonSO)
31      self.hijos.append(sonSE)
32      self.hijos.append(fsonNO)
33      self.hijos.append(fsonNE)
34      self.hijos.append(fsonSO)
35      self.hijos.append(fsonSE)
36      self.divided = True
```

2.2.2. class insert

En esta clase se verifica primero si no se inserto nada en un Octree , después se verificara si es permitido ingresar un punto sin que se subdivida nuestro Octree,si aun no se a subdividido nuestro Octree (verificado por nuestro flag divided) se aplica una subdivisión y finalmente si no es permitido ingresar puntos, se inserta el punto en los hijos subdivididos.

Código 4: Implementación del archivo *octree.py*

```
1   def insert(self,p):
2     if(self.boundary.contains(p)==False):
3       return
4     if (len(self.points) < self.capacity):
5       self.points.append(p)
6     else:
7       if(self.divided==False):
8         self.subdivide()
9       for i in range(0,8):
10        self.hijos[i].insert(p)
```

2.2.3. class query

En esta clase verificamos con cuales cubos el rango se intersecta, si esto pasa entonces verificamos si el rango contiene al los puntos que se encuentran en ese subcubo, si este subcubo esta dividido entonces hacemos una llamada recursiva en los hijos de este.

Código 5: Implementación del archivo *octree.py*

```
1   def query(self,ranges, found):
2     if(self.boundary.intersects(ranges)==False):
```

```
      return
    for i in self.points:
      if(ranges.contains(i)):
        found.append(i)
    if(self.divided):
      for i in range(0,8):
        self.hijos[i].query(ranges,found)
```

Con esta función vamos a mostrar las divisiones.

Código 6: Implementación del archivo *octree.py*

```
def show(self):
    rect (self.boundary.x,self.boundary.y,self.boundary.z,self.boundary.w,self.boundary.h,self.
    ↪ boundary.f)

    if(self.divided):
      self.sonNO.show()
      self.sonNE.show()
      self.sonSO.show()
      self.sonSE.show()
      self.fsonNO.show()
      self.fsonNE.show()
      self.fsonSO.show()
      self.fsonSE.show()

    for i in self.points:
      point(i.x,i.y,i.z)
```

2.3. Implementación con VTK

En esta parte del codigo vamos a crear nuestro cubo que sera el octree, y el punto como una esfera.

Código 7: Implementación del archivo *octree.py*

```
  def rect(self,ren,x,y,z,xx,yy,zz,flag=0):
    colors = vtk.vtkNamedColors()


    cube = vtk.vtkCubeSource()
    cube.SetBounds(x-xx,x+xx,y-yy,y+yy,z-zz,z+zz)
    cube.Update()
    cubeMapper = vtk.vtkPolyDataMapper()
    cubeMapper.SetInputData(cube.GetOutput())
    cubeActor = vtk.vtkActor()
    cubeActor.SetMapper(cubeMapper)
    if(flag==1):
      cubeActor.GetProperty().SetColor(colors.GetColor3d("red"))
      cubeActor.GetProperty().SetOpacity(0.5)
    else:
      cubeActor.GetProperty().SetColor(colors.GetColor3d("green"))
```

```
17          cubeActor.GetProperty().SetOpacity(0.3)
18      ren.AddActor(cubeActor)
19      ren.ResetCamera()
20      ren.GetActiveCamera().Azimuth(30)
21      ren.GetActiveCamera().Elevation(30)
22      ren.ResetCameraClippingRange()
23      ren.SetBackground(colors.GetColor3d("white"))
24
25  def point(self,ren,x,y,z,flag=0):
26      colors = vtk.vtkNamedColors()
27
28
29
30      sphereSource = vtk.vtkSphereSource()
31      sphereSource.SetCenter(x, y, z)
32      sphereSource.SetRadius(10)
33
34      mapper = vtk.vtkPolyDataMapper()
35      mapper.SetInputConnection(sphereSource.GetOutputPort())
36      actor = vtk.vtkActor()
37
38      actor.SetMapper(mapper)
39      if(flag==1):
40          actor.GetProperty().SetColor(colors.GetColor3d("red"))
41      else:
42          actor.GetProperty().SetColor(colors.GetColor3d("yellow"))
43      ren.ResetCamera()
44      ren.GetActiveCamera().Azimuth(30)
45      ren.GetActiveCamera().Elevation(30)
46      ren.ResetCameraClippingRange()
47      ren.SetBackground(colors.GetColor3d("white"))
48
49      ren.AddActor(actor)
50
51    def show(self,ren):
52
53      self.rect (ren,self.boundary.x , self.boundary.y , self.boundary.z , self.boundary.w, self.boundary.h
        ↪ ,self.boundary.f)
54      if(self.divided):
55          for i in range(0,8):
56              self.hijos[i].show(ren)
57
58      for i in self.points:
59          self.point(ren,i.x,i.y,i.z)
```

2.4. funcion setup

En nuestra función principal, vamos a crear nuestra ventana, nuestro respectivo octree y vamos a realizar la inserción de datos

Código 8: Implementación del archivo *octree.py*

```
def setup():
  ren = vtk.vtkRenderer()
  renWin = vtk.vtkRenderWindow()
  renWin.SetWindowName("OcTree")
  renWin.AddRenderer(ren)
  iren = vtk.vtkRenderWindowInteractor()
  iren.SetRenderWindow(renWin)
  renWin.SetSize(600, 600)

  iren.Initialize()
  cubo = Rectangle(200,200,200,200,200,200)
  octcubo = OcTree(cubo,4)
  for i in range(100):
    p= Point(random.random()*400,random.random()*400,random.random()*400)
    octcubo.insert(p)
  octcubo.show(ren)
  iren.Start()

setup()
```

3. Resultados

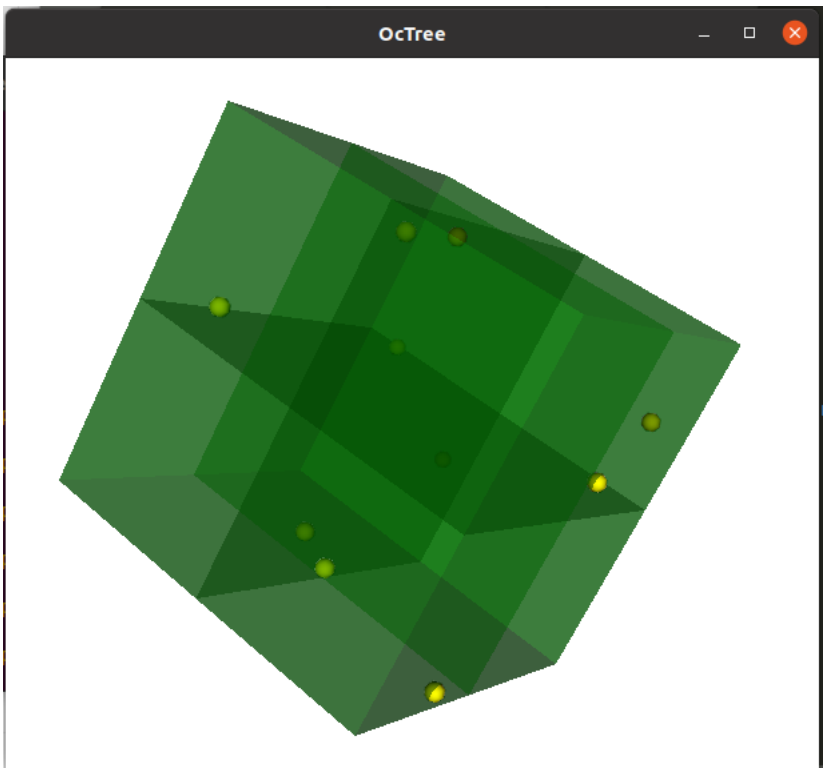

Figura 4: Octree con 10 puntos insertados.

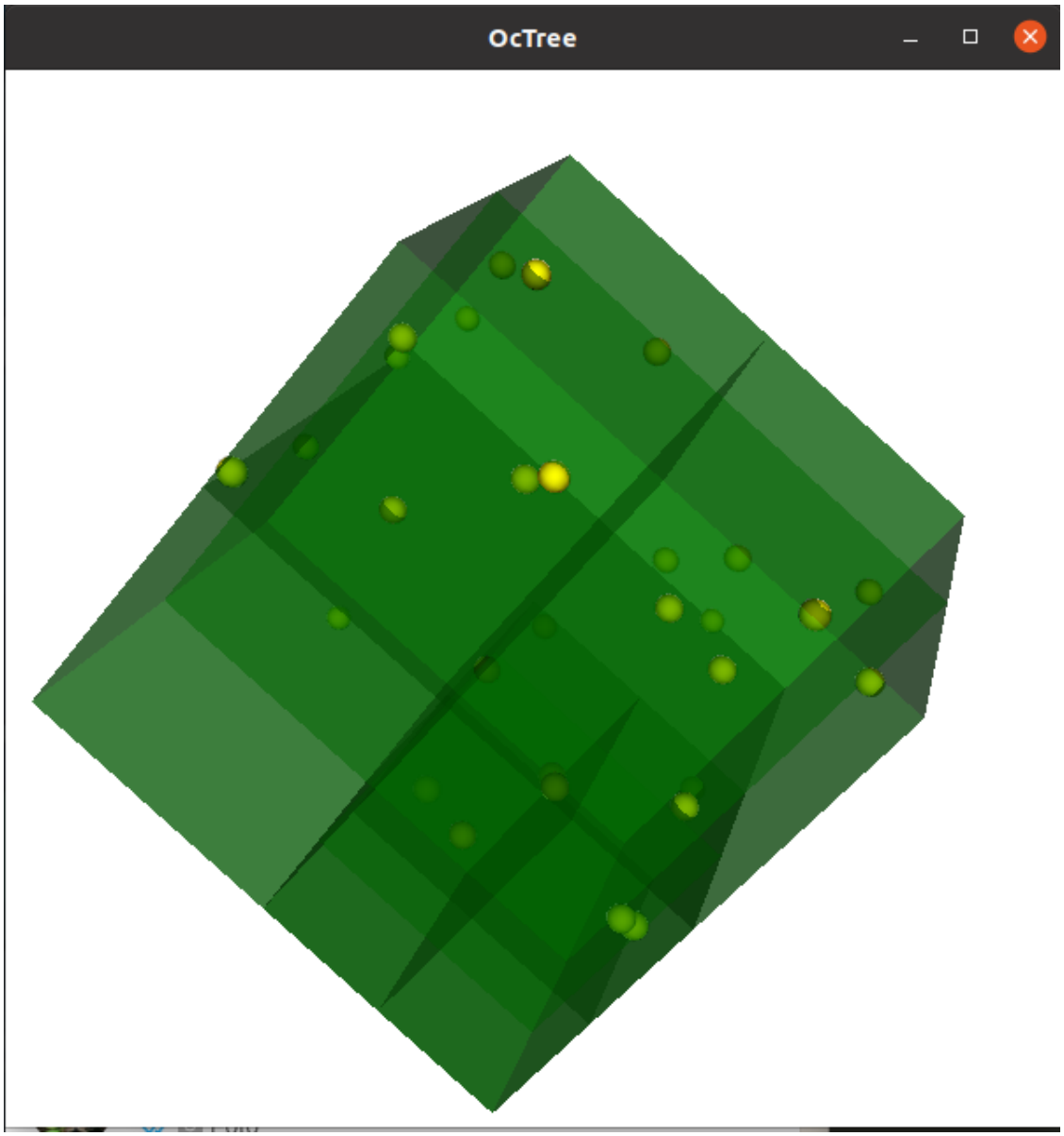

Figura 5: Octree con 30 puntos insertados.

3.1. Nota

La siguiente implementación se encuentra en un repositorio de Github, dirigíais a Referencias [2].

4. Conclusiones

- En los resultados podemos observar el OCtree como un cubo que es el padre o la raiz y este contiene a sus nodos o hijos.

- La idea de utilizar un octree para encontrar los nodos intransitables es para aprovechar la estructura y definición del octree para hacer un menor numero 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 seran los nodos no transitables.

Referencias

[1] Template Informe en LaTeX. *¡Revisa el manual online de este template!*
https://latex.ppizarror.com/Template-Informe/

[2] *¡Repositorio de Github*
https://github.com/RenatoCespedes/Estructura-de-Datos-Avanzado/tree/master/PrimerExamenParcial