Universidad Nacional de San Agustín

Escuela Profesional de
Ciencia de la Computación

PRIMER EXÁMEN DE ESTRUCTURA DE DATOS AVANZADO - OCTREE

PRIMER EXÁMEN DE ESTRUCTURA DE DATOS AVANZADO - OCTREE

Alumno :
Nicoll del Rosario Aza Mamani
Miguel Alexander Herrera Cooper
Profesor:
Vicente Enrique Machaca Arceda

2 de octubre de 2019

Índice

1. Introducción

Un octree es una estructura de datos en árbol en la que cada nodo interno tiene exactamente ocho hijos . Octrees son los más utilizados para dividir un espacio tridimensional por recursivamente subdividir en ocho octantes. Los octreos son el análogo tridimensional de los cuadrúpedos . El nombre está formado por oct + tree , pero tenga en cuenta que normalmente se escribe .ºctree çon solo una "t". Los octrees a menudo se usan en gráficos 3D y motores de juegos 3D .

Cada nodo en un octree subdivide el espacio que representa en ocho octantes . En un octree de región de punto (PR), el nodo almacena un punto tridimensional explícito , que es el çentro"de la subdivisión para ese nodo; el punto define una de las esquinas para cada uno de los ocho hijos. En un octree basado en matriz (MX), el punto de subdivisión es implícitamente el centro del espacio que representa el nodo. El nodo raíz de un octree PR puede representar un espacio infinito; El nodo raíz de un árbol MX debe representar un espacio limitado finito para que los centros implícitos estén bien definidos. Tenga en cuenta que los octreos no son lo mismo que los árboles k -d : los árboles k -d se dividen a lo largo de una dimensión y los octrees se dividen alrededor de un punto. También k? ¿Cómo? '. K-d los árboles son siempre binarios, que no es el caso de los octrees. Al utilizar una búsqueda de profundidad, los nodos deben atravesarse y solo deben verse las superficies requeridas.

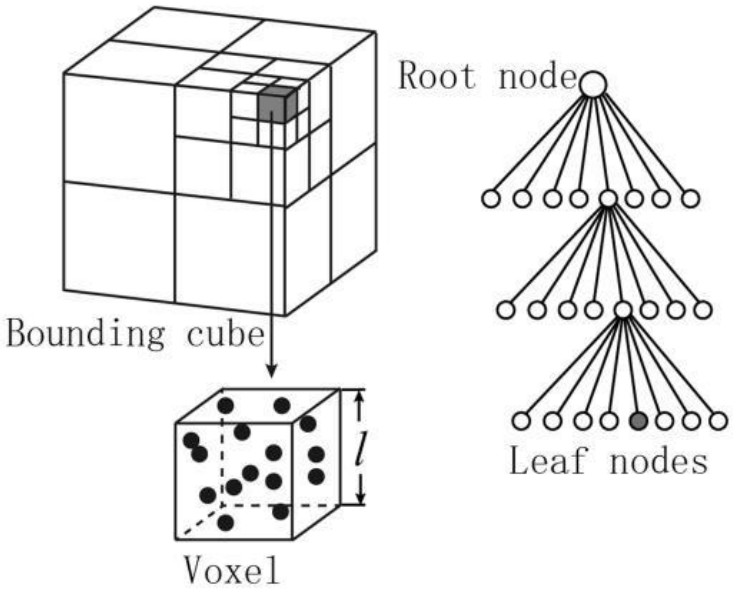
1.1. Usos Comunes

- Nivel de representación de detalle en gráficos 3D por computadora.
- Indexación espacial.
- Búsqueda de vecino más cercano.
- Detección eficiente de colisiones en tres dimensiones.
- Ver el sacrificio de Frustum.
- Método rápido multipolo.
- Rejilla no estructurada.
- Análisis de elementos finitos.
- Escaso voxel octree.
- Estimación del estado.
- Establecer estimación.

1.2. La subdivisión del Espacio

Al tratar de describir, un objeto mediante todo los voxels que le pertenecen, cuando cada cubo puede ser descrito por sus esquinas,se generan grandes listas de datos. La enumeración completa de los modelos hechos por descomposición tiene muchas ventajas, tales como su simplicidad, su generalidad y permiten un gran número de algoritmos. Sin embargo, el consumo de memoria es alto; se puede cambiar la división regaular del espacio por algo más eficiente, una división adaptativa.

A este esquema de trabajo basado en la subdivisión del espacio se logra por simple observación de la rejilla descriptiva de la enumeración completa de un espacio 3D, donde muchas veces un cubo blanco tiene de vecinos otros cubos del mismo tipo "1". Al observar la combinación y codificarla en modelos de datos comparativamente similares. podemos reducir la cantidad de memoria utilizada por dicha rejilla.



1.3. Algoritmo de los Árboles Octales

onsiste en la metodología sistemática para generar y desplegar el modelo, donde la enumeración de atrás hacia adelante permite que los nodos sean listados, desplegando del más lejano al más cercano, desde un punto de VISIÓN, luego sus tres vecinos, en cualquier orden y así recursivamente.

Un modelador de sólidos con las funciones completas debe incluir los algoritmos que realicen las tareas siguientes:

- Generador del árbol, crea el octree desde las primitivas parametrizar o desde

otros tipos de modelos geométricos.

- Operaciones de Conjunto, que pueden llevarse acabo entre dos octrees, con un espacio idéntico de interés, con el cálculo del octree resultante dado por la unión, intersección o diferencia de conjunto de los argumentos.
- Operaciones geométricas, que se hacen sobre un octree y calcular un nuevo octree como resultado de una traslación, rotación o escalamiento de los objetos modelados. Otro tipo de operación geométrica calcula el nuevo octree obtenido a partir de operaciones como laas transformaciones de perspectivas. Algunos de estos problemas requieren un algoritmo complejo, muchas veces no intuitivo. La traslación de un octree es una operación bastante compleja y difícil.
- Procedimiento de análisis, calcula ciertas propiedades del objeto tales como su volumen o área de superficie del octree; un procedimiento de componentes conectado, este análisis es una operación muy compleja.
- Generador de desplegado, crea la imgen gráfica del obejto modelado por el octree.

2. Implementación de Octree

- Implementación del OCTREE en python y vtk

```
    import vtk

puntos=[]#Arrays de Puntos
actors=[]#Puntos mapeados

def rect(x,y,z,w,h,p,color):
    cube=vtk.vtkCubeSource()
    cube.SetXLength(w)
    cube.SetYLength(h)
    cube.SetZLength(p)
    cube.SetCenter(x,y,z)

    cubeMapper=vtk.vtkPolyDataMapper()
    cubeMapper.SetInputConnection(cube.GetOutputPort())

    cubeActor=vtk.vtkActor()
    cubeActor.GetProperty().SetColor(color[0],color[2],color[1])
    cubeActor.GetProperty().SetOpacity(0.5)
```

```
19      cubeActor.SetMapper(cubeMapper)
20
21      actors.append(cubeActor)
22
23  def punto(x,y,z,color):
24      cube=vtk.vtkSphereSource()
25      cube.SetRadius(6.0)
26      cube.SetCenter(x,y,z)
27
28      cubeMapper=vtk.vtkPolyDataMapper()
29      cubeMapper.SetInputConnection(cube.GetOutputPort())
30
31      cubeActor=vtk.vtkActor()
32      cubeActor.GetProperty().SetColor(color[0],color[0],color[2])
33      cubeActor.SetMapper(cubeMapper)
34
35      puntos.append(cubeActor)
36
37  def Pantalla():
38      ren = vtk.vtkRenderer()
39
40      for act in puntos:
41          ren.AddActor(act)
42
43      for act in actors:
44          ren.AddActor(act)
45
46      renWin = vtk.vtkRenderWindow()
47      renWin.AddRenderer(ren)
48      renWin.SetSize(600, 600)
49      iren = vtk.vtkRenderWindowInteractor()
50      iren.SetRenderWindow(renWin)
51      ren.SetBackground(0,0,0)
52      renWin.Render()
53      iren.Start()
54
55  class Point:
56      def __init__(self, x, y, z):
57          self.x = x
58          self.y = y
59          self.z = z
60
61  class Rectangulo:
62      def __init__(self, x, y, z, w, h, p):
63          self.x = x
```

```
64          self.y = y
65          self.z = z
66          self.w = w
67          self.h = h
68          self.p = p
69      def cotiene(self, point):
70          return 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.p and point.z <= self.z +
    self.p
71      def intersect(self, rango):
72          return not (rango.x-rango.w>self.x+self.w or rango.x+rango
    .w<self.x-self.w or rango.y-rango.h>self.y+self.h or rango.y+
    rango.h<self.y-self.h or rango.z-rango.p >self.z+self.p or
    rango.z+rango.p<self.z-self.p)
73
74
75  class Octree:
76      def __init__(self, perimetro, n,color=[0,0,1]):
77          self.perimetro = perimetro
78          self.capacidad = n
79          self.points = []
80          self.divided = False
81          self.color=color
82
83      def Subdividir(self):
84          x = self.perimetro.x
85          y = self.perimetro.y
86          z = self.perimetro.z
87          w = self.perimetro.w/2
88          h = self.perimetro.h/2
89          p = self.perimetro.p/2
90
91          noup = Rectangulo(x-w,y+h,z+p,w,h,p);
92          neup = Rectangulo(x+w,y+h,z+p,w,h,p);
93          soup = Rectangulo(x-w,y-h,z+p,w,h,p);
94          seup = Rectangulo(x+w,y-h,z+p,w,h,p);
95
96          nodw = Rectangulo(x-w,y+h,z-p,w,h,p);
97          nedw = Rectangulo(x+w,y+h,z-p,w,h,p);
98          sodw = Rectangulo(x-w,y-h,z-p,w,h,p);
99          sedw = Rectangulo(x+w,y-h,z-p,w,h,p);
100
101         self.sonNOup = Octree(noup,self.capacidad,[noup.x/254,noup
    .y/250,self.perimetro.p-noup.z/236])
```

```
102         self.sonNEup = Octree(neup,self.capacidad,[neup.x/254,neup
    .y/250,self.perimetro.p-neup.z/236])
103         self.sonSOup = Octree(soup,self.capacidad,[soup.x/254,soup
    .y/250,self.perimetro.p-soup.z/236])
104         self.sonSEup = Octree(seup,self.capacidad,[seup.x/254,seup
    .y/250,self.perimetro.p-seup.z/236])
105 
106         self.sonNOdw = Octree(nodw,self.capacidad,[nodw.x/254,nodw
    .y/250,self.perimetro.p-nodw.z/236])
107         self.sonNEdw = Octree(nedw,self.capacidad,[nedw.x/254,nedw
    .y/250,self.perimetro.p-nedw.z/236])
108         self.sonSOdw = Octree(sodw,self.capacidad,[sodw.x/254,sodw
    .y/250,self.perimetro.p-sodw.z/236])
109         self.sonSEdw = Octree(sedw,self.capacidad,[sedw.x/254,sedw
    .y/250,self.perimetro.p-sedw.z/236])
110 
111         self.divided = True
112 
113 
114 
115     def Insertar(self, point):
116         if not self.perimetro.cotiene(point):
117             return
118         if len(self.points) < self.capacidad:
119             self.points.append(point)
120         else:
121             if self.divided==False:
122                 self.Subdividir()
123             self.sonNOup.Insertar(point)
124             self.sonNEup.Insertar(point)
125             self.sonSOup.Insertar(point)
126             self.sonSEup.Insertar(point)
127             self.sonNOdw.Insertar(point)
128             self.sonNEdw.Insertar(point)
129             self.sonSOdw.Insertar(point)
130             self.sonSEdw.Insertar(point)
131 
132 
133 
134     def query(self,rango,found):
135         if(not self.perimetro.intersect(rango)):
136             return
137         for p in self.points:
138             if(rango.cotiene(p)):
139                 found.append(p)
```

```
140         if(self.divided == True):
141             self.sonNOup.query(rango,found)
142             self.sonNEup.query(rango,found)
143             self.sonSOup.query(rango,found)
144             self.sonSEup.query(rango,found)
145             self.sonNOdw.query(rango,found)
146             self.sonNEdw.query(rango,found)
147             self.sonSOdw.query(rango,found)
148             self.sonSEdw.query(rango,found)
149 
150     def Mostrar(self):
151         rect(self.perimetro.x,self.perimetro.y,self.perimetro.z,
    self.perimetro.w*2,self.perimetro.h*2,self.perimetro.p*2,self.
    color)
152         if(self.divided):
153             self.sonNOup.Mostrar()
154             self.sonNEup.Mostrar()
155             self.sonSOup.Mostrar()
156             self.sonSEup.Mostrar()
157             self.sonNOdw.Mostrar()
158             self.sonNEdw.Mostrar()
159             self.sonSOdw.Mostrar()
160             self.sonSEdw.Mostrar()
161         for p in self.points:
162             punto(p.x,p.y,p.z,self.color)
163 
164 from random import*
165 
166 
167 def main():
168     perimetro = Rectangulo (200,200,200,200,200,200)
169     ot= Octree(perimetro,8)
170     for i in range(0,77):
171         val1 = randrange(400)
172         val2 = randrange(400)
173         val3 = randrange(400)
174         p = Point(val1,val2,val3)
175         ot.Insertar(p)
176     ot.Mostrar()
177     rango = Rectangulo(randrange(200),randrange(200),randrange
    (200),randrange(100),randrange(100),randrange(100))
178     rect(rango.x,rango.y,rango.z,rango.w*3,rango.h*3,rango.p
    *3,[255,0,0])
179 
180     rango_punto= []
```

```
181     ot.query(rango,rango_punto)
182     for it in rango_punto:
183         punto(it.x,it.y,it.z,[255,0,0])
184     print(rango_punto)
185     Pantalla()
186 main()
187
```

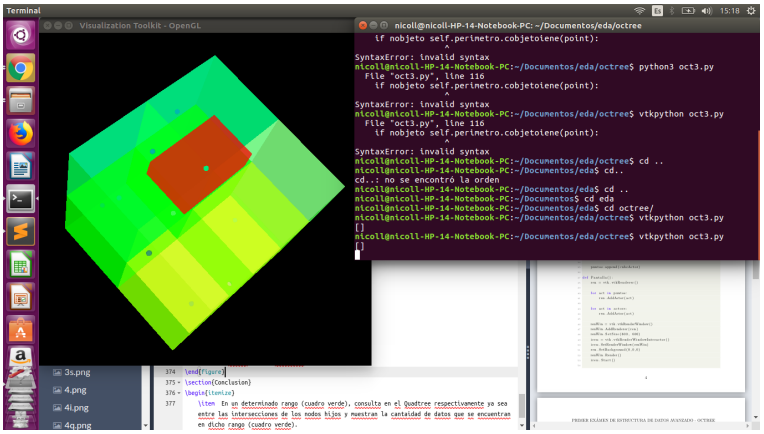


Figura 1: Prueba con 10 elementos

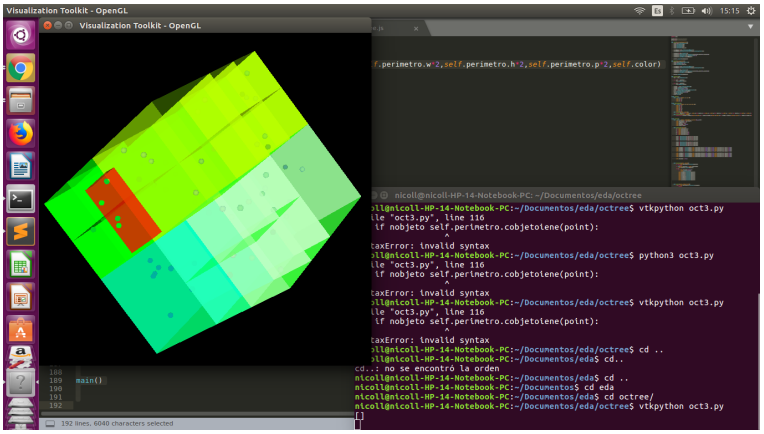


Figura 2: Prueba con 77 elementos

3. Conclusiones

- En un determinado rango (cuadro verde), consulta en el Quadtree respectivamente ya sea entre las intersecciones de los nodos hijos y muestran la cantaidad de datos que se encuentran en dicho rango (cuadro verde).
- Es una forma de búsqueda que se da al Quadtree que es un respectivo árbol

- Gráficamente podemos visualizar a un octree como un cubo perfecto que representa al nodo raíz el cual encierra toda la geometría de la escena. Cada nodo está subdividido en ocho cubos más pequeños que representan a sus nodos hijo los cuales son llamados octantes. Cada uno de estos cubos también es subdividido en otros ocho octantes cada uno y así sucesivamente hasta llegar a una condición de término a la que tipicamente se llega en el momento en que los octantes son de cierto tamaño o cuando no llegan a contener un cierto número mínimo de polígonos.
- Muchas de las operaciones que se pueden hacer, son ejecutadas de acuerdo al paradigma nodo-a-nodo, por ello, las propiedades integrales como volumen, centro de masa y momentos de inercia se pueden calcular por un simple algoritmo de árbol transversal.
- La idea de utilizar un octree para encontrar los nodos intransitables es para aprovechar la estructura y definición del octree 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 seran los nodos no transitables.
- Conversion and Integration of Boundary Representations with Octrees*

4. Referencias

- https://www.gamedev.net/articles/programming/general-and-gameplay-programming/intro to-octrees-r3529/
- Martti Mäntyla, An introduction to solid modeling, p. 68.
- Universidad de Oviedo - Esquemas de Modelado.
- http://catarina.udlap.mx/u$_d l_a$/*tales*/*documentos*/*msp*/*mora*$_{lm}$*a*/*capitulo*2.*pdf*