Universidad Nacional de San Agustín de Arequipa

Estructuras de Datos Avanzadas

OcTree

Integrantes del GRUPO 2A:
- Portocarrero Espirilla, Diego Armando
- Coayla Zuñiga, Gonzalo Eduardo
- Jara Huilca, Arturo Jesús
- Pucho Zevallos, Paul Kelvin

Profesor :
- Machaca Arceda Vicente Enrique

15 de octubre de 2020

Índice

1. Introducción

El trabajo fue desarrollado en el lenguaje Python utilizando la libreria de visualización VTK

1.1. VTK

VTK es un sistema de software de código abierto para procesamiento de imágenes, gráficos 3D, renderizado y visualización de volumen. VTK incluye muchos algoritmos avanzados (por ejemplo, reconstrucción de superficies, modelado implícito, diezmado) y técnicas de renderizado (por ejemplo, renderizado de volumen acelerado por hardware, control LOD).

1.2. Repositorio

https://github.com/khannom/EDA-Grupo/tree/master/Octree

2. Código Fuente

2.1. Clases Point y Cube

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

class Cube:
    def __init__(self, x, y, z, w, h, d):
        self.x = x
        self.y = y
        self.z = z
        self.w = w
        self.h = h
        self.d = d

    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.d and point.z <= self.z + self.d):
            return True
        return False

    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.d > self.z + self.d or
            range.z + range.d < self.z - self.d):
            return False
        return True
```

2.2. Clase OcTree

```
class Octree:
    def __init__(self, volumen, n, color):
        self.volumen = volumen
        self.capacity = n
        self.points = []
        self.divided = False
        self.color = color

    def subdivide(self):
        x = self.volumen.x
        y = self.volumen.y
        z = self.volumen.z
        w = self.volumen.w
        h = self.volumen.h
        d = self.volumen.d

        color = self.color

        nefront = Cube(x + w / 2, y + h / 2, z + d / 2, w / 2, h / 2, d / 2)
        self.northeastfront = Octree(
            nefront, self.capacity, color)

        nwfront = Cube(x - w / 2, y + h / 2, z + d / 2, w / 2, h / 2, d / 2)
        self.northwestfront = Octree(
            nwfront, self.capacity, (boundedSum(color[0], 0.2),
            color[1], color[2]))

        sefront = Cube(x + w / 2, y - h / 2, z + d / 2, w / 2, h / 2, d / 2)
        self.southeastfront = Octree(
            sefront, self.capacity, (color[0],
            boundedSum(color[1], 0.2), color[2]))

        swfront = Cube(x - w / 2, y - h / 2, z + d / 2, w / 2, h /
```

```
        2, d / 2)
        self.southwestfront = Octree(
            swfront, self.capacity, (color[0], color[1],
            boundedSum(color[2], 0.2)))

        neback = Cube(x + w / 2, y + h / 2, z - d / 2, w / 2, h /
        2, d / 2)
        self.northeastback = Octree(
            neback, self.capacity, (boundedSum(color[0], 0.5),
            boundedSum(color[1], 0.5), color[2]))

        nwback = Cube(x - w / 2, y + h / 2, z - d / 2, w / 2, h /
        2, d / 2)
        self.northwestback = Octree(
            nwback, self.capacity, (color[0], boundedSum(color[1],
            0.7), boundedSum(color[2], 0.5)))

        seback = Cube(x + w / 2, y - h / 2, z - d / 2, w / 2, h / 2, d / 2)
        self.southeastback = Octree(
            seback, self.capacity, (boundedSum(color[0], 0.7),
            color[1], boundedSum(color[2], 0.7)))

        swback = Cube(x - w / 2, y - h / 2, z - d / 2, w / 2, h / 2, d / 2)
        self.southwestback = Octree(
            swback, self.capacity, (boundedSum(color[0], 0.1),
            boundedSum(color[1], 0.1), boundedSum(color[2], 0.1)))

        self.divided = True

    def insert(self, point):
        if (not self.volumen.contains(point)):
            return
        if (len(self.points) < self.capacity):
            new_point = Point(point.x, point.y, point.z)
            self.points.append(new_point)

        else:
            if (not self.divided):
                self.subdivide()
            self.northeastfront.insert(point)
            self.northwestfront.insert(point)
            self.northeastback.insert(point)
            self.northwestback.insert(point)
```

```
            self.southeastfront.insert(point)
            self.southwestfront.insert(point)
            self.southeastback.insert(point)
            self.southwestback.insert(point)

    def query(self, rango, found):
        if (not rango.intersects(self.volumen)):
            return found
        for i in range(len(self.points)):
            if (rango.contains(self.points[i])):
                found.append(self.points[i])
        if (self.divided):
            self.northeastfront.query(rango, found)
            self.northwestfront.query(rango, found)
            self.southeastfront.query(rango, found)
            self.southwestfront.query(rango, found)
            self.northeastback.query(rango, found)
            self.northwestback.query(rango, found)
            self.southeastback.query(rango, found)
            self.southwestback.query(rango, found)
        return found

    def show(self):
        cube = vtk.vtkCubeSource()
        cube.SetCenter(self.volumen.x, self.volumen.y,
        self.volumen.z)
        cube.SetXLength(self.volumen.w * 2)
        cube.SetYLength(self.volumen.h * 2)
        cube.SetZLength(self.volumen.d * 2)
        cube.Update()

        # Mpear
        cubeMapper = vtk.vtkPolyDataMapper()
        cubeMapper.SetInputData(cube.GetOutput())

        # Actor.
        cubeActor = vtk.vtkActor()
        cubeActor.SetMapper(cubeMapper)
        cubeActor.GetProperty().SetOpacity(0.3)
        cubeActor.GetProperty().SetColor(self.color)

        if (self.divided):
            self.northeastfront.show()
```

```
            self.northwestfront.show()
            self.northeastback.show()
            self.northwestback.show()
            self.southeastfront.show()
            self.southwestfront.show()
            self.southeastback.show()
            self.southwestback.show()

        for i in range(len(self.points)):
            sphereSource = vtk.vtkSphereSource()

            xs = self.points[i].x
            ys = self.points[i].y
            zs = self.points[i].z
            sphereSource.SetCenter(xs, ys, zs)
            sphereSource.SetRadius(10)

            # Make the surface smooth.
            sphereSource.SetPhiResolution(100)
            sphereSource.SetThetaResolution(100)

            mapper = vtk.vtkPolyDataMapper()
            mapper.SetInputConnection(sphereSource.GetOutputPort(
            ))

            actor = vtk.vtkActor()
            actor.SetMapper(mapper)

            actor.GetProperty().SetColor(self.color)

            renderer.AddActor(actor)

        renderer.AddActor(cubeActor)
```

2.3. Generación de OcTree, elementos y renderizado

```
volumen = Cube(0, 0, 0, 400, 400, 400)
qt = Octree(volumen, 4, (0.0000, 1, 0))

for i in range(200):
    xs = random.uniform(-399, 399)  # -400,400
    ys = random.uniform(-400, 400)  # -400,400
    zs = random.uniform(-400, 400)  # -400,400
```

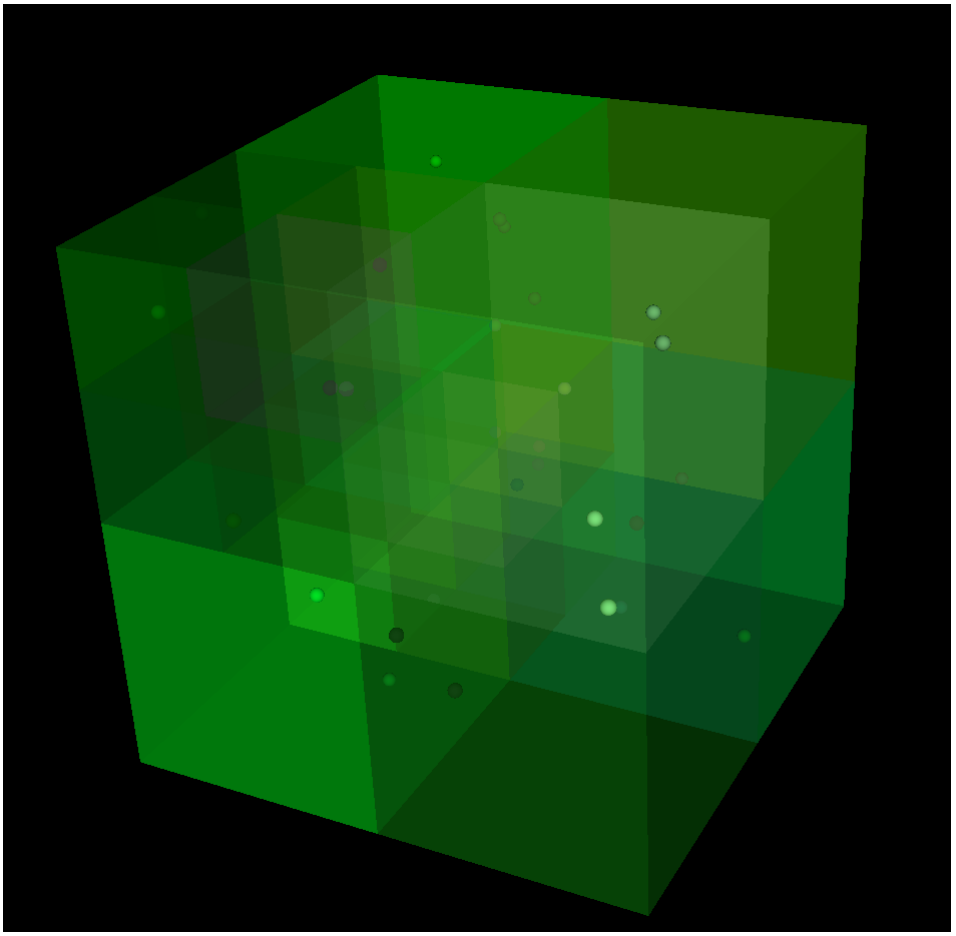
```
    p = Point(xs, ys, zs)
    qt.insert(p)

qt.show()

busqueda = Cube(0, 300, 0, 120, 120, 120)
cube = vtk.vtkCubeSource()
found = []
qt.query(busqueda,found)
for i in range(len(found)):
    sphereSource = vtk.vtkSphereSource()
    xs = found[i].x
    ys = found[i].y
    zs = found[i].z
    sphereSource.SetCenter(xs, ys, zs)
    sphereSource.SetRadius(20)

    # Make the surface smooth.
    sphereSource.SetPhiResolution(100)
    sphereSource.SetThetaResolution(100)

    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInputConnection(sphereSource.GetOutputPort())

    actor = vtk.vtkActor()
    actor.SetMapper(mapper)

    actor.GetProperty().SetColor(0,0,0)

    renderer.AddActor(actor)
cube.SetCenter(0, 300, 0)
cube.SetXLength(120 * 2)
cube.SetYLength(120 * 2)
cube.SetZLength(120 * 2)
cube.Update()
cubeMapper = vtk.vtkPolyDataMapper()
cubeMapper.SetInputData(cube.GetOutput())

# Actor.
cubeActor = vtk.vtkActor()
cubeActor.SetMapper(cubeMapper)
cubeActor.GetProperty().SetOpacity(0.01)
cubeActor.GetProperty().SetColor(254, 0, 0)
```

```
renderer.AddActor(cubeActor)

renderer.SetBackground((0.0, 0.0, 0.0))

renderWindow.SetSize(1000, 1000)
renderWindow.Render()
renderWindowInteractor.Start()
```

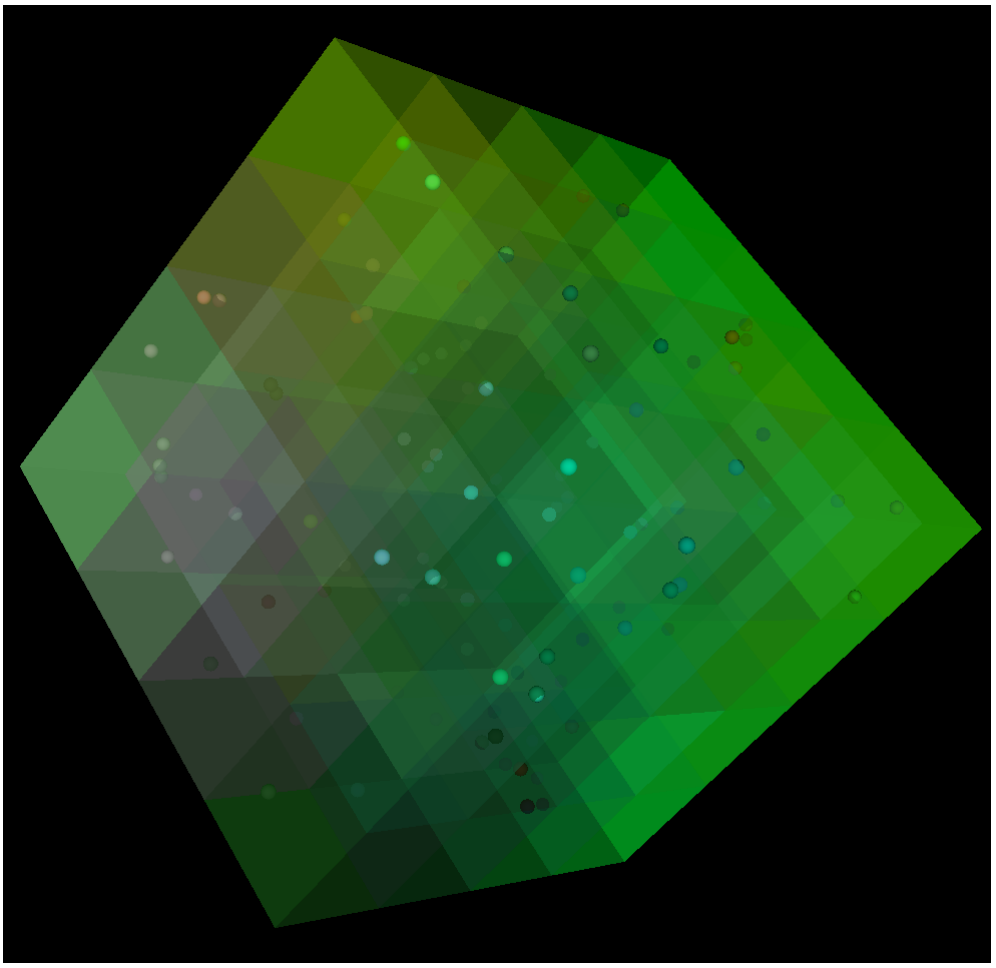
3. Resultados

3.1. OcTree con elementos aleatorios

3.1.1. Con 30 elementos



3.1.2. Con 100 elementos



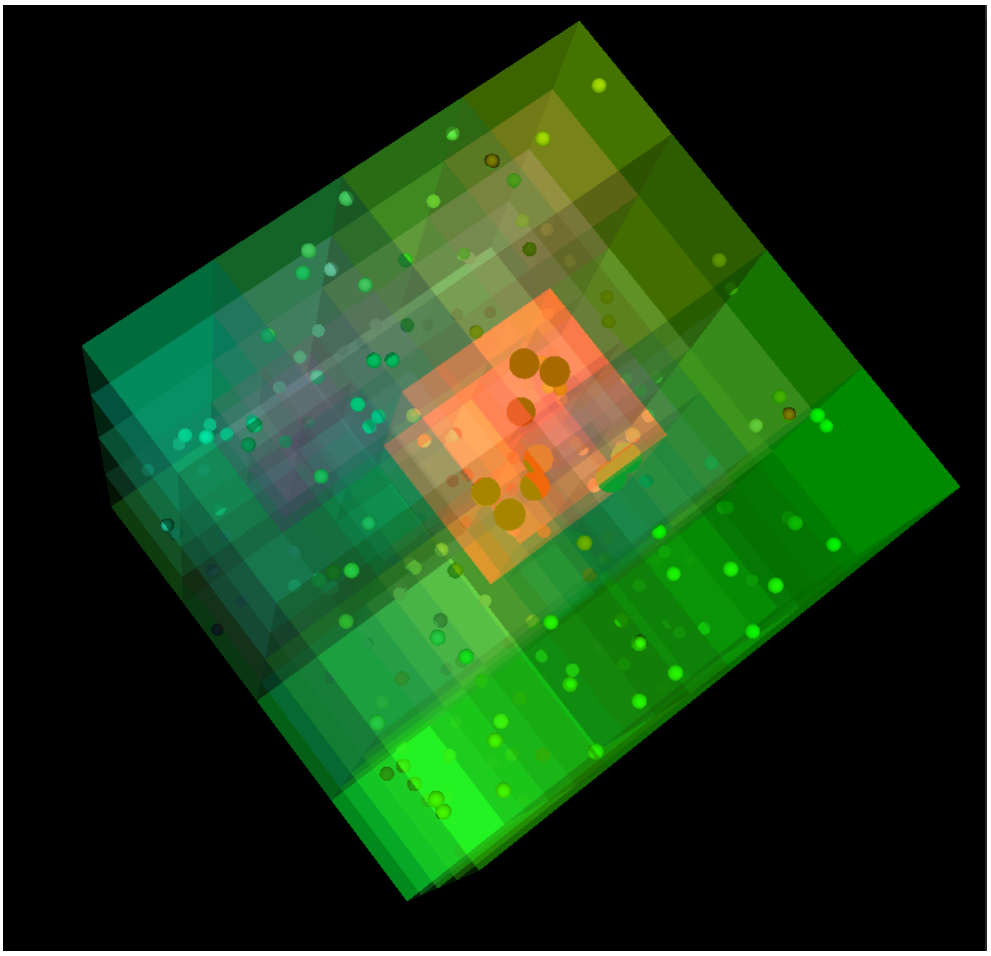
3.2. Búsqueda en OcTree

3.2.1. Código Query

```
def query(self, rango, found):
    if (not rango.intersects(self.volumen)):
        return found
    for i in range(len(self.points)):
        if (rango.contains(self.points[i])):
            found.append(self.points[i])
    if (self.divided):
        self.northeastfront.query(rango, found)
        self.northwestfront.query(rango, found)
        self.southeastfront.query(rango, found)
        self.southwestfront.query(rango, found)
        self.northeastback.query(rango, found)
```

```
            self.northwestback.query(rango, found)
            self.southeastback.query(rango, found)
            self.southwestback.query(rango, found)
        return found
```

3.2.2. Ejemplo búsqueda



Para realizar la busqueda luego de crear el OcTree se crea un cubo con el cual se realiza la búsqueda, todos los elementos que se encuentran dentro del cubo se pintan de color negro y doblan su tamaño para facilitar su visualización como se puede apreciar en el ejemplo.

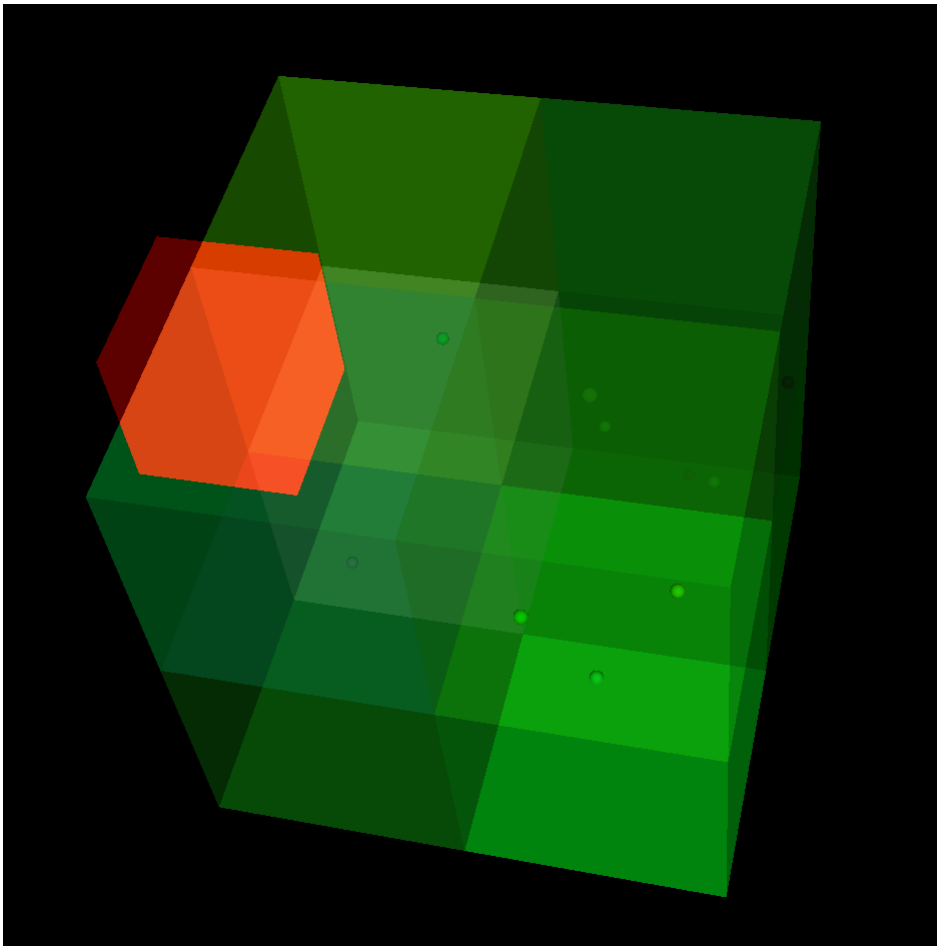
3.3. Agregar un elemento en tiempo de ejecución

3.3.1. Código Insertar un elemento al pulsar una tecla

```
def keypress_callback(obj, ev):
    key = obj.GetKeySym()
```

```
if(key == 'l'):
    qt.insert(Point(random.uniform(-399, 399),
    random.uniform(-400, 400),random.uniform(-400, 400)))
    renderer.RemoveAllViewProps()
    renderer.AddActor(cubeActor)
    findAndDraw()
    qt.show()
    renderWindow.Render()
```

3.3.2. Generandose al iniciar



3.3.3. Luego de 20 inserciones