ALGORITMOS DE ORDENAMIENTO QUADTREE

Elizabeth Yasmin Huanca Parqui*, Imanol Brayan Moscoso Apaza†, Gabriel Adriano Valencia Arana‡

Escuela profesional de Ciencia de la Computación,

Universidad Nacional de San Agustin Arequipa

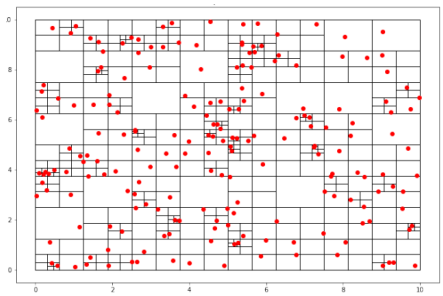

Figure 1: *Quadtree*

Abstract

Implementar el QuadTree en un lenguaje de programación con las funciones básica de inserción y búsqueda por rango, así como de visualzación. Guardando lo implementado en un repositorio en github, bitbucket o gitlab.Analizar sus funciones de inserción y búsqueda por rango. Insertar 50 puntos aleatorios que se encuentre dentro de los limites (boundary). Generar una gráfica que permita visualizar los puntos y los diferentes nodos creados. Documentar sus hallazgos en un formato de artículo de investigación (formato de libre elección), considerando aspectos del marco teórico (estado del arte), metodología, resultados, conclusiones, discusión y bibliografía.

Keywords: Quadtree, estructura, punto, cuadrante

1 Introducción

El término Quadtree, o árbol cuaternario, se utiliza para describir clases de estructuras de datos jerárquicas cuya propiedad común es que se basan en el principio de descomposición recursiva del espacio. En un QuadTree de puntos, el centro de una subdivisión siempre está en un punto. Cuando inserta un nuevo elemento, el espacio se divide en cuatro cuadrantes. Repetir el proceso nuevamente divide el cuadrante en cuatro cuadrantes, y así sucesivamente.

Existe una amplia variedad de estructuras jerárquicas para representar datos espaciales. Una técnica de uso común es Quadtree. El desarrollo de estos fue motivado por la necesidad de guardar los datos ingresados con valores iguales o similares. Este artículo trata sobre la representación de datos en un espacio bidimensional. Quadtree también se utiliza para representar datos en espacios tridimensionales o hasta 'n' dimensiones.

La detección de colisiones implica determinar cuándo un objeto penetra en otro. Claramente, esta es una propuesta costosa, especialmente cuando hay una gran cantidad de objetos involucrados y los objetos tienen formas complejas, como es el caso de los proyectos de software de entretenimiento. Muchos algoritmos de detección de colisiones son bastante complejos y deben cumplir muchos casos especiales, en este artículo presentamos el uso de la estructura de datos quadtree que tiene como objetivo mejorar el costo de tiempo y espacio en el desarrollo de proyectos de software de entretenimiento.

2 Marco Teórico

2.1 Quadtrees

Los Quadtrees son un tipo de estructura de datos en el que cada nodo tiene a su vez cuatro nodos hijos, sería algo similar a un árbol binario, pero en vez de dos ramas son cuatro ramas. El término Quadtree , o árbol cuaternario, se utiliza para describir clases de estructuras de datos jerárquicas cuya propiedad común es que están basados en el principio de descomposición recursiva del espacio. En un QuadTree de puntos, el centro de una subdivisión está siempre en un punto. Al insertar un nuevo elemento, el espacio queda dividido en cuatro cuadrantes. Al repetir el proceso, el cuadrante se divide de nuevo en cuatro cuadrantes, y así sucesivamente.

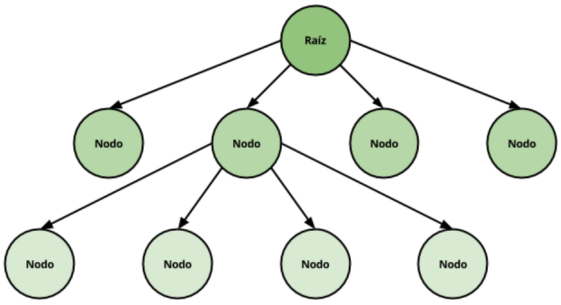

Figure 2: *Quadtree*

El Quadtree divide un espacio en cuadrantes y estos a su vez se divi-

*ehuancap@unsa.edu.pe
†e-mail:author2@cs.rutgers.edu
‡e-mail:author3@cs.rutgers.edu

den en más cuadrantes, de manera tal que se indexen cada elemento del espacio. Por ejemplo en la imagen de abajo, se visualiza el espacio, cada círculo representa un dato. La idea es que al tener todo el espacio dividido en cuadrantes resulta mucho más fácil hacer consultas, como la cantidad de puntos que están en un rango determinado, ver los puntos con los que colisionan, etc. Al incrementar en cada cuadrante la profundidad se divide este en nuevos cuatro cuadrantes, debido a que empieza a tener demasiados objetos y es demasiado costoso comprobar todas las colisiones que suceden en él. Así que la solución es subdividir a su vez este cuadrante en otro nodo Quadtree.

El quadtree es una estructura de datos que representa una partición bidimensional del espacio dividiendo la región en cuatro cuadrantes iguales. Estos cuadrantes se dividen en subcuadrantes y así sucesivamente hasta que tengan regiones que contengan solo un elemento. Cada nodo del árbol tiene exactamente cuatro hijos o ninguno en el caso de las hojas. Como veremos más adelante, el quadtree es un caso especial de trie. Un quadtree con una profundidad de n puede ser usado para representar un arreglo de $2^{(n)}x2^n$. Si este arreglo está compuesto de solo 0's (1's), entonces el quadtree consiste en una raíz (representando todo el arreglo) y, si alguna región (cuadrante) no está completamente llena con 1's o 0's, entonces se crea una subdivisión. Entre los usos que se le pueden dar al quadtree están la representación de imágenes, información como la temperatura en una región del espacio o la representación de un conjunto de puntos (como la latitud y longitud de un conjunto de ciudades)

2.2 La detección de colisiones en aplicaciones de realidad virtual

Los métodos son aplicables a todos los modelos poligonales generales. Combinan los beneficios de la detección de colisiones mediante el uso de jerarquías de volumen límite con la capacidad de calcular resultados dinámicos de detección de colisiones. Los resultados se utilizan como entrada para simulaciones posteriores, por ejemplo. Simulación dinámica o de contacto.

En el artículo "Detección dinámica de colisiones en aplicaciones de realidad virtual" presenta estructuras de datos y algoritmos para la detección dinámica de colisiones en aplicaciones de realidad virtual.

2.3 Ventajas de Quadtree

1. Representación de imágenes usando la estructura de Quadtree.

2.4 Desventajas de Quadtree

1. Una de las desventajas de la estructura de Quadtree es que hay que almacenar una gran cantidad de píxeles diferentes en una imagen con muchos colores, ya que puede adquirir un tamaño excesivamente grande, viendo cuanto tiempo y costo puede ocacionar esta accion, lo miramos como una gran desventaja al momento de la visualización.

2. Dado que esta estructura de datos de Quadtree es la memoria principal, cuando queremos representar imágenes muy grandes, este no se puede almacenar en esta memoria varias veces. En tal caso, se utilizan estructuras alternativas como por ejemplo mencionamos un árbol B +, para compensar esta limitación.

2.5 Explicación del Pseudocódigo

1. Si la raíz es cero, P se convierte en la raíz.

2. Si la raíz no es GRIS, significa que solo hay un nodo

3. Se realiza una búsqueda desde la raíz para encontrar el cuadrante al que pertenece P.

2.6 Usos y aplicaciones

- Representación de imágenes.
- Indexación de direcciones espaciales.
- Detección de colisiones bidimensional eficiente.
- Suelta el tronco de la vista de datos de campo.
- Almacenar datos raros, como formatear información para una hoja de trabajo o para algunos cálculos matriciales.
- Solución de campo multidimensional (dinámica de fluidos computacional, electromagnetismo)
- Quadtrees es el análogo bidimensional de octrees.

3 Metodologia

Para la implementación de la estructura del quadtree se optó por el lenguaje de programación python debido a su gran cantidad de librerias, funciónes y conjunto de modulos, en este caso se usará el conjunto de modulos pygame para la representación y visualización gráfica del quadtree. El código presentado es una aplicación básica de un quadtree con el nodo prade y sus cuatro nodos hijos, en este caso solo implementaremos las funcionalidades de inserción y búsqueda. A nuestro código lo pondremos a prueba con la inserción de 50 puntos aleatorios que se encuentran dentro de los limites junto con su gráfica, vamos a poder apreciar el comportamiento de los quadtree.

4 Resultados

Cabe resaltar que por la naturaleza de pygame, las coordenadas para la visualización con respecto al eje y se encuentran invertidos. Mostraremos el código del Quadtree, algunos casos solo mostraremos fragmentos debido a la longitud del código. Además mostraremos la visualización del árbol.

4.1 código

Listing 1: *Clase Punto-Constructor*

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

lines = []
```

Listing 2: *Clase Nodo-Constructor*

```
class Node():
        def __init__(self, x0, y0, x1, y1):
                self.x0 = x0
                self.y0 = y0
                self.x1 = x1
                self.y1 = y1
                self.nodes = []
                self.lines = [(x0,y0, (x1-x0), 1),
                (x0,y0, 1,(x1-x0)), (x0, (y1+y0)/2
                ,(y1-y0), 1), ((x1+x0)/2,y0,1,
                (y1-y0))]
                self.sub = []
                for i in range(4):
                        self.sub.append(None)
```

Listing 3: *Clase Nodo-fragmento de la función divide*

```
def divide(self, pnt):
```

```
  for point in self.nodes:
    if point.x >= self.x0 and point.x <
        self.x1/2+self.x0/2 and point.y >=
        self.y0 and point.y <
        self.y1/2+self.y0/2:
          if self.sub[0] == None:
                self.sub[0] = Node(self.x0,
                self.y0, self.x1/2+self.x0/2,
                self.y1/2+self.y0/2)
                self.sub[0].insert(point,
                "cuadrante_1")
                print("aqui")
          else:
                self.sub[0].insert(point,
                "cuadrante_1")
```

Listing 4: *Clase Nodo-función insert*

```
def insert(self, point, cuadrante):
   if not point in self.nodes:
          if point.x >= self.x0 and point.x <
          self.x1 and point.y >= self.y0 and
          point.y < self.y1:
                if len(self.nodes) < 4:
                      self.nodes.append(point)
                else:
                      self.divide(point)
                return
```

Listing 5: *Clase Nodo-funciones get y getLines*

```
def get(self,lines):
  if len(self.nodes) >=4:
        for i in range(4):
              lines.append(self.lines[i])
        for i in range(4):
              if self.sub[i] != None:
                    self.sub[i].get(lines)
def get_lines(self):
        lines = []
        self.get(lines)
        return lines
```

Listing 6: *Clase Nodo-funciones getintersect*

```
def getintersect(self, x1,x2,y1,y2):
  points = []

  for i in range(len(self.nodes)):
        if self.nodes[i] != None:
              print(self.nodes[i].x,
              self.nodes[i].y)
           if self.nodes[i].x < x2 and
              self.nodes[i].x > x1 and
              self.nodes[i].y < y2 and
              self.nodes[i].y > y1:
                print("aqui")
                points.append(self.nodes[i])

        return points
```

Listing 7: *fragmento del código para la visualización*

```
WIDTH = 1000
HEIGHT = 1000
BLACK = (0,0,0)
Qtree = Node(0,0,700,700)
points = []
pygame.init()
screen = pygame.display.set_mode((700,700))

for i in range(200):
        p = Point(int(round(random.uniform(0,10)
        , 2)*70),int(round(random.uniform(0,10),
        2)*70))
        points.append(p)
count = 0
listade_sprites_activas = pygame.sprite.Group()
reloj = pygame.time.Clock()
rect = []

for i in range(len(points)):
        print(points[i].x, points[i].y)
a = Rect((0,0,0,0))

aux = 0
```

4.2 Visualización

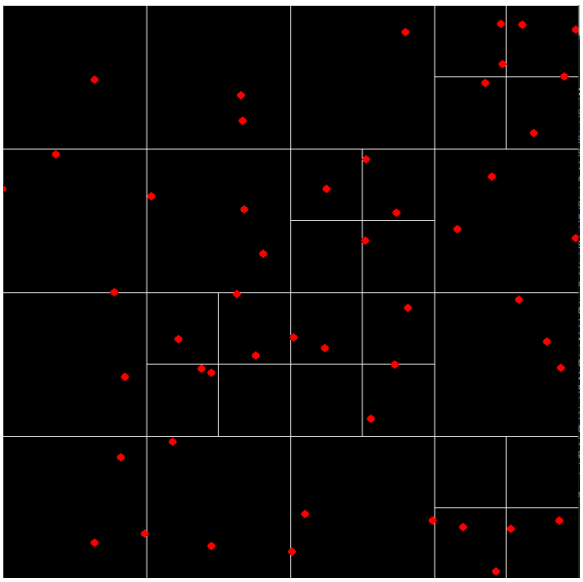

Figure 3: *Inicialización con 50 datos*

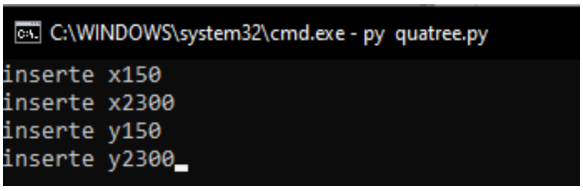

Figure 4: *Insertar datos en las coordenadas para la busqueda por rangos (x_1,x_2,y_1,y_2)*

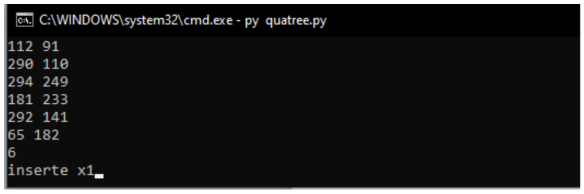

Figure 5: *retorno de los datos tras range query donde se muestran las coordenadas de los puntos y el numero de puntos interceptados*

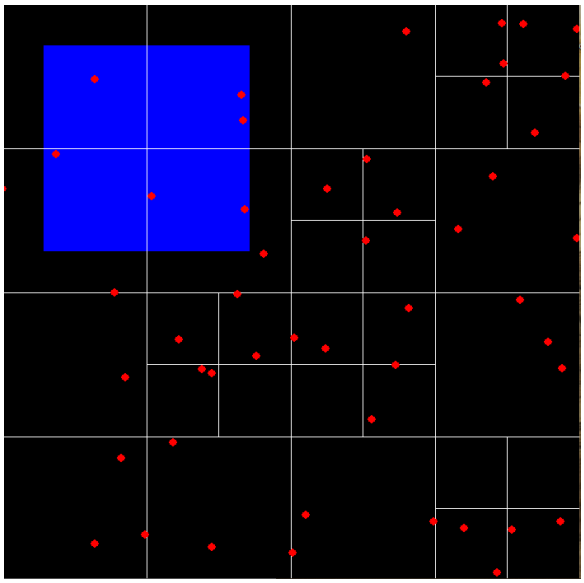

Figure 6: *visualización grafica del rango creado*

5 Conclusiones

En el artículo hemos podido apreciar las estructuras de datos Quadtree y gracias a la ayuda del lenguaje de programación Python y la herramienta Pygame logramos apreciar el comportamiento que poseen este árbol. Destacando su importancia para dividir un espacio bidimensional recursivamente en cuatro cuadrantes. Este árbol llega a ser una herramienta adecuada para la búsqueda y organización de información.

6 Discusión

El imprevisto que presento Pygame con respecto a la visualización en el eje Y puede ser un imprevisto si queremos exponer nuestro resultado, ademas que una desventaja que presenta el quadtree es que consume bastante recursos cuando representa grandes tamaños de datos, junto con pyhton no tendriamos el resultado deseado en un tiempo optimo. Por ello para futuros proyectos planeamos en implementarlo en otro lenguaje inclu

7 Referencias

Sullivan, G. J., Baker, R. L. (1994). Efficient quadtree coding of images and video. IEEE Transactions on image processing, 3(3), 327-331.

Shaffer, C. A., Samet, H. (1987). Optimal quadtree construction algorithms. Computer Vision, Graphics, and Image Processing, 37(3), 402-419.

Spann, M., Wilson, R. (1985). A quad-tree approach to image segmentation which combines statistical and spatial information. Pattern Recognition, 18(3-4), 257-269.