Octree - Implementación y Visualización con C++/VTK

Roque Sosa, Owen Haziel
Facultad de Ciencias de la Computación
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
oroque@unsa.edu.pe

***Abstract*—En este artículo, exploramos a fondo la estructura de datos Octree, una herramienta esencial en la computación gráfica y la representación espacial tridimensional. Presentamos su detallada implementación en C++, destacando su versatilidad y eficiencia en la gestión de datos tridimensionales. Además, abordamos la visualización de la información utilizando la poderosa biblioteca de visualización *VTK* (Visualization Toolkit). A lo largo del artículo, proporcionamos ejemplos prácticos y consideraciones clave para facilitar la comprensión y aplicación de esta estructura de datos en escenarios del mundo real.**

***Index Terms*—Estructuras de datos tridimensionales, Representación espacial, Visualización tridimensional, Algoritmos de búsqueda**

I. Introducción

La computación gráfica y la representación tridimensional de datos han experimentado un crecimiento exponencial, impulsando la necesidad de estructuras de datos eficientes que gestionen volúmenes de puntos de manera óptima. En este contexto, el Octree emerge como una herramienta fundamental, proporcionando una representación jerárquica y espacialmente eficiente de datos tridimensionales.

Este informe aborda la descripción, implementación y visualización de la estructura de datos Octree, destacando su papel crucial en el manejo de grandes conjuntos de datos tridimensionales. El objetivo principal es presentar una implementación detallada del algoritmo Octree, explorando las funciones de *inserción* y *búsqueda* que permiten manipular eficazmente la estructura. Además, se analiza el funcionamiento del Octree en la visualización de volúmenes de puntos, utilizando herramientas como el Visualization Toolkit (VTK).

A lo largo del informe, se detallarán las actividades realizadas, desde la implementación del algoritmo Octree hasta la visualización de conjuntos de datos representativos como los puntos de un modelo tridimensional. Las pruebas con diferentes cantidades de puntos almacenados en cada nodo proporcionarán una visión crítica de la eficiencia de la estructura en escenarios prácticos.

II. Marco Teórico

La representación eficiente de datos tridimensionales es esencial en diversos campos, como la computación gráfica, la simulación y la visualización científica. En este contexto, el Octree, también conocido como árbol octal, se erige como una estructura de datos jerárquica y espacialmente eficiente que facilita la gestión y manipulación de grandes conjuntos de puntos tridimensionales.

A. Estructura del Octree

El Octree organiza el espacio tridimensional de manera recursiva mediante la subdivisión de octantes. Inicialmente, un cubo que representa el espacio total se divide en ocho octantes más pequeños, y este proceso se repite de manera recursiva en cada octante hasta alcanzar un nivel deseado de profundidad o hasta que se cumpla un criterio predefinido. Cada nodo del Octree puede contener información asociada y se clasifica como nodo interno si tiene hijos, o nodo hoja si representa una región mínima y contiene datos específicos. [1] Durante la visualización, se enfocará únicamente en los nodos hoja del Octree, los cuales representan regiones mínimas del espacio tridimensional y contienen los puntos a visualizar. Este enfoque optimizado mejora significativamente la eficiencia del proceso de renderizado.

B. Aplicaciones

La versatilidad del Octree lo hace adecuado para diversas aplicaciones, destacando su utilidad en la representación espacial de objetos tridimensionales, colisiones en simulaciones físicas, y en la aceleración de algoritmos de búsqueda espacial. En este informe, nos centraremos en su aplicación específica en la gestión de volúmenes de puntos, explorando cómo esta estructura puede optimizar el almacenamiento y acceso a información tridimensional.

III. Implementación de métodos de inserción y búsqueda en el Octree

La correcta manipulación de datos tridimensionales mediante la estructura de datos Octree implica la implementación precisa de métodos de inserción y búsqueda. Estos métodos son cruciales para garantizar la coherencia y eficiencia en la representación jerárquica del espacio tridimensional. En esta sección, describiremos detalladamente la implementación de los métodos **insert** y **exists** en el contexto de nuestra clase Octree.

A. Clase Octree

```
class Octree {
private:
   // Octantes
   std::vector<Octree*> children;
   // Puntos por nodo
   int nPoints;
   std::vector<Point> points;
   // Bandera si es Nodo Interno
   bool isInternal;
   // Punto BottomLeftFront y h
   // Para representar espacio
   Point BottomLeftFront;
   double h;
public:
   Octree(int np);
   Octree(const Point& p,
   double height, int np);
   ~Octree();
   void setOctant(const Point& p, int& oct);
   bool exists(const Point& p);
   void insert(const Point& p);
   // Para renderizar octree con VTK
   vtkSmartPointer<vtkActor> createCubeActor()
       ;
   void visualizeOctree(vtkSmartPointer<
       vtkRenderer> renderer);
   void visualizeOctreeNode(vtkSmartPointer<
       vtkRenderer> renderer, Octree* node);
};
```

La clase Octree almacena el vector de nodos hijos que serán 8, así como el valor para establecer la capacidad del vector de puntos, una bandera para determinar si es un nodo interno, y un punto y double para representar el espacio respectivo del cubo/octante.

B. Inserción

```
Funcion insert(const Point& p):
   Si exists(p) Entonces
      Punto ya existe.
      Retornar
   Fin Si
   setOctant(p, octPos)
   Si octPos == -1 Entonces
      Fuera de Limites.
      Retornar
   Fin Si
   Si No EsInterno(children[octPos]) Entonces
      Descender recursivamente.
      children[octPos].insert(p)
      Retornar
   Fin Si
   Si size(children[octPos].points) < nPoints
       Entonces
      AgregarPunto(children[octPos].points, p)
      AsignarBottomLeftFront(children[octPos],
          octPos)
      children[octPos].h = h / 2
      Insertado.
      Retornar
   Fin Si
   Sino
      Split.
      Vector temp = Copiar(children[octPos].
          points)
      Eliminar(children[octPos])
      Segun octPos Hacer
         children[octPos] = CrearNuevoOctree(
             nuevas dimensiones segun octante)
      Fin Segun
      Reinsertar vector temp.
      children[octPos].insert(p)
   Fin Si
Fin Funcion
```

Este método primero se encarga de ubicar el punto en el octante respectivo del Octree, buscándolo previamente para evitar duplicados (redundancia). *setOctant* se encarga de:

- Verificar que el punto en cuestión no exceda los limites del cubo,
- Obtener las coordenadas del 'centro' del octante actual,
- Determinar en qué sub-octante del cubo principal está ubicado el punto.

De modo que devolverá un valor que se almacenará en *octPos*. A continuación vienen 3 casos principales:

a) Es nodo interno: Se seguirá buscando recursivamente en el octante apropiado hasta llegar a un nodo hoja.

b) Es nodo hoja con espacio disponible: En este caso se insertará en el vector de puntos del octante previamente seleccionado, asi también se definirá el punto y longitud necesarios para definir el espacio.

c) Es nodo hoja lleno: En este caso se tiene que dividir el nodo en 8 sub-octantes, es decir, convertirse en nodo interno. Para ello se almacena temporalmente el vector de puntos del octante/nodo actual, se elimina y se define un nuevo Octree en su lugar, con el punto y longitud acorde a octPos. A continuación se reinsertan el conjunto de puntos en este nuevo Octree (recursivamente) y finalmente se procede a insertar nuevamente el punto inicial.

C. Búsqueda

```
Funcion exists(const Point& p):
   setOctant(p, octPos)
   Si octPos == -1 Entonces
      Retornar False
   Fin Si
   Si EsInterno(children[octPos]) Entonces
      Retornar children[octPos].Existe(p)
   Fin Si
   Sino Si EstaVacio(children[octPos].points)
       Entonces
      Retornar False
   Fin Si
   Sino
      Para cada point en children[octPos].
          points Hacer
         Si p.x == point.x y p.y == point.y y
             p.z == point.z Entonces
            Retornar True
         Fin Si
      Fin Para
   Fin Si
   Retornar False
Fin Funcion
```

La lógica de este algoritmo es similar a la de inserción. Se determina el octante apropiado para ubicar al punto, si este excede los límites se retorna False. A continuación se determina la existencia del punto en base a casos muy parecidos a los expuestos en el método anterior:

a) Es nodo interno: Se seguirá buscando recursivamente en dicho octante.

b) Es nodo hoja vacío: En dicho caso no existe dicho punto en el Octree hasta el momento.

c) Es nodo hoja con elementos: Buscar en el contenedor de puntos.

D. Resultados

En esta sección, presentamos los resultados obtenidos de la implementación y visualización del Octree en diferentes escenarios, utilizando conjuntos de datos tridimensionales representativos. Los experimentos se realizaron con el propósito de evaluar la eficiencia y efectividad de la estructura Octree en la gestión de grandes volúmenes de puntos. Los resultados se basan en la implementación en C++ y la visualización utilizando la librería VTK. Se utilizaron dos conjuntos de datos tridimensionales significativos: un modelo de gato y un modelo de dragón, representados como conjuntos de puntos en archivos CSV. Se llevaron a cabo pruebas variando la cantidad de puntos almacenados en cada nodo del Octree para evaluar su comportamiento en diferentes situaciones.

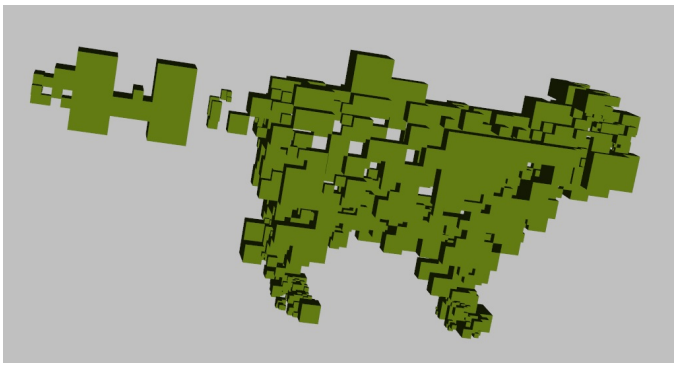

Fig. 1. Gato.csv con 1 punto por nodo

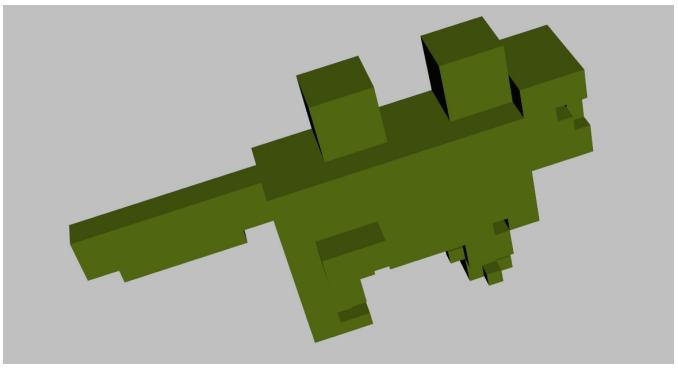

Fig. 2. Gato.csv con 8 puntos por nodo

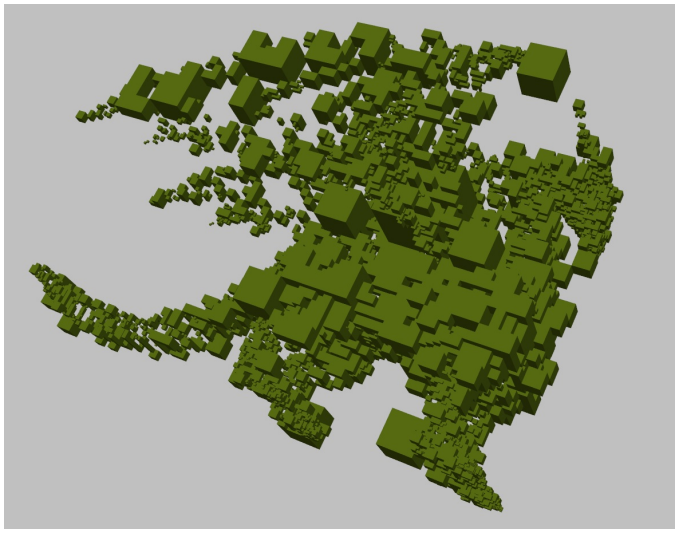

Fig. 3. Dragon.csv con 1 punto por nodo

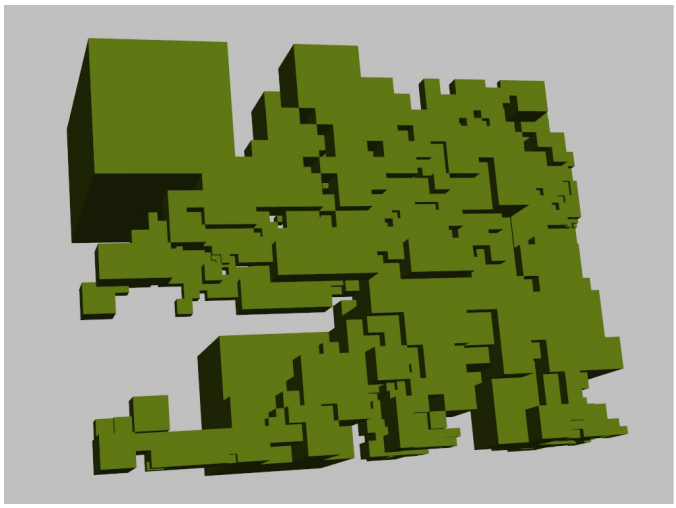

Fig. 4. Dragon.csv con 8 puntos por nodo

Durante las pruebas con diferentes cantidades de puntos almacenados en cada nodo del Octree, se observó un comportamiento eficiente en términos de tiempo de ejecución y consumo de memoria. La estructura Octree demostró ser especialmente beneficiosa al optimizar la búsqueda y manipulación de datos tridimensionales, manteniendo un rendimiento estable incluso con volúmenes de datos significativos.

Además, se demostró que la profundidad del Octree juega un papel muy importante en la calidad de la visualización y la eficiencia en la manipulación de datos. Se observó que un equilibrio adecuado en la profundidad permitía representaciones visuales claras sin comprometer significativamente el rendimiento, especialmente el de la visualización.

E. Conclusiones

En este estudio, hemos explorado la implementación y visualización de la estructura de datos Octree para la gestión eficiente de conjuntos de datos tridimensionales. Los resultados

obtenidos respaldan la eficacia del Octree en la representación y manipulación de grandes volúmenes de puntos, demostrando su capacidad para optimizar las operaciones espaciales y la visualización. La implementación en C++ y la visualización con VTK han proporcionado una plataforma robusta para experimentar con la estructura Octree, destacando su versatilidad en escenarios que van desde la computación gráfica hasta la simulación tridimensional. Además, los experimentos han permitido identificar la influencia clave de la profundidad del Octree en la calidad de la visualización y la eficiencia operativa. Encontrar un equilibrio adecuado en la profundidad se revela como un factor crítico para lograr representaciones visuales claras sin comprometer el rendimiento del sistema. En resumen, este estudio resalta el valor y el potencial continuo del Octree como una herramienta fundamental en el ámbito de la representación y manipulación eficiente de datos tridimensionales.

REFERENCES

[1] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures," Theoretical Foundations of Computer Graphics and CAD, pages 51–68, 1988.

[2] H. Samet, "Foundations of multidimensional and metric data structures," Elsevier - Morgan Kaufmann, 2006.