Octree

[*]Esta es la primera entrega del proyecto final del curso: "Algoritmos y Estructuras de Datos" en la Universidad de Ingeniería y Tecnología (UTEC)

José Ernesto Guerrero Cueva
Ciencias de la Computación
UTEC
Lima, Perú
jose.guerrero@utec.edu.pe

Leonardo Alfredo Montoya Solorzano
Ciencia de Datos
UTEC
Lima, Perú
leonardo.montoya@utec.edu.pe

Esteban José Sulca Infante
Ciencia de la Computación
UTEC
Lima, Perú
esteban.sulca@utec.edu.pe

Abel Escobar
Ciencia de la Computación
UTEC
Lima, Perú
yuri.escobar@utec.edu.pe

***Abstract*—The Octree is a hierarchical spatial data structure designed to efficiently manage three-dimensional (3D) spatial data. The Octree partitions 3D space into eight octants at each node, enabling recursive subdivision of volumetric regions. This structure is generally used in areas such as computer graphics, 3D modeling, geographic information systems (GIS), and physics simulations due to its ability to reduce memory usage and improve query performance. This paper analyzes the architecture of the Octree, discusses related spatial structures, and explores its practical applications. Additionally, it presents and compares the Octree with alternative spatial partitioning structures such as k-d trees, highlighting its advantages in managing sparse and complex volumetric data. The paper also provides algorithms and pseudocode for insertion, search, and deletion operations, demonstrating how to maintain spatial balance and efficiency in dynamic environments.**

I. Introducción

Un Octree, en términos generales, es una estructura de datos jerárquica diseñada para gestionar y procesar información espacial en tres dimensiones. Tal como su nombre sugiere, un octree "is a tree data structure where each internal node has exactly 8 children" [1]. Un octree basa su utilidad en la división recursiva de un espacio tridimensional (denotado por el nodo padre) en octantes. A cada uno de los 8 espacios se les asigna uno de los 8 nodos hijos, para así continuar subdividiendo el espacio hasta que se satisfaga una condición de salida. Esta división permite organizar datos espaciales, como puntos, objetos o volúmenes, de manera eficiente, reduciendo la complejidad de consultas y operaciones en entornos tridimensionales no regulares.

La importancia de los Octrees radica en su capacidad para optimizar tareas computacionales en diversas áreas de la informática, con especial énfasis en el procesamiento de gráficos por computadora. Los Octrees se utilizan para acelerar el renderizado, implementando técnicas como el *frustum culling*, que descarta objetos fuera del campo de visión de la cámara [2], o para gestionar texturas volumétricas en motores de juegos como Unity o Unreal Engine. Asimismo, son útiles para la detección de colisiones, dividiendo el espacio en regiones manejables y facilitando la identificación de intersecciones.

A diferencia de estructuras lineales, como arreglos, que requieren recorrer todos los elementos para consultas espaciales ($O(n)$), el Octree implementa las operaciones básicas con una complejidad en el peor caso de $O(\log n)$. Por ello, se le considera una estructura eficiente.

Consiguientemente, esta investigación explorará la arquitectura del Octree, sus operaciones fundamentales y sus aplicaciones prácticas, proporcionando una base sólida para comprender su relevancia en la informática moderna. Asimismo, se comparará con otras estructuras espaciales, como los Quadtrees y los k-d trees, para destacar sus ventajas y limitaciones en diferentes contextos.

II. Estructuras de datos relacionadas

Así como el Octree, existen otras estructuras de datos útiles para el manejo de puntos en el espacio. En esta sección, se explorarán otras estructuras de datos espaciales que comparten principios o aplicaciones similares, ofreciendo diferentes enfoques para la partición y organización del espacio. Con ello, se busca comprender mejor las características particulares de la estructura elegida frente a otras que también tienen aplicaciones específicas según los requerimientos de cada problema.

A. Octree vs KD Tree

TABLE I
TABLA 2 COMPARACIÓN DE OCTREE Y KD-TREE

Aspecto	**Octree**	**KD-Tree**
Estructura Fundamental y Partición del Espacio	Divide el espacio recursivamente en ocho octantes iguales (para 3D). Partición regular del espacio. Nodos internos con hasta 8 hijos [8], [9].	Divide el espacio recursivamente a lo largo de un eje a la vez, alternando dimensiones. Plano de división usualmente en la mediana. Partición adaptativa. Nodos internos con 2 hijos [3], [7].
Número de Dimensiones Soportadas	Diseñado específicamente para 3 dimensiones. Su extensión a dimensiones mayores es posible, pero su eficiencia disminuye debido al crecimiento exponencial del número de hijos por nodo (2^d) [8] (donde d representa la dimensión).	Diseñado para k dimensiones. Flexible y eficiente en espacios multidimensionales ($k \geq 2$) [3], [7].
Dependencia de la Distribución de los Datos	La estructura de partición es regular e independiente de la distribución de datos; la ocupación de nodos sí depende de ella [8], [9].	La estructura del árbol (ejes y puntos de división) depende directamente de la distribución de los datos [3], [7].
Balance del Árbol	Puede ser desbalanceado en términos de puntos por subárbol si los datos están agrupados, aunque la subdivisión sea regular [8].	Tiende a ser balanceado en profundidad ($O(\log N)$) si se construye usando medianas, balanceando el número de puntos por subárbol [5], [7].
Complejidad de Inserción y Eliminación (Dinámica)	Inserción: $O(\log N)$. Eliminación: Puede ser compleja, implicando fusión o marcado [8].	Inserción: $O(\log N)$. Eliminación: Compleja y puede desbalancear el árbol; a menudo se usa eliminación "perezosa" [3], [7].
Uso de Memoria	Puede ser alto con datos dispersos si se subdivide profundamente. Nodos internos almacenan hasta 8 punteros. $O(N)$ para Octrees "comprimidos" o bien poblados [8], [9].	Generalmente $O(N)$ para N puntos. Cada nodo almacena eje, valor de corte y 2 punteros [3], [7].
Casos de Uso Típicos y Ventajas	Computación gráfica (voxelización, detección de colisiones 3D), simulación, indexación espacial 3D. Simplicidad de partición regular [8], [9].	Búsqueda de vecinos más cercanos (k-NN), bases de datos espaciales, aprendizaje automático. Adaptabilidad a la distribución de datos [3], [5], [7].

III. Aplicación y comparación de los usos del Octree

A. Usos del Octree

El Octree es una estructura de datos usualmente utilizada en contextos donde la eficiencia en la gestión de espacios tridimensionales es crucial. Su capacidad para subdividir el espacio de forma jerárquica lo convierte en una herramienta muy útil en múltiples dominios.

- **Gráficos por computadora y videojuego:** El Octree permite representar escenas tridimensionales complejas y optimizar procesos de renderizado. Por ejemplo, es empleado en algoritmos de occlusion culling, donde solo se renderizan los objetos visibles desde la cámara, descartando aquellos que están fuera del campo de visión o bloqueados por otros elementos [5], [12].
- **Motores gráficos:** En casos como Unity o Unreal Engine, los Octrees se integran para gestionar colisiones, dividir el espacio en regiones adaptables y reducir la complejidad de cálculos físicos en tiempo real [13], [14]. Esto mejora significativamente el rendimiento, sobre todo en escenas con gran densidad de objetos.
- **Ray tracing:** Técnica de generación de imágenes realistas mediante simulación de rayos de luz. Aquí, el Octree permite acelerar la intersección entre rayos y objetos al evitar comprobar cada rayo contra todos los objetos del escenario [12].
- **Robótica, simulación física y navegación autónoma:** El Octree se aplica para la construcción de mapas tridimensionales (como en OctoMap) donde cada nodo representa un volumen cúbico del entorno y contiene información sobre la ocupación del espacio, facilitando la planificación de rutas y la detección de obstáculos [15].
- **Computación médica:** El octree es utilizado en la reconstrucción volumétrica de órganos a partir de imágenes de resonancia magnética o tomografía computarizada, mejorando el almacenamiento y procesamiento de grandes volúmenes de datos tridimensionales [16].

B. Comparativa

Según la figura 1, se compara el tiempo de construcción de un árbol Octree frente a un árbol K-d para distintos volúmenes de datos (número de puntos). Se observa que el tiempo de construcción del árbol Octree es menor que el del K-d tree, especialmente conforme aumenta el número de puntos. Esto implica que, para grandes volúmenes de datos, el Octree ofrece una mayor eficiencia en la fase de construcción del índice..

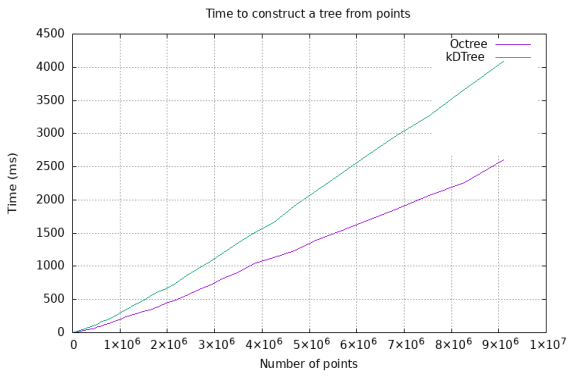

Fig. 1. Construcción de los árboles con distintos inputs con volumen creciente, según [10]

La Figura 2, presenta una comparación del desempeño entre estructuras Octree y K-d tree en tareas de búsqueda de vecinos. En la primera subgráfica (izquierda), se analiza el tiempo que toma realizar búsquedas K-NN para distintos valores de ($K = 8, 16, 32$). Se observa que el Octree mantiene un tiempo de búsqueda bajo y constante, mientras que los tiempos para K-d tree aumentan significativamente con el tamaño del conjunto y el valor de K. La subgráfica de la derecha muestra los resultados de una búsqueda por rango utilizando distintos radios ($\rho = 0.05,\ 0.10,\ 0.15$). Aquí también el Octree demuestra un rendimiento más eficiente, con tiempos de consulta más bajos en comparación con el K-d tree.

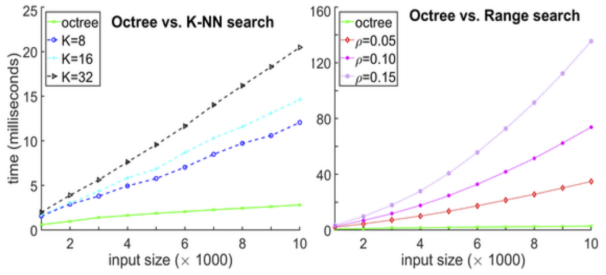

Fig. 2. Comparación de un octree con un K-d tree con distintos valores de K y ρ, según [11]

IV. Definición del problema

Como bien es sabido, la computación busca reducir el tiempo de ejecución de un proceso a través del diseño de un buen algoritmo y la elección de una buena estructura de datos adecuada. La estructura de datos es donde, como bien dice el nombre, viven los datos, y estos tienen distinta naturaleza. Además, desde la década de los 60 hasta la actualidad, dado que la capacidad computacional avanza, los que trabajamos en este ámbito nos vemos en la posición de ayudar a distintos sectores tales como la medicina, la minería, y todos en donde sea necesario simular o modelar visualmente. Es por esto que, hasta el día de hoy, se busca una buena estructura para manejar datos de naturaleza 3D. Las primeras propuestas fueron los K-d trees, que podían almacenar este tipo de datos (1984, Guttman) y los Bounding Volume Hierarchies. Sin embargo, ambas estructuras carecían de escalabilidad y manejaban datos de k dimensiones.

A. Método de solución propuesta:

En vista de lo expuesto anteriormente, surgió la propuesta de crear una estructura que solo maneje datos de naturaleza tridimensional: el Octree. Esta estructura es implementada para que sea lo más eficiente posible en cuanto le brindes data de naturaleza 3D, tales como coordenadas espaciales.

A grandes rasgos, el Octree gestiona los datos espaciales tridimensionales dividiendo recursivamente el espacio en ocho subespacios cúbicos (octantes). A diferencia del K-d tree, que realiza divisiones binarias a lo largo de un solo eje en cada nivel, el Octree divide el volumen completo en partes iguales desde el punto central del cubo que representa ese nodo. Esta división geométrica y uniforme permite representar eficientemente espacios 3D complejos, especialmente cuando los datos están dispersos de forma irregular.

El aspecto clave del Octree es que su profundidad se adapta dinámicamente según la densidad de los datos. Cada nodo representa un cubo que abarca una porción del espacio tridimensional y, si contiene más de un objeto o excede el límite determinado, se subdivide en ocho subcubes iguales. Esta subdivisión ocurre simultáneamente a lo largo de los tres ejes, generando así ocho octantes hijos que juntos cubren el mismo volumen que el cubo padre. Este proceso puede repetirse recursivamente en los nodos hijos si estos también exceden los criterios definidos. A diferencia de otras estructuras como los K-d trees, el Octree no divide por planos, sino por volumen completo, lo que permite una representación espacial más natural y balanceada para datos tridimensionales. Esta estructura facilita operaciones eficientes de inserción, búsqueda y eliminación, sin perder su coherencia geométrica como se muestra en la Figura 3.

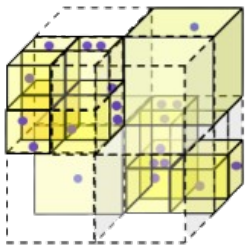

Fig. 3. Particionado espacial 3D de Octree, según [4]

V. Implementación y Operaciones Básicas

Un Octree es una estructura jerárquica que particiona de forma recursiva un espacio cúbico en ocho subcubes. Inicialmente el árbol está vacío; tras la primera inserción

se crea el nodo raíz con centro y longitud de lado definidos. A medida que agregamos puntos, se desciende por los nodos adecuados (minimizando la contención espacial) y, si un nodo supera su capacidad máxima, se subdivide. Tras la subdivisión, sus puntos se redistribuyen y, cuando la carga de varios hijos baja de un umbral, estos se fusionan (merge) de nuevo. A continuación describimos en detalle las operaciones más comunes:

A. Inserción

La inserción en un Octree comienza en la raíz, descendiendo recursivamente al hijo que contenga el punto. Si el nodo hoja no está lleno y puede seguir subdividiéndose, se almacena el punto; en caso contrario, se divide y redistribuyen los puntos.

Case I: Inserción en nodo con capacidad Se almacena directamente en el vector interno, sin subdividir.

Algorithm 1 Insertar punto en Octree

```
1:  function InsertarOctree(nodo, punto p)
2:      if not contains(nodo, p) then
3:          return false
4:      end if
5:      if |nodo.points| < MAX_POINTS ∧
    nodo.sidelength/2 ≥ minSidelength then
6:          nodo.points.PUSH_BACK(p)
7:          return true
8:      else
9:          if not nodo.tieneHijos then
10:             subdividir(nodo)
11:         end if
12:         for all hijo en nodo.children do
13:             if InsertarOctree(hijo, p) then
14:                 return true
15:             end if
16:         end for
17:         nodo.points.PUSH_BACK(p)
18:         return true
19:     end if
20: end function
```

Case II: Subdivisión de nodo lleno

Algorithm 2 Subdividir nodo

```
1: function subdividir(nodo)
2:     h ← nodo.sidelength /2
3:     q ← nodo.sidelength /4
4:     for all (dx, dy, dz) ∈ {−1, +1}^3 do
5:         newCenter ← nodo.center +(dx·q, dy·q, dz·q)
6:         Crear hijo con
   (newCenter, h, nodo.minSidelength)
7:     end for
8:     redistribuirPuntos(nodo)
9: end function
```

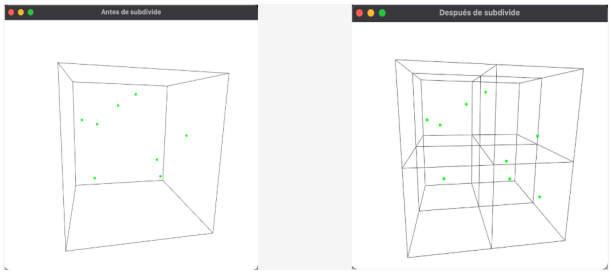

Fig. 4. Representación del antes y después del insertar un punto, elaboración propia

Case III: Redistribución y merge

Algorithm 3 redistribuirPuntos y merge

```
1: function redistribuirPuntos(nodo)
2:     temp ← nodo.points
3:     nodo.points.clear()
4:     for all p en temp do
5:         if not InsertarOctree(algún hijo, p) then
6:             nodo.points.push_back(p)
7:         end if
8:     end for
9:     if total de puntos en hijos < MAX_POINTS then
10:        merge(nodo)
11:    end if
12: end function
```

B. Búsqueda de Vecinos Cercanos

El método getNearby recupera todos los puntos dentro de un radio r alrededor de un punto de consulta q. Se basa en una poda espacial eficaz y en un filtrado puntual, y consta de tres fases:

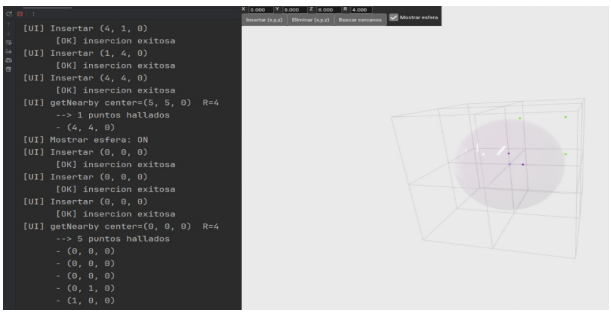

Fig. 5. Representación de la búsqueda de vecinos más cercanos, elaboración propia

1) **Poda inicial (líneas 2-3):** Se comprueba la intersección entre la *Axis-Aligned Bounding Box* (AABB) del nodo y la esfera de búsqueda centrada en q con radio r. Si no se intersectan, se descarta todo el subárbol en tiempo $O(1)$.
2) **Filtrado local (líneas 5-9):** Para cada punto p almacenado en el nodo, se calcula la distancia al cuadrado

$$d^2 = (p.x - q.x)^2 + (p.y - q.y)^2 + (p.z - q.z)^2$$

y se compara con r^2. Sólo los puntos que cumplen $d^2 \leq r^2$ se añaden al resultado.

3) **Recursión en hijos (líneas 11-13):** Si el nodo contiene hijos, se invoca recursivamente getNearby en cada uno, aplicando las mismas dos fases anteriores.

Algorithm 4 GETNEARBY en `OctreeNode`

```
1: function GETNEARBY(nodo, q, r, resultados)
2:     if not AABBVSESFERA(nodo.center, nodo.sidelength, q, r) then
3:         return
4:     end if
5:     for all p en nodo.points do
6:         d² ← (p.x − q.x)² + (p.y − q.y)² + (p.z − q.z)²
7:         if d² ≤ r² then
8:             resultados.push_back(p)
9:         end if
10:    end for
11:    if nodo.tieneHijos then
12:        for all h en nodo.children do
13:            GETNEARBY(h, q, r, resultados)
14:        end for
15:    end if
16: end function
```

C. Eliminación

Para borrar un punto, se busca recursivamente; tras eliminarlo, si los hijos quedan con pocos puntos, se fusionan.

Algorithm 5 ELIMINAROCTREE: eliminar punto en `Octree`

```
1: function ELIMINAROCTREE(nodo, p)
2:     if not CONTAINS(nodo, p) then
3:         return false
4:     end if
5:     if p ∈ nodo.points then
6:         Eliminar p de nodo.points
7:         return true
8:     end if
9:     if nodo.tieneHijos then
10:        for all h en nodo.children do
11:            if ELIMINAROCTREE(h, p) then
12:                if total de puntos en hijos < MAX_POINTS then
13:                    MERGE(nodo)
14:                end if
15:                return true
16:            end if
17:        end for
18:    end if
19:    return false
20: end function
```

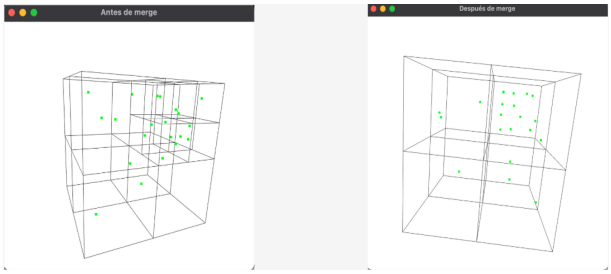

Fig. 6. Representación del antes y después del remover un punto, elaboración propia

D. Funciones Auxiliares

Algorithm 6 CONTAINS: prueba de pertenencia en `AABB`

```
1: function CONTAINS(nodo, p)
2:     h ← nodo.sidelength /2
3:     if |p.x − nodo.center.x| > h or |p.y − nodo.center.y| > h or |p.z − nodo.center.z| > h then
4:         return false
5:     end if
6:     return true
7: end function
```

Algorithm 7 MERGE: fusión de hijos con pocos puntos

```
1: function MERGE(nodo)
2:     if not nodo.tieneHijos then
3:         return
4:     end if
5:     collector ← {}
6:     for all h en nodo.children do
7:         COLLECTALLPOINTS(h, collector)
8:         delete h; h ← NULL
9:     end for
10:    nodo.points ← nodo.points + collector
11: end function
```

VI. Complejidad de las Operaciones

Para analizar el coste de las operaciones en un Octree, definimos los siguientes parámetros:

- N: número total de puntos almacenados en el Octree.
- $M = 8$: número máximo de hijos por nodo (constante en un octree).
- h: altura del árbol, en un escenario equilibrado $h = O(\log_M N) = O(\log_8 N)$.
- k: número de puntos reportados en una consulta de rango.

A. Inserción

- Caso promedio: cada inserción desciende un camino de longitud h, comprobando **contains** y, en ocasiones, subdivisiones cuyo coste amortizado es constante.

$$T_{\text{ins,prom}} = O(h) = O(\log_8 N).$$

- Peor caso: si siempre subdividimos hasta alcanzar minSidelength, el árbol tiene altura h máxima y no hay poda adicional, sigue siendo $O(h) = O(\log_8 N)$.

B. Búsqueda de Vecinos Cercanos

- Caso promedio: la poda de ramas no relevantes nos lleva a inspeccionar $O(h)$ nodos, y en cada uno examinamos sólo sus puntos locales. Además, se reportan k resultados:

$$T_{\text{busq,prom}} = O(h + k) = O(\log_8 N + k).$$

- Peor caso: si la esfera de búsqueda cubre todo el espacio (o el árbol está muy desequilibrado), se recorren todos los puntos:

$$T_{\text{busq,peor}} = O(N).$$

C. Eliminación

- Caso promedio: igual que la inserción, descendemos h niveles para hallar y borrar el punto, más un posible merge amortizado en tiempo constante.

$$T_{\text{elim,prom}} = O(h) = O(\log_8 N).$$

- Peor caso: mismo análisis que el promedio, pues la fusión de nodos no altera la cota asintótica.

$$T_{\text{elim,peor}} = O(\log_8 N).$$

TABLE II
COMPLEJIDAD DE LAS OPERACIONES EN UN OCTREE

Operación	Caso promedio	Peor caso
Inserción	$O(\log_8 N)$	$O(\log_8 N)$
Búsqueda	$O(\log_8 N + k)$	$O(N)$
Eliminación	$O(\log_8 N)$	$O(\log_8 N)$

VII. Ventajas

1) **Eficiencia en la subdivisión espacial tridimensional:**
   - El Octree divide el espacio de forma recursiva en ocho regiones hijas, lo cual permite representar eficientemente escenas 3D complejas y dispersas. Esto facilita operaciones como búsqueda, inserción y eliminación en tiempo sublogarítmico en espacios equilibrados [5].
2) **Reducción en la complejidad de colisiones y renderizado:**
   - En entornos como videojuegos y simulaciones físicas, el uso del Octree permite reducir la cantidad de comparaciones requeridas en detección de colisiones y renderizado, mejorando el rendimiento general [12], [13].
3) **Optimización de consultas espaciales:**
   - Operaciones como la búsqueda de objetos cercanos dentro de un radio definido, intersecciones o conteo de entidades en un volumen específico se ejecutan más rápido debido a la estructura jerárquica del Octree [3], [15].
4) **Uso eficiente de memoria para datos dispersos:**
   - El Octree almacena solo los nodos necesarios, evitando desperdicio de memoria en regiones vacías [5], [15].

VIII. Desventajas

1) **Sobrehead en estructuras densas o balance deficiente:**
   - En entornos donde los datos están muy concentrados, el Octree puede volverse ineficiente, generando un gran número de subdivisiones y afectando el tiempo de acceso [5].
2) **Difícil de mantener balance óptimo:**
   - Aunque no es un árbol de búsqueda tradicional, el balance del Octree depende de la distribución espacial de los objetos, y no existen mecanismos automáticos de balanceo como en AVL-trees [5].
3) **No ideal para datos dinámicos altamente móviles:**
   - Si los objetos cambian de posición frecuentemente, se requiere reinsertarlos constantemente, lo que puede rebajar el rendimiento [14].
4) **Mayor complejidad algorítmica para operaciones masivas:**
   - Cuando se realizan operaciones sobre todos los nodos (como compresión o serialización), la estructura jerárquica puede dificultar una implementación eficiente [16].

IX. Conclusiones

El Octree se define como una estructura de datos adecuada para la representación eficiente de espacios tridimensionales, destacando por su capacidad de subdivisión jerárquica y su adaptabilidad en dominios como gráficos por computadora, robótica y simulación. Asimismo, se han descrito y analizado los algoritmos de inserción, búsqueda y eliminación, mostrando cómo estas operaciones se adaptan a la estructura del Octree para mantener su eficiencia. La implementación práctica de una simulación en Unity permitió validar visualmente su funcionamiento, resaltando el proceso de subdivisión y la localización de objetos en un radio definido.

No obstante, también se ha reconocido que el Octree no está exento de limitaciones, especialmente en entornos altamente dinámicos o con alta densidad de datos, donde la sobrecarga estructural puede afectar el desempeño. Frente a estas situaciones, puede ser más adecuado optar por estructuras alternativas como el k-d trees, dependiendo del dominio y los requisitos de aplicación.

En conclusión, el Octree sigue siendo una herramienta valiosa y vigente en el manejo de datos espaciales tridimensionales, cuya elección debe evaluarse cuidadosamente

según las características del problema a resolver. Su comprensión y correcta implementación pueden proporcionar mejoras sustanciales en sistemas que requieren operaciones espaciales eficientes.

References

[1] OpenGenus, "Octree: An efficient 3D data structure," [En línea]. Disponible: https://iq.opengenus.org/octree/. [Accedido: mayo 2025].

[2] J. Six, "Frustum culling," [En línea]. Disponible: https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling. [Accedido: 17 mayo 2025].

[3] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[4] H. Lei, N. Akhtar, and A. Mian, "Octree guided CNN with spherical kernels for 3D point clouds," *arXiv preprint arXiv:1903.00343*, 2019.

[5] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Heidelberg: Springer-Verlag, 2008.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, Sep. 1977.

[8] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, Jun. 1982.

[9] A. M. Mirzendehdel, M. Behandish, and S. Nelaturi, "Octree data structure for support accessibility and removal analysis in additive manufacturing," *Additive Manufacturing*, vol. 29, pp. 100811, 2019.

[10] CGAL, "Orthtree: A generic octree data structure," CGAL – *Computational Geometry Algorithms Library*, Apr. 27, 2021. [Online]. Available: https://www.cgal.org/2021/04/27/Orthtree/

[11] B. Lei, X. Liu, and Y. Zhu, "Octree guided CNN with spherical kernels for 3D point clouds," *ResearchGate*, 2019. [Online]. Available: https://www.researchgate.net/publication/331485436_Octree_guided_CNN_with_Spherical_Kernels_for_3D_Point_Clouds

[12] J. Amanatides, "Ray tracing through an octree" *Proc. 1987 Workshop on Interactive 3D Graphics*, 1987.

[13] Unity Technologies, "Optimizing large scenes with spatial partitioning," *Unity Docs*. [Online]. Available: https://docs.unity3d.com

[14] Epic Games, "Level Streaming and World Partition in Unreal Engine," *Unreal Engine Documentation*, 2023. [Online]. Available: https://docs.unrealengine.com

[15] A. Hornung et al., "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013. doi: 10.1007/s10514-012-9321-0

[16] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit (VTK)*, 4th ed. Kitware, 2006.