Aplicación de curvas de llenado del espacio en nubes de puntos 3D almacenadas en Octrees

Pablo Díaz, Miguel Yermo, José C. Cabaleiro, Francisco F. Rivera

Junio 2025

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
CAMPUS VIDA
CITIUS Centro Singular de Investigación en Tecnoloxias Intelixentes

Índice

1. Búsquedas de vecinos

2. Curvas de llenado del espacio (SFCs)

3. Octrees

4. Resultados

Búsquedas de vecinos

Dada una nube de puntos tridimensional $\mathcal{P}$, un centro c y un radio r, buscamos encontrar todos los puntos $q \in \mathcal{P}$ con $\|q - c\| < r$.
Esta operación extrae la estructura local de la nube, y tiene innumerables usos en el procesado de nubes de puntos: clusterización, segmentación, extracción de propiedades, etc.

Problema: extraer los vecinos de un gran número de puntos es costoso, y de hecho suele ser la parte más computacionalmente demandante del procesado de la nube.

Nubes LiDAR

- Gran tamaño, altamente irregulares tanto en densidad como en geometría
- Varios tipos (sensor aéreo vs terrestre)
- Necesidad de algoritmos y estructuras de datos optimizados para el cálculo de vecinos

→ Conjunto de nubes variado: datasets *Paris-Lille*, *DALES*, *Semantic3D* y *Speulderbos*.

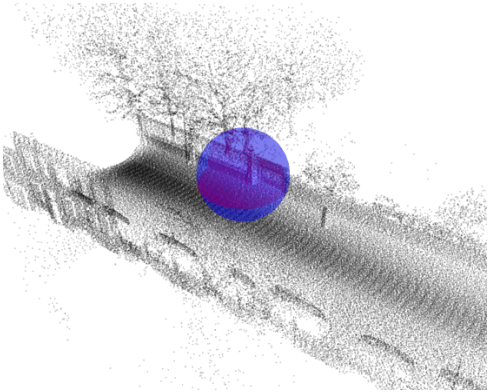


Figura: Vecindario esférico en la nube *Lille_0*

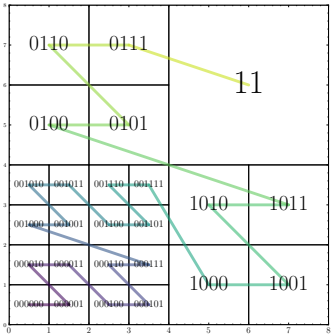
Curvas de llenado del espacio

- Normalmente, los datos de las nubes se almacenan como un vector de puntos, en el mismo orden de escaneo del sensor LiDAR → **Mala localidad espacial**
- Puntos muy cercanos en el espacio tridimensional pueden encontrarse muy alejados en memoria
- ¿Cómo mejorar la localidad? → Reordenando la nube mediante una curva de llenado del espacio *(**S**pace **F**illing **C**urve)* [1]

SFCs de Morton y Hilbert

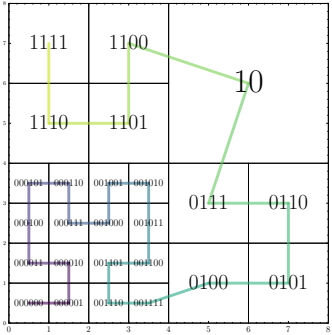
SFC de Morton [6]

- Presenta saltos, menor localidad
- Cálculo en unas pocas instrucciones (LUTs, vectorización)



SFC de Hilbert [4, 5]

- Continua, mejor localidad teórica (continuidad en $\|\cdot\|_\infty$)
- Cálculo iterativo más lento

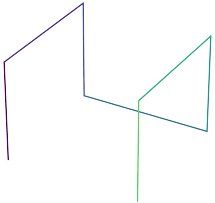


Reordenando la nube

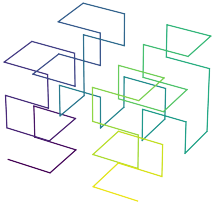
1. Discretización: hallar la *bounding box* y trasladar todos los puntos al espacio $S_L = [0, 2^L) \times [0, 2^L) \times [0, 2^L) \subset \mathbb{Z}^3$.
2. Para cada punto $p = (x, y, z) \in S_L$, calcular su código c de Morton o de Hilbert de $3L$ bits, marcando su orden en la curva de profundidad L.
3. Una vez se han hallado los códigos, **utilizarlos como índice para reordenar**.

Reordenando la nube

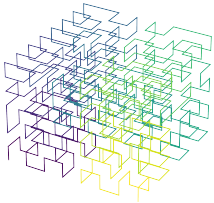
1. Discretización: hallar la *bounding box* y trasladar todos los puntos al espacio $S_L = [0, 2^L) \times [0, 2^L) \times [0, 2^L) \subset \mathbb{Z}^3$.
2. Para cada punto $p = (x, y, z) \in S_L$, calcular su código c de Morton o de Hilbert de $3L$ bits, marcando su orden en la curva de profundidad L.
3. Una vez se han hallado los códigos, **utilizarlos como índice para reordenar**.



Hilbert $L = 1$



Hilbert $L = 2$



Hilbert $L = 3$

Reordenando la nube

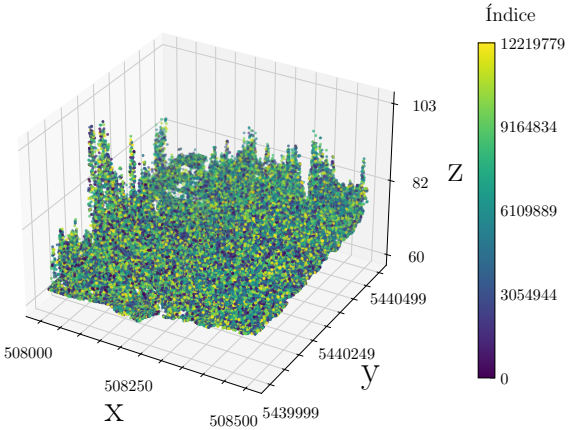
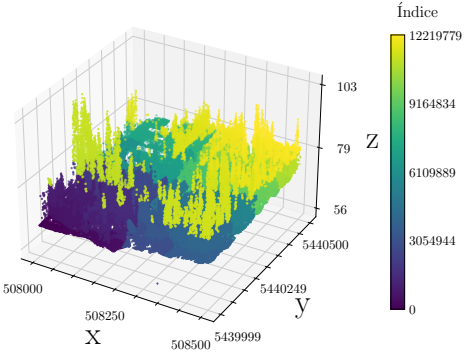
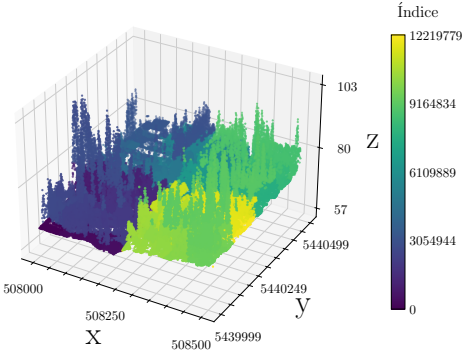


Figura: Nube original del dataset *DALES*, con color según orden de índices.

Reordenando la nube



(a) Morton, $L = 21$



(b) Hilbert, $L = 21$

Figura: Nube reordenada con dos curvas distintas.

Octrees

1. Estructura jerárquica de volúmenes (*BVH*) donde el espacio se subdivide recursivamente en 8 octantes.
2. Los nodos internos tienen 8 suboctantes, las hojas tienen puntos de la nube.
3. Se subdivide hasta que las hojas no tienen más de N_{max} puntos.

¿Cómo almacenar los nodos?

→ Octrees basados en punteros
→ Octrees lineales

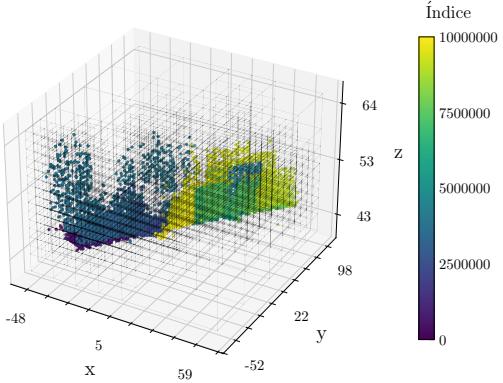


Figura: Construcción de un Octree sobre la nube Lille_0, reordenada mediante la curva de Hilbert

Búsqueda de vecinos en Octrees

Algoritmo de búsqueda en profundidad

1. Si el nodo es una rama, calculamos su posición relativa respecto al *kernel*:
   - 1a. Octante interseca al *kernel* → Seguir descendiendo.
   - 1b. Octante fuera del *kernel* → Parar.
2. Si llegamos a una hoja, comprobamos punto a punto si pertenecen o no al *kernel* propuesto.
3. Finalmente, devolvemos todos los puntos en el interior del *kernel*.

Ventajas

→ Mucho mejor que fuerza bruta: no se comprueban todos los puntos.
→ Los puntos de las hojas están próximos en memoria gracias a las SFCs.

Desventajas

→ Se debe recorrer el árbol hasta las hojas
→ Muchas redirecciones de punteros, peor localidad del árbol

Optimización con Octrees lineales

El Octree lineal propuesto es una variante de la estructura *cornerstone-octree* dada por Keller et al. [3], originalmente usada en simulaciones de partículas.

Ideas principales

→ Cada hoja o nodo interno se puede representar mediante el **rango de índices** de la nube reordenada, gracias al reordenamiento por SFCs.
→ La información para enlazar el árbol y realizar las búsquedas también se puede comprimir en un array indicando el primer hijo de cada octante.

- Por ejemplo, bajo el i-ésimo hijo del nodo raíz ($0 \leq i \leq 7$) tenemos los puntos con SFC en $[i \cdot 8^{L-1}, (i+1) \cdot 8^{L-1})$, con índices en el rango $[a_i, a_{i+1})$, siendo $a_0 = 0$ y $a_8 = N$.

El Octree lineal cuenta con las siguientes características:

1. Construcción rápida y altamente paralelizable.
2. **Rangos de índices en la rama disponible en todos los nodos, y consecutivos gracias a las SFCs.**
3. Estructura compacta y contenida en unos pocos arrays: muy buena localidad y poco overhead en memoria.

Algoritmo optimizado de búsquedas

1. Si el nodo es una rama, calculamos su posición relativa respecto al *kernel*:
   - 1a. **Octante contenido en kernel → Insertar puntos secuencialmente, usando la estructura lineal con acceso directo a los puntos.**
   - 1b. Octante interseca al *kernel* → Seguir descendiendo.
   - 1c. Octante fuera del *kernel* → Parar.
2. Si llegamos a una hoja, comprobamos punto a punto si pertenecen o no al *kernel* propuesto.
3. Finalmente, devolvemos todos los puntos en el interior del *kernel*.

Configuración experimental

→ 2 experimentos principales:

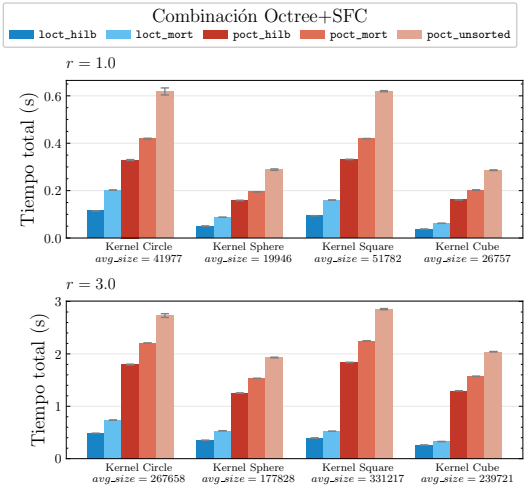
1. **Búsquedas aleatorias:** 5000 búsquedas en un subconjunto de centros aleatorio $v_c \subset P$, con varios radios, *kernels* (esfera, círculo, cubo, cuadrado) y combinaciones SFC+Octree.
2. **Búsquedas completas:** lo mismo pero con $v_c = P$, y realizando las búsquedas con el orden de la nube tras el reordenamiento, para más localidad.

Octree	Reordenamiento	Nombre
Punteros	Ninguno	`poct_unsorted`
	Morton	`poct_mort`
	Hilbert	`poct_hilb`
Lineal	Morton	`loct_mort`
	Hilbert	`loct_hilb`

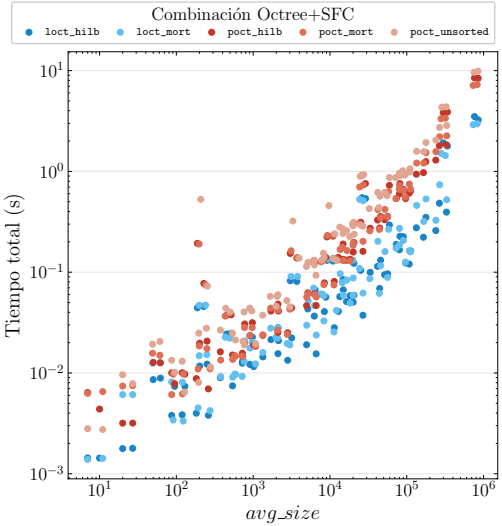
Configuración experimental

→ Paralelización a nivel de bucle con OpenMP, 40 hilos sobre arquitectura NUMA.

Procesador: *4× Intel(R) Xeon(R) CPU E5-4620 v4 @ 2.10 GHz*
Núcleos: *10 núcleos físicos por procesador (40 en total)*
Arquitectura: *Broadwell*
Memoria RAM: *251 GiB*
Sistema Operativo: *Alma Linux 8.6 (Sky Tiger) Kernel 4.18.0*
Compilador: *GCC 12.1.1 (-O2)*
Opciones de compilación: *-O2*



Búsquedas aleatorias - Tiempos en *Paris_Luxembourg_6*, para $r = 1.0m$ y $r = 3.0m$ y varios kernels



Búsquedas aleatorias - Tiempo vs tamaño promedio de los vecindarios en varias nubes (log-log)

Resultados

Búsquedas completas - Varias nubes, kernel esférico

Nube	n	r (m)	Octree	SFC	Tiempo (s)
Lille_0	10.00M	3.0	Punteros	-	445.24
				Morton	332.11
				Hilbert	324.87
			Lineal	Morton	130.96
				Hilbert	**130.20**
5080_54400	12.22M	10.0	Punteros	-	98.04
				Morton	64.99
				Hilbert	62.80
			Lineal	Morton	**42.66**
				Hilbert	46.31
bildstein_station1	29.70M	0.1	Punteros	-	118.44
				Morton	106.35
				Hilbert	103.08
			Lineal	Morton	**60.66**
				Hilbert	65.11
sg27_station8	429.62M	0.05	Punteros	-	2549.36
				Morton	2243.24
				Hilbert	2208.13
			Lineal	Morton	**1273.68**
				Hilbert	1427.66

Conclusiones

- El reordenamiento por curvas de Morton y Hilbert mejora la localidad de la nube, lo cuál reduce el tiempo de las búsquedas.
- La implementación empleada para el Octree lineal aporta una mejora muy considerable frene a la estructurada basada en punteros.
- La reordenación mediante SFCs y el Octree lineal tienden a mejorar el coeficiente de paralelización de las búsquedas.

Últimos avances y trabajo futuro

→ Últimos avances:

- Método de búsqueda en Octree lineal devolviendo rangos de índices en vez de lista de punteros / índices → Gran mejora para r grande.
- Mejorada la comprobación geométrica para kernels esféricos, a partir del trabajo de Behley et al. [2]
- Con todo esto, búsquedas mucho más rápidas ($\approx 10x$) que los Octrees y KD-Trees de PCL y $2-3x$ veces más rápidas que el Octree desarrollado por Behley et al. [2]

→ Trabajo futuro:

- Probar búsquedas k-NN.
- Comparar con más librerías.
- Profiling de memoria.
- ...

¡Muchas gracias!

pablo.diaz.vinambres@rai.usc.es

Referencias I

[1] Tetsuo Asano et al. "Space-filling curves and their use in the design of geometric data structures". En: *Theoretical Computer Science* 181.1 (1997), págs. 3-15.

[2] Jens Behley, Volker Steinhage y Armin B Cremers. "Efficient radius neighbor search in three-dimensional point clouds". En: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, págs. 3625-3630.

[3] Sebastian Keller et al. "Cornerstone: Octree Construction Algorithms for Scalable Particle Simulations". En: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '23. ACM, jun. de 2023, 1–10. DOI: 10.1145/3592979.3593417. URL: http://dx.doi.org/10.1145/3592979.3593417.

[4] Warren M Lam y JH Shapiro. "A class of fast algorithms for the Peano-Hilbert space-filling curve". En: *Proceedings of 1st International Conference on Image Processing*. Vol. 1. IEEE. 1994, págs. 638-641.

Referencias II

[5] Yohei Miki y Masayuki Umemura. "GOTHIC: Gravitational oct-tree code accelerated by hierarchical time step controlling". En: *New astronomy* 52 (2017), págs. 65-81.

[6] Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. IBM, Ottawa, Canada. 1966.