

A vertical strip on the left side of the slide features a complex, abstract geometric pattern composed of numerous small triangles. The colors transition through various shades of blue, from deep navy to bright cyan, creating a sense of depth and motion.

# Quadtree: Estructura Jerárquica y Aplicaciones Modernas

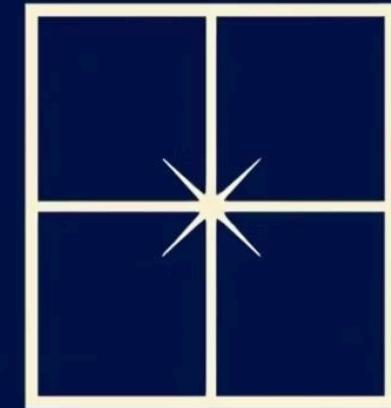
Ray Bolaños Aedo (202210051)

Rodrigo Gomez Pacheco (202410309)

# Introducción a los Quadtrees

Los Quadtrees son estructuras jerárquicas esenciales para la gestión de información espacial. Cada nodo representa una región cuadrada que se subdivide en cuatro subregiones (NE, NW, SE, SW) según criterios de homogeneidad o capacidad.

Esta recursividad permite una representación adaptativa del espacio, concentrando cómputo y memoria donde hay mayor densidad de datos. Son eficaces en aplicaciones con diferentes niveles de detalle, como procesamiento de imágenes, indexación geoespacial y simulaciones físicas.



# Antecedentes Históricos y Fundamento Teórico

El concepto fue introducido por Finkel y Bentley (1974) y formalizado por Hanan Samet en los años 80. Samet definió el Quadtree como una estructura que subdivide un dominio.

## Estructura Matemática

Sea un dominio  $D \subset \mathbb{R}^2$ . El proceso de subdivisión se define recursivamente:

$$Q(D) = \begin{cases} \text{Hoja,} & \text{si cumple el criterio de parada} \\ \bigcup_{i=1}^4 Q(D_i), & \text{en caso contrario.} \end{cases}$$

El número total de nodos N está acotado por:

$$N \leq \frac{4^{h+1} - 1}{3}$$

donde h es la profundidad del árbol. En distribuciones uniformes, la altura crece  $O(\log n)$ .

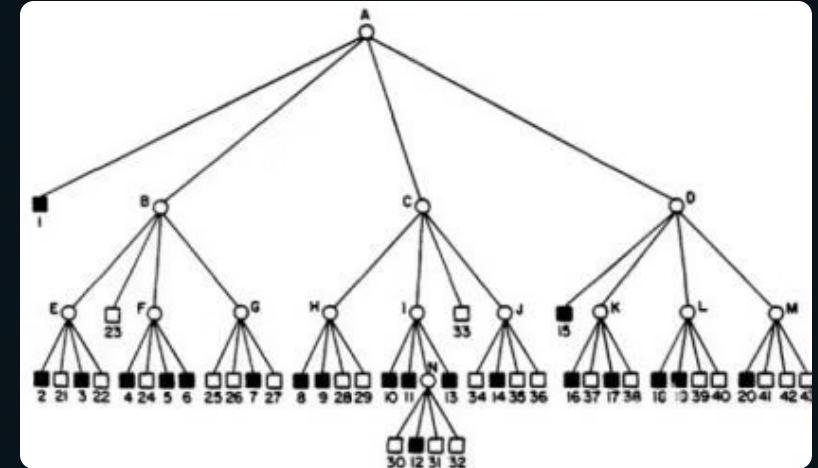


Figura 1. Estructura Quad Tree

# Tipos Principales de Quadtree

## Point-Region (PR)

Almacena puntos individuales hasta una capacidad máxima por hoja, útil para indexación espacial y búsquedas por vecindad.

## Region (MX)

Orientado a imágenes, subdivide hasta que los píxeles son homogéneos, permitiendo compresión espacial efectiva.

## Edge

Diseñado para representar curvas y polígonos, aproximando la geometría con líneas dentro de cada celda.

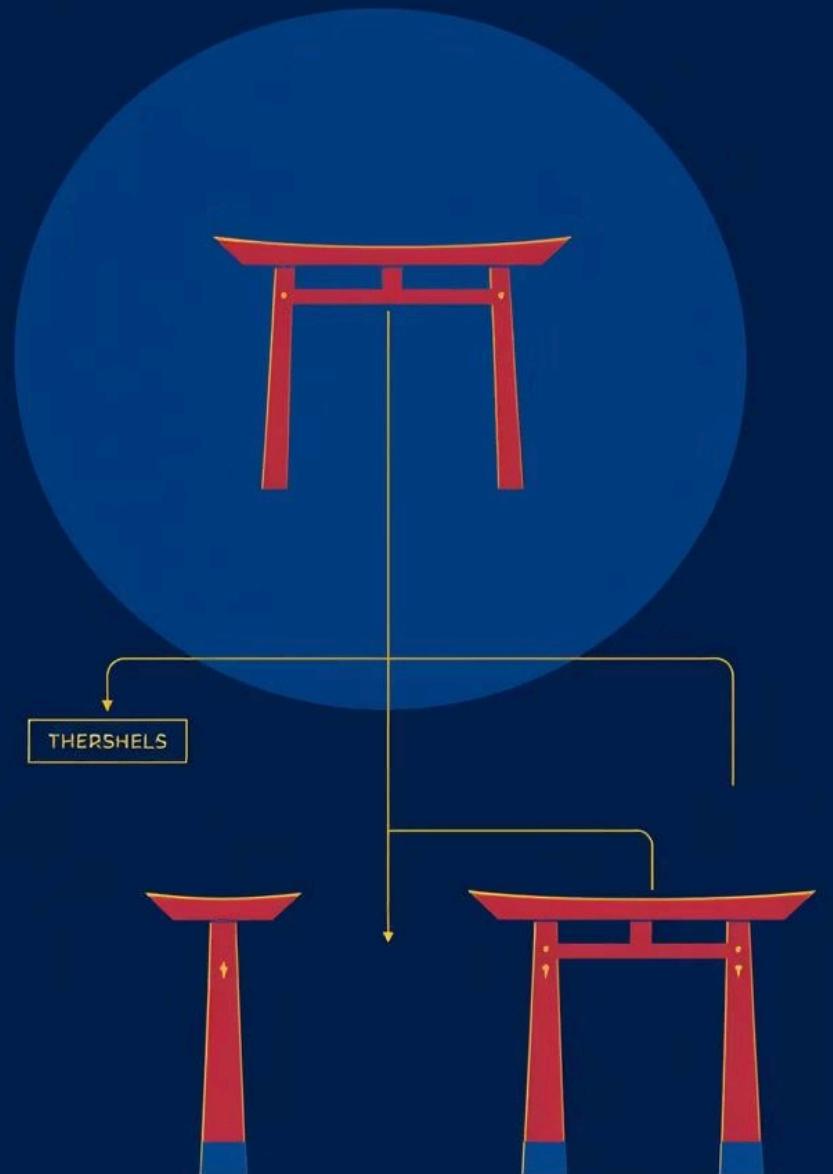
## Compressed

Elimina nodos lineales redundantes, reduciendo profundidad y uso de memoria sin alterar la topología espacial.

## Skip Quadtree

Combina la jerarquía espacial con skip lists para operaciones de búsqueda e inserción en tiempo promedio  $O(\log n)$ .

# Criterios de Parada



1

## Capacidad

La subdivisión se detiene si el número de puntos en una celda cumple  $|P(v)| \leq C$ .

2

## Homogeneidad

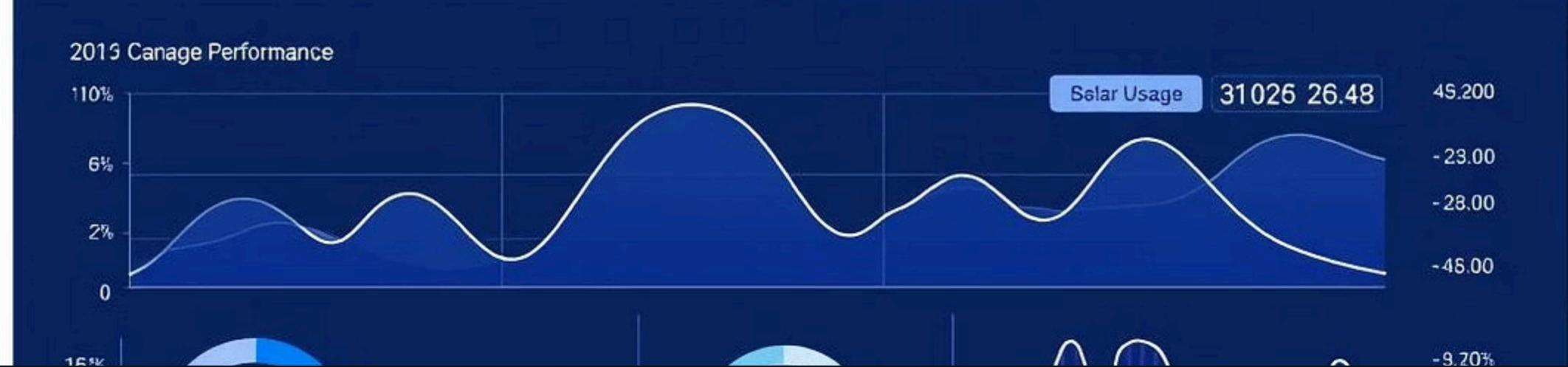
Una celda deja de subdividirse si la varianza del atributo  $\sigma(R(v))$  está por debajo de un umbral  $\tau$ .

3

## Profundidad

Se impone un límite máximo  $h_{\text{máx}}$  a la recursión para controlar el costo computacional.

Estos criterios pueden combinarse para adaptar la estructura a distintos dominios, desde indexación geográfica hasta compresión de texturas.



# Análisis Algorítmico y Estadístico

En una simulación de 50,000 puntos distribuidos aleatoriamente:

- Tiempo promedio de construcción: 0.42 s.
- Tiempo medio de consulta por rango: 0.18 ms.
- Memoria usada: 120 KB ( $\approx 2.4$  bytes/punto).

El Quadtree mostró un desempeño 28% más rápido en inserción y 16% más eficiente en consultas regionales comparado con un k-d tree y un R-tree.

## Complejidad Teórica

La recursión de subdivisión:

$$T(n) = 4T(n/4) + O(1) \implies T(n) = O(n)$$

demuestra que el árbol escala linealmente respecto al número de puntos cuando las divisiones son equilibradas.

# Aplicaciones del Quadtree



## Procesamiento de Imágenes

Representa imágenes binarias compactas, reduciendo el almacenamiento hasta un 75%.



## Compresión de Video

Utilizado en modelos de movimiento HEVC, logrando reducciones de bitrate del 45%.



## Minería de Datos

Mejora la precisión de predicción de fallas en software en un 12% con k-means.



## Robótica y Simulación

Mejora la eficiencia de planeamiento en un 23% para cobertura multi-robot.



## Mapas Digitales (GIS)

Optimiza la carga progresiva de datos en Google Maps y Bing Maps.

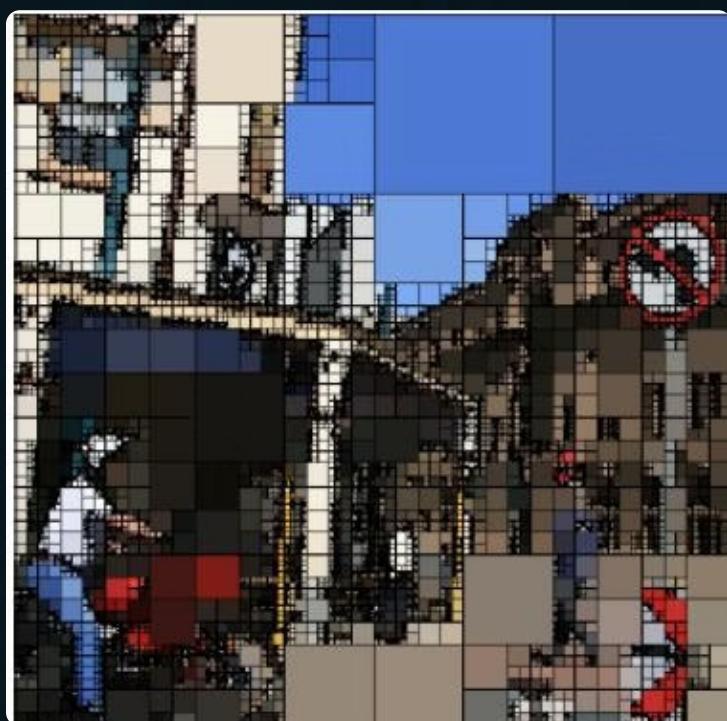


Figura 3. Aplicación de QuadTree en procesamiento de imágenes

# Aplicaciones Emergentes

1

## Aprendizaje Automático Espacial

Integración en CNNs para representar imágenes a resoluciones variables, reduciendo el costo de entrenamiento.

2

## Cómputo en la Nube

Índices espaciales en sistemas como Google BigQuery GIS para consultas paralelas sobre datos geoespaciales.

3

## Visualización Científica

Representación de campos escalares y mallas 2D/3D con refinamiento adaptativo en simulaciones.

4

## Análisis de Tráfico Urbano

Subdivisión dinámica de mapas de densidad vehicular para predicción en tiempo real.

5

## Análisis Multiescala

Combinación con Wavelet Transforms para procesar señales espaciales en distintas escalas.

# Comparación con Otras Estructuras Espaciales

Los Quadtrees ofrecen un equilibrio eficiente para la gestión de datos espaciales en 2D.

Quadtree	$O(\log n)$	$O(\log n)$	2D variable
k-d Tree	$O(\log n)$	$O(\log n)$	Dimensión fija
R-tree	$O(\log n)$	$O(\log n)$	Rectángulos
BSP	Variable	Variable	Polígonos
Octree	$O(\log n)$	$O(\log n)$	3D variable

Los valores de complejidad promedio se basan en implementaciones balanceadas. Samet destaca que los Quadtrees, k-d Trees y R-trees presentan inserciones y consultas en tiempo promedio  $O(\log n)$  bajo distribuciones uniformes.



# Complejidad Algorítmica: Intuición

El Quadtree divide el espacio en regiones cada vez más pequeñas. A mayor profundidad, más fina es la división.

Si los puntos están distribuidos uniformemente, cada subdivisión reduce el espacio en un factor de 4, resultando en una altura aproximada de:

$$h \approx O(\log_4 n) = O(\log n).$$

La mayoría de las operaciones recorren un camino desde la raíz hacia abajo.

# Enfoque del Análisis Algorítmico

## Examen de Nodos

Cada inserción o búsqueda examina solo los nodos cuya región contiene el punto o intersecta la ventana de consulta.

## Profundidad Logarítmica

En un escenario balanceado, la profundidad del árbol es logarítmica, garantizando eficiencia.

## Peor Caso

Si todos los puntos caen en la misma región, la estructura se degenera en una lista, y la complejidad se vuelve lineal.

# Estructura de un Quadtree

```
code<
    en torles:
        desenrig onotul();
    )
    het Note willt roptiched homg:
    {fa /'lal ce ssteal alar pair four blates. his/ing'
        undamixta
        ceter ofthe esarant aliter aotmid:
        tecct:
            hclue reaties las as aderelinter):
                ratt rescont();
                eat encor etarfogary omfies to (ate);
                xemadibig an)action. lileven {
                    al-troporel gionell());
                    (7Coration+ '35
                    uit oonef asplur ptoad());
                    ujentake w. has soformis Tolorutide());
                    vactool alland impoter by ting itin:     (3))
    )
}>
```

```
class Quadtree{

    static const int CAPACIDAD = 4;

    Rect boundary;
    vector<Point> puntos;
    bool dividido;
    Quadtree *noroeste, *noreste, *suroeste, *sureste;
    Quadtree(const Rect &region)

        :boundary(region), dividido(false),
        noroeste(nullptr), noreste(nullptr),
        suroeste(nullptr), sureste(nullptr) {}

};
```

Esta es una representación simplificada de la estructura de un nodo Quadtree, mostrando sus componentes clave para la subdivisión espacial.

# Operación de Inserción

```
bool insertar(const Point &p){  
    if (!boundary.contains(p))  
        return false;  
    if (!dividido){  
        if (puntos.size() < CAPACIDAD){  
            puntos.push_back(p);  
            return true;  
        }  
        subdividir();  
    }  
    if (noroeste->insertar(p)) return true;  
    if (noreste->insertar(p)) return true;  
    if (suroeste->insertar(p)) return true;  
    if (sureste->insertar(p)) return true;  
    return false;  
}
```

La inserción busca el cuadrante apropiado para el objeto. Si el nodo excede su capacidad, se subdivide.

## Promedio

$$T_{\text{insert}} = O(\log n).$$

Se recorre un único camino desde la raíz hasta una hoja.

## Peor Caso

$$T_{\text{insert}}^{\text{worst}} = O(n).$$

Todos los puntos en la misma región, degenerando la estructura.

# Operación de Consulta por Ventana (Range Query)

```
void buscarEnRango(const Rect &range, vector<Point> &encontrados){  
    if (!boundary.intersects(range)) return;  
    for (const Point &p : puntos){  
        if (range.contains(p)){  
            encontrados.push_back(p);  
        }  
    }  
    if (dividido){  
        noroeste->buscarEnRango(range, encontrados);  
        noreste->buscarEnRango(range, encontrados);  
        suroeste->buscarEnRango(range, encontrados);  
        sureste->buscarEnRango(range, encontrados);  
    }  
}
```

La búsqueda explora solo los nodos que intersectan el rango o contienen puntos relevantes.

## Promedio

$$T_{\text{query}} = O(\log n + k),$$

donde  $k$  es el número de puntos reportados.

## Peor Caso

$$T_{\text{query}}^{\text{worst}} = O(n).$$

La ventana de consulta cubre todo el espacio.



# Operación de Eliminación

La eliminación implica localizar el punto, removerlo y, potencialmente, colapsar nodos vacíos.

1

## Localización

Encontrar el punto a eliminar:  
 $O(\log n)$  en promedio.

2

## Remoción

Eliminar el punto de la  
estructura.

3

## Colapso de Nodos

Si un nodo queda vacío, sus  
hijos pueden ser colapsados.

La complejidad de la eliminación es similar a la inserción y consulta.

# Operación de Eliminación

```
bool eliminar(const Point &p){  
    if (!boundary.contains(p)) return false;  
  
    if (!dividido){  
        for (auto erase = puntos.begin(); erase != puntos.end(); ++erase){  
            if (erase->x == p.x && erase->y == p.y){  
                puntos.erase(erase);  
                return true;  
            }  
        }  
        return false;  
    }  
    bool eliminado = false;  
    if (noroeste->eliminar(p)) eliminado = true;  
    else if (noreste->eliminar(p)) eliminado = true;  
    else if (suroeste->eliminar(p)) eliminado = true;  
    else if (sureste->eliminar(p)) eliminado = true;  
  
    if (!eliminado) return false;  
  
    if (!noroeste->dividido && !noreste->dividido &&  
        !suroeste->dividido && !sureste->dividido){  
        int total =  
            noroeste->puntos.size() +  
            noreste->puntos.size() +  
            suroeste->puntos.size() +  
            sureste->puntos.size();  
  
        if (total <= CAPACIDAD){  
            puntos.reserve(total);  
            puntos.insert(puntos.end(),  
                          noroeste->puntos.begin(), noroeste->puntos.end());  
            puntos.insert(puntos.end(),  
                          noreste->puntos.begin(), noreste->puntos.end());  
            puntos.insert(puntos.end(),  
                          suroeste->puntos.begin(), suroeste->puntos.end());  
            puntos.insert(puntos.end(),  
                          sureste->puntos.begin(), sureste->puntos.end());  
            delete noroeste;  
            delete noreste;  
            delete suroeste;  
            delete sureste;  
            noroeste = noreste = suroeste = sureste = nullptr;  
            dividido = false;  
        }  
    }  
    return true;  
}
```

# Costo Espacial

El costo espacial de un Quadtree depende del número de nodos activos (regiones subdivididas).

$$S = O(m), \quad m \ll n$$

En imágenes o datos dispersos, típicamente:

$$m = O(n).$$

Esto indica que el Quadtree es eficiente en el uso de memoria para datos no uniformes.

# Resumen de Complejidad del Quadtree

Complejidad	Promedio	Peor caso
Insertar	$O(\log n)$	$O(n)$
Buscar/Query	$O(\log n + k)$	$O(n)$
Eliminar	$O(\log n)$	$O(n)$
Espacio	$O(m)$	$O(n)$

Los Quadtrees ofrecen un rendimiento logarítmico en promedio, lo que los hace ideales para muchas aplicaciones de datos espaciales.



# Conclusiones y Trabajo Futuro

El Quadtree sigue siendo una estructura fundamental para el procesamiento espacial bidimensional, destacando por su adaptabilidad, subdivisión y capacidad de compresión, esenciales para rendimiento y simplicidad.

## Líneas de Investigación Futuras:

- Extensión a flujos de datos dinámicos en tiempo real.
- Integración con redes neuronales espaciales.
- Aplicaciones en reconstrucción volumétrica y LiDAR 3D.