Estructuras Geométricas Jerárquicas para la Modelización de Escenas 3D

P.Brunet, F.J.Santisteve, A.Vilanova*
L.Chiarabini†
G.A.Patow‡
E.Staffetti§
J.Surinyac¶

{pere,lucac,gpatow,fjsantis,jordis,annav}@lsi.upc.es
estaffetti@iri.upc.es

26 de febrero de 1999

El presente trabajo incluye una descripción de las principales estructuras de datos jerárquicas utilizadas en la modelización geométrica de escenas 3D, junto con los algoritmos básicos asociados, algunas aplicaciones recientes y una breve discusión comparativa. El trabajo es el resultado del curso de doctorado "Estructuras geométricas jerárquicas"impartido durante el segundo quatrimestre del curso académico 1997/98 dentro del programa de doctorado de Software de la UPC.

*Departamento de Lenguajes y Sistémas Informáticos (LSI-UPC).
†Hewlett-Packard.
‡Instituto de Matemática Aplicada (IMA-UdG).
§Instituto de Robótica e Informática Industrial (IRI-UPC).
¶Departamento de Lenguajes y Sistémas Informáticos (LSI-UVic).

Índice General

1 Introducción.

El presente trabajo incluye una descripción de las principales estructuras de datos jerárquicas utilizadas en la modelización geométrica de escenas 3D, junto con los algoritmos básicos asociados, algunas aplicaciones recientes y una breve discusión comparativa. El trabajo es el resultado del curso de doctorado "Estructuras geométricas jerárquicas" impartido durante el segundo quatrimestre del curso académico 1997/98 dentro del programa de doctorado de Software de la UPC.

Las primeras cuatro secciones presentan las principales estructuras de datos incluyendo para cada una de ellas sus principales caracteristicas, las alternativas básicas para su representación e implementación y los principales algoritmos asociados (transformación geométrica, algoritmos de búsqueda, construcción y visualización, localización de vecinos en direcciones geométricas determinadas, y balanceado de los árboles). En la última de estas secciones (cuarta) se incluye una tabla comparativa que caracteriza y clasifica las distintas estructuras de datos. En relación al espacio de trabajo, podemos clasificar las estructuras entre las basadas en la subdivisión en espacio imagen (octrees o árboles octales, bintrees, los diversos tipos de quadtrees -point quadtrees.. - y los vector octrees) y las que trabajan en espacio objecto (k-d trees, BSP, R-trees, prism-trees, sphere trees, BV-trees y los diferentes tipos de Box-trees); las primeras - subdivisión en espacio imagen - subdividen recursivamente el espacio de forma fija e independiente de la posición concreta de los objetos de la escena; las segundas trabajan en espacio objeto lo que significa que la subdivisión o agrupación se apoya en los objetos de la escena y por tanto depende de su posición concreta. Por lo que respecta a la semántica de la jerarquia, en algunos casos el árbol representa una subdivisión del espacio (en los diversos tipos de quadtrees y octrees, k-d trees y BSP), en otros casos representa una subdivisión de los objetos de la escena (prism trees, R-trees) y finalmente puede representar una agrupación de objetos o elementos geométricos cercanos espacialmente (box trees, sphere trees, BV-trees).

La mencionada tabla comparativa (sección 4) incluye otras clasificaciones como la relativa al criterio de hoja (criterio para determinar, tanto en el caso de subdivisión como en el de agrupamiento, el momento en que debe detenerse el proceso de construcción del árbol), la estructura del árbol -número máximo de hijos por nodo -, y aspectos de implementación y representación.

La sección 5 incluye algunas aplicaciones recientes de estas estructuras de datos, como la utilización de estructuras de tipo octree para modelos de volumen, la representación de imágenes mediante árboles de partición binaria del espacio (BSP), el uso de mapas de oclusión jerárquicos - tipo quadtree - para el preprocesado de visibilidad de escenas muy complejas, y finalmente

una introducción a los wavelets, a las representaciones multiresolución y a la representación wavelet de datos en dominios esféricos.

2 Estructuras de datos.

2.1 Bintrees.

Los Bintrees **(SaW-87)** son estructuras de datos jerarquicas que se caracterízan por:

Espacio de trabajo:	Imagen
Tipos de datos primarios:	Objetos
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del espacio
Tipo de arbol:	Binario, no balanceado ni restringido

Para construir el arbol se subdivide recursivamente el espacio, que se supone finito. En cada nivel del arbol se realiza una bisección y se obtienen dos paralelepípedos que se asocian a los hijos del nodo(figura 1).

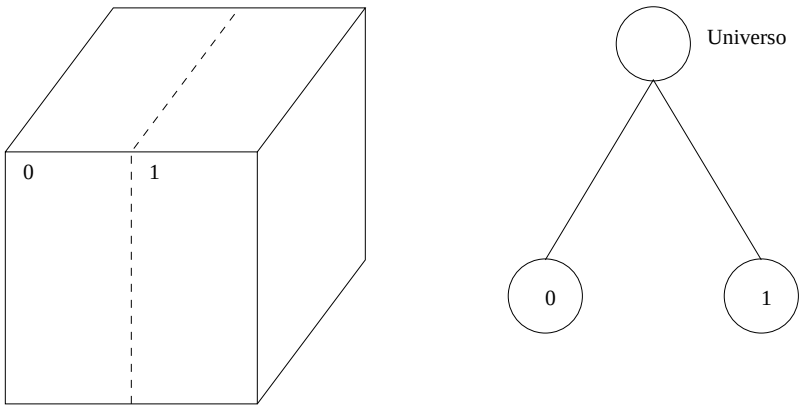


Figura 1: Cada nodo se subdivide en dos paralelepípedos.

En cada nivel del arbol se usa, iterativamente, un plano de bisección distinto. Los planos de bisección son: XY, YZ y ZX. Los nodos obtenidos se denominan:

- **G**rises, los que pueden ser subdivididos y se subdividen.
- **B**lancos, los completamente exteriores.
- **N**egros, los completamente interiores.

- **G**rises **T**erminales, aquellos nodos grises que pudiendo ser subdivididos ya no se subdividen.

El criterio de hoja[1] usado es:

	1 objeto	grupo de objetos
Criterio de hoja	exterior ó interior ó tam. $\leq$ mínimo	$m \leq tb$

Donde m es el numero de puntos en la región y tb es el tamaño del *bucket*[2].

Un Bintree puede ser construido a partir de una representación CSG con un coste $O(n)$ **(SaW-87)** y, en general, el coste medio de insertar un nuevo nodo es $O(log_2 n)$, siendo n el número de nodos del Bintree. Una vez construido, el coste de buscar una hoja es $O(log_2 n)$. Dentro de esa hoja puede haber más de un elemento, en ese caso el coste de localizar el elemento buscado es $O(p)$ siendo p el número de elementos.

Si denominamos:

- G: Número de nodos grises del arbol.
- T: Número de hojas del arbol.

En todo Bintree se cumple que: $G = T - 1$.

2.2 Octrees.

Los Octrees **(SaW-87)** son estructuras de datos jerarquicas que se caracterízan por:

[1] Que nos indica cuando un nodo del arbol se puede considerar hoja y por tanto detener la subdivisión

[2] *Bucket*: Lista asociada a cada nodo del arbol, nos indica el número máximo de objetos que caben en un nodo.

Espacio de trabajo:	Imagen
Tipos de datos primarios:	Objetos
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del espacio
Tipo de arbol:	Octal, no balanceado ni restringido

Para construir el arbol se realiza una subdivisión recursiva del espacio, que se supone finito. En cada nivel del arbol se subdivide la escena, mediante los planos XY, YZ y ZX, en ocho cubos identicos (figura 2). Cada uno de estos cubos es numerado y, siguiendo esta numeracion, posteriormente es asignado a cada uno de los hijos del nodo. La numeración de cada uno de los octantes nos da el orden entre los hijos.

La nomenclatura de los nodos y el criterio de hoja son los mismos que los usados en los Bintrees.

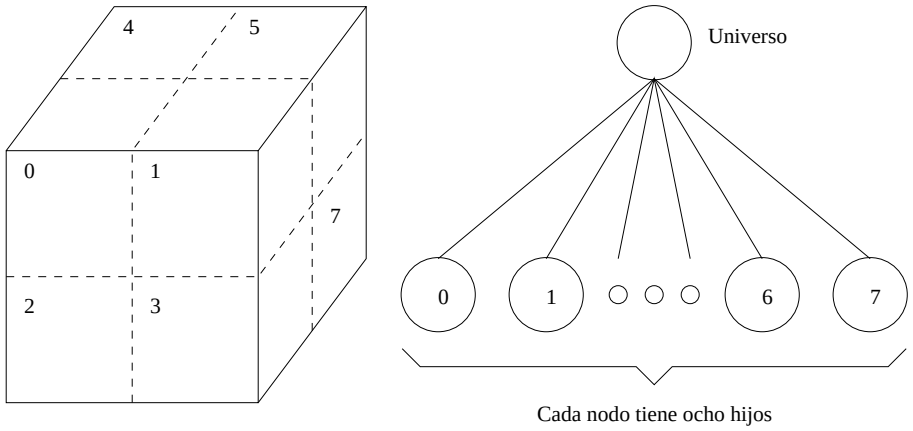


Figura 2: Cada nodo se subdivide en ocho cubos.

El coste medio de insertar un nuevo nodo en un Octree es $O(log_8 n)$, siendo n su número de nodos. Una vez construido, el coste de buscar una hoja es $O(log_8 n)$. Dentro de esa hoja puede haber más de un elemento, en ese caso el coste de localizar el elemento buscado es $O(p)$ siendo p el número de elementos.

Si denominamos:

- G: Número de nodos grises del arbol.
- T: Número de hojas del arbol.

En todo Octree se cumple que: $G = (T - 1)/7$.

2.3 Point-Quadtrees.

Los Point-Quadtrees **(FiB-74)** son estructuras de datos jerárquicas que se caracteizan por:

Espacio de trabajo:	Objeto
Tipos de datos primarios:	Puntos
Otros tipos de datos:	Ninguno
Subdivisión:	Del espacio
Tipo de árbol:	Cuatro hijos, no balanceado ni restringido

Un Point-Quadtree para un conjunto de puntos $P = (p_1, p_2, ..., p_N)$ en 2 diemensiones es un árbol quaternario que cumple:

- Por cada elemento de P contiene: su localización en 2 dimensiones, sus propiedades y tiene asociados 4 subárboles que se corresponden a las 4 direcciones del espacio (NO,NE,SO,SE).
- Por cada nodo P del árbol se cumple que:
  - Si p_1 es un elemento del subárbol NO $p_1.x < p.x$ y $p_1.y \geq p.y$
  - Si p_1 es un elemento del subárbol NE $p_1.x \geq p.x$ y $p_1.y \geq p.y$
  - Si p_1 es un elemento del subárbol SE $p_1.x \geq p.x$ y $p_1.y < p.y$
  - Si p_1 es un elemento del subárbol SO $p_1.x < p.x$ y $p_1.y < p.y$

La definición en dos dimensiones se puede fácilmente generalizar a dimensión k.

Para la construcción del árbol se subdivide el espacio de forma recursiva en las cuatro zonas. Las zonas de subdivisión se determinan usando líneas paralelas a los ejes de coordenadas que passan por uno de los puntos del dominio.

Hay diferentes métodos de elección del punto por el cual se subdivide, por ejemplo el punto que esté más centrado.

Si N es el número de elementos del árbol, se puede decir que:

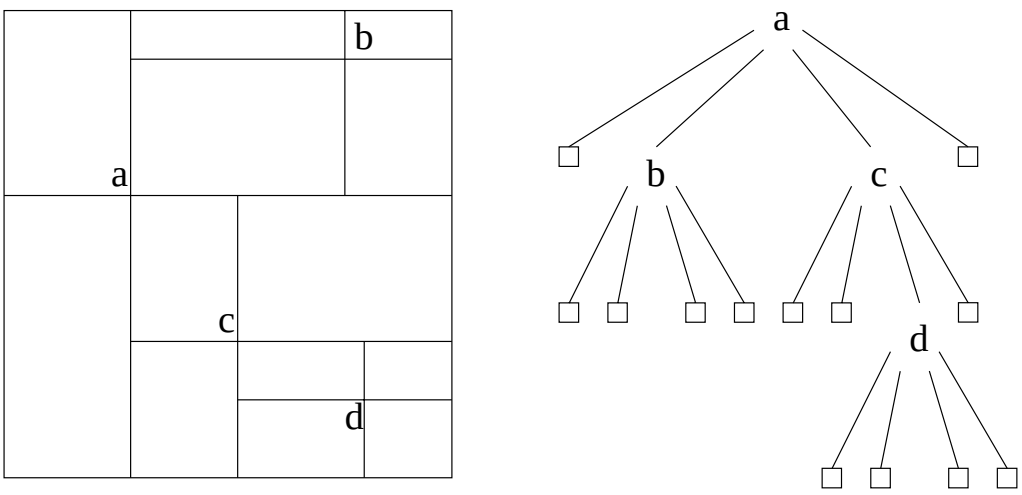


Figura 3: Representación de un Point-Quadtree en 2 dimensiones.

- El coste de la construcción en el peor de los casos es de:

$$\frac{N*(N-1)}{2}$$

- La profundidad h esta acotada:

$$\log_4 \frac{3N+1}{4} \leq h \leq N-1$$

El criterio de hoja usado es:

- *Objeto:*

  Diremos que un nodo del árbol es hoja cuando tiene una lista vacía de elementos en cada uno de sus hijos.

2.4 K-d trees.

Los K-d trees **(Ben-75)** son estructuras de datos jerárquicas que se caracterizan por:

Un k-d trees para un conjunto de elementos $P = (p_1, p_2, ..., p_N)$ es un árbol binario que cumple:

- Cada nodo de P tiene una llave multi-dimensional (Ej: Coordenadas del punto) y un discriminante que es con el que se subdividen los elementos (Ej: Coordenada X).

Espacio de trabajo:	Objeto
Tipos de datos primarios:	Puntos
Otros tipos de datos:	Ninguno
Subdivisión:	Del espacio
Tipo de árbol:	árbol Binario, no balanceado ni restringido

- Por cada nodo p_i del árbol con llave y discriminante se cumple:
  - Cualquier elemento en el subárbol izquierdo de p_i tiene un valor menor para el discriminante del nodo.
  - Cualquier elemento en el subárbol derecho de p_i tiene un valor mayor o igual para el discriminante del nodo.
- La raiz tiene profundidad 0 y los nodos con la misma profundidad tienen el mismo discriminante.

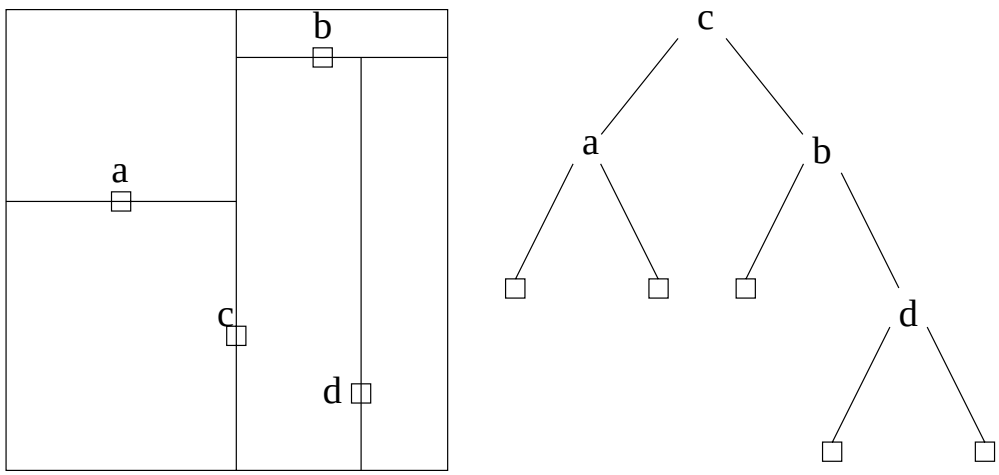


Figura 4: Representación de un k-d-tree en 2 dimensiones.

Para la construcción del árbol, en el caso de que los elementos son puntos bidimensionales, se subdivide el espacio de forma recursiva en dos zonas. Las zonas de subdivisión se determinan usando como discriminantes los valores de las coordenadas en los puntos nodo. Se alterna el sentido de división entre X,Y.

Si N es el numero de elementos del árbol se puede decir que la profundidad h está acotada:

$$\log_2 \frac{N+1}{2} \leq h \leq N-1$$

El criterio de hoja usado es:

- *Objeto:*

  Diremos que un nodo del árbol es hoja cuando tiene una lista vacía de elementos.

2.5 Vector octrees.

Octree cuyas hojas pueden contener información geométrica (SaW-87, BrN-92).

Espacio de trabajo:	Imagen
Tipo de datos primarios:	Objetos poliédricos
Otros tipos de datos:	Grupos de objetos o objetos de caras curvas aproximables por caras planas
Subdivisión:	Del espacio
Tipo de árbol:	Octal, no balanceado, ni restringido

Se construyen *top-down* de la manera habitual. La única diferencia respecto a los octrees clásicos es el criterio de hoja. Un nodo puede contener, por ejemplo, un trozo de cara (parte de la frontera) del objeto sin tener que subdividirse hasta llegar a nodos llenos y vacíos. En cambio, este nodo tendrá que almacenar información geométrica de esa cara (su ecuación). De esta manera se ahorran muchos nodos pero las hojas son más complejas.

Una nodo puede ser Blanco (vacío), Negro (lleno) o Gris (contiene frontera). Pero además podrá contener otra información, que dará nombre al árbol. Cada uno de estos tipos de árboles incluye al anterior.

- **Face octree** (Pla-93) Una cara plana es contenida en una hoja (nodo C). Los nodos serán B, N, G y C.
- **Face & Edge octree** Una arista es contenida en una hoja (nodo A). En esta arista pueden converger más de dos caras. Los nodos serán B, N, G, C y A.

- **Extended octree** (ABJN-85, CCV-85) Un vértice es contenido en una hoja (nodo V). En este vértice pueden converger más de tres caras. Los nodos serán B, N, G, C, A y V.

Si un nodo no cumple ninguno de los anteriores criterios de hoja, se subdivide hasta que todos sus descendientes sí los cumplan. En caso de llegar a una profundidad máxima preestablecida sin cumplir ningún criterio, el nodo se asigna a Gris.

Implementación de los nodos: Cada tipo de nodo tendrá una implementación diferente:

- **Blanco, Negro:** Hojas que sólo almacenan su color.
- **Gris:** Solamente nodos interiores. Tienen punteros a sus ocho hijos.
- **Cara:** Puntero a la cara.
- **Arista:** Punteros a las caras (normalmente dos) y la configuración de la arista: en qué semiespacio se encuentra el objeto.
- **Vértice:** Punteros a las caras y la configuración del vértice.

Interfaz:

- **Inserción:** Para insertar un nuevo objeto se tiene que bajarlo (e ir subdividiéndolo) por el árbol. Al llegar a una hoja seguramente no se cumplirá el criterio de hoja, se asignará Gris al nodo y se subdividirá.
- **Supresión:** Al suprimir un objeto, todas sus hojas se convierten en Blancas. Los hermanos se comprimen *bottom-up* a nodos Blancos mientras se pueda.
- **Transformaciones geométricas:** Depende de la casuística. Pequeñas transformaciones pueden no modificar al árbol, sólo a las ecuaciones de sus caras. En transformaciones mayores, hará falta recalcular todo o parte del árbol.
- **Point location:** Empezar la búsqueda por la raíz bajando por la rama que contenga el punto, hasta llegar a una hoja. Si es Blanca o Negra, la respuesta es inmediata. En otro caso (C,A,V) hará falta sustituir el punto en las ecuaciones de las caras.
- **Range searching:**

- **Ventana:** Empezando por la raíz, bajar por todas las ramas tales que intersecten con el prisma o pirámide de visión. Si la hoja es Blanca o Negra, la respuesta es inmediata. En otro caso, intersectar el prisma o la pirámide con las caras de la hoja.
- **N cercanos:** Explorar los nodos más cercanos (hermanos, primos...) de menor a mayor distancia radial.
- **Búsqueda direccional:** Dada la subdivisión espacial del octree, se halla fácilmente el nodo que se encuentra a un determinado lado de otro nodo. Si aquél tiene hijos, se baja por la rama correspondiente.
- **Interferencias:** Las intersecciones entre dos objetos se representan por hojas A y V. Se comprueba si las caras de tales hojas pertenecen o no al mismo objeto.

Complejidad de los algoritmos:

(BrN-92) Las mismas que en octrees clásicos. Hay que tener en cuenta que habrán muchos menos nodos en el árbol y también que el tiempo de proceso en un nodo será mayor (procesar caras) pero sólo en un factor constante.

Usos/aplicaciones:

El Vector octree es un híbrido entre el octree clásico y la B-Rep. La representación de los objetos poliédricos es exacta pero consume más memoria en los objetos con superfícies complejas.

2.6 BSP.

Los Binary Space Partitioning Trees **(Fuc-80 Tn-87 Nat-90)** conocidos a través de la abreviación BSP, son estructuras de datos jerarquicas que se caracterízan por:

Espacio de trabajo:	Objeto
Tipos de datos primarios:	Objetos (poliedros)
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del espacio
Tipo de arbol:	Binario, no balanceado ni restringido

Para construir el arbol se subdivide recursivamente el espacio. En cada nivel del arbol se realiza una bisección mediante un plano arbitrario que no necesariamente está alineado con los ejes. Normalmente se suele utilizar

algún tipo de heurística durante el proceso de creación del BSP para determinar el emplazamiento concreto de los planos de sección, con el objetivo de optimizar de alguna forma el arbol resultante.

En cada nodo interno del arbol el plano de bisección origina dos subespacios. En caso de que un subespacio cumpla una determinada condición (que sea "homogeneo" o contenga un determinado número de objetos) será etiquetado como nodo hoja del arbol. En caso contrario se procede recursivamente con la subdivisión hasta alcanzar una clasificación total del universo inicial.

Originariamente los BSP estaban orientados a representación de poliedros. En dicho caso se suele hacer coincidir los planos de bisección con los planos de las caras del objeto que se desea representar, aunque a veces se prefiere introducir planos arbitrarios para obtener un BSP mas óptimo. (figura 5).

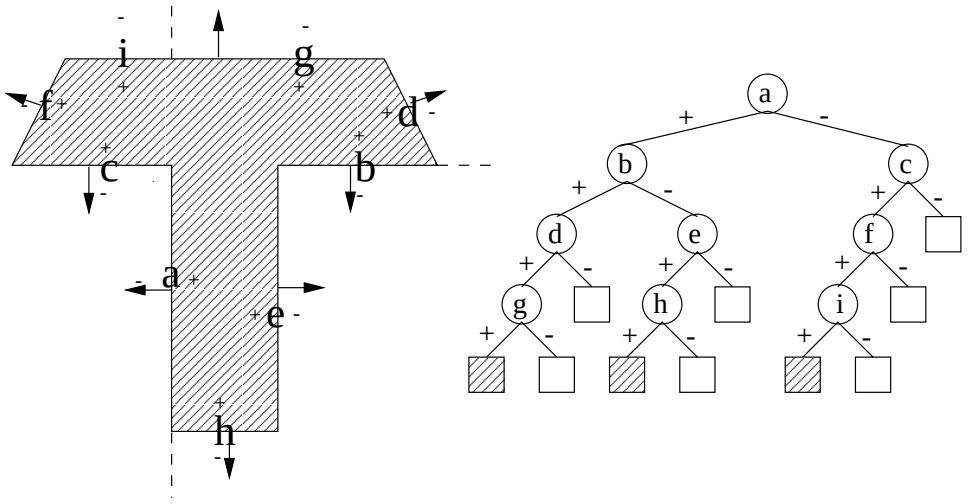


Figura 5: BSP de un poliedro.

Por otro lado, los BSP que clasifican grupos de objetos suelen utilizar planos de sección arbitrarios, seleccionados convenientemente con el fin de evitar intersecciones con los elementos propios de la escena (6). Ello permite conseguir un BSP más optimo que almacena todos los elementos de la escena únicamente en los nodos hoja del arbol. En caso de que un plano de sección intersecte algún elemento será necesario introducir algún mecanismo que permita manejar esta ambigüedad o bien duplicando los elementos intersecados en cada subespacio resultante o bien asociandolos al priopio nodo interior.

El criterio de hoja[3] usado es:

[3]Que nos indica cuando un nodo del arbol se puede considerar hoja y por tanto detener

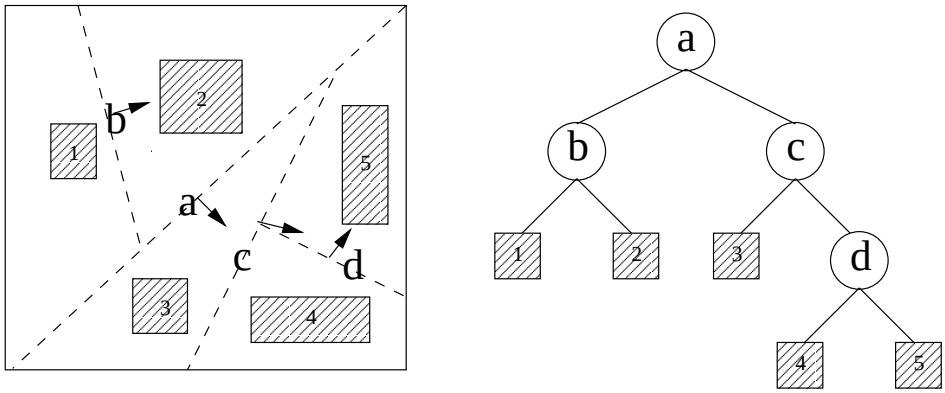


Figura 6: BSP de un grupo de objetos.

	1 objeto	grupo de objetos
Criterio de hoja	exterior ó interior	$m \leq tb$

Donde m es el numero de puntos en la región y tb es el tamaño del *bucket*[4].

Un BSP puede ser construido a partir de una representación B-rep con un coste $O(n)$, siendo n el número de nodos del BSP. De todos modos la optimización de un arbol BSP suele ser muy costosa debido a la componente heurística y de backtracking que conlleva normalmente. En general, el coste medio de insertar un nuevo nodo en un BSP es $O(log_2 n)$. Una vez construido, el coste de buscar una hoja es $O(log_2 n)$. Dentro de esa hoja puede haber más de un elemento, en ese caso el coste de localizar el elemento buscado es $O(p)$ siendo p el número de elementos.

La transformación geométrica se pueden aplicar directamente a los arboles BSP transformando convenientemente los planos de sección contenidos en cada nodo interior. También existen algoritmos especiales que permiten realizar operaciones booleanas entre BSPs de manera muy eficiente.

la subdivisión

[4] *Bucket*: Lista asociada a cada nodo del arbol, nos indica el número máximo de objetos que caben en un nodo.

2.7 R-trees.

Los R-trees **(Gut-84 Duc-98)** son estructuras de datos jerarquicas que se caracterízan por:

Espacio de trabajo:	Objeto
Tipos de datos primarios:	Objetos
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del objeto
Tipo de arbol:	Binario, no balanceado ni restringido

El objeto se considera formado por un numero indeterminado de elementos simples (normalmente polígonos). Para construir el correspondiente R-tree se realiza una subdivisión recursiva del objeto (o de la escena) en subconjuntos de elementos mas simples, aplicando un determinado criterio generalmente basado en heuristica geométrica.

El arbol se contruye top-down, es decir, los elementos geométricos que componen el objeto se asocian únicamente a los nodos hoja del arbol. En los nodos interiores tan solo se mantienen referencias a los nodos hijos y una mínima información geométrica referente a la rama: la caja englobante (figura 7). En los R-trees las dos cajas englobantes resultantes de una subdivisión son normalmente no disjuntas.

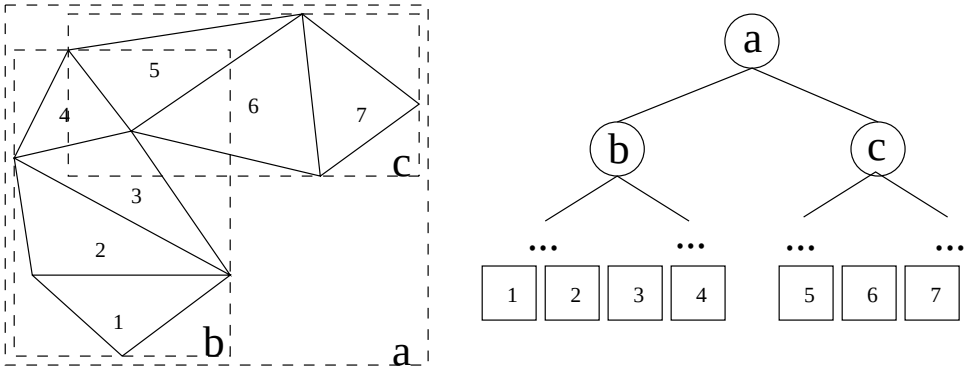


Figura 7: R-tree de un objeto.

Existen ligeras variantes de los R-tree originales conocidos como R^+-trees y R^*-trees. La aportación principal de los R^+-trees consiste precisamente en

utilizar cajas englobantes completamente disjutas. Ello se consigue intersecando las cajas englobantes obtenidas en cada subdivisión con el objeto original, a costa de introducir nuevos elementos geométricos resultantes de la sección de los poligonos frontera.

El criterio de hoja [5] usado en todos los casos es:

	1 objeto	grupo de objetos
Criterio de hoja	elemento sencillo	$m \leq tb$

Donde m es el numero de puntos en la región y tb es el tamaño del *bucket*[6]. La contrucción completa de un R-tree suele ser simple ya que en ningun momento se opera con la geometría original, sino que sólo se clasifica (esto no sucede en los R^+-trees). El coste medio de insertar un nuevo nodo en un R-tree es $O(log_2 n)$, siendo n su número de nodos. Una vez construido, el coste de buscar una hoja es $O(log_2 n)$. Dentro de esa hoja puede haber más de un elemento, en ese caso el coste de localizar el elemento buscado es $O(p)$ siendo p el número de elementos.

2.8 Box trees.

Arbol cuyos nodos son cajas que contienen los objetos (BCG-96).

Espacio de trabajo:	Objeto
Tipo de datos primarios:	Caras triangulares de una malla
Otros tipos de datos:	Objetos en general o grupos de ellos
Subdivisión:	Del espacio
Tipo de árbol:	Binario(arbitrario), no balanceado, ni restringido

[5] Que nos indica cuando un nodo del arbol se puede considerar hoja y por tanto detener la subdivisión

[6] *Bucket*: Lista asociada a cada nodo del arbol, nos indica el número máximo de objetos que caben en un nodo.

Los Box trees se construyen *bottom-up*. Los objetos están representados por sus respectivas cajas contenedoras y son las hojas del árbol. Agrupando pares de cajas vecinas en una caja mayor que contenga íntegramente ambas, se crea un nodo interno en el nivel inmediatamente superior. De esta manera, recursivamente, se construye la jerarquía.

Normalmente, el criterio utilitzado para seleccionar qué dos cajas vecinas se unirán en una de nueva, es el de mínimo volumen desperdiciado en la nueva caja contenedora.

En general, las cajas pueden intersectarse entre ellas.

Según sea la caja contenedora (AABB[7], OBB[8], Pie slices[9] u otras) habrá diferentes tipos de árboles. Incluso un mismo árbol puede contener cajas de diferentes formas.

Implementación de los nodos:

Datos de la caja contenedora y punteros a los hijos. Las hojas contienen además, información del objeto. Los nodos internos pueden tener una lista de todos los objetos de sus descendientes.

Interfaz:

- **Inserción:** Empezando por la raíz, bajamos por el hijo tal que, si insertamos el objeto en aquella rama, provocará el menor incremento de volumen de las cajas.
- **Supresión:** Subir desde la hoja que contiene el objeto hasta la raíz, actualizando los nodos. Puede suceder que los criterios utilizados en la agrupación de cajas ahora no se mantengan.
- **Transformaciones geométricas:** La variación de la posición, tamaño u orientación realizada en la caja se tiene que propagar desde la hoja hasta la raíz. Si ésto implica que se violan los criterios de agrupación, se tiene que realizar una supresión y una inserción del objeto modificado.
- **Point location:** Empezar la búsqueda por la raíz bajando por la rama que contenga el punto, hasta que lleguemos a una hoja o a un

[7] *AABB: Axis-aligned Bounding Box* o caja (prisma rectangular) contenedora alineada con los ejes coordenados.

[8] *OBB: Object Bounding Box* o caja contenedora de orientación arbitraria.

[9] *Pie slice*: Cilindro con un sector circular por sección, de orientación arbitraria. Literalmente, *trozo de tarta*.

nodo interior tal que el punto no esté contenido en ningún hijo.

- **Range searching:**
  - **Ventana:** Empezando por la raíz, bajar por todas las ramas tales que la caja de su raíz intersecte con el prisma o pirámide de visión. Si intersecta parcialmente, puede ser que algún descendiente lo haga. Si intersecta totalmente (la caja está contenida en el prisma), todos los descendientes lo harán. Si no intersecta, ningún descendiente lo hará.
  - **N cercanos:** Dado que en el criterio de agrupamiento interviene la proximidad, tenemos que hallar los "parientes" de la hoja más cercanos a la hoja en cuestión: hermano, primos (nietos del mismo abuelo), biznietos de bisabuelo... Se parte de la hoja y se exploran estos nodos cercanos.
  - **Búsqueda direccional:** Se comprueban los nodos hermanos de la hoja, del padre, del abuelo... Cuando alguno de ellos esté en la dirección especificada, bajamos por sus descendientes y comprobamos cuáles están en la dirección correcta. Si llegamos hasta la raíz, no hay ningún elemento en la dirección dada.
  - **Interferencias:** Si no hay intersección entre dos cajas, entre los objetos tampoco. En caso contrario hay que intersectar los dos objetos.

Complejidad de los algoritmos:

Casi todos ellos implican subir o bajar por una rama del árbol. Por lo tanto serán $O(\log n)$. Los que bajan simultáneamente por varias ramas y los que investigan vecinos, son $O(n)$.

Usos/aplicaciones:

Las cajas permiten cálculos rápidos de intersecciones, por lo que la estructuración de una escena con un árbol de boxtrees permite realizar rápidamente cálculos de colisiones entre objetos y aplicar algoritmos de ray-tracing.

2.9 Sphere trees.

A partir de esferas que contienen objetos o partes de un objeto, se agrupan jerárquicamente para formar un árbol (Hub-96).

El árbol se construye en dos pasos:

1. Creación de las esferas que, entre todas ellas, contienen el objeto. Se construye el diagrama de Voronoi de unos puntos distribuidos regularmente por el objeto. En cada vértice del diagrama que sea interior

Espacio de trabajo:	Objeto
Tipo de datos primarios:	Un objeto
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del espacio
Tipo de árbol:	Octal(arbitrario), no balanceado, ni restringido

al objeto se centra una esfera de radio tal que contenga los puntos asociados al vértice. La unión de estas esferas aproxima al objeto.

Este paso es trivial si los datos son un conjunto de objetos: para cada objeto simplemente se halla su esfera contenedora.

2. Agrupación de las esferas en otras esferas mayores para formar el árbol. Es una construcción *top-down*. Comenzamos con una esfera inicial que contiene todo el objeto (a los puntos que lo definen). Con las esferas del paso 1 (son las hojas del árbol) obtenidas de los puntos que definen a la esfera padre, vamos uniendo esferas vecinas. La esfera resultante puede unirse a su vez con cualquier otra esfera vecina, sea obtenida en el paso 1 o bien resultado de una unión. Probamos todas las combinaciones, quedándonos con las uniones que tengan una distancia de Hausdorff menor entre la esfera y el objeto. Paramos cuando quedan sólo ocho esferas. Estas son las ocho hijas de la esfera padre. Para cada esfera hija realizamos el mismo proceso.

   La unión de dos esferas vecinas se realiza creando una nueva esfera que contenga todos los puntos que definen a las dos esferas.

No hay ninguna relación entre la posición de una esfera en el árbol, su tamaño y su posición en el espacio.

En general, las esferas se intersectan entre ellas.

Implementación de los nodos: Centro y radio de la esfera y puteros a los hijos. Las hojas contienen, además, información del objeto.

Interfaz:

- **Inserción:** Bajar el objeto por la jerarquía, según su posición, empezando por la raíz. Si una hoja lo contiene totalmente, se queda aquí. Sinó, aumentar el radio de la esfera para que el objeto pueda caber. Si la esfera fuese demasiado grande, poner el objeto en un nodo (y esfera) nuevo y recalcular la jerarquía.

- **Supresión:** Sacar el objeto de la hoja corresponente y suprimirla. Se podría subir por la rama, disminuyendo el radio de los nodos ascendentes. En caso de nodos internos con pocos hijos, podría hacer falta recalcular la jerarquía.
- **Transformaciones geométricas:** Readaptar la esfera (hoja) al nuevo tamaño y/o posición del objeto y propagar las modificaciones hacia arriba.
- **Point location:** Empezar la búsqueda por la raíz bajando por la rama cuya esfera contenga el punto hasta que lleguemos a una hoja, donde comprobaremos si el punto está en el interior del objeto.
- **Range searching:**
  - **Ventana:** Empezando por la raíz, bajar por todas las ramas tales que la esfera de su raíz intersecte con el prisma o pirámide de visión. Si una esfera no intersecta, ninguno de sus descendientes lo hará. Si está incluida en el prisma, todos sus descendientes también. En cualquier otro caso, bajar recursivamente.
  - **N cercanos:** No hay ninguna relación espacial entre hermanos, por lo tanto hay que recorrer el árbol partiendo desde la hoja buscando entre sus "parientes" más cercanos.
  - **Búsqueda direccional:** Se comprueban los nodos hermanos de la hoja, del padre, del abuelo... Cuando alguno de ellos esté en la dirección especificada, bajamos por sus descendientes y comprobamos cuáles están en la dirección correcta. Si llegamos hasta la raíz, no hay ningún elemento en la dirección dada.
  - **Interferencias:** Conociendo el centro y radio de las esferas contenedoras de dos objetos, si aquéllas no intersectan, sus objetos contenidos tampoco. Si intersectan, se tiene que realizar el test de colisión entre ambos objetos.

Complejidad de los algoritmos:

Casi todos ellos implican subir o bajar por una rama del árbol. Por lo tanto serán $O(\log n)$. Los que bajan simultáneamente por varias ramas y los que investigan vecinos, son $O(n)$.

Usos/aplicaciones:

Detección de colisiones entre objetos.

2.10 BV-trees.

Es una jerarquía de cajas contenedoras las cuales son una aproximación de *convex hulls* por un número n de planos predeterminados (HKM-96, KHM-96).

Espacio de trabajo:	Objeto
Tipo de datos primarios:	Objeto triangulado
Otros tipos de datos:	Grupos de objetos
Subdivisión:	Del espacio
Tipo de árbol:	Binario(arbitrario), no balanceado, ni restringido

Se aproxima el objeto por medio de n planos. Estos planos tienen una orientación predeterminada. El único grado de libertad es su posición en el espacio, que utilizaremos para situarlos tangentes al objeto. La normal apunta en la dirección opuesta al objeto. La intersección de todos los semiespacios interiores es un volumen cerrado que engloba el objeto. Este volumen se llama n-gono (concreción de polígono) o también k-dop (k-discrete orientation polytope).

La raíz del árbol es el n-gono del objeto o escena inicial. Se asocia cada una de las partes del objeto (triángulos) a un punto. Se escoge el eje coordenado en el que la proyección de los puntos tenga una mayor variancia. Se halla la posición media de los puntos en el eje escogido y según ella dividimos las partes en dos grupos. Creamos los dos n-gonos respectivos y los situamos como hijos de la raíz. Recursivamente formamos el árbol.

Los n-gonos más utilizados son:

- 6-gonos. Las normales de los planos son los ejes coordenados. Dos planos según los dos sentidos de cada eje. Los 6-gonos son cajas AABB.
- 14-gonos. 6-gonos más los planos que truncan los ocho vértices.
- 18-gonos. 6-gonos más los planos que truncan las doce aristas.
- 26-gonos. La combinación de todos los anteriores.

Se pueden utilizar otro número de planos y/o orientaciones diferentes.

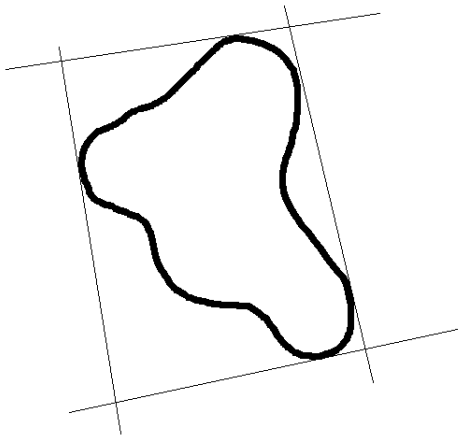


Figura 8: Ejemplo de 4-gono en el plano 2D.

Implementación de los nodos:

Los punteros a los hijos y los datos de los n planos. Dado que la orientación de cada plano es conocida, sólo hace falta el término independiente de su ecuación (un real). Las hojas tienen información del triángulo contenido. Un nodo interno puede tener la lista de los triángulos que contiene.

Interfaz:

- **Inserción:** Empiezar por la raíz bajando por el hijo que corresponda (siguiendo las normas de creación del árbol) hasta llegar a una hoja. Aquí se realiza una división poniendo aquella hoja y el nuevo elemento como hermanos. Finalmente, actualizar hacia arriba el tamaño de los n-gonos.
- **Supresión:** Suprimir la hoja correspondiente y subir un nivel a la hoja hermana. Actualizar hacia arriba el tamaño de los n-gonos.
- **Transformaciones geométricas:** Traslaciones y escalados implican recalcular el n-gono y actualizar hacia arriba. En las rotaciones, si un nodo guarda los vértices del n-gono contenido, una rápida aproximación es rotar estos vértices y construir encima de ellos un nuevo n-gono.
- **Point location:** Empezar la búsqueda por la raíz bajando por el hijo que contenga el punto, hasta que lleguemos a una hoja o el punto no esté contenido en ningún hijo. Si llegamos a una hoja, hacer la intersección del punto con el triángulo contenido. Si en un momento dado el punto está contenido en dos hermanos, bajar por ambos; uno de los dos fallará.

- **Range searching:**
  - **Ventana:** Empezando por la raíz, bajar por todas las ramas tales que el n-gono de su raíz intersecte con el prisma o pirámide de visión. Si intersecta parcialmente, puede ser que algún descendiente lo haga. Si intersecta totalmente (el n-gono está contenido en el prisma), todos los descendientes lo harán. Si no intersecta, ningún descendiente lo hará.
  - **N cercanos:** Dado que en el criterio de agrupamiento interviene la proximidad, tenemos que hallar los "parientes" de la hoja más cercanos a la hoja en cuestión: hermano, primos (nietos del mismo abuelo), biznietos de bisabuelo... Se parte de la hoja y se exploran estos nodos cercanos.
  - **Búsqueda direccional:** Se comprueban los nodos hermanos de la hoja, del padre, del abuelo... Cuando alguno de ellos esté en la dirección especificada, bajamos por sus descendientes y comprobamos cuáles están en la dirección correcta. Si llegamos hasta la raíz, no hay ningún elemento en la dirección dada.
  - **Interferencias:** Si no hay intersección entre dos n-gonos, entre los objetos tampoco. En caso contrario hay que intersectar los dos triángulos.

Complejidad de los algoritmos:

Casi todos ellos implican subir o bajar por una rama del árbol. Por lo tanto serán $O(\log n)$. Los que bajan simultáneamente por varias ramas y los que investigan vecinos, son $O(n)$.

Usos/aplicaciones:

Los BV trees se usan para la detección de colisiones de objetos que se mueven en tiempo real. también se usan en aplicaciones de ray-tracing por la facilidad con que se calculan las intersecciones de los rayos con los planos.

2.11 Jerarquía de mallas/multitriangul.

Las mallas triangulares son muy utilizadas en la representación tridimensional de objetos debido a la simplicidad de sus elementos constitutivos y a la facilidad con la cual se pueden procesar y visualizar. En muchas aplicaciones, para ahorrar en términos de tiempo de calculo y de espacio de memoria utilizados, por ejemplo, en el procesado de detalles que no son relevantes para una aplicación, es útil disponer de la posibilidad de adaptar el nivel de

resolución de la malla. Han sido propuestos muchos modelos con estas propiedades y, a continuación, se presentaran las caracteristicas más relevantes de algunos de allos.

Las triangulaciones jerárquicas **(Flo-95)** se representan mediante un árbol cuyos nodos son triangulaciones de un dominio triangular. Cada triangulación refina la parte de superficie asociada al nodo padre. La estructura jerárquica de estos modelos tiene como inconvenientes que no son utilizables para cada conjunto de datos y en cada aplicación y que la información necesaria para obtener el mismo nivel de detalles es, en general, más alta respecto a mallas sin esta estructura.

En **(Cig-95)** ha sido propuesta una estructura de datos para mallas triangulares en que la representación está formada por una triangulación de Delaunay de base y una serie de sus modificaciones. Con ésta representación los triángulos generados en cada modificación y las adyacencias entre ellos se guardan en la estructura que por consiguiente resulta poco practica de utilizar. Además la extracción a resolución variable es posible solo en el caso en que el error de aproximación requerido sea creciente al crecer la distancia desde el punto de vista dado.

El modelo de las mallas progresivas descrito en **(Hop-96)** esta constituido por una secuencia de mallas a con nivel de resolución creciente. Cada una de ellas se obtiene de la anterior mediante modificaciones locales que involucran los triángulos incidentes en el mismo vértice. Este vértice se divide en dos y forma una nueva arista de la triangulación. No se representa explicitamente las secuencias de mallas y una de ellas, correspondiente a una cierto nivel de resolución constante sobre toda la malla, se obtiene a partir de la de base y de la secuencia de modificaciones que son las únicas informaciones que se guardan. Un nivel de resolución variable se puede obtener no efectuando el refinamiento sobre algunas regiones de algunas mallas de la secuencia cosa que pero perjudica la posibilidad de refinar dichas regiones posteriormente.

La técnica descrita en **(Kle-96)** es similar al método de las mallas progresivas. En este caso la secuencia está formada por triangulaciones de Delaunay que se reconstruyen a partir de una malla de base y de una secuencia de modificaciones que corresponden a la inserción de un nuevo vértice en la malla de Delaunay. En el mismo trabajo se describe un algoritmo para obtener un nivél de resolución variable sobre una misma malla.

Se puede afirmar que entre los modelos que soportan resolución variable, los que son más eficientes resultan costosos en términos de espacio de memoria requerido y viceversa.

El inconveniente que presentan los dos últimos trabajos descritos en la generación de mallas con resolución variable es debido a que ellos utilizan una la ordenación total de las modificaciones de la malla de base que corresponde a

una secuencia de mallas con ordenación total, mientras que la relación entre las modificaciones locales que se deben efectuar para obtener una malla de resolución variable es de ordenación solo parcial.

Una ordenación parcial de las modificaciones ha sido utilizada en la técnica llamada en **(Flo-96)** *Multiresolution Triangulation* (MT). En esta técnica, se utilizan componentes constituidas por porciones de mallas triangulares y una ordenación parcial que permite combinar las componentes de distintas maneras para obtener mallas con diferentes niveles de resolución. En este trabajo ha sido demostrado que todas las mallas que se pueden formar con los triángulos que aparecen en la MT pueden ser construidas utilizando la ordenación parcial. Mediante este tipo de información es posible también obtener en tiempo lineal la representación en multiresolución con un numero de triángulos mínimo para un nivel de resolución dado. Además, el modelo MT incluye como casos particulares todos los modelos en multiresolución presentes en la literatura En **(Flo-97)** han sido desarrolladas y analizadas estructuras de datos compactas y algoritmos eficientes para MT. Las MT se describen mediante grafos dirigidos aciclicos en que los nodos son las mallas triangulares básicas.

3 Representación/implementación.

Hay tres implementaciones típicas de octrees:

Pointer octree. Un nodo interno tiene ocho punteros a sus hijos, que se disponen según un orden preestablecido.

Las hojas no tienen ningún puntero, sólo la información correspondiente del objeto.

Hace falta un bit que indique si un nodo es una hoja o bien un nodo interno. Este bit se guarda, no en el nodo en cuestión, sinó en su padre.

No se almacena información que permita volver al padre. Se supone que si fuese necesario, cuando se baja hacia un nodo a partir de la raíz, se pueden guardar en una pila la dirección de los nodos por los que se pasa.

DF-expression. Depth first-expression (KaE-80, KEY-80, KEM-83), también *treecode* (OlW-83a, OlW-83b). Consiste en la lista de los nodos (representados por su color: N, B, G) dispuestos según el recorrido en preorden del árbol. Para recorrer la lista y reconstruir la jerarquía hay que tener en cuenta:

- Los nodos N y B son hojas.
- Los nodos G son padres de ocho hijos, excepto en el caso que sean terminales. Por esta razón hace falta un bit que indique si estos nodos son hojas o no.

Donde los nodos blancos representan a B, los nodos negros a N y los círculos a G. En este caso la DF-expression equivalente es:

GGBNBBBBBBNBGBBBBBNBBBGBNBBBBBBBB

Una variante consiste en substituir la G por un paréntesis abierto, que se cierra al terminar los ocho hijos. El ejemplo anterior queda:

((BNBBBBBB)NB(BBBBBNBB)B(BNBBBBBB)BB)

Esta implementación es muy útil en aplicaciones que atraviesen todos los nodos del octree (unión, intersección, cálculo de áreas...). Pero puede ser un inconveniente si se desea visitar una rama o unos nodos en concreto sin pasar por los anteriores.

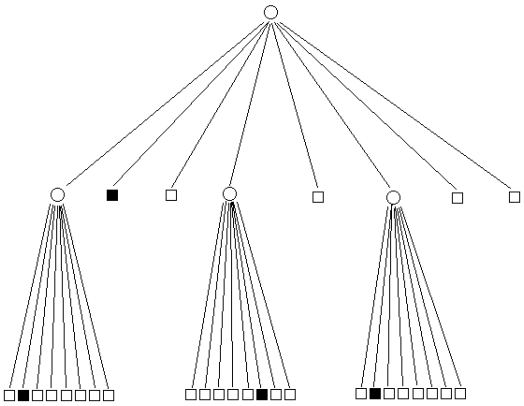


Figura 9: Ejemplo de octree.

Linear octree. Usa los *locational codes*, también llamado Dewey-decimal encoding (Knu-75) y también *leafcode* (OlW-83a, OlW-83b). Si un cubo se divide en octantes, éstos se pueden numerar de la siguiente manera:

0 - Delante-Arriba-Izquierda
1 - Delante-Arriba-Derecha
2 - Delante-Abajo-Izquierda
3 - Delante-Abajo-Derecha
4 - Detrás-Arriba-Izquierda
5 - Detrás-Arriba-Derecha
6 - Detrás-Abajo-Izquierda
7 - Detrás-Abajo-Derecha

Cualquier otra numeración también sirve al propósito.

Siguiendo esta numeración un nodo se representa por un número que indica el nivel del nodo y una secuencia de dígitos en base ocho que representan sus antecesores en el árbol. Cada dígito indica qué hijo es respecto al nodo padre; es decir, qué octante es respecto el cubo padre cuando se divide.

No hace falta guardar todos los nodos, sólo las hojas N (llenas) y no las B (vacías). En el caso de que la región espacial tome diferentes valores y no tenga sentido hablar de espacio vacío, no habrá nodos B. Pero aún así tampoco se tendrán que guardar los nodos internos.

El ejemplo anterior se representa de la siguiente manera:

{ 2-01, 1-1, 2-35, 2-51 }

Se han estudiado algoritmos aplicados a esta implementación concreta (Gar-82a, Gar-82b, Gar-82c, Gar-83).

Se ha realizado un estudio sobre qué implementación (pointer vs. linear) consume menos memoria (SaW-87). La conclusión es que si se cumple la desigualdad

$$2^{(7/8)\cdot(3n+log_2(n))-(1+log_2(8/7))} < L$$

donde n es la profundidad máxima del árbol y L es la cantidad de hojas
entonces la implementación lineal ocupa menos memoria que la de punteros. Se deduce que cuantos menos niveles tenga el octree y mayor sea la dimensión del objeto, los punteros consumirán menos memoria.

También se ha demostrado con el Complexity Theorem (Hun-78, HuS-79) que la cantidad de nodos en un octree es lineal respecto al tamaño de la frontera.

4 Algoritmos básicos.

4.1 Octrees restringidos, balanceados o l-irregulares.

Se denominan octrees balanceados a aquellos que no tienen elementos adyacentes que se diferencien *mucho* de tamaño. Esta propiedad de continuidad entre los elementos de un octree ha sido redescubierta en varias ocasiones en diferentes trabajos y resulta muy util a la hora de obtener la malla de triángulos que compone la superficie de ese octree. Tambien se denominan l-irregulares o restringidos. Este *mucho* debe ser cuantificado, en función de la diferencia entre elementos adyacentes del octree podemos hablar de 0-balanceados, 1-balanceados, etc.

En un octree se define recursivamente el *nivel* de una hoja como uno más que el de su padre, así hasta llegar a la raiz que tiene *nivel* 0. Dos elementos de un mismo nivel del octree son k-vecinos (figura 10) si tienen una cara[10] k-dimensional en comun.

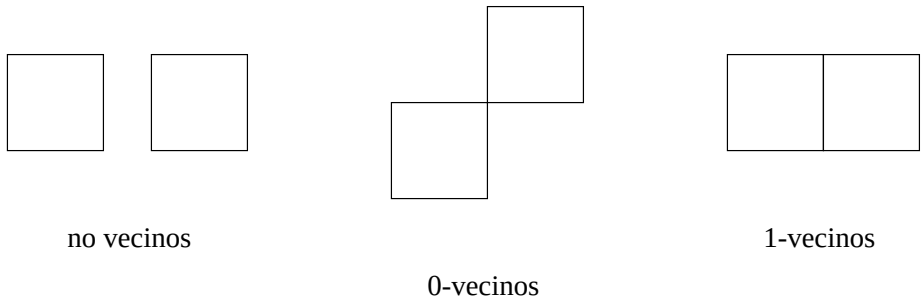


Figura 10: Ejemplo de no vecindad, 0-vecindad y 1-vecindad en quadtrees.

Un octree es k-balanceado (figura 11) si en un cierto nivel n ningún elemento (no dividido) de este nivel posee k-vecinos que se subdividan más alla del nivel $n+1$. De esta forma:

- Un octree se considera 0-balanceado si, dado un nivel n, ningun elemento no dividido de ese nivel posee un 0-vecino que se subdivida más allá de $n+1$.
- Un octree se considera 1-balanceado si, dado un nivel n, ningun elemento no dividido de ese nivel posee un 1-vecino que se subdivida más allá de $n+1$.

[10] Una cara 0-dimensional es un punto, una cara 1-dimensional es una arista, etc.

- Un octree se considera 2-balanceado si, dado un nivel n, ningun elemento no dividido de ese nivel posee un 2-vecino que se subdivida más allá de $n + 1$.

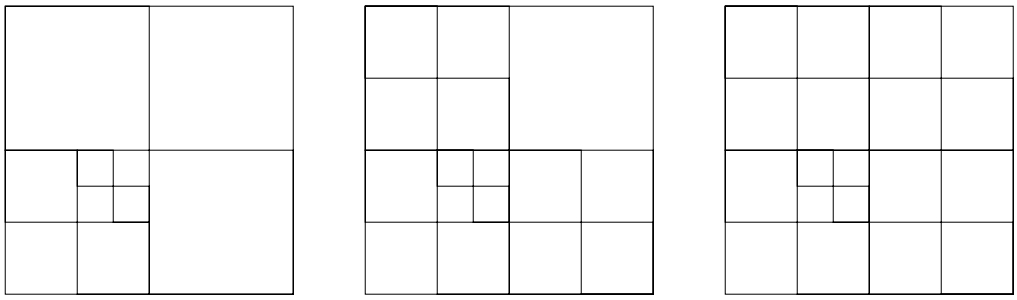


Figura 11: Ejemplos de 0-balanceamientos y 1-balanceamientos en quadtrees.

En la figura 11 se puede ver que la continuidad entre elementos del octree es mayor cuanto menor es el balanceo, de forma que 0-balanceado es más restrictivo que 1-balanceado.

Desgraciadamente, balancear un octree hace que este crezca en tamaño, aunque solo en un orden constante. Se saben ciertas las siguientes afirmaciones sobre quadtrees:

- Existe un único k-balanceo de un quadtree del que se puede derivar cualquier otro otro de sus k-balanceos. Esto permite hablar de balancear un quadtree sin especificar ningun algoritmo.
- Weiser demostró **(BSW-83)** que dado un quadtree $T1$ con n nodos, se puede obtener un 0-balanceo $T2$ con no más de Cn nodos, donde C es el número máximo de 0-vecinos que posea un elemento de $T1$, incluyendose a si mismo (figura 12).
- Moore demostró **(Moo-95)** que dado un quadtre cuadrangular $T1$ con n nodos, se puede obtener un 1-balanceo $T2$ con no más de $8n$ nodos. Ademas, tambien demostró que existen infinitos quadtrees cuadrangulares que al ser 1-balanceados crecen $8n - c$, donde c es una constante lineal. Lo que garantiza que $8n$ es un límite de crecimiento bastante ajustado.
- Usando quadtrees triangulares Moore demostró **(Moo-95)** que dado un quadtre triangular $T1$ con n nodos, se puede obtener un 1-balanceo

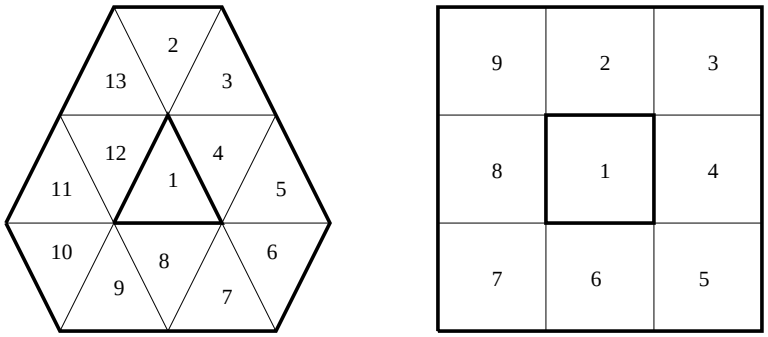


Figura 12: $C \leq 9$ en los quadtrees cuadrangulares, $C \leq 13$ en los quadtrees triangulares, etc.

> $T2$ con no más de $10n$ nodos. Ademas, tambien demostró que existen infinitos quadtrees triangulares que al ser 1-balanceados crecen $10n-c$, donde c es una constante lineal. Lo que garantiza que $10n$ es un límite de crecimiento bastante ajustado.

Se pueden aplicar técnicas similares a las usadas, para acotar el crecimiento de quadtrees de mayor grado y dimensión.

4.2 Transformaciones geométricas.

Los algoritmos geométricos existentes pueden clasificarse, básicamente, en dos tipos diferentes: aquellos que realizan algoritmos de transformación inversa, requiriendo la existencia de dos octrees en memoria para poder remuestrear el octree de origen, y aquellos que realizan computaciones de direcciones recorriendo el árbol y calculando las posiciones de destino de cada nodo negro y moviéndolo en consecuencia. En general los del primer tipo pueden resolver cualquier tipo de transformación, mientras que los del segundo, aunque reconocidos como los más rápidos, son mejor adaptados para el cálculo de traslaciones. Un algoritmo de fuerza bruta **(Gar-83)** para calcular transformaciones geométricas realiza la descompocición de cada nodo negro en nodos de tamaño unidad, calcula la destinación final de cada uno y, finalmente, comprime nuevamente el árbol.

4.2.1 Algoritmos de Translación.

Una optimización con respecto a esto parte de la observación que, si la longitud a trasladar es un múltiplo del tamaño del nodo, no es necesario

fragmentar el nodo. Si esta condición no se mantiene, **(AN84)** descompone el nodo hasta que se cumpla, logrando conservar el costo asintótico.

En **(SNV97)** un acercamiento diferente es utilizado, partiendo de la observación que no es necesario realizar la descomposición de los nodos para conocer su localización final en una translación: utilizando una notación recursiva especial para referirse a la localización de los nodos y una operación aritmética de suma de distancias adecuadamente definida en esta base de localizaciones, es posible calcular la nueva ubicación de un extremo de un nodo de cualquier tamaño: Cada octante es numerado del 0 al 7, siguiendo el esquema de la figura 13, izquierda, y cuando se realiza una división recursiva cada nuevo sub-octante será numerado de idéntica manera, formando su localización un número de n dígitos, $d_1, d_2, ..., d_n$ dónde n es la profundidad del nodo en el árbol (en la figura 13, derecha, puede verse un ejemplo para quadtrees). La dirección de destino se calcula mediante la adición a este número del desplazamiento expresado en binario, utilizando unas simples reglas de cálculo cuidadosamente explicadas en **(SNV97)**.

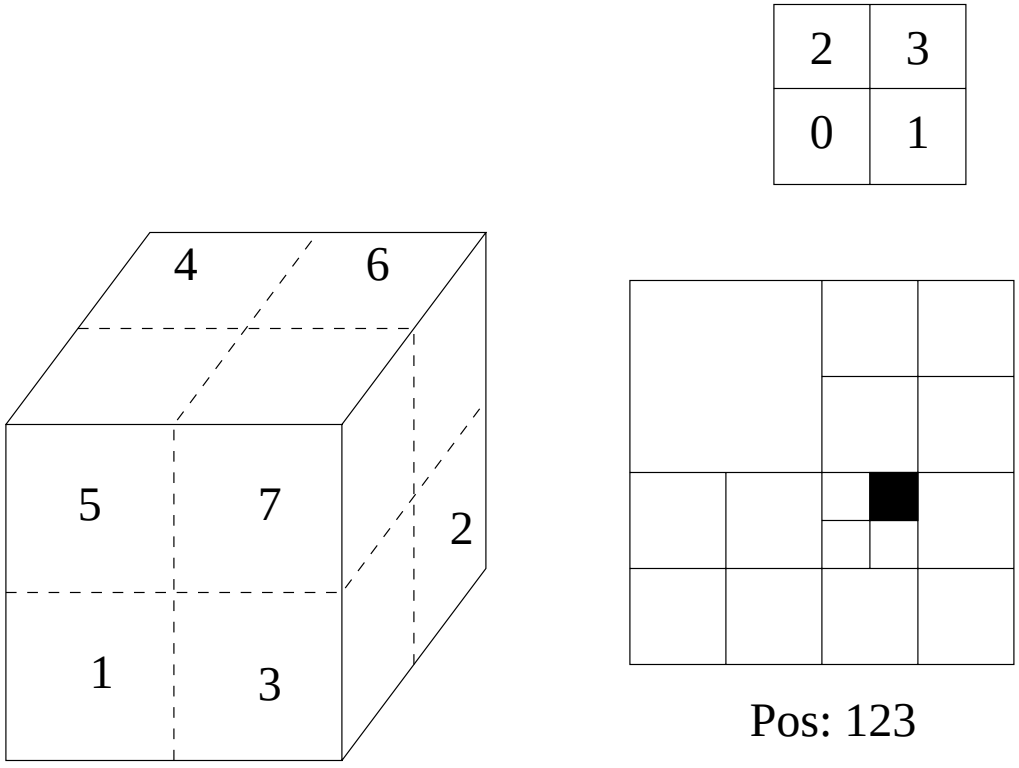


Figura 13: Numeración de octantes y ejemplo de un quadtree recursivo.

Para trasladar un nodo cúbico completo basta con trasladar dos cubitos de tamaño unidad localizados en extremos opuestos del nodo, y realizar luego

una operación elemental de llenado (*fill*) del espacio comprendido entre ellos en sus posiciones finales. Armados de estas dos ideas básicas es posible realizar translaciones arbitrarias de una capa de nodos cualquiera o incluso de un árbol completo. Finalmente para pequeños desplazamientos se estudian varios casos posibles, hallándose optimizaciones para cada uno que acelera enormemente el cálculo de translaciones, basándose en la idea de buscar realizar la menor cantidad de alteraciones posibles al octree original cuando se lo traslada. El coste de cada optimización es calculado y comparado con el método no-optimizado para determinar si la optimización es relevante y los resultados justifican su utilización. Se observa que este algoritmo es mejor que los anteriores para el caso de realizar pequeños desplazamientos.

4.2.2 Algoritmos de cizalla y Rotacion.

En el área de procesamiento de imágenes, la descomposición de rotaciones 2D en tres cizallas a lo largo de los ejes coordenados es ago bien conocido. Utilizando esto, para poder realizar una rotación arbitraria de un octree, los autores se basan en la simple pero poderosísima observación de que una rotación puede ser escrita como la combinación de tres cizallas en los tres ejes coordenados. De esta forma, descomponiendo las rotaciones en productos de tres rotaciones alrededor de los ejes coordenados, y concatenando las resultantes $3 * 3 = 9$ cizallas podemos realizar cualquier rotación en el espacio (esto también se lograría con sólo tres **cizallas**, pero se utilizará la descomposición más larga dado que las cizallas en el caso corto difieren de la matriz identidad en dos términos, mientras que las presentadas en **(P90)**, debido a so origen bidimensional, difieren de la identidad en sólo un término. Estas son más simples de usar debido a que sólo una coordenada es afectada en cada paso.

Sin pérdida de la generalidad, considérese una rotación en θ a lo largo del eje X, expresado en notación matricial por

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix}$$

que puede descomponerse como el producto de tres matrices de cizalla de la forma

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -tan\frac{\theta}{2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & sin\theta & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -tan\frac{\theta}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

Una cizalla es calculada en un octree de forma casi trivial: se recorre el árbol desde adelante hacia atrás y, para cada nodo negro encontrado, se calcula su verctor de traslación y se lo mueve usando el algoritmo mencionado arriba.

Para disminuir el número de operaciones en punto flotante, es posible calcular también traslaciones de nodos grises, de forma de poder reusar cálculos del padre para sus hijos. Es también posible reducir las operaciones de punto flotante a un mínimo. Dado que una cizalla divide al octree en capas de nodos que comparten un mismo vector de translación, podemos reducir el número de operaciones recorriendo el árbol mediante un orden de capa por capa. Aún más, las operaciones de punto flotante pueden eliminarse completamente si se usa un algoritmo *á la* Bresenham para calcular los vectores de traslación de una capa a la siguiente usando solamente adiciones de enteros.

4.3 Búsquedas de vecinos.

En muchas aplicaciones, en una representación mediante octrees, es necesario conocer el voxel adyacente en una dirección dada a un voxel dado.

En **(Gar-82b)** cada voxel está representado por un número entero en base 8 en un sistema ponderado en que el peso 8^{n-1} está asociado con el octante más grande que lo contiene, el peso 8^{n-2} al segundo octante más grande, etc. Un voxel Q está representado por la expresión $Q = q_{n-1}8^{n-1} + q_{n-2}8^{n-2} + \cdots + q_0 8^0$ con $q_l \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ para $l = 0, 1, \ldots, n-1$. La codificación en éste código octal de un voxel con subindices $I, J, K = 0, 1, \cdots, 2^n - 1$ se efectúa a través de la codificación binaria de J, I y K, o sea, mediante los coeficientes contenidos en las expresiones siguientes

$$\begin{aligned} J &= c_{n-1}2^{n-1} + c_{n-2}2^{n-2} + \cdots + c_0 2^0 \\ I &= d_{n-1}2^{n-1} + d_{n-2}2^{n-2} + \cdots + d_0 2^0 \\ K &= e_{n-1}2^{n-1} + e_{n-2}2^{n-2} + \cdots + e_0 2^0 \end{aligned}$$

Obtenidos los coeficientes de la representación binaria, la representación octal se obtiene mediante la formula siguiente $q_l = e_l 2^2 + d_l 2^1 + c_l 2^0,\ l = n-1, n-2, \ldots, 0$. Si se utiliza esta representación el código octal del vecino de Q en dirección E se obtiene mediante el algoritmo siguiente

algorithm OCT-W$\rightarrow E$ $(n; q_0, q_1, \ldots, q_{n-1})$
  begin
    $j := 1$
    $i := 1$
    if q_0 is even **then**
      $E(q_0) := q_0 + 1$
    else
      $E(q_0) :=\mid q_0 + 1 \mid_8$
    while $i \neq n$ **and** $j \neq (n-1)$ **do**
      if q_{i-1} is even **then**

```
        E(q_j) := q_j,  j = i, i+1, ..., n-1
      else
        if q_i is even then
          E(q_i) :=| q_i + 1 |_8
        else
          E(q_i) :=| q_i + 7 |_8
          i := 1 + 1
    end
  end
```

Si q_0 es par el nodo adyacente a Q en dirección E pertenece al mismo octante de Q y por tanto será $E(q_0) = q_0 + 1$ y $E(q_j) = q_j,\ j = 1, 2, \ldots, n-1$. Si q_0 es impar Q y el nodo adyacente pertenecen a dos octantes que difieren por lo menos en una subdivisión, y una transición de un octante se obtiene sumando 7 modulo 8 a q_0, o sea, $E(q_0) =| q_0 + 7 |_8$. Para $q_1, q_2, \ldots$ se sigue el mismo razonamiento hasta encontrar un q_i impar. Desde entonces el octante no cambiará más. Para encontrar el nodo vecino en dirección W se puede utilizar el mismo algoritmo intercambiando los términos "par"y "impar"y la operación de "suma"con la de "resta"mientras que para adaptar el algoritmo a la búsqueda de vecinos en otras direcciones se deben realizar modificaciones más importantes a éste algoritmo.

En **(Sam-89)** han sido presentados otros algoritmos para la búsqueda de vecinos adyacentes a una cara, a una arista o a un vértice de un nodo dado P. A continuación se presentará solo el caso de búsqueda del vecino Q se dimensión mayor o igual a la dimesión de P adyacente en dirección I a una cara de P. El valor inicial de L representa el nivel del árbol en que se encuentra P mientras que su valor final es el nivel en que se encuentra Q.

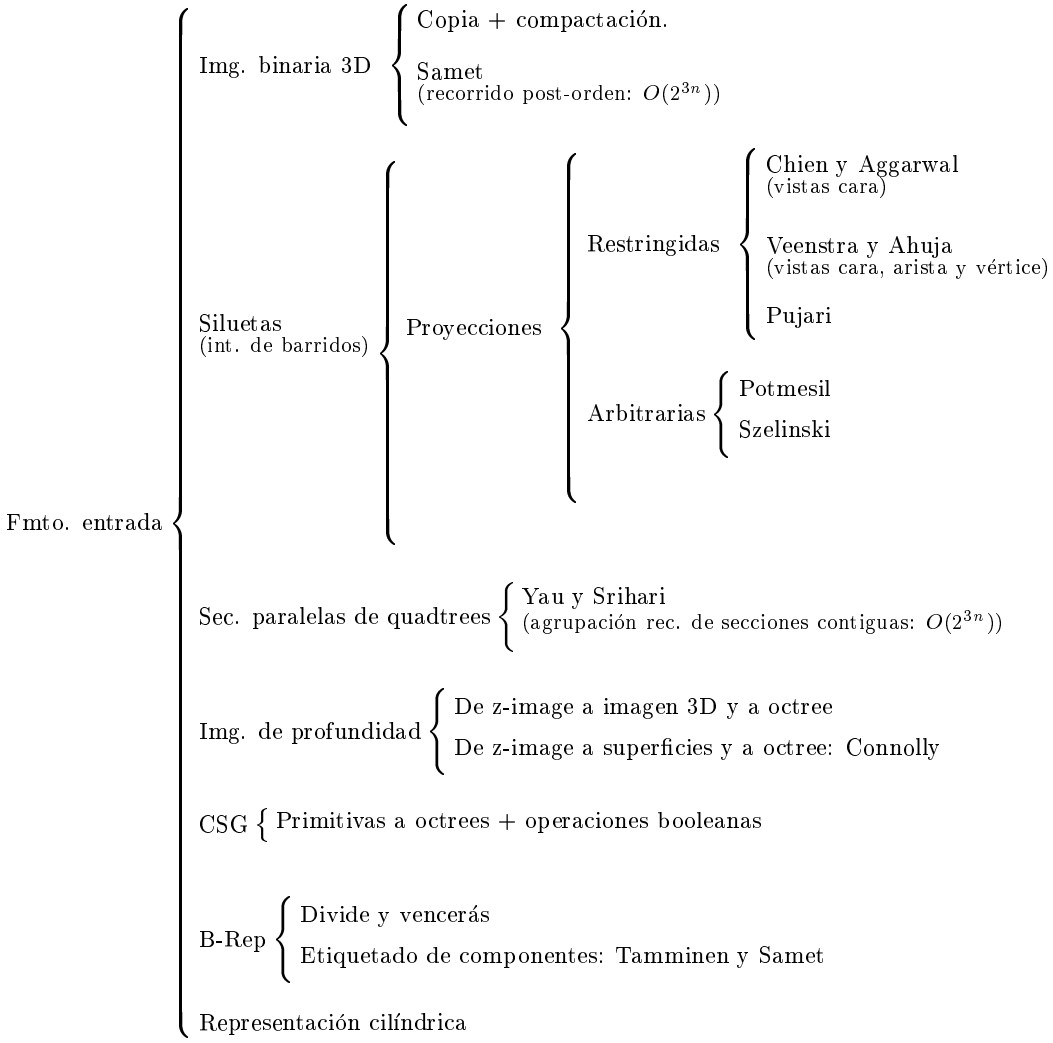
```
algorithm OT_GTEQ_FACE_NEIGHBOR2(P, I, Q, L)
    begin
    L := L + 1;
      if not(null(FATHER(P))) and ADJ(I,SONTYPE(P)) then
        /* si el nodo es adyacente a su padre */
        OT_GTEQ_FACE_NEIGHBOR2(FATHER(P),I, Q, L);
      else
        /* si el nodo no es adyacente a su padre */
        Q := FATHER(P)
      if not(null(Q)) and GRAY(Q) then
        /* si Q no es un nodo terminal */
        begin
          /* se sigue el camino reflejado */
          Q := SON(Q,REFLECT(I,SONTYPE(P)));
```

```
            L := L - 1;
        end
    end
```

Si P es un puntero a un nodo y O es un octante, FATHER(P) es el nodo padre y SON(P,O) es el nodo hijo de P correspondiente al octante O. La función SONTYPE(P) determina el octante en el cual se encuentra el nodo P y retorna el valor O si SON(FATHER(P),O)=P. En este algoritmo el predicado ADJ(I,O) es "verdadero"sii el octante O es adyacente a la cara I del bloque que contiene O. La función REFLECT(I,O), basada en una tabla, retorna el valor SONTYPE del bloque de igual tamaño que comparte la cara I.

4.4 Construcción y visualización.

Se Entiende por construcción (o entrada) de un octree, a la obtención, a partir de un sólido representado en otro formato, del octree que representa dicho sólido. Los algoritmos de construcción de octrees se pueden clasificar según el formato de los datos de entrada. La siguiente figura muestra una posible taxonomía basada en dicho criterio, extraida de **(ChH-88)**:



Veamos cada uno de ellos por separado:

1. **Entrada a partir de imágenes binarias 3D** Una imagen binaria 3D es una representación basada en una tabla tridimensional en la que cada entrada tiene un valor binario que representa si el voxel correspondiente es interior (negro u objeto) o exterior (blanco o vacío) al objeto. Existen dos aproximaciones para la construcción de octrees a partir de arrays 3D.

   (a) Una aproximación consiste en construir el octree completamente desplegado para después compactarlo. El color de los nodos hoja del octree se obtiene directamente consultando el color del

correspondiente voxel del array 3D. A continuación sigue un proceso de compactación del octree, en el cual los nodos grises con hijos del mismo color son convertidos a nodos terminales del color correspondiente.

(b) Otra posibilidad es una extensión del algoritmo propuesto por Samet para quadtrees. Se basa en visitar los elementos del array en un orden similar a un recorrido en post-orden de los nodos del octree. La ventaja es que los nodos hojas sólo se crean cuando se conoce que son nodos maximales. De esta forma, no es necesaria la fusión de nodos hijos del mismo color, ni el cambio del color del padre de gris a blanco o negro. La complejidad es lineal con el número de voxels: $O(2^{3n})$

2. **Construcción a partir de siluetas** Una silueta es una imagen binaria 2D correspondiente al contorno e interior del objeto desde cierto punto de vista. El principio básico de construcción de octrees a partir de siluetas es muy simple: cuando la silueta de un objeto se extiende a 3D a lo largo de la correspondiente dirección de visualización, forma un cilindro (proyección axonométrica) o un cono (proyección en perspectiva). Este volumen corresponde a un volumen englobante del objeto, por lo que una única silueta únicamente ofrece parte de información sobre el objeto. Para obtener mejores aproximaciones, es necesario intersectar otros volumenes englobantes procedentes de diferentes puntos de vista.

   El octree se puede construir a partir de un octree inicial correspondiente al primer volumen englobante y refinamientos sucesivos mediante intersecciones del octree con otros volumenes. El principal problema es la intersección octree-volumen englobante. Existen varias aproximaciones. Algunas trabajan con proyecciones axionométricas, tanto arbitrarias como restringidas a algunas direcciones de vista concretas. Otras trabajan con proyecciones en perspectiva. En general, conviene proyectar el octree en construcción sobre el plano de la imagen silueta, de forma que las intersecciones se realizan en 2D.

   El principal problema de la reconstrucción a partir de siluetas es que las partes del objeto cuya proyección no se refleja en las imágenes (agujeros, concavidades...) no pueden ser reconstruidas octree final.

   (a) Siluetas de proyecciones axionometricas restringidas Existe una familia de algoritmos de construcción de octrees a partir de siluetas que se basan en restringir las direcciones de proyección de imágenes axionométricas a unas direcciones concretas, de forma que se puede establecer una correspondencia entre octantes y zonas (cuadrantes) de la imagen.

El refinamiento del octree al incluir una nueva silueta es sencillo cuando se conoce la correspondencia entre octantes del octree y cuadrantes de la imagen. De esta forma no son necesarias intersecciones geométricas, sino únicamente intersecciones booleanas.

Por ejmplo en *Linear Octrees by Volume Intersection* Lavakusha, Pujari y Reddy proponen un algoritmo para la construcción de octrees a partir de 3 siluetas axionométricas ortogonales entre sí. Se basa en la intersección de 3 cilindros obtenidos al realizar un barrido paralelo de las siluetas en las respectivas direcciones. La principal limitación es que sólo es aplicable a objetos convexos sin agujeros.

El octree resultante se obtiene de forma directa, sin necesidad de compactación. La representación es la de un octree lineal: una lista de la clave de localización de los nodos negros terminales. El coste total del algoritmo es $O(nodosoctreefinal * Nimagen^2)$.

(b) Siluetas de proyecciones en perspectiva arbitrarias En estos algoritmos las direcciones de proyección de las imágenes axionométricas son libres, por ejemplo en *Generating Octree Models of 3D Objects from their Silhouettes in a Sequence of Images* Potmesil propone un algoritmo basado en la intersección de volúmenes cónicos generados a partir de las siluetas. El octree se genera mediante refinamientos sucesivos. Algunas características del algoritmo son:

- En un solo octree representa una escena que puede estar formada por varios objetos.
- Información sobre el color o la intensidad de las imágenes puede retenerse en los modelos 3D creados.
- Las proyecciones de las imágenes pueden ser axionométricas o en perspectiva, y el punto de vista es arbitrario.
- La intersección se realiza en espacio imagen (2D).

El algoritmo anterior tiene la limitación de que todas las imágenes deben mostrar todo el universo dentro de su campo de vista. Esto implica que el punto de vista de las imágenes debe estar a una distancia considerable del universo, lo cual puede resultar poco práctico en algunas aplicaciones de visión artificial, donde se desea reconstruir el área de trabajo de un robot. Para salvar esta limitación, el autor propone algunas variaciones.

Finalmente, una vez se ha obtenido un octree que representa fielmente el objeto, se procede a mejorar el octree. Para ello se realizan tres pasos:

Conectivity-labeling Consiste en asignar etiquetas distintas a objetos inconexos.

Normales a la superficie Se basa en incluir en los nodos hojas una normal a la superficie basada en el análisis de los 26-adyacentes.

Texture mapping Consiste en incorporar a los nodos hoja el color o la intensidad que tenían en la imagen.

En *Rapid Octree Construction from Image Sequences* Szelinski propone un algoritmo paralelo para obtener un octree a partir de siluetas, en el que el volumen es *esculpido* a base de extraer regiones exteriores al objeto. El test falla dando como grises algunos cubos que en realidad son negros o blancos, pero esto no afecta al algoritmo puesto que estos nodos grises son subdivididos. Sin embargo, es necesario un postproceso de compactación del octree.

Características:

- Construye el octree por niveles, desde la raíz hasta las hojas.
- En cada revolución se obtiene una versión más aproximada del objeto, añadiendo un nivel al octree. No se examina un nivel concreto del octree hasta que el nivel inferior no ha sido completamente generado.
- El cálculo de intersecciones se realiza en espacio imagen.

3. **Construcción a partir de secciones paralelas de quadtrees** La entrada consiste en una serie de secciones de un objeto, donde cada sección está representada por un quadtree. Un ejemplo es el algoritmo propuesto por Yau y Srihari. Consiste en mezclar pares de quadtrees correspondientes a secciones adyacentes. El proceso continua hasta que se han fusionado todas las secciones.

   Cuando se fusionan dos secciones, los quadtrees correspondientes son recorridos en paralelo para decidir el color de los nodos fusionados. La complejidad es lineal con el número de voxels: $O(2^{3n})$

4. **Construcción de octrees a partir de imágenes de profundidad** Una imagen de profundidad *depth image* es una imagen 2D donde cada pixel tiene asociada la profundidad de un punto del objeto (z-buffer). Las z-imagenes tienen la ventaja sobre las siluetas de permitir la reconstrucción de algunas concavidades.

   Existen distintos métodos para obtener el octree correspondiente al objeto 3D representado:

   (a) Un camino indirecto consiste en convertir los datos de profundidad en una imagen binaria 3D, y a partir de ahí obtener el octree utilizando las técnicas aplicables a arrays 3D.

(b) Otra posibilidad consiste en segmentar los datos de la imagen de profundidad en pedazos de superficies y obtener el octree a partir de los métodos aplicables a modelos de sólidos. Un algoritmo basado en esta última aproximación es el diseñado por Connolly.

5. **Construcción de octrees a partir modelos CSG** La representación CSG de un objeto es un árbol binario donde los nodos no-hoja contienen operadores booleanos (unión, intersección, etc.) y los nodos hojas representan primitivas. La conversión CSG a octree se basa en la conversión a octrees de las primitivas y en la fusión de estos octrees según los operadores de conjuntos. Puesto que los objetos primitivas son muy simples y admiten una expresión analítica, los cálculos de intersecciones requeridos en la conversión son extremadamente simples. Coste: $O(nprimitivasncubosoctree)$

6. **Construcción de octrees a partir de B-rep** Existen dos aproximaciones para la obtención de octrees a partir de la representación de fronteras de un poliedro:

   (a) Divide-y-vencerás Se empieza con un octree formado por un único nodo de color negro, el cual constituye la aproximación inicial del poliedro. El espacio se subdivide recursivamente en octantes para refinar el octree. El trabajo crítico es calcular la intersección de un octante con el poliedro, para averiguar el color del nodo correspondiente.

   (b) Etiquetado de componentes Se basa en utilizar la información de conectividad ofrecida por el Brep para rellenar el interior de una frontera cerrada. Consta de dos fases: en la primera se convierte la frontera del objeto en un árbol binario 3D. Los nodos hoja negros de este árbol representan las celdas que intersectan con la frontera del objeto. El resto de nodos, de color desconocido, se convierten en blancos o negros en la segunda fase, en la que se sigue la técnica de etiquetado según la conectividad. Finalmente, el árbol binario se transforma en un octree.

7. **Construcción de octrees a partir de una representación cilíndrica** Una representación cilíndrica de un objeto 3D se obtiene aplicando la subdivisión recursiva únicamente a 2 dimensiones del espacio. El resultado puede representarse mediante un quadtree donde los nodos contienen información sobre las coordenadas máxima y mínima de la dimensión no subdividida.

   Los nodos hojas almacenan las coordenadas máxima y mínima de la correspondiente columna. Estos nodos terminales pueden ser blancos o negros, de forma análoga a los octrees.

Los nodos no terminales almacenan las coordenadas máxima y mínima de sus columnas hijas. Estos nodos pueden ser grises o azules. Un nodo es azul si sus hijos están dentro del objeto pero tienen distintas coordenadas máxima y mínima. Un nodo es gris cuando algunos de sus hijos son interiores al objeto y otros no.

5 Discusión y tablas comparativas.

En la siguiente tabla se realiza una comparativa entre las doce estructuras de datos estudiadas anteriormente. La columnas contienen el tipo de árbol y las filas contienen los criterios de comparación.

Espacio de trabajo. En qué tipo de espacio se encuentra el árbol. Puede ser de espacio imagen o espacio objeto.

Subdivisión/agrupamiento. En unos tipos de árboles un nodo interno significa una subdivisión (partición). Esta puede ser del objeto o bien del espacio (con el objeto contenido en él). En los otros tipos un nodo significa un agrupamiento de elementos.

Criterio de hoja. En qué momento el contenido de un nodo se considera suficientemente "simple" como para no tener que realizar una subdivisión y convertir así el nodo en una hoja. En los árboles que agrupan nodos (y no los subdividen) el criterio es trivial: las hojas son los elementos considerados.

Número de hijos por nodo. En cuántos hijos se descompone un nodo interno.

Implementación/representación. Qué información tiene que contener un nodo. En algunos casos se diferencian los nodos internos de las hojas.

Comentarios. Características remarcables.

Se pueden realizar las siguientes observaciones:

Bintree, k-d tree y BSP són árboles binarios en los que únicamente varía la libertad que se tiene para hacer la subdivisión.

Sólo los árboles que se crean agrupando elementos son de construcción *bottom-up*.

	Octree	Bintree	Point quadtree	k-d tree
Espacio de trabajo	Imagen	Imagen	Imagen	Objeto
Subdivisión/ agrupamiento	Subdivisión espacio	Subdivisión espacio	Subdivisión espacio	Subdivisión espacio
Criterio de hoja	Interior/ exterior/ tam. mínimo/ núm. máx. de objetos	Interior/ exterior/ tam. mínimo/ núm. máx. de objetos	1 solo elemento	1 solo elemento
Número de hijos por nodo	8	2	4	2
Implementación/ representación	8 punteros Hoja: B/N	8 punteros Hoja: B/N/GT	4 punteros Hoja: objeto elemental	2 punteros Tipo de división. Hoja: objeto elemental
Comentarios	Top-down	Top-down Bisección por planos ortogonales La dirección de bisección depende de la profundidad del nodo	Top-down Los objetos definen el punto de paso de la división	Top-down Los objetos definen la coordenada de subdivisión División por planos ortogonales

	Vector octree	BSP	R-tree	Prism-tree
Espacio de trabajo	Imagen	Objeto	Objeto	Objeto
Subdivisión/ agrupamiento	Subdivisión espacio	Subdivisión espacio	Subdivisión objeto	Subdivisión objeto
Criterio de hoja	Interior/ exterior/ tam. mínimo/ frontera simple	0 planos en una región	1 solo elemento	1 solo elemento
Número de hijos por nodo	8	2	2(arbitrario)	2(arbitrario)
Implementación/ representación	8 punteros Hoja: B/N/ puntero a la primitiva	Plano 2 punteros Hoja: B/N	Caja 2 punteros Hoja: caja y obj.elemental	Caja 2 punteros Hoja: caja y obj.elemental
Comentarios	Top-down El tipo de frontera simple (C/A/V) determina el tipo del árbol	Top-down Subdivisión binaria del espacio basada en el objeto	Top-down Cajas intersectables Cajas orientadas según ejes Criterio de división según la aplicación	Top-down Cajas intersectables Cajas no orientadas según ejes Criterio de división según la aplicación Criterio de qué caja se ajusta mejor según la aplicación

	Box tree AABB	Box tree OBB	Sphere tree	BV-tree
Espacio de trabajo	Objeto	Objeto	Objeto	Objeto
Subdivisión/ agrupamiento	Agrupación objetos	Agrupación objetos	Agrupación objetos	Subdiv. y agrupación objetos
Criterio de hoja	1 solo elemento	1 solo elemento	Fragmento pequeño	1 solo elemento
Número de hijos por nodo	2(arbitrario)	2(arbitrario)	8(arbitrario)	2(arbitrario)
Implementación/ representación	Caja 2 punteros Hoja: caja y objeto elemental	Caja 2 punteros Hoja: caja y objeto elemental	Esfera 8 punteros Hoja: esfera y objeto elemental	Caja 2 punteros Hoja: caja y objeto elemental
Comentarios	Bottom-up Cajas intersectables Cajas orientadas según ejes Criterio de agrupación de vecinos según la aplicación	Bottom-up Cajas intersectables Cajas no orientadas según ejes Criterio de agrupación de vecinos según la aplicación	Bottom-up Esferas intersectables	Top-down Cajas con un número arbitrario de caras

6 Algunas aplicaciones.

6.1 Octrees y volumen.

En el artículo **"Colision Detection for Volumetric Objects " (HeK-97)** los autores proponen un modelo de probabilidad para poder tratar intersecciones entre objetos volumétricos complicados.

Introducción

La detección de colisiones es un problema fundamental en applicaciones con interacción gráfica en 3D, como animación y realidad virtual. La mayoría de trabajos se han centrado en la interacción entre objetos basados en superficies, y hay muy poca investigación en la detección de colisiones entre objetos volumétricos como la que se presenta en este artículo.

En los objetos volumétricos es difícil tener una superficie de forma explícita debido a su representación discreta. Además, el volumen contiene más información a cerca del objeto *(como la estructura interna)* y eso proporciona una descripción más detallada de la interacción entre los dos objetos.

La interacción entre dos objetos volumetricos no es una operación binaria, es un proceso no-determinista donde la probabilidad de colisión se proporciona en cada punto del espacio. Lo que se busca son las propiedades de la interacción entre los objetos en las regiones de intersección.

En el articulo se presenta un modelo de probabilidad para la descripción y una estructura jerárquica para la detección de las colisiones.

Modelo de probabilidad

Un **mapa de probabilidad** es una función que asigna la probabilitat $[0, 1]$ a cada punto del espacio según las propiedades del objeto. Este valor representa la probabilidad que exista una superficie en ese punto. El usuario define los mapas de probabilidad y eso proporciona suficiente flexibilidad para tratar con diferentes escenarios ya que genera diferentes modelos de probabilidad.

Un ejemplo de mapa general, es cuando las propiedades de un objeto volumétrico en un punto x pueden ser simulados como probabilidades de

partículas del medio participativo.

Otro es el generado a partir de objetos geométricos que se han escaneado para representarlos como un volumen y luego se les applica un filtro pasa bajos para eliminar el aliasing de la discretización. El objeto queda definido como una función de densidades y dependiendo del filtro aplicado estas densidades estiman la distancia entre el punto y la superficie real.

Detección jerarquica de las colisiones.

Una vez construido el modelo no-determinista de probabilidades se tiene que evaluar la probabilidad de colisión entre los objetos.
Se usa un sistema multivolumen. Cada objeto se define en su propio volumen con un sistema de coordenadas local que define su posición y orientación en el mundo.

Una opción para detectar colisiones es mostrear la intersección entre cada vóxel de los objetos. Este sistema es muy ineficiente a medida que la resolución augmenta. Para mejorar, se usa una detección jerárquica de colisiones que consiste en la construcción de un árbol de volumenes englobantes. Una vez construido el árbol, se empieza por la raíz un algoritmo recursivo que detecta la colisión a cada nivel. Eso evita hacer el testeo de la interacción entre regiones que no es posible que colisionen.

Estos árboles tienen que satisfacer:

- Los volumenes englobantes tienen que ser lo mas ajustados posible al objeto original para evitar superposiciones.
- La detección de superposiciones entre nodos tiene que ser lo más rápida posible.
- El árbol tiene que ser fácil y eficientemente construido.

Para tal efecto se estudia el **octree** (proporciona cajas muy ajustadas) y el **sphere-tree** (es fácil hacer la detección de superposiciones). El árbol se construye en 2 fases:

- **Fase bottom-up: Construcción del árbol**

Para el octree, las hojas son los vóxels y se aplica un simple algoritmo recursivo.

Para el sphere-tree, si es necesario, se subdivide el espacio del volumen para que las dimensiones del vóxel sean cúbicas. Cada hoja del árbol es la esfera que circumscribe a cada uno de los vóxels. Para subir de nivel se agrupan las ocho hojas vecinas. La raíz, es especial, y se considera el volumen como una caja rectangular.

A cada nodo del árbol se le asocia un rango de valores $[S_{min}, S_{max}]$ que son la mínima y la máxima probabilidad de colisión de todos los puntos dentro del espacio rectangular o cúbico inscrito.

- **Fase top-down: simplificación**

  Dados dos valores de threshold T_{max} sólido y vacío T_{min}. Se empieza por la raíz y se sigue según la busqueda de *breath-first*. Siendo S el nodo a tratar:

  - Si $S_{max} < T_{min}$ se eliminan el nodo y sus hijos.
  - Si $S_{min} > T_{max}$ se eliminan los hijos.
  - Si hay hijos directos S_1 de S que sólo tiene un hijo S_2 se elimina S_1 y S_2 pasa a ser el hijo de S.

El valor del indice de rangos se deriva del modelo de probabilidad. Si la función de probabilidad es P y el cubo inscrito es R^3 luego $S_{min} = minimo(P(x))$ y $S_{max} = maximo(P(x))$ por todo $x \in R^3$
Si P es simple se pude calcular analíticamente. Sino se usa un sistema de estimación conservativo.

Mapa de distancias

En la subdivisión jerárquica del espacio no se tiene en cuenta la coherencia de los datos. Para paliar dicho efecto, el mapa de distancias ayuda a testear cuando el objeto esta completamente fuera o dentro del otro objeto. En cada vóxel se guarda la distancia Euclidea al próximo vóxel no sólido, esta distancia es la distancia más corta entre qualquier punto dentro del vóxel y el punto más próximo dentro de un vóxel $T_{min} \geq p \geq T_{max}$.

Antes de descender un nodo del árbol y tratar sus hijos, se mira el mapa de distancias del nodo con el del segundo objeto; si el valor es mayor que el

radio del nodo, el nodo que se trata está en la parte homogenia y el recorrido puede detenerse.

Este método implica la inclusión de más memoria pero se puede usar el mapa de distancias con menos resolución que el original.

Evaluación de la probabilidad

Cómo se calcula de forma eficiente la probabilidad de colisión una vez detectada ?
Cuando se usa un mapa simple de probabilidad se puede aproximar de forma analítica. Un sistema es asignando directamente la función densidad y usar la inerpolación trilineal para reconstruir la función de densidad continua del resultado.

Si la fórmula de la probabilidad de colisión es polinomial, la evaluación de la máxima probabilidad se reduce a encontrar el valor máximo de la función polinomial en la zona de intersección (Usa métodos numéricos *Lagrange*).

Los autores proponen usar un método aproximado donde se calcula el valor de la probabilidad como el valor máximo del producto de probabilidades entre k puntos discretos mostreados en el espacio. La precisión del método depende del numero de muestras pero permite generar un metodo de refinamiento progresivo.

Conclusiones y implementación

Para decidir cual de las estructuras jerárquicas es mejor intentan encontrar un compromiso entre el número de operaciones necesarias para calcular la intersección y el ajuste de los volumenes englobantes. Se llega a la conclusión que el spheres-tree es en media 3 veces mas rapido que el octree para calcular la intersección. Pero en el mejor de los casos (cuando los vóxels son cubos) las esferas tienen el 0.37 porciento de ajuste. Debido a que a medida que las dimensiones del vóxel son menos regulares el ajuste del spheres-tree es peor, se sugiere que la mejor estructura para la detección de colisiones es el octree.

En el artículo muestra los resultados obtenidos aplicando el sistema a diferentes datos volumétricos. Para la implementación del árbol usan una lista de nodos en que cada nodo contiene 14 bytes.

Una aplicación es la colisión entre un CT de una cabeza de resolución 258x258x111 y un rayo de radiación 15x57x15.

Los tiempos conseguidos en una SGI Challenge 196MHz son:

Sphere-tree
33.7 segundos para la construcción del árbol.
24.6 segundos para el mapa de distancias.
412 ms detección de colisiones.

Octree
10.8 segundos para la construcción del árbol.
15.2 segundos para el mada de distancias.
252 ms detección de colisiones

6.2 Imagenes en BSP.

En **(Sub-97)** se describe una aplicación bastante interesante basada en estructuras BSP. El objetivo es facilitar la codificación, manipulación y visualización de imagenes (2D) y volúmenes (3D) en tiempo real. La aplicación principal que presenta el artículo es la de poder realizar *probing* para visualización médica, tomografía, rayos X, etc. Básicamente se trata de permitir escarbar en las estructuras visualizadas para revelar detalles internos que de otra forma no serían visibles. Según los autores este tipo de técnica ofrece mejores resultados que la visualización de la representación geometríca con secciones.

Según el articulo, se decide utilizar estructuras de tipo BSP porqué tienen una serie de ventajas frente a otros tipos de representación. Estas se pueden resumir básicamente en las siguientes propiedades:

- Transformaciones Geométricas

  Al igual que las representaciones B-rep los BSP permiten transformaciones geométricas exactas, eficientes y sencillas. Esto no sucede con otras estructuras jerárquicas como por ejemplo Quadtrees u Octrees y tampoco sucede con las estructuras de datos basadas en Voxels.

- Operaciones Booleanas

A diferencia de la representación B-rep los BSP permiten realizar operaciones booleanas de forma eficiente a través de algoritmos especialmente diseñados para ello. La eficiencia de dichas operaciones es tal alta como la obtenido de Quadtrees y Octrees. Esto permite realizar intersecciones y uniones de los BSP en tiempo real, resultando un instrumento ideal para detección de colisiones, tareas de *picking*, *clipping* o *probing*.

- Visibilidad y Ordenación

  A diferencia de la representación B-rep los BSP incorporan de forma implícita información relativa a la visibilidad de la estructura, normalmente ello permite realizar la visualización sin necesidad de hardware gráfico. Asímismo se permite implementar efectos de semitransparencía de forma correcta. Aunque existen varion algoritmos específicos este tipo de efectos son muy costosos de implementar a través de algoritmos de visualización genéricos, tal como el Z-Buffer.

- Requerimientos de Memoria

  A diferencia de lo que sucede con representaciones de tipo Voxel (o mapa de bits) los BSP son mucho mas eficientes en lo que se refiere a requerimientos de memoria, acercandose a los niveles alcanzados por Quadtrees y Octrees. El concepto de clasificación está inherentemente incorporado en la representación BSP de imágenes y volúmenes, esto permite desarrollar resistencia al ruido y alcanzar mayores niveles de compresión.

- Estructura Jerárquica

  A diferencia de la representación B-rep y Voxel, los BSP son estructuras jerárquicas y por consiguiente permiten asociar el nivell de detalle con la profundidad alcanzada al procesar el arbol. Esto permite utilizar algoritmos de simplificación muy sencillos, que pueden ejecutarse incluso en tiempo real. Al igual que Quadtrees y Octrees la propiedad de *multiresolución* está implícitamente ligada a la propia representación.

Al comparar una aplicación basada en BSP con otras parecidas basadas en Marching Cubes los autores concluyen que este nuevo enfoque permite detectar discontinuidades (no sólo contornos) y que la estructura resultante no precisa de complejos algoritmos de simplificación, al ser inherentemente multiresolución. Por el contrario se reconoce que en general, Marching Cubes permite reconstruir una representación mas detallada.

6.2.1 Construcción del BSP

La construcción del BSP representación de una imágen se contruye aplicando un algoritmo de básicamente cuatro pasos (figura 14). De este modo la representación discretizada del espacio que correspondía a la imágen se convierte en una representación segmentada de regiones sin discontinuidades, es decir, "homogeneas". La extensión a 3D del algoritmo permite representar volúmenes a partir de una serie de imágenes correspondientes a secciones transversales, de las típicamente utilizadas en medicina.

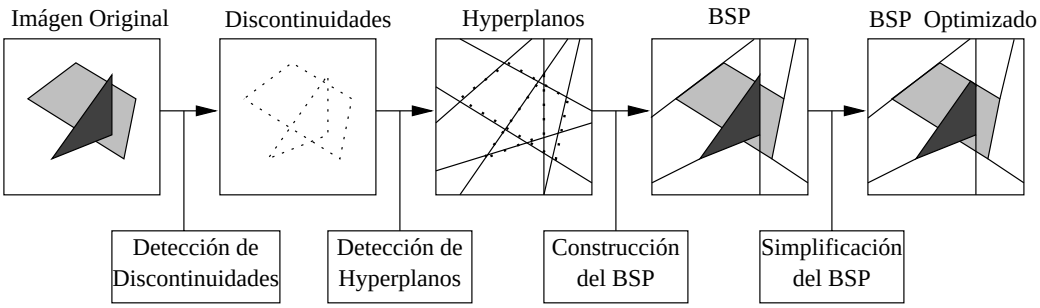


Figura 14: R-tree de un objeto.

- Detección de Discontinuidades

  Esta fase del proceso utiliza algoritmos tradicionales de procesado de imágen para determinar todos los posibles puntos de discontinuidad. Para ello el articulo propone un filtro multiple basado en pasabajos + operador gradiente + detección de maximo + binarización. De todas formas, no se descarta la posibilidad de crear filtros específicos dependiendo de la naturaleza de las imágenes.

- Detección de Hyperplanos

  Para determinar los hyperplanos candidatos el articulo presenta el metodo de Hough. Dicho metodo, de tangible base matemática, produce una mejor solución porqué sensible a las propiedades topológicas de los datos. De todas formas resulta impracticable a grandes conjuntos de datos, debido a sus monstruosos requerimientos de memoria.

Aplicado a reconstrucción de volúmenes el metodo de Hough no permite manejar conjuntos de datos de magnitud similar o superior a los 256x256x256 voxels. En estos casos se opta por una solución basada en la ordenación de los hyperplanos. El resultado conseguido por este metodo puede que no sea tan óptimo pero al menos los requerimientos de memoria se vuelven asequibles.

- Construcción del BSP

  La construcción del resultante BSP está eminentemente basada en metodos heurísticos. Se intenta determinar que hyperplanos es mejor utilizar y en que orden de tal forma que el BSP generado sea óptimo. Se entiende como óptimo un BSP que se pueda simplificar fácilmente y genere una estructura lo más compacta y balanceada posible.

  El articulo presenta la posibilidad de almacenar funciones polinómicas (y no valores) en los nodos terminales del arbol BSP, de tal forma que se puedan representar no sólo zonas estrictamente "homogeneas" sino zonas con determinada "textura", al permitirse almacenar varios atributos en cada nodo. Aunque la aplicación implementada se limita a utilizar únicamente funciones constantes o lineales esto aumenta el rendimiento del algoritmo sensiblemente ya que le permite representar como nodos hoja del arbol no sólo zonas "homogéneas" sino también zonas de "transición".

- Simplificación del BSP

  El articulo aconseja realizar una construcción del arbol BSP mas exhaustiva y luego proceder a su simplificación, colapsando las sub-ramas del arbol en hojas finales siempre que presenten un error inferior a un determinado máximo. Según los autores, esto permite generar estructuras BSP de mayor detalle.

6.2.2 Utilización del BSP

Una vez el BSP ha sido convenientemente generado es posible realizar su almacenaje, transmisión, manipulación y/o visualización. Normalmente, todos estos procesos se podrán realizar en tiempo real.

Para realizar el efecto de *probing* es suficiente construir el BSP representación de la herramienta a utilizar (por ejemplo un cubo semitransparente)

y realizar a continuación la unión con el BSP de la escena. Gracias a las propiedades de los BSP esta operación se puede realizar en tiempo real.

La visualización, incluso si ésta implica efecto de semi-transparencia, se puede realizar simplemente recorriendo el arbol BSP y aplicando el algoritmo del pintor sin necesidad de utilizar ningún hardware gráfico especializado.

En el articulo también se presenta una técnica de *poda interactiva* que permite al usuario seleccionar el nivel de detalle requerido acelerando asíla visualización. De este modo los propios usuarios pueden ajustar el nivel de interactividad de la aplicación al nivel de detalle requerido en cada momento. Esto permite manejar adecuadamente estructuras BSP extraordinariamente complejas, evitando sacrificar a priori la interactividad del proceso o los detalles de la información.

6.3 Jerarquía de mapas de oclusión (HOM)

Objetivo

Enviar menos objetos al pipeline de visualización, comparando los objetos con unos oclusores. Puede enviar objetos que no se visualizarán (ZMH-96).

Entorno

Escenas con gran complejidad en z: gran profundidad y muchos objetos repartidos a lo largo del eje z.
Cualquier objeto puede ser un oclusor. No hay ningún requerimento de tamaño ni de geometría.
Adecuado para escenas dinámicas.

Requerimientos

Se tiene que poder leer el frame buffer.
Como estructuras de datos sólo requiere una lista de matrices bidimensionales, de tamaño decreciente (los mapas).

Esquema general del algoritmo

Repetimos por cada imagen:

- Seleccionamos los oclusores.

- Construimos la jerarquía (que es lineal) de mapas.
- Construimos el buffer de estimación de la profundidad de los oclusores.
- Los objetos que están dentro del view frustum pasan un doble test:
  ¿La proyección del objeto intersecta en su totalidad con los oclusores? (Se compara el objeto con la jerarquía de mapas).
  ¿El objeto está a una profundidad mayor que los oclusores? (Se compara la profundidad del objeto con la de los oclusores usando el buffer de estimación de la profundidad).
  En caso de pasar afirmativamente los dos tests, el objeto no es visible y no se envía al pipeline.

Creación del primer mapa de oclusión

Enviamos los oclusores al pipeline, sin renderizar (por ejemplo, de color blanco). Leyendo el frame buffer tenemos el primer mapa, el de nivel cero. Podemos tener en cuenta aquí la opacidad de los oclusores.

Creación de los siguientes mapas

Agrupamos los valores de un nivel en cuadrados de 2x2 ó 4x4 posiciones. Dado un cuadrado de un nivel, se calcula una posición en el siguiente nivel promediando las opacidades del cuadrado. Puede usarse hardware de texturas para acelerar este proceso.

Intersección de un objeto con el HOM

Seleccionamos los objetos en el frustum de visión. Hallamos la caja contenedora (orientada según los ejes) y proyectamos sus vértices. Construimos el cuadrado contenedor de estos ocho vértices.
Escogemos el nivel en que el tamaño de las posiciones sea parecido al del cuadrado contenedor. Si una posición intersectada por el cuadrado no es opaca (mayor que un umbral), se hace recursión al nivel anterior. Pasa el test si todas las posiciones son opacas.

Construcción del buffer de estimación de la profundidad

Cuadriculamos el plano de proyección. Para cada oclusor hallamos la máxima z de los vértices de su caja contenedora. Dados los oclusores que intersectan una cuadrícula, asignamos a ésta la máxima de las z de los oclusores.

Intersección de un objeto con el buffer de estimación

Obtenemos la proyección de la caja contenedora de un objeto y hallamos su cuadrado contenedor. Le asignamos la menor z de los vértices de la caja. Comparamos ésta con la de todas las cuadrículas intersectadas por el cuadrado contenedor. Si la z del objeto es mayor que las de todas las cuadrículas, el objeto está detrás y pasa el test.

Selección de oclusores

En el preproceso seleccionamos unos cuantos objetos, evitando los pequeños y difíciles de proyectar.
Para cada imagen, de esta lista escogemos los que entren en el frustum de visión y, de éstos, los N más cercanos. Estos serán los oclusores de aquella imagen. Podemos aplicar algun tipo de coherencia temporal entre imágenes sucesivas.

Entornos dinámicos

No se realiza la preselección de oclusores. Obtendremos los oclusores no de una pequeña lista sinó de la lista de todos los objetos.

6.4 Wavelets. Wavelets esféricos.

En esta sección describiremos brevemente el formalismo de las wavelets, para poder dar la base que nos permitirá describir las wavelets esféricas, una estructura jerárquica de gran aplicabilidad para describir funciones definidas en la esfera, S^2 (ver *(SS95)*).

6.4.1 Wavelets, una muy breve introducción

Las wavelets son una base de funciones que permiten representar una dada función a múltiples niveles de detalle *(SDS96)*. Dado que son localizadas tanto espacial como temporalmente, son muy buenas para aproximaciones suaves de funciones: la localidad espacial viene de su suavidad (decaimiento para altas frecuencias) y sus momentos nulos ('*vanishing moments*'). Existen algortimos rápidos (de orden lineal) para calcular los coeficientes de las wavelets para varios problemas computacionales.

En el caso clásico de wavelets en el eje real, las wavelets se definen como díadas de traslaciones y escalados de una función particular y fija. Típicamente se construyen con la ayuda de una *función de escala*. Tanto estas como las funciones de wavelets soportan relaciones de refinamiento, lo que significa que una función de escala o wavelet a una dada resolución (j) puede escribirse como una combinación lineal de bases de funciones de escala, de la misma forma pero escaladas, a un nivel más fino (nivel $j+1$).

wavelets (multirresolución). Consideremos el espacio de funciones $L^2 = L^2(S^2, d\omega)$, es decir, todas las funciones de cuadrado integrable sobre S^2. Se define un análisis multirresolución a una secuencia de subespacios cerrados $V_j \subset L^2$, con $j \geq 0$, tal que

- $V_j \subset V_{j+1}$, los espacios más refinados con índices mayores.
- $\cup_{j\geq 0} V_j$ es densa en L^2.
- Para cada j, las funciones de escala $\varphi_{j,k}$ con $k \in \kappa(j)$ existen y son tal que $\{\varphi_{j,k} \mid k \in \kappa(j)\}$, es una base de Riesz para V_j. Una base de Riesz de un espacio de Hilbert es un subconjunto $\{f_k\}$ contable tal que todo elemento de f del espacio puede escribirse un fomra unívoca como $f = \sum_k c_k \ f_k$, y existen dos constantes positivas A y B tal que $A \|f\|^2 \leq \sum_k |c_k|^2 \leq B \|f\|^2$.

Conviene pensar acerca de $\kappa(j)$ como un conjunto generalizado de índices tal que $\kappa(j) \subset \kappa(j+1)$. En el caso de la línea real podemos tomar $\kappa(j) = \left\{0, 2^{-j}, ..., 1-2^{-j}\right\}$. Notar que, a diferencia del análisis multirresolución clásico, las funciones de escala no necesitan ser transladadas o escaladas de una función particular.

La primera propiedad implica que para toda función $\varphi_{j,k}$ existe un conjunto de valores $\{h_{j,k,l}\}$ tal que

$\varphi_{j,k} = \sum_l h_{j,k,l} \ \varphi_{j+1,l}$,

con los $h_{j,k,l}$ definidos para $j \geq 0$, $k \in \kappa(j)$ y $l \in \kappa(j+1)$. Cada función de escala satisface una relación de refinamiento diferente. En el caso clásico tenemos que $h_{j,k,l} = h_{l-2k}$, es decir, las secuencias $h_{j,k,l}$ son independientes de de escala y posición.

Cada análisis de multirresolución está acompañado por un análisis de multirresolución *dual*, consistente en un conjunto de espacios anidados $\widetilde{V}_j$ con bases dadas por funciones de escala $\widetilde{\varphi}_{j,k}$ que son biortogonales a las funciones de escala: $\langle \varphi_{j,k}, \ \widetilde{\varphi}_{j,k'} \rangle = \delta_{k,k'}$ para $k, k' \in \kappa(j)$. Estas funciones duales satisfacen relaciones de refinamiento con coeficientes $\left\{\widetilde{h}_{j,k,l}\right\}$.

En el caso de que las funciones de escala y las funciones de escala duales coincidan ($\varphi_{j,k} = \widetilde{\varphi}_{j,k}$ para todo j y k), las funciones de escala forman una base *ortogonal*. En el caso de que el análisis multirresolución y el análisis multirresolución dual coincidan ($V_j = \widetilde{V}_j$ para todo j pero no necesariamente $\varphi_{j,k} = \widetilde{\varphi}_{j,k}$) las funciones de escala se dicen *semiortogonales*. Ortogonalidad y semiortogonalidad muchas veces implican bases de funciones globalmente soportadas, lo que tiene sus lógicas desventajas. En lo siguiente no asumiremos ninguno y trabajaremos en el marco más general posible, en que ni el análisis multirresolución ni las funciones de escala coinciden.

Uno de los pasos cruciales para un análisis multirresolución es la construcción de wavelets. Estas representan la diferencia entre dos niveles de representación sucesivos, es decir que forman una base para los espacios W_j, dónde $V_j \oplus W_j = V_{j+1}$. Sea el conjunto de funciones $\{\psi_{j,m} \mid j \geq 0, m \in M(j)\}$ con $M(j) \subset \kappa(j+1)$ un nuevo conjunto de índices. Si

- El conjunto es una base de Riesz para $L^2(S^2)$
- El conjunto $\{\psi_{j,m} \mid m \in M(j)\}$ es la base de Riesz de $W(j)$.

decimos que las $\psi_{j,m}$ definen una base de wavelets esféricas.

Una propiedad importante es el número de momentos nulos: Una wavelet tiene $\widetilde{n}$ momentos nulos si $\widetilde{n}$ polinomios independientes $P_i, 0 \leq i \leq \widetilde{n}$ existen tal que $\langle \psi_{j,m}, P_i \rangle = 0$, para todo $j \geq 0,\ m \in M(j)$. Aquíse definió a los polinomios P_i como restricciones en la esfera de polinomios en $\Re^3$. Observar que polinomios independientes en $\Re^3$ pueden volverse dependientes luego de restringirlos a la esfera, como por ejemplo $\left\{1, x^2, y^2, z^2\right\}$.

Para un dado conjunto de wavelets tenemos las funciones $\widetilde{\psi}_{j,m}$ que son biortogonales a las wavelets, es decir $\left\langle \psi_{j,m}, \widetilde{\psi}_{j',m'} \right\rangle = \delta_{m,m'}\ \delta_{j,j'}$ para $j, j' \geq 0,\ m \in M(j),\ m' \in M(j')$. Esto implica $\left\langle \widetilde{\psi}_{j,m}, \varphi_{j,k} \right\rangle = \langle \widetilde{\varphi}_{j,k}, \psi_{j,m} \rangle = 0$, para $m \in M(j)$ y $k \in \kappa(j)$, y para $f \in L^2$ podemos escribir la expansión

$f = \sum_{j,m} \left\langle \widetilde{\psi}_{j,m}, f \right\rangle \psi_{j,m} = \sum_{j,m} \gamma_{j,m} \psi_{j,m}$

Dadas las relaciones arriba mencionadas, también podremos escribir a las funciones de escala $\varphi_{j+1,l}$ como combinación lineal de funciones de escala y wavelets menos detalladas usando las secuencias duales

$\varphi_{j+1,l} = \sum_k \widetilde{h}_{j,k,l} \varphi_{j,k} + \sum_m \widetilde{g}_{j,m,l} \psi_{j,m}$

de no ser explicitado de otra manera, se asumirá que los índices corren en $m \in M(j)$, $k \in \kappa(j)$ y $l \in \kappa(j+1)$.

Dado el conjunto de coeficientes de escala de una dada función f, $\{\lambda_{n,k} = \langle f, \widetilde{\varphi}_{j,k} \rangle, k \in \kappa(n)\}$, con n un dado nivel de resolución mayor, la transformada rápida calcula los $\{\gamma_{j,m}, n > j \geq 0, m \in M(j)\}$, es decir, las aproximaciones menos finas a la

función subyacente. Un paso de la transformada rápida calcula los coeficientes de un dado nivel (j) basándose en el nivel inmediatamente superior $(j+1)$: $\lambda_{j,k} = \sum_l \widetilde{h}_{j,k,l}\ \lambda_{j+1,k}$ y $\gamma_{j,m} = \sum_l \widetilde{g}_{j,m,l}\ \lambda_{j+1,l}$ y un paso de la transformada inversa calcula $\lambda_{j+1,k} = \sum_k h_{j,k,l}\ \lambda_{j,k} + \sum_m g_{j,m,l}\ \lambda_{j,m}$.

6.4.2 wavelets de segunda generación.

La filosofía básica detrás de las *wavelets de segunda generación* es construir wavelets con todas las propiedades deseadas (localización, transformada rápida) adaptadas a dominios mucho más generales que la recta real. Para ello se necesita construir wavelets que sean *adaptadas a una medida* de la superficie. Denotaremos la medida usual para la esfera como $d\omega$. Con respecto a las wavelets originalmente definidas (ver **(SDS96)**) sólo se mantiene la noción de que una función de la base puede escribirse como una combinación lineal de funciones de la base a un nivel más subdividido y refinado, perdiéndose la propiedad no fundamental de cambios de escala y traslaciones para definir las wavelets. Los coeficientes del filtrado de las wavelets de segunda generación no son las mismas en todo el dominio, pueden cambiar localmente para reflejar la cambiante naturaleza de una superficie y su medida. Ya no puede utilizarse la transformada de Fourier, y el mecanismo de construcción utilizado es el esquema de elevación, o *Lifting Scheme*.

Wavelets, el 'lifting scheme' La idea básica de este método de construcción es comenzar con un análisis multirresolución básico y construir uno cuyas funciones de base sean más suaves, o las wavelets tengan más momentos nulos. Si las bases son finitas, las bases "*elevadas*" también lo serán. Denotaremos los coeficientes del análisis original con el superíndice ^{a} de antiguo, comenzando por los los filtros $h^a_{j,k,l}$, $g^a_{j,m,l}$, $\widetilde{h}^a_{j,k,l}$, $\widetilde{g}^a_{j,m,l}$. el *esquema de elevación* dice que puede encontrarse un nuevo conjunto de filtros como

$$h_{j,k,l} = h^a_{j,k,l} \quad g_{j,m,l} = g^a_{j,m,l} - \sum_k s_{j,k,m}\ h_{j,k,l}$$
$$\widetilde{g}_{j,m,l} = \widetilde{g}^a_{j,m,l} \quad \widetilde{h}_{j,k,l} = \widetilde{h}^a_{j,k,l} + \sum_m s_{j,k,m}\ \widetilde{g}_{j,m,l}$$

y que, para cualquier elección de $\{s_{j,k,m}\}$, los nuevos filtros serán automáticamente biortogonales, lo que nos dará una transformación invertible. Las funciones de escala $\varphi_{j,l}$ son las mismas en el análisis multirresolución original y el elevado, mientras las funciones de escala duales y las wavelets primarias cambian, satisfaciendo ahora

$$\psi_{j,m} = \sum_l g^a_{j,m,l}\ \varphi_{j+1,l} - \sum_k s_{j,k,m}\ \varphi_{j,k}$$
$$\widetilde{\varphi}_{j,k} = \sum_l \widetilde{h}^a_{j,k,l}\ \widetilde{\varphi}_{j+1,l} + \sum_m s_{j,k,m}\ \widetilde{\psi}_{j,m}$$

Observar que en la ecuación superior las únicas incógnitas son las $s_{j,k,m}$, y las elegimos libremente para obtener alguna propiedad deseada para las

wavelets $\psi_{j,m}$.

La Transformada rápida de wavelets puede escribirse ahora

$\gamma_{j,m} = \sum_l \widetilde{g}^a_{j,m,l}\ \lambda_{j+1,l}$

$\lambda_{j,k} = \sum_l \widetilde{h}^a_{j,k,l}\ \lambda_{j+1,l} + \sum_m s_{j,k,m}\ \gamma_{j,m}$

Es decir, una secuencia de dos pasos: primero los filtros duales viejos, luego la actualización de los coeficientes de la función de escala con los coeficientes de las wavelets usando los coeficientes $\{s_{j,k,m} \mid k\}$. La transformada inversa es

$\lambda_{j+1,l} = \sum_k h^a_{j,k,l}\ (\lambda_{j,k} - \sum_m s_{j,k,m}\ \gamma_{j,m}) + \sum_m g^a_{j,m,l}\ \gamma_{j,m}$

La transformada rápida de Wavelets elevadas Tanto el análisis directo (análisis) como el inverso (síntesis) se realizan por cada nivel. El primero empieza desde el nivel más refinado hasta la raíz, mienras que el segundo realiza el camino inverso. AnálisisI calcula los coeficientes no elevados del nivel superior (padres) mientras que AnálisisII calcula la elevación si estamos usando una base elevada, siendo vacía en otro caso. SíntesisI calcula el elevación inversa, mientras que SíntesisII calcula las funciones de escala al nivel inferior (hijos).

Análisis

```
For nivel=NivelHojas to NivelRaiz

   AnalisisI (nivel)

   AnalisisII (nivel)
```

Síntesis

```
For nivel=NivelRaiz hasta NivelHojas

  SintesisI (nivel)

  SintesisII (nivel)
```

Básicamente, las wavelets vienen en dos tipos: 1- Elevadas de la Wavelet perezosa (*lazy wavelet*), lo que involucra funcione sinterpolantes y transformadas de vértices, y 2- elevadas de la Wavelet de Haar, lo que involucra wavelets basadas en caras.

6.4.3 Funciones de escala interpolantes

Primero veremos la transformación "perezosa", llamada asíporque no realiza nada, solo **submuestrea los coeficientes**. Sus filtros se pueden escribir como $h^a_{j,k,l} = \widetilde{h}^a_{j,k,l} = \delta_{k,l}$ y $g^a_{j,m,l} = \widetilde{g}^a_{j,m,l} = \delta_{m,l}$. En el caso de funciones interpolantes siempre podemos tomoar como duales a las distribuciones de Dirac: $\widetilde{\varphi}_{j,k}(x) = \delta(x - x_{j,k})$ que son inmediatamente biortogonales, ademas de conducir a productos internos triviales.

El conjunto de funciones resultantes de funciones de escala interpolantes y funciones de Dirac como duales puede verse como una elevación dual de la wavelet perezosa, lo que implica que $h_{j,k,k'} = \delta_{k,k'}$, $h_{j,k,m} = \widetilde{s}_{j,k,m}$, $\widetilde{g}_{j,m,k} = -\widetilde{s}_{j,k,m}$, $\widetilde{g}_{j,k,m'} = \delta_{m,m'}$. Las wavelets estarán dadas por $\widetilde{\psi}_{j,m} = \varphi_{j+1,m}$ y las wavelets duales por $\widetilde{\psi}_{j,k}(x) = \delta(x - x_{j+1,m}) - \sum_k \widetilde{s}_{j,k,m}\delta(x - x_{j,k})$. Dado que hemos aplicado la elevación dual, las wavelets primarias no han cambiado y entonces aún no tienen momentos nulos. Podemos aplicar ahora el lifting primario para mejorarl este problema y asegurar al menos un momento nulo para las wavelets primarias:

$\widetilde{h}_{j,k,l} = \delta_{k,l} + \sum_m s_{j,k,m}\ \widetilde{g}_{j,m,l}$

$g_{j,m,l} = \delta_{m,l} - \sum_k s_{j,k,m}\ h_{j,k,l}$

con lo que la wavelet resultante puede escribirse como $\psi_{j,m} = \varphi_{j+1,m} - \sum_k s_{j,k,m}\ \varphi_{j,k}$.

Bases de vértices Consideremos los conjuntos de índices localmente localizados alrededor de un dado sitio $x_{j+1,m}$, como se muestra en la figura 15. El índice del sitio se nota $m \in M(j)$ y los vértices vecinos tienen índices v, f y e respectivamente. Podemos pensar a los sitios $m \in M(J)$ como yaciendo en medio de una cierta arista padre (índices "impares"), mientras que los extremos de esta arista $k \in \kappa(j)$ forman los índices "pares", de forma tal que $\ell \in \kappa(j) \cup M(j) = \kappa(j+1)$ dá el conjunto de todos los índices. Para ellos utilizaremos un esquema local para referirnos a los índices, $k \in \kappa_m \subset \kappa(j)$, ver figura 15.

Para todos las bases de vértices los coeficientes de escala no elevados son simplements submuestreados durante el análisis y "sobremuestreados" durante la síntesis.

AnálisisI(j)

- $\forall k \in \kappa(j) : \lambda_{j,k} := \lambda_{j+1.k}$
- $\forall m \in M(j) : \gamma_{j,m} := \lambda_{j+1,m} - \sum_{k \in \kappa_m} \widetilde{s}_{j,k,m}\ \lambda_{j,k}$

SíntesisII(j)

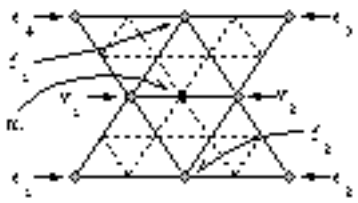


Figura 15: Vecinos utilizados para nuestras bases. Los miembros de los conjuntos de indices usados en las transformaciones se muestran en el diagrama.

- $\forall k \in \kappa(j) : \lambda_{j+1,k} := \lambda_{j,k}$
- $\forall m \in M(j) : \lambda_{j+1,m} := \gamma_{j,m} + \sum_{k \in \kappa_m} \widetilde{s}_{j,k,m}\ \lambda_{j,k}$

Veamos algunos casos particulares:

- **Perezosa**: Los pasos de análisis y síntesis se convierten en:$\gamma_{j,m} := \lambda_{j+1,m}$ y $\lambda_{j+1,m} := \gamma_{j,m}$, es decir que $\{\widetilde{s}_{j,k,m}\} = \emptyset$.
- **Lineal**: Utiliza $k \in \kappa = \{v_1, v_2\}$ para análisis y síntesis: $\gamma_{j,m} := \lambda_{j+1,m} - \frac{1}{2}(\lambda_{j+1,v_1} + \lambda_{j+1,v_2})$ y $\lambda_{j+1,m} := \gamma_{j,m} + \frac{1}{2}(\ \lambda_{j,v_1} + \lambda_{j,v_2})$. Notar que este esquema tiene en cuenta en forma correcta la geometría de los sitios m al nivel $j+1$. Aquí$\widetilde{s}_{j,v_1,m} = \widetilde{s}_{j,v_2,m} = \frac{1}{2}$.
- **Cuadrática**: El esténcil para estas bases está dado por $\kappa_m = \{v_1, v_2, f_1, f_2\}$ y explota los grados de libertad para matar las funciones x^2, y^2, z^2 y la función constante, lo que nos dará un simple sistema de 4 ecuaciones con 4 incógnitas para los $\widetilde{s}_{j,k,m}$. Este es un ejemplo de elevación dual con $\widetilde{s}_{j,k,m} = h_{j,k,m} = -\widetilde{g}_{j,k,m}$.
- **Butterfly (mariposa)**: utiliza todos los vecinos de la figura 15. Esta vez $\widetilde{s}_{v_1} = \widetilde{s}_{v_2} = \frac{1}{2}, \widetilde{s}_{f_1} = \widetilde{s}_{f_2} = \frac{1}{8}$ y $\widetilde{s}_{e_1} = \widetilde{s}_{e_2} = \widetilde{s}_{e_3} = \widetilde{s}_{e_4} = -\frac{1}{16}$.

Para asegurar que las wavelets tengan al menos un momento nulo, utilizaremos nuevamente la elevación, lo que no mejora la habilidad de la wavelet dual de eliminar más funciones, por lo que la capacidad de compresión no se ve aumentada. Propondremos wavelets de la forma $\psi_{j,m} = \varphi_{j+1,m} - s_{j,v_1,m}\ \varphi_{j,v_1} - s_{j,v_2,m}\ \varphi_{j,v_2}$. Los pesos $s_{j,k,m}$ se eligen de manera tal que la wavelet resultante tenga integral nula: $s_{j,k,m} = I_{j+1,m}/2I_{j,k}$ con $I_{j,k} = \int_{S^2} \varphi_{j,k} d\omega$. Vemos que cualquiera de las bases recién vistas puede elevarse con la misma expresión. Las integrales $I_{j,k}$ pueden calcularse al nivel más fino y recursivamente calcularse para niveles más gruesos:

AnálisisII(j)

$$\forall m \in M(j) : \left\{ \begin{array}{l} \lambda_{j,v_1} += s_{j,v_1,m}\ \gamma_{j,m} \\ \lambda_{j,v_2} += s_{j,v_2,m}\ \gamma_{j,m} \end{array} \right\}$$

SíntesisI(j)

$$\forall m \in M(j) : \left\{ \begin{array}{l} \lambda_{j,v_1} -= s_{j,v_1,m}\ \gamma_{j,m} \\ \lambda_{j,v_2} -= s_{j,v_2,m}\ \gamma_{j,m} \end{array} \right\}$$

Antes, los coeficientes de las funciones de escala eran simplemente muestras de la función a ser expendida, pero ahora, ene le caso elevado, se definen como el producto interno entre la función y las nuevas funciones de escala duales. Estas nunca se calculan explícitamente, son el límite de un esquema de subdivisión no estacionario.

6.4.4 Wavelets generalizadas de Haar y bases de caras

Consideremos triángulos esféricos resultantes de una construcción geodésica esférica $T_{j,k} \subset S^2$, con $k \in \kappa(j)$ (observar que son distintas a las anteriores), que satisfacen: $S^2 = \cup_{k \in \kappa(j)} T_{j,k}$ y su unión es disjunta; para cada j y k, $T_{j,k}$ puede escribirse como la unión de cuatro triángulos hijos $T_{j+1,k}$.

Si $\alpha(T_{j,k})$ es el área de un triángulo y definimos las funciones de escala y sus duales como $\varphi_{j,k} = \chi_{T_{j,k}}$ y $\widetilde{\varphi}_{j,k} = \alpha(T_{j,k})^{-1}\ \chi_{T_{j,k}}$, dónde $\chi_{T_{j,k}}$ es la función que vale 1 para $x \in T_{j,k}$ y 0 en otro caso. Definimos $V_j = \mathit{cerrar\ generado}\,\{\varphi_{j,k} \mid k \in \kappa(j)\}$, que generan un espacio multirresolución de $L^2(S^2)$.

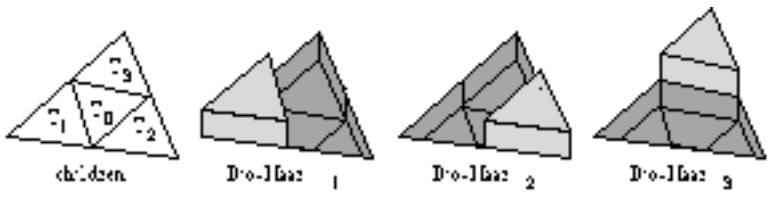


Figura 16: Las wavelets Bio-Haar.

Consideremos el conjunto de hijos de un triángulo $T_{j,*}$, que denominaremos $T_{j+1,l=0,1,2,3}$: llamaremos funciones Bio-Haar a las bases de la figura 16. Las wavelets se eligen como $\psi_{j,m=1,2,3} = 2(\varphi_{j+1,m} - I_{j+1,m}/I_{j+1,0}\ \varphi_{j+1,0})$ de forma que tengan integral nula. Un conjunto de wavelets duales semi ortogonal estará dado por $\widetilde{\psi}_{j,m} = \frac{1}{2}(\widetilde{\varphi}_{j+1,m} - \widetilde{\varphi}_{j,*})$. Estas wavelets tienen sólo un momento nulo, pero usando el esquema d eelevación dual podemos lograr que las wavelets duales tengan más momentos nulos. Sea $T_{j,k=4,5,6}$ los triángulos vecinos de $T_{j,*}$ (al nivel j) y $\kappa_m = \{*, 4, 5, 6\}$. Las nuevas

wavelets duales serán $\widetilde{\psi}_{j,m} = \frac{1}{2}(\widetilde{\varphi}_{j+1,m} - \widetilde{\varphi}_{j,*}) - \sum_{k \in \kappa_m} \widetilde{s}_{j,k,m}\, \widetilde{\varphi}_{j,k}$. Podemos elegir los $\widetilde{s}_{j,k,m}$ de forma que las wavelets duales "eliminen" polinomios $P = x^2, y^2, z^2, 1$. lo que conduce a un simple sistema de ecuaciones. Podemos escribir la elevación como

AnálisisII(j)

$\forall m \in M(j) : \gamma_{j,m} \quad -= \sum_{k \in \kappa_m} \widetilde{s}_{j,k,m}\ \lambda_{j,k}$

SíntesisI(j)

$\forall m \in M(j) : \gamma_{j,m} \quad += \sum_{k \in \kappa_m} \widetilde{s}_{j,k,m}\ \lambda_{j,k}$

6.4.5 Propiedades

Para reconstruir las funciones de escala sólo es necesario correr el algoritmo de reconstrucción haciendo $\lambda_{j_0,k} = \delta_{k,k_0}$ y todos los $\gamma_{j,m} = 0$,lo que converge para $j \to \infty$ a φ_{j_0,k_0}.

No existe una demostración de que las wavelets asíconstruidas formen una base no condicionada a exceción de las wavelets de Haar.

El esquema original de Buterfly tiene continuidad C^1 siempre que la conectividad de los vértices sea al menos 4. El esquema aquípresentado es C^1 siempre que exista una transformación C^1 de la triangulación esférica a una plana. Desgraciadamente, la división geodésica usada no posee esta propiedad.

6.4.6 Implementación

Para implementar esto se utilizó un "bosque" de Quadtrees, en dónde el nivel de raíz posee 4, 8 o 20 triángulos según se trate de tetrahedros, octahedros o icosahedros. Cada uno es dividido recursivamente en 4 subtriángulos. Las aristas son llamadas según el vértice opuesto y los hijos según el vértice que retienen (el central es T_0), lo que da un sistema consistente. Encontrar un vecino es $O(1)$ usando operaciones de bits sobre los nombres de aristas y vèrtices para guiar la búsqueda de los punteros. Cada vértice sólo se almacena una vez, y cada nivel posee punteros a él. Cada vértice almacena un único valor λ y γ para bases de vértice, y bases de triángulos almacenen un único valor λ y γ por cada triángulo esférico.

7 Bibliografía.

- **(AN84)** Ahuja, N. y Nash, Charles
"Octree representations of mooving objects"Computer
Vision, Graphics and Image Processing, 26(2):207-216, mayo 1984.

- **(ABJN-85)** D.Ayala, P.Brunet, R.Juan, I.Navazo
"Object representation by means of nonminimal division quadtrees and octrees"
ACM Transactions on Graphics, 4, 1(January 1985), 41-59

- **(BSW-83)** R.Bank, A.H.Sherman, A.Weiser.
"Refinement algorithms and data structures for regular local mesh refinement".
Scientific Computing, Eds. North-Holland Pbl. Co, pp 3-17, 1983.

- **(BCG-96)** G.Barequet, B.Chazelle, L.J.Guibas, J.S.B.Mitchell, A.Tal
"BOXTREE: A Hierarchical Representation for Surfaces in 3D"
EUROGRAPHICS '96. Volume 15, Number 3. pp. C-387 - C-396

- **(Ben-75)** J.L.Bentley
"Multidimensional binary search trees used for associative retrieval."
Comunications of ACM, 18(9):509-517, September 1975.

- **(BrN-92)** P.Brunet, I.Navazo
"Geometric Modelling of Volumes"
Eurographics Technical Report Series, Tutorials EG'92, EG92-TN-5, 1992

- **(CCV-85)** I.Carlbom, I.Chakravarty, D.Vanderschel
"A hierarchical data structure for representing the spatial decomposition of 3-D objects"
IEEE Computer Graphics and Applications 5, 4(April 1985), 24-31

- **(ChH-88)** H.Chen, T.Huang.
"A Survey of Construction and Manipulation of Octrees".
Computer Vision and Image Processing 43, pp 409-431, 1988.

- **(Cig-95)** P. Cignoni, E. Puppo, R. Scopigno,
"Representation and Visualization of Terrain Surfaces at Variable Resolution,"
Proceedings International Symposium on Scientific Visualization, pp. 50-68, R. Scateni (Ed.), World Scientific, 1995;

- **(Duc-98)** A. Duch.
"Design and Analysis of Spatial Data Structures".
Proyecto de Tesis.

- **(FiB-74)** R.A. Finkel and J.L. Bentley
"Quad trees: a data structure for retrieval on composite key".
Acta informatica, 4(1):1-9 1974.

- **(Flo-95)** L. De Floriani, P. Magillo, E. Puppo,
"Hierarchical Triangulation for Multiresolution Surface Description,"
ACM Transactions on Graphics, Vol. 14 No. 4, pp. 363-411, 1995;

- **(Flo-96)** L. De Floriani, P. Magillo, E. Puppo,
"Variable Resolution Terrain Surfaces,"
Proceedings Eighth Canadian Conference on Computational Geometry, pp. 202-210, 1996;

- **(Flo-97)** L. De Floriani, P. Magillo, E. Puppo,
"Efficient Encoding and Retrieval of Triangle Meshes at Variable Resolution,"
FMP '97;

- **(Fuc-80)** H. Fuchs, Z.M. Kedem, B. Naylor.
"On visible surface generation by a priori tree structure".
Proc. SIGGRAPH 80, Computer Graphics, 14 pp 124-133.

- **(Gar-82a)** I.Gargantini
"An effective way to represent quadtrees"
Communications of the ACM 25, 12(December 1982), 905-910

- **(Gar-82b)** I.Gargantini
"Linear Octrees for Fast Processing of Three-Dimensional Objects,"
Computer Graphics and Image Processing, Vol. 20 No. 4, pp. 356-364, 1982;

- **(Gar-82c)** I.Gargantini
"Detection of connectivity for regions represented by linear quadtrees"
Computers and Mathematics with Applications 8, 4(1982), 319-327

- **(Gar-83)** I.Gargantini
"Traslation, rotation and superposition of linear quadtrees"
International Journal of Man-Machine Studies, 18(3):253-263, marzo 1983.

- **(Gut-84)** A. Guttman.
"R-trees: A dynamic index structure for spatial searching".
Proc. ACM SIGACT-SIGMOD, Conf. Principles Database Systems, pp 569-592.

- **(HeK-97)** T.He, A.Kaufman.
"Collision Detection for Volumetric Objects".
Proceedings IEEE, Visualization'97.

- **(HKM-96)** M.Held, J.T.Klosowski, J.S.B.Mitchell, H.Sowizral, K.Zikan
"Realtime collision detection for motion simulation within complex environments"
Technical Report, Applied Math, SUNY, Stony Brook, 1996

- **(Hop-96)** H. Hoppe,
"Progressive Meshes,"
Computer Graphics Proceedings SIGGRAPH '96, pp. 99-108, 1996;

- **(Hub-96)** P.M.Hubbard.
"Approximating polyhedra with spheres for time-critical collision detection"
ACM Transactions on Graphics, 15(3):179-210, 1996

- **(Hun-78)** G.M.Hunter
"Efficient computation and data structures for graphics"
Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978

- **(HuS-79)** G.M.Hunter, K.Steiglitz
"Operations on images using quadtrees"

IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 2(April 1979), 145-153

- **(KaE-80)** E.Kawaguchi, T.Endo
"On a method of binary picture representation and its application to data compression"
IEEE Transactions on Pattern Analysis and Machine Intelligence 2, 1(January 1980), 27-35

- **(KEM-83)** E.Kawaguchi, T.Endo, J.Matsunaga
"Depth-first expression viewed from digital picture processing"
IEEE Transactions on Pattern Analysis and Machine Intelligence 5, 4(July 1983), 373-384

- **(KEY-80)** E.Kawaguchi, T.Endo, M.Yokota
"DF-expression of binary-valued picture and its relation to other pyramidal representations"
Proceedings of the Fifth International Conference on Patter Recognition, Miami Beach, December 1980, 822-827

- **(Kle-96)** R. Klein, W. Strasser,
"Generation of Multiresolution Models from CAD Data for Real Time Rendering,"
Theory and Practice of Geometric Modeling,
W. Straßer, R. Klein, R. Rau (Eds.), Springer Verlag, 1996;

- **(KHM-96)** J.T.Klosowski, M.Held, J.S.B.Mitchell, H.Sowizral, K.Zikan
"Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs"
IEEE Transactions on Visualization and Computer Graphics, 4(1) January-March 1998

- **(Knu-75)** D.E.Knuth
"The Art of Computer Programming, vol. 1, Fundamental Algorithms"
Second Edition, Addison-Wesley, Reading, MA, 1975

- **(Moo-95)** D.Moore.
"The cost of balancing generalized quadtrees".

Proceedings ACM, Solid Modeling'95.

- **(Nat-90)** B. Naylor, J. Amanatides, W. Thibault.
"Merging BSP trees yields polyhedral set operations".
ACM Computer Graphics, Vol 24, pp 115-124.

- **(OlW-83a)** M.A.Oliver, N.E.Wiseman
"Operations on quadtree-encoded images"
Computer Journal 26, 1(February 1983), 83-91

- **(OlW-83b)** M.A.Oliver, N.E.Wiseman
"Operations on quadtree leaves andy related image areas"
Computer Journal 26, 4(November 1983), 357-380

- **(Pla-93)** N.Pla
"Boolean operation and spatial complexity of Face Octrees"
EUROGRAPHICS Conf. Proc., volume 12, pages 153-164. Blackwell Publishers, 1993

- **(Sam-87)** H.Samet, R.E.Webber.
"Hierarchical data structures and algorithms for computer graphics".
CAR-TR-248 Computer Science Department Univ. of Maryland, 1987.

- **(SaW-87)** H.Samet, R.E.Webber
"Hierarchical data structures and algorithms for computer graphics".
CAR-TR-248 Computer Science Department Univ. of Maryland, 1987.

- **(Sam-89)** H.Samet,
"Implementing Ray Tracing with Octrees and Neighbor Finding,"
Computer & Graphics, Vol. 13 No. 4, pp. 445-460, 1987.

- **(SNV97)** C.Saona, I.Navazo, A.Vinacua
"Geometric Transformations in Octrees using Shears"
Departamento de Software, UPC, 1997, Ref: LSI-97-62-R.

- **(SDS96)** E.J.Stollnitz, T.D.DeRose, D.H.Salesin
"Wavelets for Computer Graphics: Theory and Applications",
Morgan Kaufmann Publishers, San Francisco, CA, 1996

- **(SS95)** P.Schröder, W.Sweldens
"Spherical Wavelets: Efficiently Representing Functions on the Sphere",
Computer Graphics Proceedings (SIGGRAPH 95), pp. 161-172, 1995

- **(Sub-97)** K. Subramanian, B. Naylor.
"Converting Discrete Images to Partitioning Trees".
IEEE Transactions on Visualization and Computer Graphics, Vol 3, No 3, pp 273-288.

- **(Tn-87)** W. Thibault, B. Naylor.
"Set operations on polyhedra using binary space partitioning trees".
ACM Computer Graphics, Vol 21, pp 153-162.

- **(ZMH-96)** H.Zhang, D.Manocha, T.Hudson, K.E.Hoff III
"Visibility Culling using Hierarchical Occlusion Maps"
Proc. of ACM Siggraph'97