CEIG - Spanish Computer Graphics Conference (2013)
M. Carmen Juan and Diego Borro (Editors)

EBP-Octree: An Optimized Bounding Volume Hierarchy for Massive Polygonal Models

A. Aguilera[1], F. Feito[1] and F.J. Melero[2]

[1]Dpto. Informática, Universidad de Jaén
[2]Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada

{angel,ffeito}@ujaen.es, fjmelero@ugr.es

Abstract

This paper presents a data structure to efficiently handle a hierarchy of bounding volumes on massive polygonal models, such as those obtained from 3D scanning devices. The Extended Bounding-Planes Octree (EBP-Octree) is able to manage massive polygonal models by using a spatial indexation of the surface and storing a hierarchy of bounding volumes composed of planes from the original surface. In this work we detail the geometric and design criteria that have been considered in order to create, just once for each model, an out-of-core data structure that will be dynamically loaded in run-time while traversing the octree in environments such as collision detections of progressive transmission. Due to the applications that might use this data structure, the main goals are to obtain a tight volume at each node and a fast transition among disk and main memory while loading and releasing the octree branches.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—

1. Introducción

Con la generalización del uso de escáneres 3D, es común la obtención de modelos con un gran detalle de representación, formados por varias decenas de millones de polígonos. Trabajar con estos modelos de forma interactiva y en tiempo real hace que necesitemos trabajar con ordenadores muy potentes que tengan una gran capacidad de cálculo y de memoria principal, a pesar de que para la mayoría de las operaciones no es necesario tener cargado todo el modelo a su mayor resolución. Por esto en la literatura podemos encontrar diferentes métodos o técnicas para poder representar y visualizar modelos de forma progresiva que permiten una carga a diferente resolución del modelo dependiendo de una serie de criterios como puede ser el punto de vista del observador y la cercanía a cierta parte del modelo [Hop96] [Hop97] .

En este trabajo se presenta una extensión de la estructura de jerarquía de volúmenes BP-Octree [MCT08], que se ha modificado para que pueda trabajar con modelos de datos formados por varias decenas de millones de polígonos. La nueva jerarquía de volúmenes, denominada EBP-Octree (Extended Bounding Planes Octree) trabaja con datos de 64 bit, soportando árboles de hasta 20 niveles de profundidad. Además, se han optimizado los algoritmos de selección y construcción de los volúmenes envolventes, se ha definido una estructura de archivos para la gestión de memoria externa durante la construcción y manipulación de la estructura de datos, y se ha realizado un estudio de tiempos y memoria frente a volumen envuelto para justificar el criterio de selección de planos utilizado.

El EBP-Octree se calcula una sola vez para cada modelo poligonal, y se almacena en una serie de archivos que permiten cargar en memoria principal el árbol tantas veces como se necesite, sin tener que volver a realizar los cálculos. Otra característica que incorpora el EBP-Octree es que no se carga completamente en memoria principal la estructura de datos, sino que se realiza una gestión de los datos a modo de caché de forma que se carga en memoria sólo una pequeña parte del modelo, y va realizándose un trasvase de memoria a disco según van cambiando las necesidades que se tengan en un instante dado.

2. Trabajos previos

El trabajo que aquí presentamos se puede enmarcar en la gestión interactiva de grandes modelos. Los diferentes métodos o técnicas de representación de modelos que se pueden encontrar publicados se pueden clasificar según [Sha02] en dos grupos, las representaciones implícitas y constructivas o las representaciones enumerativas y combinatoriales. Las representaciones implícitas y constructivas consisten en definir una función que nos clasifique un conjunto de puntos del espacio indicando si estos puntos pertenecen o no al objeto. Las representaciones enumerativas y combinatoriales proporcionan un serie de reglas que sirven para poder generar un conjunto de puntos que pertenecen al modelo.

Dentro de las representaciones enumerativas y combinatoriales se encuentran un conjunto de técnicas que utilizan una representación del modelo con diferentes niveles de detalle (LOD) en su geometría, encontrándose técnicas que tienen almacenado para un mismo objeto diferentes niveles de detalle [Cla76] utilizando en cada caso la que más convenga, u otras que adaptan el objeto dependiendo de la posición del observador [Hop96] [Hop97] cargando a mayor resolución las partes del mismo que están más próximas al observador. Otras técnicas para representar objetos sólidos consiste en la descomposición espacial del objeto mediante estructuras de datos jerárquicas [ABJN85] [SW83], como podría ser la de incluir el objeto en un voxel y subdividir este voxel inicial en ocho subvoxel, dando esto una estructura jerárquica la cual se representa con un árbol de ocho hijos (octree).

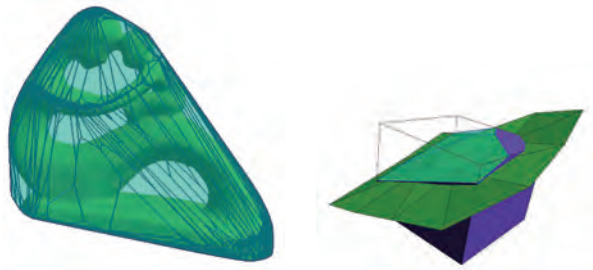
Independientemente de la visualización de los objetos en la escena, nos interesa que el EBP-Octree sea una estructura que permita al usuario interaccionar en tiempo real con el modelo representado e incluso ser capaces de detectar colisiones no sólo en el contexto de un sistema de interacción háptica sino también en entornos de colisiones entre varios modelos de alta resolución. Para ello existen diferentes técnicas clásicas como puede ser utilizar la jerarquía de volúmenes envolventes (AABB [Ben97], OOBB [GLM96], k-DOPs [KHM*98], etc...), árboles de indexación espacial (octrees [SW83] , BSP-Trees [RLVN91], KD-Trees [Ben75]) o árboles que indexen jerárquicamente volúmenes envolventes (SP-Octrees [MCT05], BP-Octrees [MCT08, MCT10]).

En todos los trabajos anteriormente citados, las dimensiones de los modelos utilizados son bastante modestas. Actualmente los trabajos que se desarrollan para trabajar con modelos muy grandes están enfocados bien a la visualización [CGG*04] [GM05] [YSGM04] o bien al cálculo de la detección de colisiones en entornos dinámicos [LGS*09] [VMTS10] [SPO10] [PKS10] [TMHT10] [BJ10], pero estos últimos trabajan con modelos que no llegan a los dos millones y medio de polígonos. La única excepción a esta tónica es el trabajo [YSLM04] que consigue manejar modelos con un tamaño similar a los que trabaja el EBP-Octree, si bien los propios autores reconocen un gran número de falsos positivos y su rendimiento no permitiría su uso en entornos hápticos.

2.1. El BP-Octree

Este trabajo constituye una ampliación y mejora del BP-Octree [MCT08], que consiste en un árbol octal con cuatro tipo de nodos: *blancos* y *negros*, al uso de los octrees clásicos; *grises*, que almacenan un conjunto de planos cuya intersección de semiespacios interiores forma un volumen convexo que envuelve la geometría original del modelo; y *hojas*, que además del conjunto de planos envolventes contiene la parte de geometría original del modelo poligonal. En la figura 1 se puede contemplar a la izquierda el nodo raíz de un BP-Octree, representando en un tono translúcido la envolvente convexa, y a la derecha un nodo hoja con la geometría original en verde y en translúcido la envolvente que forman los planos seleccionados.

En el BP-Octree se requiere que los planos que crean la envolvente formen parte de la geometría original del modelo, y se utilizan bien en su posición original o desplazándolos un *offset* hasta que engloban toda la geometría del nodo (en el caso de nodos hoja) o todas las envolventes de los nodos hijo (en el caso de nodos grises). De esta forma se garantiza que conforme se asciende por el árbol, la envolvente de nivel *n-1* contiene completamente a todos los volúmenes envolventes de los nodos de nivel *n*.



(a) (b)

Figura 1: *a). Volumen envolvente a nivel 0 de un BP-Octree. b) Nodo hoja [Mel08]*

3. Construcción del EBP-Octree

La idea fundamental del EBP-Octree es poder tratar modelos formados por varias decenas de millones de polígonos, montando una estructura de volúmenes envolventes definidos por los propios planos del modelo original. Esta estructura de datos será calculada y almacenada en disco una sola vez para cada modelo, de forma que los ficheros generados para cada modelo podrán ser cargados en memoria principal de forma rápida en tiempo de ejecución, sin tener que recalcular ninguno de los elementos que lo forman. A continuación se describen todos los pasos que se dan para

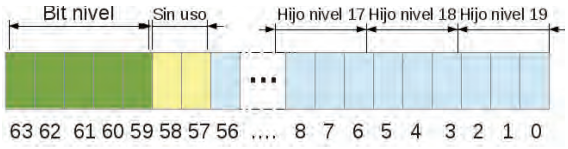


Figura 2: *Octcode de 64 bits.*

calcular y almacenar en disco esta estructura jerárquica de planos envolventes. Al ser muchos de estos pasos comunes al BP-Octree, nos centraremos en aquellos aspectos novedosos, partiendo del hecho que el EBP-Octree es de 64 bits, y no de 32 como su predecesor.

3.1. Dimensionamiento del octree

El nodo raíz en nuestra estructura de datos es la caja envolvente alineada a los ejes (AABB, [Ben97]) del modelo. Se realiza una construcción ascendente o *bottom up* del árbol por lo que se realiza una pre-definición del máximo nivel del árbol. Éste se estimará en función del tamaño de los triángulos del modelo original, de forma que en un nodo hoja del árbol se pueda contener un número significativo de triángulos que permita una rápida simplificación del modelo. El nivel máximo del árbol será aquel cuyas celdas puedan contener completamente a un triángulo equilátero de longitud de arista igual a la media de los lados de los triángulos del modelo poligonal. Para evitar recorrer los millones de polígonos que se manejan en el EBP-Octree, la media de la longitud de los lados se calcula tomando el 1% de los polígonos que forman el modelo original, tomándolos de forma dispersa en el vector de caras del modelo original para evitar el sesgo que pueda producir la coherencia espacial de la muestra.

Cada uno de los nodos del árbol octal vendrá referenciado por un *octcode* basado en los códigos Morton [Mor66]. Como se ha comentado anteriormente, el uso de *octcodes* de 64 bits permite alcanzar 20 niveles de profundidad y por tanto manejar modelos de una gran resolución. A modo de ejemplo, un modelo digital de un campo de fútbol (100x65 m.) estaría representado al máximo nivel de detalle con celdas de 0.19mm de lado, algo actualmente inalcanzable por los dispositivos de captura 3D. Como se aprecia en la figura 2, los cinco bits más significativos se utilizan para almacenar el nivel al que pertenece el nodo, y los 57 bits menos significativos indican la ruta desde el raíz hasta el nodo en cuestión. Este *octcode* se almacena como un entero largo.

3.2. Clasificación de los triángulos del modelo original y creación de nodos hoja

En el BP-Octree la indexación espacial de los triángulos se realizaba íntegramente en memoria, ya que los modelos con los que trabaja no eran muy grandes (<2M de polígonos). El trabajar con modelos formados por varias decenas de millones de polígonos obliga al uso de estructuras auxiliares en disco, mediante el uso de archivos temporales de 200MB. El número de estos archivos dependerá del tamaño del modelo original.

La indexación de cada triángulo se realiza mediante el cálculo de los octcodes de los nodos hoja que son atravesados por dicho triángulo, correspondiendo cada octcode a un nodo hoja. Se determina la celda del árbol que contiene completamente al triangulo, y se realizan recursivamente test de inclusión triángulo en caja para cada uno de los hijos, repitiendo recursivamente en caso de resultado positivo hasta alcanzar el máximo nivel determinado para el árbol. Para cada uno de los nodo hoja alcanzados se almacena una tupla <octcode, idTriangulo>en un vector que, al alcanzar los 200MB de espacio en memoria es ordenado por octcode y guardado en disco.

Esta ordenación permite obtener agrupados todos los triángulos de un mismo nodo mediante la aplicación de un clásico algoritmo de mezcla, como se muestra en la figura 3, de forma que se crea un nodo hoja por cada octcode que tenga asociados triángulos.

3.3. Creación de envolventes y nodos internos

La creación de los volúmenes envolventes sigue la filosofía del BP-Octree, si bien se han tenido que adaptar algoritmos para poder gestionar la ingente cantidad de datos que ahora se maneja. Uno de ellos es la clasificación de las esquinas de los nodos hoja [Mel08] con respecto a la superficie original del modelo (dentro o fuera), que ahora también se realiza *out-of-core* y en tres pasadas: las alineadas al eje X, a continuación las alineadas al eje Y y finalmente las alineadas con el eje Z. Esta clasificación es necesaria pues se carece inicialmente de nodos negros o blancos en la estructura de hojas creadas.

Para cada una de las doce aristas de cada nodo hoja se tiene que calcular cuantas veces son cortadas por un polígono y determinar la clasificación de los vértices de cada arista en función del número y orientación de los cortes. Gracias a que estas aristas son comunes a varios nodos hoja, para ahorrar espacio y tiempo de cálculo, se cambia el enfoque con respecto a [Mel08] y se aplica un recorrido no por los nodos hoja sino por las aristas de éstos. En la figura 4 se ve como el nodo1 y nodo2 comparten una arista, y es lógico trabajar sobre aristas en lugar de recorrer los nodos y calcular dos veces los puntos de corte común.

Una vez se tienen correctamente clasificadas las esquinas de todos los nodos hoja, se procede a la creación del volumen envolvente para cada uno de ellos, con la misma filosofía que en el BP-Octree.

Aunque el BP-Octree ya ofrece un criterio de selección de planos para la envolvente, mediante el uso del algoritmo *k-medianas* [KR90] sobre el conjunto de planos del nodo, hemos realizado un estudio comparativo con otras alternativas

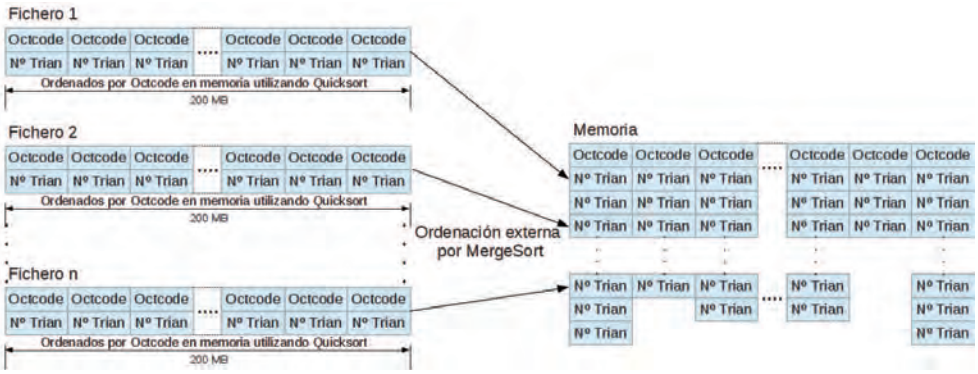


Figura 3: *Estructura de archivos temporales para indexación espacial.*

para la selección del porcentaje de planos que nos queremos quedar:

- Escoger el $k\,\%$ de planos que menos *offset* tengan aplicado para formar parte de la envolvente.
- Escoger un $k\,\%$ de planos de forma aleatoria

Según se desprende de la gráficas de volumen ocupado (figura 7 y tabla 2), resulta claro que a mayor porcentaje de planos utilizados en las envolventes mejor es el ajuste de éstas al modelo original, si bien en contrapartida supone un mayor tiempo de cálculo y un mayor espacio en disco, tal y como se refleja en la tablas 4 y 3 respectivamente. Dado que el tiempo de construcción sólo ha de computarse una única vez para la construcción del EBP-Octree, y la variación en los tiempos de carga del modelo es inferior al 10 % entre la selección aleatoria y el k-medianas, concluimos que el criterio de selección de planos por el agrupamiento o *clustering* de los mismos según su orientación es el más adecuado por las ventajas que proporciona.

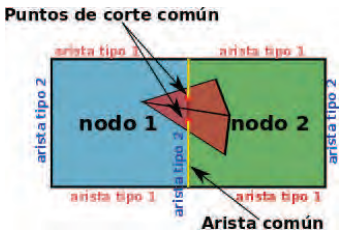


Figura 4: *Arista común a dos nodos, y sus puntos de corte.*

En cuanto al valor del parámetro k, esto es, a cuantos planos seleccionar, habrá que delegar en estudios posteriores con aplicaciones prácticas (p.ej. la interacción háptica) la determinación del umbral óptimo, pues a priori no tenemos más medida que el volumen envolvente.

Para el cálculo de las envolventes en los nodos internos, se sigue exactamente el mismo procedimiento del BP-Octree clásico, de forma que la envolvente en cada nodo está formada por un subconjunto de los planos que forman las envolventes en los nodos hijos, siguiendo el criterio de selección anteriormente descrito. La diferencia fundamental con la aproximación del BP-Octree es el uso de memoria externa para realizar una gestión eficiente de los recursos.

4. Organización de ficheros EBP-Octree

Para poder trabajar con modelos formados por varias decenas de millones de polígonos, se crea una estructura de ficheros que nos permita acceder a los datos deseados de forma eficiente. Por esto se utilizan siete ficheros para guardar toda la información necesaria del EBP-Octree. Esta división se realiza así para poder acceder a la información que se desee directamente, ya que por ejemplo, los datos que se necesitan para visualizar el modelo son diferentes a los que se necesitan para calcular un test de inclusión. Las extensiones de los ficheros que se utilizan son:

- `.vtx`, donde se guarda la información de los vértices que forman los polígonos del modelo original.
- `.tgl`, donde se almacena información de todos los polígonos del modelo. Cada línea del archivo es el número de vértices que forman cada polígono y los desplazamientos en el fichero .vtx para leer las coordenadas de cada uno de los vértices.
- `.fcn`, que es el fichero donde se guardan las normales de los polígonos que forman parte del modelo poligonal original.
- `.geo`, que guarda para cada nodo hoja el número de triángulos que lo cortan y la posición de cada triángulo en el fichero `.tgl`.
- `.bvp`, almacena la información de los vértices que forman la envolvente calculada de los nodos. Más concretamente se guarda para cada plano: un offset del fichero `.fcn` con la posición de la normal del plano, el número de vértices que forman el polígono de la envolvente definido por dicho plano y por último las coordenadas de cada uno de estos vértices.

- `.bpl`, donde se guarda la información de los planos seleccionados para definir la envolvente. Se guarda el número de planos seleccionados y para cada plano, el puntero a la normal en el fichero `.fcn` y el offset que se le ha aplicado al plano para que pueda formar parte de la envolvente. También se guarda el número de planos que forman la envolvente recortada del nodo y un puntero hacia el fichero ".bvp" donde están almacenados los vértices que forman los planos que se han obtenido tras recortar el nodo.
- `.oct`, que es el archivo que almacena el octree en sí, guardando la información necesaria para poder montar el árbol. La información de los nodos que forman el árbol se almacena por niveles, empezando por el nodo raíz y acabando con los nodos hoja. La información que se almacena para los nodos interiores es diferente a la que se almacena para los nodos hoja. La información que se guarda para un nodo interior es:
  - Dos bytes para saber el tipo de sus nodos hijos, si son negros, blancos, grises o nodos hoja (2 bits por hijo).
  - Un puntero o desplazamiento, en este mismo fichero, hacia donde esta almacenado el primer nodo hijo.
  - Otro puntero o desplazamiento dentro del archivo `.bpl` donde se almacenan consecutivamente los planos que forman la envolvente en dicho nodo.

  Para los nodos hoja, la información que se guarda es:
  - Un puntero o desplazamiento en el archivo `.geo` hacia los triángulos reales del modelo que cortan a ese nodo hoja.
  - Otro puntero o desplazamiento dentro del archivo `.bpl` donde se almacenan consecutivamente los planos que forman la envolvente en dicho nodo.

Toda esta estructura de archivos y su relación entre sí se muestra gráficamente en la figura 5.

5. Análisis de espacio ocupado y tiempo de construcción

En la tabla 1 mostramos el espacio ocupado en disco por el conjunto de archivos para el modelo Amazona de 25 millones de triángulos para cada uno de los tres criterios de selección de planos anteriormente propuestos. Puede observarse cómo el archivo mas grande es el que contiene los puntos de la envolvente en cada nodo, obtenidos a partir de la intersección de los planos entre sí. Este archivo sólo es necesario para los cálculos de colisiones entre dos EBP-Octrees y para la visualización, por lo que podría descartarse en caso de usar la estructura para interacción háptica.

En cuanto al tiempo de construcción de toda la estructura, podemos ver en la tabla 4 como en algunos modelos se puede alcanzar casi una hora de procesamiento, la carga posterior se realiza en cualquier caso en menos de 10 segundos, por lo que es asumible el tiempo de cómputo ya que sólo se realizará una vez. Es destacable la sobrecarga que supone la

Archivo	Aleatorio	K-medianas	K-medianas*	Offset
.blp	240.2	360.4	333.7	238.30
.bpo	76.3	76.3	76.3	76.3
.bvp	4505.6	5632.0	5427.2	4505.6
.geo	164.9	164.9	164.9	164.9
.oct	107.3	107.3	107.3	107.3
.tgl	19.1	19.1	19.1	19.1
.vtx	171.7	171.7	171.7	171.7
Total	5285.1	6531.7	6300.2	5283.2

Tabla 1: *Espacio en disco de la estructura EBP-Octree en MBytes. 40 % de planos seleccionados salvo en la columna marcada con *, que ha sido un 30 %.*

ejecución del algoritmo k-medianas, si bien su buena selección de planos (representada gráficamente en la figura 10) se nota especialmente en los niveles del árbol que se encuentran siempre en memoria, lo que redundará en una detección de colisiones con menos fallos de caché o en una visualización a bajo nivel de detalle más aproximada al modelo original. La diferencia que se produce entre el modelo Amazona y el modelo Moldura es debido al número de polígonos del modelo original que cortan los nodos hoja: mientras que el EBP-Octree de la Amazona se ha forzado para que tenga nivel máximo del octree de 10, teniendo tan sólo 3 polígonos de media que cortan a los nodos hoja, mientras la Moldura a nivel máximo del octree de 10 tiene 22, lo que redunda en un mayor tiempo de cómputo del algoritmo k-medianas en cada nodo, también se puede inferir que la Amazona podría ser gestionada en un árbol con un nivel máximo del octree menor.

La diferencia fundamental entre el BP-Octree y EPB-Octree es que los algoritmos del primero no están pensados para trabajar con modelos de datos muy grandes, mientras que el EBP-Octree sí, pudiendo éste tratar modelos formados por varias decenas de millones de polígonos. En este sentido, el EBP-Octree es del orden de unas 10 veces más rápido según los modelos con los que se ha podido comparar (tabla 4).

De los tres métodos de selección de planos el k-medianas es el que nos genera un volumen envolvente mejor, aunque los tiempos de construcción del EBP-Octree sean mayores. Para seleccionar qué porcentaje de planos envolventes para el k-medianas se hizo un estudio del volumen envolvente generado para los diferentes modelos, tomando como porcentajes los valores comprendidos entre el 10 % y el 40 %, con un intervalo de 10. Como se ve en la tabla 2 para el modelo de la Amazona cuanto mayor es el porcentaje mejor es el ajuste, lo que visualmente se puede observar en la figura 6. El problema que se genera al tomar un porcentaje mayor de planos envolventes es que las envolventes son más complejas, necesitando un mayor número puntos para poder almacenarlas, y es por esto por lo que los archivos que se generan tienen un mayor tamaño (tabla 3). Observando las

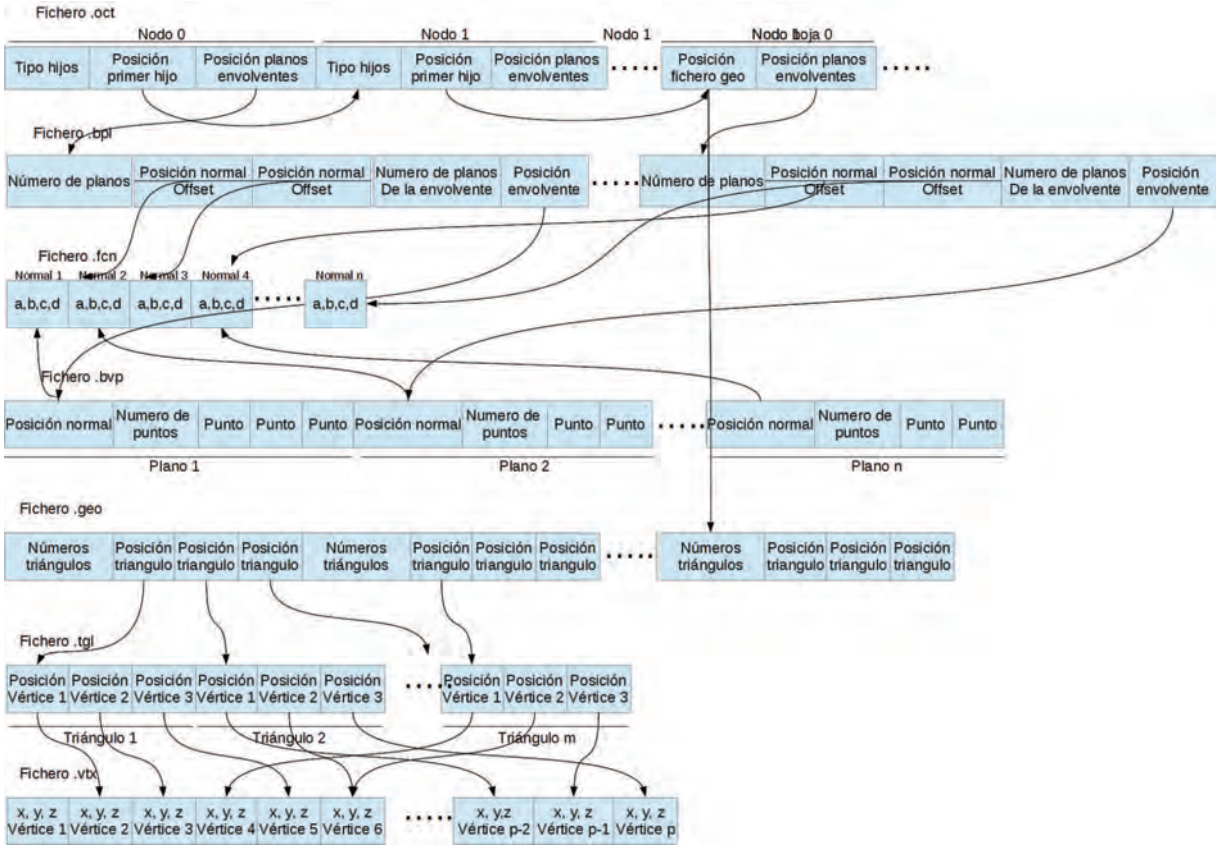


Figura 5: *Estructura de archivos para la gestión del EBP-Octree.*

gráficas 8 y 7 se puede deducir que un porcentaje de selección de planos envolventes optimo podría estar entre el 30 o el 40 por ciento.

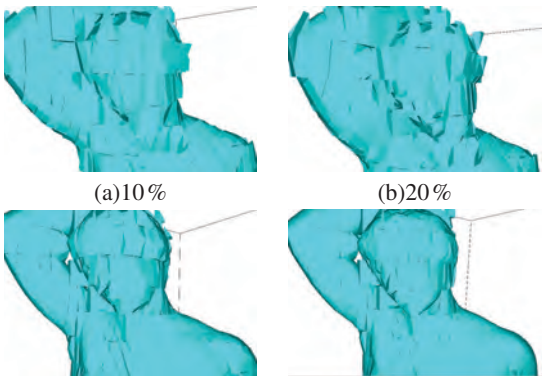


Figura 6: *Envolvente de nivel 6 generada seleccionando los planos con el método de k-medianas y variando el porcentaje de planos seleccionado.*

El volumen englobado en cada nivel del EBP-Octree, con respecto al volumen original del modelo poligonal, nos permite obtener una métrica de ajuste de las envolventes a la

Nivel	k-m 40 %	k-m 30 %	k-m 20 %	k-m 10 %
0	384,84 %	540,02 %	623,28 %	619,99 %
1	301,92 %	417,87 %	523,44 %	534,85 %
2	187,08 %	244,53 %	301,42 %	318,92 %
3	137,75 %	156,10 %	188,38 %	190,79 %
4	115,47 %	122,85 %	134,91 %	136,17 %
5	106,78 %	109,29 %	113,42 %	114,04 %
6	102,92 %	103,74 %	105,14 %	105,39 %
7	101,34 %	101,63 %	102,04 %	102,12 %
8	100,75 %	100,84 %	100,95 %	100,96 %
9	100,52 %	100,54 %	100,57 %	100,57 %
10	100,44 %	100,44 %	100,45 %	100,45 %

Tabla 2: *Volumen de la envolvente del EBP-Octree para el modelo Amazona con respecto al volumen del modelo poligonal original usando para la selección de planos el algoritmo k-medianas y variando el porcentaje de planos seleccionados.*

Modelo	*Tamaño*	*Nivel*	*AvgTri*	EBPO-Alea	EBPO-Km	EBPO-Km*	EBPO-Offset	BPO
Amazona	25M	11	3	570.4	733.1	670.7	568.6	-
Moldura	26M	11	22	2735.9	4161.5	3508.6	2697.0	-
Lucy	28M	11	14	2847.9	4142.0	3595.9	2845.0	-
Armadillo	150K	7	22	15.1	23.4	19.9	15.0	283.7
Gárgola	1.7M	9	12	140.2	206.2	180.6	138.7	2658.8

Tabla 4: *Tiempo de construcción de la estructura EBP-Octree en segundos. 40 % de planos seleccionados salvo en la columna marcada con *, que ha sido del 30 %. Tamaño en triángulos. Nivel: Nivel máximo alcanzado en el EBPO. AvgTri: Media de triángulos por nodo hoja. Modelos mostrados en la Figura 9.*

superficie original. En la figura 10 se muestra el volumen contenido en cada nivel, y si bien los valores numéricos (tabla 5) a partir del nivel 6 descienden por debajo del 10 % de volumen extra en todas las configuraciones, se puede apreciar como el algoritmo de k-medianas con el 30 % de los planos seleccionados alcanza mejor ajuste a partir de nivel 3 que el aleatorio o el que usa el criterio del offset con un 40 % de los planos candidatos.

El análisis de datos de espacio en disco, tiempo de construcción y volumen englobante nos hace determinar que el criterio predominante deberá ser el del volumen englobante, pues, como se aprecia en la figura 6, también un volumen más ajustado produce una visualización más cercana al original, con menos elementos distorsionadores.

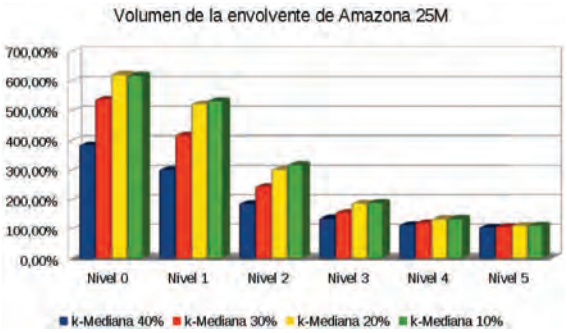


Figura 7: *Volumen de las envolventes hasta nivel 6 de Amazona.*

Archivo	k-m 40 %	k-m 30 %	k-m 20 %	k-m 10 %
.bpl	360,4	333,7	315,5	314,4
.bpo	76,3	76,3	76,3	76,3
.bvp	5632	5427,2	5227,7	5219,5
.geo	164,9	164,9	164,9	164,9
.oct	107,3	107,3	107,3	107,3
.tgl	19,1	19,1	19,1	19,1
.vtx	171,7	171,7	171,7	171,7
Total	6531,7	6300,2	6082,4	6073,2

Tabla 3: *Tamaño de los ficheros en MB generados con EBP-Octree para el modelo Amazona.*

Todos los datos se han obtenido tras ejecutar los programas y los modelos en un ordenador personal con procesador i3-2310M a 2.1GHz, memoria principal 6 GB DDR3.

6. Carga dinámica del EBP-Octree

El proceso descrito en el apartado anterior sólo se tiene que realizar una sola vez para cada modelo. Una vez construido el EBP-Octree, ya se puede cargar el modelo tantas veces como se quiera sin tener que realizar los cálculos pre-

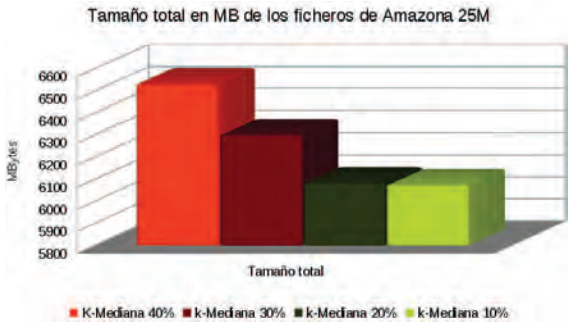


Figura 8: *Tamaño de los archivos generados con EPB-Octree para Amazona.*

Nivel	Aleatorio	K-medianas	K-medianas*	Offset
0	472.18 %	384.84 %	540.02 %	533.13 %
1	374.86 %	301.92 %	417.87 %	423.41 %
2	230.47 %	187.08 %	244.53 %	273.47 %
3	161.37 %	137.75 %	156.10 %	178.84 %
4	127.32 %	115.47 %	122.85 %	133.80 %
5	111.70 %	106.78 %	109.29 %	113.56 %
6	104.88 %	102.92 %	103.74 %	105.25 %
7	102.12 %	101.34 %	101.63 %	102.14 %
8	101.04 %	100.75 %	100.84 %	100.99 %
9	100.62 %	100.52 %	100.54 %	100.58 %
10	100.46 %	100.44 %	100.44 %	100.45 %

Tabla 5: *Volumen del EBP-Octree del modelo Amazona con respecto al volumen del modelo poligonal original. 40 % de planos seleccionados salvo en la columna marcada con *, que ha sido del 30 %.*

vios. Basta con leer la información de los ficheros que necesitemos, según se vaya a visualizar, a realizar test de inclusión punto en solido, detección de colisiones o trazado de rayos, por citar algunas de las aplicaciones.

Al trabajar con modelos formados por varias decenas de millones de polígonos, los datos que se necesitarían, por ejemplo, para poder visualizar el modelo a su máxima resolución, desbordarían la memoria principal. Para evitar este problema y viendo que en un instante dado sólo se utiliza una pequeña parte del modelo, el octree no se carga entero en memoria, sino que en un instante dado sólo se mantienen en memoria los nodos de los primeros niveles y aquellos subárboles a mayor nivel que sean estrictamente necesarios, lo que se representa esquemáticamente en la figura 11. Se define *nivel de corte* como el nivel máximo del octree que se mantiene en memoria principal. El nivel de corte del octree se calcula dependiendo del tamaño de la memoria principal del ordenador donde se va a cargar el modelo, de forma que en un ordenador con mayor capacidad de memoria se cargarán más niveles del octree que en otro con menos memoria.

Figura 9: *Algunos modelos utilizados en las pruebas (Amazona, Lucy, Moldura y Gárgola).*

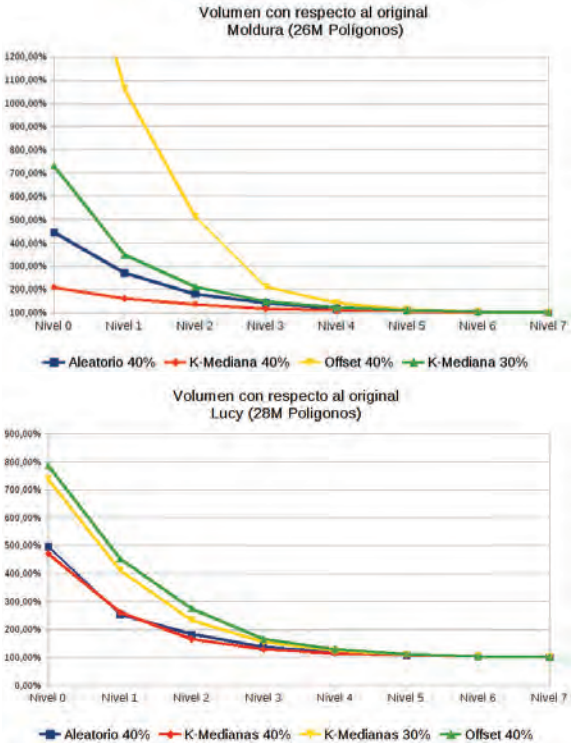


Figura 10: *Volumen con respecto al modelo poligonal del EBP-Octree de Moldura (superior) y Lucy (inferior).*

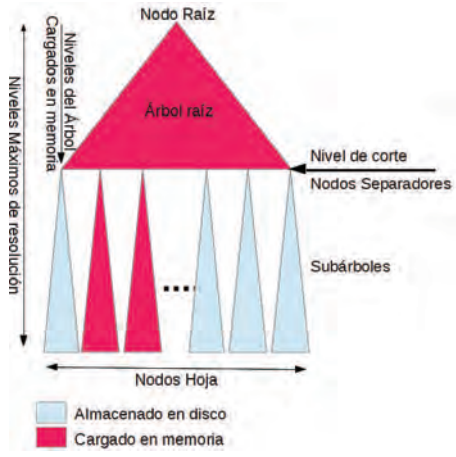


Figura 11: *Esquema de carga dinámica.*

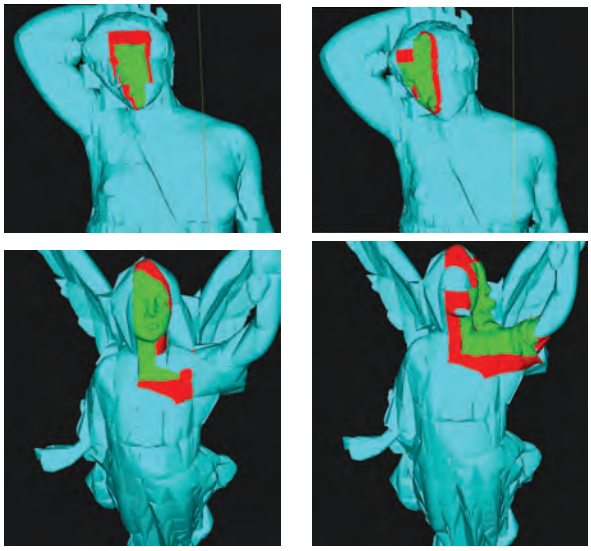


Figura 12: *Carga dinámica de nivel de detalle.*

El sistema de caché de subárboles resulta muy adecuado para las aplicaciones como la visualización adaptativa o la interacción háptica del modelo, donde es muy difícil que se produzcan grandes saltos en el espacio y por tanto sea necesaria una carga/descarga masiva de datos. En la figura 12 se pueden apreciar en verde los nodos a máximo nivel de profundidad, en rojo aquellos que están en un estado de precarga y en azul los nodos a nivel de corte del árbol (en este caso nivel 5 para la Amazona y 4 para Lucy).

Las primeras pruebas realizadas de esta carga dinámica han determinado que el árbol permanente en memoria ocupa entre 8MB y 12MB en función del algoritmo de selección de planos utilizado, y que cada subárbol de máximo nivel en memoria supone unos 250KB, cantidades fácilmente asumibles incluso en dispositivos móviles.

7. Conclusiones y trabajos futuros

Se ha presentado un método de construcción de una estructura de volúmenes envolventes espacialmente indexados (EBP-Octree) que es capaz de manejar en tiempo real modelos de varias decenas de millones de polígonos. Al ser el proceso de construcción del EBP-Octree independiente de la carga y posterior gestión del mismo, se puede generar la estructura de archivos en un ordenador más potente si hubiese restricciones de tiempo, y después usar la estructura de ficheros en ordenadores con menor potencia de cálculo.

También se ha descrito un método de carga dinámica adaptativa de la jerarquía de volúmenes, de manera que se conserve en memoria una estructura que englobe alrededor del 110 % del volumen original y se cargue a modo de caché aquellos subárboles que estén siendo objeto de uso más detallado (como en el caso de una interacción háptica o una visualización adaptativa).

Entre los trabajos futuros se encuentran la paralelización de la construcción del EBP-Octree, la aplicación de la jerarquía de volúmenes envolventes al cálculo de inclusiones punto-en-sólido que serán directamente aplicables en interacción háptica, y la detección de colisiones entre dos o más objetos de varias decenas de millones de polígonos. Asimismo, se está trabajando en la optimización del tamaño del fichero `.bvp`, pues hay mucha información redundante.

8. Agradecimientos

Agradecemos a Francisco Soler por su inestimable ayuda en la discusión y resolución de los problemas presentados en la implementación de la gestión de memoria externa, y a los revisores que nos han proporcionado valiosos comentarios para la mejora y ampliación del trabajo.

Agradecimientos también al repositorio de modelos escaneados 3D de Stanford por la utilización del modelo Lucy, al VCG-ISTI por el modelo de la Gárgola, y al Museo Municipal de Ecija por el modelo de La Amazona Herida que es propiedad de la Junta de Andalucía.

Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad y la Unión Europea (a través de los fondos FEDER) a través del proyecto de investigación TIN2011-25259.

Bibliografía

[ABJN85] AYALA D., BRUNET P., JOAN R., NAVAZO I.: Object representation by means of nonminimal division of quadtrees and octrees. *ACM Transaction on Graphics 4*, 1 (1985). 2

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (1975), 509–517. doi:http://doi.acm.org/10.1145/361002.361007. 2

[Ben97] BENGEN V. D.: Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4 (1997), 1–13. 2, 3

[BJ10] BARBIC J., JAMES D. L.: Subspace self-collision culling. *SIGGRAPH 10 ACM SIGGRAPH 2010 papers* (2010). 2

[CGG*04] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Transaction on Graphics 23*, 3 (2004), 796–803. doi:http://doi.acm.org/10.1145/1015706.1015802. 2

[Cla76] CLARK J. H.: Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (1976), 547–554. doi:http://doi.acm.org/10.1145/360349.360354. 2

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: Obb-tree: A hierarchical structure for rapid interference detection. *Proceedings SIGGRAPH'96* (1996), 171–180. 2

[GM05] GOBBETTI E., MARTON F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3d

models on commodity graphics platforms. *ACM Trans. Graph. 24*, 3 (2005), 878–885. doi:http://doi.acm.org/10.1145/1073204.1073277. 2

[Hop96] HOPPE H.: Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 99–108. 1, 2

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. *Computer Graphics 31*, Annual Conference Series (1997), 189–198. 1, 2

[KHM*98] KLOSOWSKI J., HELD M., MITCHELL J., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE transactions on Visualization and Computer Graphics 4*, 1 (1998), 21–36. 2

[KR90] KAUFMAN L., ROUSSEEUW P. J.: *Finding Groups in Data – An Introduction to Cluster Analysis*. John Wiley & Sons, 1990. 3

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast bvh construction on gpus. *Computer Graphics Forum 28* (2009), 375–384. 2

[MCT05] MELERO F., CANO P., TORRES J.: Combining sp-octrees and impostors for multiresolution visualization. *Computer and Graphics 29* (2005), 225–233. 2

[MCT08] MELERO F., CANO P., TORRES J.: Bounding-planes octree: A new volume-based lod scheme. *Computer and Graphics 32*, 4 (2008), 385–392. 1, 2

[MCT10] MELERO F., CANO P., TORRES J.: Deteccíon de colisiones en grandes modelos geométricos. In *Actas del Congreso Español de Informática Gráfica 2010* (2010). 2

[Mel08] MELERO F.: *BP-Octree: Una estructura jerárquica de volúmenes envolventes*. PhD thesis, Univ. Granada, 2008. 2, 3

[Mor66] MORTON G.: *A computer oriented geodetic data base and a new technique in file sequencing*. Tech. rep., IBM Ltd., 1966. 3

[PKS10] PABST S., KOCH A., STRASSER W.: Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. *Computer Graphics Forum 29* (2010), 1605–1612. 2

[RLVN91] RADHA H., LEOONARDI R., VETTERLI M., NAYLOR B.: Binary space partitioning tree representation of images. *Journal of Visual Communications and Image Processing 2(3)* (1991). 2

[Sha02] SHAPIRO V.: *Handbook of Computer Aided Geometry Design*. Elsevier, 2002, ch. Solid Modeling, pp. 473–518. 2

[SPO10] SCHVARTZMAN S. C., PEREZ A. G., OTADUY M. A.: Star-contours for efficient hierarchical self-collision detection. *SIGGRAPH 10 ACM SIGGRAPH 2010 papers 29* (2010). 2

[SW83] SAMMET H., WEBBER R.: Hierarchical data structures and algorithms for computer graphics. *IEEE Comp. Graphics and Applications 8*, 3 (1983), 48–68. OCTREE. 2

[TMHT10] TANG M., MANOCHA D., HILL C., TONG R.: Fast continuous collision detection using deforming non-penetration filters. *I3D '10 Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), 7–13. 2

[VMTS10] VOGIANNOU A., MOUSTAKAS K., TZOVARAS D., STRINTZIS M. G.: Enhancing bounding volumes using support plane mappings for collision detection. *Computer Graphics Forum 29* (2010), 1595–1604. 2

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D.: Quick-vdr: interactive view-dependent rendering of massive models. 131–138. doi:10.1109/VISUAL.2004.86. 2

[YSLM04] YOON S., SALOMON B., LIN M. C., MANOCHA D.: Fast collision detection between massive models using dynamic simplification. *Eurographics Symposium on Geometry Processing* (2004), 136–146. 2