Transmisión progresiva de grandes modelos usando una jerarquía de planos envolventes

Fco. Javier Melero Pedro Cano Juan Carlos Torres

fjmelero@ugr.es pcano@ugr.es jctorres@ugr.es

Dept. Lenguajes y Sistemas Informáticos

Universidad de Granada

ETS Ingenierías Informática y de Telecomunicaciones

18071 Granada

Resumen

En este artículo presentamos una nueva estructura de datos que permite transmitir y visualizar a traves de la red grandes modelos tridimensionales. Esta estructura, denominada *BP-Octree* (Bounding-Planes Octree) está basada en la descomposición espacial del modelo usando un octree, asignando a cada nodo del árbol un conjunto de planos que forman un volumen envolvente de la parte del modelo contenido en dicho nodo. Estos planos son obtenidos de los polígonos reales del modelo, y se seleccionan en cada nivel garantizando que engloban totalmente a los volúmenes resultantes de intersecar los planos envolventes seleccionados en los nodos hijos.
Usando esta representación, la transmisión del modelo a través de la red no se realiza por polígonos, sino que se envían los planos que se necesitan para la visualización en cada momento, o sus índices si ya han sido previamente usados. El algoritmo de visualización de la parte cliente se encarga de realizar la extracción de la geometría correspondiente mediante un simple algoritmo de recorte de mallas.

1. Introducción

Uno de los campos que la Informática Gráfica aún no ha conseguido resolver completamente es la visualización de modelos tridimensionales a través de la red. Diversos estándares han sido propuestos para representar modelos 3D (VRML, X3D, etc...) y en la literatura podemos encontrar numerosas referencias a técnicas aplicadas a la visualización remota [10, 12], pero ninguna de ellas ha sido adoptada por la industria de internet.

Por otro lado, los usuarios requieren modelos cada vez más detallados y por tanto más pesados en cuanto a tamaño. Este hecho hace imprescindible el uso de técnicas de visualización progresiva y adaptativa de los objetos tridimensionales, no sólo en la transmisión de modelos a través de la red (obvio por las limitaciones de ancho de banda) sino también en aplicaciones locales, debido al límite de memoria y procesamiento. Son ya clásicos los trabajos que se pueden encontrar en [5, 3, 8, 6].

La visualización interactiva de tales modelos complejos requiere un gran esfuerzo del hardware gráfico para conseguir el nivel de detalle y realismo esperado, y estas espectativas están reñidas con el concepto de interactividad, ya que la frecuencia de refresco ha de superar un umbral de 30fps. Este conflicto entre nivel de detalle y velocidad de visualización ha motivado el desarrollo de diversas técnicas que permitan compaginar ambos objetivos, mostrando el modelo simplificado cuando el usuario no puede apreciar los detalles por su lejanía y suministrando los detalles sólo cuando éstos son necesarios.

Las técnicas de nivel de detalle (LOD) basadas en la geometría se pueden clasificar como:

discretas [7], cuando se tienen varias instancias del mismo objeto a distintos niveles de detalle; *progresivas* [10], si el detalle se va obteniendo de una única estructura de datos; y *dependientes del observador* [11], que son una extensión de las anteriores, de forma que el nivel de detalle no es uniforme en todo el modelo, sino que es anisotrópico dependiendo del punto de vista.

Otra posibilidad para representar volúmenes y sólidos con divestintos nivels de detalle es representarlos con esquemas basados en descomposición espacial, usando estructuras de datos jerárquicas. Dentro de estas soluciones podemos encontrar el *Octree* y numerosas extensiones del mismo (Extended-Octrees, PM-Octrees, SP-Octrees, etc.) [1],[4].

La estructura de datos que presentamos en este trabajo está basada en un octree, con la particularidad de que en cada nodo, tanto interno como hoja, almacenamos una serie de planos que delimitan de forma convexa la superficie del modelo comprendida en el nodo, por lo que la denominamos *BP-Octree*(Bounding-Planes Octree).

2. BP-Octree

La idea básica con la que se construye el BP-Octree es la de almacenar en cada nodo un conjunto de planos que intersecados entre sí y con la celda del nodo, formen una envolvente convexa (no necesariamente la envolvente de mínimo volumen) de la geometría contenida en dicho nodo. Además, en los nodos hoja, se incluye información sobre los polígonos originales que corresponden a cada nodo. La particularidad de la estructura propuesta es que el conjunto de planos utilizados para crear la envolvente se extraen de las caras originales del modelo, ya que cada plano es homólogo a una cara.

En la figura 1 se muestra un pseudocódigo de la estructura de datos utilizada para los nodos, y un esquema general de la estructura de datos completa se puede ver en la figura 2. La figura 3 muestra el nodo raíz del BP-Octree para varios modelos.

Para cada plano en el nodo almacenamos

```
typedef long int Index;
typedef struct {
     Index planeIndex;
     double d;
} BPlane;

typedef long int Octcode;

class BPNode {
    vector<BPlane> boundingPlanes;
    Octcode oct;
}

class BPLeafNode:public BPNode {
    vector<Index> faces;
};
```

Figura 1: Estructura de datos básica del BP-Octree

únicamente el índice del plano, *planeIndex*, que referencia a un plano almacenado en una estructura de datos externa, accesible por el árbol. De esta forma, ahorramos espacio y eliminamos redundancias. Idéntica aproximación se sigue para la geometría final almacenada en los nodos hoja. Esta idea permite que, al trabajar con modelos geométricos muy grandes, sea el sistema operativo el que gestione los accesos a disco o a memoria.

El campo d de la estructura `BPlane` refleja un valor de desplazamiento que se puede aplicar al plano envolvente. Inicialmente, los planos se crean con desplazamiento cero, es decir, la cara cuya normal se utiliza para definirlo es coplanaria al mismo. Si dicho plano no deja en su semiespacio interior a todos los vértices de la zona del modelo de interés en un nodo determinado, se calcula el desplazamiento a lo largo de la normal necesario para que satisfaga este criterio, y éste es el valor que se almacena en el campo d.

Otro de los objetivos de este trabajo es conseguir la construcción de toda la estructura de datos en un tiempo de cómputo razonable. Para ello usamos un enfoque ascendente: inicialmente repartimos los polígonos en un grid tridimensional, que representa el último nivel predefinido del octree. A continuación, se asigna cada uno de estos polígonos a los nodos

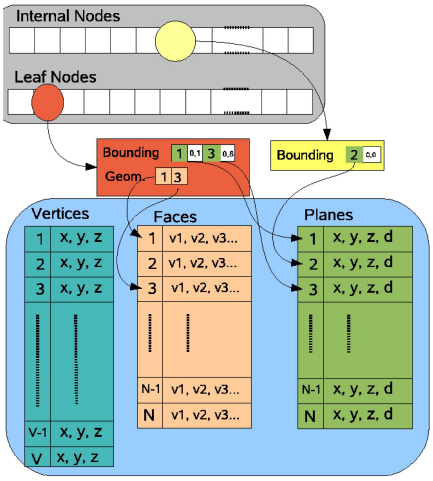

Figura 2: Esquema general del BP-Octree.

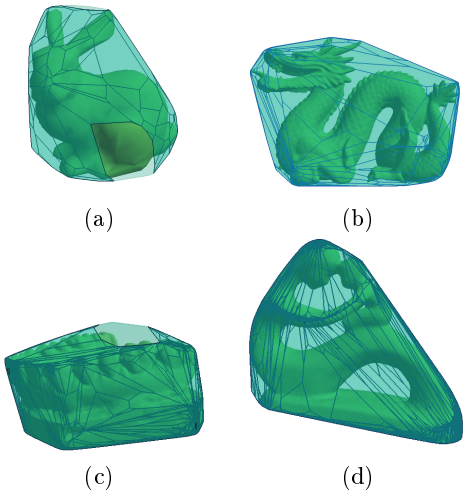

Figura 3: Envolventes diversas a nivel 0.

hoja, y se comienza a crear la envolvente partiendo de las envolventes en dichos nodos hoja.

Al diseñar la estructura de datos, hay ciertos aspectos que han tenido que ser tomados en cuenta para obtener una solución completa y correcta:

- *Gestión de grandes modelos.* Nuestra estructura de datos es capaz de manejar grandes modelos geométricos, de varios cientos de miles de polígonos. Uno de nuestros objetivos es la simplificación del modelo de forma jerárquica para poder realizar la transmisión progresiva y visualización adaptativa de los mismos, que sólo tiene sentido si hablamos de modelos con un importante nivel de detalle. Para conseguir esto, hemos utilizado una indexación espacial de los polígonos que permite, en caso de que sea necesario, una fácil gestión y clasificación de los mismos mediante memoria externa.

- *Garantizar que la totalidad de la geometría contenida en el nodo es usada al calcular su envolvente.* Como el octree es un entorno discreto, tenemos que asignar pri-

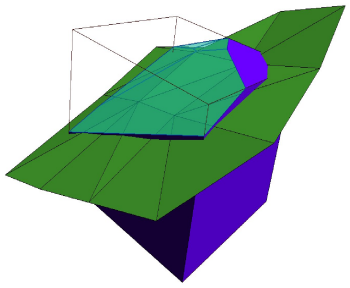

Figura 4: Ejemplo de nodo. En verde, la geometría del modelo. En azul claro, la envolvente calculada.

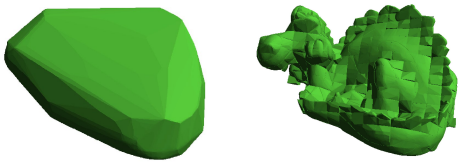

Figura 5: Phlematic Dragon y sus planos envolventes (cyan) en el nodo raíz (a) y en el nivel 4 del BP-Octree (b).

mitivas contínuas (polígonos) a voxels, y hacerlo suficientemente rápido. La solución obvia sería recortar todos los polígonos para que encajen exactamente en nuestros voxeles tridimensionales, pero esto nos llevaría a un tiempo de cálculo prohibitivo. En su lugar, usamos un algoritmo 3DDDA para recorrer cada polígono y determinar qué voxeles atraviesa.

- *Asegurarse de que la envolvente a nivel* n *está completamente contenida en la evolvente de nivel* n-1. Es importante mantener la coherencia entre niveles, ya que no tiene sentido que estemos en un nivel determinado del BP-Octree y al obtener un mayor detalle, resulte que la envolvente se ajusta peor al modelo -cuando debería ocurrir lo contrario. Para satisfacer este criterio, en los nodos hoja se calculan las envolventes usando la geometría real del modelo, y para los nodos internos, consideramos la geometría generada por la intersección de los planos envolventes de sus hijos.

- *Evitar huecos entre nodos adyacentes.* Al usar envolventes aproximadas, es casi imposible que dos nodos adyacentes compartan planos en su cara común, por lo que al visualizarlos, se vería un agujero en la unión. Nosotros hemos considerado conveniente añadir una serie de planos ficticios para taparlos, que no son más que las caras de cada nodo que son visibles en parte (ver polígonos color violeta en la figura 4).

- *Tratamiento de concavidades.* Cuando la geometría no es convexa, es bastante complicado encontrar una cara cuyo plano englobe a toda la geometría. Entonces, permitimos que los planos se "despeguen"de su cara origen con un desplazamiento tal que dejen en su semiespacio interior a todos los vértices de la geometría en cuestión, convirtiéndose en un *plano envolvente* por desplazamiento (ver figura 11).

- *Las zonas convexas consiguen menor simplificación que las irregulares.* En el caso de una esfera, todas sus caras (planos) satisfacen el criterio de que son envolventes de la geometría, con lo que el 100 % de las caras formarían parte del nodo raíz. Para evitar dicha situación, limitamos el número de planos en cada nivel, y éstos son elegidos siguiendo un criterio de distribución estadística. Un ejemplo de esta situación se puede ver en la figura 6

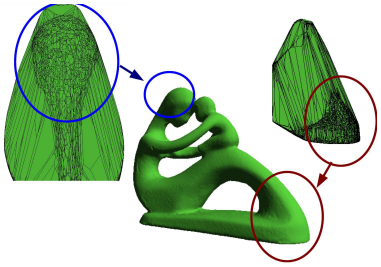

Figura 6: Alta densidad de planos debido a áreas del modelo extremadamente convexas.

Procedemos a explicar con más detalle el algoritmo de construcción.

2.1. Indexación espacial

Al ser el octree una estructura discreta, y el espacio tridimensional un entorno contínuo, está claro que la primera tarea que hay que realizar es discretizar nuestro modelo, es decir, asignar cada polígono al conjunto de nodos hoja que atraviesa. Esto lo realizamos usando un *octcode*, calculado como un código Morton tradicional [14].

El nodo raíz del árbol es una caja envolvente alineada a los ejes (AABB, [2]), es decir, no es necesariamente un cubo, sino que se ajusta al objeto en la dirección de los ejes y se determina por dos puntos significativos: $BB_{min} = (x_{min}, y_{min}, z_{min})$ and $BB_{max} = (x_{max}, y_{max}, z_{max})$

El punto básico para indexar todos los polígonos es determinar la celda del grid discreto en la que un punto p está. Para cada dimensión D, en un nivel de profundidad l, la coordenada discreta N_D se calcula como:

$$N_D = floor(\frac{2^l}{w_d}(d_p - d_{min})) \qquad (1)$$

donde w_d es la longitud del nodo raiz en dicha dimensión, d_{min} es el valor mínimo de la coordenada correspondiente a la dimensión D y d_p es el valor de la coordenada D del punto p.

2.1.1. Código Morton

El *código Morton* de un voxel se calcula entrelazando las coordenadas discretas (expresadas en binario) de cualquiera de los puntos que corresponden a dicho voxel.

En la figura 7 podemos ver cómo se intercalan las coordenadas X e Y resultando un número entero, único para cada nodo de un nivel determinado, pero no único para cualquier nodo de cualquier nivel. Podemos ver en la figura 7.c como el número 9 identifica tanto una celda del segundo como otra distinta del tercer nivel, siendo éstas de diferentes tamaños y coordenadas.

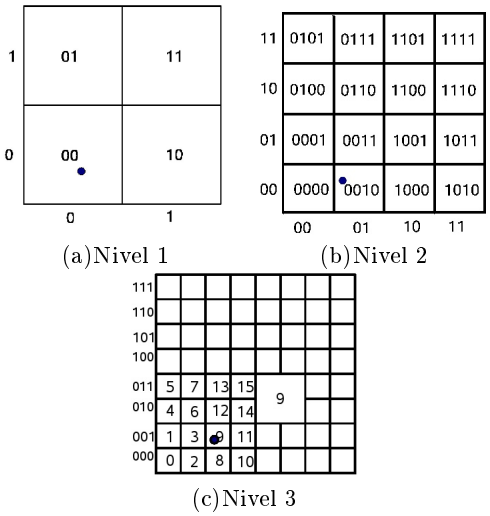

Figura 7: Numeración de los nodos

Para solucionarlo, añadimos al código del nodo información sobre el nivel al que pertenece, también en binario. Esto convierte el código Morton en un código localizador, ya que nos permite con un simple número saber las coordenadas exactas que delimitan el voxel. Hay dos opciones a la hora de añadir la información del nivel:

- Añadir los bits de nivel como los más significativos (figura 8a), lo que nos permite una ordenación primero en anchura de los octcodes.
- Añadir los bits de nivel como los menos significativos (figura 8b), teniendo un orden en profundidad de los octcodes.

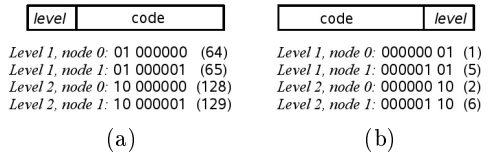

Figura 8: a) Orden en anchura; b) Orden en profundidad

En nuestra propuesta hemos elegido el orden en anchura para los octcodes, ya que al almacenarlos en dicho orden en un fichero, esta ordenación conserva la coherencia espacial para una transmisión progresiva del modelo.

2.1.2. Definición de la máxima profundidad

Dado el código locacional anteriormente descrito, podemos fácilmente estimar la memoria necesaria para direccionar cada voxel. El octcode se puede dividir en dos partes: el nivel y el índice de la celda en dicho nivel (Figura 8). Cada nivel de profundidad supone tres bits al índice de la celda, por lo que son necesarios $3l$ bits para alcanzar un nivel l,y el propio nivel l se representa con $log_2 l$ bits.

Por tanto, con cuatro bytes (nueve niveles) podemos tener 2^{27} = 134,217,728 nodos hoja. Según las pruebas realizadas con modelos de hasta 1.5M de polígonos, estos nueve niveles son más que suficientes para alcanzar una buena distribución de los polígonos, y 4 bytes son un `long int` en la mayoría de las implementaciones.

2.2. Direccionando Polígonos

Para identificar qué polígonos pertenecen a cada nodo, ejecutamos un algoritmo 3D-DDA [9] sobre cada polígono, identificando aquellos nodos que son atravesados por el polígono calculando el octcode tal y como se ha mostrado en la fórmula 1. A continuación, este polígono será tenido en cuenta para calcular la superficie envolvente en todos esos nodos.

Con este algoritmo garantizamos que todos los nodos atravesados por un polígono lo usarán para calcular su envolvente. Si no nos preocupa el tener en cada nodo una envolvente del modelo, sino una representación aproximada, podemos seleccionar sólo un subconjunto de dichos nodos, obteniendo por supuesto un mejor tiempo de construcción.

3. Construyendo el árbol

Tras la indexación de los polígonos, podemos deducir directamente los nodos hoja que nuestro árbol tendrá. Primero creamos los nodos hoja con la información dada por el índice espacial, identificandolos por su *octcode* e incluyendo en ellos los índices de los planos que pertenecen a dicho nodo.

Es posible que la indexación nos proporcione nodos con muy pocos polígonos (es el caso de modelos con relativamente pocos polígonos o polígonos muy grandes en relación a las dimensiones totales de la caja englobante del modelo). En este caso, como es escasa la aportación de cada nodo, podemos agrupar nodos hoja hermanos hasta que se cumpla un criterio determinado. Alguno de estos criterios pueden ser:

1. Agrupar mientras el nodo hoja resultante tenga menos de N polígonos asignados, o
2. Agrupar hasta que el nodo hoja resultante tenga más de M polígnos asignados, o
3. agrupar hasta que la media de polígonos entre los nodos hermanos sea al menos X, etc...

De esta forma, reducimos el número de nodos hoja hasta un 85 %, con el correpondiente ahorro de memoria y tiempo de cálculo.

Una vez conocemos el octcode de cada hoja, es muy simple leerlo y extraer la ruta completa desde el nodo raíz hasta la hoja, creando los nodos internos necesarios para reconstruir toda la ruta, como se muestra en la figura 9, donde se aprecia con líneas punteadas el camino que traza el nodo hoja hasta llegar a él, y los nodos internos que hay que crear.

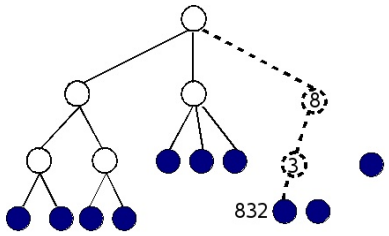

Figura 9: Construyendo el árbol.

4. Cálculo de las envolventes.

Como se ha comentado anteriormente, en cada nodo guardamos los índices de un conjunto de planos que delimitan completamente la parte del modelo 3D contenida en el nodo. Para determinar este conjunto de planos, seguimos un procedimiento recursivo ascendente, de forma que en cada nodo se realiza el menor número de cálculos posible, es decir, seguimos un paradigma *divide y vencerás*.

```
computeBounding(Node_T node){
  si esHoja(node)
    node->selectBoundingPlanes();
  si_no {
    para cada ch hijo de node {
      computeBounding(ch);
      node->addBPlanes(ch.getBPlanes());
      node->addBVertices(ch.getBVertices());
     }
    node->selectBoundingPlanes();
  }
}
```

Figura 10: Cálculo recursivo de la envolvente en cada nodo.

Como puede verse en la figura 10, en cada nodo seleccionamos sólo los planos que engloban la geometría de las envolventes de sus hijos con `addBVPlanes` (o la geometría real del modelo si estamos en una hoja) y sólo se usan los vértices de los hijos (`addBVVertices`) para garantizar que el volumen de la envolvente aumenta conforme subimos en el árbol.

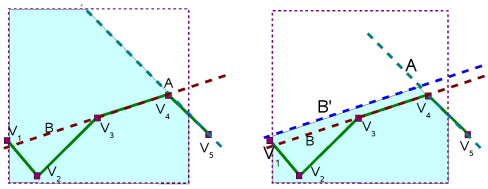

Figura 11: Seleccion de planos envolventes

El método utilizado para determinar si un plano forma parte de la envolvente o no es relativamente simple, y se muestra gráficamente en 2D mediante la figura 11. Partimos de un conjunto de planos *candidatos*, que son los que forman las envolventes de los nodos hijos (o la geometría original en el caso de que estemos en un nodo hoja). Comprobamos para cada plano candidato -con su desplazamiento, si lo tuviera- la posición de los vértices con respecto a su orientación. Si todos los vértices están en el semiespacio interior o son coplanares, el plano es tomado tal cual para formar parte de la envolvente. Este es el caso del plano A en la figura 11.a. El volumen envolvente resultante es el área cyan.

Si por el contrario, hay vértices en el semiespacio exterior, se calcula la distancia al punto más alejado, y se toma ese valor como desplazamiento con el cual se considera que el plano forma parte de la envolvente. En la figura 11.b podemos ver cómo el plano B es añadido al conjunto de planos envolventes con un desplazamiento tal que el vértice V_1, inicialmente en el exterior, queda contenido en el nuevo plano B'. La distancia d es exactamente la distancia de V_1 a B.

Obviamente, no todos los planos se incluyen en la envolvente, sino que se aplica un criterio de selección, que puede ser desde incluir todos los planos que disminuyan el volumen de la envolvente hasta un criterio estadístico que seleccione los planos que mejor describan la superficie subyacente. En cualquier caso, los primeros planos cuya utilidad es comprobada son aquellos con menor desplazamiento, ya que esto garantiza un mejor ajuste a la forma original.

En la tabla 1 se muestran los tiempos de construcción del BP-Octree para distintos modelos tridimensionales.

Modelo	Poligonos	Segs.
Bunny	69.451	23,16
Dragon	202.520	55,72
Teeth	233.204	86,75
Igea	268.686	127,88
Fertility	483.226	171,51
Angelo	562.879	131,12
Phlematic Dragon	715.933	197,24

Cuadro 1: Tiempos de construcción del BP-Octree para distintos modelos.

5. Transmisión progresiva

La estructura de datos que se presenta tiene como característica inherente la reutilización de datos, ya que los planos de un nivel dado son el conjunto del cual se seleccionan aquellos que forman la envolvente en el nivel inmediatamente superior del árbol. Es por ello por lo que se puede realizar una transmisión del modelo bien desde el nodo raíz o bien desde un nivel más bajo, en función del ancho de banda disponible.

En la tabla 2 mostramos la distribución del número de planos en cada uno de los niveles para cuatro modelos representativos: Stanford bunny, de 69K polígonos, Stanford Dragon, de 202K Polígonos, Fertility, de 483K polígonos, y el Phlematic Dragon de 715K polígonos. Se puede observar en la tabla 3 cómo en el primer nivel, apenas se utilizan más del 0.50 % del total de planos posibles, y como se puede observar en la figura 12, con alrededor del 10 % de los planos podemos obtener aproximaciones bastante buenas visualmente.

Nivel	Bunny	Dragon	Fertility	Phlematic
0	391	529	1777	743
1	1225	1658	3267	2166
2	3087	4230	7418	6031
3	7120	10822	18691	17663
4	16019	27637	42056	43295
5	33502	61272	91574	102967
6	46534	116649	185686	230607
7	-	52331	296797	423267
8	-	67	3399	144246

Cuadro 2: Número de planos en cada nivel del BP-Octree.

Nivel	Bunny	Dragon	Fertility	Phlematic
Level 0	0,56 %	0,26 %	0,37 %	0,10 %
Level 1	1,16 %	0,54 %	0,30 %	0,19 %
Level 2	2,45 %	1,20 %	0,82 %	0,52 %
Level 3	5,05 %	2,90 %	2,17 %	1,52 %
Level 4	10,27 %	7,11 %	4,35 %	3,24 %
Level 5	19,11 %	13,24 %	8,65 %	7,21 %
Level 6	19,82 %	21,08 %	15,39 %	14,57 %
Level 7	-	7,77 %	20,57 %	22,75 %
Level 8	-	0,01 %	0,22 %	7,27 %

Cuadro 3: Porcentaje del total de planos que son usados por primera vez en cada nivel.

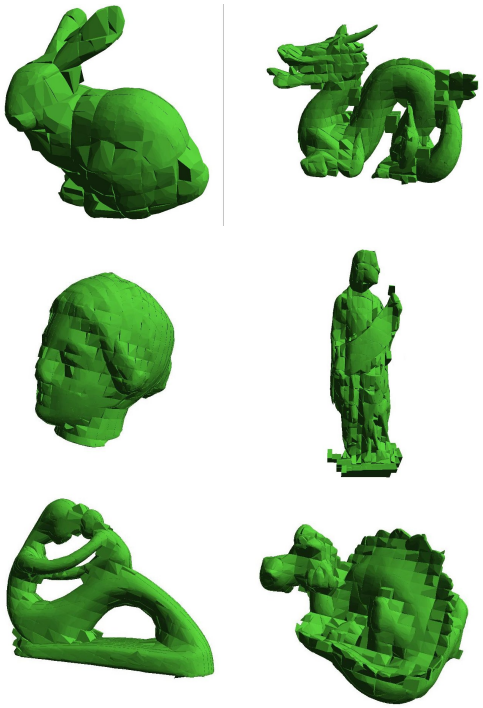

Figura 12: Distintos modelos a nivel 4, con menos del 10 % de los planos del modelo original.

Hay que destacar que el número de planos por nivel no supone la transmisión de cuatro coeficientes reales por plano, más su respectivo desplazamiento, sino que la mayoría de ellos ya han sido usados en niveles superiores o han sido utilizados previamente por otros nodos del mismo nivel, en cuyo caso sólo se transmite un índice entero y el desplazamiento.

En la figura 15 mostramos los distintos niveles de dos modelos, y en la figura 13 podemos consultar el volumen de datos transferido de forma acumulada por nivel. Se puede apreciar como el nivel 6 de la jerarquía multirresolución equivale en bytes transmitidos a la transmisión del modelo geométrico original, y en la figura 15 se puede observar como desde los primeros niveles del árbol se obtienen ya modelos con un aceptable nivel de detalle.

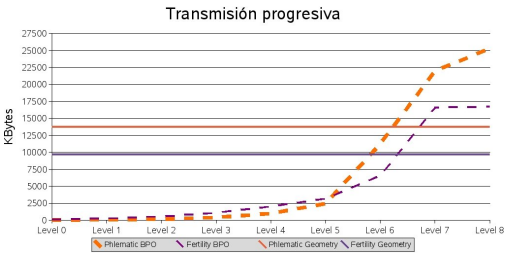

Figura 13: Gráfica acumulada de los bytes transmitidos para la visualización progresiva de los modelos Fertility y Phlematic Dragon. Se muestra también el volumen de datos necesario para disponer de la geometría completa.

6. Visualización adaptativa

La estructura que sustenta a los BP-Octrees, permite de una forma sencilla y trivial realizar una visualización adaptativa de los modelos. Los criterios para decidir si se desciende o no de nodo son de nuevo variables, pudiendo ir desde la distancia al observador a un criterio de percepción como la resolución de cada nodo.

A modo de ejemplo las imágenes que se muestran en al figura 14 han sido tomadas de forma que la diagonal de cada nodo visible mide, como máximo 40 pixeles. En primer plano podemos observar la geometría a máximo nivel de detalle, mientras que en segundo plano se observan los nodos a inferior resolución.

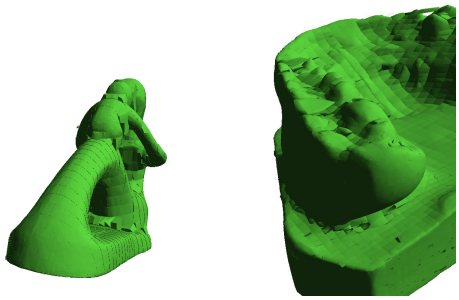

Figura 14: Visualización adaptativa de los modelos Fertility y Teeth.

7. Conclusiones y trabajos futuros.

Se ha desarrollado una estructura de datos que es capaz de manejar grandes modelos poligonales, independientemente de si tienen agujeros o sus caras no son triangulares. La construcción del BP-Octree se realiza en un tiempo razonable, y los planos están repartidos a lo largo de los distintos niveles de forma que la transmisión se puede realizar de una manera progresiva e incluso dependiente del observador sin una gran transferencia de datos.

Queda por mejorar la visualización del modelo jerárquico, bien realizando una difusión de las normales del modelo original, o usando técnicas basadas en imágenes, como los impostores mostrados en [13].

Entre las aplicaciones que se están investigando sobre esta estructura, destacan la detección rápida de colisiones en dispositivos hápticos y la aceleración de algoritmos de raytracing.

8. Agradecimientos.

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología y los fondos FEDER, a través del proyecto TIN2004-06326-C03-02 y por la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía a través del proyecto de excelencia TIC-401

Los modelos Bunny y Dragon han sido cedidos por el Stanford University Computer Graphics Laboratory. Phlematic Dragon es el modelo suministrado por los organizadores del congreso Eurographics 2007. Los modelos Fertility, Igea y Teeth proceden del Aim@Shape Repository. El modelo Angelo es un resultado del proyecto VIHAP3D.

Referencias

[1] D. Ayala, P. Brunet, R. Joan, and I. Navazo. Object representation by means of nonminimal division of quadtrees and octrees. *ACM Transaction on Graphics*, 4(1), 1985.

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

[3] M. Callieri, P. Cignoni, F. Ganovelli, G. Impoco, C. Montani, P. Pingi, F. Ponchio, and R. Scopigno. Visualization and 3d data processing in david's restoration. Technical report, Istituto di Scienza e Tecnologie dell'Informazione (ISTI), Consiglio Nazionale delle Ricerche, Pisa, Italy, Jan. 2004.

[4] P. Cano, J. Torres, and F. Velasco. Progressive transmission of polyhedral solids using a hierarchical representation. 2003.

[5] B. Chamberlain, T. DeRose, D. Lischinski, D. Salesin, and J. Snyder. Fast rendering of complex environments using a spatial hierarchy. In W. A. Davis and R. Bartels, editors, *Graphics Interface '96*, pages 132–141. Canadian Human-Computer Communications Society, 1996.

[6] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[7] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[8] C. Erikson, D. Manocha, and I. William V. Baxter. Hlods for faster display of large static and dynamic environments. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 111–120, New York, NY, USA, 2001. ACM Press.

[9] A. Fujimoto, T. Tanaka, and K. Iwata. *ARTS: accelerated ray-tracing system*, pages 148–159. Computer Science Press, Inc., New York, NY, USA, 1988.

[10] H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM Press.

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

[12] I. M. Martin. Arte, an adpative rendering and transmission environment for 3d graphics. In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, pages 413–415, New York, NY, USA, 2000. ACM Press.

[13] F. Melero, P. Cano, and J. Torres. Combining sp-octrees and impostors for multiresolution visualization. *Computer and Graphics*, 2005.

[14] G. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., 1966.

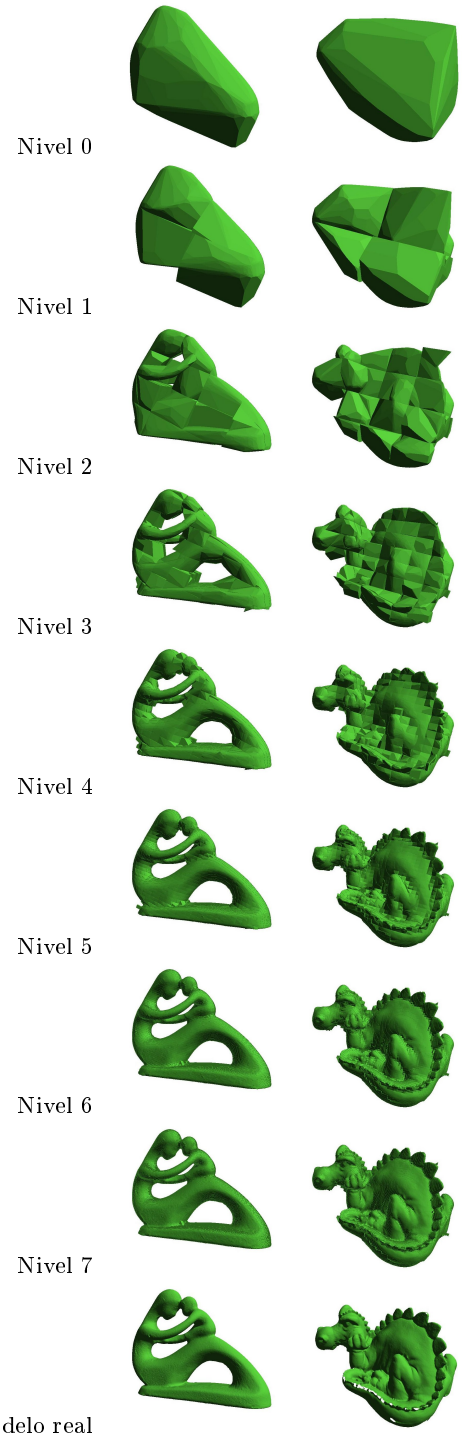

Figura 15: Transmisión progresiva de los modelos Fertility y Phlematic Dragon.