Visualización interactiva de SP-Octrees utilizando *impostores*

F.J. Melero, P. Cano, J.C. Torres
Dpto. Lenguajes y Sistemas Informáticos, Universidad de Granada
E.T.S. Ingeniería Informática
C/ Periodista Daniel Saucedo Aranda s/n
e-mail: {fjmelero, pcano, jctorres}@ugr.es

Resumen

Los árboles SP-Octrees permiten una representación multiresolución de objetos poliédricos, que se ha usado para realizar transmisión progresiva. No obstante, la visualización de los objetos en niveles de resolución intermedios presenta cierta inexactitud al aproximar los nodos internos grises por nodos llenos, generando imágenes que en situaciones son relativamente poco atractivas.
En este artículo se presenta el uso de *impostores* sobre los planos no-reales almacenados en la estructura jerárquica. De esta forma, se puede tener en todo momento una visualización realista del objeto que se desea representar con un nivel de detalle variable. En cada momento, el *impostor* utilizado depende del punto de vista del observador. Cada SP-Octree tiene asociado un conjunto de *impostores*, que son vistas del sólido complejo modelado en la estructura del árbol.

Palabras clave: Visualización adaptativa, SP-Octree, *impostores*, LOD, texturas, realidad virtual.

1. Introducción

La visualización interactiva de modelos en 3D es tradicionalmente una de las aplicaciones más importantes de la informática gráfica. Su aplicación abarca disciplinas tan dispares como la medicina, la arquitectura, la arqueología, etc... La visualización interactiva de modelos tridimensionales complejos exigen un enorme esfuerzo por parte del hardware gráfico, debido al gran número de polígonos necesarios para representar la geometría al nivel de detalle y realismo requerido. Esta gran necesidad de recursos conlleva inexorablemente una pérdida de interactividad.

Este conflicto entre el nivel de detalle representado y la velocidad de visualización

ha motivado a la comunidad científica a encontrar diversas soluciones que permitan conciliar ambos objetivos.

1.1. Nivel de detalle (LOD)

Bajo el concepto de nivel-de-detalle (LOD, *level-of-detail*), se encuentran los trabajos encaminados a presentar de una forma más simple una geometría compleja que por su distancia al observador no necesita ser representada al máximo nivel de exactitud.

Las técnicas de LOD se suelen clasificar en: *discretas* [1], donde el objeto tiene varias instancias, cada una de ellas a un distinto nivel de detalle; *progresivas* [2], en las que el nivel de detalle se extrae de una única estructura de datos en tiempo de ejecución; y *dependientes del observador* [3], que son una extensión de las progresivas, de manera que el nivel de detalle no se adquiere por igual en todo el modelo, sino que se es un nivel de detalle *anisotrópico* en función de la posición y la dirección del observador.

Aunque el concepto de LOD no está circunscrito a una representación concreta del modelo, se suele aplicar tradicionalmente a la simplificación de mallas poligonales, mayoritariamente triangulares.

Otra opción para representar sólidos y volúmenes con nivel de detalle variable es usar esquemas basados en la descomposición del espacio, y que utilizan estructuras jerárquicas para almacenar el modelo. Entre estos métodos nos podemos encontrar la Partición Binaria del Espacio (*Binary Space Partition, BSP*) y los *Octrees* y extensiones a los mismos (Extended-Octrees, PM-Octrees, etc.).

Los árboles binarios de partición del espacio (*BSP-Trees*) dividen recursivamente el espacio en dos semiespacios separados por un plano con cualquier orientación y posición. Inicialmente creados para mejorar el proceso de eliminación de partes ocultas [4], también se han utilizado para re-presentar objetos poliédricos de forma exacta [5].

Un Octree es la representación de un modelo mediante una estructura de árbol octal obtenida mediante divisiones recursivas de la caja envolvente del objeto a representar [6, 7, 10].

1.2. Uso de *impostores*

De forma opuesta, aunque complementaria, a estas técnicas multiresolución basadas en la geometría, se encuentran las técnicas de visualización basadas en imágenes. La idea que subyace bajo todas las técnicas es la sustitución de una geometría compleja por una imagen 2D de aquello que debería estar representado. Esta imagen es lo que se denomina un *impostor*.

Se han presentado muchas técnicas para representar sólidos basándose en imágenes [13, 14, 15, 16, 17]. Todas ellas pueden considerarse dependientes del observador, ya que en todos los casos, en función del punto de vista en un instante, se selecciona una imagen entre las tomadas del modelo real desde posiciones bien conocidas. La

complejidad del modelo geométrico sobre el que se aplican estas imágenes a modo de textura varía desde un simple plano hasta mallas más o menos complejas.

2. SP-Octrees

El Octree clásico presenta un inconveniente muy importante: las representaciones obtenidas con este esquema son aproximadas y para reducir el error cometido en esa aproximación, debemos definir un nivel de profundidad del árbol muy grande, con el consiguiente coste de almacenamiento.

Para solventar dicho inconveniente, se han presentado diversas extensiones al esquema de representación Octree, realizando las modificaciones del esquema en tres puntos distintos de la representación:

a) Alterar los planos de corte utilizados en el proceso de subdivisión [8].

b) Modificar la información contenida en los nodos hoja del árbol octal que representa el objeto. Para ello, podemos añadir información sobre la parte de la frontera del objeto que aparece en cada nodo terminal. De esta forma, podemos representar de forma exacta determinados objetos. Pero esa información a veces se repite en nodos terminales vecinos, como ocurre en los Octrees Extendidos.

c) Modificar la información almacenada en los nodos internos del árbol.

Basándonos en la modificación del esquema Octree clásico tanto en los nodos terminales utilizados como en la información almacenada en los nodos internos del árbol, se ha diseñado un nuevo esquema jerárquico de representación de sólidos denominado **SP-Octrees** (de *Space Partition Octrees*) [9],

Al incluir también en los nodos internos información de la frontera que defina parcialmente, en ese nivel, el objeto representado en cada nodo, la información de ciertas caras frontera podrán aparecer en los niveles superiores del árbol y no será necesario repetir información en nodos terminales vecinos que comparten las caras almacenadas en sus ancestros. Esto permitirá poder obtener información de la frontera del sólido representado en niveles superiores del árbol, lo que acelerará determinadas operaciones sobre el modelo.

2.1. Tipos de nodos

Utilizando la misma estructura de árbol octal que en Octrees clásicos y en Octrees Extendidos, definimos un conjunto de tipos de nodos que van a permitir incluir parte de la información frontera, tanto en los nodos terminales como en los no terminales.

Básicamente, la idea es clasificar cada nodo en función de las características (convexidad, concavidad) de los planos que aparecen en él. Además, en los nodos internos se almacenará la información de los planos cuya configuración se mantiene igual en sus nodos hijos, por lo que no es necesario almacenarlos en ellos, sino que se guardan

sólo en los nodos padre. Esta información permite acotar la zona del espacio en que está definido el sólido dentro del nodo.

Cuando un nodo está completamente dentro o fuera del sólido representado lo clasificamos como **negro** o **blanco**, respectivamente, de la misma forma que en Octrees clásicos (Figura 1).

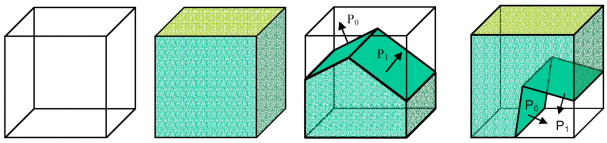


Figura 1: Nodos blanco, negro, convexo y cóncavo.

Cuando la intersección del sólido con un nodo del árbol sea convexa, utilizaremos un nodo **convexo** (Figura 1). En este caso, incluiremos en el nodo *el conjunto de planos* que definen los semiespacios cuya intersección forma el objeto convexo.

Si la intersección del nodo y el sólido representado es cóncava, definiremos un nodo **cóncavo**. Podemos definir el nodo *cóncavo* como la diferencia de la caja englobante del nodo con la intersección de los semiespacios incluidos en él, invirtiendo la orientación (la normal) de los planos que los definen.

Cuando dentro de un nodo existen concavidades y convexidades a la vez, lo clasificamos como **gris**, dividiendo el nodo en ocho octantes iguales de la misma forma que en Octrees clásicos, pero manteniendo en el nodo información de los planos que forman parte de la envolvente convexa de la parte de sólido representada. De esta forma, un nodo *gris* lo podemos ver como un nodo *convexo* con hijos (figura 2).

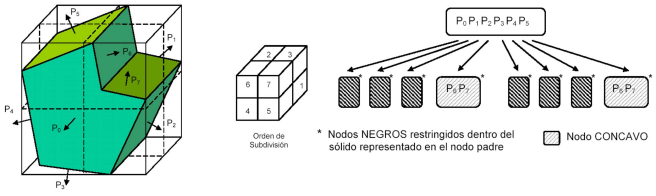


Figura 2: Nodo gris, orden establecido para los hijos y árbol correspondiente.

La información de los planos que mantenemos en los nodos grises no se repite en los distintos nodos hoja que los compartirían, sino que en ellos sólo necesitamos representar los planos frontera que no estén en esa envolvente convexa y que forman las concavidades del sólido existentes dentro del nodo.

Con este criterio de clasificación de los planos que aparecen en un nodo, nos podemos encontrar con situaciones en las que no sea posible clasificar el nodo como ninguno de los tipos hasta ahora definidos [9].

Para resolver este problema y poder representar de forma exacta cualquier sólido poliédrico necesitamos recurrir a un nuevo tipo de nodo terminal que permita representar este tipo de situaciones: el nodo **vértice**.

Un nodo vértice es aquel que contiene un único vértice del sólido representado en el que convergen tanto aristas cóncavas como convexas, y en el que al subdividirlo con el proceso definido para los nodos grises, se repite la misma configuración geométrica en el hijo que contiene el vértice (figura 3).

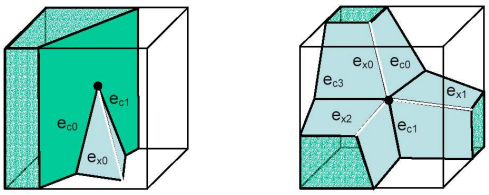


Figura 3: Dos nodos vertice

En los nodos vértice almacenaremos, además del vértice y la información de los planos que soportan las caras que lo comparten, la configuración de cada una de las aristas que convergen en él (cóncavas o convexas).

Con estos tipos de nodos mantenemos la misma expresividad que con los Octrees Extendidos [11] ya que los nodos terminales utilizados en estos pueden ser traducidos directamente a los tipos de nodos definidos en nuestro esquema.

2.2. Representación interna

Para representar internamente el esquema propuesto utilizamos las siguientes estructuras de datos, de forma similar a como se utilizan en los Octrees Extendidos:

Una matriz con las ecuaciones de los planos, $ax + by + cz + d = 0$, que definen los semiespacios utilizados en cada nodo para representar el sólido. La normal a dichos planos se utilizará para distinguir el interior y el exterior de cada semiespacio. Definimos el interior de un semiespacio como el conjunto de puntos (x, y, z) que satisfacen la desigualdad $ax + by + cz + d < 0$.

La codificación del árbol octal donde:

a) En los nodos terminales, almacenamos el tipo de nodo, el número de planos incluidos en él y una referencia a cada uno de ellos.

   En el caso del nodo vértice almacenaremos también las coordenadas del vértice y la configuración de las aristas que convergen en él.

b) En los nodos internos almacenamos el número de planos que están en ese nodo y la referencia a la posición de los mismos en la matriz auxiliar. Además, almacenaremos un enlace a los ocho hijos.

Los nodos hijos pueden contener la información de los planos que forman concavidades en el nodo padre, y podrán ser tanto nodos terminales como internos.

Como elementos de la matriz de planos tenemos vectores de cuatro flotantes que almacenan los coeficientes de cada plano. Además, esta matriz permite evitar duplicados, de forma que si varias caras frontera del sólido pertenecen al mismo plano, sólo almacenamos su ecuación una única vez.

Al almacenar en cada nodo sólo las referencias a los planos de la matriz auxiliar, y al no almacenar internamente la geometría del objeto representado sino sólo las ecuaciones de los planos frontera, la representación obtenida es compacta y reduce las necesidades de almacenamiento

Esta estructura nos permite además almacenar varios modelos a la vez, de forma que si comparten planos frontera no es necesario repetirlos, sino que aparecen una única vez en la matriz auxiliar de planos.

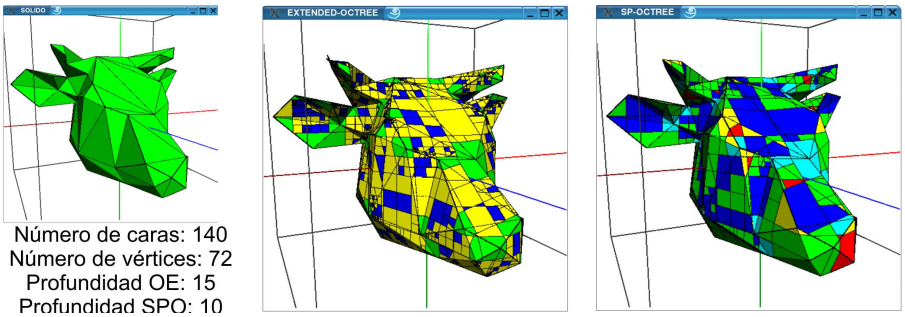


Figura 4: De izda a dcha: sólido original, Octree-extendido, SP-Octree.

Color	Usado en...
Verde	Planos de nodos convexos o ancestros de estos
Azul	Planos de nodos cóncavos
Amarillo	Planos en ancestros de un nodo cóncavo
Rojo	Planos en ancestros de nodos convexos sin planos
Celeste	Planos en nodos vértice
Morado	Nodos grises de último nivel

Cuadro 1: Colores utilizados en la representación de los Octrees

En la figura 4 mostramos un sólido de 140 polígonos representado con un Extended Octree de profundidad 14 y el mismo nivel de detalle con un SP-Octree de profundidad 10. En la tabla 1 encontramos qué representa cada uno de los colores.

2.3. Visualización del SP-Octree

Una de las ventajas de los Octrees Clásicos es el orden implicito que se puede establecer entre los nodos, lo que facilita el proceso de visualización del modelo definiendo el orden de representación de los nodos. En el esquema que proponemos se mantiene ese orden implícito entre los nodos del árbol.

Con esta idea, y siguiendo el mismo algoritmo de visualización que en Octrees Clásicos, para visualizar un modelo en el esquema SP-Octree recorremos los nodos del árbol, representando para cada nodo la información geométrica almacenada en él.

Al no almacenar información geométrica de la frontera explícitamente en los nodos, al igual que ocurre con los Octrees Extendidos, el proceso de visualización de cada nodo es similar al realizado para representarlo en OE.

Al visualizar un nodo, debemos recortar cada plano con la propia caja englobante del nodo y con los otros planos pertenecientes al mismo. Además, también debemos tener en cuenta los planos que aparece en sus ancestros y que, por tanto, forman parte de la envolvente del sólido. Por ello, también debemos recortar con esos planos el resultado obtenido anteriormente para obtener el polígono final.

El hecho de mantener información de parte de la frontera en los nodos internos permite realizar una visualización aproximada del sólido, simplemente limitando el nivel de profundidad máximo que queremos representar, haciendo que los nodos grises de ese nivel se visualicen como si se trataran de nodos convexos, representándolos por tanto de forma aproximada.

2.3.1. Visualización del árbol

El algoritmo de visualización recibe como entrada el nodo raíz de un SP-Octree, el nivel máximo a visualizar y una lista de planos perteneciente a los ancestros del nodo, que inicialmente estará vacía. Si ese nodo raíz del árbol es un nodo terminal (deberá ser un nodo convexo), se llama directamente a la función encargada de visualizar un nodo terminal.

Si el nodo es gris, añadimos los planos que aparecen en el nodo en la lista de planos ancestros y procedemos a llamar a la función de forma recursiva para cada uno de los nodos hijo.

Se puede hacer una visualización aproximada dibujando los nodos grises como si se tratasen de nodos convexos.

2.3.2. Visualización de un nodo terminal

Para cada plano en el nodo obtenemos el polígono resultante al recortarlos con la caja englobante y los planos convexos de los ancestros.

Para obtener el polígono a visualizar debemos recortar el resultado obtenido en función del tipo de nodo en el que nos encontremos:

- Si el nodo es convexo, recortamos dicho polígono con el resto de planos del nodo.
- Si el nodo es cóncavo, debemos recortarlo con el resto de planos pero con la normal invertida para quedarnos con la zona del recorte exterior a esos planos, ya que forman concavidad con el polígono en cuestión.
- Si el nodo es vértice, recortamos el polígono con los planos vecinos, teniendo en cuenta la configuración (cóncava o convexa) de la arista que los une.

Con esto obtenemos el polígono resultante a visualizar para cada plano en el nodo. Aplicando un mecanismo de eliminación de caras ocultas entre todos los polígonos del nodo, obtenemos la correcta visualización del modelo.

3. Aplicación de *impostores*

La proyección de una textura (*projective texture mapping, PTM*) fue propuesta por Segal [12], y es parte del estándar de OpenGL [19]. Aunque en el artículo original se usaba solamente para acelerar la generación de sombras y otros efectos de iluminación, esta técnica es directamente aplicable a la visualización basada en imágenes, ya que puede simular el proceso inverso de tomar fotografías con una cámara, es decir, permite proyectar imágenes sobre la escena como si fuese un cañón proyector.

Para realizar la proyección de la textura, el usuario especifica la posición y orientación del proyector, así como un plano virtual donde se proyecta la imagen. Con estos datos, y las matrices de transformación y proyección de la escena actual, se calculan las coordenadas de textura adecuadas.

De hecho, las transformaciones que se realizan sobre los vértices para obtener sus coordenadas de textura son muy similares a las que se realizan sobre los propios vértices durante el proceso de visualización al pasarlos de coordenadas mundiales a coordenadas de imagen, tal y como se puede contemplar en la figura 5.

Como se ha comentado anteriormente, OpenGL soporta desde su versión 1.0 el *PTM*, y la clave de todo el proceso se encuentra en la función de generación automática de coordenadas de texturas, *texgen*. De esta forma, para proyectar en OpenGL una textura sobre un modelo (o parte de él), tan sólo es necesario controlar las matrices *modelview* y *perspective* del proyector de la textura y del observador de la escena. Con estos datos, y en función del modo de generación de las texturas, se combinan dichas matrices para obtener la matriz de textura, que es la que dirige las transformaciones que se aplican sobre los vértices para obtener las coordenadas de textura. Dicho proceso se puede reproducir fácilmente siguiendo lo dispuesto en [18, 19]

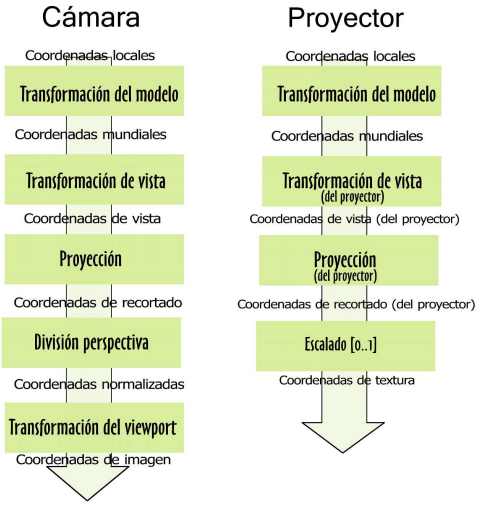


Figura 5: Transformaciones que se aplican a los vértices de la geometría (para su visualización y la obtención de coordenadas de textura).

3.1. Generación de *impostores*

La generación del conjunto de imágenes a utilizar como *impostores* se realiza una única vez por objeto a representar, y se obtienen desde diversos puntos de la esfera que contiene a la caja englobante del sólido. Las imágenes se han tomado renderizando el objeto a su máxima resolución, y colocando la cámara en los vértices que se obtienen al formar una esfera a partir de un tetraedro. Dicha esfera puede construirse a distinto nivel de detalle, lo que por tanto permite generar más o menos vistas del objeto.

Otra posibilidad hubiera sido realizar una determinación de los puntos de visibilidad de cada plano almacenado en la estructura del SP-Octree y realizar un muestreo de imágenes para cada uno de ellos desde dichos lugares, tal y como se realiza en [20]. Sin embargo, consideramos que esto conlleva un aumento considerable de las necesidades de memoria, y un aumento en el tiempo de cálculo, ya que cada plano tiene su propia textura y la operación de intercambio de imágenes consume cierto tiempo.

Variando la resolución de la esfera (y por tanto el número de imágenes utilizadas), y comprobando la calidad de la visualización interactiva del modelo tridimensional en cada caso, observamos que con un nivel 4 de detalle (fig. 6), en el que se generan 258 vértices, es suficiente para que la transición de una textura a otra se pueda realizar con suavidad, incluso sin realizar fundido de texturas durante la transición entre ellas.

Dados los vértices, se coloca la cámara en dicho punto, orientada hacia el centro de la esfera en todo momento. No es necesario controlar el sentido del vector *up*, ya

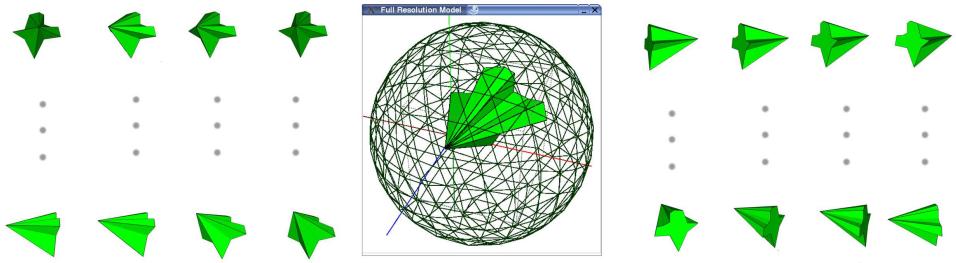


Figura 6: Esfera de nivel 4 usada para tomar los *impostores* del modelo.

que toda la información acerca de la posición y orientación de la cámara se almacena en la matriz de transformación, con la que OpenGL trabaja para realizar el mapeado automático de la textura.

La proyección utilizada para tomar las imágenes es la ortográfica, para obtener una vista del objeto independiente de la distancia del plano de proyección a él. Lo único que hay que controlar es que el objeto quepa totalmente en el plano de proyección.

Todas las imágenes se almacenan en un único fichero binario, incluyendo en éste información acerca de la matriz de proyección utilizada para la toma de las imágenes, así como cada una las matrices de transformación utilizadas para posicionar la cámara en cada una de las tomas. Todos los píxeles pertenecientes al objeto tienen un valor *alpha* 0.0, mientras que los correspondientes al fondo, totalmente blancos, se almacenan con el valor *alpha* a 1.0, es decir son transparentes.

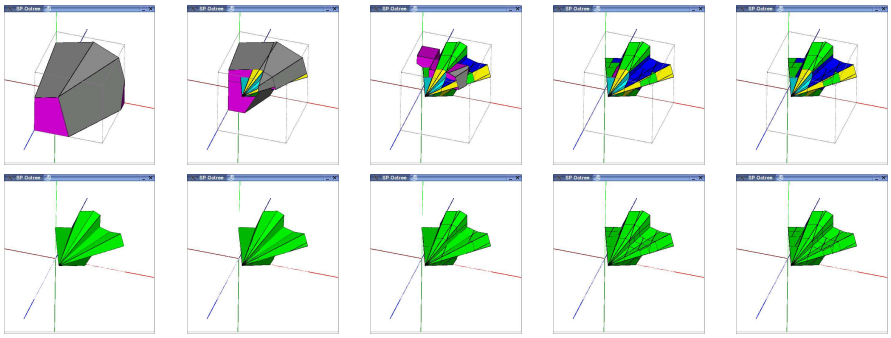
3.2. Visualización del SP-Octree con *impostores*

Para visualizar el modelo, se recorre el árbol de forma descendente, generando los polígonos correspondientes a cada nivel. Si el polígono generado no forma parte de la frontera del sólido, es decir, es un polígono de un nodo gris que se ha decidido pintar de forma aproximada, se le aplica el *impostor*; si, por el contrario, es un polígono contenido en la frontera del sólido, se le aplica su textura original.

Como para cada imagen almacenamos su matriz de transformación, es posible conocer el vector de vista utilizado cuando fue tomada. En lugar de trabajar con coordenadas cartesianas, y dado que el módulo de dicho vector es indiferente, trabajamos con las coordenadas polares (*azimut* y *zenit*) del mismo.

Más concretamente, seguimos el siguiente proceso:

1) En cada momento de la visualización, se comprueba la posición del observador con respecto al objeto. Esta posición determina el vector de vista que nos va a servir para localizar la imagen que fue tomada desde un punto de vista similar. Nótese como no tenemos en cuenta el lugar hacia donde mira el observdor, sino



(Nivel 0) (Nivel 1) (Nivel 2) (Nivel 3) (Nivel 4)

Figura 7: Visualización de un SP-Octree sin *impostores* (superior) y SP-Octree con *impostores* (inferior)

la posición relativa del mismo con respecto al objeto, ya que de no hacerse así, se cargaría un *impostor* erróneo.

2) Se busca en la tabla de *impostores* aquél que mejor se ajusta a la vista que se va a dibujar, y se toma el identificador de textura que tiene asignado.

3) Se recorre el árbol en preorden hasta el nivel deseado, generando los polígonos que corresponda en cada nodo. Si el polígono forma parte de la frontera del objeto, se dibuja con su textura original (o con el código de colores si se carece de textura). Si, por el contrario, nos encontramos ante un plano de un nodo gris no perteneciente a la frontera, se le aplica el impostor seleccionado previamente.

4. Resultados

La estructura de SP-Octree presenta una fácil visualización por niveles de detalle del sólido que representa, y suponen un ahorro en espacio con respecto a otras estructuras jerárquicas.

Este trabajo supone un aumento en la calidad de visualización de los modelos representados mediante SP-Octrees, ya que para tener una imagen del objeto real, no es necesario descender totalmente en la jerarquía, sino que se puede obtener desde el principio.

En la figura 7 se puede observar cómo la selección del impostor adecuado en función de la posición relativa del observador con respecto al objeto hace que lo observado se ajuste al modelo final, y en todo momento, la sensación es estar trabajando con el objeto al máximo nivel de detalle.

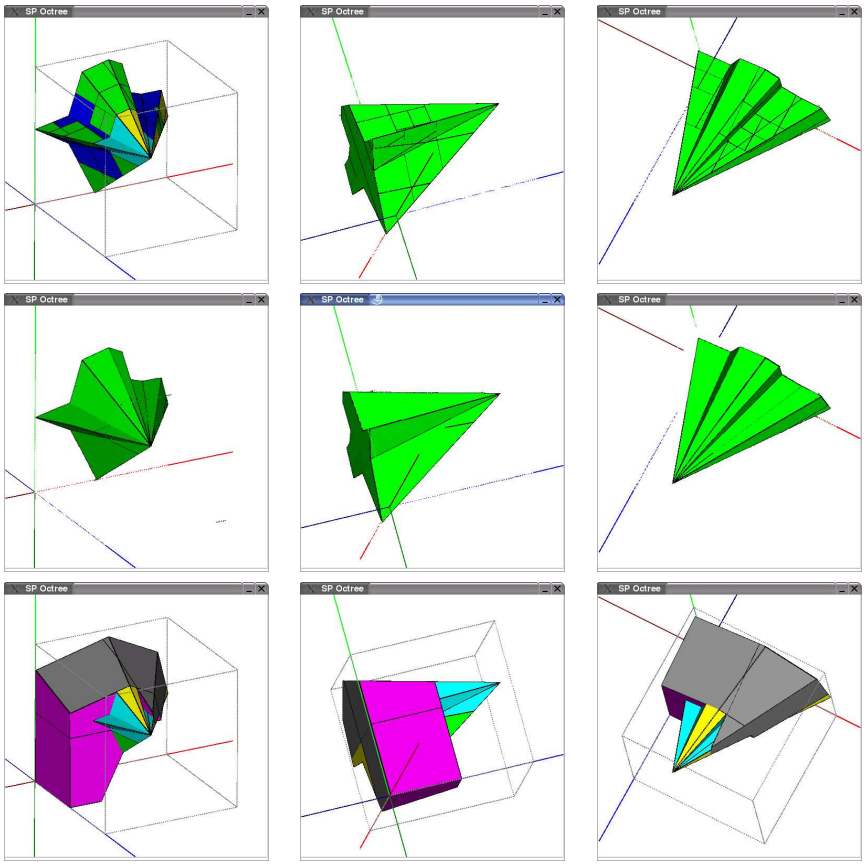


Figura 8: Distintas vistas de un SP-Octree clásico en el último nivel (superior), con *impostores* en el nivel 1 (centro), y sin usar *impostores* en el nivel 1 (inferior)

En la secuencia 8 podemos apreciar como desde distintos puntos de vista, en cada momento se selecciona el *impostor* más adecuado.

5. Conclusiones y trabajos futuros

El uso de *impostores* junto con la estructura jerárquica de SP-Octrees permite el uso de ésta última en entornos interactivos complejos, ya que la visualización de los modelos en todo momento será como si se trabajara con el modelo real.

Como se ha comentado anteriormente, los SP-Octrees permiten a su vez el anida-

miento de estructuras jerárquicas, lo que permite construir las escenas complejas a distinto nivel de detalle, en función de la posición del observador en la misma.

Además, al poder utilizar la estructura de SP-Octrees para transmisión progresiva [21], es posible que el refinamiento de las estructuras se haga en función del observador de las escena, obviando las partes ocultas de los objetos y utilizando el mínimo nivel de detalle geométrico en aquellos que se encuentren más alejados.

En el futuro se prevé realizar el cálculo de oclusores haciendo uso de la estructura espacial del octree como índice, permitir la navegación por el interior del octree, añadiendo nuevos *impostores* en el interior de la jerarquía y aplicar el esquema propuesto a representaciones virtuales de yacimientos arqueológicos y modelos médicos.

6. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología y los fondos FEDER, dentro del proyecto TIC2001-2099-C03-02.

Referencias

[1] Clark, J.H. "Hierarchical Geometic Models for Visible Surface Algorithms", Communications of the ACM, vol. 19(10), pp 547-554, 1976

[2] Hoppe, H. "Progressive Meshes". Proceedings of SIGGRAPH 96, pp. 99-108, 1996

[3] Hoppe, H. "View-Dependent Refinement of Progressive Meshes". Proceedings of SIGGRAPH 97, ACM SIGGRAPH 1997

[4] Fuchs H., Kedem Z., Naylor. B, "On visible Surface Generation by a priori Tree Structures", ACM Computer Graphics 17(3), pp 65-72, 1983

[5] Tibhault W.C., Naylor B.. "Set Operations on Polyhedra Using Binary Space Partitioning Trees", ACM Computer Graphics 21(4), pp 153-162, 1987

[6] Megher, D. "Geometric modelling using octree encoding", Computer Graphics and Image Processing, 19 (2), pp 129-147, 1982

[7] Fujimura, K., Yamaguchi, K; Kunii, T.: "Octree-related data structures and algorithms", IEEE Computer Graphics and Applications, 4, pp 53-59, 1984

[8] Cano P., Torres J.C., "Octrees Modificados: una extensión a Octrees clásicos", II Jornadas Andalusíes de Informática Gráfica, pp 1-19, Sevilla, 200

[9] Cano P., Torres J.C., "Representation of Polyhedral Objects using SP-Octrees", Journal of WSCG, vol 10(1), pp 95-101, 2002

[10] Jackins C.H., Tanimoto S.L., "Octrees and their use in representing three-dimensional objects",Computer Graphics and Image Processing, 14, pp 249-270, 1980

[11] Brunet P. Navazo I., "Solid Representation and Operation using Extended Octrees", ACM Transactions on Graphics, Vol. 9, n.º 2, pp 170-197,1990

[12] Segal M. et al.. "Fast shadows and lighting effects using texture mapping", En Proceedings of SIGGRAPH'92, pp. 249-252, 1992

[13] Grossman J.P. and Dally W.J., "Point sample Rendering", 9th eurographics Workshop on Rendering, 1998, pp 181-192.

[14] Maciel P.W.C an Shirley P., "Visual navigation of large environments using textured clusters". En Pat Hanrahan y Jim Winget, editores, 1995 Symposium on Interactive 3D Graphics, pp. 95-102, ACM SIGGRAPH, 1995

[15] Aliaga D.G., "Visualization of complex models using dynamic texture-based simplification", En IEEE Visualization'96.IEEE, Octubre 1996

[16] Sillion F., Drettakis G. and Bodelet B., "Efficient impostor manipulation for real-time visualization of urban scenery", En D. Fellner y L. Szirmay-Kalos, editores, Computer Graphics Forum (Proc. of Eurographics'97), volume 16, Budapest, Hungría, 1997

[17] Decoret X., Schaufler G, Sillion F. and Dorsey J, "Multi-layered impostors for accelerated rendering", En P. Brunet and R. Scopigno, editores, Computer Graphics Forum (Proc. of Eurographics'99), volume 18(3), 1999

[18] Everitt C., "Projective Texture Mapping", from developer.nvidia.com

[19] Segal M., Akeley K. "The OpenGL Graphics System: A Specification (Version 1.2.1)". http://www.opengl.org

[20] Debevec P., Yu Y. and Borshukov G., "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping", Eurographics Workshop on Rendering 1998, pp 105-116, June 1998

[21] Cano P., Torres J.C., Velasco F., "Progressive transmission of Polyhedral Solids using a Hierarchical Representation Scheme", Journal of WSCG, vol 11(1), pp 95-101, 2003