Representación de Modelos Multirresolución con Abanicos de Triángulos*

J. Ribelles, A. López, I. Remolar, O. Belmonte, M. Chover
Departamento de Informática
Universitat Jaume I. 12080. Castellón
e-mail: ribelles@uji.es

Resumen

El modelado multirresolución de superficies poligonales se ha presentado como solución para la visualización interactiva de escenas formadas por cientos de miles de polígonos. Sin embargo, estos modelos no aprovechan las ventajas que supone el uso de tiras o abanicos de triángulos, y que ofrecen una disminución del tiempo de visualización y un importante ahorro de memoria. Juntar ambas técnicas es el principal objetivo que se persigue en este trabajo. Se presenta una nueva representación que, utilizando el esquema Mallas Ordenadas Multirresolución (MOM), permite almacenar y visualizar cada nivel de detalle como un conjunto de abanicos de triángulos. Se estudian las ventajas e inconvenientes de esta nueva aproximación utilizando diferentes modelos y comparándola con la representación previa de MOM.

Palabras clave: modelado multirresolución, abanico de triángulos, nivel de detalle.

1 Introducción

Uno de los principales objetivos del modelado multirresolución [1] es conseguir visualizar de forma interactiva objetos formados por cientos de miles de polígonos. Un modelo multirresolución almacena un amplio rango de aproximaciones o niveles de detalle (lod, *level of detail*) y permite, ante una petición, recuperar de forma eficiente el lod requerido.

Por otra parte, han sido desarrollados diversos mecanismos para acelerar el proceso de visualización de modelos poligonales. Por ejemplo, las tiras o los abanicos de

* Trabajo financiado por el proyecto TIC1999-0510-C02-02 (CICYT, Ministerio de Educación y Ciencia)

triángulos aparecen como primitivas de dibujo en librerías gráficas como *OpenGL*. Este tipo de representaciones facilita una visualización más rápida. Para dibujar un abanico de *n* triángulos, por ejemplo, hay que transmitir al procesador gráfico *n+2* vértices en lugar de *3n*. Esto nos lleva a un importante ahorro de memoria y tiempo de computación al tener que operar sobre un número menor de vértices.

Resulta sencillo hacer uso de las tiras y abanicos de triángulos cuando se trabaja con modelos estáticos. Sin embargo, en un modelo multirresolución, la conectividad de la superficie cambia cada vez que lo hace el nivel de detalle, probablemente a cada cuadro, obligando a que las tiras o abanicos hayan de generarse de forma dinámica.

En este artículo presentamos un nuevo esquema para almacenar y extraer directamente un nivel de detalle de un modelo multirresolución como un conjunto de abanicos de triángulos. Para ello, nos basamos en el modelo Mallas Ordenadas Multirresolución (MOM) [2]. El nuevo esquema, que denominamos MOM-FAN, define una nueva estructura de datos, cuya unidad básica de representación es el abanico de triángulos, y un algoritmo de recorrido que determina los abanicos que pertenecen al lod requerido.

2 Trabajo previo

Hoppe, [3] presenta el modelo *View-dependent of Progressive Meshes*, *VDPM*, basado en una estructura jerárquica de vértices construida a partir de una secuencia de contracciones de aristas. El nivel de detalle se determina a partir de una serie de parámetros como son la iluminación, el volumen de la vista, etc. El cambio de estas condiciones origina que también lo haga el lod requerido, y propone calcular tiras de triángulos una vez determinados los triángulos que pertenezcan a él (figura 1).

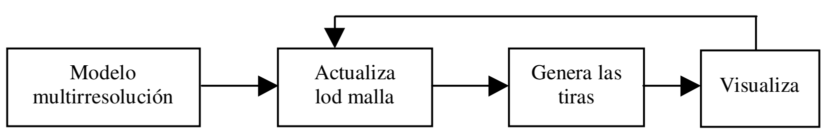

Figura 1. Proceso de visualización de un lod en VDPM.

El-Sana et al. [4] presentan una estructura de datos, *Skip-Strips*, que mantiene tiras de triángulos aunque cambie el nivel de detalle. Un *Skip-Strips* se construye en tiempo de ejecución a partir del modelo multirresolución. Al mismo tiempo se obtienen tiras de triángulos a partir del modelo original. Cada vez que cambia el nivel de detalle, se actualiza la estructura *Skip-Strips* basándose en el lod requerido, lo que permite actualizar las tiras, y finalmente visualizarlo (figura 2). Como la longitud de las tiras obtenidas son generalmente cortas, propone añadir una etapa de concatenación de tiras previa a la visualización.

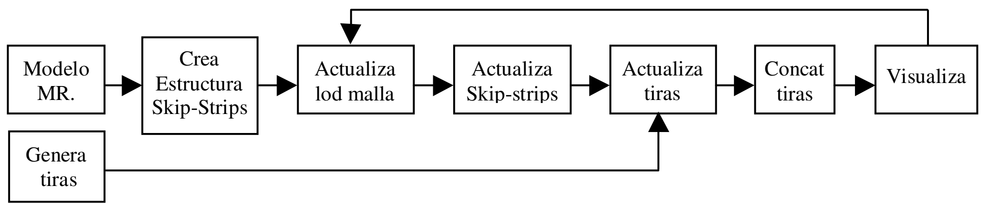

Figura 2. Proceso de visualización de un lod en Skip-Strips.

Nuestra propuesta simplifica el esquema para visualizar un nivel de detalle. El propio modelo multirresolución se codifica mediante abanicos de triángulos, siendo sólo necesario realizar una pequeña adecuación del abanico según el lod requerido (figura 3). En la figura 4 se muestran tres lods de un objeto MOM-FAN, donde se han pintado del mismo color los triángulos que forman cada abanico.

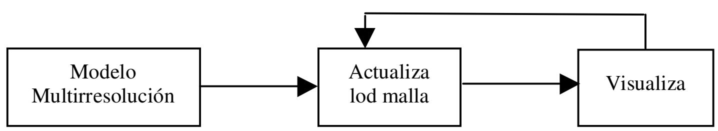

Figura 3. Proceso de visualización de un lod en MOM-FAN.

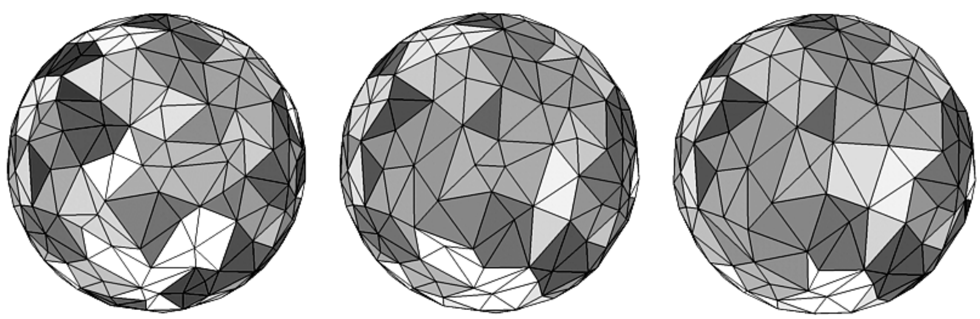

Figura 4. Tres niveles de detalle visualizados con abanicos.

2.1 Revisión de Mallas Ordenadas Multirresolución

Mallas Ordenadas Multirresolución se presentó con el objetivo de aumentar las prestaciones en la visualización interactiva de complejos modelos poligonales. Posteriormente, se extendió para explotar la propiedad de la coherencia cuadro a cuadro en [5]. Esto permitió acelerar el proceso de recuperación del lod requerido, pero no disminuir el tiempo de visualización. Finalmente, en [6] se presentó una comparación de MOM con Progressive Meshes [7].

Sean M y M^r la malla original y su representación multirresolución, respectivamente. M^r almacena explícitamente los vértices, V^r, y los triángulos, F^r, utilizados para representar cualquier resolución:

$$M^r = \left\{ V^r, F^r \right\} \tag{1}$$

Para construir M^r con n niveles de detalle, se aplica $n\text{-}1$ veces un método de simplificación iterativamente. Cada simplificación S_i, $0 \le i < n\text{-}1$, produce un nuevo nivel de detalle M_{i+1} y puede representarse como la tupla $S_i = \{V_i, F_i, V'_i, F'_i\}$ donde V_i y F_i son los conjuntos de vértices y triángulos que son eliminados de M_i, y V'_i y F'_i son los conjuntos de vértices y triángulos que son añadidos a M_i generando, finalmente, M_{i+1}. Por tanto, podemos expresar el objeto resultante, M_{i+1}, como:

$$M_{i+1} = \left(M_i - \left\{ V_i, F_i \right\} \right) \cup \left\{ V'_i, F'_i \right\} \quad 0 \le i < n-1 \tag{2}$$

Dado que M^r almacena todos los vértices y caras que pueden usarse en la representación de cualquier nivel de detalle, M^r puede definirse como:

$$M^r = \bigcup_{i=0}^{n-1} M_i \quad n \ge 1 \tag{3}$$

De las ecuaciones (1)-(3), se obtiene que M^r se puede expresar como la malla inicial, $M=M_0$, más todos los vértices y triángulos generados en cada iteración del proceso de simplificación o, también, como la malla correspondiente al peor nivel de detalle, M_{n-1}, más todos los vértices y triángulos eliminados en cada iteración del proceso de simplificación:

$$V^r = V_0 \cup V_1 \cup \ldots \cup V_{n-2} \cup V_{n-1} = V_{n-1} \cup \bigcup_{i=0}^{n-2} V_i \tag{4}$$

$$F^r = F_0 \cup F_1 \cup \ldots \cup F_{n-2} \cup F_{n-1} = F_{n-1} \cup \bigcup_{i=0}^{n-2} F_i \tag{5}$$

La idea principal de MOM se basa en las expresiones mostradas en estas dos últimas ecuaciones (4) – (5). Esto es, almacenar de forma ordenada las secuencias de vértices y triángulos eliminados, V_i y F_i, $0 \le i < n-2$, más los vértices y triángulos correspondientes al peor nivel de detalle $M_{n-1}=\{V_{n-1}, F_{n-1}\}$. MOM-FAN se basa en la misma idea básica, con la diferencia de que se utilizan abanicos de triángulos en lugar de triángulos aislados. En la sección 3 se muestra cómo almacenarlos en la estructura de datos, y en la sección 4 cómo extraer los abanicos que forman un nivel de detalle dado.

3 Estructura de datos

La estructura de datos que se presenta está basada en una lista de listas similar a la que se describe en MOM. La diferencia fundamental es la lista de triángulos, que ahora se sustituye por una lista de vértices que almacena secuencias de abanicos.

```
struct Vertice {
  float coordenada[3];
  int primerVerticeAbanico;
};

struct VerticeAbanico {
  int id;
  struct Vertice *v;
};

struct ControlLOD {
  int verticesEliminados;
  int triangulosGenerados;
};

struct MOM-FAN {
  int nVertices, nVerticesAbanico, nLods;
  struct Vertice listaVertices[];
  struct VerticeAbanico listaVerticesAbanicos[];
  struct ControlLOD listaControlLOD[];
};
```

Figura 5. Estructura de datos de MOM-FAN.

En la figura 5 se muestran las estructuras de datos para la representación del modelo multirresolución. Tres son las listas que lo forman:

- *Lista de vértices.* Almacena todos los vértices de la malla de forma ordenada según se han ido eliminando en el proceso de simplificación. Cada uno de ellos representa el vértice inicial de un abanico.
- *Lista de abanicos.* Esta lista almacena los abanicos como secuencias de vértices (excepto el vértice inicial). Cada abanico está formado por los vértices de los triángulos que desaparecen cuando el vértice común es eliminado, y un identificador de cada uno de los triángulos. Los abanicos también se encuentran ordenados según han sido eliminados en el proceso de simplificación.

- *Lista de control.* Lista donde, para cada lod, se almacena la información necesaria que permite recuperar los datos que pertenecen a dicho lod.

3.1 Proceso de construcción

El proceso de construcción del modelo se divide en dos etapas. La primera, involucra las tareas que se han de repetir para cada iteración del proceso de simplificación. La segunda, agrupa un conjunto de operaciones que terminan de construir el modelo multirresolución M^r. Para hacer más simple la explicación se va a suponer que cada simplificación realizada supone la eliminación de un único vértice.

Inicio. Antes de empezar este proceso es necesario inicializar M^r con M_0, ya que este es el primer nivel de detalle. Para ello, es necesario actualizar la lista de control con los datos de la malla M_0, rellenando los campos *verticesEliminados=0* y *triangulosGenerados*=$|F_0|$. La lista de vértices y la de abanicos estarán vacías.

Primera etapa. Para cada iteración del proceso de simplificación hay que realizar las siguientes tareas:

1. Almacenar en *listaVerticesAbanicos* los vértices que forman el abanico excepto el vértice eliminado que será el vértice común.
2. Almacenar en *listaVertices* el vértice eliminado y hacer que apunte al primero del resto de los vértices del abanico.
3. Actualizar *listaControlLOD* con los vértices eliminados y el total de triángulos generados.

En la figura 6 se muestra la malla que se va a utilizar para ilustrar el proceso y el resultado de inicializar y completar la primera etapa eliminando el vértice v_4. El resultado de aplicar una nueva simplificación eligiendo el vértice v_1 puede verse en la figura 7.

Segunda etapa. Completar la estructura de datos (ver figura 8):

1. Añadir la malla que representa el peor lod como un conjunto de abanicos de la misma forma que se hizo en la primera etapa. Los vértices que no tengan un abanico asociado se colocan al final de la lista.
2. Actualizar los punteros de los vértices de abanicos para que apunten a los vértices de la nueva lista de vértices.

listaControlLOD

0	1
9	13

listaVertices

v_4
0

listaVerticesAbanicos

2	3	4	8	7	6	6
v_1	v_3	v_5	v_6	v_8	v_7	v_1

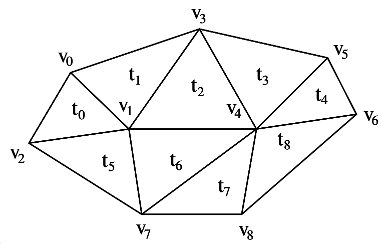

Figura 6: Malla inicial y estructura de datos tras la eliminación de v_4.

listaControlLOD

0	1	2
9	13	16

listaVertices

v_4	v_1
0	7

listaVerticesAbanicos

2	3	4	8	7	6	6	0	1	10	9	5	5
v_1	v_3	v_5	v_6	v_8	v_7	v_1	v_2	v_0	v_3	v_8	v_7	v_2

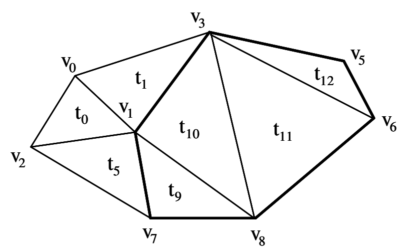

Figura 7: Resultado del proceso de construcción tras la eliminación de v_1.

listaControlLOD

0	1	2
9	13	16

listaVertices

v_4	v_1	v_3	v_0	v_2	v_5	v_6	v_7	v_8
0	7	13	18					

listaVerticesAbanicos

2	3	4	8	7	6	6	0	1	10	9	5	5	12	11	15	14	14	13	13
v_1	v_3	v_5	v_6	v_8	v_7	v_1	v_2	v_0	v_3	v_8	v_7	v_2	v_5	v_6	v_8	v_7	v_0	v_7	v_2

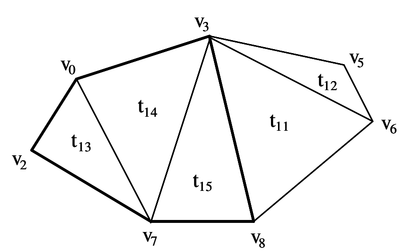

Figura 8: Resultado del proceso de construcción tras la segunda etapa.

Con esto concluye el proceso de construcción de la estructura de datos, que puede generalizarse a la eliminación de más de un vértice en el proceso de simplificación de forma similar a como se hace en [2].

3.2 Coste espacial

Suponiendo que el coste de almacenar un real, un entero y un puntero es de una palabra, y que se almacenan tres listas, una por cada uno de los tipos de datos del modelo (ver figura 5), el coste de almacenamiento será de $4|V^r| + 2|VA^r| + 2n$ palabras, siendo $|VA^r|$ el tamaño de *listaVerticesAbanicos*.

Respecto a $|V^r|$, dado que el número de vértices que se añaden en cada una de las fases del proceso de construcción siempre es menor que el de vértices que existen, $|V'_i| < |V_i|$, se puede concluir que el orden en el peor caso, será cuadrático respecto al número de vértices del modelo original, $|V_0|$. El mejor caso aparece cuando se utiliza un método de simplificación por eliminación de vértices [8], donde $|V^r| = |V_0|$, y puede llegar a $|V^r| = 2|V_0|$, en el caso de utilizar un método de simplificación por eliminación de aristas [9].

Respecto a $|VA^r|$, en el peor caso, el coste también es de orden cuadrático respecto a $|F_0|$. Si suponemos que el número de caras alrededor de un vértice es por término medio 6, se utiliza un método de simplificación basado en eliminación de vértices y que $|V_i| \approx |F_i| / 2$, $0 \leq i < n$-1, se tiene que $|VA^r| \approx 3'5\ |F_0|$. Con un método basado en eliminación de aristas, se obtiene que $|VA^r| \approx 5'5\ |F_0|$.

Para concluir podemos comparar el coste de almacenamiento del modelo con abanicos y MOM (ver tabla 1).

	Coste de almacenamiento	Eliminación vértices	Eliminación aristas
MOM	$4\|V^r\|+3\|F^r\|+2n$	$\|V^r\| = \|V_0\|$ $\|F^r\| \approx 3\ \|F_0\|$	$\|V^r\| = 2\|V_0\|$ $\|F^r\| \approx 5\|F_0\|$
MOM-FAN	$4\|V^r\|+2\|VA^r\|+2n$	$\|V^r\| = \|V_0\|$ $\|VA^r\| \approx 3'5\ \|F_0\|$	$\|V^r\| = 2\|V_0\|$ $\|VA^r\| \approx 5'5\ \|F_0\|$

Tabla 1. Resumen de los costes teóricos de almacenamiento.

4 Algoritmo de visualización de un lod

Suponga que se desea visualizar el lod de mayor detalle del ejemplo de las figuras 6-8. En este caso, deben visualizarse los triángulos que aparecen en la malla de la figura 6. Los abanicos que se deben generar, ordenados según la aparición de los vértices en la estructura de datos son:

- (v_4 v_1 v_3 v_5 v_6 v_8 v_7 v_1) que corresponden a los triángulos (t_2 t_3 t_4 t_8 t_7 t_6).
- (v_1 v_2 v_0 v_3) y (v_1 v_7 v_2), que corresponden a los triángulos (t_0 t_1) y (t_5), respectivamente.

El primer vértice del abanico siempre corresponde al vértice eliminado. El resto de vértices están almacenados en *listaVerticesAbanicos*, y asociados al vértice eliminado. Sin embargo, puede ocurrir que algunos vértices no pertenezcan al lod requerido. En el ejemplo, debido a que t_{10} y t_9, implícitamente asociados a v_1, no se deben visualizar, hay un salto desde el vértice v_3 al vértice v_7 (ver figura 7). Para resolver estos saltos sin tener que partir los abanicos es necesario introducir el vértice v_1 entre v_3 y v_7, de modo que se producen dos triángulos degenerados. Así, mientras el primer abanico se resuelve sin triángulos degenerados (v_4 v_1 v_3 v_5 v_6 v_8 v_7 v_1), el segundo abanico (v_1 v_2 v_0 v_3 v_1 v_7 v_2) incluye dos, que son (v_1 v_3 v_1) y (v_1 v_1 v_7).

El algoritmo de visualización de un lod, que se muestra en la figura 9, requiere calcular inicialmente el número de triángulos a visualizar en ese lod. Después, recorre la estructura de datos recuperando los abanicos de triángulos que pertenecen al lod requerido.

4.1 Cálculo del número de triángulos de un lod.

Dado un lod, en la lista de control tenemos almacenados el número de vértices que no pertenecen a dicho lod, *VE*, y el número de triángulos generados, *TG*. A partir de estos datos es fácil obtener cuántos triángulos pertenecen al lod requerido:

```
VE= listaControlLOD[lod].verticesEliminados
TG= listaControlLOD[lod].triangulosGenerados
Número de triángulos= TG- (listaVertices[VE].primerVerticeAbanico -VE)
```

4.2 Algoritmo.

Dado un lod, para cada vértice no eliminado (indicado en la lista de control) se pinta el abanico asociado a él y se cuentan los triángulos dibujados, para terminar cuando no queden más triángulos por pintar. De cada abanico se procesan sólo los *t* primeros vértices de los *t+1* vértices de que consta, ya que cada uno de ellos referencia a uno de los *t* triángulos representados, mientras que el último vértice sirve solamente para completar el abanico. Al procesar cada uno de estos vértices puede ocurrir dos cosas: a) el identificador del vértice indica que ese triángulo debe pintarse, b) el identificador del vértice indica que ese triángulo no debe pintarse.

Cuando ocurre un caso b), significa que hay una interrupción en el abanico, de modo que si éste continúa más adelante es necesario insertar el vértice inicial. Por tanto, la primera vez que se cumple el caso a) después de una interrupción se debe realizar esta inserción, (caso a.1 en el algoritmo).

```
PARA cada vértice v desde VE y mientras haya triángulos por pintar HACER
   interrupcionabanico <- cierto  {fuerza caso a.1)}
   PARA cada vértice vi del abanico asociado a v, desde el primero
   hasta el penúltimo HACER
      SI listaVérticesAbanicos[vi].id < TG ENTONCES {caso a) pintar vi}
         SI interrupcionabanico ENTONCES
            pinta(v)
            interrupcionabanico <- falso {no repetir caso a.1)}
         FINSI
         pinta(vi)
         NT <- NT-1
      SINO {caso b) no visualizar vi}
         SI no interrupcionabanico ENTONCES
            {es necesario pintar vi para cerrar triángulo}
            pinta(vi)
            interrupcionabanico <- cierto {fuerza caso a.1)}
         FINSI
      FINSI
  FINPARA
  SI no interrupcionabanico ENTONCES
    {último vértice vi de la lista para cerrar abanico }
    pinta(vi) {se pinta solamente si no hay interrupción}
  FINSI
FINPARA
```

Figura 9. Algoritmo de visualización

4.3 Análisis del coste computacional

El coste temporal del algoritmo de extracción del lod k depende del número total de vértices no eliminados, que es como máximo $|V^r|$ - k, y el número total de vértices de los abanicos asociados a estos vértices, que es como máximo $|VA^r|$ - $2k$. Como se vio al calcular el coste de almacenamiento, $|V'|$ y $|VA'|$ son en el peor caso, de orden cuadrático respecto a $|V_0|$ y $|F_0|$, respectivamente, pero este caso dista mucho del caso normal. Con un método de simplificación por eliminación de vértices y suponiendo que por término medio $|F_k| = 2|V_k|$, el coste es $O(8\ |V_k|)$ y con un método por eliminación de aristas es $O(13\ |V_k|)$.

5 Resultados

Los experimentos han sido realizados utilizando:

- **Hardware**: estación de trabajo Silicon Graphics RealityEngine 2, con un procesador MIPS R10000 a 194MHz y 256Mb RAM.
- **Software**: la codificación del modelo ha sido realizada en C++ y se ha utilizado OpenGL como librería gráfica. El método de simplificación utilizado para la construcción de las representaciones multirresolución es el propuesto por Garland y Heckbert [9] basado en contracción de aristas.
- **Mallas**: los objetos provienen de "*Stanford University Computer Graphics Laboratory*" (http://www-graphics.stanford.edu/data/3Dscanrep/) y "*Cyberware*" (http://www.cyberware.com/models/).

En la tabla 2 se resumen las características y los costes de almacenamiento de los objetos utilizados en los experimentos. Para cada uno de ellos se indica el número de vértices y caras del modelo original, y su coste de almacenamiento suponiendo una estructura basada en una lista de vértices y una lista de triángulos [10]. También se ha asumido que el coste de una palabra (entero, real o puntero) es de 4 bytes. Respecto a la representación multirresolución MOM, se indica el número de niveles de detalle, el número de caras y el coste total de almacenamiento. Respecto a la nueva representación propuesta en este artículo, se indica el número de *VerticeAbanico* y el nuevo coste de almacenamiento.

	Original			MOM			MOM con Abanicos	
	Vértices	Caras	Mb.	lods	Caras	Mb	V. Aba.	Mb.
Cow	2.905	5.804	0'100	2.803	14.982	0'237	17.852	0'202
Sphere	15.315	30.624	0'526	15.264	83.486	1'306	98.793	1'104
Bunny	34.835	69.451	1'193	33.990	182.192	2'876	216.759	2'445
Phone	83.045	165.963	2'850	81.668	441.181	6'940	523.844	5'887
Isis	187.871	375.736	6'450	187.370	993.559	15'667	1.181.373	13'309
Buda	543.653	1.085.636	18'646	543.106	2.754.083	43'957	3.297.593	37'598

Tabla 2. Características y coste espacial.

Se puede observar que el número de vértices abanico almacenados en la nueva representación es mayor que el de caras de la representación MOM. Sin embargo, dado que el coste de almacenar un vértice abanico es menor que el de una cara, el coste de almacenamiento de la nueva lista supone un ahorro de memoria alrededor del 20% respecto de la lista de caras. La repercusión de esta reducción sobre el coste total del modelo es muy importante, aproximadamente el 15%, debido a que precisamente se ha reducido el coste de la lista que mayor peso tiene en el modelo (compárese el número de caras con el de vértices o lods).

En la figura 11 se muestran los comportamientos de la nueva representación utilizando los objetos Bunny y Buda (figura 10). En el eje X se representa el nivel de detalle, donde 0 corresponde con el peor y 1 con el mejor. En el eje Y se representa el tiempo invertido (en segundos) por el modelo en recuperar los datos que pertenecen a un lod dado y visualizarlos. El comportamiento es similar al que se obtenía con el esquema previo de MOM y se puede observar la respuesta lineal respecto al número de triángulos del lod visualizado. La mejora que supone el utilizar abanicos de triángulos se ve mermada por el ligero aumento en el proceso de recuperación de los datos debido a una mayor complejidad del algoritmo, la corta longitud de los abanicos (3’3 triángulos de valor medio por abanico), los triángulos degenerados y la sobrecarga que supone el ejecutar para cada abanico la pareja de instrucciones *glBegin* y *glEnd* en la implementación realizada con *OpenGL*. En la figura 12 se representa el número de triángulos no degenerados y el número de triángulos degenerados (alrededor de un 23% del total de triángulos no degenerados) enviados al subsistema gráfico por nivel de detalle. En la figura 13, se representa el número de vértices enviados por lod en la representación MOM y los enviados en la nueva representación (sobre un 40% menos).

6 Conclusiones y trabajo futuro.

En este artículo se ha presentado un nuevo esquema multirresolución que permite almacenar y visualizar los distintos niveles de detalle como abanicos de triángulos.
El objetivo que se ha perseguido es doble, disminuir el tiempo de visualización y reducir el coste espacial. Los resultados experimentales muestran una reducción de un 15% del coste espacial respecto a la representación previa de MOM, en la que se basa esta nueva aproximación. Sin embargo, el comportamiento del nuevo modelo respecto al tiempo de visualización es similiar al de su antecesor. Una longitud media de los abanicos corta, un porcentaje alto de triángulos degenerados y el adecuar los abanicos al lod requerido en tiempo real, hace que no se obtenga una mejora global en el tiempo de visualización del modelo.

Figura 10. Modelos Bunny (izquierda) y Buda (derecha).

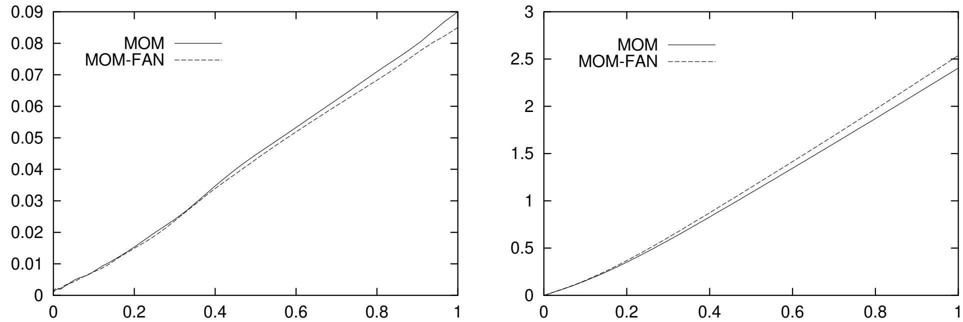

Figura 11. Tiempos de visualización (izquierda bunny, derecha buda)

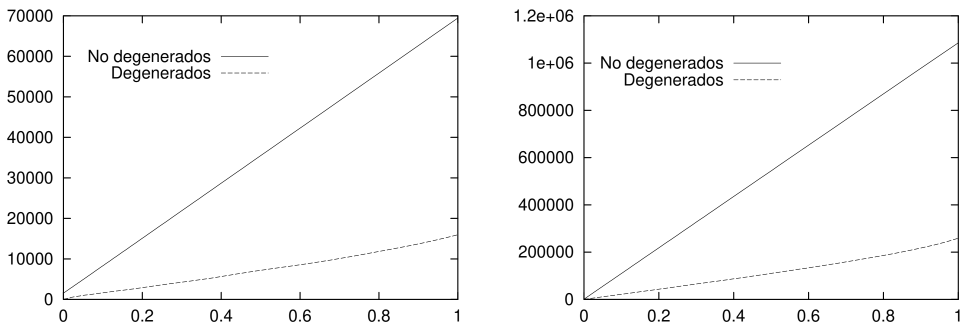

Figura 12. Triángulos no degenerados y degenerados (izq. bunny, der. buda)

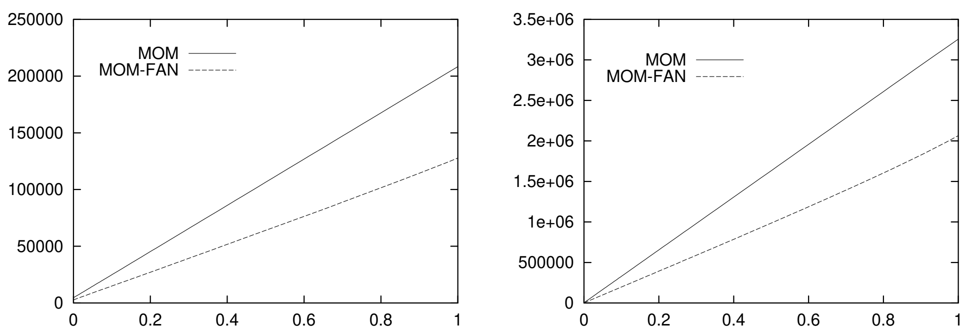

Figura 13. Vértices enviados en MOM y MOM-FAN (izq. bunny, der. buda)

El trabajo se encamina a partir de ahora a utilizar tiras que, en principio, van a permitir un número medio de triángulos por tira mayor que el obtenido utilizando abanicos. De esta forma se aumentaría aún más la reducción del coste espacial y se mejorarían los tiempos de visualización.

Referencias

[1] M. Garland, “Multiresolution Modeling: Survey & Future Opportunities”, State of the Art Reports of EUROGRAPHICS '99, 1999, pag 111 - 131.

[2] J. Ribelles, M. Chover, J. Huerta, and R. Quirós, “Multiresolution Ordered Meshes”, Proc. of 1998 IEEE Conference on Information Visualization, 1998, pag 198 - 204.

[3] H. Hoppe, “View-dependent Refinement of Progressive Meshes”, Proc. of SIGGRAPH '97, 1997, pag 189 - 198.

[4] J. El-Sana, E. Azanli y A. Varshney, “Skip Strips: Maintaining Triangle Strips for View-dependent Rendering”, Proc. of IEEE Visualization ‘99, 1999, pag 131 - 137.

[5] J. Ribelles, M. Chover, A. López, y J. Huerta, “Frame-to-frame Coherence of Multiresolution Ordered Meshes”, Proc. of CEIG '99, 1999, pag 91 - 104.

[6] J. Ribelles, M. Chover, A. López, and J. Huerta, “A First Step to Evaluate and Compare Multiresolution Models”, Short Papers and Demos of EUROGRAPHICS '99, 1999, pag 230 - 232.

[7] H. Hoppe, ”Progressive Meshes”, Proc. of SIGGRAPH '96, 1996, pag 99 - 108.

[8] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, ”Decimation of Triangle Meshes”, Proc. of SIGGRAPH '92, 1992, pag 65 - 70.

[9] M. Garland and P. Heckbert, “Surface Simplification Using Quadratic Error Metrics”, Proc. of SIGGRAPH ’97, 1997, pag 209 - 216.

[10] James D. Foley et al. "Computer Graphics. Principles and Practice", Addison-Wesley Publishing Company, 1990.