La Representación Progressive Fans (PFRep)

Antonio Cortés Carrillo
Lavinia 30-34, E-08902 Barcelona, España
Tlf: +34-934222749
e-mail: pfcortes@teleline.es

Resumen

Gran número de sistemas que soportan la interacción con modelos 3D utilizan representaciones basadas en triángulos. El tamaño de estas representaciones impone límites en las aplicaciones tanto en el proceso de visualización como durante la transmisión de modelos a través de líneas de comunicación. Las representaciones multiresolución aportan una solución al problema permitiendo en una misma representación la carga/visualización progresiva del modelo completo o de determinadas zonas del modelo. En este documento se presenta la *Progressive Fans (PFRep)*, una nueva representación multiresolución para mallas poligonales manifold o non-manifold. Un preproceso previo, permite la construcción de la PFRep aplicando recursivamente una primitiva de simplificación sobre una malla poligonal. El resultado es una estructura estática y compacta que permite la visualización progresiva/selectiva de la malla usando la primitiva fan, reduciendo a la mitad el proceso de visualización sobre APIs que soporten dicha primitiva como OpenGL. Su disposición estática en memoria minimiza los fallos de caché y permite tener múltiples instancias de un mismo modelo con un coste adicional de tan sólo $|V|$ *bits* por instancia.

Palabras clave: Multiresolución, refinamiento progresivo, refinamiento selectivo, compresión, simplificación incremental

1 Introducción

Motivación. Los *modelos multiresolución* se plantean como una solución al problema de visualización, almacenamiento y transmisión de complejos modelos 3D. Estos modelos mantiene una descripción de la geometría y propiedades

de la superfice del modelo con diferentes resoluciones o niveles de detalle (LOD, Level Of Detail), permitiendo su compresión, transmisión progresiva, refinamiento progresivo y, en algunos casos, *refinamiento selectivo*, consiguiendo aumentar o disminuir el detalle allí donde se desea (*criterio de refinamiento*).

Problemas. En la actualidad las representaciones con soporte para refinamiento selectivo son pocas, complejas y ávidas consumidoras de recursos. Todas ellas se contruyen una vez cargado el modelo y requieren una costosa estructura dinámica que se actualiza a cada frame y que en muchos casos debe duplicarse para controlar más de una instancia de un mismo modelo.

Contribución. En el presente documento se describe una nueva representación para modelos multiresolución a la que se ha denominado PFRep (Progressive Fans Representation). La representación se estructura en una tabla estática que permite la visualización progresiva y selectiva del modelo en tiempo real utilizando la primitiva *fan*[1], y el uso de multiples instancias de un mismo modelo con tan sólo $|V|$ bits por instancia.

Organización. El siguiente documento esta organizado de la siguiente forma: Inicialmente describiremos algunas de las representaciones existentes actualmente para modelos multiresolución. Seguidamente se analiza la aplicación de un proceso de simplificación incremental a una malla triangular arbitraria y la representación requerida para almacenar todo este proceso en una única representación multiresolución denominada PFRep. Para concluir analizaremos las propiedades y los resultados obtenidos con la nueva representación, compararándolos con otras representaciones existentes.

2 Trabajo previo

Los procesos de simplificación incremental [2] se caracterizan porque de forma iterativa eliminan un vértice o colapsan una arista o triángulo de una malla, definiendo implícitamente una jerarquía de diferentes niveles de detalle sobre el objeto simplificado. La diferencia entre dos niveles diferentes es relativamente pequeña reduciendo los artefactos visuales y en muchos casos esta diferencia es de carácter local no interfiriendo en el resto del modelo. Esta "no interferencia" permite aumentar o disminuir el detalle en determinadas zonas del modelo en función de un criterio de refinamiento como visibilidad, luminosidad, etc. A

este proceso se le denomina *refinamiento selectivo* y al criterio usado *criterio de refinamiento*.

Las primeras soluciones que hacen uso de la idea de refinamiento selectivo aparecen en algoritmos específicos para aproximar mallas regulares o mapas de alturas, en el contexto de Sistemas de Información Geográfica ([3][4]).

La aplicación del refinamiento selectivo en mallas arbitrarias surge con las Progressive Meshes (PM) de Hoppe [5]. Las PM son producto de la simplificación de una malla mediante una secuencia de eliminación de aristas. La representación final consta de una malla base y una sucesión de operaciones (vsplit) correspondientes a las operación inversa de la secuencia original. Para llevar a cabo el refinamiento selectivo de la malla, basta recorrer la secuencia de vsplits y aplicar las operaciones de refinamiento necesarias según el criterio requerido. Para evitar LODs no deseados propone un método que tiene en cuenta las depdendencias entre vértices de la secuencia.

Xia y Varshney [6] extienden el esquema de las PM añadiendo una jerarquía de dependencias entre vértices, o *Merge Tree*, la cual es utilizada para recuperar todas las operaciones de refinamiento necesarias para alcanzar el LOD deseado. El refinamiento selectivo se consigue mediante una búsqueda ascendente en el árbol de dependencias, con lo cual precisa visitar toda la estructura incluso para obtener la malla más sencilla posible. Posteriormente Hoppe adapta el esquema propuesto por Xia y Varshney a las PM, aplicando el criterio requerido únicamente sobre los vértices activos y sobre los inmediatamente dependientes que pueden ser objeto de simplificación ([7][8]).

Desafortunadamente tanto las PM como el algoritmo de Xia y Varshney asumen que el modelo es dos-manifold [1] limitando el proceso de simplificación. Luebke y Erikson proponen el uso de un 'vertex-tree' donde cada nodo representa un grupo de vértices de la escena [9]. Su funcionamiento es similar a un proceso de 'vertex-clustering' permitiendo el refinamiento selectivo sobre mallas no-manifold.

El-Sana y Varshney [10] utilizan la primitiva pair-contraction en el proceso de simplificación, generando una arbol de dependencias más compacto en memoria que el Merge Tree. Durante el proceso de refinamiento el arbol de dependencias se actualiza dinámicamente para obtener la malla deseada.

E. Puppo [11] introduce las *Multi-Triangulations (EM)*, un modelo general basado en la organización de una colección de fragmentos de triangulaciones,

[1] Un modelo dos-manifold o manifold es una superficie donde el contorno infinitesimal de todo punto es topológicamente equivalente a un disco (o un sector para contornos)

en un grafo acíclico dirigido. Esta organización permite obtener un determinado nivel de detalle en un tiempo lineal.

Por último Jose Ribelles propone las MOMs (Mallas Ordenadas Multiresolución) [12], que permiten la extracción eficiente de cualquier LOD de la secuencia original, pero no de LODs arbitrarios (refinamiento selectivo). En artículos posteriores se realizan diversas mejoras [13] y finalmente se propone una adaptación que permite la extracción de LODs arbitrarios [14] pero que requiere recorrer toda la estructura en cada iteracción.

3 Progressive Fans Representación (PFRep)

En la siguiente sección describiremos una sencilla codificación de un modelo multiresolución creado durante el proceso de simplificación incremental, y como esta codificación, denominada malla multiresolución, puede representarse eficientemente utilizando la Progressive Fans.

3.1 Malla triangular

En adelante utilizaremos las siguientes definiciones:

Definicion 3.1 (Malla triangular). *Consideraremos una malla triangular M como una tupla (V, F) y una función $\phi : \mathbb{N} \rightarrow \mathbb{R}^3$, donde V representa un conjunto de vértices $\mathbf{v_j}$ con coordenadas $(x_j, y_j, z_j) \in \mathbb{R}^3$, F es un conjunto de tripletas ordenadas $\{v_j, v_k, v_l\}$ que especifican los vértices de una cara triangular y ϕ es una función inyectiva que permite asignar a v_j sus coordenadas en V. Utilizaremos $\mathbf{v_j}$ en negrita para identificar las coordenadas del vértice y v_j en cursiva para representar una referencia al mismo de forma que $\phi(v_j) = \mathbf{v_j}$. De igual forma utilizaremos v para identificar el número de vértices o $|V|$.*

Definicion 3.2 (Vecindad de un vértice). *Dada una malla triangular M y un vértice $\mathbf{v} \in M$, definimos el conjunto de todos los triángulos que inciden en $\mathbf{v}$ como vecindad (Neighboorhood) de $\mathbf{v}$:*

$$N(\mathbf{v}) = (T_1, T_2, \ldots T_n) \tag{1}$$

3.2 Simplificación incremental

Sea M^v un modelo 3D arbitrario representado por una malla triangular con v vértices, consideraremos la secuencia de mallas de complejidad decreciente

obtenida mediante una secuencia de transformaciones basadas en una primitiva de eliminación de vértices $\bigtriangledown$ e iniciadas en una malla M^v:

$$M^v \xrightarrow{\bigtriangledown_0} M^{v-1} \xrightarrow{\bigtriangledown_1} \dots \xrightarrow{\bigtriangledown_{n-1}} M^{v-n}$$

Nota. *El superíndice de cada malla M^{v-i} denota el número de vértices y el subíndice de cada primitiva $\bigtriangledown_i$ denota la posición dentro de la secuencia de transformaciones y a la vez el número exacto de vértices eliminados de M^v antes de la transformación. En adelante utilizaremos el superíndice para identificar la conectividad de un vértice en una malla concreta de la secuencia, así por ejemplo, $N(v_r)^{v-3}$ denota la vecindad del vértice v_r en la malla M^{v-3}, la cual puede no coincidir con la vecindad de v_r en otras mallas de la secuencia. Igualmente utilizaremos n para identificar el número de primitivas $\bigtriangledown$ aplicadas y por consiguiente el número de vértices simplificados.*

Consideraremos una primitiva $\bigtriangledown$ restringida tal que:

a) Se elimina un único vértice al que denominaremos $\mathbf{v_r}$

b) Se eliminan todas las caras de la vecindad del vértice $N(\mathbf{v_r})$ y únicamente esas caras.

La figura 1 muestra tres primitivas $\bigtriangledown$ válidas. El método para determinar la secuencia óptima de transformaciones no es único y depende de una función heurística que minimice el error entre las sucesivas mallas. En lo sucesivo, supondremos la existencia de un módulo externo capaz de determinar una secuencia óptima de transformaciones.

Una transformación *primitiva* $\bigtriangledown_i$ consiste en la eliminación de un vértice $\mathbf{v_{r_i}}$ y la substitución de la vecindad de ese vértice $N(\mathbf{v_{r_i}})$ por un nuevo conjunto de triángulos que denominaremos $F_{\mathbf{v_{r_i}}}$. Dado que $N(\mathbf{v_{r_i}})$ es un conjunto derivable de $\mathbf{v_{r_i}}$, para aplicar una primitiva $\bigtriangledown_i$ solo es necesario conocer $\mathbf{v_{r_i}}$ y $F_{\mathbf{v_{r_i}}}$. Toda primitiva $\bigtriangledown_i$ con estas características se puede codificar utilizando *triángulos dependientes*.

Definicion 3.3 (Triángulo dependiente). *Dada una malla $M^v = (V^v, F^v)$ y una secuencia de primitivas $\bigtriangledown_0, \bigtriangledown_1, \dots \bigtriangledown_{n-1}$, cada una de las primitivas $\bigtriangledown_i$ con parámetros $\mathbf{v_{r_i}}, F_{v_{r_i}}$, puede representarse como un conjunto de triángulos dependientes de un vértice v_{d_i} tal que $\phi(v_{d_i}) = \mathbf{v_{r_i}}$. Formalmente*

$$\bigtriangledown = \{\mathbf{v_r}, F_{\mathbf{v_r}}\} \equiv \{\bar{T}_0^{v_d}, \bar{T}_1^{v_d}, \dots \bar{T}_k^{v_d}\} = \bar{F}^{v_d} \qquad \phi(v_d) = \mathbf{v_r}$$

Todos los triángulos que se han creado como resultado de una operación $\bigtriangledown_i$ pueden representarse de esta forma. Los triángulos que pertenecen a la malla original M^v consideraremos que son triángulos dependientes donde la dependencia es nula.

Un triángulo dependiente $\bar{T}_q^{v_d} = \{v_d, \{v_j, v_k, v_l\}\}$ pertenece a una malla $M^{v-i} = (V^{v-i}, F^{v-i})$ de la secuencia si y solo si:

- *$\phi(v_d) \notin V^{v-i}$, es decir, el vértice del que depende ha sido eliminado en alguna de las operaciones $\bigtriangledown_0, \bigtriangledown_1, \ldots \bigtriangledown_{i-1}$, y*
- *$\phi(v_j), \phi(v_k), \phi(v_l) \in V^{v-i}$ es decir, los 3 vértices que definen el triángulo no han sido eliminados.*

3.3 Malla multiresolución

Utilizando la definición de triángulo dependiente, una malla M^v puede describirse como un conjunto de triángulos dependientes y añadir cada una de las primitivas $\bigtriangledown_i$ del proceso de simplificación. Únicamente queda identificar el orden de cada una de las operaciones $\bigtriangledown_i$ mediante una nueva función inyectiva $\gamma : \mathbb{N} \rightarrow \mathbb{N}$ que asigna a una entrada $v_d \in M$ un valor entero para identificar el orden de eliminación del vértice $\mathbf{v_d}$. Si $\mathbf{v_d}$ no ha sido eliminado el resultado de la función $\gamma(v_d)$ es n. El resultado final tras n primitivas $\bigtriangledown$ es una malla multiresolución denotada por $\bar{M}^{v-n}$. Formalmente:

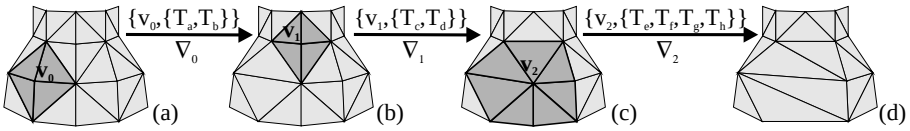
$$\begin{aligned}
\bar{M}^{v-n} &= (V, F) + \bigtriangledown_0 + \bigtriangledown_1 + \ldots \bigtriangledown_{n-1} \\
&= (V, \bar{F}^{\emptyset}) + \bigtriangledown_0 + \bigtriangledown_1 + \ldots \bigtriangledown_{n-1} \\
&= (V, \bar{F}^{\emptyset} + \bar{F}^{v_{d_0}} + \bar{F}^{v_{d_1}} + \ldots \bar{F}^{v_{d_{n-1}}}) \\
&= (V, \bar{F}^{v-n}) \quad \gamma(v_{d_0}) = 0, \gamma(v_{d_1}) = 1, \ldots \gamma(v_{d_{n-1}}) = n-1
\end{aligned}$$

En la figura 1 puede verse el resultado tras aplicar tres operaciones $\bigtriangledown$.

Nótese que no se eliminan vértices ni triángulos de la malla, unicamente se añaden triángulos dependientes. Suponiendo una conectividad de grado ≈ 6 por vértice [2] obtenemos que para una malla $M = (V, F)$ es necesario codificar un total de $3|F|$ triángulos.

En la siguiente sección veremos como una representación adecuada para $\bar{M}^{v-n}$ permite la extracción eficiente de cualquier malla de la secuencia.

[2] Apartir de la fórmula de Euler ($V - E + F = 2$) y dado que $\Sigma_{|V|}(degree(v_i)) = 2E$ se puede obtener que $degree(v_i) = \frac{6V-4}{V} \approx 6$



$(a)\quad \bar{F}^{v-0} = \{\ldots,\ldots\}$

$(b)\quad \bar{F}^{v-1} = \{\ldots,\bar{T}_a^{v0},\bar{T}_b^{v0},\ldots\} \quad \gamma(v_0)=0$

$(c)\quad \bar{F}^{v-2} = \{\ldots,\bar{T}_a^{v0},\bar{T}_b^{v0},\bar{T}_c^{v1},\bar{T}_d^{v1},\ldots\} \quad \gamma(v_0)=0,\gamma(v_1)=1$

$(d)\quad \bar{F}^{v-3} = \{\ldots,\bar{T}_a^{v0},\bar{T}_b^{v0},\bar{T}_c^{v1},\bar{T}_d^{v1},\bar{T}_e^{v2},\bar{T}_f^{v2},\bar{T}_g^{v2},\bar{T}_h^{v2},\ldots\} \quad \gamma(v_0)=0,\gamma(v_1)=1,\gamma(v_2)=2$

$\mathbf{PF}^{v-3}$			
V	**L**		
$\mathbf{v_0}=\{x_0,y_0,z_0\}$	$\{\bar{T}_1^{\emptyset},\bar{T}_2^{\emptyset},\bar{T}_3^{\emptyset},\bar{T}_4^{\emptyset}\}$	$=N(v_0)^{v-0}$	$,\gamma(v_0)=0$
$\mathbf{v_1}=\{x_1,y_1,z_1\}$	$\{\bar{T}_5^{\emptyset},\bar{T}_6^{\emptyset},\bar{T}_7^{\emptyset},\bar{T}_8^{\emptyset}\}$	$=N(v_1)^{v-1}$	$,\gamma(v_1)=1$
$\mathbf{v_2}=\{x_2,y_2,z_2\}$	$\{\bar{T}_c^{v1},\bar{T}_a^{v0},\bar{T}_b^{v0},\bar{T}_9^{\emptyset},\bar{T}_{10}^{\emptyset},\bar{T}_{11}^{\emptyset},\bar{T}_{12}^{\emptyset}\}$	$=N(v_2)^{v-2}$	$,\gamma(v_2)=2$
$\mathbf{v_3}=\{x_c,y_c,z_c\}$	$\{\ldots\}$	$\subseteq N(v_3)^{v-3}$	
$\mathbf{v_4}=\{x_d,y_d,z_d\}$	$\{\ldots\}$	$\subseteq N(v_4)^{v-3}$	
$\ldots$	$\ldots$		
$\mathbf{v_{v-1}}$	$\{\ldots\}$	$\subseteq N(v_{v-1})^{v-3}$	

Figura 1: Arriba: Construcción de una malla multiresolución. Por cada primitiva $\bigtriangledown$ se añaden el conjunto de triángulos dependientes que codifican dicha primitiva y se actualiza la función γ. Abajo: La representación PFRep.

3.4 La representación Progressive Fans

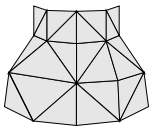
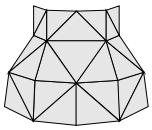
A partir de la malla multiresolución $\bar{M}^{v-n} = (V, \bar{F}^{v-n})$ y la función γ se procede a particionar el conjunto $\bar{F}^{v-n}$ tal y como sigue:

Definicion 3.4 (Progressive fans). *Dada una malla multiresolución* $\bar{M}^{v-n} = (V, \bar{F}^{v-n})$ *definimos una Progressive Fans (PF) como una tupla* $(V, \bar{L}^{v-n})$ *donde* $\bar{L}^{v-n}$ *describe una partición de* $\bar{F}^{v-n}$ *en v subconjuntos* $\{\bar{L}_0, \bar{L}_1, \ldots \bar{L}_{v-1}\}$ *tal que para todo i, si* $\gamma(i) = n$, $\bar{L}_i$ *es un subconjunto de la vecindad del vértice* $\phi(i)$ *en la malla original y si* $0 \leq \gamma(i) < n$, $\bar{L}_i$ *es equivalente a todo la vecindad del vértice* $\phi(i)$ *en la malla* $M^{v-\gamma(i)}$. *Formalmente:*

$$PF^{v-n} = (V, \bar{L}^{v-n}) = (V, \{\bar{L}_0, \bar{L}_1, \ldots \bar{L}_{v-1}\}) \tag{2}$$

$$donde \quad \forall i \begin{cases} \bar{L}_i \subseteq N(\phi(i))^v & si\ \gamma(i) = n \\ \bar{L}_i = N(\phi(i))^{v-\gamma(i)} & si\ 0 \leq \gamma(i) < n \end{cases} \tag{3}$$

Una forma de codificar esta representación es utilizar una tabla con v entradas ordenadas según el valor de γ, tal que cada entrada represente un conjunto $\bar{L}_p$ y las coordenadas del vértice $\phi(p) \in V$. Para una entrada i tal

(a)  $k = 1$ M^{v-1} (b)  $I = [0, 1, 0]$

$\mathbf{PF}^{v-3}$	
V	**L**
$\mathbf{v_1}$	$\{\bar{T}_5^{\emptyset}, \bar{T}_6^{\emptyset}, \bar{T}_7^{\emptyset}, \bar{T}_8^{\emptyset}\}$
$\mathbf{v_2}$	$\{\quad, \bar{T}_a^{v_0}, \bar{T}_b^{v_0}, \bar{T}_9^{\emptyset}, \bar{T}_{10}^{\emptyset}, \bar{T}_{11}^{\emptyset}, \bar{T}_{12}^{\emptyset}\}$
$\mathbf{v_3}$	$\{\ldots\}$
$\mathbf{v_4}$	$\{\ldots\}$
$\ldots$	$\ldots$

	$\mathbf{PF}^{v-3}$	
I	**V**	**L**
0	$\mathbf{v_0}$	$\{\bar{T}_1^{\emptyset}, \bar{T}_2^{\emptyset}, \bar{T}_3^{\emptyset}, \bar{T}_4^{\emptyset}\}$
1		
0	$\mathbf{v_2}$	$\{\bar{T}_c^{v_1}, \quad, \quad, \bar{T}_9^{\emptyset}, \bar{T}_{10}^{\emptyset}, \bar{T}_{11}^{\emptyset}, \bar{T}_{12}^{\emptyset}\}$
0	$\mathbf{v_3}$	$\{\ldots\}$
0	$\mathbf{v_4}$	$\{\ldots\}$
0	$\ldots$	$\ldots$

Figura 2: (a) Ref. progresivo. Para extraer una malla k de la secuencia, se discriminan las entradas $[0..k)$ y se seleccionan los triángulos donde v_d es nulo o menor que k. (b) Ref. selectivo. Se discriminan las entradas simplificadas (bit en I a 1) y se seleccionan los triángulos donde v_d es nulo o esta eliminado.

que $0 \leq i < n$ el conjunto $\bar{L}_p$ correspondiente es tal que $\gamma(p) = i$; para las entradas $n \leq i < v$ el orden de los conjuntos $\bar{L}_p$ es indiferente (figura 1).

3.5 Propiedades de la PFRep

El resultado de esta estructuración es una tabla con v entradas que representa una malla multiresolución $\bar{M}^{v-n}$ y de la que pueden extraerse una serie de propiedades como el refinamiento progresivo y selectivo (figuras 2a y 2b).

3.5.1 Refinamiento progresivo

A partir de una PF^{v-n} es posible obtener cualquier malla M^{v-k} de la secuencia original de simplificación de forma rápida y sencilla. Solo es necesario realizar las siguientes operaciones:

1. Se discriminan las entradas desde la 0 a la k
2. Para todo $\bar{T}_i^{v_d} = \{v_d, \{v_j, v_k, v_l\}\}$ del resto de entradas, el triángulo $\{v_j, v_k, v_l\}$ es válido si v_d es nulo o $v_d < k$ (versión reducida de la def. 3.3).

3.5.2 Refinamiento selectivo

En los procesos de simplificación incremental, es posible aplicar una primitiva de simplificación de la secuencia sin necesidad de haber aplicado todas las

primitivas anteriores. La PFRep ofrece una estructura adecuada para conocer esta dependencia entre primitivas. Para ello consideraremos:

- Una secuencia binaria I de longitud n (número total de primitivas) asociada a la PFRep que determina si un vértice i esta simplificado (primitiva $\bigtriangledown_i$ aplicada) con el valor '1' o no con el valor '0'.
- Dos funciones recursivas SimplificaVertex e InsertaVertex que permiten aplicar o recuperar, respectivamente, una primitiva $\bigtriangledown_i$, gestionando adecuadamente las dependencias entre primitivas.

```
funcion SimplificaVertex(PF^{v-n}, I, i)
    //si es simplificable y no esta simplificado
    si  i < n and I_i = 0  entonces
        I_i := 1;
        para todo  T̄_r = {v_{d_r}, {v_j, v_k, v_l}} ∈ L̄_i
            SimplificaVertex(PF^{v-n}, I, v_{d_r});
        fpara
    fsi
```

```
funcion InsertaVertex(PF^{v-n}, I, i)
    //si es simplificable y esta simplificado
    si  i < n and I_i = 1  entonces
        I_i := 0;
        para todo  T̄_r = {v_{d_r}, {v_j, v_k, v_l}} ∈ L̄_i
            si  v_{d_r} = ∅ or I_{v_{d_r}} = 1  entonces
                InsertaVertex(PF^{v-n}, I, [v_j, v_k, v_l]);
            fsi
        fpara
    fsi
```

  - SimplificaVertex. Un vértice $\mathbf{v_i}$ puede simplificarse, si y solo si todos los vértices dependientes $v_{d_k} \in \bar{L}_i$ estan simplificados.
  - InsertaVertex. Un vértice $\mathbf{v_i}$ puede insertarse, si y solo si para cada triángulo dependiente $\{v_d, \{v_j, v_k, v_l\}\} \in \bar{L}_i$ tal que v_d esta simplificado o es nulo, los vértices v_j, v_k y v_l deben estar insertados.

- Un criterio de refinamiento que determina cuando debe insertarse o simplificarse un vértice. El criterio utilizado es similar al propuesto por Hoppe en [7], que utiliza una envolvente esférica, un cono de normales y dos valores escalares por vértice para aplicar el refinamiento.

A partir de la PF^{v-n} y la máscara I asociada se procede como sigue:

1. Se discriminan las entradas con el bit en I a 1 (entradas simplificadas)
2. Para todo $\bar{T}_i^{v_d} = \{v_d, \{v_j, v_k, v_l\}\}$ del resto de entradas, el triángulo $\{v_j, v_k, v_l\}$ es válido si v_d es nulo o el bit I_{v_d} es 1.

3.6 Codificación eficiente de la PFRep

3.6.1 Descomposición en coronas

A partir de la definición de *vecindad de un vértice* (Def. 3.2) y dado que todo L_i es un subconjunto de la vecindad de $\mathbf{v_i}$ todo triángulo $T_r = \{v_j, v_k, v_l\} \in L_i$

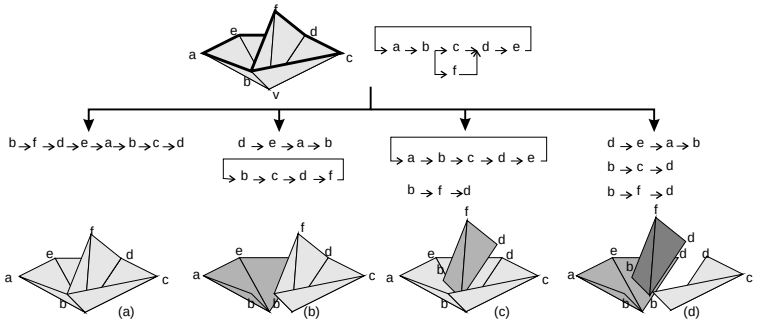


Figura 3: Arriba: Grafo obtenido a partir de la vecindad de v. Abajo: Cuatro descomposiciones válidas del grafo y su interpretación geométrica. La descomposición óptima del grafo es de una corona (caso (a)).

puede representarse mediante el par $\{v_p, v_q\}$ no incidente en $\mathbf{v_i}$. Todo el conjunto de pares ordenados de L_i determina un grafo dirigido G_i, no necesariamente cíclico[3]. En la figura 3 puede verse un ejemplo de grafo.

El grafo G_i puede descomponerse en r recorridos o *Coronas* $\{C_0, \dots C_{r-1}\}$ tal que todo arco del grafo G_i se encuentra representado en una única corona (véase la figura 3). Contrariamente los nodos sí pueden estar repetidos dentro de una misma corona o entre ellas (necesario para mallas no-manifold). Una descomposición óptima es aquella que genera el mínimo número de coronas.

Finalmente las coronas $\{C_0, \dots C_{r-1}\}$ puede concatenarse (previa conversión de coronas cerradas en abiertas) en un único recorrido cíclico Cf_i de arcos etiquetados, donde la etiqueta indica si el arco es válido (1) o no (0). El recorrido Cf_i se codifica con un array $A_i = [v_0, v_1, \dots v_{p-1}]$ y una máscara binaria $B_i = [b_0, b_1, \dots b_{p-1}]$ donde para todo $j \in [0 \dots (p-1)]$, la entrada v_j representa el nodo correspondiente en Cf_i, y b_j identifica si el arco $\{v_j, v_{(j+1)mod\ p}\}$ (un triángulo en su interpretación geométrica) es válido (1) o no (0).

Únicamente queda codificar las dependencias para lo cual basta particionar la mascara binaria B_i en k máscaras. Finalmente se obtiene:

$$\bar{L}_i \equiv (A_i, D_i = \{(v_{d_0}, B_{d_0}), (v_{d_1}, B_{d_1}), \dots (v_{d_{k-1}}, B_{d_{k-1}})\}) \quad \sum_{j=0\dots k-1} B_{d_j} = B_i$$

Donde v_{d_j} identifica una dependencia y B_{d_j} los triángulos asociados.

[3]Nótese que si el vértice pertenece a un contorno el grafo es abierto

```
struct   PFPck {
         int nVertex, nCompress, nBase;
         Vector v[], normal[];
         PFVertex *vert[];
};
struct   PFDependency {
         int vd; //indice al dependiente
         byte dmask[]***;
};
```

```
struct   PFVertex {
         // datos ref. selectivo
         float radio, dotcone, surface, dist;

         int NumA, NumD;

         int A[]*;
         PFDependency D[]**;
};
```

	A_i				D_i			
PFVertex	$A[0]$	$A[1]$	...	$A[NumA-1]$	$D[0]$	$D[1]$	...	$D[NumD-1]$
24 bytes	$(4 * NumA)$ bytes				$(z * NumD)$ bytes			

Figura 4: Arriba: Estructuras requeridas para codificar una PFRep. Abajo: Disposición de una entrada $\bar{L}_i$ en memoria; z es exactamente $4+(NumA+7)/8$

3.6.2 Estructura

La figura 4 muestra las estructuras necesarias para la codificación eficiente de una PFRep utilizando la notación en lenguaje C. En las pruebas realizadas se obtiene una conectividad de ≈ 6 y un número de dependencias de ≈ 3 por vértice, obteniendo un coste medio de 91 *bytes* por vértice (asumiendo que tanto un *float* como un *integer* requieren 4 bytes). Considerando la vecindad de un vértice menor de 256, el coste se reduce a 85 *bytes* por vértice.

3.6.3 Visualización usando triangle-fans

La definición de corona es similar a la primitiva de visualización **lista fan**. Para una entrada L_i el pivote se corresponde con el vértice v_i y cada secuencia conexa de "1" en B_i identifica una lista fan. La función *glDrawFanL* implementa este proceso de obtención de listas fan; el uso de ´[]´ denota el acceso a un elemento de la secuencia y en el caso de B_i el acceso a un único bit.

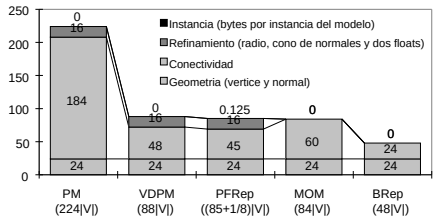
```
funcion glDrawSelectivePF(PF, I)
    para i := 0 hasta v − 1
        //si la entrada i no esta simplificada
        si I_i = 0 entonces
            // Obtener la mascara binaria bMask
            bMask := ∅
            para todo (v_{d_j}, B_{d_j}) ∈ D_i
                //si v_{d_j} es nulo o esta simplificado
                si v_{d_j} = ∅ OR I_{v_{d_j}} = 1 entonces
                    bMask := bMask or B_{d_j};
                fsi
            fpara
            glDrawFanL( V, i, A_i, bMask );
        fsi
    fpara
```

```
funcion glDrawFanL(V, pivot, A, bM)
    int nVert := |A|;
    para j := 0 hasta nVert − 1
        mientras (j < nVert and bM[j] <> 1) j++;
        si j < nVert entonces
            open := j; // Inicio del conjunto de bits a 1
            mientras (j < nVert and bM[j] <> 0) j++;
            close := j; // Final del conjunto de bits a 1

            glBegin( GL_TRIANGLE_FAN );
            glVertex3fp( V[pivot] );
            para open hasta close
                glVertex3fp( V[A[open %nVert]] );
            fpara
            glEnd();
        fsi
    fpara
```

Modelo	$\|\tilde{V}\|$	$\|\tilde{F}\|$	Mb	bytes/v
venus	134345	268686	11.15	83.0
santa	75781	151558	6.35	83.8
rabbit	67038	134074	5.61	83.7

Modelo	$\|\tilde{V}\|$	$\|\tilde{F}\|$	Mb	bytes/v
horse	48485	96966	4.03	83.1
bunny	35947	69451	2.89	83.0
tricer.	2832	5651	0.235	83.3



	PM	VDPM	PFRep	MOM	BRep
	Memoria estatica				
Geometria	24	24	24	24	24
Conectividad	184	48	45	60	24
Refinamiento	16	16	16	0	0
Total	**224**	**88**	**85**	**84**	**48**
	Memoria dinamica				
Por vert. activo	112	112	0	0	0
Por vertice	0	0	0.125	0	0

Figura 5: Arriba:Detalle de los modelos usados. Abajo:Memoria estática y dinámica requerida para la representación PM, VDPM, PFRep, MOM y BRep.

4 Resultados y comparativa

El hardware utilizado en los test consta de un AMD K6 III con 64Mb y chip gráfico Riva TNT (NVidia). El software puede obtenerse en la dirección http://www.terra.es/personal3/atoniman.

4.1 Espacio

El espacio estático requerido por la PFRep es similar al usado en las VDPM y las MOM[4]) (Figura 5), pero a diferencia de esta última, la PFRep permite el refinamiento selectivo en tiempo real de manera eficiente.

En comparación con otras estructuras para refinamiento selectivo y en concreto para las VDPM de Hoppe, la PFRep no requiere una estructura dinámica para la malla intermedia y tan sólo es necesario una máscara binaria I de longitud $|V|$ por cada instancia del modelo. La VDPM por el contrario requiere una estructura de tamaño $112|A|$ (A es el número de vértices activos).

4.2 Coste computacional

Ref. progresivo . El coste de extracción secuencial sólo depende de la malla de salida ($\theta(A)$). En la figura 6 se aprecia como el uso de listas fan compensa este coste, mejorando los tiempos de render frente al mismo modelo en BRep. Los tiempos de extracción secuencial y aleatorio son equivalentes dado que el algoritmo no depende del LOD anterior (véase 3.5.1).

[4]Las estimaciones para las MOM se han obtenido a partir de lo datos en [12] y [13].

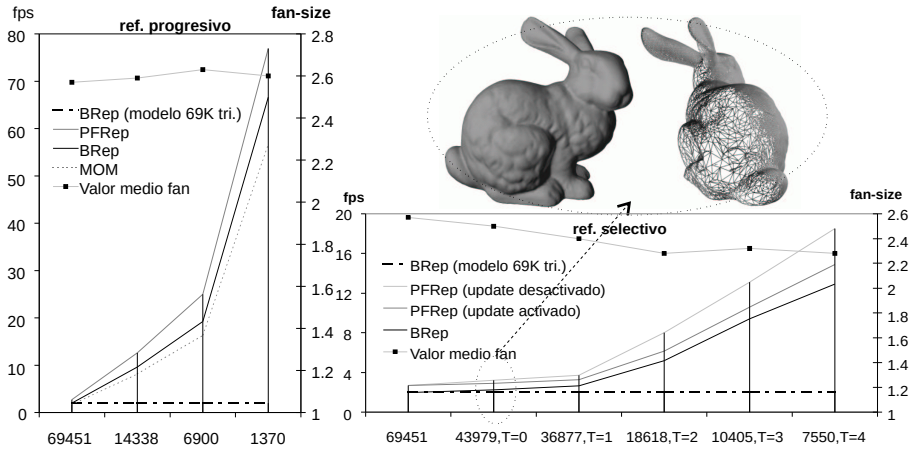


Figura 6: Coste computacional para el ref. progresivo y selectivo.

Ref. selectivo . Para aplicar el ref. selectivo se ha de realizar un recorrido lineal sobre I ($\theta(V)$) y calcular el criterio sobre los vértices activos ($\theta(A)$). La figura 6 demuestra como en todos los niveles de tolerancia probados, el coste del recorrido binario es insignificante y el requerido para la extracción y el cálculo del criterio de refinamiento se ve compensado, nuevamente, por el uso de listas fan y por la eliminación de triángulos por culling (véase fig. Bunny).

5 Conclusiones y futuros trabajos

En el siguiente trabajo se ha presentado una nueva representación para modelos multiresolución manifold y non-manifold que soporta el refinamiento progresivo/selectivo. La PFRep utiliza una estructura estática basada en listas fan y un array de bits para aplicar el refinamiento selectivo y visualizar en tiempo real el modelo, sin necesidad de una estructura dinámica para la malla intermedia. Las pruebas realizadas demuestran que el uso de listas fan compensa los cálculos requeridos para la extracción del LOD y el cálculo del criterio de refinamiento, mejorando significativamente el rendimiento.

La simplicidad del algoritmo de extracción, la estructuración en listas fan y el bajo coste en memoria estática y dinámica (sólo $|V|$ bits por instancia), hacen de la PFRep una opción a tener en cuenta para la visualización en tiempo real de grandes y pequeños modelos.

Futuros desarrollos se encaminan hacia la elaboración de un método eficiente de compresión de la PFRep para su transmisión progresiva y a la implementación de un proceso de "carga selectiva"donde el modelo se encuentra en memoria secundaria y se carga únicamente las zonas requeridas por el criterio de refinamiento. Este último desarrollo permitiría una independencia de la memoria del sistema para la visualización de modelos masivos en tiempo real.

6 Agradecimientos

Me gustaría agradecer especialmente la colaboración y ayuda de *Carlos Andújar Gran.* del Dpto. de LSI de la Universidad Politécnica de Cataluña (UPC).

Referencias

[1] J. Neider, T. Davis, M. Woo "OpenGL Programming Guide" *Addison-Wesley.*

[2] P. Cignoni and C. Montani and R. Scopigno. "A comparison of mesh simplification algorithms" *Computers Graphics*, vol. 22, pp. 37–54, 1998.

[3] De Floriani, L., Marzano, P., and Puppo, E. "Multiresolutions models for topographic surface description" *The Visual Computer 12*, 7 (1996), pp 317-345.

[4] P. Lindstrom, D. Koller, W. Ribarsky, F. Hodges, N. Faust and G.A. Turner. "Real-time continuous level of detail rendering of height fields." *Sigraph 96*, pp. 109-118.

[5] Hoppe, H. "Progressive Meshes" In *Computer Graphics* Sigraph 96, pp. 99-108.

[6] Xia, J., and Varshney, A. "Dynamic view-dependent simplification for polygonal models" *Visualizacion '96 Proceedings*, 1996 pp. 327-334.

[7] Hoppe, H. "View-dependent refinement of progressive meshes" *Computer Graphics* (SIGGRAPH '97), pp. 189-198.

[8] Hoppe, H. "Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering" *Microsoft Research*

[9] Luebke, D., Erikson, C. "View-dependent simplification of arbitrary polygonal environments". *Computer Graphics* (SIGGRAPH '97), pp. 199208.

[10] Jihad El-Sana, Amitabh Varshney "Generalized View-Dependent Simplification". *Computer Graphics Forum* (Eurographics '99), pp. 83-94.

[11] E. Puppo "Variable Resolution Triangulations" *Computational Geometry: Threory and Applications*, vol. 11, 1998

[12] J. Ribelles, M. Chover, J. Huerta, R. Quirós "Un Modelo Multirresolución para Visualización Interactiva" *CEIG 1998*, pp. 267-280

[13] J. Ribelles, A. López, I. Remolar, O. Belmonte, M. Chover "Representación de Modelos Multirresolución con Abanicos de Triángulos" *CEIG 2000*, pp. 51-64

[14] J. Ribelles, A. López, O. Belmonte, I. Remolar, M. Chover "Variable Resolution Level-Of-Detail of Multiresolution Ordered Meshes" *WSCG 2001*, Vol. 2, 299-306,