3 Eficiencia con polígonos

Introducción
Técnicas de aceleración
Nivel de detalle
Hardware gráfico

Introducción

- Representación poligonal
  - Los triángulos mejoran las prestaciones del Hw
  - Representación principal
    - Facilidad de creación de objetos poligonales: digitalizador 3D, escáner 3D, descripción matemática, barrido, conversión poligonal
    - Algoritmos de visualización realista (Hw). Algoritmos de sombreado por interpolación (Gouraud, Phong)
  - Problema fundamental (escalado de geometría)
    - Dibujar muchos polígonos en pocos píxeles (simuladores de vuelo, Realidad Virtual, Juegos por Ordenador)
    - Solución: representación jerárquica
    - Problema: animación y control de la jerarquía

Introducción

- Eficiencia con polígonos
  - Las tarjetas gráficas visualizan más tri/seg. De los que pueden enviarse por el bus
    - Ej. 1 Millón de triángulos a 30Hz (36 bytes/tri) → 1080 MB/sec
    - Ej. Geforce 6800 Ultra
      - Memoria: 512 MB
      - Ancho de banda de la memoria: 33.6 GB/sec.
      - Frecuencia de relleno: 6.4 billones de texels/sec.
      - Vértices por segundo: 600 Millones
      - Texturas por píxel: 16
  - Soluciones
    - Reducir la información por vértice
    - Reducir el número de vértices
    - Reducir el número de polígonos (LOD)

Técnicas de aceleración

- Compresión de geometría
  - Información mínima por vértice (28 bytes sin compresión)
    - (x,y,z) coordenadas 3D (12 bytes)
    - (u,v) coordenadas de textura (8 bytes)
    - (u,v) coordenadas mapa de iluminación (8 bytes). Sin normales
  - Técnicas de compresión
    - Compartir mapas de texturas
    - Compresión estándar: duplas, incrementos, entropía
    - Reducir la exactitud
      - Ej. Motor de un coche (coordenadas 10bits -> exactitud 0.5 mm)
  - Problemas
    - Falta de exactitud en posición
    - Desplazamiento del color

Técnicas de aceleración

- Codificar la conectividad
  - malla general
    - 10 bits por coordenada para un objeto de n vértices

Coste por vértice = $30n$ bits

    - Suponiendo que en una malla general $t = 2v$ (Euler)

Coste de conectividad = $6n \log_2 n$ bits

Coste Total = $30n + 6n \log_2 n$ bits

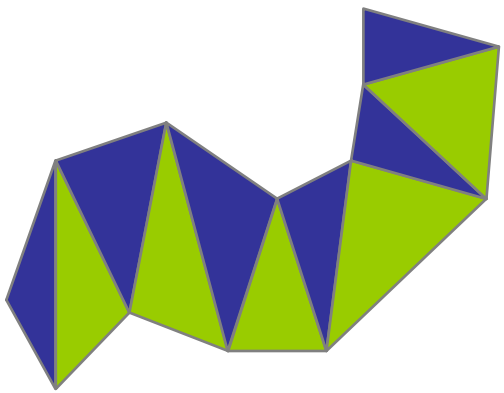
    - Rossingnac 1999

Coste por vértice = $31n$ bits
Coste de conectividad = $6n + 5n \log_2 n$ bits

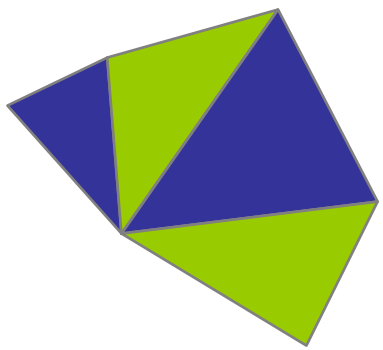
para objetos de más de 64 vértices

Técnicas de aceleración

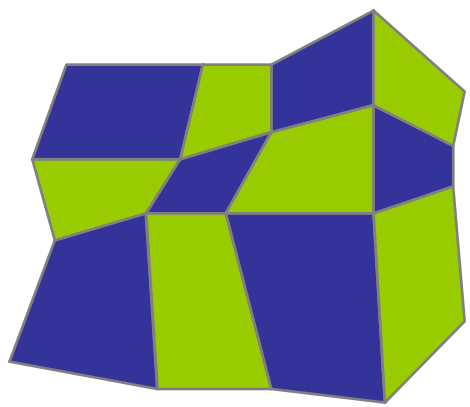
- Conectividad implícita
  - Cinta de triángulos (triangle strip)
    - para n vértices produce (n-2) triángulos conexos
  - Abanico de triángulos (triangle fan)
    - para n vértices produce (n-2) triángulos conexos
  - Malla de cuadriláteros (quadrilateral mesh)
    - para n por m vértices produce (n-1) x (m-1) cuadriláteros



Tiras de triángulos



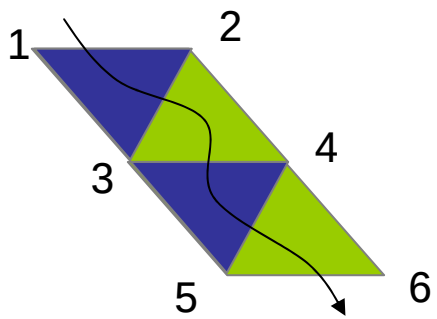
Abanico de triángulos



Malla de cuadriláteros
(mallas de elevación)

Técnicas de aceleración

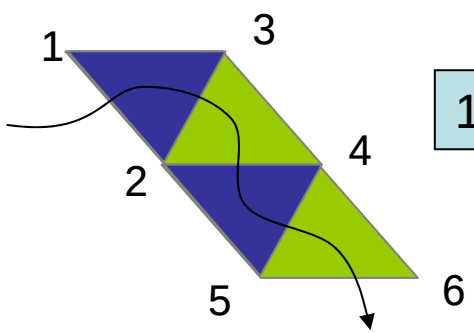
- Tiras de triángulos
  - Sin swaps (OpenGL)



123456 → Genera →

123
234
345
456

  - Con swaps (GL). Tiras generalizadas



123 SWAP 456 → Genera →

123
232
vacío
324
245
456

Técnicas de aceleración

- Coste con cintas de triángulos
  - Reduce los vértices por cinta de 3*t* a *t*+2 (siendo *t* el número de triángulos)
  - Si t_n es el número de triángulos en la n-ésima tira, el número total de vértices a enviar al procesador para dibujar un objeto formado por *s* tiras es

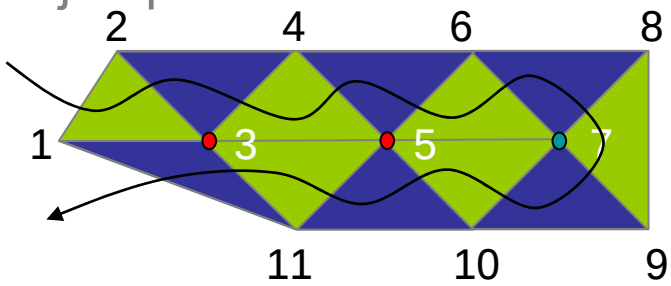
Tira 1 $\rightarrow$ $v_1=t_1+2$
Tira 2 $\rightarrow$ $v_2=t_2+2$
.....
Tira s $\rightarrow$ $v_s=t_s+2$

$$v = t+2s$$

- Objetivo
  - Encontrar tiras largas (Algoritmos de stripificación)
  - Reducir el número de swaps
    - Coste swap (1 vértice frente a iniciar nueva tira 2 vértices)

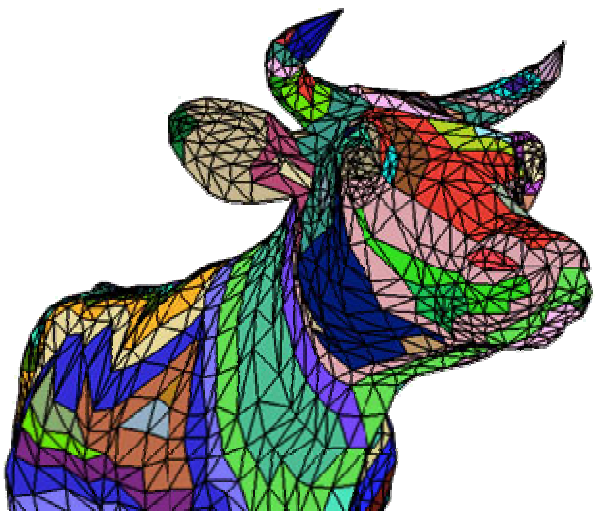
Técnicas de aceleración

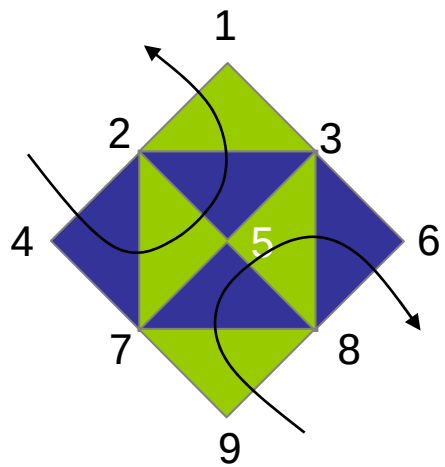
– Ejemplo



1 2 3 4 5 6 7 8 SWAP 9 SWAP 10 5 11 3 1

Malla representada con una única tira con vértices referenciados 2 veces



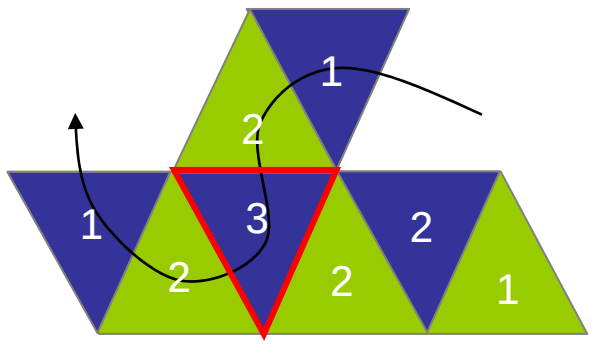


4 2 7 SWAP 5 SWAP 3 1
9 7 8 SWAP 5 SWAP 3 6

Malla que necesita dos tiras para su representación

Técnicas de aceleración

- Algoritmos de búsqueda de tiras
  - Locales
    - Las tiras se hacen crecer a través de los triángulos con el menor índice de adyacencia (número de triángulos vecinos que aún no pertenecen a ninguna tira) Akeley 1990 SGI



  - Globales
    - Idea similar excepto que los modelos no están totalmente triangulados (http://www.cs.sunysb.edu/~stripe). Evans 1996.

Técnicas de aceleración

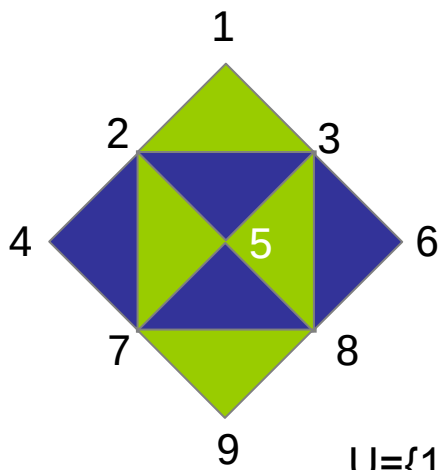
- Buffer de vértices
  - Reutilización de vértices almacenados en el sistema gráfico
    - Deering (1995)
      - Pila de tamaño 16, permite el acceso aleatorio a cualquier vértice almacenado en la pila
    - Bar-Yehuda y Gotsman (1996)
      - Un tamaño de buffer de 12.72 √n es suficiente para generar una secuencia mínima para cualquier malla de triángulos de n vértices con coste n (coste sin explotar el buffer más de 2n)
      - La malla se convierte en una secuencia de comandos de pila
        - » Push(v) el vértice es enviado a la GPU y almacenado en la pila
        - » Draw(v1,v2,v3) dibuja un triángulo con los vértices de la pila
        - » Pop(k) elimina de la pila k vértices
    - Hughes Hoppe (1999)
      - Analiza la influencia del tamaño del buffer sobre reutilización de los vértices. Las tiras más largas no siempre son las más eficientes

Técnicas de aceleración

- Teorema (Considerando una malla de triángulos como un grafo)
  - Un grafo G(V,E) puede dividirse en 3 conjuntos U, S y W de forma que ningún arco en E conecte un vértice de U con uno de W. O(n)

```
Dibuja_secuencia(M(V))
  si |V|≠ entonces
    obtener U,S,W
    salida("push("S")")
    push(pila,S)
    para todo triangulo t de M con
    un vértice en S y dos en la pila
      salida("draw("indice(v1(t)),
                   indice(v2(t)),
                    indice(v3(t))
    Dibuja_secuencia(M(U))
    Dibuja_secuencia (M(W))
    salida("pop("|S|")")
    pop(pila,|S|)
```

Salida: push(3), push(5), push(7), push(2), draw(2,3,5), draw(2,5,7), push(1), draw(1,2,3), pop(1), push(4), draw(2,4,7), pop(2), push(8), draw(3,5,8), draw(5,8,7), push(6), draw(3,6,8), pop(6), push(9), draw(7,8,9)



U={1,2,4)
S={3,5,7}
W={6,8,9}

9 vértices enviados frente a 12 en el caso de tiras

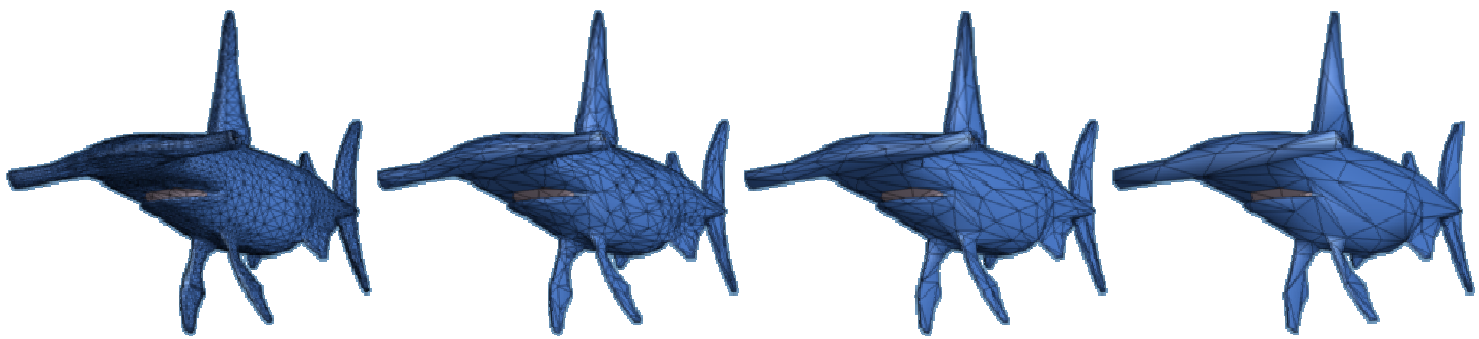
Nivel de detalle

- Modelado por nivel de detalle
  - Definición
    - Un modelo multirresolución es un modelo que almacena un amplio rango de aproximaciones de un objeto y que puede mostrar cualquiera de ellas en un momento dado
  - Ventajas
    - Permite adaptar automáticamente el nivel de detalle a los requerimientos HW (imágenes por segundo, ancho de banda, ...) y de aplicación (juegos, R.V., simulación, visualización científica, etc.)

"No tiene sentido utilizar 500 polígonos para describir un objeto que ocupa 20 píxeles...."

Nivel de detalle

- Tipos
  - Modelos discretos
    - Descripción de una secuencia de pocas mallas cada una de las cuales representa el objeto a diferente resolución
  - Modelos continuos
    - Descripción de un conjunto virtualmente continuo de mallas representando el objeto con incremento de resolución
- Aplicaciones
  - Visualización interactiva
  - Transmisión progresiva
  - Compresión de la malla
  - Edición con resolución variable

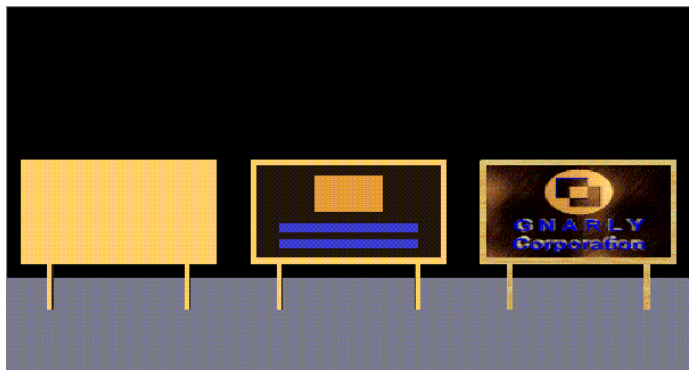


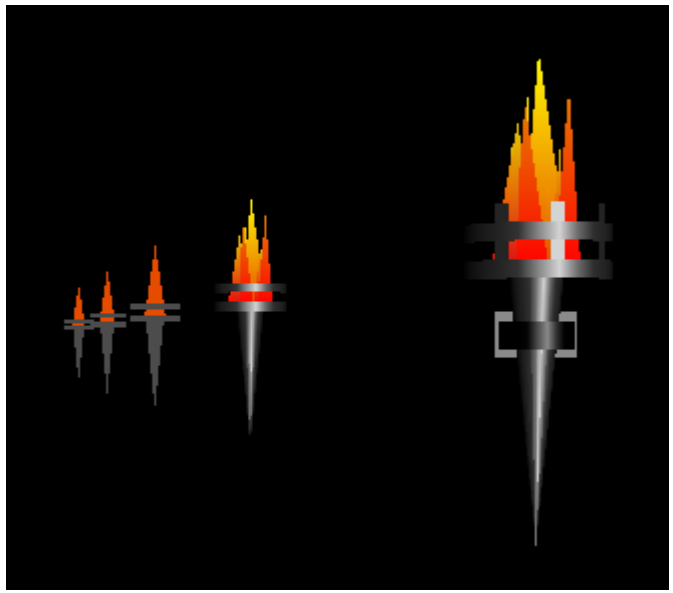
Nivel de detalle

- Modelos discretos
  - Características
    - Cada malla puede obtenerse por simplificación
    - Cada malla se asocia con un rango de niveles de detalle
    - Número limitado de representaciones
    - Tecnología estándar en Vrml y OpenInventor
  - Desventajas
    - Cada malla se almacena independientemente
    - Dificultad de adaptar el nivel de detalle a la aplicación
    - Popping
    - Resolución uniforme
  - Soluciones
    - Morphing
    - Blending

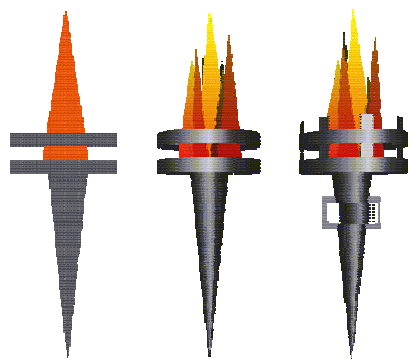
Nivel de detalle

- Ejemplo
  - VRML



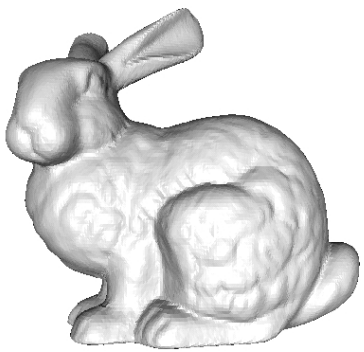


```
#VRML V2.0 utf8
LOD {
    range [ 15, 25 ]
    level [
        Inline { url "torchigh.wrl"},
        Inline { url "torchmed.wrl"},
        Inline { url "torchlow.wrl"}
    ]
}
```



Nivel de detalle

- Modelos continuos
  - Características
    - Eficiente extracción del LOD en tiempo real
    - Tamaño adecuado (entre 2 y 3 veces el modelo original)
    - Mallas continuas
    - Transición suave entre niveles

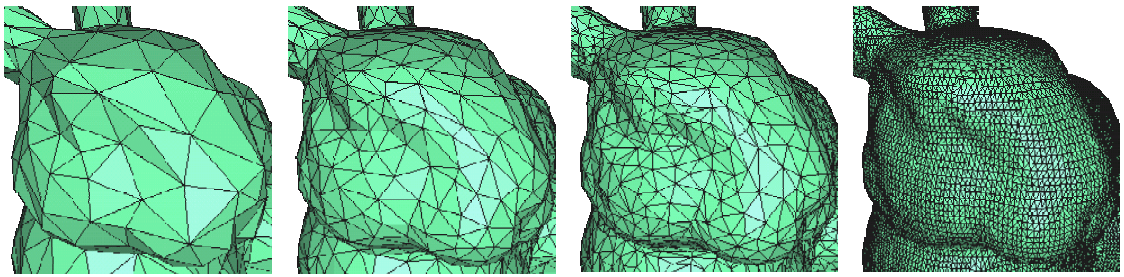


Modelo original 1.222 Mb
Progressive Meshes 3.065 Mb

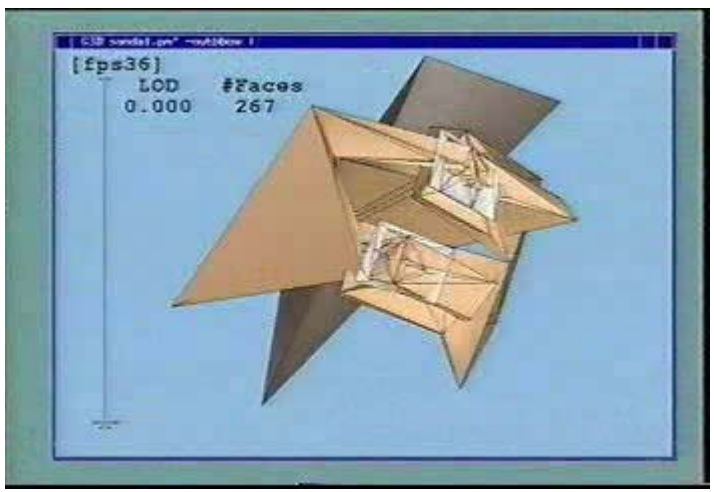
Modelo original 2.918 Mb
Progressive Meshes 7.316 Mb

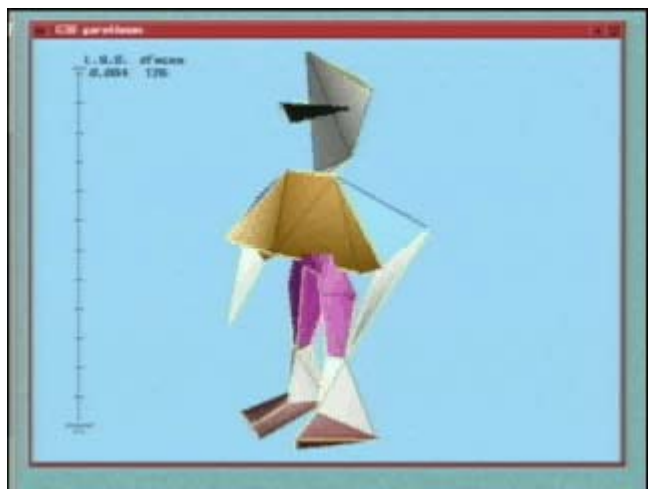
Nivel de detalle

- Tipos
  - Dominio uniforme



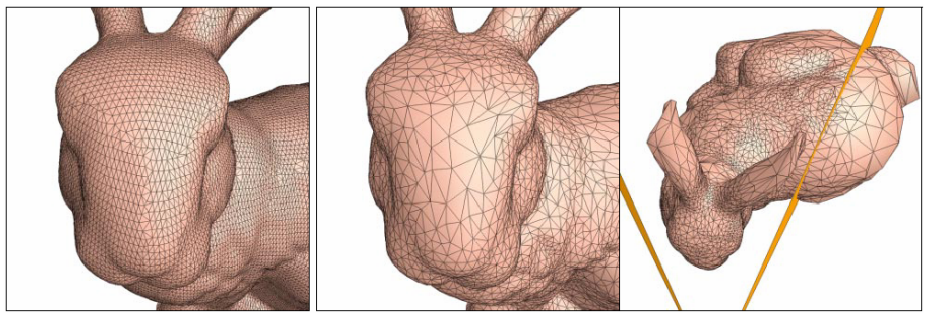
2000 Caras | 5000 Caras | 10000 Caras | 69451 Caras



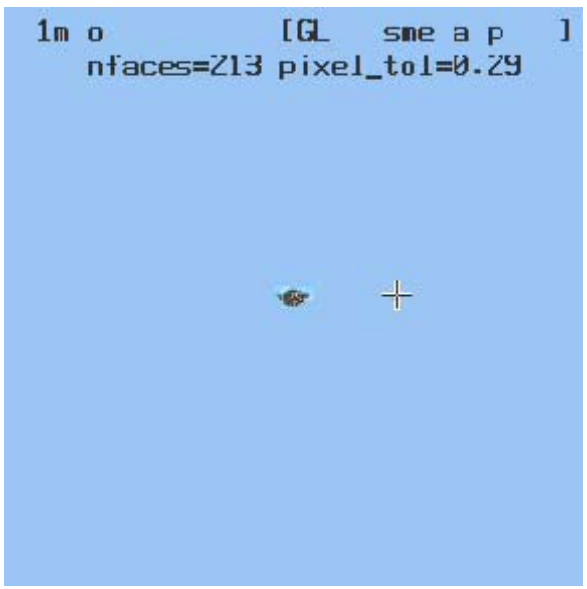


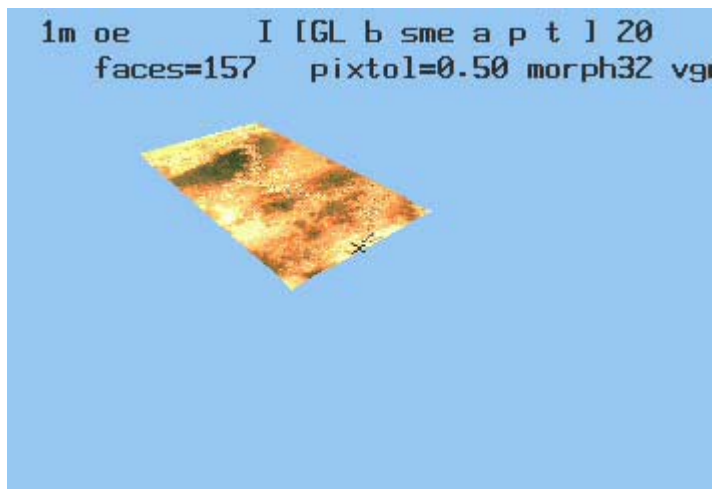
Nivel de detalle

- Dominio variable
  - Criterios de refinamiento (volumen de la vista, orientación, silueta, error en la imagen, etc.)



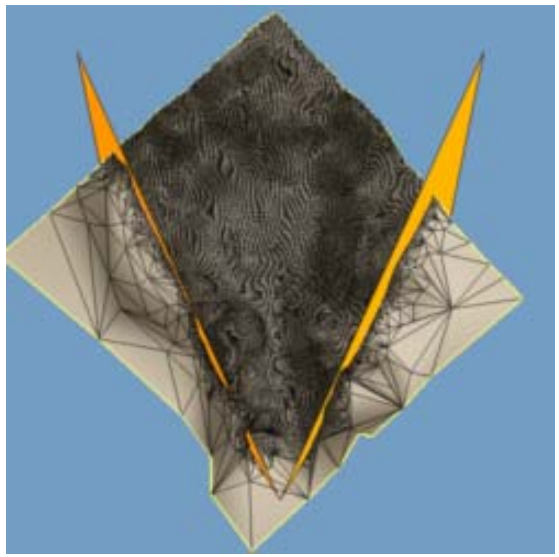
Original 69.473 caras Vista frontal y superior 10.528 caras

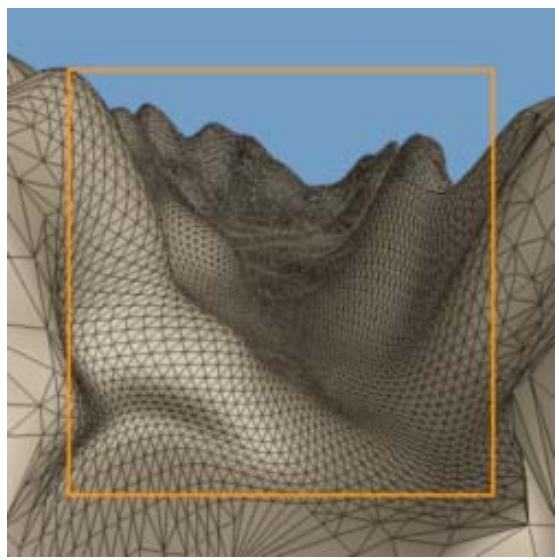




Nivel de detalle

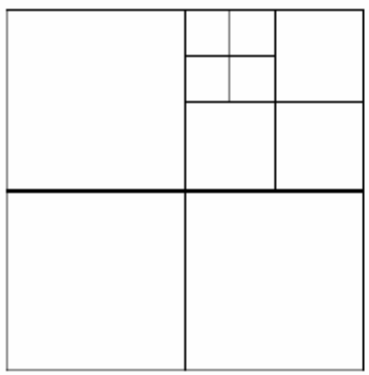
- Clasificación
  - Modelos basados en árboles
    - Basados en la subdivisión anidada de la superficie
    - Cada celda en la subdivisión se refina de forma independiente
  - Modelos de evolución
    - Basados en realización de modificaciones locales (refinamiento o simplificación)
    - Pueden obtenerse diferentes mallas combinando distintos grupos de modificaciones

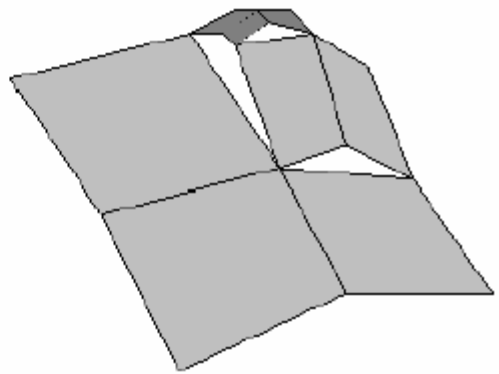




Nivel de detalle

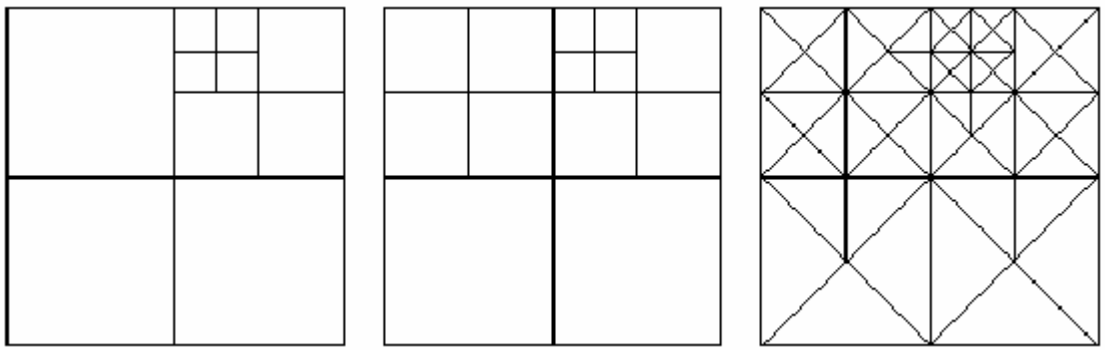
- Modelos basados en árboles
  - Quadtrees
    - Subdivisión recursiva en cuadrantes
    - Una malla formada por cuadrantes de diferentes niveles puede tener agujeros
    - Los octrees son la extensión para manejar información volumétrica

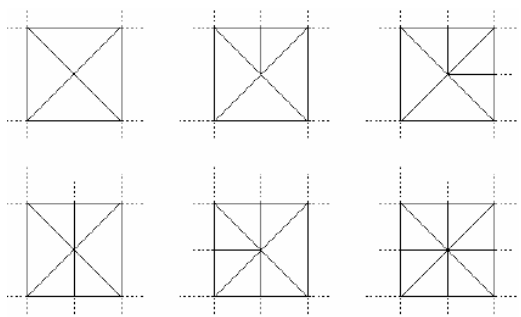


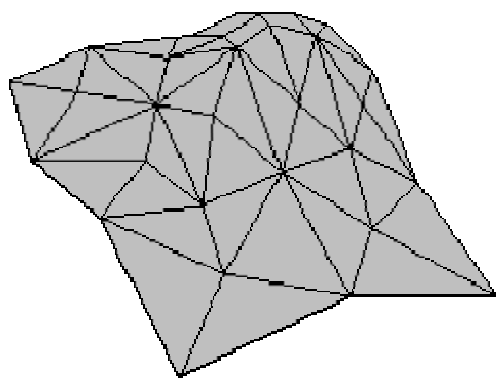


Nivel de detalle

- Quadtrees restringidos
  - Evitan los agujeros
  - Cada cuadrante pude dividirse de diferentes formas

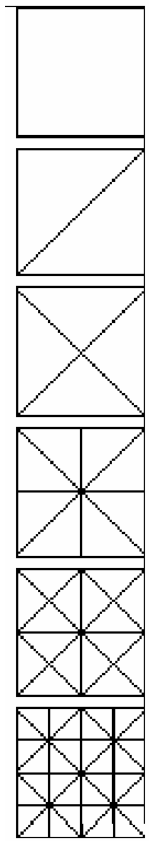


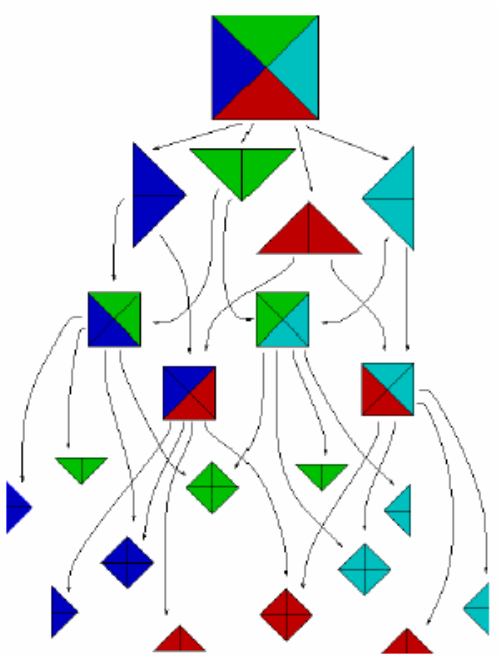


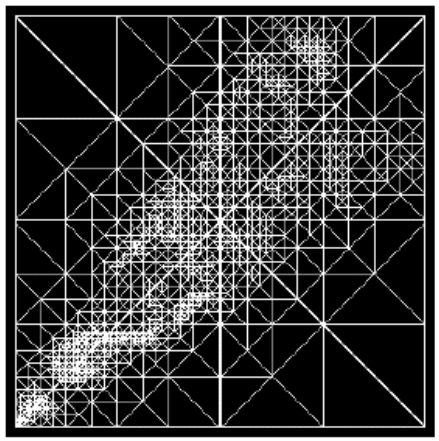


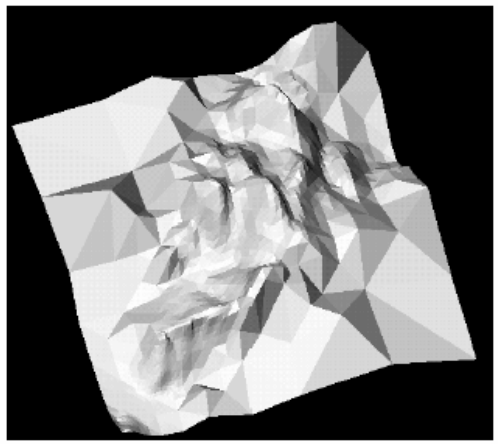
Nivel de detalle

- Bintrees
  - Se divide el dominio cuadrado por la diagonal
  - Cada triángulo se divide recursivamente en dos por su arista mayor
  - La jerarquía se describe en un árbol binario





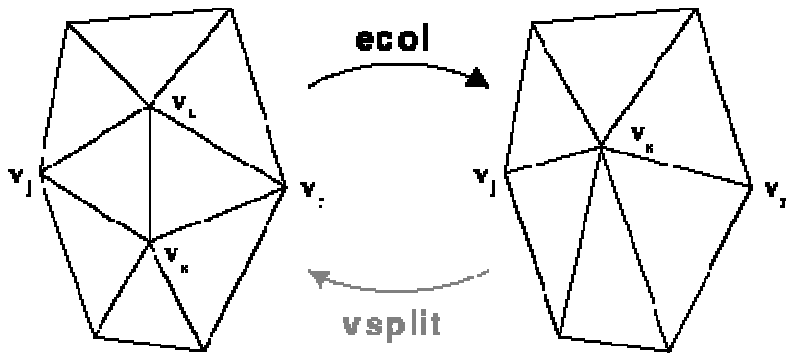


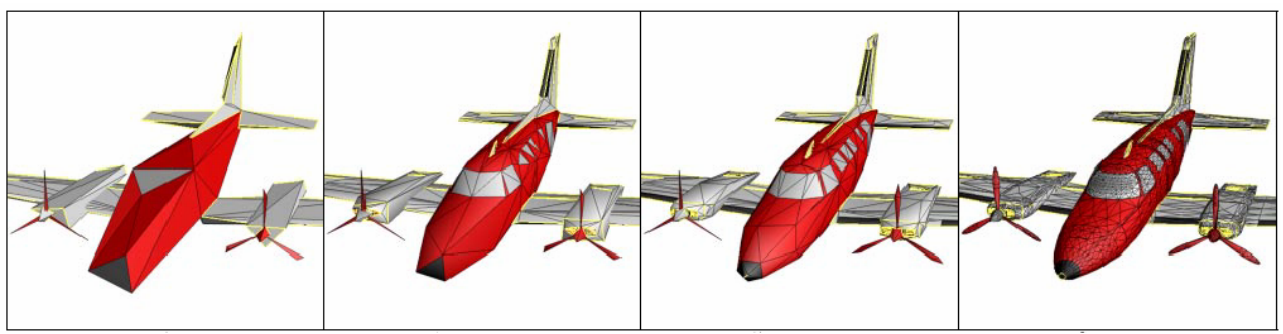


Refinamiento selectivo con el volumen de la vista e incremento de error con la distancia

Nivel de detalle

- Modelos de evolución
  - Secuencias progresivas (Progressive Meshes)
    - M0+vSplit 0+...+vSplit i+....vSplit n-1
    - El proceso es invertible
    - Resolución uniforme

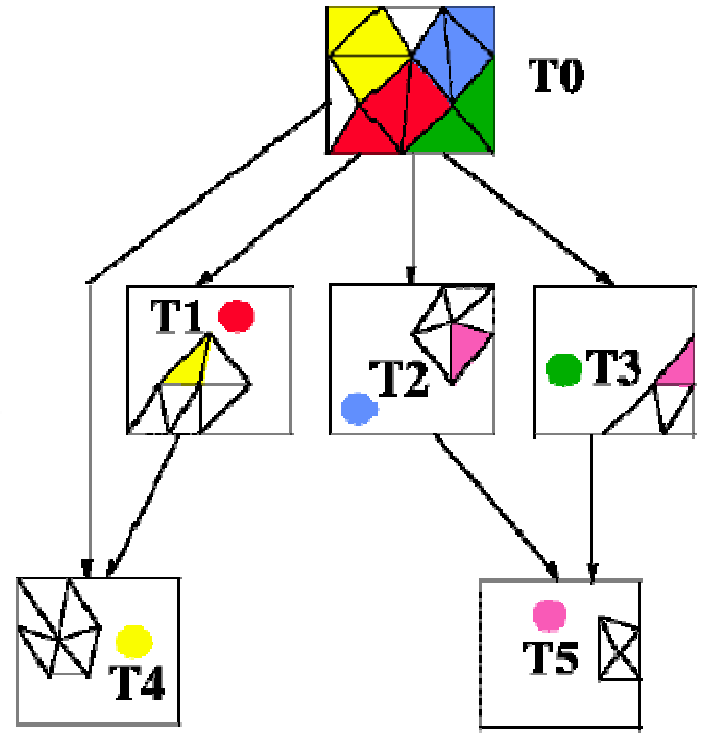


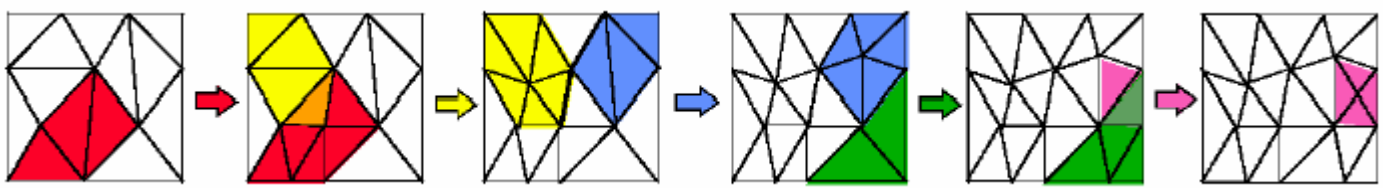


150 caras | 500 caras | 1000 caras | 13546 caras

Nivel de detalle

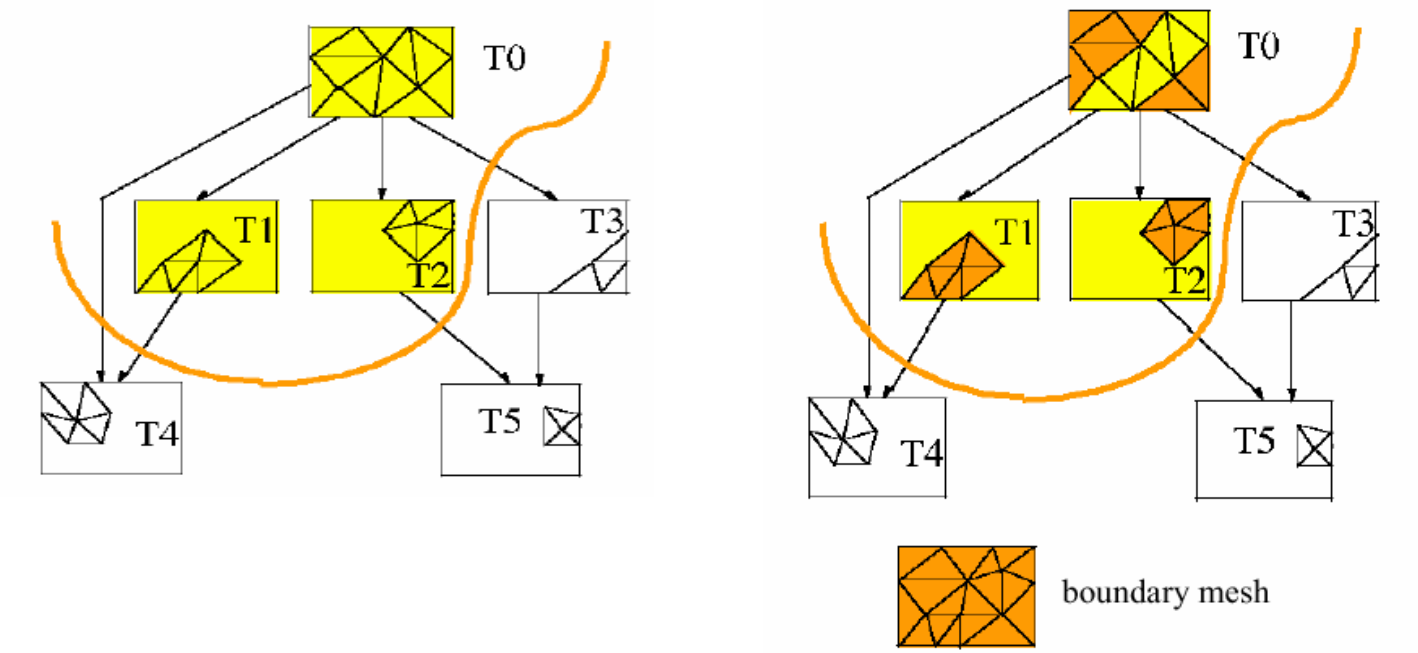
- Un esquema general (la multitriangulación MT)
  - Malla inicial +
  - Colección de modificaciones locales en orden formando un grafo acíclico dirigido (DAG)
  - Los nodos son mallas de triángulos
  - Los arcos describen el orden
  - Unas modificaciones son dependientes, otras independientes





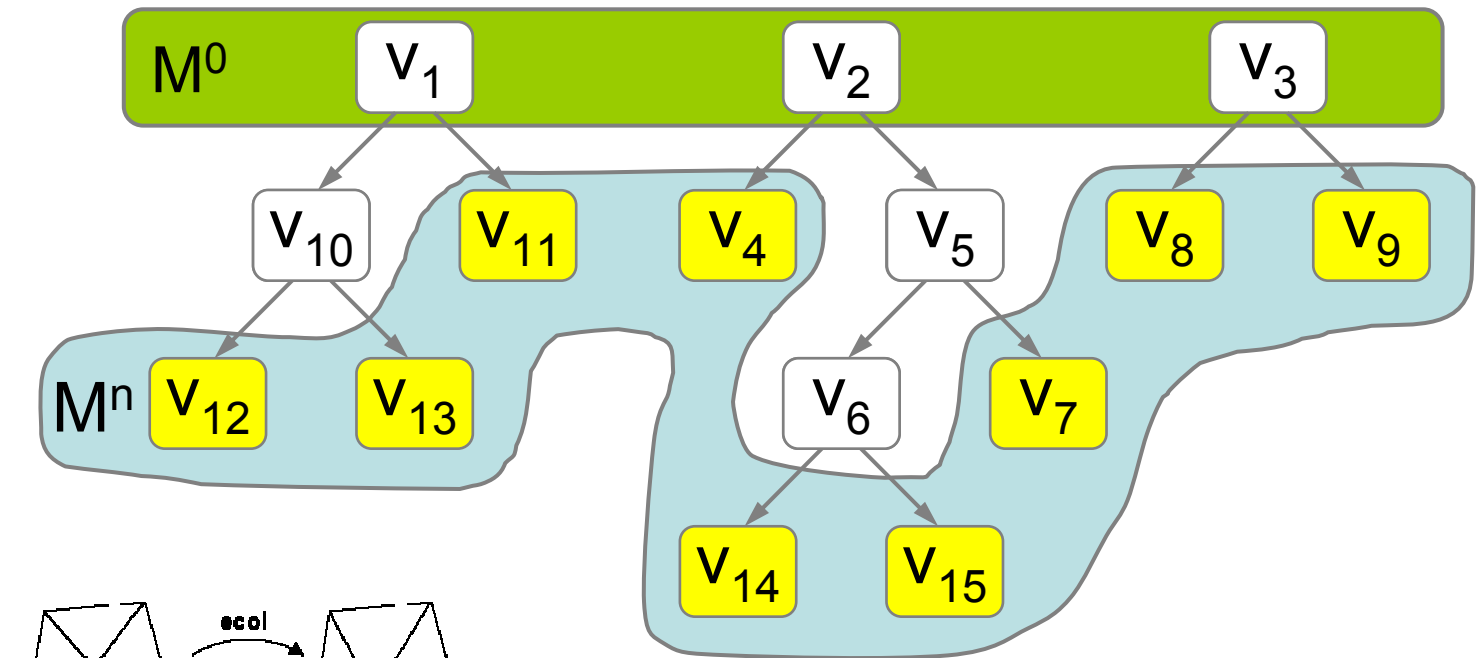
Nivel de detalle

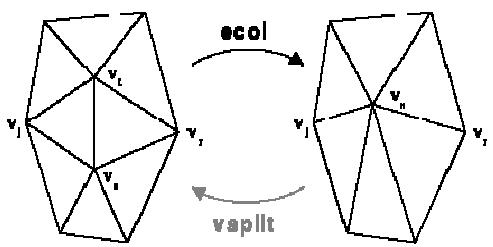
- Una subMT de una MT M es un subgrafo M' donde:
  - M' contiene la raíz
  - Si Ti pertenece a M', entonces todos los padres de Ti también pertenecen a M'
- Cualquier malla construida con triángulos de M' es una malla válida (si tenemos M entonces tenemos la malla al máximo detalle)



Nivel de detalle

- Jerarquía de vértices (VDPM)
  - Extensión de PM que permite resolución variable
  - Restringido a la utilización de un método de simplificación por eliminación de aristas

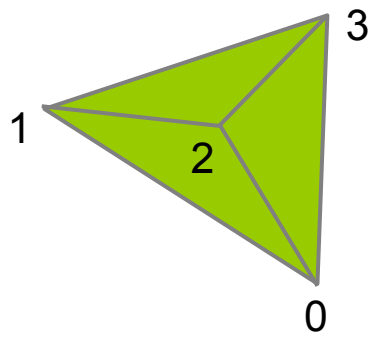




Hardware gráfico

- Vectores de vértices e índices
  - OpenGL (vertex arrays), DirectX (vertex buffers)
  - Almacenan toda la información del modelo en segmentos continuos de memoria
  - Los datos no se copian de la aplicación al *driver*, se almacenan en *buffers*. El *driver* accede a ellos a través de punteros.
  - Se utilizan menos llamadas al API (esto es menos importante con las GPUs actuales). En cada llamada se envían más datos
  - Clasificación
    - Si residen en memoria principal o en la memoria de vídeo
    - Si son sólo de escritura o de lectura/escritura
    - Si se mantienen diferentes vectores (vértices, normales, texturas, …) o si se usa un único vector con todos los datos (interleaved)

Hardware gráfico



begin n_0 p_0 n_1 p_1 n_2 p_2 end
begin n_1 p_1 n_3 p_3 n_2 p_2 end
begin n_2 p_2 n_3 p_3 n_0 p_0 end

Las *strips* suelen ser la forma más eficientes de representar mallas. Si el objeto esta formado por varias *strips* se suelen unir con triángulos degenerados

Triángulos separados

p_0 p_1 p_2 p_1 p_3 p_2 p_2 p_3 p_0	posiciones
n_0 n_1 n_2 n_1 n_3 n_2 n_2 n_3 n_0	normales

Strip

p_0 p_1 p_2 p_3 p_0	posiciones
n_0 n_1 n_2 n_3 n_0	normales

Strip interleaved

p_0 n_0 p_1 n_1 p_2 n_2 p_3 n_3 p_0 n_0	vértices

Triángulos indexados

p_0 n_0 p_1 n_1 p_2 n_2 p_3 n_3	vértices
0 1 2 1 3 2 2 3 0	índices

Strip indexada

p_0 n_0 p_1 n_1 p_2 n_2 p_3 n_3	vértices
0 1 2 3 0	índices