Simplificación de mallas para juegos

Carlos González
Dept. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus ESTCE
12071 Castellón de la Plana
cgonzale@lsi.uji.es

Jesús Gumbau
Dept. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus ESTCE
12071 Castellón de la Plana
jgumbau@lsi.uji.es

Miguel Chover
Dept. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus ESTCE
12071 Castellón de la Plana
chover@lsi.uji.es

Pascual Castelló
Dept. de Lenguajes y Sistemas Informáticos
Universitat Jaume I
Campus ESTCE
12071 Castellón de la Plana
castellp@lsi.uji.es

Resumen

Las mallas poligonales usadas en aplicaciones en tiempo real, como juegos, suelen estar formadas por una serie de submallas. Estas submallas a su vez están formadas por una serie de vértices, los cuales tienen sus atributos. El hardware gráfico requiere que la malla cumpla ciertos requisitos. Estos requisitos hacen que muchas veces haya que replicar vértices para asignarles diferentes atributos, dependiendo de las propiedades de los triángulos que los usan. Por esta razón, cuando se trabaja con mallas de juegos, el propio algoritmo simplificador tiene que tener en cuenta estas restricciones de la malla para poder ofrecer una simplificación correcta. En este artículo se presenta un método de simplificación que tiene en cuenta estas restricciones. Además, el método tiene en consideración atributos que contribuyen a una apariencia más realista en el objeto simplificado, como las coordenadas de textura y las normales.

1. Motivación

En los últimos años se ha producido un gran auge de las aplicaciones interactivas, como juegos. Los métodos de simplificación suponen un gran paso adelante para este tipo de aplicaciones. Estos métodos permiten no almacenar y procesar toda la geometría de los objetos en la escena, simplificándolos y produciendo otros objetos con menor información geométrica. Este hecho reduce el trabajo de la GPU.

Muchos métodos de simplificación se basan únicamente en la geometría de los objetos y tienen como meta producir buenos resultados geométricos. Pero recientemente han sido desarrollados métodos basados en el punto de vista del usuario. Estos métodos no pretenden producir únicamente buenos resultados geométricos, sino también resultados más afines a los originales para el usuario, eliminando, por ejemplo, partes del objeto que no son visibles para el usuario.

Pero no sólo la información geométrica es importante en los objetos finales. Los modelos suelen tener atributos adicionales, como coordenadas de textura y normales, que tienen relevancia en el aspecto final. Si estos atributos no son considerados en la simplificación se producirán grandes distorsiones en el aspecto del objeto final.

Las mallas usadas en aplicaciones interactivas suelen tener una serie de características específicas. Estas mallas suelen estar formadas por submallas. Las submallas pueden tener más de un vértice en la misma coordenada espacial, ya que cada uno de estos vértices puede estar complementado por una serie de atributos como son las normales y las coordenadas de textura.

En este artículo se presenta un método de simplificación basado en colapso de aristas que tiene en cuenta las necesidades de este tipo de aplicaciones. Así, se tienen en cuenta los vértices con más de un atributo y se preservan las fronteras. En cuanto a los atributos, el método de simplificación tiene en cuenta las texturas a la hora de asignar los costes de simplificación. Además, tras cada contracción se recalculan las coordenadas de textura, y en un post-proceso se recalculan las normales.

El método presentado es especialmente útil para la simplificación de submallas, ya que la mayoría de métodos actuales tienden a unir los vértices que coinciden espacialmente, perdiendo así información de sus atributos, como normales y coordenadas de textura. Éste método mantiene el

número de normales y coordenadas necesario para obtener una correcta apariencia final.

El resto de artículo se divide de la siguiente manera. En el apartado 2 se introduce el trabajo previo. El apartado 3 hace una explicación detallada del método. En el apartado 4 se presentan los resultados. Y en el apartado 5 se realizan las conclusiones.

2. Trabajo previo

Los algoritmos de simplificación de mallas poligonales pueden ser clasificados en las siguientes clases:

- *Eliminación de vértices* [4][20]. Estos métodos normalmente usan una selección iterativa de vértices a borrar. Una vez un vértice ha sido borrado, todas las caras que comparten ese vértice son también eliminadas y el agujero resultante se rellena con triángulos. Este tipo de algoritmos se limitan a mallas *manifold*, debido a sus esquemas de creación de nuevos triángulos.
- *Agrupamiento de vértices* [15][19]. Estos métodos se basan en una caja de inclusión dividida en celdas. Todos los vértices contenidos en una misma celda son unificados en un único vértice, y las los índices de las caras son modificados para reflejar los cambios. Estos métodos tienden a ser muy rápidos, pero la calidad de la malla resultante es relativamente baja.
- *Contracción de aristas* [6]. Estos métodos hacen uso de una selección iterativa de aristas a borrar. En cada paso una arista (o par de vértices) es seleccionada para ser borrada. Uno de los vértices de esta arista es también borrado y todas las caras afectadas son recalculadas para contener el otro vértice. Las caras degeneradas son también eliminadas de la malla.
- *Operaciones morfológicas* [17]. Estos métodos aplican operaciones morfológicas (sobre cuerpos volumétricos). Estos métodos, tales como la erosión y dilatación, son muy rápidos y ofrecen muy buenos resultados.

Los métodos más extendidos y precisos para simplificación de superficies, como por ejemplo [8][10][18], usan técnicas basadas en contracciones iterativas de aristas. Estos métodos permiten unir vértices y modificar las aristas que los usan, de modo que se preserva la conectividad con los otros vértices de las aristas.

Estos métodos asignan un peso a cada arista, de modo que la arista con menos coste es eliminada primero.

La mejora más relevante en métodos de simplificación orientados a la geometría en los últimos años fue la incorporación de los atributos, como las normales y coordenadas de textura. Por ejemplo, Hoppe extendió su trabajo inicial [11] proponiendo una nueva métrica basada en cuádricas que incorpora información de color y coordenadas de textura [12], siendo también extendido con esos atributos el algoritmo QSlim [6][7].

Cohen et al. [5] desarrollaron un algoritmo basado en colapso de aristas que pasa la posición de los vértices, las normales y los colores a mapas de normales y de textura. Este algoritmo se basa en una métrica de desviación de la textura.

Lindstrom et al. [14] manejaron el problema del parecido visual con una métrica puramente basada en imágenes. Básicamente su método determina el coste de una operación de colapso de aristas dibujando el modelo desde distintos puntos de vista. El algoritmo compara las imágenes dibujadas con las originales y añade el error en *luminancia* sobre todos los píxeles de todas las imágenes. Todas las aristas son ordenadas con el error total inducido en las imágenes y después el colapso de aristas que produce el mínimo error es elegido.

La principal ventaja de su método es que la métrica proporciona un balance entre la geometría del objeto y sus atributos de modo natural, sin que el usuario tenga que generar un peso arbitrario de ellos. Por otra parte, su principal inconveniente es el alto coste temporal.

Luebke et al. [16] presentaron un método de simplificación dependiente de la vista, usando métricas preceptuales. Posteriormente, Williams et al. [21] extendieron este trabajo a mallas iluminadas y con textura.

Zhang et al. [22] propusieron un nuevo algoritmo que tiene en cuenta la visibilidad. Su trabajo define una función de visibilidad entre las superficies del modelo y una serie de cámaras situadas en una esfera que envuelve al modelo. El número de cámaras influye en la precisión y en el coste temporal. Luebke et al. usaron hasta 258

cámaras. Para guiar el proceso de simplificación combinaron su medida de visibilidad con la métrica basada en cuádricas introducida por Garland et al. [6].

Recientemente, Lee et al. [13] introdujeron la idea de la *saliencia* como medida. Esta medida fue incorporada a la simplificación de mallas. Básicamente, su trabajo consiste en la generación de un mapa de saliencias, y luego una simplificación usando este mapa en el algoritmo del QSlim [22]. El nuevo coste de colapso de arista es el dado por la cuádrica multiplicado por la saliencia de la arista.

3. Método

El método de simplificación presentado tiene en cuenta las propiedades de las mallas usadas en juegos. Por lo que se considera la posible existencia de vértices repetidos con distintos atributos. Además, estos atributos son tratados en la simplificación. Se muestra en el Algoritmo 1 un pseudo-código del método.

```
Inicio
Para cada arista(a) del Modelo
   Colapsar arista a
   Calcular coste colapso
   Insertar la dupla (a, coste) en
   la cola q
   Deshacer colapso arista a
Fin Para

Mientras (cola q no está vacía
       o se tiene el número de
       polígonos deseados)
   Extraer de cola q la arista de
   menor coste (a)
   Colapsar arista a
   RecalculaCoordsText(a)
   Recalcular coste de aristas
   vecinas a a y actualizar
   posición en cola q
Fin mientras
Fin

RecalculaNormales(Modelo)
Fin
```

Algoritmo 1. Pseudo-código del método.

3.1. Métrica de error

El modelo usado para probar el método de simplificación es un modelo basado en el punto de vista del usuario [3]. Su objetivo es producir simplificaciones con el mayor parecido visual posible. Está basado en el punto de vista y aplica una métrica de error basada en la *Teoría de la Información*, la *entropía de un punto de vista*. La entropía de un punto de vista ha sido utilizada principalmente para la selección de los mejores puntos de vista de una escena u objeto. La entropía de un punto de vista se obtiene a partir de la distribución de áreas proyectadas de los polígonos. Para calcular las áreas proyectadas se analiza el *framebuffer* mediante un histograma.

Para poder simplificar uniformemente un objeto se utilizan varios puntos de vista distribuidos equidistantemente.

La variación de la entropía de cada uno de estos puntos de vista tras una operación de colapso se utiliza para calcular el error introducido por dicha operación. La operación de colapso que se aplica es el *colapso de arista medio*, donde el vértice resultante es cualquiera de los dos vértices de la arista colapsada. Además, se elige el mejor colapso de arista medio. Este colapso se obtiene calculando los dos colapsos y determinando el que produce un menor cambio en curvatura de la malla.

Se muestra en la Ecuación 1 la fórmula de la entropía de un punto de vista v. Donde H_v es la entropía de un punto de vista v, donde N_f es el número de polígonos del modelo, ai es el área proyectada del polígono i y a_t es el área total proyectada.

$$H_v = \sum_{i=0}^{N_f} \frac{a_i}{a_t} \log \frac{a_i}{a_t}, \qquad (1)$$

El proceso de simplificación es un proceso iterativo. Inicialmente, se calcula el error cometido en un colapso de arista y se almacena en una cola de prioridad. Posteriormente, se extrae de la cola el colapso que tiene menor coste. Este proceso se repite hasta que se consigue el número de polígonos deseado o no hay más colapsos en la cola.

No sólo la información geométrica es importante en los objetos resultantes en la simplificación. Los modelos normalmente contienen atributos adicionales, como coordenadas de textura y normales. Las aplicaciones interactivas, como juegos o mundos virtuales, necesitan mostrar objetos bien texturizados, ya que las texturas juegan un papel muy importante en la apariencia final del objeto.

Si no se tiene en cuenta la textura en el método de simplificación, se obtendrán distorsiones en la misma en el objeto final. Para solucionar este problema se hace uso de la extensión de la métrica de error para tener en cuenta la textura [9]. Esta extensión se basa en los bordes existentes en la textura. La detección de los bordes está basada en el método elaborado por Canny [1][2]. Tras la obtención de los bordes se comprueba si las aristas colisionan en el espacio de la textura con alguno de estos bordes. Si es así, cada arista que colisione se penaliza para provocar un retraso en su colapso. Con ello se obtiene una modificación en el orden en el que se colapsan las aristas. De modo que las zonas del modelo con grandes cambios en la textura son simplificadas más tarde. En la Figura 1 se muestran el modelo Ojo, su textura y los bordes de la misma, detectados por el método de detección de bordes. Las figuras 2 y 3 muestran tres niveles de simplificación distintos. Se pueden apreciar las diferencias entre aplicar la extensión y no aplicarla. Aplicando la extensión se obtienen resultados mucho más fieles a la apariencia original del objeto.

3.2. Solución a los agujeros

Las mallas usadas en juegos presentan distintos vértices con la misma localización espacial. Esta característica pude provocar la aparición de agujeros al simplificar una arista que contenga uno de estos vértices.

Para solucionar este problema la estrategia de simplificación hace uso de los conceptos *arista real*, *arista gemela* y *arista falsa*. La arista real se refiere a la arista colapsada, mientras que la arista gemela es una compuesta por dos vértices con las mismas coordenadas espaciales que los vértices de la arista real.

Figura 1. Modelo Ojo (izquierda), textura del modelo (centro) y borders detectados.

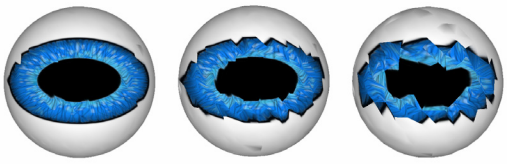

Figura 2. Modelo Ojo simplificado al 75% (izquierda), 50% (centro) y 25% (derecha) sin aplicar la extensión de texturas.

Figura 3. Modelo Ojo simplificado al 75% (izquierda), 50% (centro) y 25% (derecha) aplicando la extensión de texturas.

La Figura 4 presenta un ejemplo de este tipo de aristas, así como un paso de simplificación. La arista v_a,v_b se elige para ser colapsada (es la arista real). El simplificador decide que la arista v_c,v_d es la arista gemela, y que debe ser colapsada para evitar un agujero.

No siempre es suficiente con contraer la arista real y su arista gemela para evitar agujeros en las mallas con discontinuidades. La tercera imagen en la Figura 4 muestra como, después de contraer las aristas, queda visible un agujero en la malla. Esto es debido a que el algoritmo de simplificación está basado en colapso de aristas, pero no hay ninguna arista a colapsar que pueda tapar el agujero. Encontramos un vértice aislado, que debe conectarse a otro vértice (*vértice falso*), creando una arista falsa que puede ser colapsada para tapar el agujero. Este vértice falso estará topológicamente desconectado en el modelo

original y será usado como vértice final del colapso. Cuando añadimos el vértice falso, sus coordenadas espaciales y sus huesos asignados son clonados desde el vértice apropiado de la arista real. Para evitar agujeros en tiempo de ejecución es necesario aplicar todos estos pasos de simplificación seguidos. Se presenta un pseudo-código en el Algoritmo 2.

Este método es muy útil también para la simplificación de submallas, ya que la mayoría de métodos tienden a unir los vértices como paso previo a la simplificación, perdiendo con ello información sobre atributos adicionales a los vértices.

```
Mientras (ListaAristas ≠ Ø)

   arReal=extraer(ListaAristas)
   añadirListaSimplif(arReal)

   Si(∃Gemela(arReal,
              ListaAristas))
       arGem= Gemela(arReal,
                     ListaAristas)
       añadirListaSimplif(arGem)
   Fin Si

   Mientras (∃VertAislado(arReal,
             ListaVertices))
       vertFalso=
             VertAislado(arReal,
                    ListaVertices)
       arFalsa=ArFalsa(arReal,
                    vertFalso)
       calculaAtributos(vertFalso)
       añadirListaVerts(vertFalso)
   Fin Mientras

Fin Mientras
```

Algoritmo 2. Pseudo-código de contracción de aristas.

3.3. Recálculo de coordenadas de textura

Con el fin de obtener una mejor distribución de la textura sobre el objeto simplificado, se recalculan las coordenadas de textura de cada vértice modificado tras cada paso de simplificación. Con ello se pretende obtener una mejor apariencia del objeto simplificado. La nueva coordenada de textura se calcula basándose en el desplazamiento del nuevo vértice, usando una interpolación lineal.

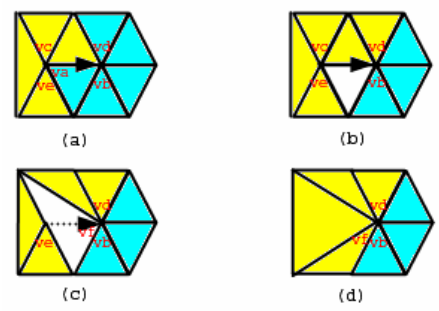

Figura 4. El colapso de la arista $v_a \rightarrow v_b$ (arista real) fuerza los colapsos de las aristas $v_c \rightarrow v_d$ (arista gemela) y $v_e \rightarrow v_f$ (arista falsa).

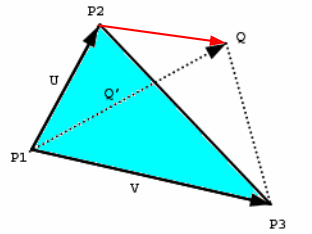

Figura 5. Desplazamiento del nuevo vértice. P_2 se colapsa en Q.

Para obtener el desplazamiento que debe ser aplicado a las coordenadas de textura del vértice modificado proponemos el sistema de ecuaciones de la Ecuación 2.

$$\begin{aligned} Q'_x &= \alpha U_x + \beta V_x + \gamma N_x \\ Q'_y &= \alpha U_y + \beta V_y + \gamma N_y \\ Q'_z &= \alpha U_z + \beta V_z + \gamma N_z \end{aligned} \qquad (2)$$

teniendo que $\vec{Q}' = \vec{Q} - \vec{P}^1, \vec{U} = \vec{P}^2 - \vec{P}^1$, $\vec{V} = \vec{P}^3 - \vec{P}^1$, donde $\{\vec{P}^1, \vec{P}^2, \vec{P}^3\}$ son los tres vértices del triángulo modificado, $\vec{Q}$ es la nueva posición del vértice modificado y $\vec{N}$ es la normal del triángulo.

Resolviendo el sistema de ecuaciones de la Ecuación 2 obtenemos α, β y γ.

α, β y γ son las coordenadas de P expresadas en el sistema de coordenadas del triángulo. También expresan cuánto se ha desplazado el vértice modificado en unidades del sistema de coordenadas del triángulo, por lo que pueden ser usados para calcular la nueva coordenada de textura del vértice modificado. Por lo tanto, α y β son la cantidad de desplazamiento que se necesita en los vectores u y v respectivamente para pasar de P_1 a Q (Figura 5). Con ello calculamos la nueva coordenada de textura (Ecuación 3).

$$T_u^{res} = \alpha(T_u^2 - T_u^1) + \beta(T_u^1 - T_u^1) + T_u^1$$
$$T_v^{res} = \alpha(T_v^2 - T_v^1) + \beta(T_v^3 - T_v^1) + T_v^1 \quad (3)$$

donde $\vec{T}_1$, $\vec{T}_2$ y $\vec{T}_3$ son las coordenadas de textura originales de los tres vértices, $\vec{T}_{res}$ es la nueva coordenada de textura del vértice modificado y α, β y γ son los coeficientes calculados.

Hay que tener en cuenta que únicamente usamos α y β de la Ecuación 2 porque las coordenadas de textura son vectores bidimensionales contenidos en el plano formado por el triángulo. Como γ es el desplazamiento a lo largo del vector de la normal del triángulo, no lo necesitamos.

Se muestra un ejemplo del la interpolación de coordenadas de textura en la Figura 6, en la que el objeto se presenta simplificado al 30% de su geometría original.

3.4. Recálculo de normales

El valor final de las normales influirá en el aspecto visual que dé el objeto simplificado. Por ello, una vez simplificado el objeto, aplicamos un proceso de recálculo de normales. En este proceso se tendrá que tener en cuenta que algunos vértices necesitan tener normales por vértice y otros por cara. Por ejemplo, en un cubo los vértices han de tener normales por cara, para que se obtenga un aspecto de esquina. Si se asignaran normales por vértice cada esquina presentaría un aspecto redondeado.

El proceso a seguir es el siguiente:

- Se despliega el modelo, de modo que cada triángulo tendrá unos índices de normales distintos.
- A cada vértice se le asigna la normal por cara del triángulo.
- Para cada vértice se recorren los demás vértices del modelo. Si coinciden espacialmente y las caras a las que pertenecen forman un ángulo menor a uno dado, se le suma la normal de ese vértice y los índices de normales pasan a ser el mismo.

Se presenta un pseudo-código en el Algoritmo 3. Y las figuras 7 y 8 muestran un ejemplo del recálculo de normales en el modelo simplificado.

Figura 6. Modelo original (izquierda), simplificación del objeto sin aplicar interpolación de coordenadas de textura (centro) y simplificación del objeto aplicándola.

```
Función Despliega
Inicio
contador= 0
Para ∀ Vértice ∈ Modelo hacer
   IndiceN(Vértice)= contador
   contador++
   Normal(Vértice)= Normal(Cara)
Fin Para
Fin


Función CalculaNormales
Inicio
Para ∀ Vértice1 ∈ Modelo hacer
Para ∀ Vértice2 ∈ Modelo hacer
   Si Vértice1≠Vértice2 y
   Coordenadas(Vértice1)==
   Coordenadas(Vértice2)
       Si Ángulo(Cara1,Cara2)<
                     EPSILON
              Normal(Vértice1)+=
              Normal(Cara2)

              IndiceN(Vértice2)=
              IndiceN(Vértice1)
       Fin Si
   Fin Si
Fin Para
Fin Para
Fin
```

Algoritmo 3. Pseudo-código de recálculo de normales

4. Resultados

Con el método de simplificación desarrollado se puede observar que se obtienen resultados que mantienen la apariencia inicial en el objeto, ya que aparte de las consideraciones geométricas se han tenido en cuenta también los atributos que afectarán al aspecto final.

Las figuras 9 y 10 muestran dos ejemplos de objetos simplificados con nuestro método. La Figura 9 muestra el modelo Ninja simplificado al 50% de su geometría original, mientras que la Figura 10 muestra dos niveles de simplificación de modelo Coche de carreras.

Se presenta además una tabla de tiempos de varias simplificaciones, indicando el número de polígonos inicial y final (Tabla 1). El mayor coste temporal se produce por el cálculo de la entropía, debido a la cantidad de redibujados de la escena que se hacen en este método de simplificación basado en el punto de vista.

Los tiempos han sido tomados sobre un Intel Pentium 4 CPU 3.01GHz, con 1GB de RAM y una tarjeta gráfica NVIDIA GeForce 7800 GTX, usando Microsoft Windows XP Profesional.

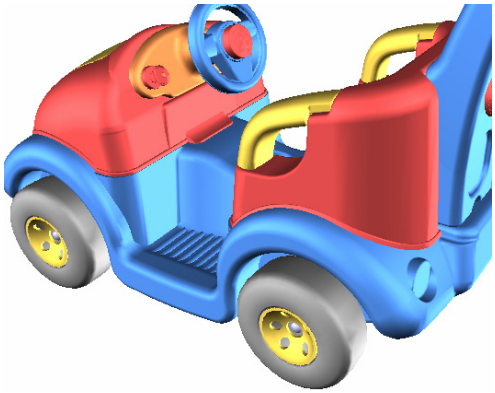

Figura 7. Modelo de cochecito de juguete original.

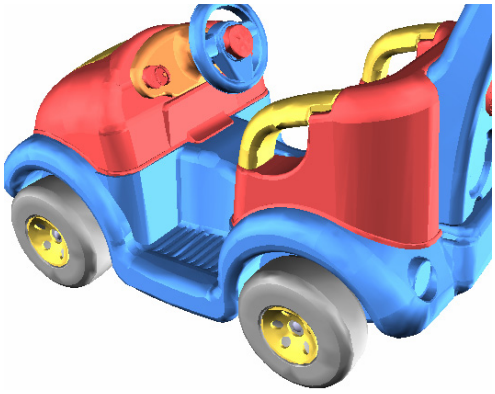

Figura 8. Modelo de cochecito de juguete simplificado al 50% de su geometría con las normales recalculadas.

Triángulos		Tiempo
Original	Final	Segundos
815	100	12.27
4 698	500	93.45
4 806	1 200	97.33
6 592	500	153.18
8 468	500	226.78
10 474	1 000	300.07
13 810	1 000	453.34

Tabla 1. Tabla de tiempos.

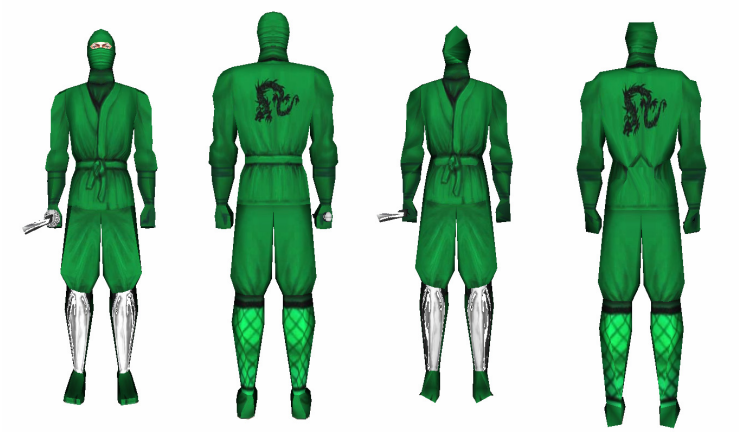

Figura 9. Modelo Ninja. De izquierda a derecha: modelo original (1008 triángulos) por delante, modelo original por detrás, simplificación al 50% (504 triángulos) por delante y simplificación al 50% por detrás.

Figura 10. Modelo Coche de carreras. De izquierda a derecha: modelo original (8345 triángulos), simplificación al 66% (5506 triángulos) y simplificación al 33% (2753 triángulos).

5. Conclusión

Se ha presentado un método de simplificación de mallas poligonales útil para juegos, ya que considera las características de las mallas utilizadas en este tipo de aplicaciones. Este método intenta preservar al máximo la apariencia final del objeto simplificado, teniendo en cuenta que los vértices que coinciden espacialmente pueden estar complementados por atributos distintos.

Además, este método tiene en consideración los atributos que van a participar en el aspecto final, como son las normales y las coordenadas de textura. Para ello, se tienen en cuenta las coordenadas de textura en la propia métrica y se recalculan tras cada paso de simplificación. Además, en un post-proceso se recalculan las normales.

Cabe destacar también que este método realiza una simplificación de submallas sin realizar grandes distorsiones en la apariencia final, ya que la manera en la que se tratan los vértices repetidos proporciona un método para ello. Esto es de especial relevancia para aquellos modelos que necesitan ser simplificados localmente, ya que la mayoría de métodos existentes tienden a unir sus vértices como pre-proceso de la simplificación, perdiendo así información sobre los atributos adicionales a los vértices, como son las normales y las coordenadas de textura. Con este método se preservan el número de coordenadas de textura y normales necesario para obtener una buena apariencia final en el objeto simplificado.

Agradecimientos

Este trabajo ha sido financiado por la Universidad Jaume I (PREDOC/2005/12) el gobierno español (TIN2005-08863-C03-03, TIN2004-07451-C03-03) y la Unión Europea (IST-2-004363).

Referencias

[1] Canny, J. A variational approach to edge detection. In AAAI-83. 1983.

[2] Canny, J. F. A computational approach to edge detection. IEEE Trans. Pattern Analysis and Machine Intelligence, 679-698. 1986.

[3] Castelló, P., Sbert, M. Chover, M. Feixas, M. Viewpoint Entropy-driven Simplification. Proc. of 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, 249-256. 2007.

[4] Ciampalini, A., Cignoni, P., Montani, C., and Scopigno, R. Multiresolution decimation based on global error. Technical report, Centre National de la Recherche Scientifique, Paris, France, 1996.

[5] Cohen, J., Olano, M., Manocha, D. Appearance preserving simplification. Proc. SIGGRAPH '98, vol. 32, pages 115-122. 1998.

[6] Garland, M. and Heckbert, P. Surface simplification using quadric error metrics. In SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

[7] Garland, M., Heckbert, P. S. Simplifying surfaces with color and texture using quadric error metrics. VIS '98: Proc. of the conference on Visualization '98. IEEE Computer Society Press, pages 263-269, Los Alamitos, CA, USA, 1998.

[8] Garland, M. Multiresolution Modeling: Survey & Future Opportunities. State of the Art Reports of EUROGRAPHICS'99, vol. 14 (4), pages 111-131. 1999.

[9] González, C., Castelló, P., Chover, M. A texture-based metric extensión for simplification methods. 2ndInternational Conference on Computer Graphics Theory and Applications (GRAPP' 07), pages 69-76. 2007.

[10] Heckbert, P. and Garland, M. Survey of polygonal surface simplification algorithms. Technical report, Multiresolution Surface Modeling Course Notes of SIGGRAPH'97, 1997.

[11] Hoppe H. Progressive meshes. SIGGRAPH '96: Proc. of the 23rd annual conference on Computer graphics and interactive techniques, pages 99-108. New York, USA, 1996.

[12] Hoppe, H. New quadric metric for simplifying meshes with appearance attributes. VIS '99: Proc. of the conference on Visualization '99. IEEE Computer Society Press, pages 59-66. Alamitos, CA, USA, 1999.

[13] Lee C.H., Varshney A., Jacobs D.W. Mesh saliency. ACM Trans. Graph. 24, 3, pages 659-666. 2005.

[14] Lindstrom, P., Turk, G. Image-driven simplification. ACM TOG 19, 3, pages 204-241. July 2000.

[15] Low, K. and Tan, T. Model simplification using vertex-clustering. In SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics, pages 75–ff. ACM Press, 1997.

[16] Luebke, D.P., Hallen, B. Perceptually-driven simplification for interactive rendering. Proc. of the 12th Eurographics Workshop on Rendering Techniques, pages 223-234. London, UK, 2001.

[17] Nooruddin, F. and Turk, G. Simplification and repair of polygonal models using volumetric techniques. IEEE Transactions on Visualization and Computer Graphics, 9(2):191–205, 2003.

[18] Puppo, E. and Scopigno, R. Simplification, lod and multiresolution - principles and applications. Tutorial Notes of EUROGRAPHICS' 99, 16(3), 1997.

[19] Rossignac, J. and Borrel, P. Multi-resolution 3d approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, Modeling in Computer Graphics: Methods and Applications, pages 455–465. Springer-Verlag, 1993.

[20] Schroeder, W. J. A topology modifying progressive decimation algorithm. In VIS '97: Proceedings of the 8th conference on Visualization '97, pages 205–ff. IEEE

Computer Society Press. Los Alamitos, CA, USA, 1997.

[21] Williams N., Luebke D., Cohen J.D., Kelley M., Schubert B. Perceptually guided simplification of lit, textured meshes. Proc. of the 2003 symposium on Interactive 3D graphics. ACM Press, pages 113-121. New York, NY, USA, 2003.

[22] Zhang E., Turk G. Visibility-guided simplification. Proc. of IEEE Visualization 2002, vol. 31, pages 267-274. Nov. 2002.