ALGORITMO DE SIMPLIFICACIÓN DE SUPERFICIES UTILIZANDO LA MÉTRICA DEL ERROR CUÁDRICO (QEM)

Surface simplification using quadric error metrics

Lic. Jhonny Roberto Felipez Andrade
jrfelizamigo@yahoo.es

INTRODUCCIÓN

Muchas aplicaciones de gráficos por computadora, requieren complejos modelos de gráficos altamente detallados (ver Figura 4). Sin embargo, el nivel de detalle que necesite actualmente una determinada aplicación, puede variar considerablemente. Para controlar el tiempo de proceso, a menudo es deseable utilizar solamente las aproximaciones a este modelo, en lugar de utilizar todo el modelo completo con sus excesivos detalles.

Este articulo muestra la implementación de uno de los primeros algoritmos de simplificación de mallas, denominado el Algoritmo de Simplificación de Superficies utilizando la Métrica del Error Cuádrico [1].

Garlan [1] plantea, contraer iterativamente pares de vértices (v_1, v_2) ® $\tilde{v}$, moviendo los vértices v_1 y v_2 a la nueva posición de $\tilde{v}$, conectando todas las aristas incidentes de v_1 y eliminando el vértice v_2. Por otro lado este algoritmo a diferencia de otros algoritmos, contempla dos tipos de contracción de vértices presentadas en la Figura 1 y Figura 2.

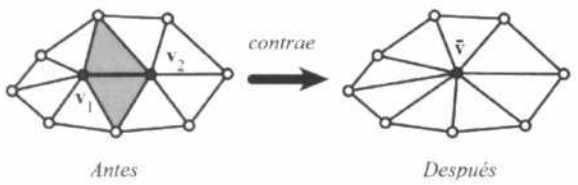

Figura 1: Contracción de aristas.

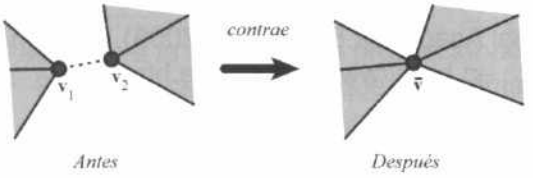

Figura 2: Contracción de no aristas.

Para elegir al par de vértices que serán contraídos, se toma en cuenta al par que tenga al vértice de menor ***error cuádrico***.

$$\mathbf{v}^T\mathbf{Q}\mathbf{v} = q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 + 2q_{23}yz + 2q_{24} + q_{33}z^2 + 2q_{34}z + q_{44}$$

Este material servirá como material de estudio de los alumnos de la materia de Programación Grafica.

Algoritmo

Paso 1:

- *Calcular Q para todos los vértices iniciales.*

Para cada cara (triángulo), encuentre la ecuación general del plano:
$ax + by + cz + d = 0$ donde $a^2 + b^2 + c^2 = 1$

Luego construya la matriz Kp de 4 x 4 para cada triángulo del modelo.

$$\mathbf{K_p} = \mathbf{pp}^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

Seguidamente genere la matriz Q de 4 x 4 para cada vértice v:

Q = suma de todos los Kp donde el plano p = [a b c d]^T pertenece al conjunto de planos que intersectan al vértice v.

Para la implementación de este algoritmo, se puede considerar las siguientes estructuras:

- *vertices*: Almacena los vértices del modelo.
- *caras*: Almacena las caras o triángulos del modelo.
- *Q* : Almacena los cuádricos de cada vértice.
- *errores*: Almacena a todos los pares vértices (v_1, v_2) del modelo, acompañado del nuevo vértice (v_barra) y del error cuádrico de este par (error).

Inicialice el Q de cada vértice:

```
for i = 1 to #vertices
   Q[i]  =  0;
```

Genere Q para cada vértice:

```
for i = 1 to #caras
   for j = 1 to 3 {
      v = caras[i].vertice[j];
      caras[i].obtiene_Q();
      Q[v] = Q[v] + caras[i].Q();
   }
```

Paso 2:

➢ *Seleccione todos los pares válidos.*

Para obtener el vértice alternativo $\bar{v}$ entre el par de vértices (v_1, v_2), se presenta estos dos casos. Si el determinante de la matriz Δ(q) (delta_q) es distinto de cero, halle $\bar{v}$ mediante:

$$\bar{\mathbf{v}} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Si el determinante es igual a cero, entonces $\bar{v}$ puede tomar los siguientes valores: $\mathbf{v}_1$, $\mathbf{v}_2$ o $(\mathbf{v}_1 + \mathbf{v}_2)/2$, dependiendo del mínimo error cuádrico de los tres.

Una vez que encuentre $\bar{v}$ obtenga nuevamente la métrica del error cuádrico de $\bar{v}$.

Obtiene los errores de cada par de vértices (v_1, v_2):

```
selecciona_pares() {
 for i = 1 to #caras
  for (j,k) = (1,2),(1,3),(2,3) {
     v1 = min(caras[i].vertice[j],
        caras[i].vertice[k]);
     v2 = max(caras[i].vertice[j],
        caras[i].vertice[k]);
     // Si no existe el par (v1,v2),
     // se inserta este par en errores
     if (Not errores.existe(v1, v2)) {
      <v1,v2,v_barra,error> =
        calcula_error(v1, v2);
      errores.insertar(
        v1, v2, v_barra, error)
     }
}
```

Calcula el error del un par de vértices en base al mínimo error cuádrico de v_1 o v_2 o del nuevo vertice (v_barra):

```
calcula_error(v1, v2) {
 q_barra = Q[v1] + Q[v2];
 delta_q = obtiene_delta_q(q_barra);
 det = delta_q.determinante();
 if (det <> 0)
  v_barra = delta_q.matriz_inversa()
            * [0 0 0 1]^T;
 else {
```

```
  v3 = (v1 + v2)/2;
  error1 = error_del_vertice(q_barra,
             v1.x,v1.y,v1.z);
  error2 = error_del_vertice(q_barra,
             v2.x,v2.y,v2.z);
  error3 = error_del_vertice(q_barra,
             v3.x,v3.y,v3.z);
  min_error = min(error1,
             min(error2, error3));
  if (min_error = error1)
     v_barra = v1;
  else if (min_error = error2)
     v_barra = v2;
  else if (min_error = error3)
     v_barra = v3;
 }
 min_error=error_del_vertice(q_barra,
     v_barra.x, v_barra.y, v_barra.z)
 return <v1, v2, v_barra, min_error>;
}

Calcula el error cuádrico = v^T Q v

error_del_vertice(q,x,y,z) {
 return (q[0]*x*x + 2*q[1]*x*y
  + 2*q[2]*x*z + 2*q[3]*x + q[5]*y*y
  + 2*q[6]*y*z + 2*q[7]*y + q[10]*z*z
  + 2*q[11]*z + q[15]);
}
```

Paso 3, 4 y 5:

- *Calcule la contracción óptima.*
- *Ubique todos los pares de costo mínimo al principio de la estructura errores.*
- *Iterativamente elimine el par (v_1, v_2) de menor costo, contraiga este par y actualice los costos de todos los pares donde se encuentra v_1.*

```
contracciones(cuantas_caras){
 selecciona_pares();
 while (caras.size() > cuantas_caras){
  // Se obtiene el par de vértices de
  // menor error cuádrico.
  <v1, v2, v_barra, error> =
     errores.minimo();
  // Se actualiza el vértice v1
  vertices[v1] = v_barra;
  // Se actualiza el Q de v1
  Q[v1] = Q[v1] + Q[v2];
  // Se eliminan las caras que
  // contengan la arista (v1,v2)
  caras.eliminar(v1,v2);
  // Se reemplaza v2 por v1.
  caras.reemplazar(v2,v1);
  // Se elimina el vértice v2
  vertices.eliminar(v2);
  // Se reemplaza (v2,? <> v1) por
  // (min(v1,?), max(v1,?)
  // y (? <> v1,v2) por
  // (min(?,v1), max(?,v1)).
  errores.reemplaza(v2,v1);
  // Se elimina el par (v1,v2)
  errores.eliminar(v1,v2);
  // Se actualiza los errores del
  // vértice v1: (v1,?) y (?,v1).
  errores.actualiza(v1);
 }
}
```

EXPERIMENTACION

Para la experimentación de este algoritmo se tomo en cuenta los modelos (.smf) de la siguiente página:

http://mgarland.org/research/quadrics.html

La Figura 3 muestra un pequeño resumen de los tiempos de ejecución del programa con el modelo Bunny (conejo) apreciados en las Figuras 4, 5 y 6.

Bunny : 34,834 vertices, 69,451 caras (triángulos)

NVert	% Vertices	Caras	% Caras	Tiempo (seg.)
18497	53	34600	50	27
9676	28	17000	24	35
4550	13	6800	10	37
2837	8	3399	5	37
1777	5	1300	2	37
1421	4	599	1	38
1269	4	300	0.4	38

Figura 3: Ejemplos de tiempos de ejecución.

Figura 4: Modelo Original Bunny con 69,451 triángulos.

Figura 5: Una aproximación utilizando solamente 1,300 triángulos.

Figura 6: Una aproximación utilizando solamente 300 triángulos.

BIBLIOGRAFIA

[1] Garland, M. and Heckbert, P. S. (1997). *Surface simplification using quadric error metrics.* In *Proceedings of the 24th Annual Conference on Computer Graphics and interactive Techniques* International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 209-216.