Tema-2-Resumen.pdf

LosCocos

Informática Gráfica

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

WUOLAH

El más PRO del lugar puedes ser Tú.

¿Quieres eliminar toda la publi de tus apuntes?

¡Fuera Publi!
Concéntrate al máximo

Apuntes a full.
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.

Quiero ser PRO

4,95 / mes

TEMA 2: MODELADO DE OBJETOS

APROXIMACIONES PARA REPRESENTAR DATOS REALES

- MODELOS BASADOS EN FRONTERAS → Se utilizan para representar sólo la superficie del objeto mediante un número finito de polígonos planos (caras)
- MODELOS BASADOS EN ENUMERACIÓN ESPACIAL → representan la superficie y el interior del objeto, así como otras propiedades del espacio. Se utilizan en Arqueología, TAC...

Para representar un modelo necesitamos modelar la geometría y la topología, para ello necesitamos una función de dibujado que lea los datos y los saque por pantalla

- Geometría: representa las coordenadas de los puntos que están en la superficie del objeto
- Topología: tiene que ver con la organización de los puntos de la geometría para dar lugar a la superficie

- Vértice: punto de la superficie
- Línea: unión de dos puntos o segmento que une dos caras adyacentes
- Cara: secuencia de puntos.

TIPOS DE MALLAS: SOPA DE TRIÁNGULOS

Lo más simple es representar cada triángulo con las coordenadas de sus vértices

Triángulo	V_1	V_2	V_3
1	X_{V_1}, Y_{V_1} Z_{V_1}	$X_{V_2} Y_{V_2}$ Z_{V_2}	X_{V_3}, Y_{V_3} Z_{V_3}
⋮			

Tiene puntos débiles

[-] Si un punto aparece en K triángulos, aparecerá K veces en la tabla

[-] Falta info explícita sobre la conectividad de los elementos, por ejemplo, cuando 2 triángulos comparten aristas, perdemos mucho tiempo en comparar coordenadas.

Podemos representarlo mediante glBegin/glEnd o glDrawElements

```
glBegin(GL_TRIANGLES);
    for (int i=0 ⟶ i < mesh→num_tri)
      for (int j=0 ⟶ j<3)
          glVertex3f (mesh→tri[i][j]);
glEnd();
```

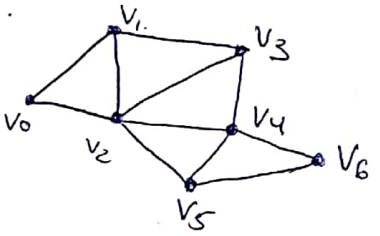
```
glEnableClientState (GL_VERTEX_POINTER);
glVertexPointer (..., mesh→triangles);
glDrawArrays (GL_TRIANGLES, 0, 3*mesh→num_tri);
```

TIPOS DE MALLAS : TIRAS DE TRIÁNGULOS

Existe una primitiva que nos permite ahorrar repetir vértices.

La primitiva GL_TRIANGLE_STRIP permite reutilizar vértices. Mejoramos el rendimiento en cuanto a número de veces aparece cada vértice pero seguimos sin tener información topológica.

```
glBegin(GL_TRIANGLE_STRIP);
    glVertex3fv (v0);
    :
glEnd();
```



TIPOS DE MALLAS : TABLA DE TRIÁNGULOS

Podemos utilizar una estructura que indexe vértices y cuya principal cuyo sean los vértices y los triángulos. Tendríamos una estructura:

```
Struct Malla {
    int num_tri;
    int num_vertices;
    Tupla3f *vertices;
    Tupla3i *caras;
};
```

Podemos representarlo:

```
glBegin(GL_TRIANGLES)
  for (int i=0 ⟶ mesh→num_tri)
    for (j=0 ⟶ j<3)
      indice = tri[i][j];
      glVertex3fv (mesh→vertices[indice]);
```

```
glEnableClientState(GL_VERTEX_POINTER)
glVertexPointer(... mesh→vertices)
glDrawElements (GL_TRIANGLES,
  3*mesh→num_tri, ..., mesh→
  triangles);
```

PLY

Permite almacenar una malla poligonal en una est- archivo ASCII o binario en forma de tabla de vértices y caras. Podemos almacenar otras propiedades de las primitivas como la textura y el color.

- Cabecera: propiedades del objeto y su formato. Indicamos nº de vértices y caras.
- Lista de vértices: uno por línea. Coordenadas X,Y,Z en formato ASCII y separados por un espacio
- Lista de caras: una por cada línea. Primero su número de vértices y luego los índices.

Cuestiones que se plantean:

- ¿Cómo calculo el ángulo entre dos caras adyacentes? Es insuficiente mirar en bucle en la lista.
- ¿Cómo expando una selección? Es insuficiente buscar caras adyacentes que expandir
- ¿Cómo detecto inconsistencias en la orientación de los triángulos? Necesito conocer el entorno de los triángulos y vuelvo a tener el mismo problema.

ARISTAS ALADAS

Ineficientes para dibujar pero permiten muchos cálculos numéricos.

La estructura de aristas aladas permite almacenar por cada arista: los vértices que la forman, su cara por la dcha y por la izq, y la arista anterior y siguiente de la cara dcha e izq. Así mismo, cada vértice y cara tienen referencias a ~~cada~~ una arista.

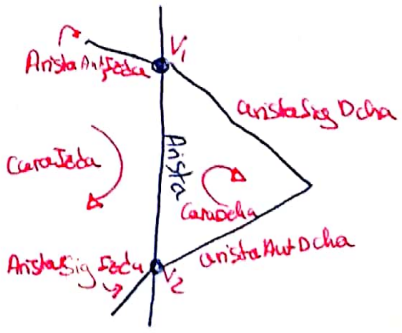
El hecho de almacenar la arista siguiente y anterior permite hacer recorridos por la tabla de aristas:

→ Con una arista y una cara podemos conocer qué aristas y vértices la forman
→ Con una arista y un vértice podemos ver qué aristas inciden en ese vértice y los triangulos que comparten ese vértice

Vértice
Tupla3f p int arista

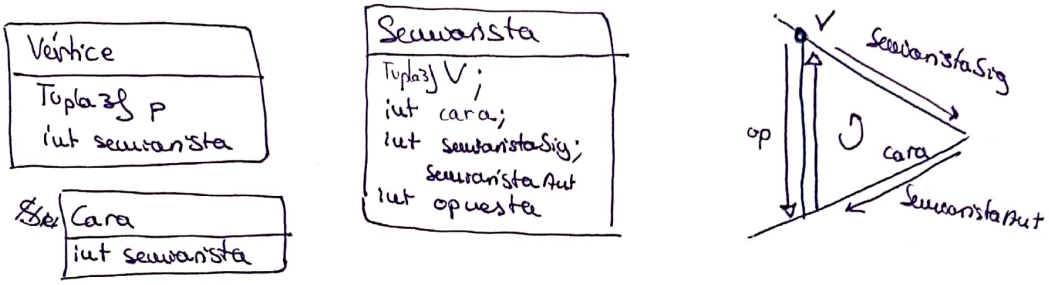
Cara
int arista

Arista
Tupla3f V1,V2 int caraDcha, caraIzda; int aristaSigDcha, aristaSigIzda, aristaAntDcha, aristaAntIzda



SEMIARISTAS ALADAS

Las aristas aladas son una estructura potente y elegante, pero con el inconveniente de que tenemos que estar comprobando la orientación de la arista antes de pasar a la siguiente. Las semiaristas aladas dividen a la arista en dos, en el sentido antihorario de la cara a la que pertenecen



ATRIBUTOS MODELOS DE FRONTERAS

- NORMALES: vectores de longitud unidad
  - Normales en las caras: vector unitario normalizado apuntando al exterior de la malla. Se precalcula con el orden de la cara

$$u = \frac{w}{\|w\|}, \text{ donde } w = e_i \times e_j, \text{ aristas adyacentes.}$$

  - Normales en los vértices: vector unitario perpendicular a la superficie donde se encuentra el punto. $u = \frac{S}{\|S\|}$ donde $S = \sum_{i=j}^{k} w_i$ (normales en las caras)
    Es una aproximación
- COLOR: RGB ó RGBA
  - Color en las caras: color homogéneo en la superficie
  - Color en los vértices: el color varía de forma continua en los vértices.
- TEXTURAS

Cada vértice tiene una normal y un color fijos. Distintas definiciones: Con glVertex() usamos glColor() y glNormal() y con glDrawElements() usamos glColorPointer() y glNormalPointer.

TRANSFORMACIONES GEOMÉTRICAS

Una transformación geométrica es calcular a las coordenadas del mundo a partir de las coordenadas locales (que solo afectan al modelo)

Se define como una aplicación matemática sobre un punto p, al que se le aplica otro punto o vector q: $q = Tp$ "q es T aplicado a p"

Una transformación geométrica modifica las coordenadas del objeto al que afecta.

- TRASLACIÓN: consiste en desplazar todos los puntos en la misma dirección la misma distancia a partir de un vector director d.

$$p'x = px + dx$$
$$p'y = py + dy$$
$$p'z = pz + dz$$

- ESCALADO: Estrechar o alargar la figura en una o distintas direcciones desde el origen de coordenadas.

$$f_x(p) = p'x = e_x px$$
$$f_y(p) = p'y = e_y py$$
$$f_z(p) = p'z = e_z pz$$

- ROTACIÓN: rotar un ángulo α sobre un eje de rotación (sobre origen)
  - Rotación en 2D: α ángulo sobre el origen
  - Rotación en 3D: cada rotación modifica las coordenadas de los ejes menos la del eje de rotación. Son rotaciones antihorarias para $\alpha > 0$

- CIZALLA: Desplazar los puntos en la misma dirección pero con distintas distancias.

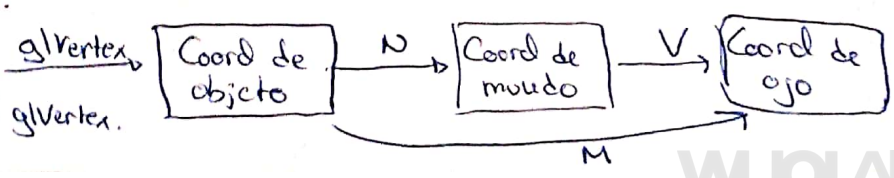
* Las coordenadas homogéneas $[x, y, z, 1]$ se utilizan para poder expresar la traslación como una matriz 4x4 y poder usarla en la matriz modelview.

MATRIZ MODELVIEW

OpenGL guarda en su estado una matriz 4x4 que codifica las transformaciones geométricas, modelview matrix. Podemos verla como la composición de las matrices V y N:

- N matriz de modelado, posiciona los puntos en su lugar en coordenadas del mundo
- V matriz de vista, que posiciona los puntos en su lugar en coordenadas relativas a la cámara.

Se aplica a todos los puntos con glVertex.



Especificación

1. Llamada a glMatrixMode(GL_MODELVIEW) para indicar que las llamadas van a afectar a la matriz M
2. glLoadIdentity para igualar M a la matriz identidad
3. gluLookAt para componer la matriz V
4. Algunas funciones para componer la matriz N: glTranslatef, glMultMatrix...

Generación

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity;
glMultMatrix(T3);
glMultMatrix(T2);
glMultMatrix(T3);
```

$M = T_3 \cdot T_2 \cdot T_1$

Las instrucciones se ejecutan a la inversa que se escriben.

MODELOS JERÁRQUICOS

Los objetos se pueden clasificar en:

- Objetos simples: que no están formados por otros objetos
- Objetos compuestos: formados por otros objetos más simples.

Representación

La estructura de un modelo jerárquico se representa con un grafo de escena, un grafo dirigido acíclico donde:

- Cada objeto compuesto es un subgrupo dentro del grafo
- Cada objeto simple es un nodo terminal
- Cada arco une dos nodos, puede ser una transformación, nodos terminales...

Visualización

Se basa en operaciones que permiten guardar o recuperar la matriz modelview

- Mediante multiplicación de matrices, empezando por la matriz identidad al inicio de cada grafo.
- Usando de mecanismo la pila de OpenGL para recuperar valores de la matriz

PILA OPENGL · OpenGL guarda una pila para las transformaciones geométricas de tal forma que cuando hacemos glPushMatrix se actualiza el tope con una copia de la matriz. Este tope es el que se consulta y transforma mientras que el de abajo solo se accede al hacer glPopMatrix. Con la mult. la pila no cambia de tamaño

GRAFOS PARAMETRIZABLES

Un grafo de escena puede tener nodos parametrizados de forma que sus valores puedan variar. Este número de parámetros define el grado de libertad de un objeto