Introducción a las mallas indexadas

A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Formas de representación de superficies

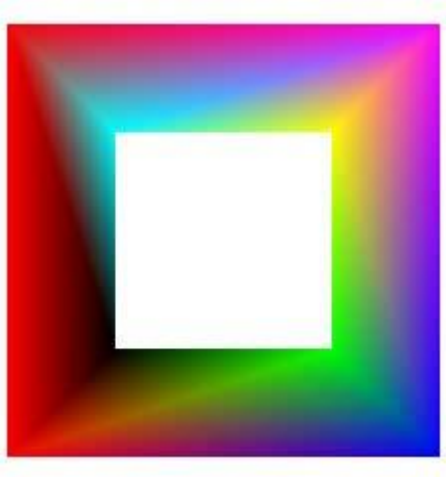
- Una malla (en inglés, mesh) es una colección de polígonos que se usa para representar la superficie de un objeto tridimensional
- Estándar de representación de objetos gráficos
  - Facilidad de implementación y transformación
  - Propiedades sencillas
- Representación exacta o aproximada de un objeto
- Elemento más en la representación de un objeto (color, material, textura)

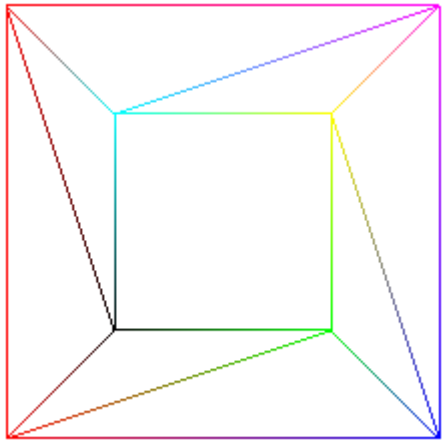
Renderización de primitivas

- Modo inmediato en OpenGL 1.0: **`glVertex(), glNormal(), …`**
  - Este modo y el siguiente son todavía ampliamente usados hoy
- Vertex arrays en OpenGL 1.1
  - Aunque se decide deprecar este modo y el anterior, se pueden seguir usando todavía en OpenGL 3.0
  - Los vertex arrays son obligatorios en las versiones últimas (OpenGL 4.3)
  - Estos modos no almacenan datos en la GPU. Cada vez que se invocan, los datos implicados se pasan a la GPU
- Vertex Buffer Objects (VBO) en OpenGL 1.5
  - Modo recomendado en la actualidad
  - A diferencia de los modos anteriores, los VBO's permiten almacenar los datos en la GPU
  - Son similares a los objetos de textura

- Cómo pasar vértices y colores en modo inmediato

```
glBegin(GL_TRIANGLE_STRIP);
        glColor3f(0.0, 0.0, 0.0); glVertex3f(30.0, 30.0, 0.0);
        glColor3f(1.0, 0.0, 0.0); glVertex3f(10.0, 10.0, 0.0);
        glColor3f(0.0, 1.0, 0.0); glVertex3f(70.0, 30.0, 0.0);
        glColor3f(0.0, 0.0, 1.0); glVertex3f(90.0, 10.0, 0.0);
        glColor3f(1.0, 1.0, 0.0); glVertex3f(70.0, 70.0, 0.0);
        glColor3f(1.0, 0.0, 1.0); glVertex3f(90.0, 90.0, 0.0);
        glColor3f(0.0, 1.0, 1.0); glVertex3f(30.0, 70.0, 0.0);
        glColor3f(1.0, 0.0, 0.0); glVertex3f(10.0, 90.0, 0.0);
        glColor3f(0.0, 0.0, 0.0); glVertex3f(30.0, 30.0, 0.0);
        glColor3f(1.0, 0.0, 0.0); glVertex3f(10.0, 10.0, 0.0);
glEnd();
```





Variante del modo inmediato

- Cómo pasar vértices y colores mediante punteros

```
static float vertices[8][3] = {
       {30.0, 30.0, 0.0}, {10.0, 10.0, 0.0}, {70.0, 30.0, 0.0}, {90.0, 10.0, 0.0},
       {70.0, 70.0, 0.0}, {90.0, 90.0, 0.0}, {30.0, 70.0, 0.0}, {10.0, 90.0, 0.0}
};

static float colors[8][3] = {
       {0.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0},
       {1.0, 1.0, 0.0}, {1.0, 0.0, 1.0}, {0.0, 1.0, 1.0}, {1.0, 0.0, 0.0}
};

// Dibujo del cuadrado anular
glBegin(GL_TRIANGLE_STRIP);
       for (int i = 0; i < 10; ++i) {
            glColor3fv(colors[i % 8]);
            glVertex3fv(vertices[i % 8]);
       }
glEnd();
```

Modo inmediato

- Ventajas de la variante
  - Los vértices y colores pueden ser reutilizados; de hecho, solo son necesarios 8 vértices (y 8 colores), y no 10, como exige la primitiva `GL_TRIANGLE_STRIP`
  - La definición de vértices y colores tiene lugar en una parte específica del código
  - Mejor uso de la memoria, menor redundancia
  - En consecuencia, más facilidad para la depuración, más eficiencia
- Inconvenientes: los del modo inmediato
  - Las mallas complejas se suelen leer de un archivo o se crean procedimentalmente y el proceso de mandar los datos a OpenGL supone un envío por vértice, desde el lugar donde se encuentren
  - Sería mejor enviar directamente un array de datos que no ejecutar millones de llamadas a **`glVertex(), glColor(), …`**

Vertex arrays

- Cómo pasar vértices y colores mediante **`vertex arrays`**
  - Primero se definen los arrays (llamados vertex arrays) que se desean pasar

```
static float vertices[] = {
  30.0, 30.0, 0.0,
  10.0, 10.0, 0.0,
  70.0, 30.0, 0.0,
  90.0, 10.0, 0.0,
  70.0, 70.0, 0.0,
  90.0, 90.0, 0.0,
  30.0, 70.0, 0.0,
  10.0, 90.0, 0.0 };
static float colors[] = {
  0.0, 0.0, 0.0,
  1.0, 0.0, 0.0,
  0.0, 1.0, 0.0,
  0.0, 0.0, 1.0,
  1.0, 1.0, 0.0,
  1.0, 0.0, 1.0,
  0.0, 1.0, 1.0,
  1.0, 0.0, 0.0 };
```

Vertex arrays

- Y en **draw()** de la clase **Mesh**, los vértices y colores se pueden recuperar con el comando **glArrayElement()**

```
glBegin(GL_TRIANGLE_STRIP);
  // Cuando se usa glArrayElement(i);
  // vertices[i] y colors[i] se recuperan a la vez
  for (int i = 0; i < 10; ++i) glArrayElement(i % 8);
glEnd();
```

- Los dos vertex arrays (**vertices** y **colors** de la transparencia anterior) se activan/desactivan (con los comandos **glEnableClientState()/glDisableClientState()**) antes/después de llamar a **draw()** del ítem anterior. Además, antes de recuperar los datos se especifica dónde están almacenados y en qué formato, con los comandos **glVertexPointer()**, **glColorPointer()**

```
// Activación de los vertex arrays
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
     // Especificación del lugar y formato de los datos
     glVertexPointer(3, GL_FLOAT, 0, vertices);
     glColorPointer(4, GL_FLOAT, 0, colors);
     draw();
// Desactivación de los vertex arrays
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Vertex arrays

- Los vértices y colores aparecen en arrays unidimensionales, aunque podrían haberse introducido también mediante arrays bidimensionales
- La instrucción

```
glArrayElement(i % 8);
```

  extrae simultáneamente el vértice y el color de lugar `(i % 8)`-ésimo. Por eso los arrays no repiten componentes y tienen tamaño 8.
- La especificación del lugar donde se encuentran los vertex arrays se hace con el comando

```
gl..Pointer(size, type, stride, *pointer);
```

  donde **`size`** es el número de datos (por vértice, por color, …), **`type`** es el tipo de los datos, **`stride`** es el offset que se debe saltar antes de empezar a leer (0, si los datos aparecen consecutivos) y **`*pointer`** es la dirección del vertex array
  - Ojo, para el vertex array de normales el comando solo tiene 3 parámetros

```
glNormalPointer(type, stride, *pointer);
```

Vertex arrays sin índices

- Para renderizar vértices y colores, sin referencia a normales ni a índices se puede usar el comando

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, numvertices);
```

y se toman los elementos de los vertex arrays activos, tal como marque la primitiva y secuencialmente. Pero como toma componentes, no lugares como **`glArrayElement()`**, los vertex arrays pueden repetir elementos

- La forma general de este comando es

```
glDrawArrays(mPrimitive, first, size());
```

que dibuja con **`mPrimitive`**, usando **`size()`** elementos de los arrays de vértices y colores, empezando en la posición **`first`**.

- Observa que esta instrucción es la que se usa en la definición de **`draw()`** de la clase **`Mesh`**

Vertex arrays con índices

- Se puede dibujar el cuadrado anular con un solo comando y sin repetir vértices ni colores

```
glDrawElements(GL_TRIANGLE_STRIP, 10, GL_UNSIGNED_INT,
  stripIndices);
```

en lugar de escribir varios comandos como con

```
for (int i = 0; i < 10; ++i) glArrayElement(i % 8);
```

o en lugar de repetir elementos como con

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, numvertices);
```

- Para usar este comando es necesario proporcionar los índices de las componentes de los vertex arrays (vertices y colors) donde se encuentran los datos, en el orden en que los tomará la primitiva que se use. En nuestro caso, la primitiva es **GL_TRIANGLE_STRIP** con lo que los índices son

```
unsigned int stripIndices[] =
        { 0, 1, 2, 3, 4, 5, 6, 7, 0, 1 };
```

Vertex arrays con índices

- En nuestro ejemplo, la instrucción:

```
glDrawElements(GL_TRIANGLE_STRIP, 10, GL_UNSIGNED_INT, sI);
```

  extrae los datos de 10 vertices en un solo comando

- Piénsese que es más eficiente una llamada a este comando que no 10 llamadas al comando

```
glArrayElement();
```

- La forma general del comando es

```
glDrawElements(mPrimitive, count, type, *indices);
```

  donde **`count`** es el número de índices y **`type`** es el tipo de los índices

- Este comando extrae elementos de los arrays de vértices y colores, pero en el orden que indican los índices. Recuérdese que, en el código del ejemplo, los índices dictan que los vértices se extraigan en el mismo orden que se siguió en el for.