Representación y renderizado de mallas

Informática Gráfica I

Material de: **Antonio Gavilanes**
Adaptado por: **Elena Gómez** y **Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es

UNIVERSIDAD COMPLUTENSE MADRID

Contenido

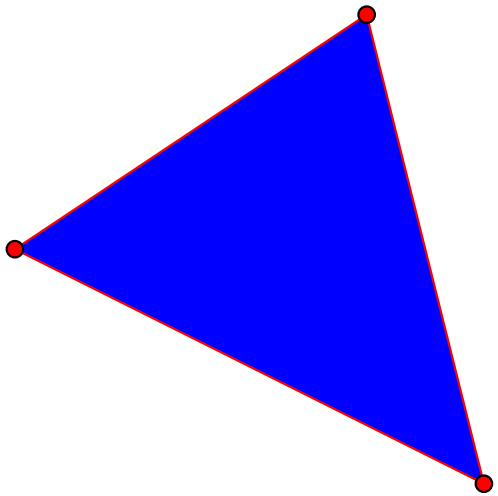
Definición

Definición

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).

Formas de representación de superficies



Posición	Color	Textura
(0, 0, 0)	(0, 0, 255)	(0.5, 1)
(1.5, 1, 0)	(0, 0, 255)	(0, 0)
(2, -1, 0)	(0, 0, 255)	(1, 0)

Renderización de primitivas

- Modo inmediato en OpenGL 1.0: `glVertex()`, `glNormal()`,...
  - Los vértices y sus atributos residen en el código.
  - Se pasan uno a uno a la biblioteca gráfica con llamadas a función.
  - Este modo y el siguiente son todavía ampliamente usados hoy.
- Vertex arrays en OpenGL 1.1
  - Los vértices y sus atributos residen en la memoria de la CPU.
  - Se cargan vectores de vértices y atributos de la CPU a la GPU en cada renderizado.
  - OpenGL 3.1 considera obsoletos este modo y el anterior, pero se pueden seguir usando todavía si la implementación de OpenGL soporta el perfil de compatibilidad.

Renderización de primitivas

- Vertex Buffer Objects (VBO) en OpenGL 1.5 y Vertex Array Objects (VAO) en OpenGL 3.0.
  - Los vértices y sus atributos reside en la GPU.
  - Modo recomendado en la actualidad y el único compatible con el *core profile*.
  - Similares a los objetos de textura.
  - Es lo que hemos usado en las prácticas.

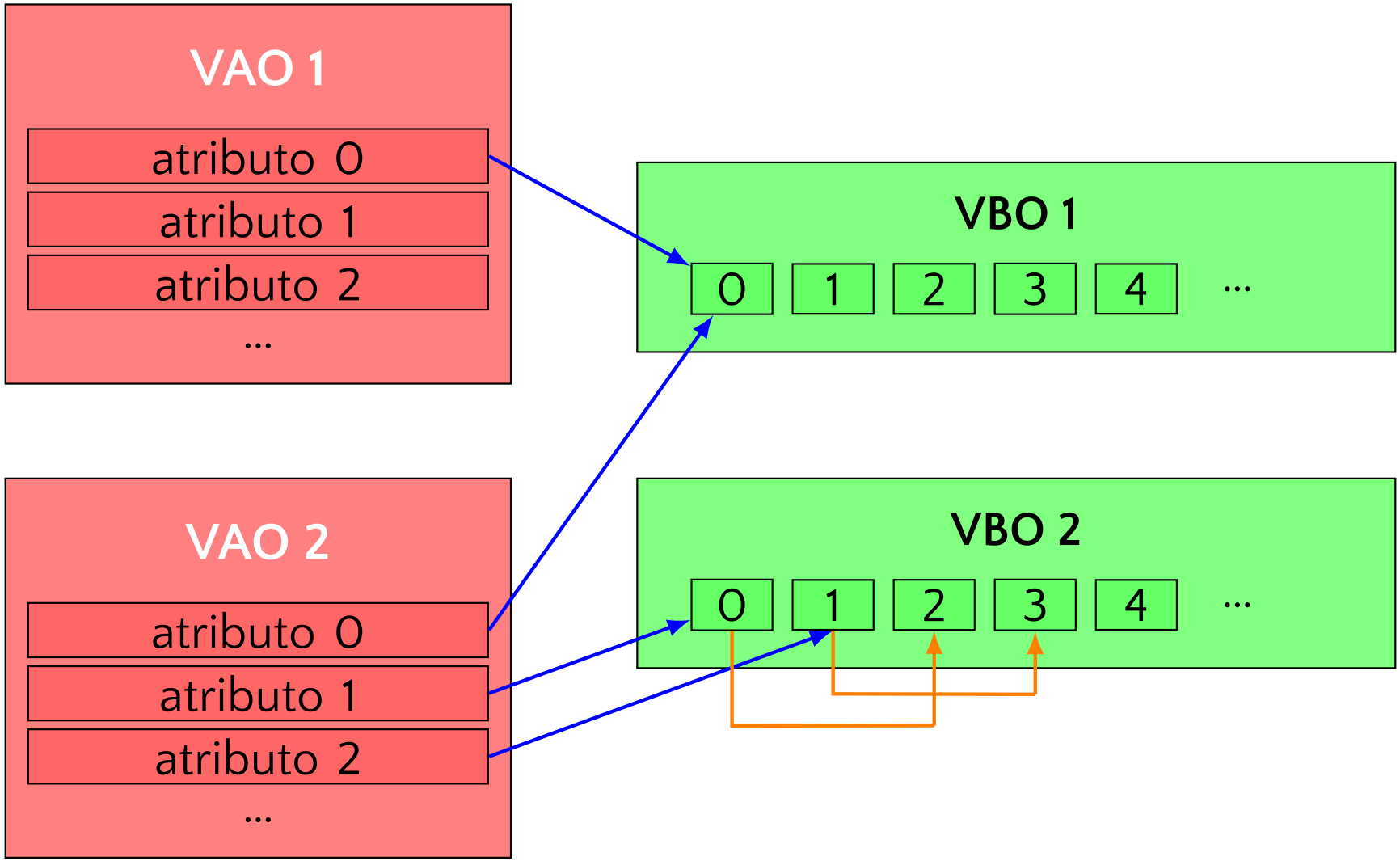
Vertex Array Objects

Vertex Array Objects

Vertex Arrays Objects

- Los **Vertex Arrays Objects (VAO)** y **Vertex Buffer Object (VBO)** permiten almacenar en la memoria de la GPU una secuencia de vértices con sus características para enviarlos al cauce gráfico.
- Un VBO es un búfer interno de la tarjeta gráfica que almacena arrays de vértices, colores, coordenadas de textura, etc.
- Un VAO es una estructura que agrupa en la GPU toda la información de vértices que requieren las primitivas gráficas.
- El VAO no almacena directamente los datos de los vértices, sino referencias (**descriptores**) a los VBO que realmente contienen los datos.

Vertex Arrays Objects



Vertex Arrays Objects: Creación

Generación	`glGenVertexArrays(1, &VaoId)`
Eliminación	`glDeleteVertexArrays(1, &VaoId)`
Activación	`glBindVertexArray(VaoId)`
Desactivación	`glBindVertexArray(0)`

- Al generar un VAO se reserva espacio para él en la memoria de la GPU, que se ha de liberar cuando ya no haga falta.
- Solo puede haber un VAO activo en cada momento y se selecciona con `glBindVertexArray` (al igual que con las texturas).
- Un VAO se necesita activar para renderizarlo y para modificarlo.

Vertex Buffer Objects: Creación

Generación	`glGenBuffers(1, &VboId)`
Eliminación	`glDeleteBuffers(1, &VaoId)`
Activación	`glBindBuffer(tipo, VboId);`
Desactivación	`glBindBuffer(tipo, 0)`

- Se generan, eliminan, activan y desactivan de forma similar a los VAO (y a las texturas).
- Los tipos más habituales son `GL_ARRAY_BUFFER` (vértices, colores, coordenadas de textura, normales) y `GL_ELEMENT_ARRAY_BUFFER` (índices).
- Solo puede haber un VBO de cada tipo activo en cada momento y hay que activarlos para modificarlos.

Vertex Buffer Objects: Carga de datos

Los datos se cargan con

```
glBufferData(tipo, tamaño, puntero a los datos, modo);
```

- El tamaño se indica en bytes y se pasa el puntero al inicio.
- El tipo es el mismo que en `glBindBuffer`.
- El modo es `GL_`*freq*`_`*nature* donde *freq* indica la frecuencia que se modificarán y usarán los datos. (`STREAM`, `STATIC`, `DYNAMIC`) y *nature* el uso que se les dará (`DRAW`, `READ`, `COPY`).
- El modo habitual será `GL_STATIC_DRAW`: los datos se modifican una sola vez y se utilizan muchas en los comandos de renderizado de OpenGL.

Vertex Buffer Objects: Carga de datos

Ejemplo de carga de datos

```
GLuint VboId;
std::vector<glm::vec3> vertices = /* ... */

glGenBuffers(1, &VboId);  // crea el VBO y guarda su ident.
glBindBuffer(GL_ARRAY_BUFFER, VboId);  // activa el VBO

glBufferData(GL_ARRAY_BUFFER,  // tipo de búfer
             vertices.size() * sizeof(glm::vec3),  // tamaño
             vertices.data(), GL_STATIC_DRAW);  // puntero, modo

glBindBuffer(GL_ARRAY_BUFFER, 0);  // desactiva el VBO
```

Atributos de un VAO

- Un VAO define varios *vertex attribute arrays* numerados para poder acceder a ellos desde el cauce gráfico y cuyos datos están en VBOs
- Vértices, colores, coordenadas de textura, etc. serán cada uno un array de atributos
- Suponiendo un VAO y VBO activos

```
glBindVertexArray(VaoId);
glBindBuffer(tipo, VboId);
```

el atributo k y sus propiedades se definen con:

```
glVertexAttribPointer(k, tamaño, tipo, normalizar,
                      stride, offset);
glEnableVertexAttribArray(k);
```

Atributos de un VAO

```
glVertexAttribPointer(k, tamaño, tipo, normalizar,
                      stride, offset);
```

- `tamaño` se refiere al número de componentes (1, 2, 3 ó 4) de cada vector. Por ejemplo, en un color RGBA será 4 y unas coordenadas de textura 2.
- `tipo` es el tipo de cada componente: `GL_FLOAT`, `GL_DOUBLE`, `GL_INT`, etc.
- `stride` y `offset` indican donde empieza el array en el VBO activo y la distancia entre elementos.
  Esto permite meter en un búfer toda la información como si cada vértice fuera un `struct` (nosotros no lo usamos así y pondremos a cero ambos parámetros).

Atributos de un VAO

Ejemplo de vinculación del array de vértices

```
GLuint vertexVBO, meshVAO;
glBindVertexArray(meshVAO);
glBindBuffer(GL_ARRAY_BUFFER, vertexVBO);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);

glBindBuffer(GL_ARRAY_BUFFER, 0);  // o vincular otro
glBindVertexArray(0);
```

. . . . . .

```
layout (location = 0) in vec3 aPos;  // así accedemos en GLSL
```

Dibujo de un VAO

1. Se enlaza el VAO con `glBindVertexArray`.
2. Se dibuja con `glDrawArrays` (o `glDrawElements` si hay índices).
3. Se desenlaza el VAO.

Ejemplo

```
glBindVertexArray(VaoId); // Enlaza el VAO
// Dibuja la geometría
glDrawArrays(GL_TRIANGLES, 0, numVertices);
glBindVertexArray(0); // Desactiva el VAO
```

Modos obsoletos

Modos obsoletos

Modo inmediato

- Cómo pasar vértices y colores en modo inmediato: Función `glVertex`
  - Vértices de puntos, líneas o polígonos.
  - Siempre entre `glBegin` y `glEnd`.
  - Coordenadas de la forma x, y, z, w.
  - Las coordenadas color, normal y textura se asocian al vértice.

Ejemplo

Define un triángulo con las coordenadas (x_1, y_1, z_1), (x_2, y_2, z_2) y (x_3, y_3, z_3)

```
glBegin(GL_TRIANGLES);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y2, z2);
    glVertex3f(x3, y3, z3);
glEnd();
```

Modo inmediato

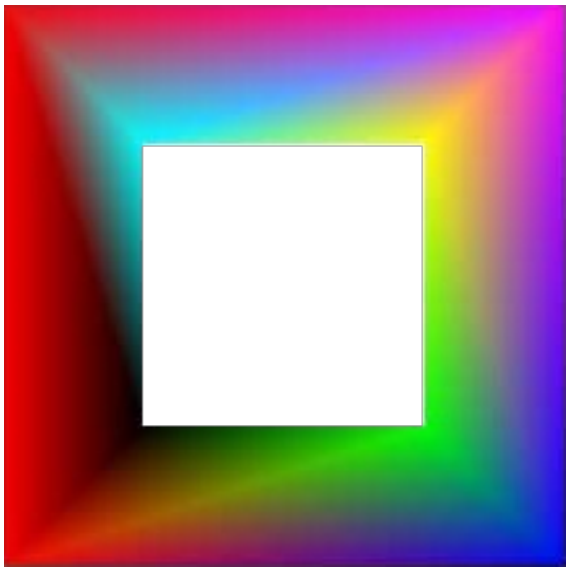
- Cómo pasar vértices y colores en modo inmediato

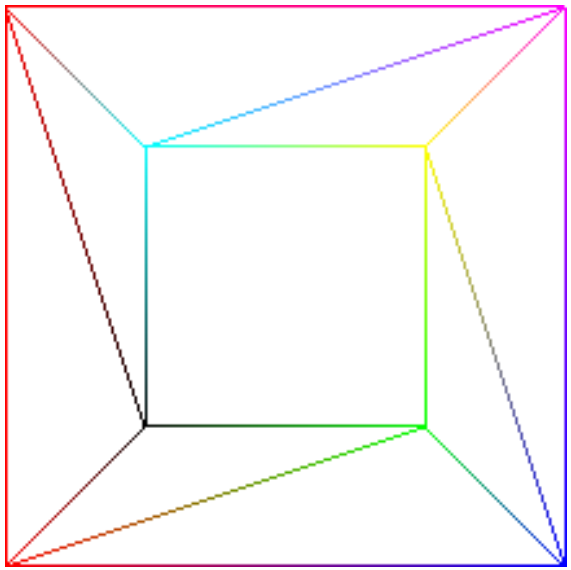
```
glBegin(GL_TRIANGLE_STRIP);
  glColor3f(0.0, 0.0, 0.0); glVertex3f(30.0, 30.0, 0.0);
  glColor3f(1.0, 0.0, 0.0); glVertex3f(10.0, 10.0, 0.0);
  glColor3f(0.0, 1.0, 0.0); glVertex3f(70.0, 30.0, 0.0);
  glColor3f(0.0, 0.0, 1.0); glVertex3f(90.0, 10.0, 0.0);
  glColor3f(1.0, 1.0, 0.0); glVertex3f(70.0,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

Modo inmediato

- 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);
  ...
glEnd();
```





Modo inmediato

Inconvenientes 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()`, ...

Arrays de vértices en la CPU

- 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 };
```

Arrays de vértices en la CPU

- Los dos *vertex arrays* (`vertices` y `colors` de la transparencia 24) se activan/desactivan (con los comandos `glEnableClientState()`/ `glDisableClientState()`) antes/después de llamar a `draw()` de la transparencia anterior.
- Además, antes de recuperar los datos se especifica dónde están almacenados y en qué formato, con los comandos `glVertexPointer()`, `glColorPointer()`.

Arrays de vértices en la CPU

```
// 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);
```

¡OJO!

Aparece `GL_FLOAT` porque los datos son `float`.
Si fueran `double`, es necesario escribir `GL_DOUBLE`.

Renderizado de arrays de vértices

- Para renderizar vértices y coloresse 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.

- 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`.