Introducción a Gráficos 3D

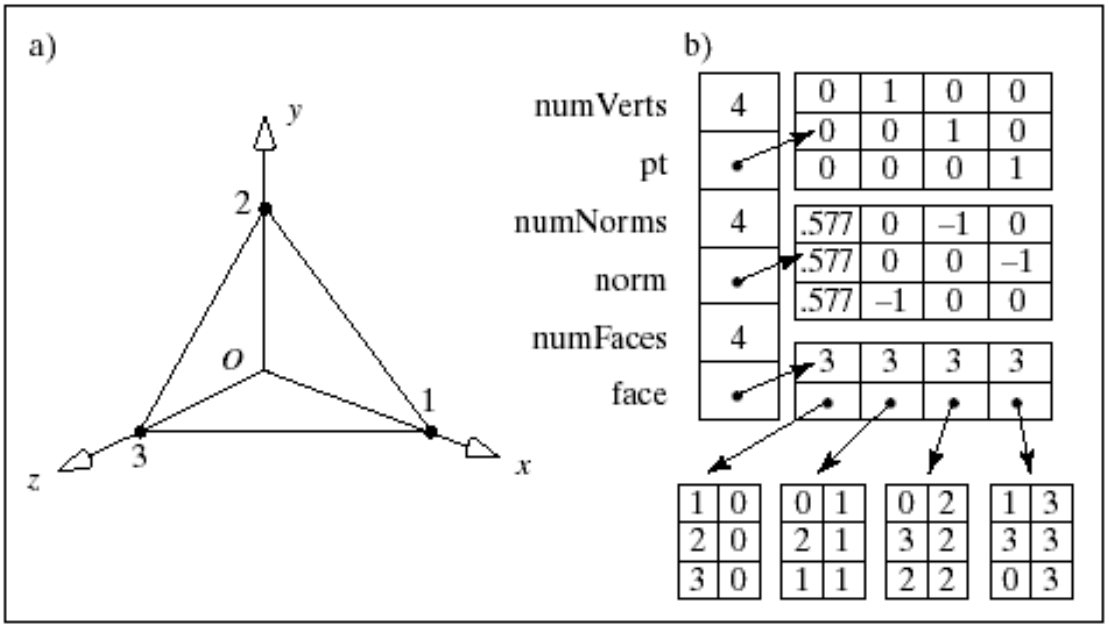
- **Descripción geométrica de objetos:**
  - **Mallas de triángulos**
  - **Vertex arrays**

Ana Gil Luezas
Departamento de Sistema Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Descripción de los objetos de la escena 3D

- Propiedades geométricas:
  - Mallas de Polígonos, que determinan la superficie del objeto. Habitualmente se trabaja con triángulos.
  - Ecuaciones paramétricas y/o implícitas de las superficies.
- Propiedades de los materiales: Color, brillo, textura, emisión de luz.
  Que requieren para su procesamiento otros datos geométricos:
  vector normal, coordenadas de textura …
- Los objetos se definen de forma local (o genérica), y se posicionan globalmente en la escena mediante transformaciones afines, que forman parte de la descripción del objeto (matriz de modelado M).

Tetraedro



Una representación para almacenar los datos de una malla, que evita la duplicación de información, es utilizar tres tablas:

- *TABLA DE VÉRTICES:* contiene las coordenadas de los vértices de la malla (información de posición)
- *TABLA DE NORMALES:* contiene las coordenadas de los vectores normales a los vértices o caras de las malla (información de orientación)
- *TABLA DE CARAS (TRIÁNGULOS):* contiene la descripción de cada cara de la malla: el número de vértices, **los índices** de sus vértices y de las normales a estos (información de conexión o topológica)
- OpenGL solo admite una tabla de índices : solo se pueden compartir vértices con los mismos atributos (normal, ...).

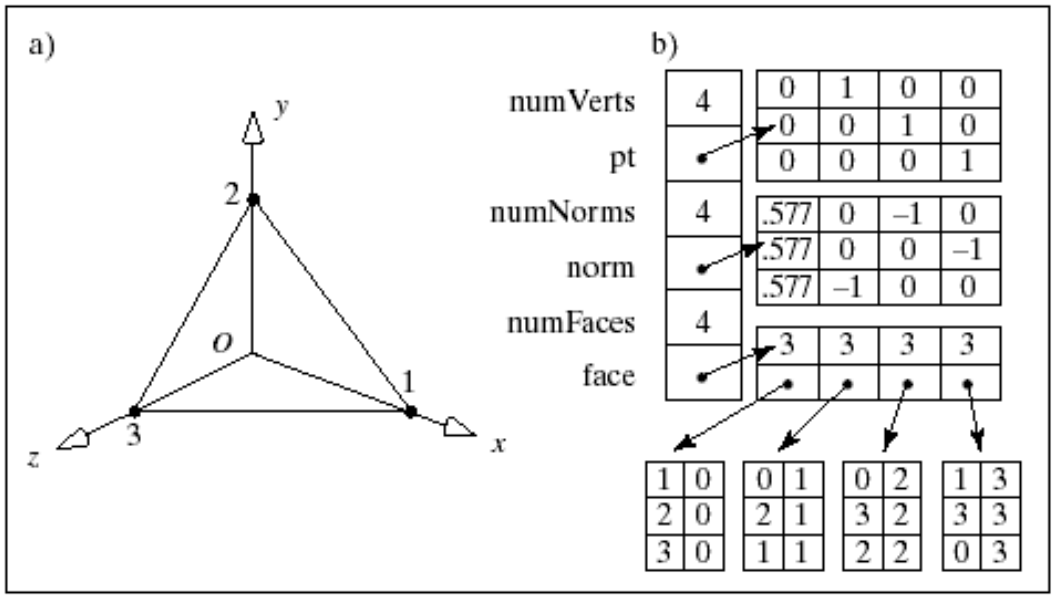
  Para pocos vértices no es interesante usar índices para compartirlos.

Representación de mallas

Representación para almacenar los datos de una malla sin tabla de índices:

Vértices (12) v1 v2 v3 v0 v2 v1 v0 v3 v2 v1 v3 v0

Normales (12) n0 n0 n0 n1 n1 n1 n2 n2 n2 n3 n3 n3



Representación de mallas

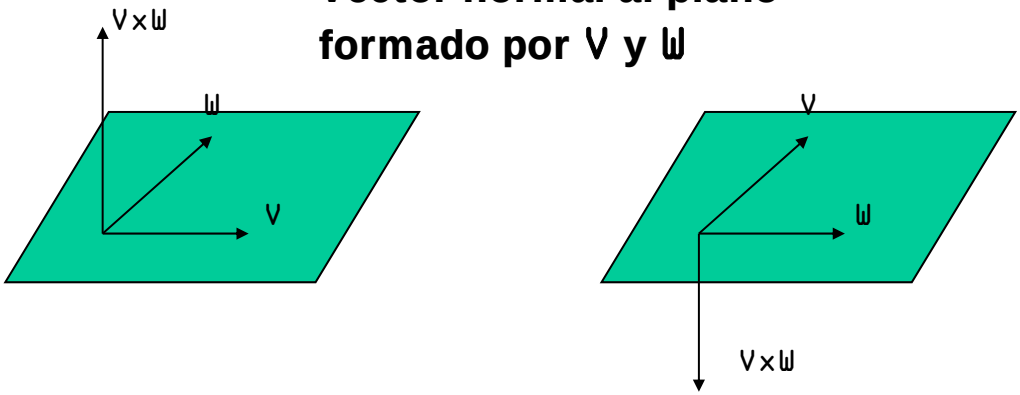
- Los vértices de cada polígono deben darse en orden contrario a las agujas del reloj (CCW) según se mira la cara del objeto desde fuera, de forma que el interior del polígono quede a la izquierda.
- El orden permite distinguir el interior del exterior y calcular la orientación (vector normal) de la cara.
- Los vectores normales deben estar normalizados (magnitud 1), se utilizan en la iluminación y detección de visibilidad.
- Cálculo del vector normal a una cara: tomamos tres vértices adyacentes **v0**, **v1** y **v2** , computamos el producto vectorial (v2– v1) x (v0– v1) y lo normalizamos.

  Si los vértices están en el orden CCW, el vector apuntará hacia fuera.

- **Producto vectorial de dos vectores**:

$$\left.\begin{aligned} V &= (v_1, v_2, v_3) \\ W &= (w_1, w_2, w_3) \end{aligned}\right\} \quad V \times W = \begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} \qquad |V \times W| = |V||W|\sin(\theta)$$

Vector normal al plano formado por V y W



OpenGL solo admite una tabla de índices: solo se pueden compartir vértices con los mismos atributos.

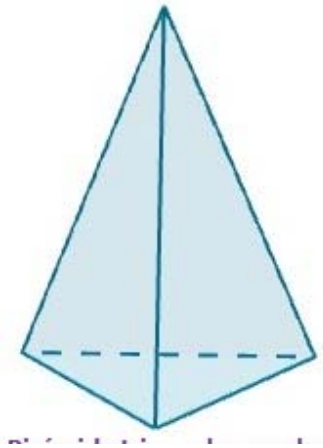
Malla con índices:

Nº de vértices y normales = 3 * 3 = 9

Nº de índices = 3 * 3 = 9

Malla sin índices:

Nº de vértices y normales = 3 * 3 = 9



OpenGL vertex array

Malla de triángulos sin índices.

```
class MallaTri{
public:
  GLuint numDat;   // tabla de vértices y normales
  PVec3 *vertices, *normales; // también pueden ser vector<PVec3>
  MallaTri(){ numDat = 0;  vértices =normales = nullptr; }
  ~MallaTri(){ delete[] vertices; delete[] normales; }
  bool load(char arch[]);  // -> generar los datos
  void draw();
protected:
  void activar();
  void desactivar();
};
```


OpenGL vertex array

```
void MallaTri::draw(){
  activar();
  glColor4d(r, g, b, a);
  glDrawArrays(GL_TRIANGLES, 0, numDat);
  desactivar();
}
// glDrawArrays(Primitiva, IndPrimerElem, NºElementos);
```

Si todos los vértices de la malla tienen el mismo vector normal, en lugar de una tabla de normales, se puede establecer el vector con glNormal3d(x, y, z) de forma análoga a glColor*()

OpenGL vertex array

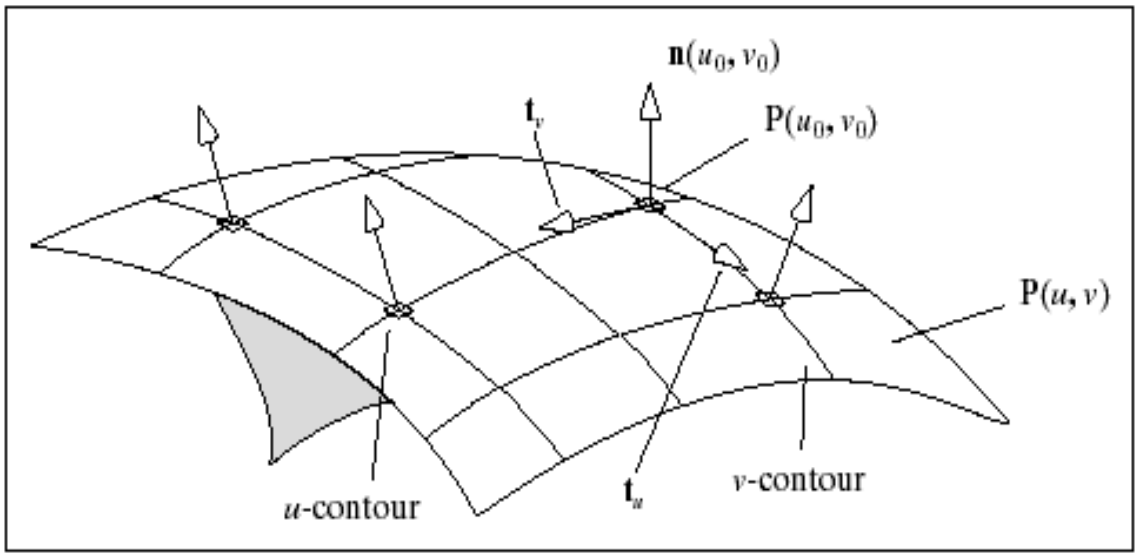
```
void MallaTri::activar(){
  glEnableClientState(GL_NORMAL_ARRAY);
  glEnableClientState(GL_VERTEX_ARRAY);
  glNormalPointer(GL_DOUBLE, 0, normales); // (Tipo, Offset, PtrArray)
  glVertexPointer(3, GL_DOUBLE, 0, vertices);
} // glVertexPointer(NºComponentes, Tipo, Offset, PtrArray)

void MallaTri::desactivar(){
  glDisableClientState(GL_VERTEX_ARRAY);
  glDisableClientState(GL_NORMAL_ARRAY);
}
```


Aproximación de superficies con mallas

- Los poliedros quedan representados exactamente por las mallas, puesto que son una colección de polígonos planos, lo que permite definirlos de forma intensiva mediante las tablas de las mallas.
- Cuando el objeto está formado por una superficie lisa (sin aristas), como es el caso de una esfera o un toro, solo podemos aproximarla generando una colección de polígonos planos (malla), a partir de la ecuación de la superficie.
- Esta malla será dibujada aplicando un algoritmo de coloreado gradual (smooth) que proporciona un aspecto de superficie lisa. Este algoritmo requiere que los vectores normales a cada vértice de la malla, se correspondan con los vectores normales a los correspondientes puntos de la superficie.

Aproximación de superficies con mallas



Aproximación de superficies con mallas

- Para que la aproximación sea buena, los polígonos deben ser suficientemente pequeños y sin cambios bruscos de dirección.
- Las coordenadas de los vértices serán calculadas evaluando la ecuación de la superficie en puntos discretos.
- Análogamente, los vectores normales serán calculados evaluando, en los mismos puntos, la ecuación de vectores normales a la superficie.
- La generación procedural de una malla, a partir de ecuaciones matemáticas, permite hacer aproximaciones con distintos niveles de detalle.
- En estos casos, es habitual utilizar **tabla de índices**.

OpenGL vertex array

Malla de triángulos con índices.

```
class MallaInd {
public:
  GLuint numDat;   // tabla de vértices y normales
  PVec3 *vertices, * normales; // también pueden ser vector<PVec3>
  GLuint numInd;  Gluint * indices; // tabla de índices
  MallaInd(){ numDat = 0; vertices = normales = nullptr;
              numInd = 0; indices = nullptr; }
  ~MallaInd(){ delete[] vertices; delete[] normales; delete[] indices; }
  bool load(char arch[]);  // generar los datos
  void draw();
protected:
  void activar();
  void desactivar();
};
```


OpenGL vertex array

```
void MallaInd::draw(){
  activar();
  glColor4d(r,g,b,a);
  glDrawElements(GL_TRIANGLES, numInd, GL_UNSIGNED_INT, indices);
  desactivar();
}
// glDrawElements(Primitiva, NºInd, Tipo, PtrArray);

void MallaInd::activar() {
// -> MallaTri::activar()
}
void MallaInd::desactivar() {
// -> MallaTri::desactivar()
}
```

Cargar una malla con índices

```
bool MallaInd::load(char arch[]) {
  abrir el archivo de datos
  leer número de datos y de índices (índices=3*caras)
  delete[] vertices; delete[] normales;  // liberar tablas
  delete[] indices;
  malla.vértices = new PVec3[numDat]; malla.normales = new PVec3[numDat];
  malla.indices = new GLuint[numInd];
  for each i in [0 … numDat)  // leer los vértices
    leer vertices[i]          // 3 coordenadas cada vértice
  for each i in [0 … numDat)  // leer las normales
    leer normales[i]          // 3 coordenadas cada vector
  for each i in [0 … numInd)  // leer los índices de los
triángulos
    leer indices[i]
}
```


Vertex Buffer Objects (VBO)

- Los Vertex Array de OpenGL no almacenan los datos en la GPU.

Cada vez que se ejecuta

```
glDrawArrays(Primitiva, IndPrimero, NºDatos);
glDrawElements(Primitiva, NºInd, Tipo, PtrArray);
```

se pasan los datos a la GPU.

Cualquier modificación realizada en la CPU se pasa a la GPU.

- Los Vertex Buffer Objects permiten almacenar los datos en la GPU.

Parecidos a los Texture Objets