

# Informática Gráfica Modelos Geométricos

Juan Carlos Torres  
**Grupos C y D**

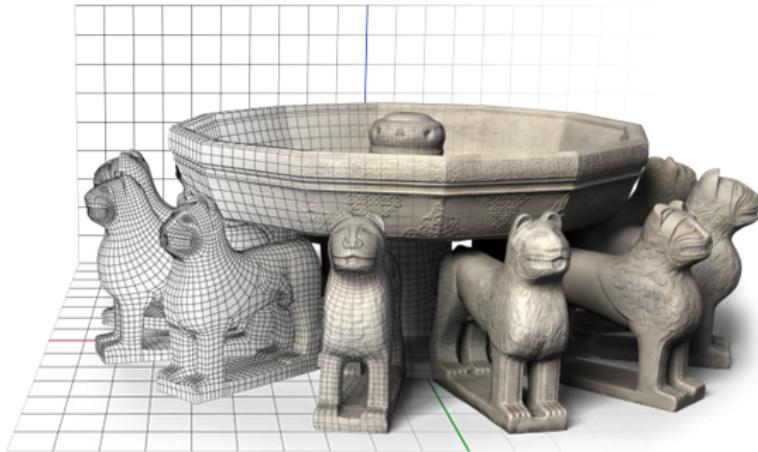
Dpt. Lenguajes y Sistemas Informáticos  
ETSI Informática y de Telecomunicación  
Universidad de Granada

---

Curso 2024-25

# Modelos Geométricos

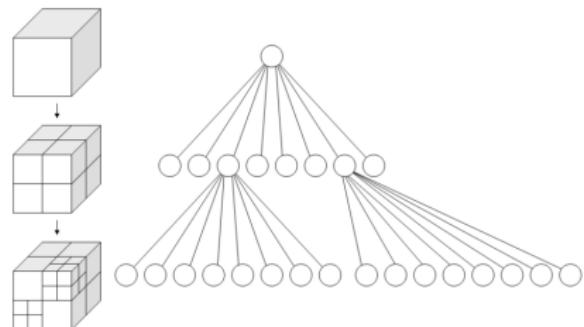
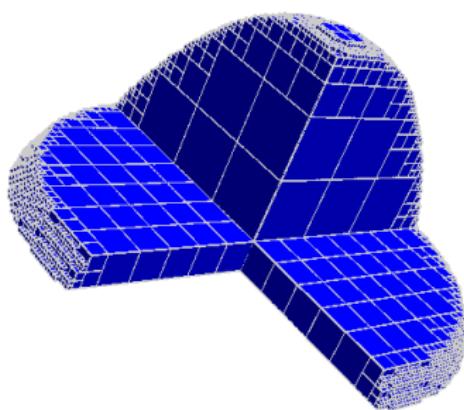
# Modelo geométrico



- El modelo describe tanto la forma como la apariencia del objeto.
  - La forma del objeto es esencialmente información geométrica.
  - La apariencia del objeto describe como se comporta ante la luz que recibe.
- En este tema nos centraremos en las propiedades geométricas.
- La representación geométrica de un objeto es un **modelo geométrico**.

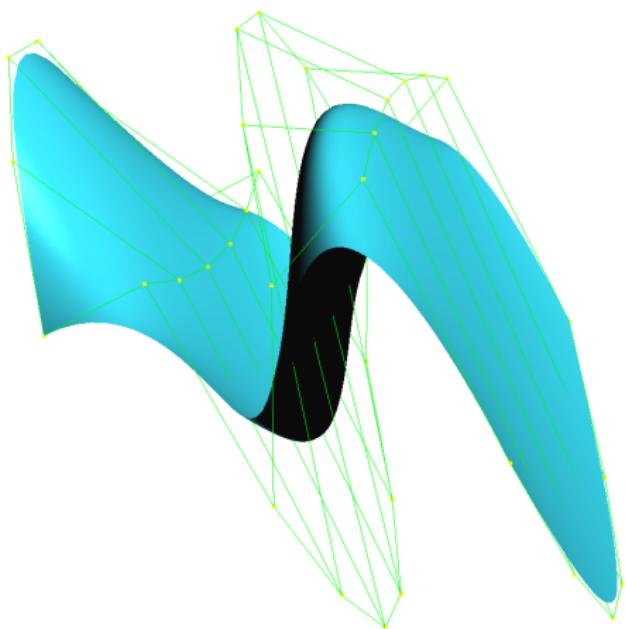
# Modelo geométrico: Octrees

- Representación por descomposición del espacio.
- Cada nodo se corresponde con un volumen cubico.
- Los nodos hijos subdividen a su nodo padre.
- Los nodos terminales indican si su volumen está ocupado por el objeto.



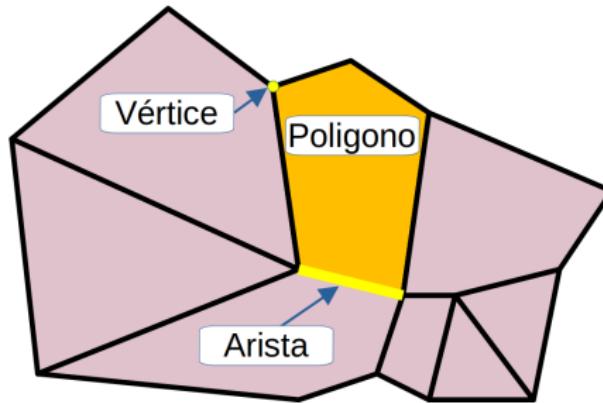
# Modelo geométrico: Superficies paramétricas

- La superficie se representa mediante una rejilla de puntos.
- La geometría del objeto está descrita por funciones polinómicas de los puntos de la rejilla.



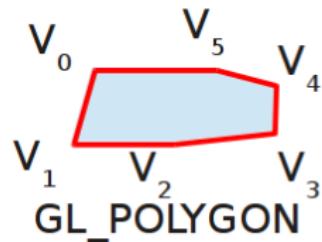
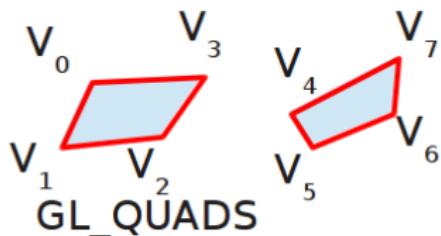
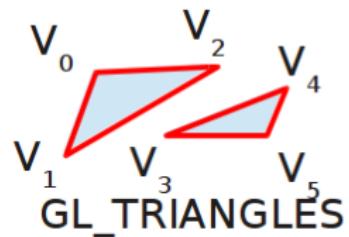
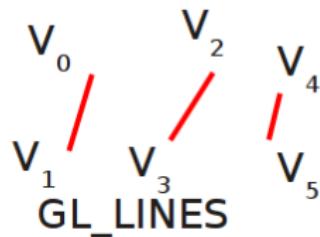
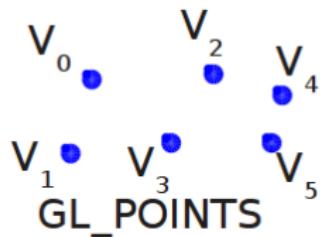
# Modelo geométrico: Superficies poligonales

- La superficie se representa mediante un conjunto de polígonos.
- Polígonos delimitados por secuencia de aristas y vértices.
- Polígonos vecinos están conectados por una arista común.



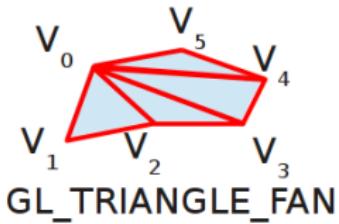
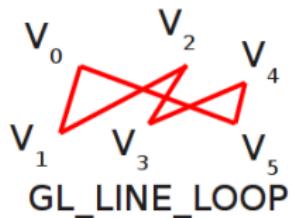
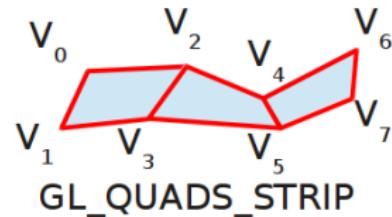
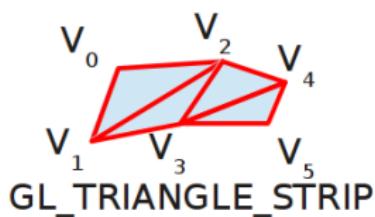
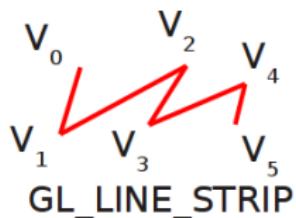
# Primitivas

```
glBegin( GL_TRIANGLES );
    glNormal3f( 0.0, 1.0, 0.0 );
    glVertex3f( t, 0, t );
    ...
glEnd();
```



# Primitivas

- Para optimizar el dibujo de mallas poligonales existen primitivas que empaquetan varios elementos simples.
- Con estas primitivas los vértices internos se procesan menos veces.

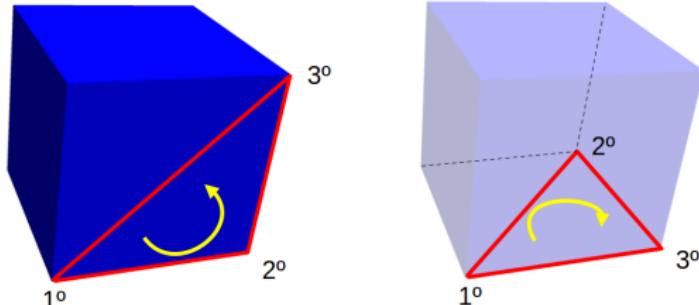


# Ejemplo simple

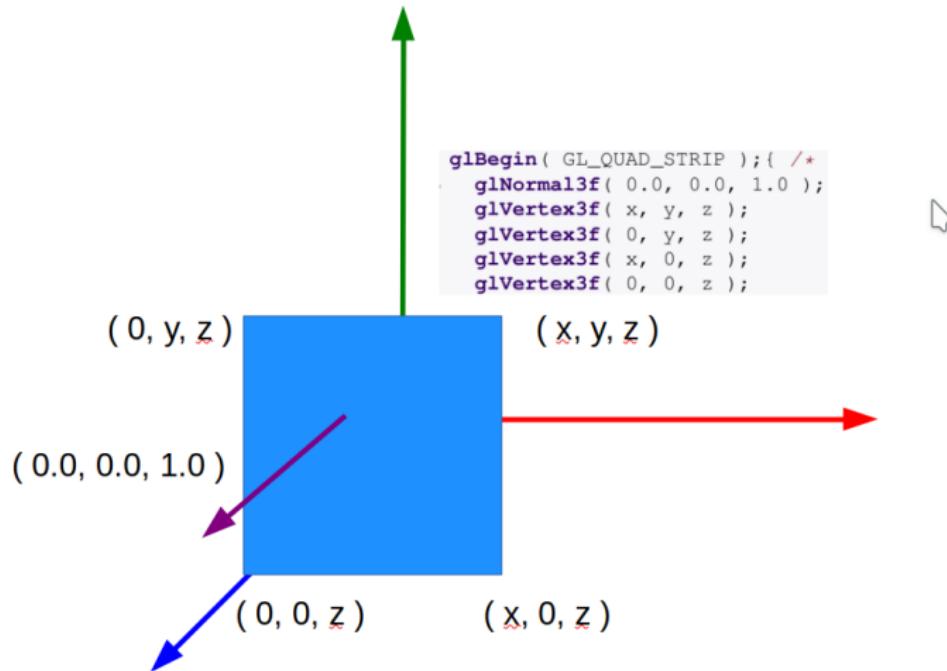
```
void cubo( GLfloat x, GLfloat y, GLfloat z )
//Construye una caja con un vertice en origen y otro en (x,y,z)
{
    float color[4]={0.8,0.0,1,1};
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, color );
    glBegin( GL_QUAD_STRIP );{ /* Caras transversales */
        glNormal3f( 0.0, 0.0, 1.0 );      /*Vertical delantera*/
        glVertex3f( x, y, z );
        glVertex3f( 0, y, z );
        glVertex3f( x, 0, z );
        glVertex3f( 0, 0, z );
        glNormal3f( 0.0, -1.0, 0.0 );     /*Inferior */
        glVertex3f( x, 0, 0 );
        glVertex3f( 0, 0, 0 );
        glNormal3f( 0.0, 0.0, -1.0 );    /* Vertical hacia atras */
        glVertex3f( x, y, 0 );
        glVertex3f( 0, y, 0 );
        glNormal3f( 0.0, 1.0, 0.0 );     /* Superior, horizontal */
        glVertex3f( x, y, z );
        glVertex3f( 0, y, z );
    glEnd();
}
```

# Ejemplo simple

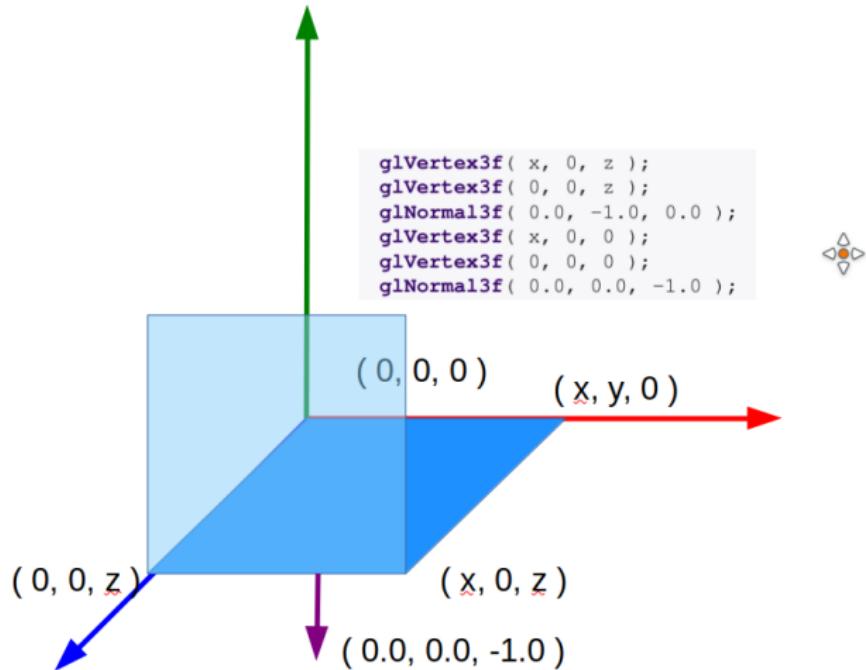
```
glBegin( GL_QUADS ); { /* Costados */
    glNormal3f( 1.0, 0.0, 0.0 );
    glVertex3f( x, 0, 0 );
    glVertex3f( x, y, 0 );
    glVertex3f( x, y, z );
    glVertex3f( x, 0, z );
    glNormal3f( -1.0, 0.0, 0.0 );
    glVertex3f( 0, 0, 0 );
    glVertex3f( 0, 0, z );
    glVertex3f( 0, y, z );
    glVertex3f( 0, y, 0 );
}
glEnd();
```



# Ejemplo simple

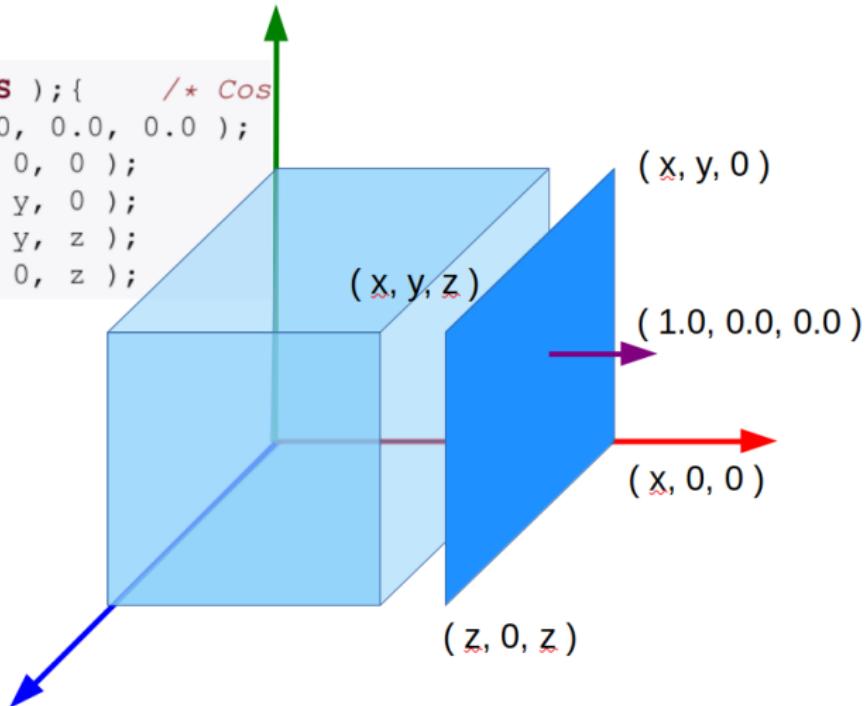


# Ejemplo simple



# Ejemplo simple

```
glBegin( GL_QUADS ) { /* Cos  
    glNormal3f( 1.0, 0.0, 0.0 );  
    glVertex3f( x, 0, 0 );  
    glVertex3f( x, y, 0 );  
    glVertex3f( x, y, z );  
    glVertex3f( x, 0, z );
```

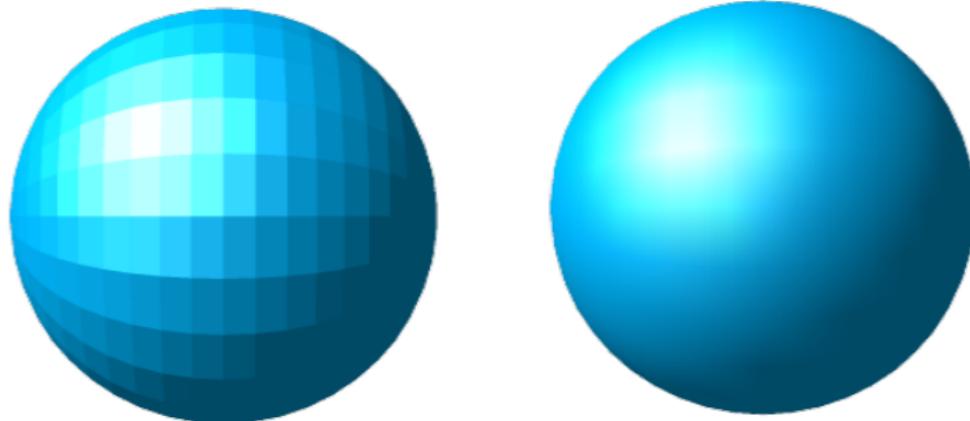


# Dibujo de superficies suaves

El color depende de la orientación de la superficie.

- Se calcula en función de la normal.
- Para visualizar una superficie suave se asigna la normal a los vértices.

```
glShadeModel(GL_FLAT)    // dibuja caras planas  
glShadeModel(GL_SMOOTH) // dibuja superficies suaves
```



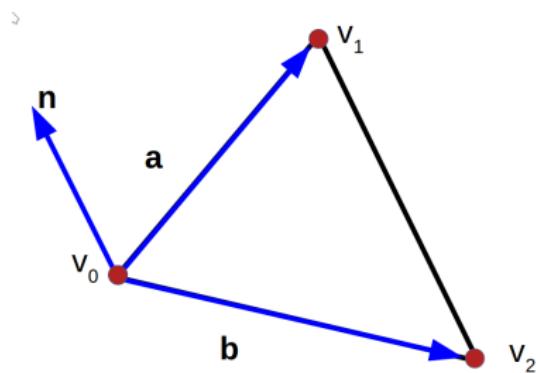
# Cálculo de normales

La normal a un triángulo se puede calcular haciendo el producto vectorial de dos aristas.

$$\mathbf{a} = \mathbf{v}_1 - \mathbf{v}_0$$

$$\mathbf{b} = \mathbf{v}_2 - \mathbf{v}_0$$

$$\mathbf{n} = (\mathbf{a} \times \mathbf{b}) / \| \mathbf{n} \|$$



El vector  $\mathbf{n}$  apunta según la regla de la mano derecha.

Para que la orientación de las normales sea consistente la orientación de todos los triángulos debe ser la misma.

# Cálculo de normales de vértices

Si la ecuación de la superficie es conocida se puede calcular la normal analíticamente.

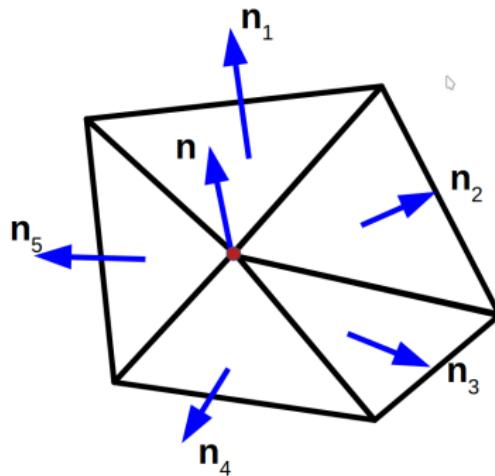
En caso contrario se puede calcular la normal en los vértices promediando las normales de las caras:

$$\mathbf{n} = \frac{\sum_{i=1}^k \mathbf{n}_i}{\|\sum_{i=1}^k \mathbf{n}_i\|}$$

siempre que

$$\left\| \sum_{i=1}^k \mathbf{n}_i \right\| \neq 0$$

siendo  $\mathbf{n}_i$  la normal en la cara  $i$  de las  $k$  que comparten el vértice.

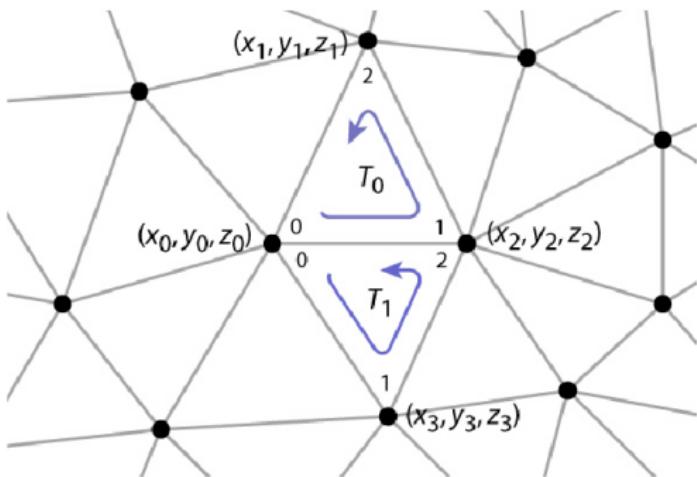


# Sopa de triángulos

Una **sopa de triángulos** es un **conjunto** de triángulos almacenados de forma independiente, que representan toda o parte de la superficie de un objeto (este es el ejemplo de estructura simple visto en el tema anterior).

- Pueden ser adyacentes.
- Pero no tienen información de sus vecinos.

	[0]	[1]	[2]
tris[0]	$x_0, y_0, z_0$	$x_2, y_2, z_2$	$x_1, y_1, z_1$
tris[1]	$x_0, y_0, z_0$	$x_3, y_3, z_3$	$x_2, y_2, z_2$
:	:	:	



# Sopa de triángulos: representación

Una **sopa de triángulos** se puede almacenar como

- Una lista de triángulos,
- en la que cada triángulo se almacena como tres vértices,
- y cada vértice tiene tres coordenadas.

```
typedef struct {
    float x, y, z;
} vertex;

typedef struct {
    vertex a, b, c;
} triangle;

typedef struct {
    int n;
    triangle* t;
} mesh;
```

# Sopa de triángulos: visualización

Para visualizar el modelo se recorre la lista dibujando cada triángulo. Para permitir que OpenGL elimine las caras traseras, los vértices deben estar ordenados de forma consistente, normalmente en sentido antihorario (CCW).

```
glBegin(GL_TRIANGLES);
for(i=0;i<Mesh.n;i++) {
    glVertex3f(Mesh.t[i].a.x,Mesh.t[i].a.y,Mesh.t[i].a.z);
    glVertex3f(Mesh.t[i].b.x,Mesh.t[i].b.y,Mesh.t[i].b.z);
    glVertex3f(Mesh.t[i].c.x,Mesh.t[i].c.y,Mesh.t[i].c.z);
}
glEnd();
```

Se puede calcular la normal de cada triángulo antes de dibujarlo o tenerla precalculada.

# Sopa de triángulos: propiedades

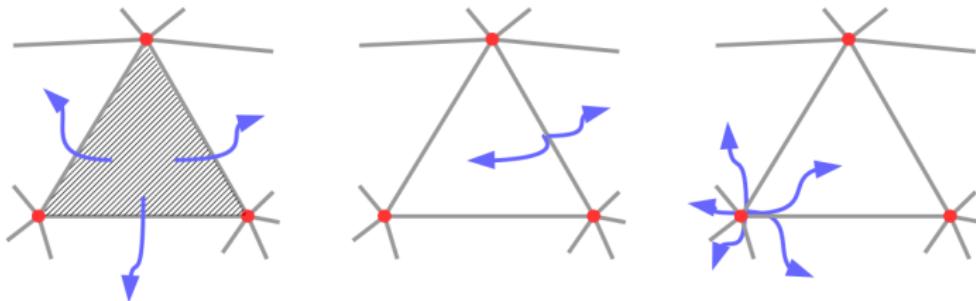
Esta representación tiene varios inconvenientes:

- Tiene información redundante.
- Pueden aparecer fisuras por errores de representación.
- No se puede usar para representar sólidos.
- No se pueden realizar operaciones de edición, salvo a nivel de triángulo.

# Malla de triángulos

Una **malla de triángulos** es una representación de un conjunto de triángulos conectados que describen una superficie y que almacena información topológica además de información geométrica.

- La información topológica aporta relaciones de adyacencia entre elementos.
- Permite recorrer la superficie.
- De cada elemento geométrico se pueden obtener sus triángulos, vértice y/o arista vecinas.



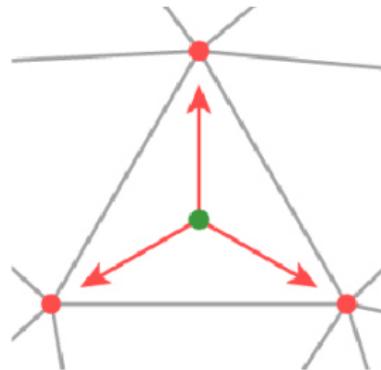
# Mallas de triángulos: Índice de vértices

Es la forma mas simple de representar una malla de triángulos.

Para evitar la redundancia de vértices se usa índices de vértices. La representación consta de:

- Una tabla de vértices.
- Una tabla de triángulos.

```
vertice {  
    float coordenada[3];  
}  
  
Mesh {  
    int nv,nt;  
    float vertice[maxVer][3];  
    int triangulo[maxTri][3];  
}
```

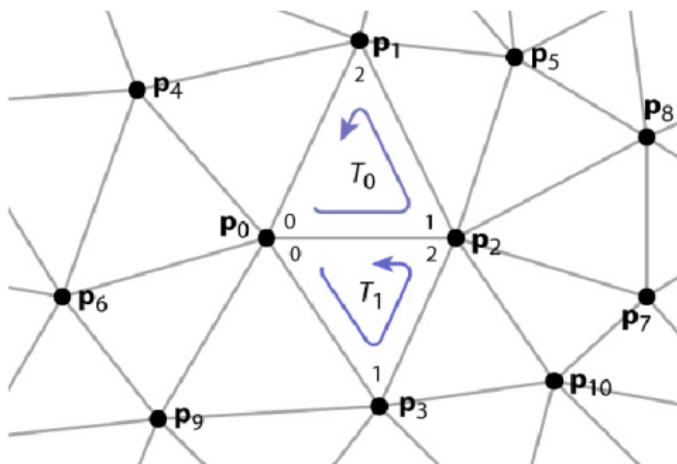


# Mallas de triángulos: Índice de vértices

- Cada vértice se almacena una sola vez.
- Cada triángulo apunta a sus tres vértices.

verts[0]	$x_0, y_0, z_0$
verts[1]	$x_1, y_1, z_1$
	$x_2, y_2, z_2$
	$x_3, y_3, z_3$
:	

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
:	



☞ <http://www.sci.utah.edu/~abe/EditMM/doc/Mulit-Resolution Mesh Editing.html>

# Formato PLY

Para almacenar los modelos 3D de forma persistente se utilizan formatos especiales de archivo.

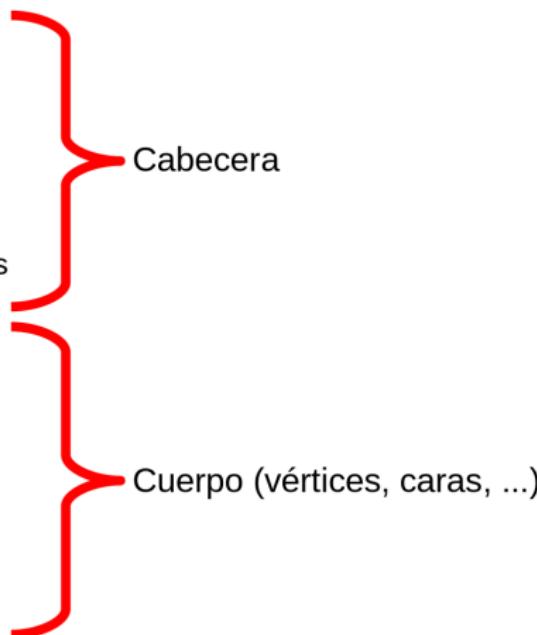
Hay muchos formatos, la mayoría son propietarios y cerrados (STL, OBJ, FBX, COLLADA, 3DS, IGES, STEP,X3D, PLY, BLEND....).

Cada formato puede almacenar datos diferentes, organizados de forma distinta.

# Formato PLY

PLY es un formato abierto, sencillo y configurable, diseñado por la Universidad de Stanford (<http://paulbourke.net/dataformats/ply/>).

```
ply
format ascii 1.0
element vertex 510272
property float32 x
property float32 y
property float32 z
element face 108176
property list uint8 int32 vertex_indices
end_header
0.971510 -1.341210 0.568620
0.982640 -1.344680 0.575240
0.989330 -1.341980 0.579250
.....
0.044910 5.677040 12.747991
3 57004 114239 114246
3 57003 57004 114239
3 57003 114232 114239
3 57002 57003 114232
.....
3 57002 114225 114232
3 57001 57002 114225
```



# Formato PLY: lectura de un modelo

```
ply
format ascii 1.0
element vertex 510272
property float32 x
property float32 y
property float32 z
element face 108176
property list uint8 int32 vertex_indices
end_header
0.971510 -1.341210 0.568620
0.982640 -1.344680 0.575240
0.989330 -1.341980 0.579250
.....
0.044910 5.677040 12.747991
3 57004 114239 114246
3 57003 57004 114239
3 57003 114232 114239
3 57002 57003 114232
.....
3 57002 114225 114232
3 57001 57002 114225
```

→ ( Comprobar que el vértice no existe)  
Añadir a la tabla de vértices  
Incrementar número de vértices

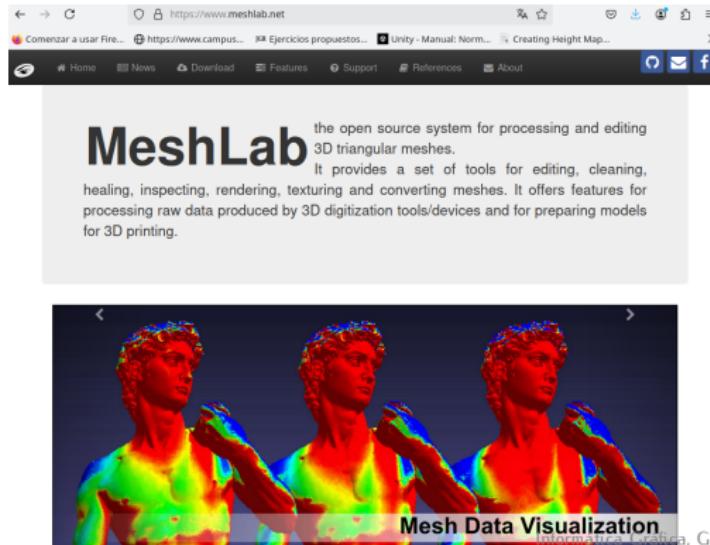
# Formato PLY: lectura de un modelo

```
ply
format ascii 1.0
element vertex 510272
property float32 x
property float32 y
property float32 z
element face 108176
property list uint8 int32 vertex_indices
end_header
0.971510 -1.341210 0.568620
0.982640 -1.344680 0.575240
0.989330 -1.341980 0.579250
.....
0.044910 5.677040 12.747991
3 57004 114239 114246
3 57003 57004 114239
3 57003 114232 114239
3 57002 57003 114232
.....
3 57002 114225 114232
3 57001 57002 114225
```

( Comprobar que la cara no existe)  
Añadir a la tabla de caras  
Incrementar número de caras

# MeshLab

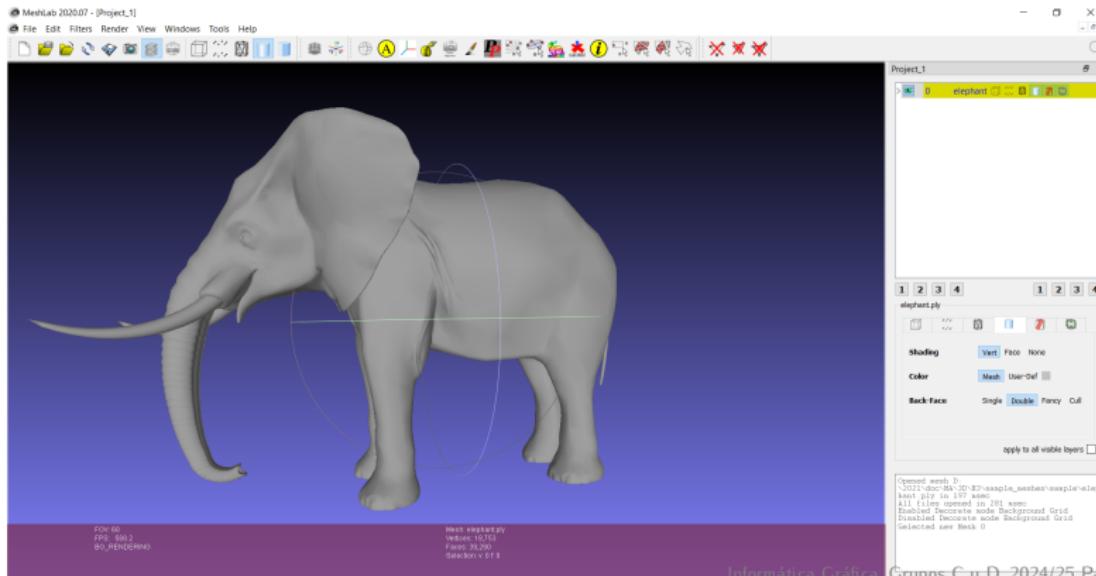
- Procesar y editar mallas triangulares 3D
- Software libre y de código abierto.
- Funciones para la limpieza, reparación, inspección y renderizado.
- Importación y exportación a numerosos formatos.
- Disponible para Linux, Windows y Mac.



# MeshLab

## Áreas de la interfaz:

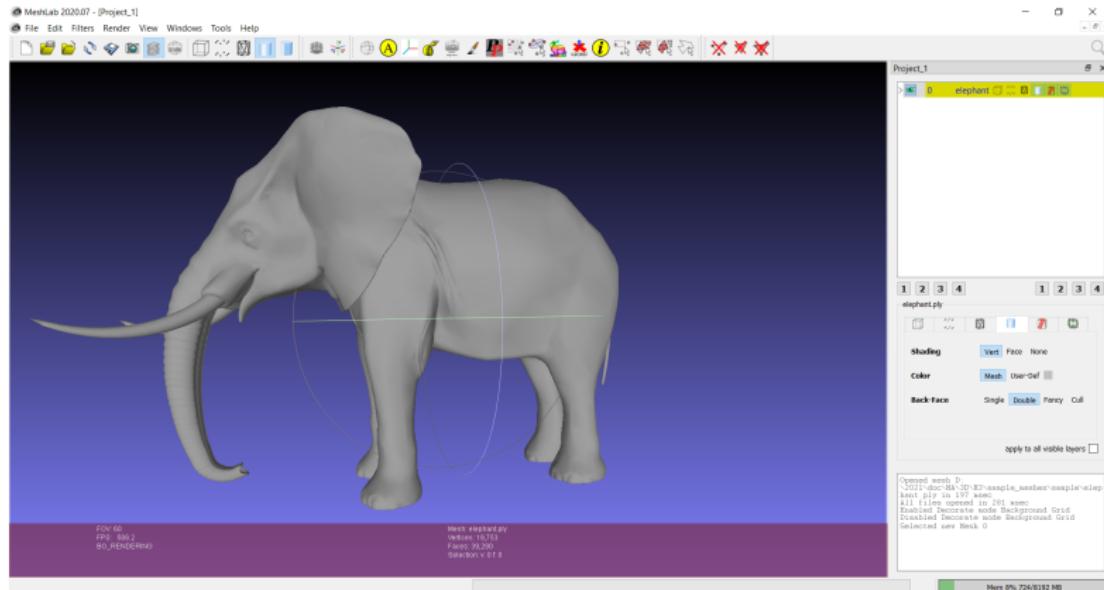
- Menús
- Proyecto
- Parámetros
- Información



# MeshLab

## Control de cámara

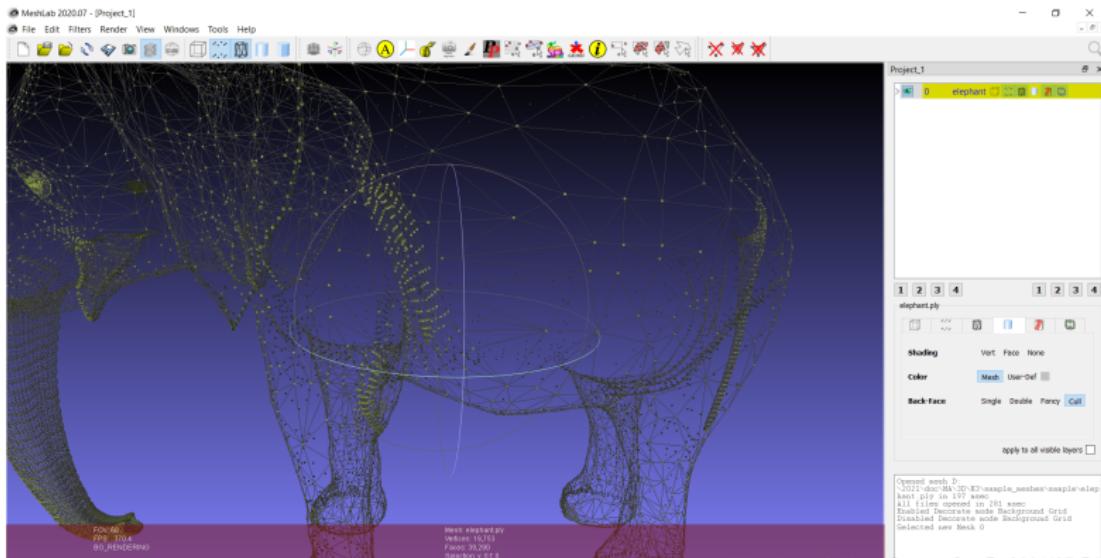
- Botón izquierdo: giro (en trackball)
- Rueda: zoom
- Botón derecho: desplazar



# Visualización

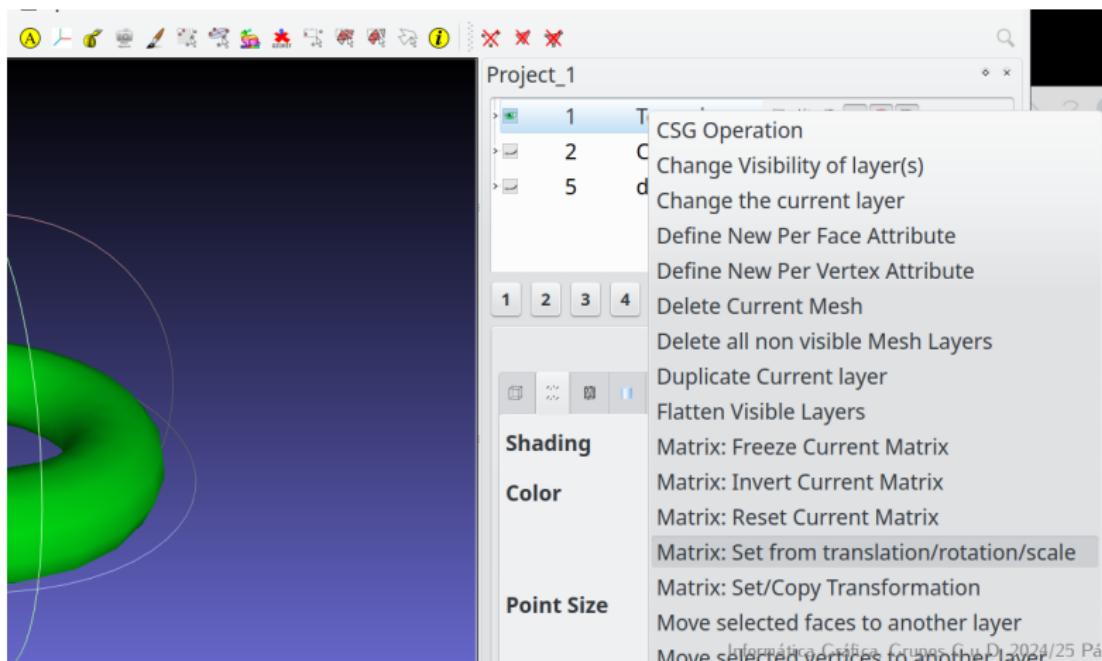
## Opciones

- Que se dibuja: Puntos, Aristas, Caras
- Modo de visualización de caras: planas, suavizadas
- Que lado: simple, doble, solo delantera (cull)
- Soimbreado: vértices, caras, ninguno



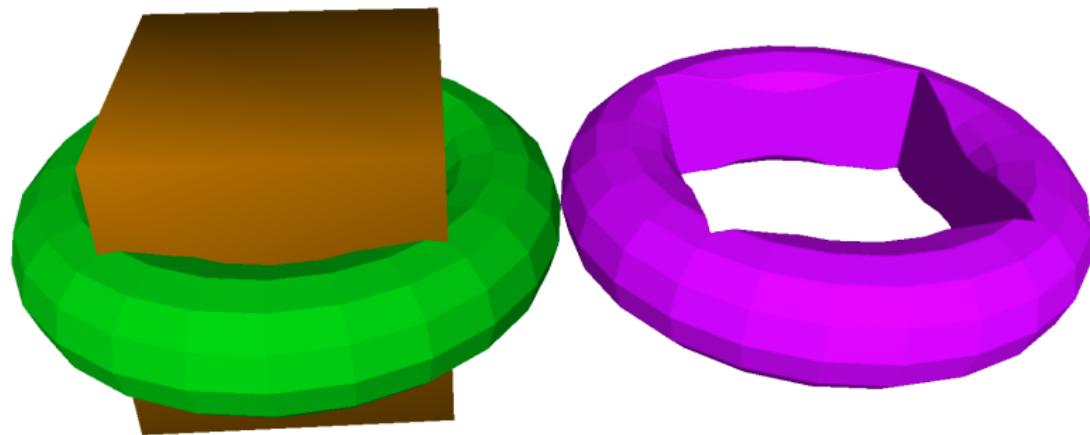
# Carga de modelos

- Importación de modelos
- Creación de modelos simples (Filters>Create New Mesh Layer)
- Transformación: botón derecho en la malla en el panel del proyecto > Matrix: Set from Translation/Rotation/Scale



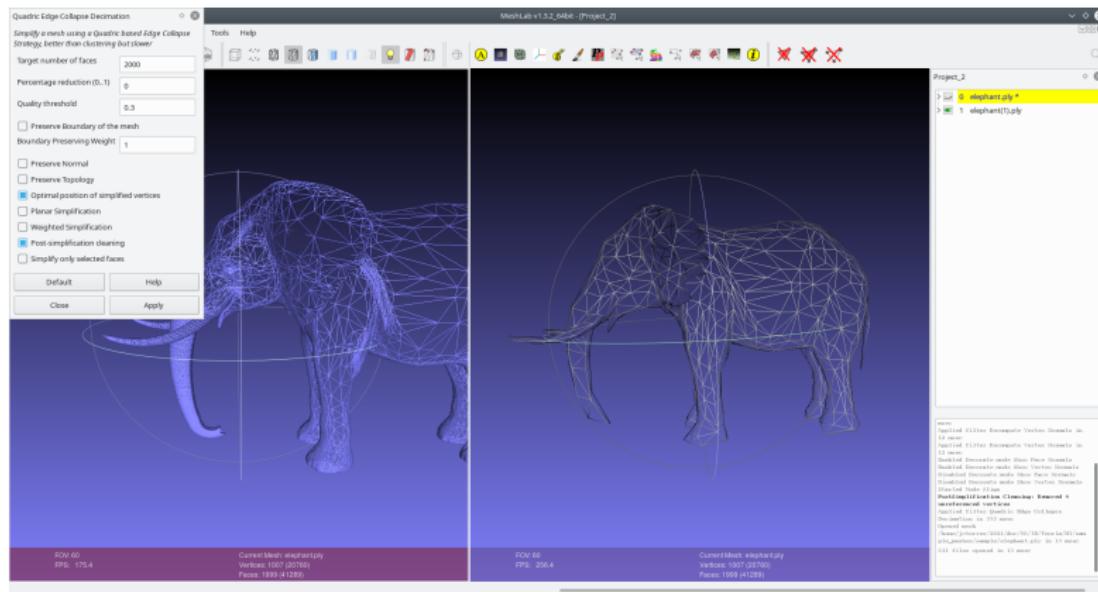
# Operaciones booleanas

Transformación: botón derecho en la malla en el panel del proyecto > CSG Operation



# Simplificación

Filters > Remeshing, simplification and Reconstruction > Quadratic Edge Collapse Decimation



# Exportación

File > Export Mesh As...

Usar formato Alias Wavefront Object

