



Daniel Calderon S.

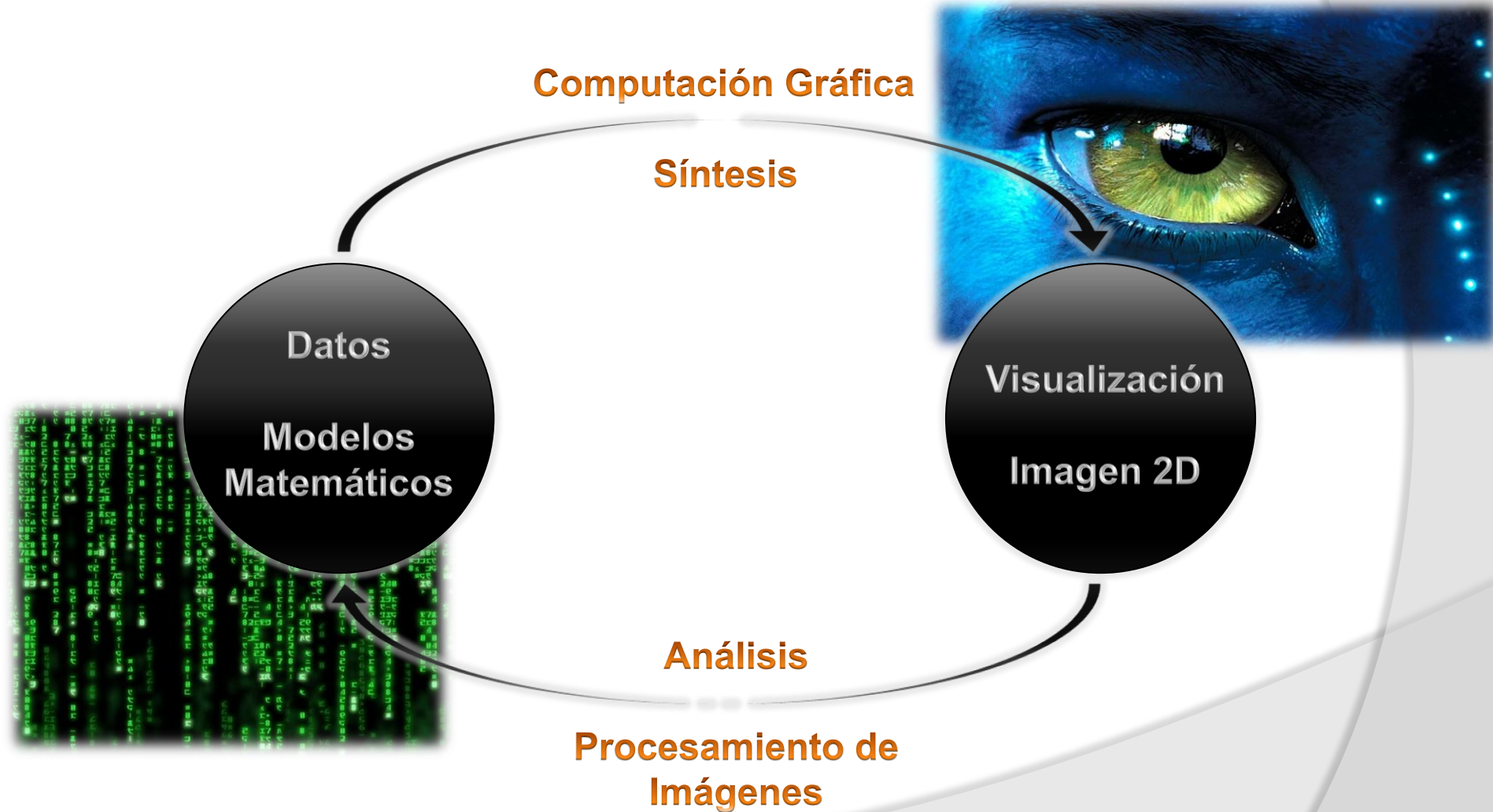
# COMPUTACIÓN GRÁFICA 3D

# Contenido

- ⦿ Introducción
- ⦿ Modelos 3D
- ⦿ Cámaras
- ⦿ Iluminación
  - Fuentes de luz
  - Materiales
  - Normales

# INTRODUCCIÓN

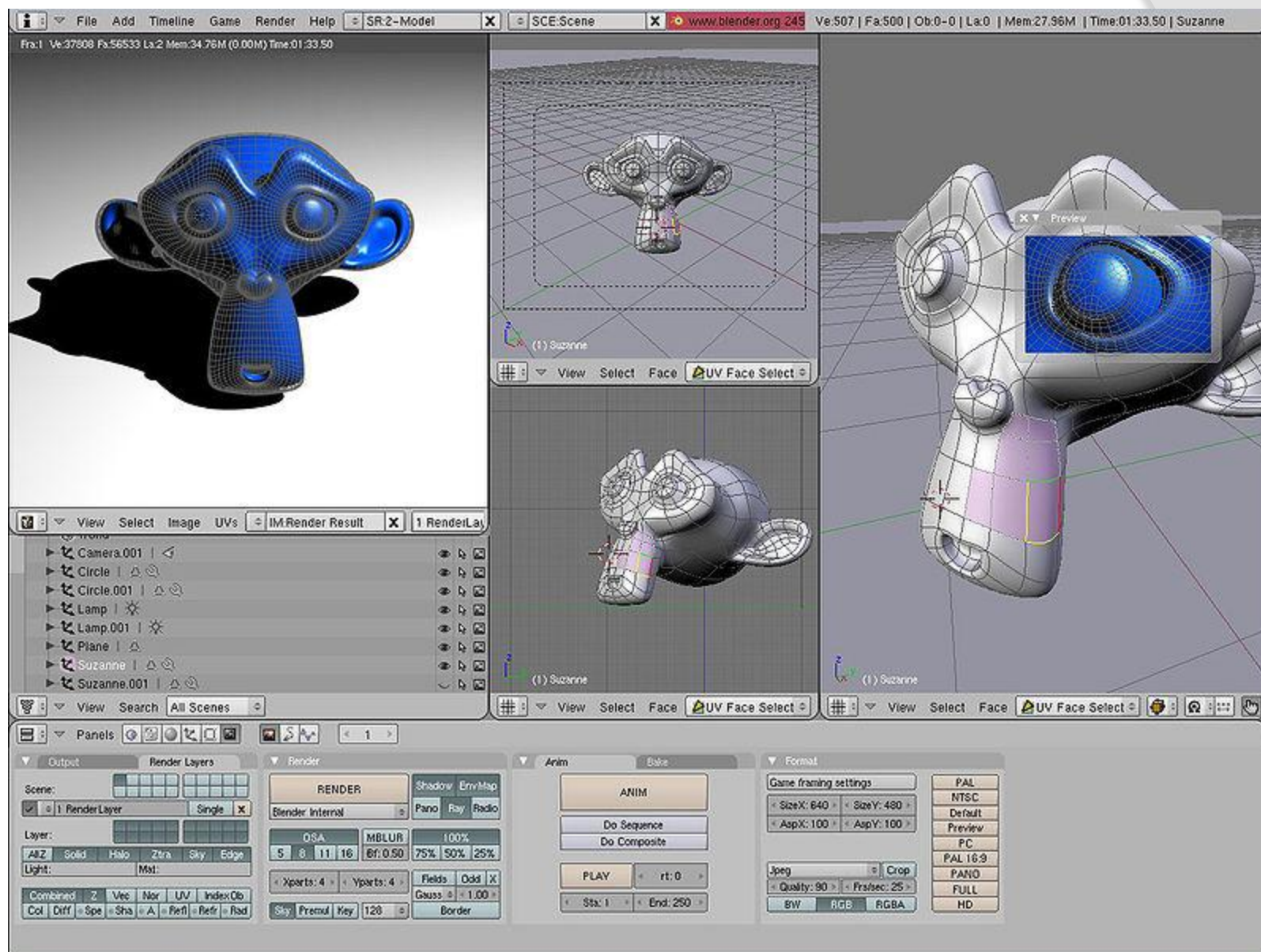
# Modelo - Visualización



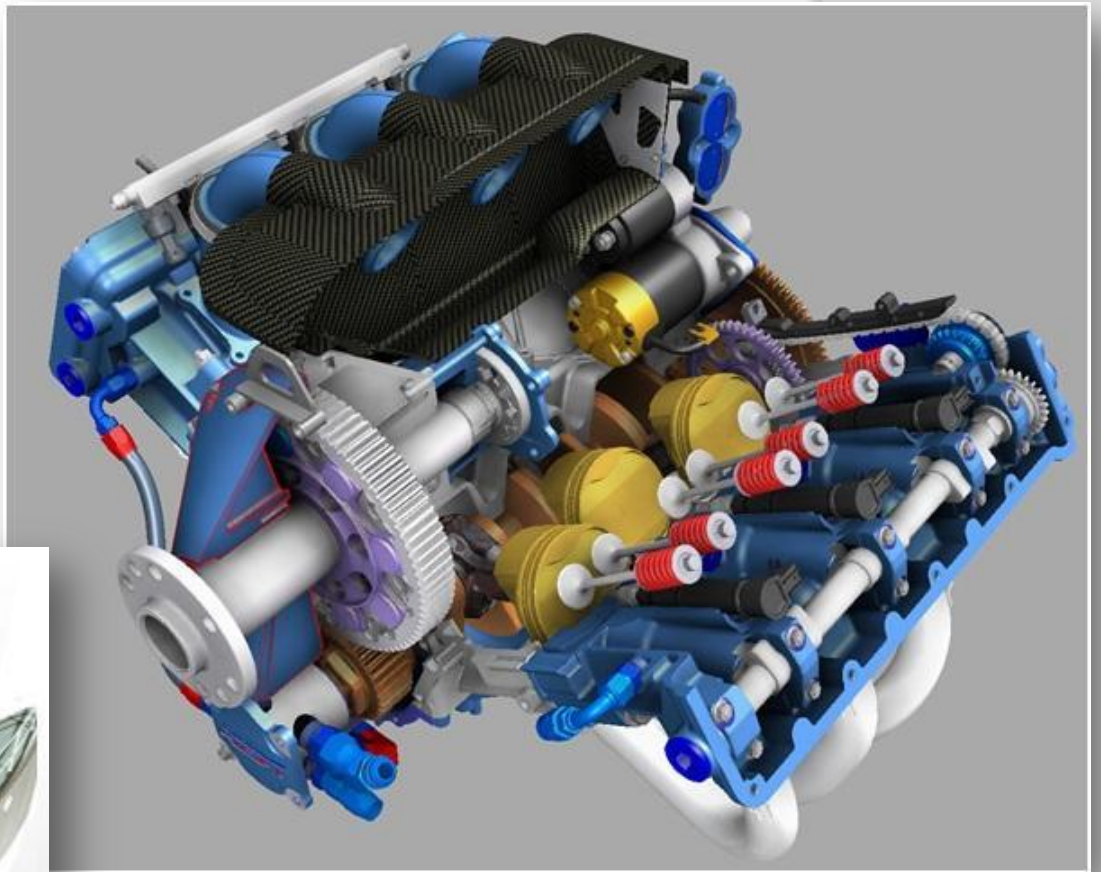














# Computador = Mundo Discreto

- ⦿ Se dibuja en base a polígonos simples.
- ⦿ Se tiene una malla tridimensional, formada por vértices y polígonos.



# CG = Tarea Pesada

- Computación gráfica
  - Tarea pesada
  - Procesar vértices
  - Procesar pixeles
- Solución:
  - Hardware específico
  - La GPU
    - Muchos procesadores en paralelo!



# Evolución de las GPU



Pong, 1972



Pac-Man, 1983



Wolfenstein 3D, 1992



Doom, 1993

Duke Nukem 3D, 1996



Quake 3 Arena, 1999



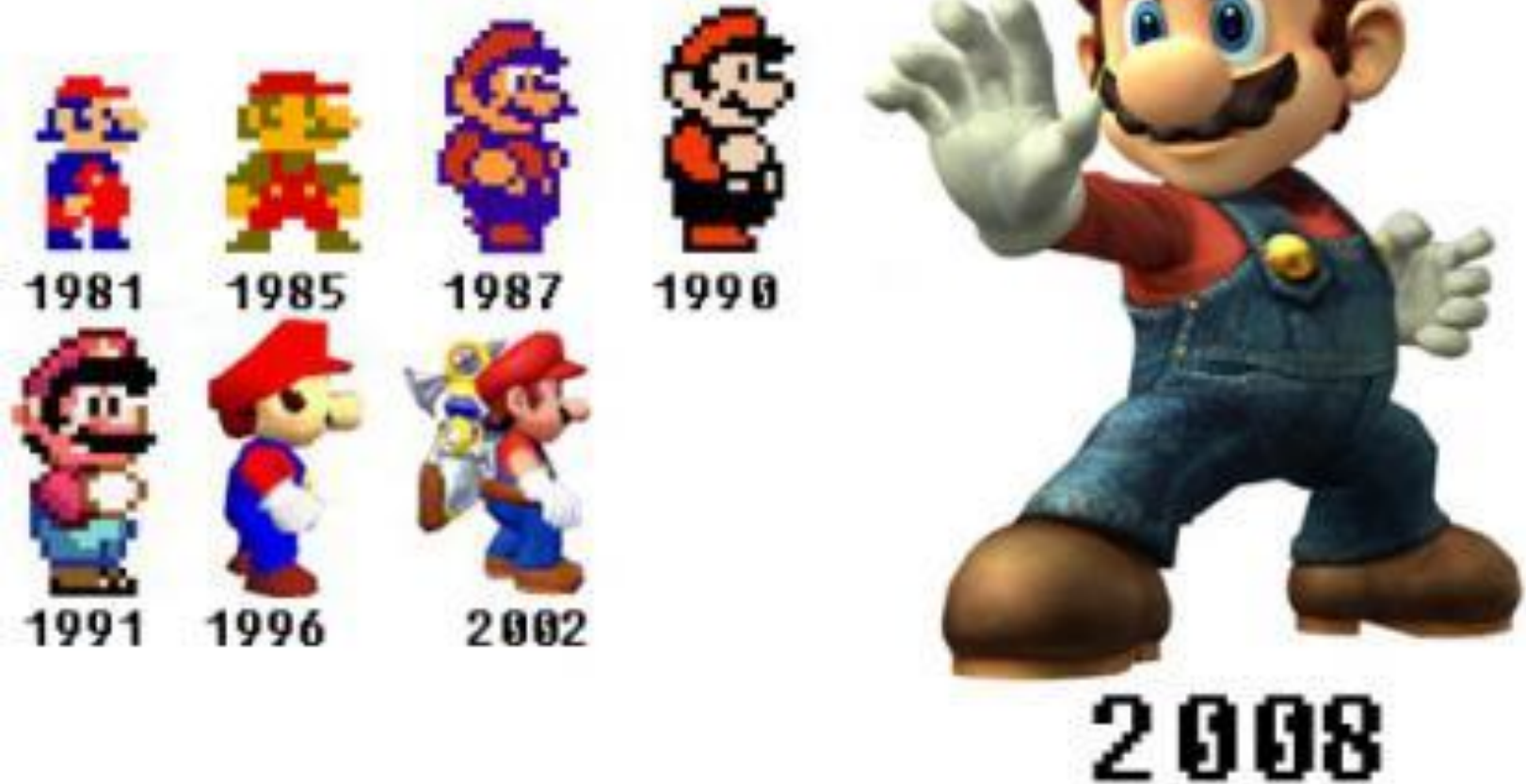
Half-Life 2, 2004



Quake 4, 2005



# The Evolution of Mario



\*Dates based on US release dates.

Geekstir.com



# CG en el cine...

Tron (1982)

- Intervienen imágenes generadas por computadora.

The Abyss (1989),  
Terminator 2 (1991)

- Las imágenes generadas por computadora adquieren un rol importante.

Jurassic Park  
(1993)

- Mezcla perfecta

Toy Story (1995)

- Primer largometraje

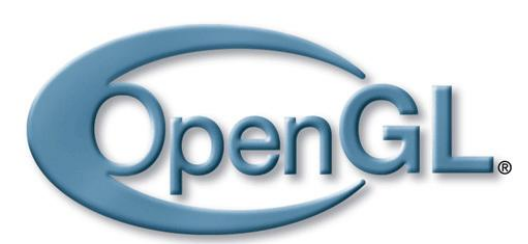
Final Fantasy: The  
Spirits Within (2001)

- Primera película realista

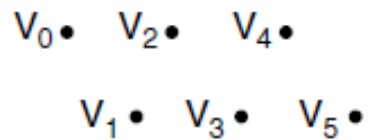
# Modelación 3D

## ⦿ Necesitamos:

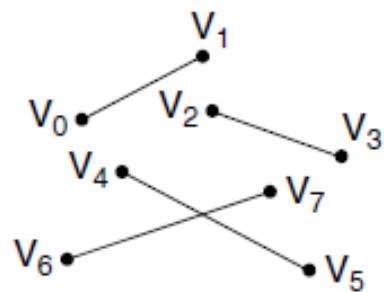
- Un modelo tridimensional, en base a vértices y polígonos que los unan.
- Caracterización de los materiales que conforman las distintas partes del modelo.
- Fuentes de luz.
- Una cámara.



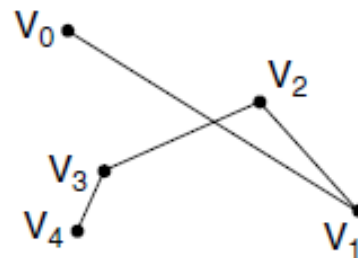
# MODELOS EN 3D



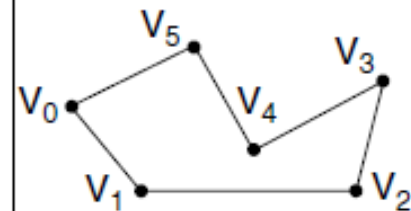
GL\_POINTS



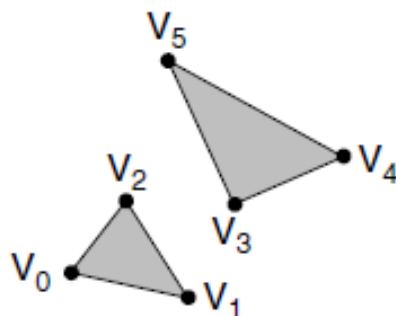
GL\_LINES



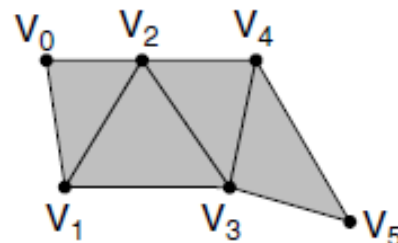
GL\_LINE\_STRIP



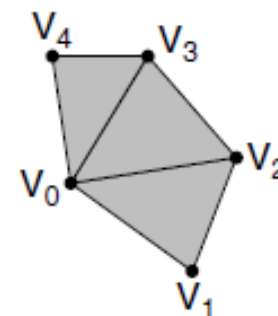
GL\_LINE\_LOOP



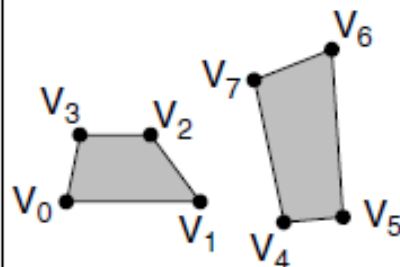
GL\_TRIANGLES



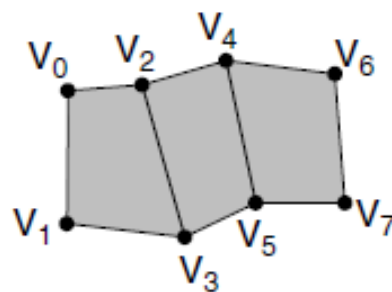
GL\_TRIANGLE\_STRIP



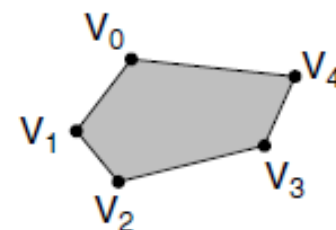
GL\_TRIANGLE\_FAN



GL\_QUADS



GL\_QUAD\_STRIP



GL\_POLYGON

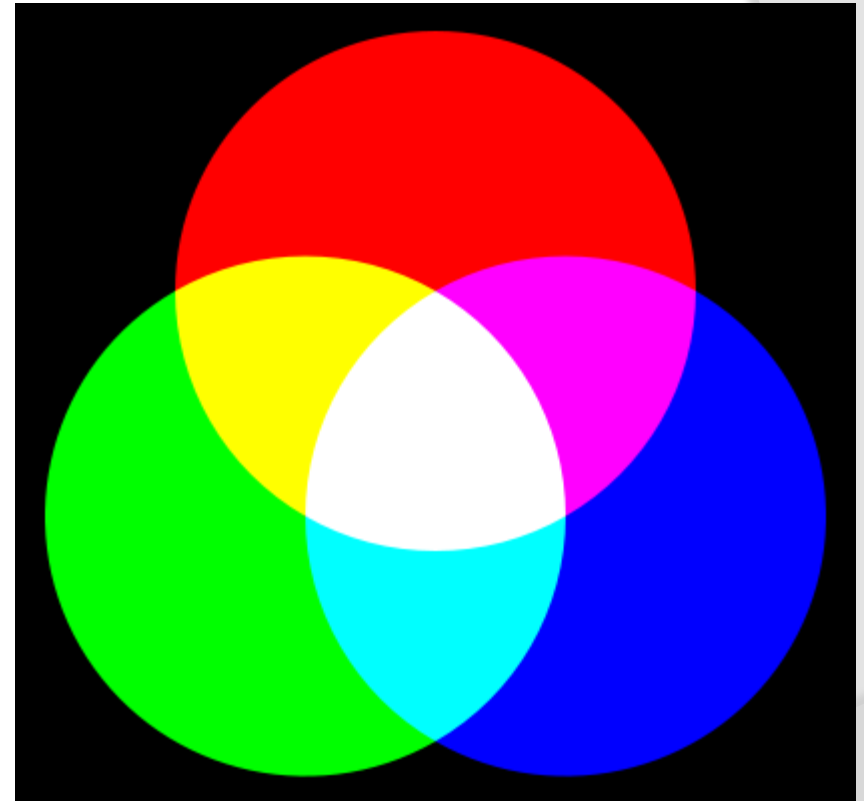


# Modelo de color

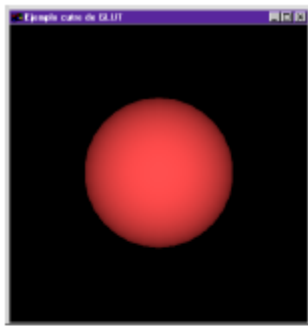
- ⦿ 4 componentes:

- Rojo
- Verde
- Azul
- Transparencia (alpha)

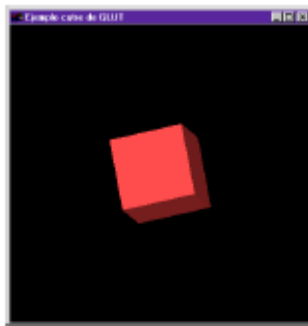
- ⦿  $[0,1]$ : rango de mínima a máxima intensidad.



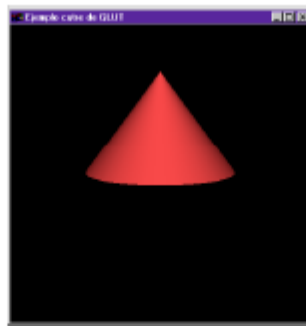
# Figuras GLUT



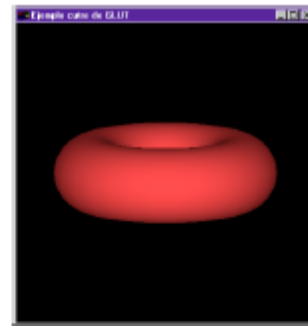
sphere



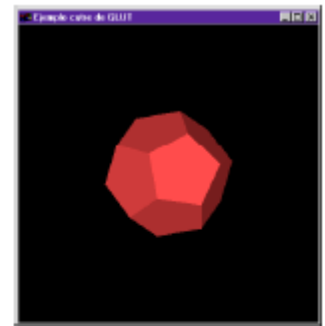
cube



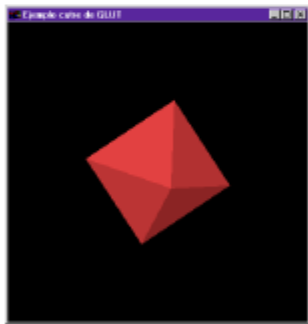
cone



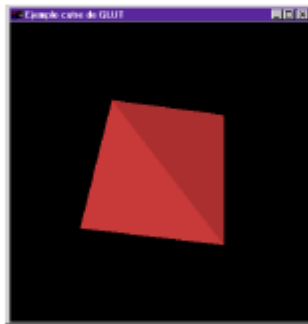
torus



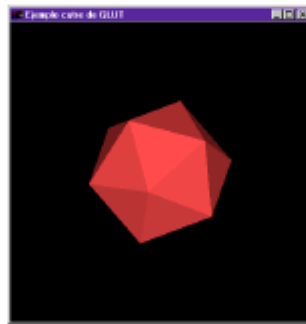
dodecahedron



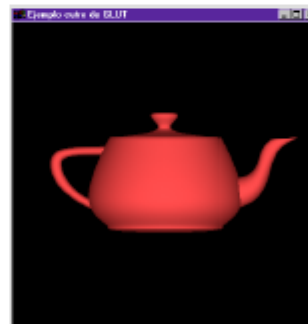
octahedron



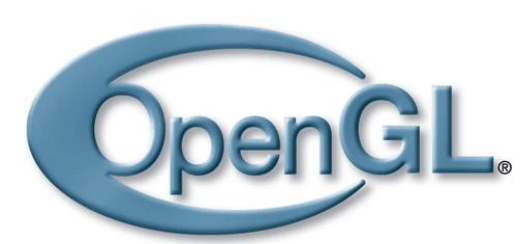
tetrahedron



icosahedron

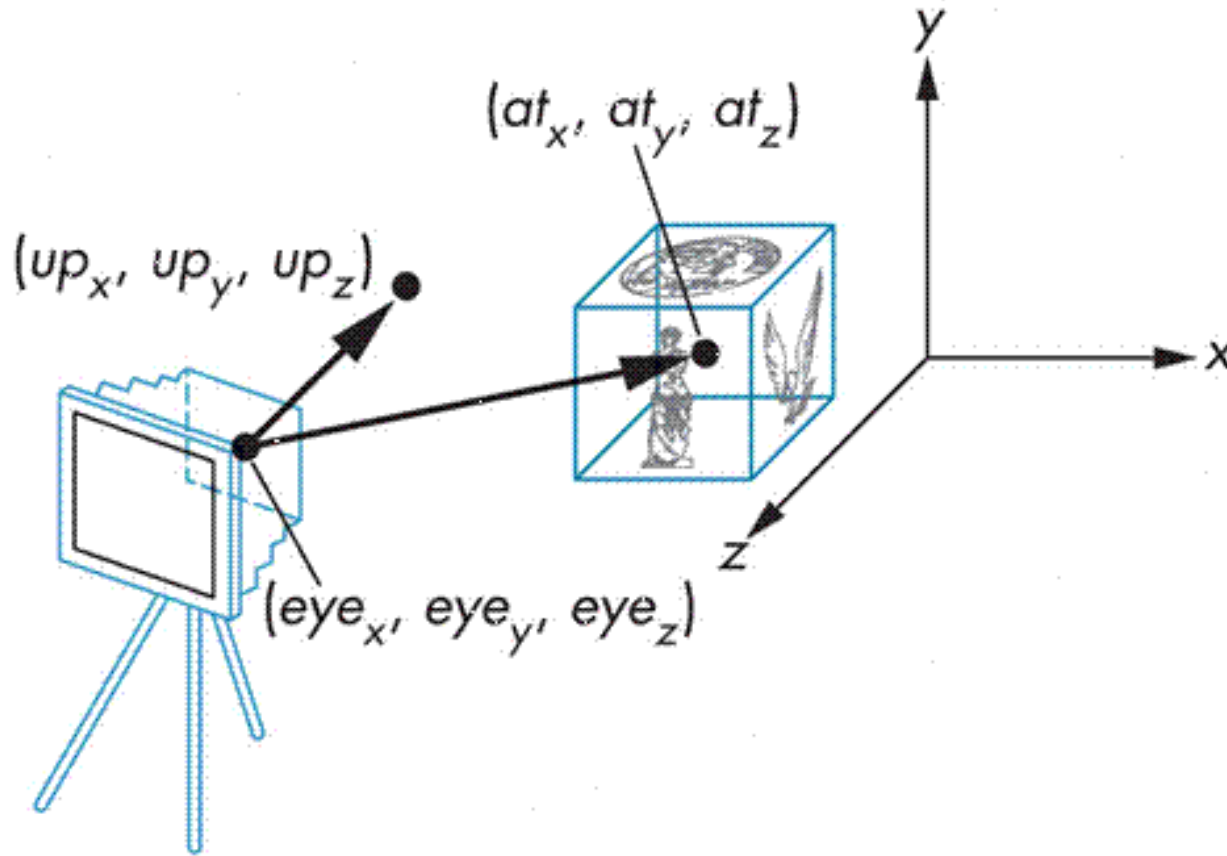


teapot



# CÁMARAS

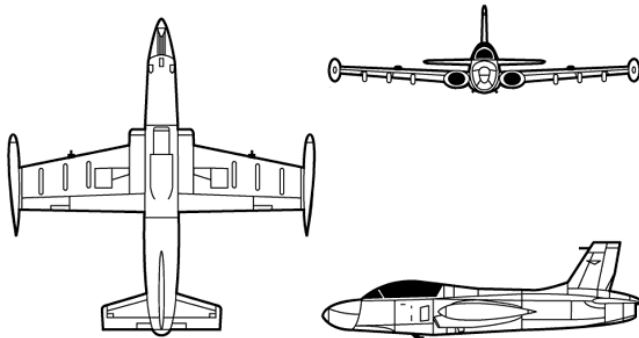
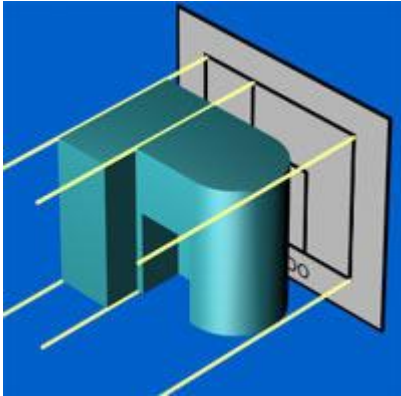
# El espacio 3D y la cámara



`gluLookAt(eyeX,eyeY,eyeZ,atX,atY,atZ,upX,upY,upZ)`



# Proyecciones

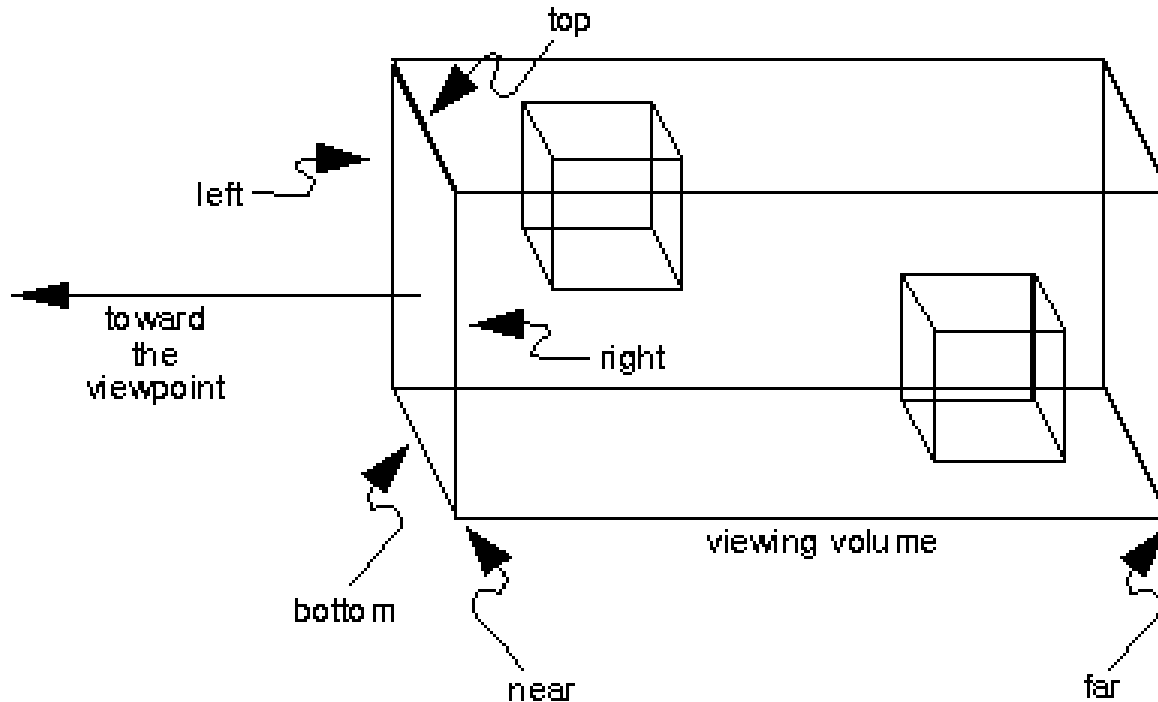


⦿ Proyección ortogonal

⦿ Proyección perspectiva

# Proyecciones

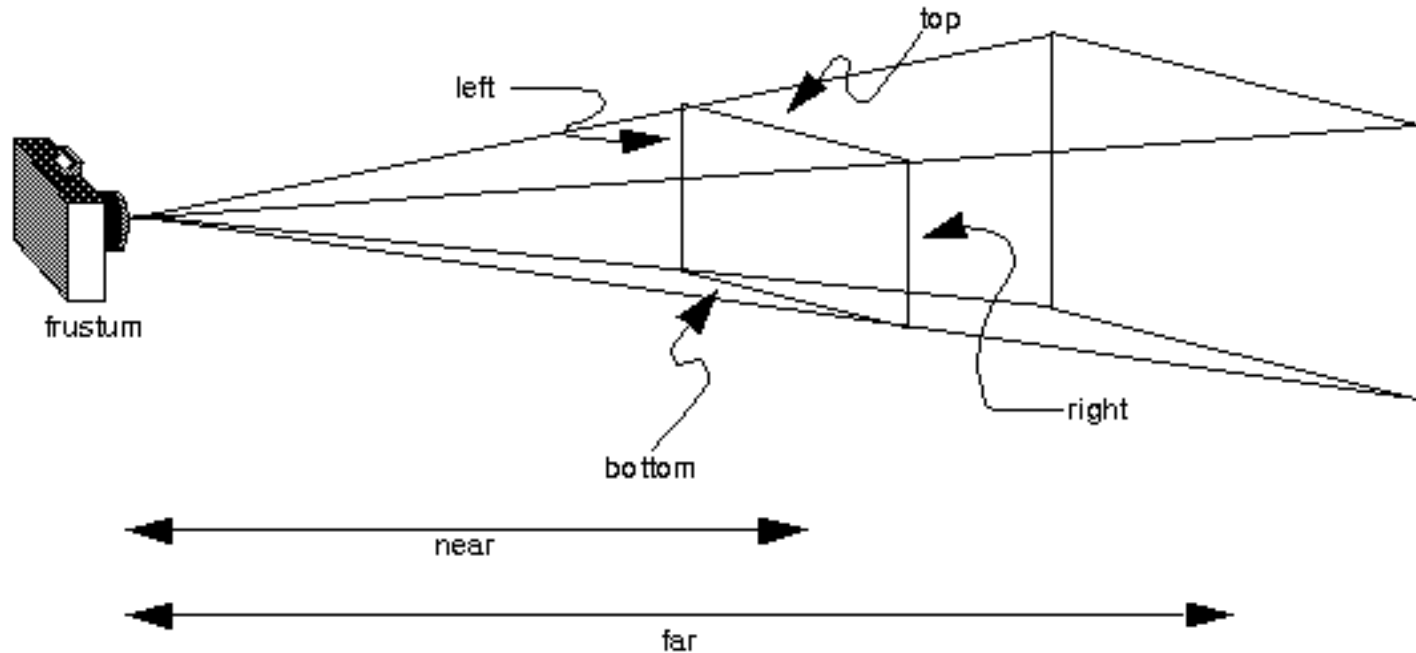
## Proyección ortogonal



```
void glOrtho(izquierda, derecha, abajo, arriba, cerca, lejos);
```

# Proyecciones

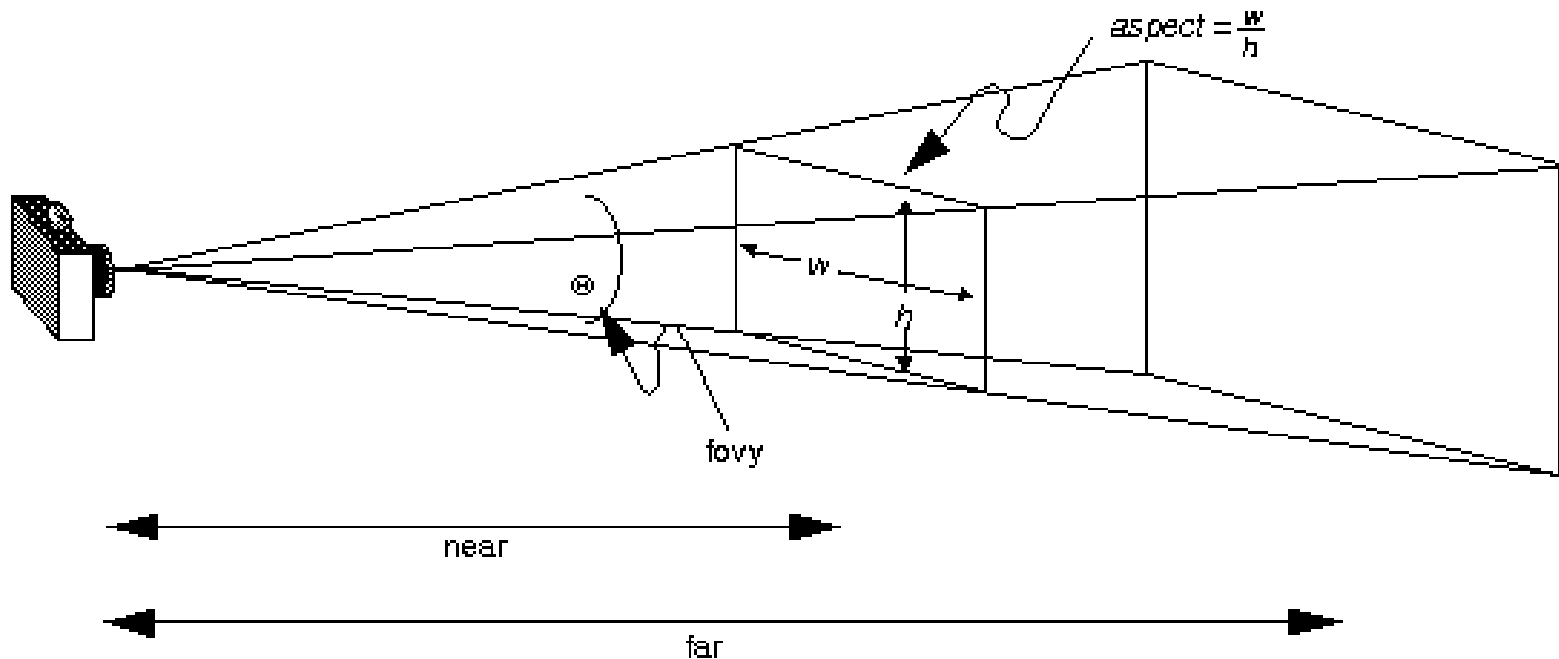
## Proyección perspectiva 1



```
void glFrustum(izquierda, derecha, abajo, arriba, cerca, lejos);
```

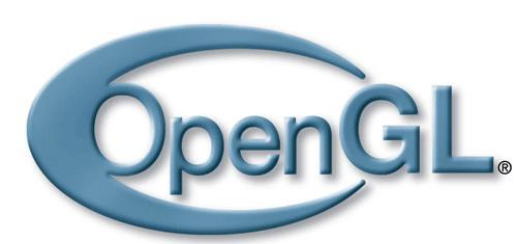
# Proyecciones

## Proyección perspectiva 2



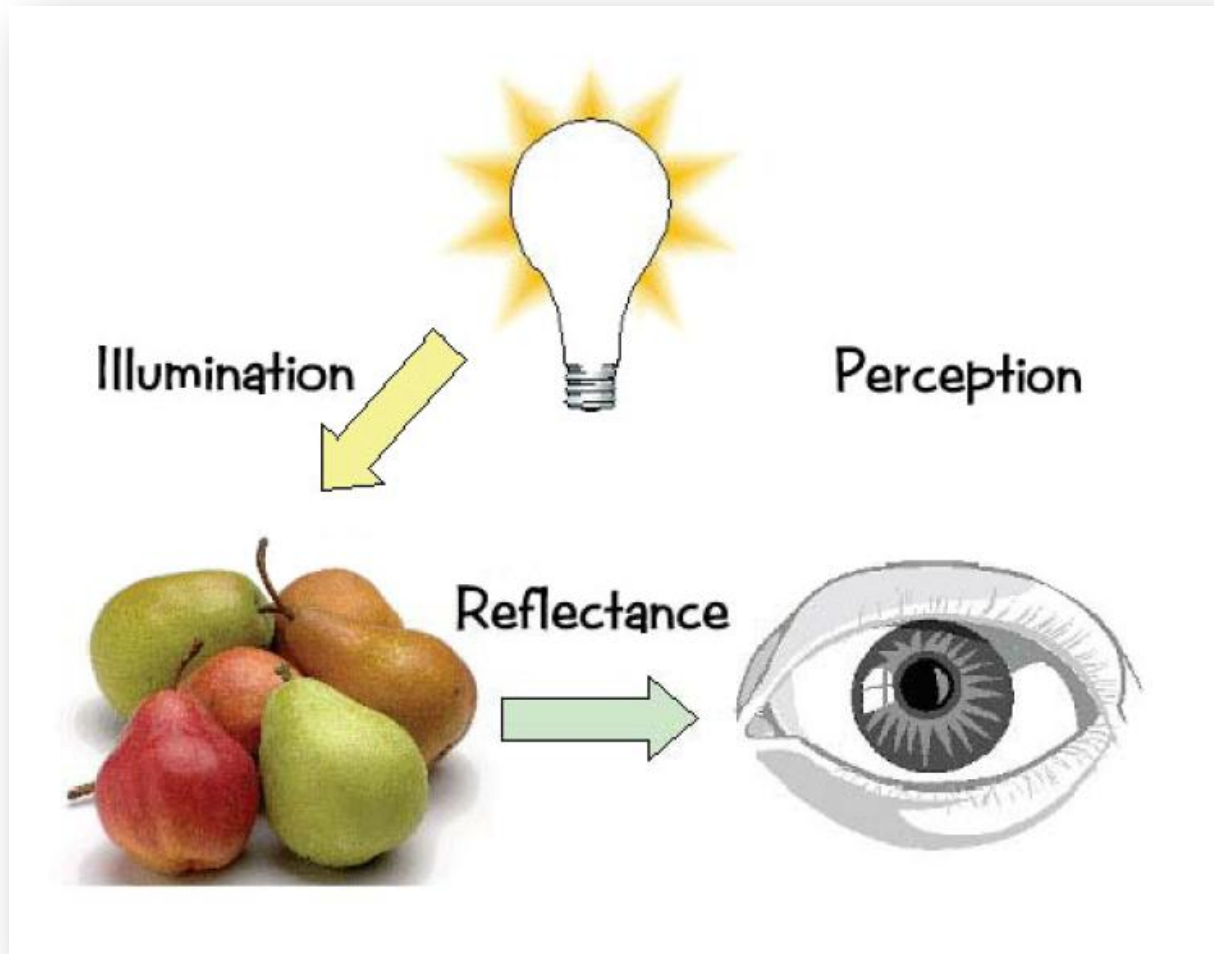
```
void gluPerspective(fovy, aspecto, cerca, lejos);
```



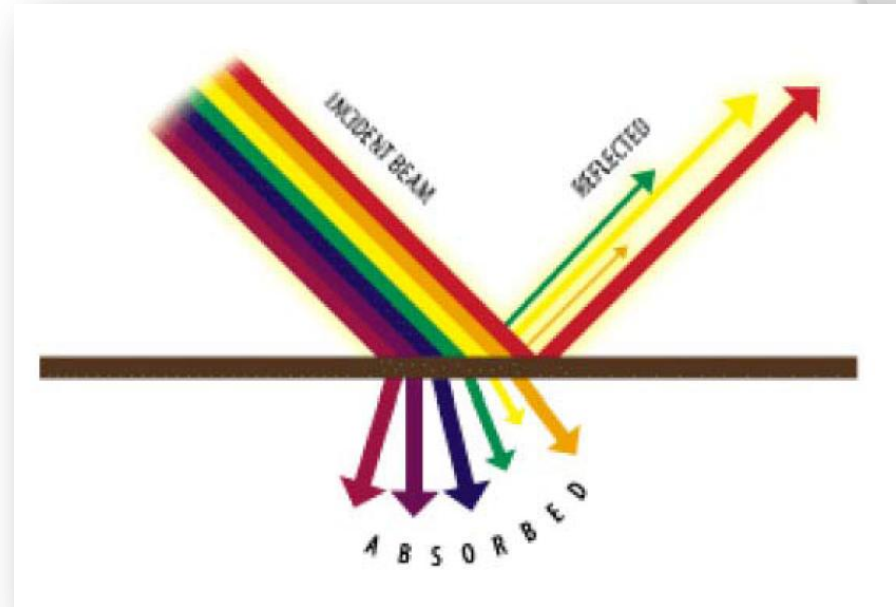


# ILUMINACIÓN

# Proceso de visión

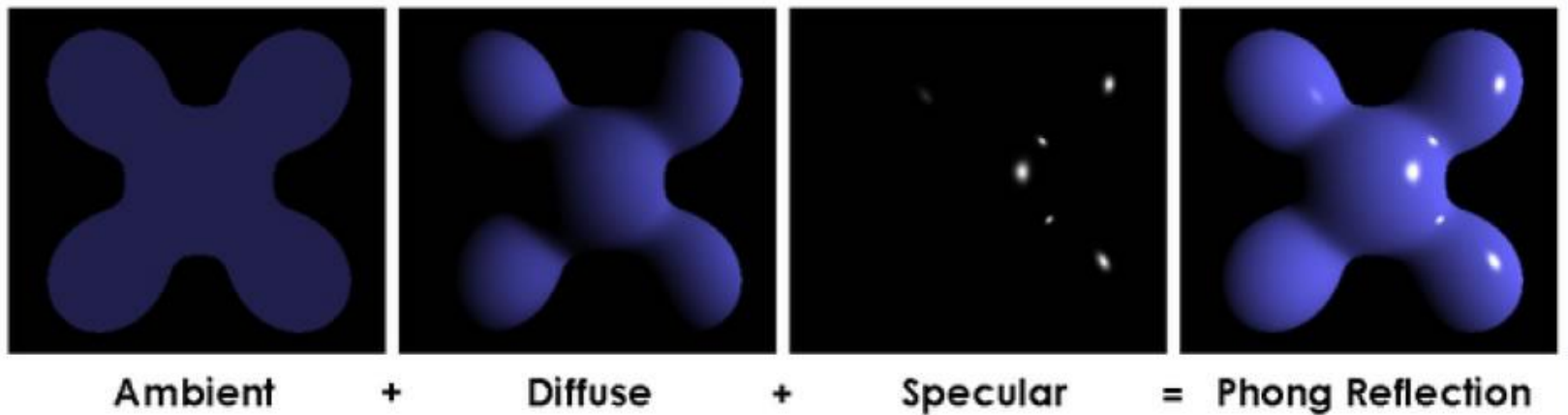


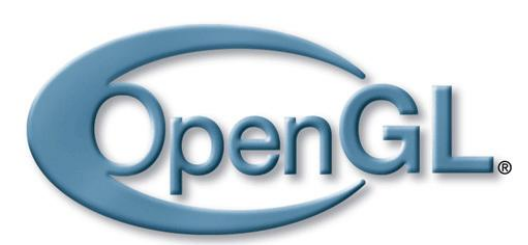
# Proceso de visión



- ⦿ Fuente de luz
- ⦿ Características del material
- ⦿ Ángulo de incidencia

# Modelo de iluminación





Iluminación

# Fuentes de Luz

# Especificando una fuente de luz en OpenGL

```
glEnable(GL_LIGHTING)
```

```
glLightfv(GL_LIGHT0, GL_POSITION, [ -2.0, 2.0, 1.0, 1.0])  
glLightfv(GL_LIGHT0, GL_AMBIENT , [ 0.2, 0.2, 0.2, 1.0])  
glLightfv(GL_LIGHT0, GL_SPECULAR, [ 1.0, 1.0, 1.0, 1.0])  
glLightfv(GL_LIGHT0, GL_DIFFUSE , [ 1.0, 1.0, 1.0, 1.0])
```

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.5)  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.5)  
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.2)
```

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0)  
glLightf(GL_LIGHT0, GL_SPOT_DIRECTION, [ -1.0, -1.0, 0.0])  
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0)
```

```
glEnable(GL_LIGHT0)
```



# Función clave

```
void glLight{if}(GLenum light, GLenum pname, TYPE param);  
void glLight{if}v(GLenum light, GLenum pname, const TYPE *param);
```

Creates the light specified by *light*, which can be GL\_LIGHT0, GL\_LIGHT1, ... , or GL\_LIGHT7. The characteristic of the light being set is defined by *pname*, which specifies a named parameter (see Table 5-1). *param* indicates the values to which the *pname* characteristic is set; it's a pointer to a group of values if the vector version is used or the value itself if the nonvector version is used. The nonvector version can be used to set only single-valued light characteristics.

# Descripción de parámetros

| Parameter Name           | Default Values                                     | Meaning   |
|--------------------------|--|---|
| GL_AMBIENT               | (0.0, 0.0, 0.0, 1.0)                               | ambient intensity of light  |
| GL_DIFFUSE               | (1.0, 1.0, 1.0, 1.0)<br>or<br>(0.0, 0.0, 0.0, 1.0) | diffuse intensity of light<br>(default for light 0 is white;<br>for other lights, black)  |
| GL_SPECULAR              | (1.0, 1.0, 1.0, 1.0)<br>or<br>(0.0, 0.0, 0.0, 1.0) | specular intensity of light<br>(default for light 0 is white;<br>for other lights, black) |
| GL_POSITION              | (0.0, 0.0, 1.0, 0.0)                               | ( $x$ , $y$ , $z$ , $w$ ) position of light   |
| GL_SPOT_DIRECTION        | (0.0, 0.0, -1.0)                                   | ( $x$ , $y$ , $z$ ) direction of<br>spotlight   |
| GL_SPOT_EXPONENT         | 0.0  | spotlight exponent  |
| GL_SPOT_CUTOFF           | 180.0  | spotlight cutoff angle  |
| GL_CONSTANT_ATTENUATION  | 1.0  | constant attenuation factor   |
| GL_LINEAR_ATTENUATION    | 0.0  | linear attenuation factor   |
| GL_QUADRATIC_ATTENUATION | 0.0  | quadratic attenuation<br>factor   |

Default Values for *pname* Parameter of `glLight*()`

# Atenuación

$$\text{attenuation factor} = \frac{1}{k_c + k_l d + k_q d^2}$$

where

$d$  = distance between the light's position and the vertex

$k_c$  = GL\_CONSTANT\_ATTENUATION

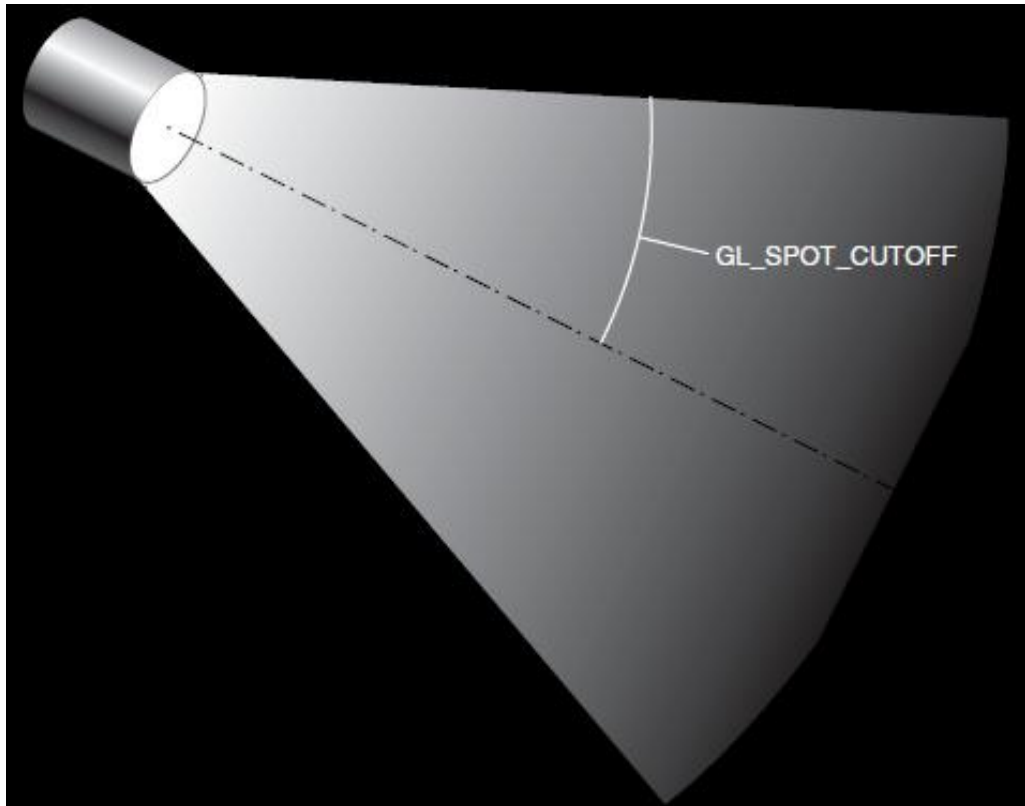
$k_l$  = GL\_LINEAR\_ATTENUATION

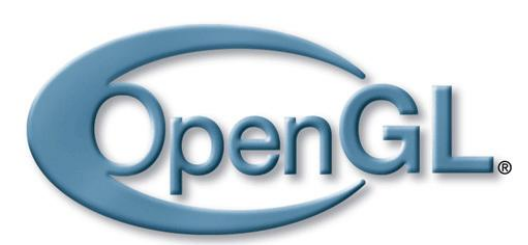
$k_q$  = GL\_QUADRATIC\_ATTENUATION

By default,  $k_c$  is 1.0 and both  $k_l$  and  $k_q$  are zero, but you can give these parameters different values:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);  
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

# Luz direccional

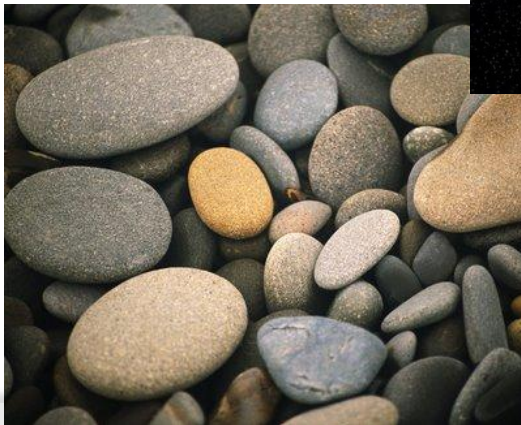




Iluminación

# Materiales

# Distintos materiales





# Especificando un material en OpenGL

```
glMaterialfv(GL_FRONT, GL_AMBIENT, [ 0.0, 0.0, 0.0, 1.0 ])
glMaterialfv(GL_FRONT, GL_DIFFUSE, [0.1, 0.5, 0.8, 1.0])
glMaterialfv(GL_FRONT, GL_SPECULAR, [ 0.0, 0.0, 0.0, 1.0 ])
glMaterialfv(GL_FRONT, GL_SHININESS, [25.0])
glMaterialfv(GL_FRONT, GL_EMISSION, [ 0.0, 0.0, 0.0, 1.0 ])
```

- El color del objeto puede ser especificado como componente ambiental o difusa.
- Un enfoque alternativo, consiste en activar colores a los materiales.

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE)
glEnable(GL_COLOR_MATERIAL)
```

```
glColor4fv([0.1, 0.5, 0.8, 1.0])
```

# Función clave

---

```
void glMaterial{if}(GLenum face, GLenum pname, TYPE param);  
void glMaterial{if}v(GLenum face, GLenum pname, const TYPE *param);
```

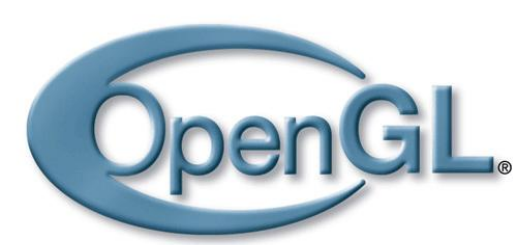
---

Specifies a current material property for use in lighting calculations. *face* can be GL\_FRONT, GL\_BACK, or GL\_FRONT\_AND\_BACK to indicate to which faces of the object the material should be applied. The particular material property being set is identified by *pname*, and the desired values for that property are given by *param*, which is either a pointer to a group of values (if the vector version is used) or the actual value (if the nonvector version is used). The nonvector version works only for setting GL\_SHININESS. The possible values for *pname* are shown in Table 5-3. Note that GL\_AMBIENT\_AND\_DIFFUSE allows you to set both the ambient and diffuse material colors simultaneously to the same RGBA value.

# Descripción de parámetros

| Parameter Name         | Default Value        | Meaning                                      |
|------------------------|----------------------|--|
| GL_AMBIENT             | (0.2, 0.2, 0.2, 1.0) | ambient color of material                    |
| GL_DIFFUSE             | (0.8, 0.8, 0.8, 1.0) | diffuse color of material                    |
| GL_AMBIENT_AND_DIFFUSE |                      | ambient and diffuse color of material        |
| GL_SPECULAR            | (0.0, 0.0, 0.0, 1.0) | specular color of material                   |
| GL_SHININESS           | 0.0                  | specular exponent                            |
| GL_EMISSION            | (0.0, 0.0, 0.0, 1.0) | emissive color of material                   |
| GL_COLOR_INDEXES       | (0, 1, 1)            | ambient, diffuse, and specular color indices |

Default Values for *pname* Parameter of `glMaterial*()`



Iluminación

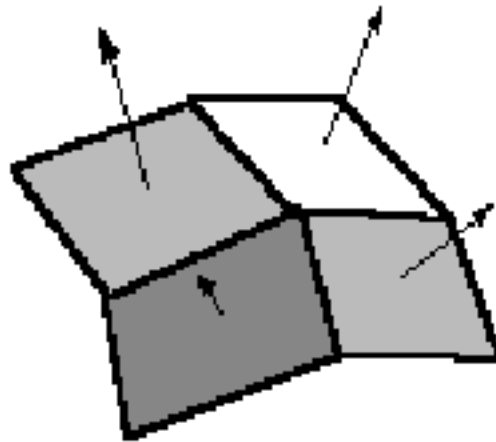
# Las Normales

# La importancia de las normales

- ⦿ La normal indica cual es la cara frontal de un triángulo. Permitiendo identificar por donde recibe la luz.
- ⦿ Distintos modelos:
  - Iluminación plana o FLAT
  - Iluminación suave o SMOOTH/GOURAUD
  - Iluminación PHONG

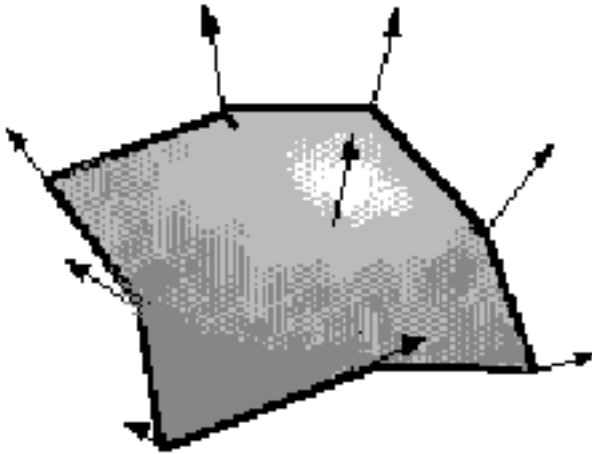
# FLAT

- ⦿ Cara cara (o triángulo) tiene una normal.
- ⦿ Luego, cada cara tiene un color plano.





# SMOOTH

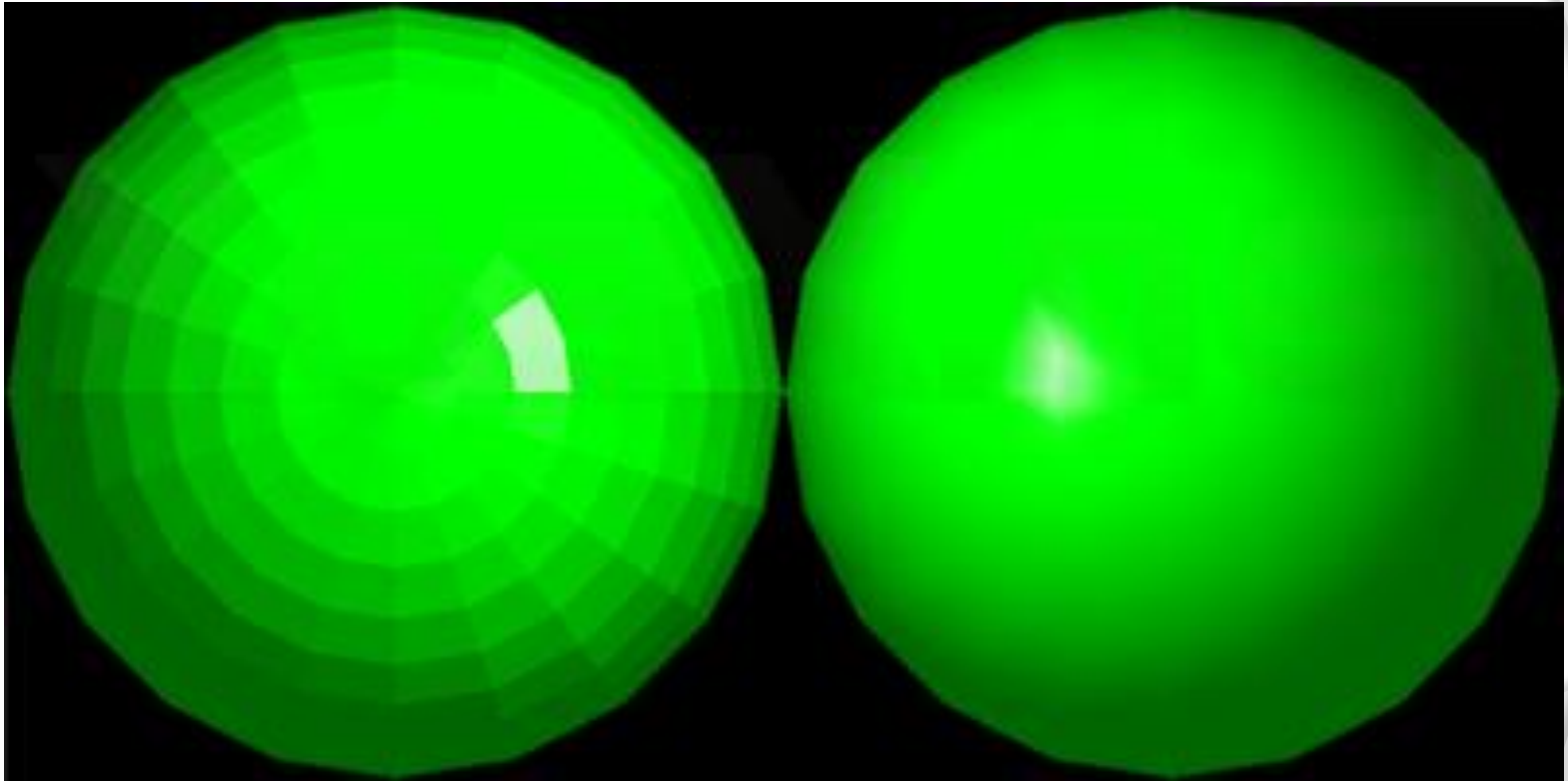


- Cada vértice tiene una normal.\*
- Acorde a cada normal, cada vértice se asocia a un color.
- La coloración de a cara se logra interpolando los colores asociados a cada vértice.

# PHONG

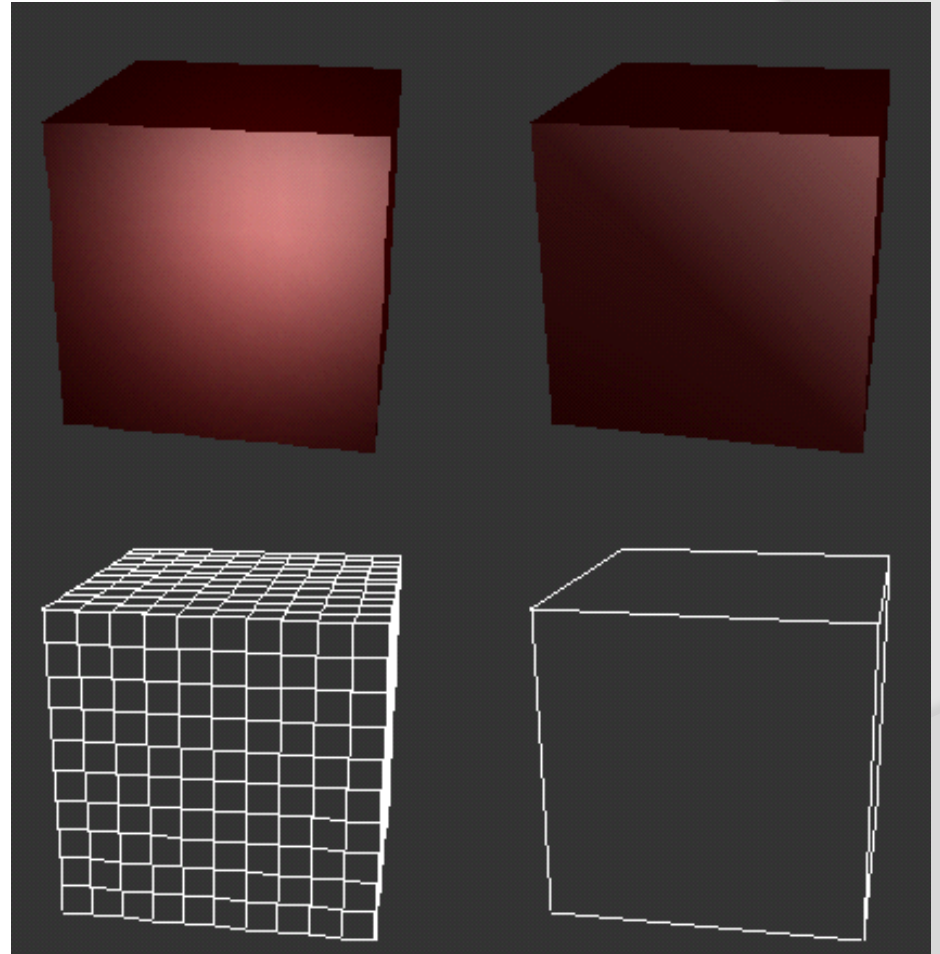
- ⦿ Cada vértice tiene una normal.
- ⦿ Se interpolan las normales para cada punto al interior de la cara.
- ⦿ El color de cada punto es determinado según su normal asociada.
- ⦿ OpenGL no lo implementa por ser muy costoso.

# Ejemplo

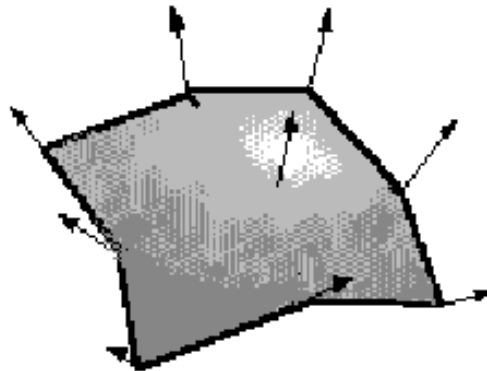
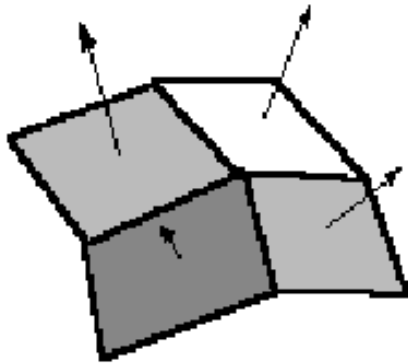


# Sobre SMOOTH ...

- Se logrará un buen efecto de iluminación en la medida que se especifique una buena cantidad de normales en la figura.
- Un cubo o un muro especificado solo por normales en los extremos no será correctamente iluminado.



# Calculando normales...



- 3 puntos definen un triángulo.
- Con esos tres puntos se pueden formar 2 vectores.
- El producto cruz entre esos vectores nos da un vector normal. Solo queda normalizarlo.
- Se ha calculado entonces la normal a una cara.
- La normal a un vértice se puede computar como el promedio de las normales asociadas a las caras que contengan dicho vértice.

# Observaciones

- ⦿ Figuras simples => fácil
  - Cilindro, Esfera, Cubo, Tetraedro, Etc...
  - Se aprovecha el sistema de coordenadas
- ⦿ Figuras complejas => difícil
  - Estructura de datos para almacenar los triángulos y vértices del modelo.
  - Pre computar normales de cada cara.
  - Encontrar todas las caras que contengan a cada vértice.
  - Calcular normal de vértice promediando las normales de cada cara.