

Introducción a OpenGL

Qué es OpenGL

- Librerías gráficas que facilitan el interfaz con el Hw gráfico
- Independiente del Hardware
- Primitivas geométricas básicas
- No incluye gestión de ventanas, teclado, etc.

Qué permite OpenGL

- Describir la geometría de los objetos a partir de las primitivas básicas
- Situar los objetos y el punto de vista
- Cálculo de color, incluyendo iluminación, texturas, etc.
- Generación de la imagen final, efectuando operaciones de culling, z-buffer, etc.

Introducción

- OpenGL como máquina de estados
- Algunas propiedades mantienen su valor hasta que se les asigne otro o se desactive.
- glEnable/glDisable
- Valores por defecto
- Funciones de acceso al valor actual

Introducción

- Color actual.
- Punto de vista.
- Transformaciones de proyección.
- Estilo de líneas y polígonos.

Introducción

- Modos de dibujar polígonos.
- Posición y características de las fuentes de iluminación.
- Propiedades de los materiales de los objetos

Introducción

- Librerías afines a OpenGL
 - OpenGL Utility Library (GLU)
 - OpenGL Auxiliary Library (AUX) /GLUT
 - OpenGL Extension to the X Windows System (GLX)

Introducción

- Librerías:
 - <http://www.opengl3d.org>
 - <http://mesa3d.sourceforge.net>
 - <http://www.sgi.com/software/opengl/glut.html>

Sintaxis de OpenGL

- Funciones con prefijo gl
 - Constantes con prefijo GL_
 - Algunas funciones con sufijo
 - Número de parámetros
 - Tipo de los parámetros
- glColor3f()
glVertex3i()

Tipos de datos

SUFIJOS	TIPO DE DATOS	DEFINICIÓN EN LENGUAJE C	DEFINICIÓN DEL TIPO EN OPENGL
b	Entero 8 Bits.	signed char	GLbyte
s	Entero 16 Bits	short	GLshort
i	Entero 32 Bits.	long	GLint, GLsizei
f	Real 32 Bits.	float	GLfloat, GLclampf
d	Real 64 Bits.	double	GLdouble, GLclampd
ub	Entero positivo 8 Bits.	unsigned char	GLubyte, GLboolean
us	Entero positivo 16 Bits	unsigned short	GLushort
ui	Entero positivo 32 Bits.	unsigned long	GLuint, GLenum, GLbitfield

Son equivalentes:

```
glVertex2i (1, 3);
```

```
glVertex2f (1.0, 3.0);
```

Son equivalentes:

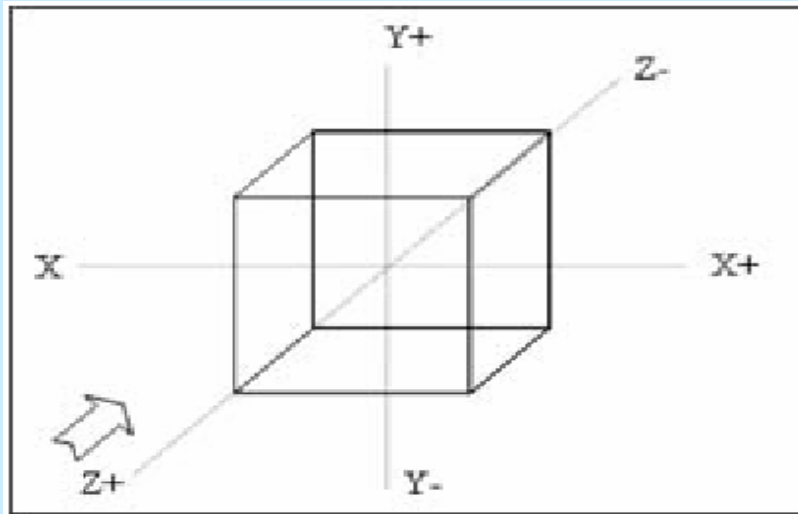
```
glColor3f (1.0, 1.0, 1.0);
```

```
float color_array[] = {1.0, 1.0, 1.0};
```

```
glColor3fv (color_array);
```

```
main ()
{
    AbrirVentana ();
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
    glBegin (GL_POLYGON);
        glVertex2f (-0.5, -0.5);
        glVertex2f (-0.5, 0.5);
        glVertex2f ( 0.5, 0.5);
        glVertex2f ( 0.5, -0.5);
    glEnd ();
    glFlush ();
}
```

Dibujar en 3D



- Limpiar ventana

`void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);`

`void glClear (GLbitfield mask);`

Posibles Buffers:

`GL_COLOR_BUFFER_BIT`
`GL_DEPTH_BUFFER_BIT`
`GL_ACCUM_BUFFER_BIT`
`GL_STENCIL_BUFFER_BIT`

- Valor por defecto: (0,0,0,0)
- Ejemplo:

```
glClearColor (0.5, 0.5, 0.5, 0.0);  
glClear (GL_COLOR_BUFFER_BIT);
```

- Especificar color

void **glColor3f**(GLclampf *red*, GLclampf *green*, GLclampf *blue*);

Función glColor3f ()	Color
glColor3f (0.0, 0.0, 0.0);	negro
glColor3f (1.0, 0.0, 0.0);	rojo
glColor3f (0.0, 1.0, 0.0);	verde
glColor3f (1.0, 1.0, 0.0);	amarillo
glColor3f (0.0, 0.0, 1.0);	azul
glColor3f (1.0, 0.0, 1.0);	magenta
glColor3f (0.0, 1.0, 1.0);	cian
glColor3f (1.0, 1.0, 1.0);	blanco

- **Forzar a la Finalización del Dibujo.**

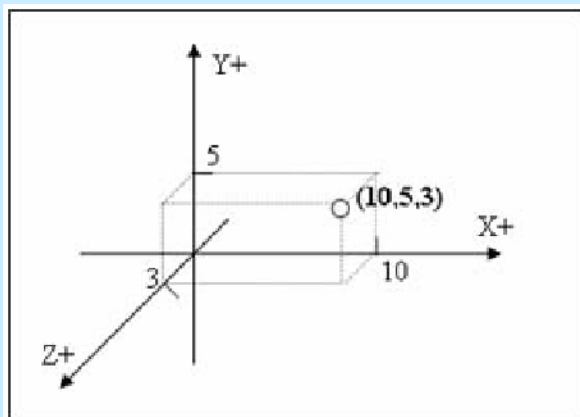
`void glFlush (void);`

Fuerza a los comandos de OpenGL a finalizar su ejecución en un tiempo finito a partir de este momento

Vértices

- `void glVertex{234}{sifd}{v}` (TYPE *coords*);

Ej.: `glVertex3f(10.0f, 5.0f, 3.0f)`



Primitivas

- Interpretación de un conjunto de vértices, dibujados de una manera específica en pantalla.

```
glBegin(<tipo de primitiva>);  
    glVertex(...);  
    glVertex(...);  
    ...  
    glVertex(...);  
glEnd();
```

Puntos (GL_POINTS)

- Los vértices se interpretan como puntos

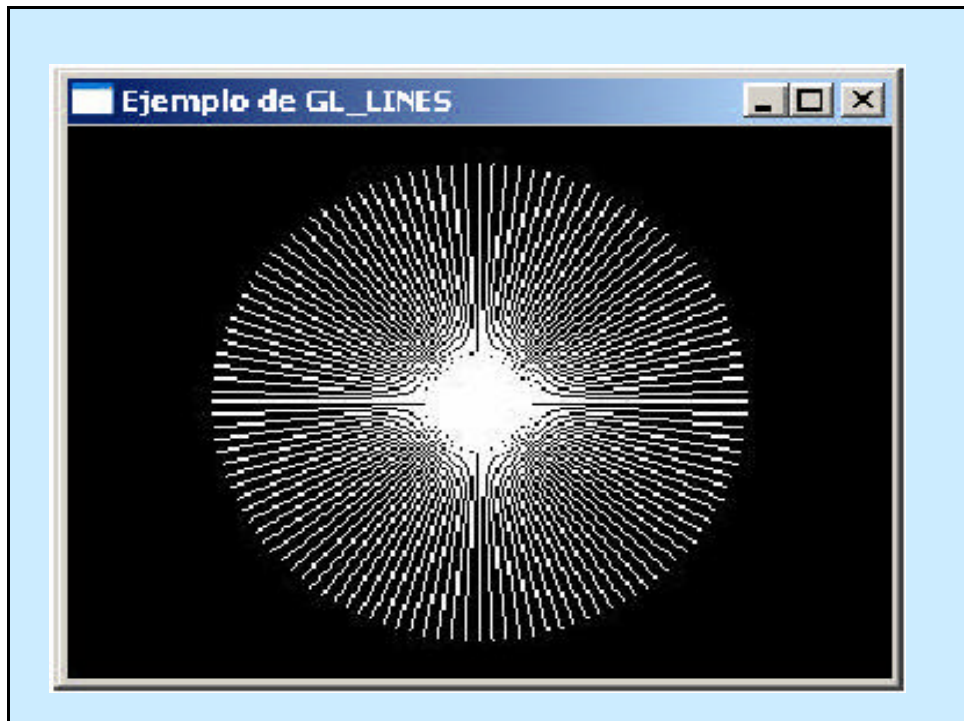
```
glBegin(GL_POINTS);  
    glVertex3f(0.0f, 0.0f, 0.0f);  
    glVertex3f(10.0f, 10.0f, 10.0f);  
glEnd();
```

Puntos (GL_POINTS)

- Tamaño de punto
void **glPointSize** (GLfloat *size*);
Por defecto size=1
- Rango de valores posibles
glGetFloatv (GL_POINT_SIZE_RANGE);

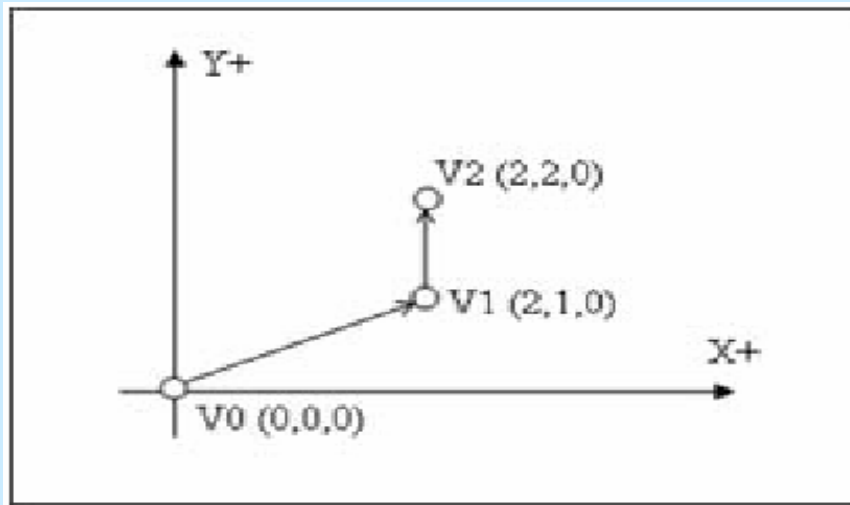
Líneas (GL_LINES)

```
GLfloat angulo;  
int i;  
glBegin(GL_LINES);  
for (i=0; i<360; i+=3)  
{  
    angulo = (GLfloat)i*3.14159f/180.0f; // grados a radianes  
    glVertex3f(0.0f, 0.0f, 0.0f);  
    glVertex3f(cos(angulo), sin(angulo), 0.0f);  
}  
glEnd();
```



Tiras de líneas

```
glBegin(GL_LINE_STRIP);  
    glVertex3f(0.0f, 0.0f, 0.0f); // V0  
    glVertex3f(2.0f, 1.0f, 0.0f); // V1  
    glVertex3f(2.0f, 2.0f, 0.0f); // V2  
glEnd();
```

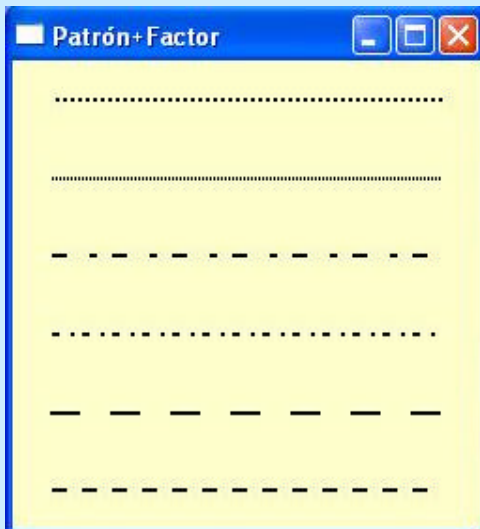
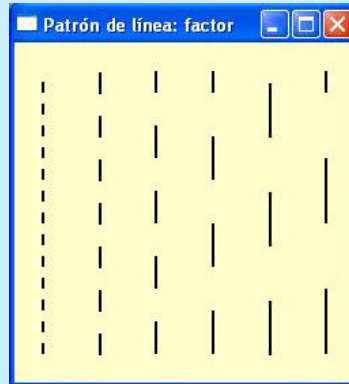


Líneas

- Grosor de la línea
void **glLineWidth** (GLfloat *width*);
- Patrón de línea
void **glLineStipple** (Glint *factor*, Gushort *pattern*);
Es necesario activarlo/desactivarlo
glEnable (GL_LINE_STIPPLE)
glDisable (GL_LINE_STIPPLE)

Líneas

```
//color de fondo
glClearColor(1.0, 1.0, 0.8, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
//color de dibujo
glColor3f (0.0, 0.0, 0.0);
//grosor de línea
glLineWidth(2.0);
//se activa el patrón de línea
glEnable(GL_LINE_STIPPLE);
for(i=1;i<7;i++){
    glTranslatef(1.0,0.0,0.0);
    glLineStipple(i,0x00FF);
    glBegin (GL_LINES);
        glVertex3f (0.0, -2.0, 0.0);
        glVertex3f (0.0, 3.0, 0.0);
    glEnd ();
}
glFlush();
```



```
glLineStipple(2, 0xAAAA);
```

```
glLineStipple(1, 0xAAAA);
```

```
glLineStipple(2, 0x0C0F);
```

```
glLineStipple(1, 0x0C0F);
```

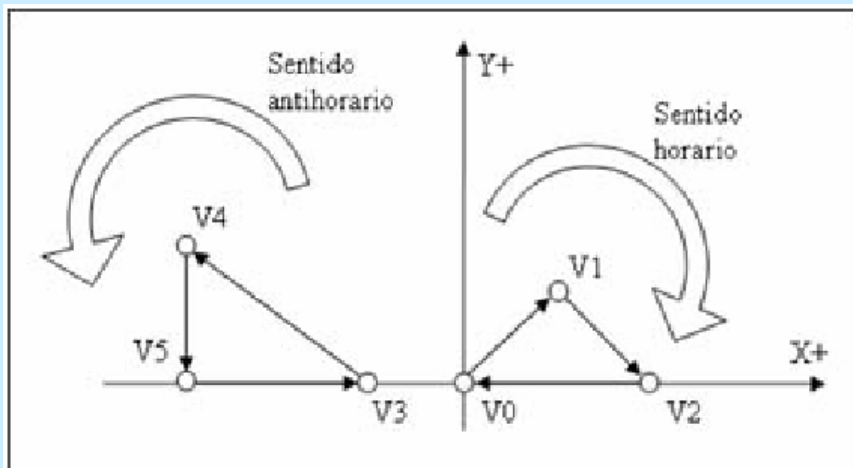
```
glLineStipple(2, 0x00FF);
```

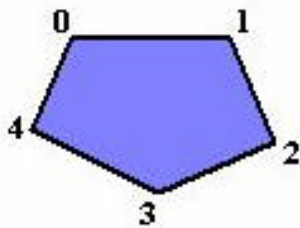
```
glLineStipple(1, 0x00FF);
```

Polígonos (Triángulos)

```
glBegin(GL_TRIANGLES);  
    glVertex3f(0.0f, 0.0f, 0.0f); // V0  
    glVertex3f(1.0f, 1.0f, 0.0f); // V1  
    glVertex3f(2.0f, 0.0f, 0.0f); // V2  
  
    glVertex3f(-1.0f, 0.0f, 0.0f); // V3  
    glVertex3f(-3.0f, 2.0f, 0.0f); // V4  
    glVertex3f(-2.0f, 0.0f, 0.0f); // V5  
glEnd();
```

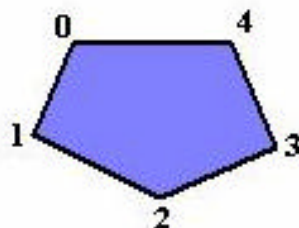
Orientación





GL_POLYGON

Sentido horario

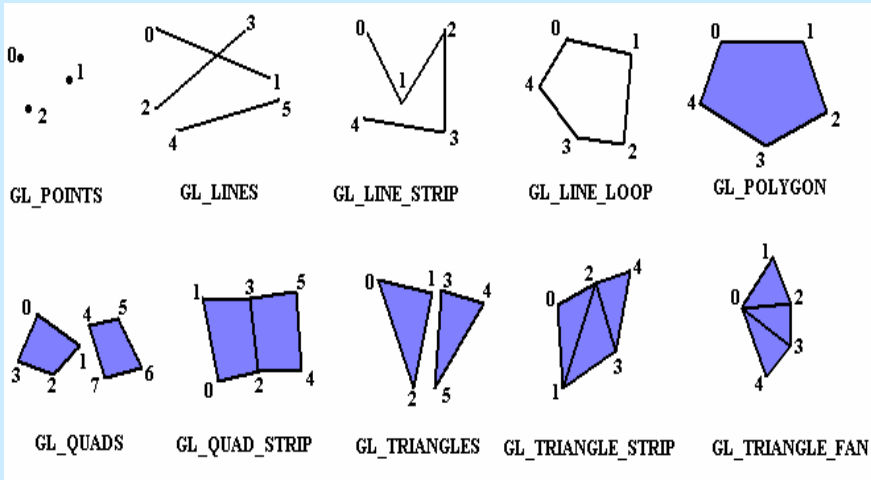


GL_POLYGON

Sentido antihorario

Primitivas

Parámetro <i>mode</i>	Significado
GL_POINTS	Puntos individuales.
GL_LINES	Cada Par de vértices forman un segmento individual.
GL_POLYGON	Vértices de polígono simple y conexo.
GL_TRIANGLES	Cada tema determina un triángulo.
GL_QUADS	Cada cuádrupla determina un cuadrado.
GL_LINE_STRIP	Series de segmentos conectados.
GL_LINE_LOOP	Series de segmentos conectados y cerrados.
GL_TRIANGLE_STRIP	Cadenas de triángulos conectados.
GL_TRIANGLE_FAN	Cadenas de polystrip (el centro es el primer vértice).
GL_QUAD_STRIP	Cuadriláteros encadenados.



Polígonos

- void **glPolygonMode** (GLenum *face*, GLenum *mode*);
Define el modo en que se dibujan los polígonos

Parámetro	Flag	Función
Face	GL_FRONT_AND_BACK	Rellena las dos caras de la misma forma.
	GL_FRONT	Sólo se ve la cara de frente al observador.
	GL_BACK	Sólo se ve la cara posterior.
Mode	GL_POINT	Sólo se ven los vértices del polígono.
	GL_LINE	Estructura de alambre.
	GL_FILL	Polígono relleno.

Polígonos

- void **glFrontFace** (GLenum *mode*);

Define qué caras se consideran anteriores.

GL_CCW (CounterClockWise). **Por defecto**

GL_CW (ClockWise)

void **glCullFace** (GLenum *mode*);

- void **glCullFace** (GLenum *mode*);

Indica qué caras pueden descartarse antes de proyectar sus coordenadas en pantalla.

Polígonos

Parámetro	Flag	Función
mode	GL_FRONT	Eliminar los polígonos anteriores.
	GL_BACK	Eliminar los polígonos posteriores.
	GL_FRONT_AND_BACK	Eliminar todos los polígonos.

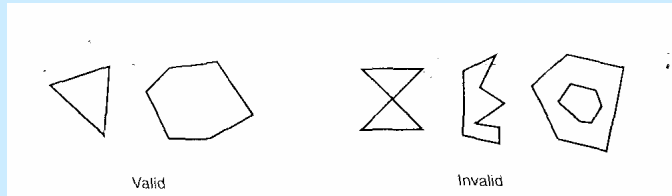
- Es necesario activarla previamente

glEnable(GL_CULL_FACE)

glDisable(GL_CULL_FACE)

Polígonos

- Restricciones de los polígonos
 - Sus segmentos no se intersecan
 - Convexos
 - Sin agujeros



Polígonos

- Los polígonos cóncavos se pueden subdividir en polígonos convexos

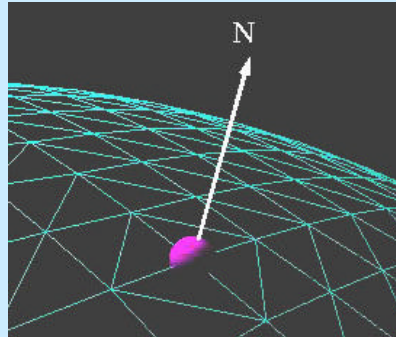
void **glEdgeFlag** (GLboolean *flag*);

Indica cuándo se debe pintar una arista.

Por defecto está a GL_TRUE.

Vectores normales

- Se asignan a los vértices
- Indican la orientación de la superficie



Vectores normales

```
void glNormal3f(TYPE nx, TYPE ny, TYPE nz);
```

```
void glNormal3fv(const TYPE *v);
```

Activan y asignan el vector pasado como argumento como vector normal a el/los vértices que se definan a continuación.

Color de relleno

- Llamada a glColor antes de la definición del polígono

```
glBegin(GL_TRIANGLES);  
    glColor3f(1.0f, 0.0f, 0.0f);  
    glVertex3f(0.0f, 0.0f, 0.0f);  
    glVertex3f(2.0f, 0.0f, 0.0f);  
    glVertex3f(1.0f, 1.0f, 0.0f);  
  
    glColor3f(0.0f, 1.0f, 0.0f);  
    glVertex3f(-1.0f, 0.0f, 0.0f);  
    glVertex3f(-3.0f, 2.0f, 0.0f);  
    glVertex3f(-2.0f, 0.0f, 0.0f);  
glEnd();
```

Modelo de sombreado

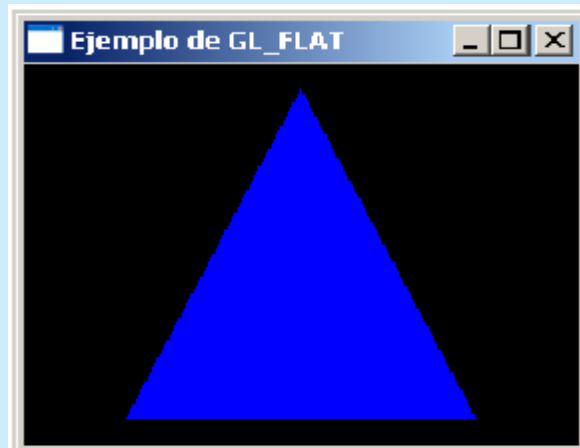
- void glShadeModel(GLenum *mode*)

Si mode=GL_FLAT color de relleno es el color activo en el momento en que se definió el último parámetro

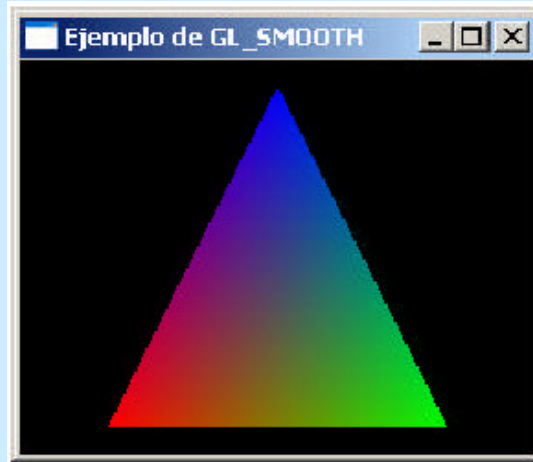
Si mode=GL_SMOOTH se rellena interpolando el color de los vértices

```
glShadeModel(GL_FLAT);  
glBegin(GL_TRIANGLE);  
    glColor3f(1.0f, 0.0f, 0.0f); // activamos el color rojo  
    glVertex3f(-1.0f, 0.0f, 0.0f);  
    glColor3f(0.0f, 1.0f, 0.0f); // activamos el color verde  
    glVertex3f(1.0f, 0.0f, 0.0f);  
    glColor3f(1.0f, 0.0f, 0.0f); // activamos el color azul  
    glVertex3f(0.0f, 0.0f, 1.0f);  
glEnd();
```

Sombreado plano



Sombreado suave



Eliminación de caras ocultas

- Eliminar zonas tapadas por otras

```
glColor3f(1.0f, 1.0f, 1.0f); // activamos el color blanco
glBegin(GL_TRIANGLES);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glVertex3f(0.0f, 1.0f, -1.0f);
glEnd();

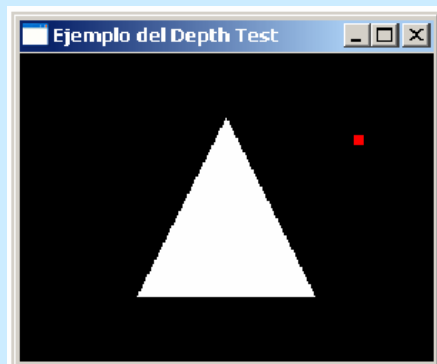
glPointSize(6.0f); // tamaño del pixel = 6, para que se vea bien
glColor3f(1.0f, 0.0f, 0.0f); // activamos el color rojo
glBegin(GL_POINTS);
    glVertex3f(0.0f, 0.0f, -2.0f);
    glVertex3f(2.0f, 1.0f, -2.0f);
glEnd();
```

Eliminación de caras ocultas



Eliminación de caras ocultas

- Aplicando test del z-buffer:
`glEnable(GL_DEPTH_BUFFER)`



Eliminación de caras ocultas

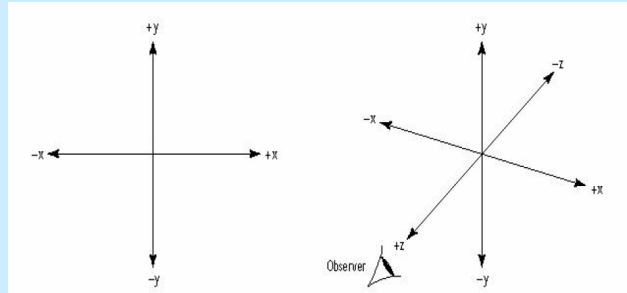
- Es necesario limpiar el buffer de profundidad antes de dibujar un frame:
`glClear(GL_DEPTH_BUFFER_BIT);`
- Se puede combinar con la limpieza del buffer de color:
`glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`

Vista tridimensional

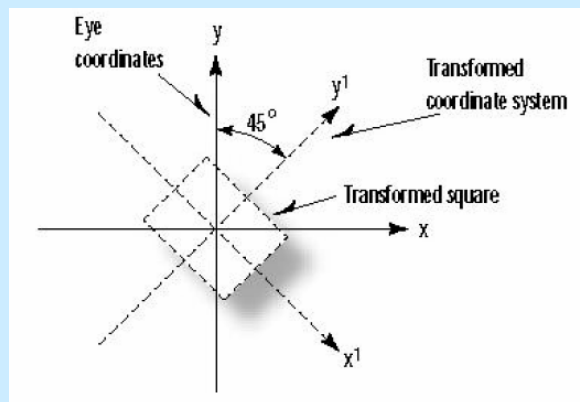
- Transformaciones de vista.
 - Sitúan al observador
- Transformaciones de modelo
 - Sitúan los objetos
- Transformaciones de proyección
 - Definen el volumen de vista
- Transformaciones de ventana (viewport)
 - Definen el tamaño de la ventana final
- Manejar la pila de matrices

Transformaciones de vista

- Permiten situar el punto de vista.
- Inicialmente situado en el $(0,0,0)$ mirando hacia el semieje Z negativo

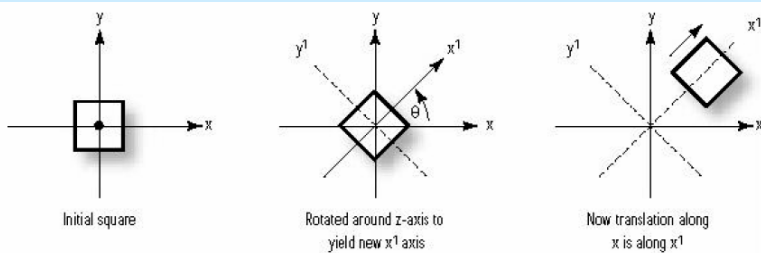
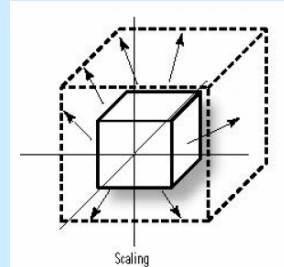
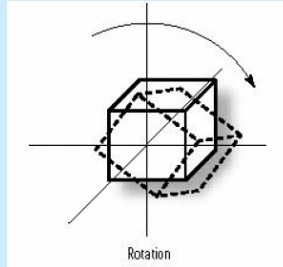
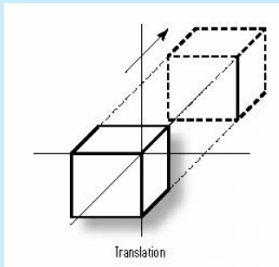


- Las transformaciones de vista deben realizarse antes de cualquier otra transformación
- Las transformaciones se realizan respecto al sistema de coordenadas oculares

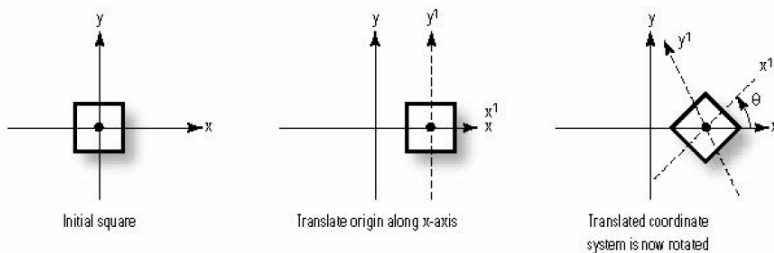


Transformaciones de modelo

- Permite situar, rotar y escalar los objetos de la escena

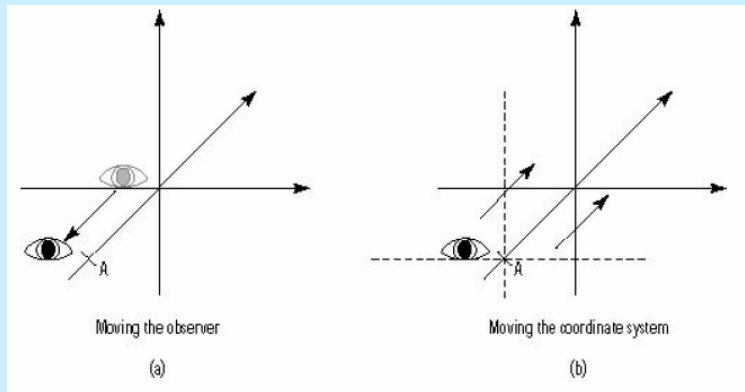


(a)



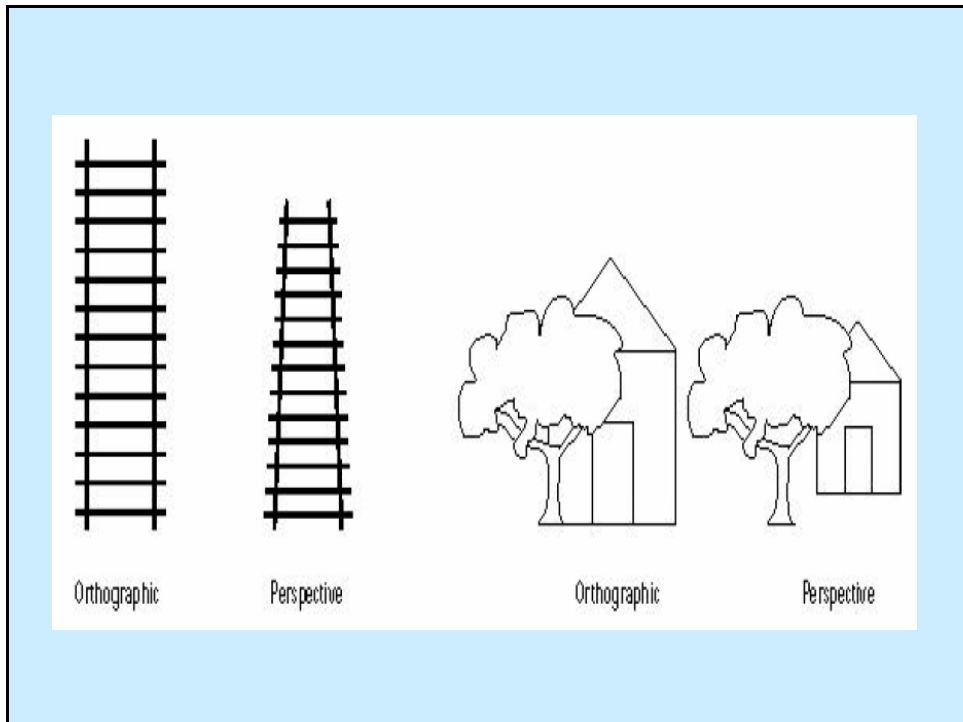
Dualidad modelo-vista

- En realidad las transformaciones de vista y de modelo son la misma

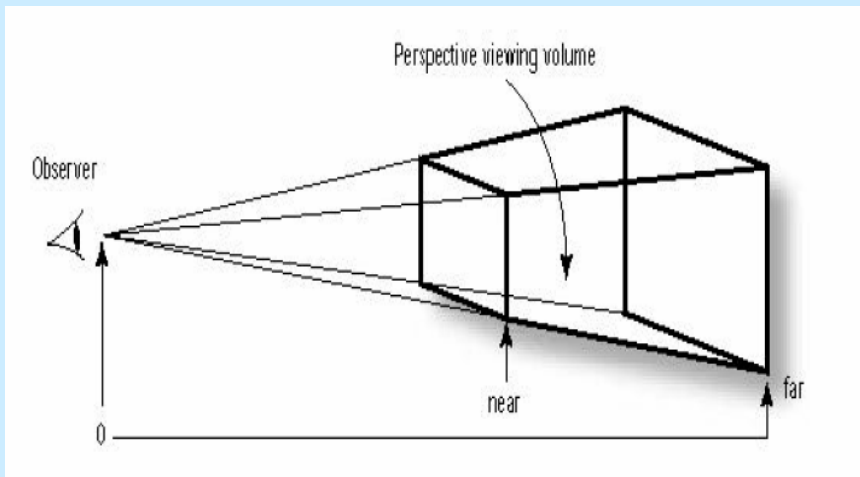


Transformaciones de proyección

- Define el volumen de visualización y los planos de corte
- Especifica como se traslada la escena final a la imagen final en pantalla.
- Dos tipos de proyección:
 - Perspectiva
 - Ortográfica



Volumen de vista en proyección perspectiva



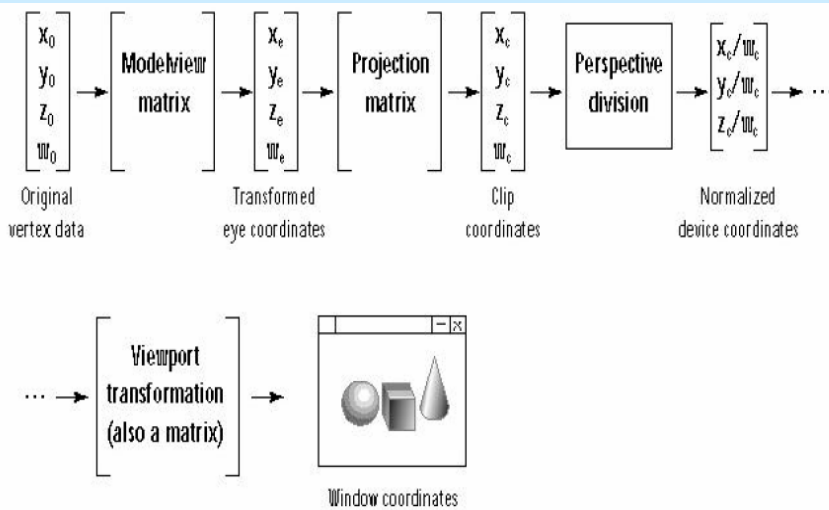
Transformaciones de ventana (viewport)

- Definen cómo trasladar la imagen 2D proyectada a coordenadas de pantalla
- Es la última transformación

Vista tridimensional

- Transformaciones basadas en multiplicaciones de matrices
- Tipos de matrices:
 - GL_MODELVIEW: Matriz modelo-vista
 - GL_PROJECTION: Matriz proyección
 - GL_TEXTURE: Matriz textura

Canal de transformaciones

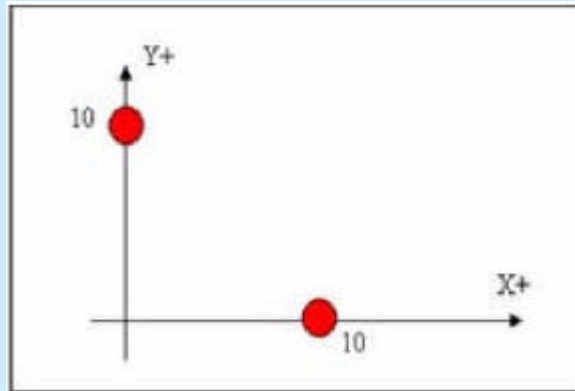


Transformaciones de modelo

- Afectan a la matriz de modelo-vista
- `glMatrixMode(GL_MODELVIEW);`
 - `void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z)`
 - `void glTranslate{fd} (TYPE x, TYPE y, TYPE z)`
 - `void glScale{fd} (TYPE x, TYPE y, TYPE z)`

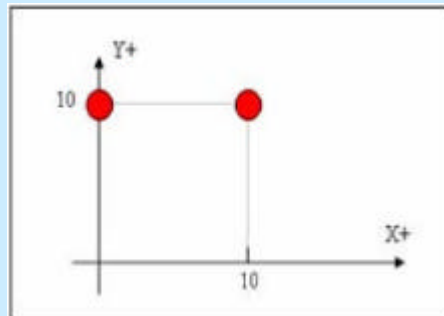
Transformaciones de modelo

- Efecto acumulativo



Transformaciones de modelo

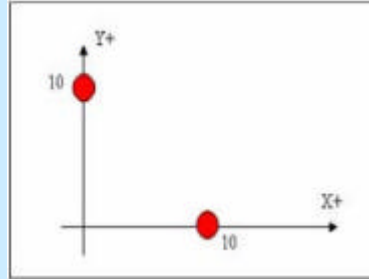
```
glTranslatef(0.0f, 10.0f, 0.0f);  
glutSolidSphere(3.0f);  
glTranslatef(10.0f, 0.0f, 0.0f);  
glutSolidSphere(3.0f);
```



Código incorrecto

Transformaciones de modelo

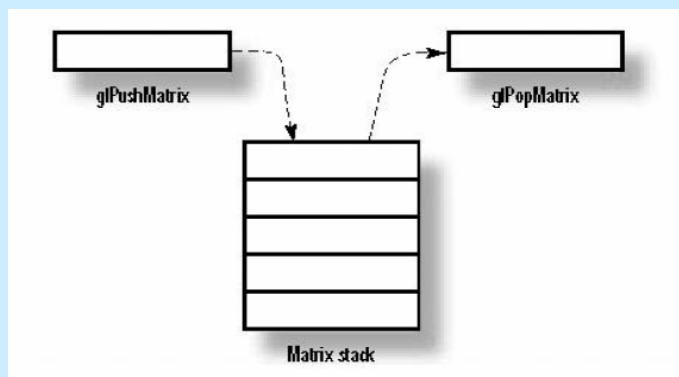
```
glTranslatef(0.0f, 10.0f, 0.0f);  
glutSolidSphere(3.0f);  
glLoadIdentity();  
glTranslate(10.0f, 0.0f, 0.0f);  
glutSolidSphere(3.0f);
```



Es necesario reiniciar la matriz `GL_MODELVIEW`

Pilas de matrices

- Para modelo-vista y proyección



Transformaciones de vista

- Utilizando `glTranslate()` y `glRotate()`

`glTranslate (0.0, 0.0, -5.0)` aplicada al inicio es equivalente a situar el punto de vista en el punto (0.0, 0.0, 5.0)

- Utilizando `gluLookAt()`

Transformaciones de vista

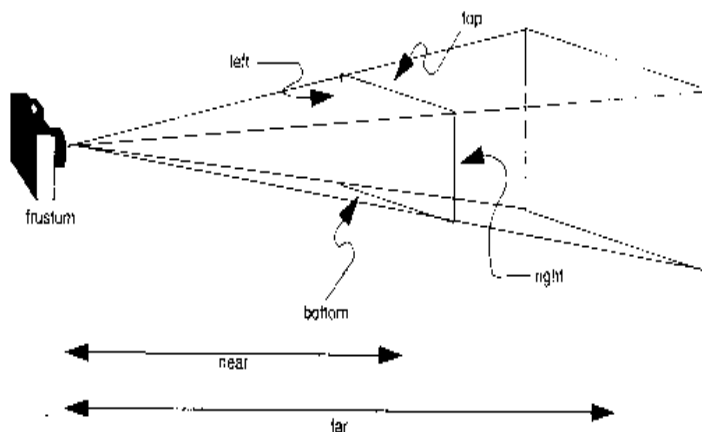
`void gluLookAt`

(`GLdouble eyex`, `GLdouble eyey`, `GLdouble eyez`,
`GLdouble centrox`, `GLdouble centroy`, `GLdouble centroz`,
`GLdouble upx`, `GLdouble upy`, `GLdouble upz`)

Transformaciones de proyección

- Define el tipo de proyección:
 - Perspectiva o paralela
- Define qué objetos serán visibles
- Afecta a la matriz `GL_PROJECTION`
`glMatrixMode (GL_PROJECTION);`

Proyección Perspectiva



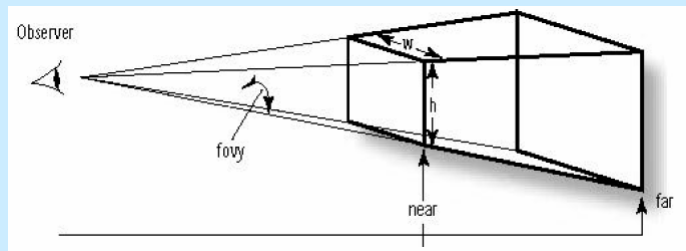
Proyección perspectiva

- void **glFrustum** (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

Define la sección piramidal

Proyección en perspectiva

- void **gluPerspective** (GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)



Transformaciones de ventana

- **void glViewport (*GLint* x, *GLint* y, *GLsizei* width, *GLsizei* height)**

Define un rectángulo de píxeles dentro de una ventana en el cual se dibuja la imagen final.

x, y definen la esquina inferior izquierda de la ventana.

Definir las transformaciones

- **Operaciones afectan a la matriz activa**
 - void glLoadIdentity (void)
 - void glLoadMatrix{fd} (const TYPE *m)
 - void glMultMatrix{fd} (const TYPE *m)

Iluminación

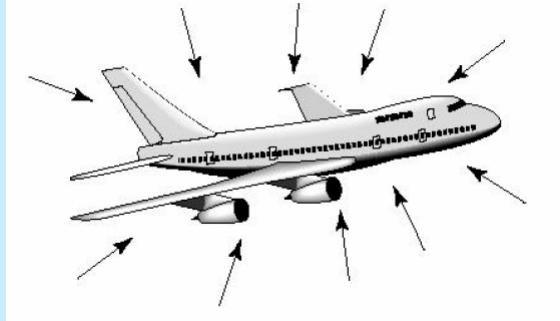
- El color final de un pixel depende de
 - Sin iluminación:
 - La propiedad de color
 - Con iluminación:
 - Las luces de la escena
 - El material del objeto

Modelo de iluminación

- Luz dividida en 4 componentes independientes
 - Emitida
 - Ambiente
 - Difusa
 - Especular
- El color de cada componente se define por sus valores RGB

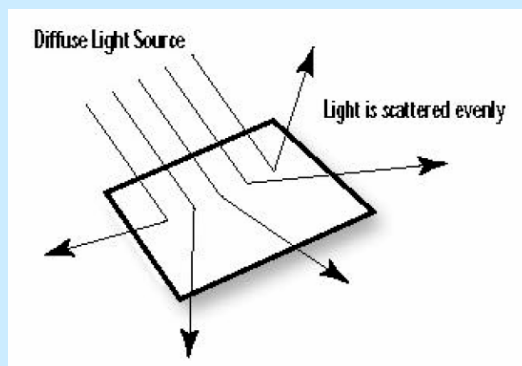
Luz ambiente

- No se puede determinar la dirección de la que proviene



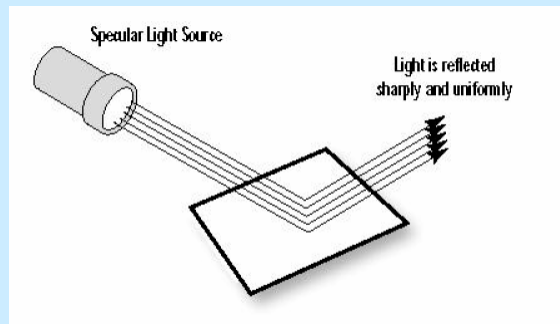
Luz difusa

- Proviene de una dirección pero se refleja por igual en todas las direcciones



Luz especular

- Proviene de una dirección y tiende a rebotar en una determinada dirección



Materiales

- Al igual que las luces, se le asignan componentes
 - Ambiente
 - Difuso
 - Especular
- Definen la cantidad de cada componente de luz incidente que reflejan

Creación de fuentes de luz

- **void glLight{if}[v] (GLenum *light*, GLenum *pname*, TYPE *param*)**
- Light puede tomar valores GL_LIGHT0...GL_LIGHT7
- Pname permite configurar diversas propiedades

Creación de fuentes de luz

Nombre del parámetro	Valor por defecto	Significado
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	RGBA-intensidad de luz ambiente
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	RGBA-intensidad de luz difusa
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	RGBA-intensidad de luz especular
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) posición de la luz
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x, y, z, w) dirección de la luz
GL_SPOT_EXPONENT	0.0	exponente "spotlight"
GL_SPOT_CUTOFF	180.0	ángulo "spotlight"
GL_CONSTANT_ATTENUATION	1.0	factor de atenuación constante
GL_LINEAR_ATTENUATION	0.0	factor de atenuación lineal
GL_QUADRATIC_ATTENUATION	0.0	factor de atenuación cuadrático

Creación fuentes de luz

Ejemplo:

```
GLfloat light_ambient[]={0.0, 0.0, 0.0, 1.0};
GLfloat light_diffuse[]={1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[]={1.0, 1.0, 1.0, 1.0};
GLfloat light_position[]={1.0, 1.0, 1.0, 0.0}; //direccional
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Luz dirigida (Spotlights)

- Posición de la luz

```
GLfloat light_position[]={0.0, 0.0, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- Extensión del cono de luz

```
glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

- Dirección del eje del cono de luz

```
GLfloat spot_direction[] = {-1.0, -1.0, 0.0}
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
```

Creación de fuentes de luz

- Después de definir la características de una fuente de luz hay que encenderla
 - `glEnable(GL_LIGHT0);`
- Y hay que habilitar los cálculos de iluminación
 - `glEnable(GL_LIGHTING);`

Seleccionar modelo de iluminación

Definir tres características:

- Intensidad de la luz global ambiente
- Posición del punto de vista: local o infinito
- Cálculos de iluminación a dos caras

Modelo de iluminación

- Además de las fuentes de luz, se puede especificar una luz global ambiente

```
GLfloat lmodel_ambient[ ]={0.2, 0.2, 0.2, 1.0};  
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  
lmodel_ambient);
```

Modelo de iluminación

- Posición del punto de vista
 - Local
 - Infinito (por defecto)

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VI  
EWER, GL_TRUE);
```

Modelo de iluminación

- Cálculo de iluminación en 2 caras.
Por defecto 1 cara

```
glLightModeli(LIGHT_MODEL_TWO_SIDE,  
GL_TRUE);
```

Materiales

- Similar a las luces

```
void glMaterial{if}[v] (GLenum face,  
GLenum pname, TYPE param )
```

- Face: GL_FRONT, GL_BACK,
GL_FRONT_AND_BACK

Materiales

Nombre del parámetro	Valor por defecto	Significado
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	color ambiente del material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	color difuso del material
GL_AMBIENT_AND_DIFFUSE		color ambiente y difuso del material
GL_SPECULAR	(0.0,0.0, 0.0, 1.0)	color especular del material
GL_SHININESS	0.0	exponente especular
GL_EMISSION	(0.0, 0.0,0.0, 1.0)	color de emisión del material
GL_COLOR_INDEXES	0.0	ambiente, difusa y índices de color especular

Reflejo difuso y ambiente

- Difuso: Le afecta el color y el ángulo de la luz difusa
- Ambiente: Le afecta la luz ambiente global y la de las fuentes de luz
- El reflejo difuso y el ambiente suelen ser del mismo color

```
GLfloat mat_amb_diff[]={0.1, 0.5, 0.8, 1.0}  
glMaterialfv(GL_FRONT_AND_BACK,  
             GL_AMBIENT_AND_DIFFUSE,mat_amb_diff);
```

Reflejo especular

- Produce brillos. Depende del pto. de vista
- Se establece el color (GL_SPECULAR) y el tamaño (GL_SHININESS)

```
GLfloat mat_specular[]={1.0, 1.0, 1.0, 1.0}  
GLfloat low_shininess[]={5.0}  
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);  
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
```

Librería AUX

- Gestionar la creación y manipulación de ventanas
- Registrar eventos de teclado o ratón
- Dibujar objetos básicos: esfera, cilindro, icosaedro, tetera, etc.

Creación de una ventana

- void **auxInitDisplayMode**(GLbitfield mask)
- void **auxInitPosition**(GLint x, GLint y, GLint width, GLint height)
 - x,y esquina superior izquierda
- void **auxInitWindow**(GLbyte *titleString)

Creación de una ventana

- Ejemplo

```
auxInitDisplayMode(AUX_SINGLE |  
AUX_RGBA);  
auxInitPosition(100,100,250,250);  
auxInitWindow("Una ventana Opengl");
```

Manejo de eventos de ventana

- Reescalado de ventana

void **auxReshapeFunc**

(void (*funcion)(Glsizei, GLsizei))

- Especifica la función a llamar cuando la ventana cambia de tamaño, se mueve o se expone

Manejo de eventos de ventana

- Eventos de teclado

void **auxKeyFunc**

(GLint key, void (*funcion))(void))

- Especifica la función que se llama al pulsar una tecla (AUX_A,...,AUX_Z, AUX_UP, AUX_RIGHT...)

Manejo de eventos de ventana

- Eventos de ratón

void **auxMouseFunc**(GLint button, GLint mode, void(*funcion)(AUX_EVENTREC *))

- button: AUX_LEFT_BUTTON, ...
- mode: AUX_MOUSE_DOWN, ...
- funcion: Función que se ejecutará cuando el botón especificado por button esté en el modo especificado por mode

Dibujar objetos simples

- void **auxWireSphere**(Gldouble radius)
- void **auxSolidSphere**(Gldouble radius)
- void **auxWireTorus**(Gldouble innerRadius, Gldouble outerRadius)
- void **auxWireCylinder**(Gldouble radius, Gldouble height)
- void **auxSolidTeapot**(Gldouble size)
- Etc.

Proceso en background

- void **auxIdleFunc**(void *func)

Especifica la función que será ejecutada si no hay ningún evento pendiente.

Si se pasa NULL como nombre de función se deshabilita la ejecución de la función.

Dibujar la escena

- void **auxMainLoop**
(void (*displayFunc)(void))

Especifica la función a ejecutar cada vez que la ventana tenga que ser actualizada (función de dibujo)

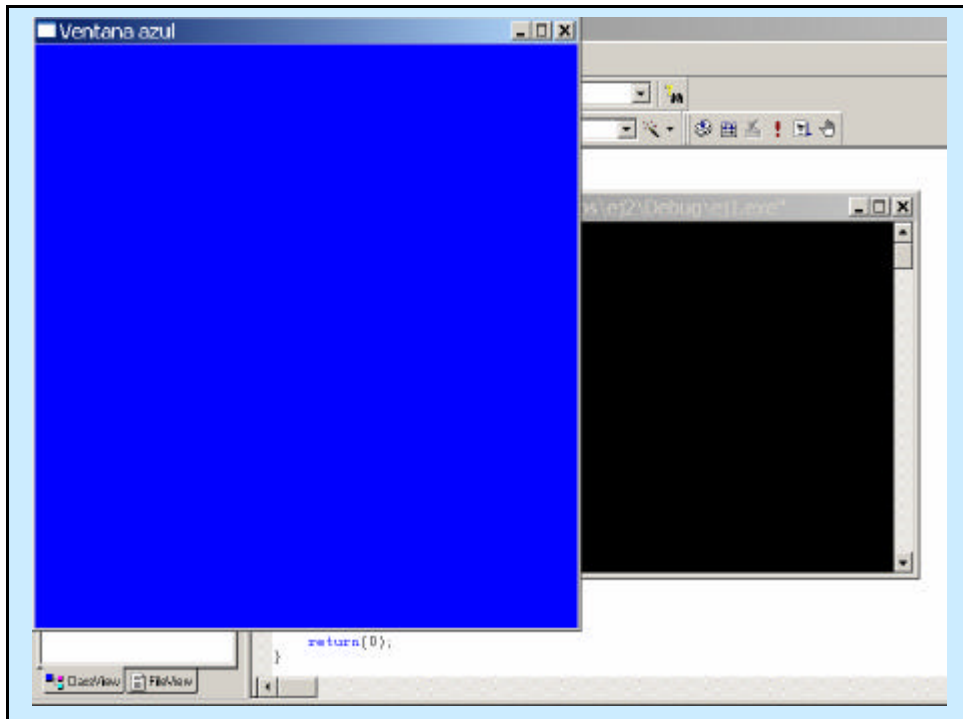
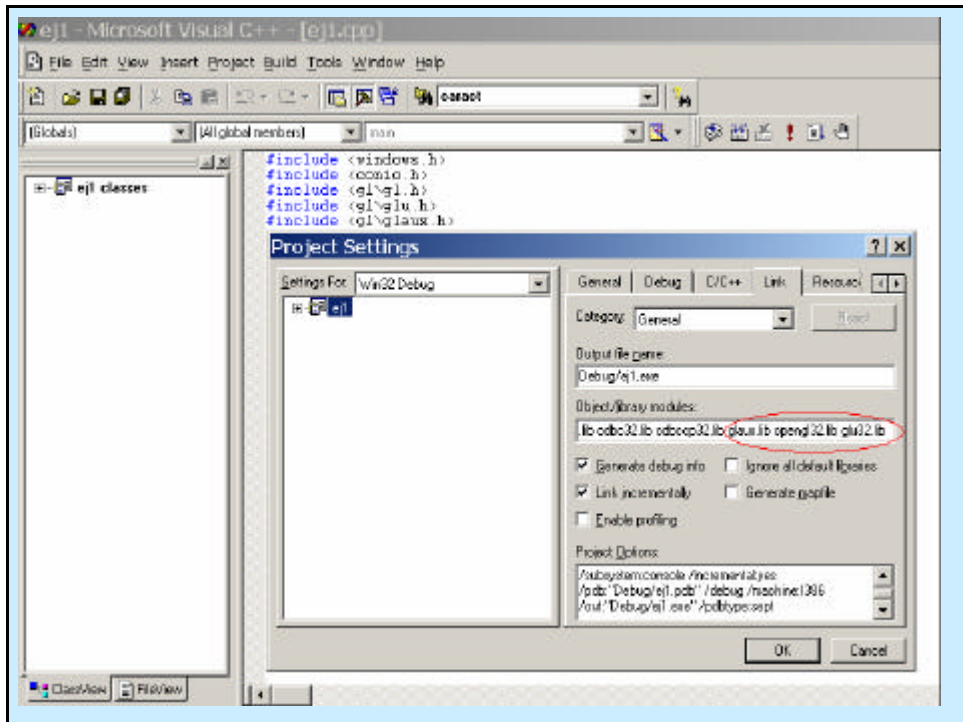
Ejemplo 1

- Ejemplo 1:
 - Creación de una ventana
 - Asignación de color de fondo
 - Sin definir auxMainLoop()
 - Sin definir auxReshapeFunc()

```
#include <windows.h>
#include <conio.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>

int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
    auxInitPosition (0, 0, 500, 500);
    auxInitWindow ("Ventana azul");
    glClearColor(0.0, 0.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
    getch();

    return(0);
}
```



Comportamiento

- No hay un bucle de refresco
 - La ventana se pinta una única vez al inicio de la ejecución
 - La ventana no se repinta en caso de ocultación
- No existe un tratamiento del evento de reescalado

Definición del bucle principal

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>

void CALLBACK display (void)
{
    glClearColor(0.0, 0.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

```
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE |
                       AUX_RGB);
    auxInitPosition (0, 0, 500, 500);
    auxInitWindow ("Ventana azul");
    auxMainLoop(display);
    return(0);
}
```

Utilizando librería GLUT

```
#include <GL/glut.h>

void display (void)
{
    glClearColor(0.0, 0.0, 1.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

```

/* Inicializacion de ventana con fondo azul
 * Bucle de pintado
 */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Ventana azul");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```

INICIALIZAR LIBRERÍA	void glutInit(int*argcp,char**argv)
INICIALIZAR UNA VENTANA	
void auxInitWindow(GLbyte *titleString)	void glutCreateWindow(GLbyte *titleString)
void auxInitDisplayMode(GLbitfield mask)	void glutInitDisplayMode(unsigned int mode);
void auxInitPosition(GLint x, GLint y, GLint width, GLint height)	void glutInitWindowSize(int width,int height); void glutInitWindowPosition(int x, int y);
MANEJO DE EVENTOS EN LA VENTANA	
void auxReshapeFunc(void (*funcion))(GLsizei, GLsizei)	void glutReshapeFunc(void(*func)(int width,int height));
void auxKeyFunc(GLint key, void (*funcion))(void)	void glutKeyboardFunc(void(*func)(unsigned char key, int x,int y));
void auxMouseFunc(GLint button,GLint mode, void (*funcion))(AUX_EVENTREC *)	void glutMouseFunc(void(*func)(int button,int state, int x,int y));
INICIALIZAR Y DIBUJAR OBJETOS TRIDIMENSIONALES	
void auxWireSphere(Gldouble radius)	void glutWireSphere (GLdouble radius, GLint slices,GLint stacks);
void auxSolidSphere(Gldouble radius)	void glutSolidSphere (GLdouble radius, GLint slices,GLint stacks);
void auxWireCube(Gldouble size)	void glutWireCube (GLdouble size);
void auxSolidCube(Gldouble size)	void glutSolidCube (GLdouble size);
etc...	etc...
MANEJAR UN PROCESO EN BACKGROUND	
void auxIdleFunc(void *func)	void glutIdleFunc(void(*func)(void));
EJECUTAR EL PROGRAMA	
void auxMainLoop(void (*displayFunc)(void))	void glutDisplayFunc(void(*func)(void)); void glutMainLoop(void);

Ejemplo 2

- Dibujar un cubo
- Con proyección en perspectiva
- Definición de bucle principal
- Definición de reescalado

Ejemplo 3

```
void CALLBACK funcionDePintado (void)
{
    //color de fondo
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);

    //color de dibujo
    glColor3f (0.0, 1.0, 0.0);
    auxWireCube(2.0);

    glFlush();
}
```

```
void CALLBACK funcionDeReescalado(GLsizei w, GLsizei h)
{

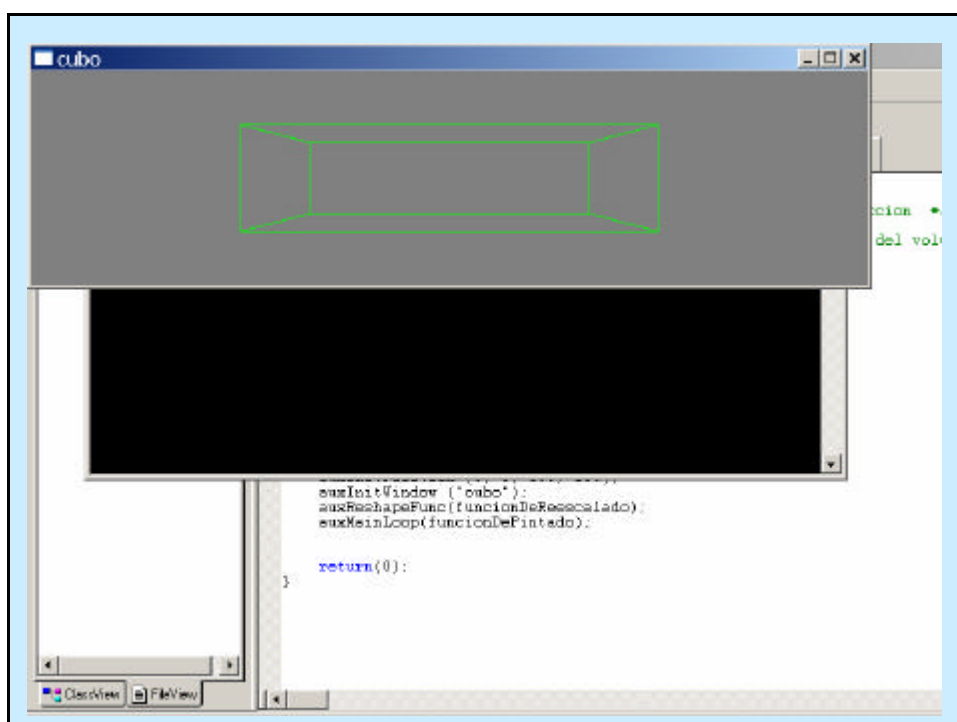
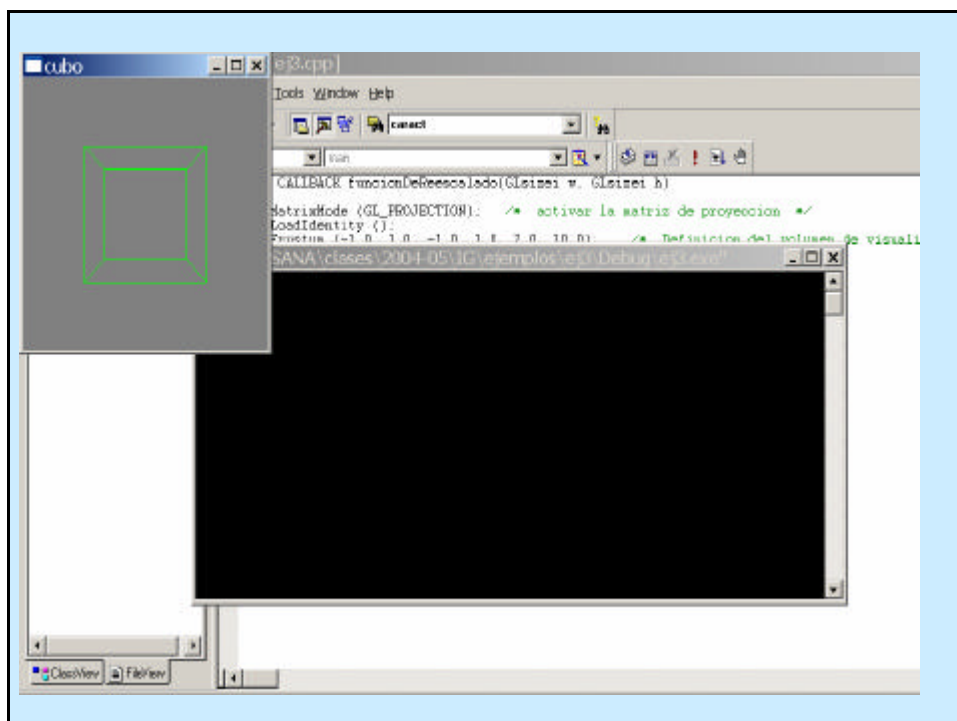
    glMatrixMode (GL_PROJECTION); //activar la matriz de
                                //proyeccion
    glLoadIdentity ();
    glFrustum (-1.0, 1.0, -1.0, 1.0, 2.0, 10.0);
                                //Definicion del volumen de
                                //visualizacion
    glViewport (0, 0, w, h); // definir el viewport
    glMatrixMode (GL_MODELVIEW);

    /* restaurar la matriz de modelo-vista como activa*/

}
```

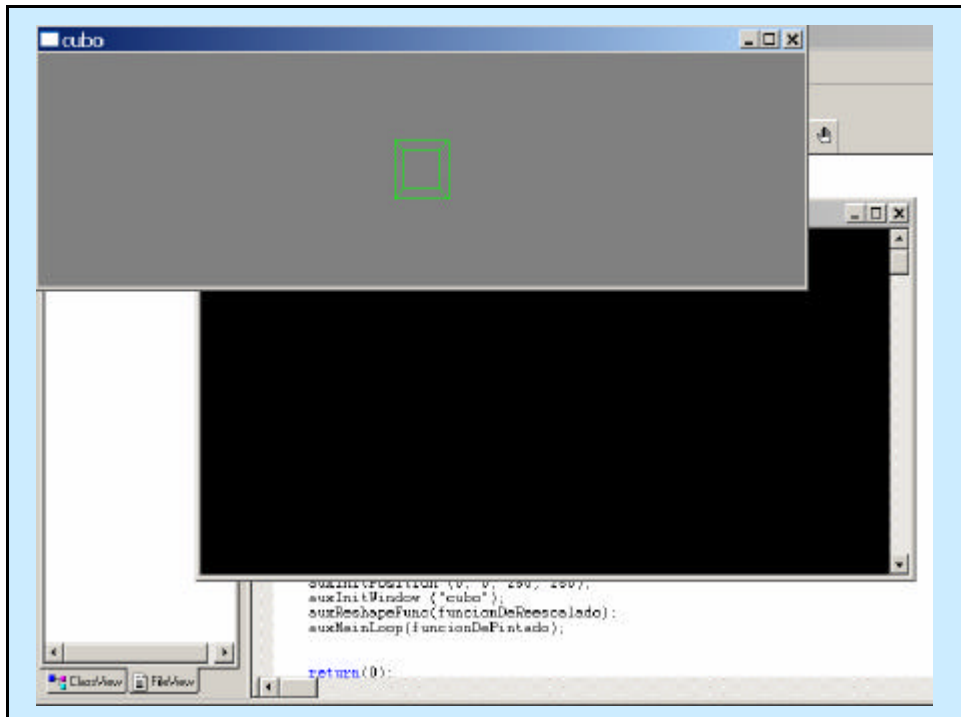
```
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
    auxInitPosition (0, 0, 250, 250);
    auxInitWindow ("cubo");
    auxReshapeFunc(funcionDeReescalado);
    auxMainLoop(funcionDePintado);

    return(0);
}
```



Mantener el aspecto en el reescalado

```
void CALLBACK funcionDeReescalado(GLsizei w,  
    GLsizei h)  
{  
    glMatrixMode (GL_PROJECTION); //activar la  
        //matriz de proyeccion  
    glLoadIdentity ();  
    gluPerspective(90.0, (float)w/(float)h,  
        2.0, 7.0);  
  
    glViewport (0, 0, w, h);  
    glMatrixMode (GL_MODELVIEW);  
}
```



Ejemplo evento teclado

- Detectar pulsación de la tecla 'r'
- Cada vez que se pulse, el cubo rotará un determinado ángulo sobre el eje Z
- Utilizando la función auxKeyFunc()

```
<includes ...>

GLfloat angulo=0.0;

void CALLBACK funcionDePintado (void)
{
    //color de fondo
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    glRotatef(angulo, 0.0, 0.0, 1.0);

    //color de dibujo
    glColor3f (0.0, 1.0, 0.0);
    auxWireCube(2.0);

    glFlush();
}
```

```
void CALLBACK funcionDeReescalado(GLsizei w, GLsizei h)
{
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(90.0, (float)w/(float)h, 2.0, 7.0);

    glViewport (0, 0, w, h);

    glMatrixMode (GL_MODELVIEW);
    /* restaurar la matriz de modelo-vista como activa*/
}
```

```
void CALLBACK funcionRotar(){

    angulo+=2.0;
    if(angulo>=360) angulo=0.0;
    funcionDePintado();

}
```

```
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE |
                       AUX_RGB);
    auxInitPosition (0, 0, 250, 250);
    auxInitWindow ("cubo");

    auxReshapeFunc (funcionDeReescalado);
    auxKeyFunc(AUX_r, funcionRotar);
    auxMainLoop(funcionDePintado);
    return(0);
}
```

Ejemplo animación

- El objeto gira autónomamente
- Definir una función de giro que se asociará mediante auxIdleFunc()
- Utilización de doble buffer (AUX_DOUBLE y auxSwapBuffers())

```

GLfloat angulo=0.0;

void CALLBACK funcionDePintado (void)
{

    //color de fondo
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    glRotatef(angulo, 0.0, 0.0, 1.0);

    //color de dibujo
    glColor3f (0.0, 1.0, 0.0);
    auxWireCube(2.0);

    glFlush();
    auxSwapBuffers();
}

```

```

void CALLBACK funcionDeReescalado(GLsizei w,
    GLsizei h)
{

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(90.0, (float)w/(float)h, 2.0,
    7.0);

    glViewport (0, 0, w, h);

    glMatrixMode (GL_MODELVIEW);
    /* restaurar la matriz de modelo-vista como
    activa*/

}

```

```
void CALLBACK funcionIdle(){  
  
    angulo+=0.5;  
    if(angulo>=360) angulo=0.0;  
  
    Sleep(50);  
  
    funcionDePintado();  
}
```

```
int main(int argc, char** argv)  
{  
    auxInitDisplayMode (AUX_DOUBLE |  
                        AUX_RGB);  
    auxInitPosition (0, 0, 250, 250);  
    auxInitWindow ("cubo");  
    auxReshapeFunc(funcionDeReescalado);  
    auxIdleFunc(funcionIdle);  
    auxMainLoop(funcionDePintado);  
  
    return(0);  
}
```

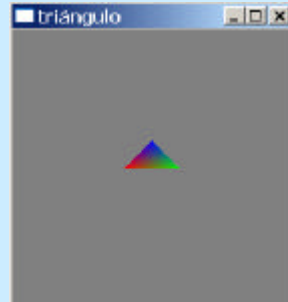
Modelo de sombreado

```
void CALLBACK funcionDePintado
(void)
{
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    glShadeModel(GL_SMOOTH);

    glBegin(GL_TRIANGLES);
        glColor3f (1.0, 0.0, 0.0);
        glVertex3f(-1.0, 0.0, 0.0);
        glColor3f (0.0, 1.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f (0.0, 0.0, 1.0);
        glVertex3f(0.0, 1.0, 0.0);
    glEnd();

    glFlush();
}
```



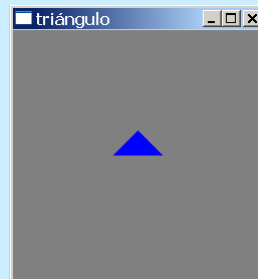
Modelo de sombreado

```
void CALLBACK funcionDePintado
(void)
{
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    glShadeModel(GL_FLAT);

    glBegin(GL_TRIANGLES);
        glColor3f (1.0, 0.0, 0.0);
        glVertex3f(-1.0, 0.0, 0.0);
        glColor3f (0.0, 1.0, 0.0);
        glVertex3f(1.0, 0.0, 0.0);
        glColor3f (0.0, 0.0, 1.0);
        glVertex3f(0.0, 1.0, 0.0);
    glEnd();

    glFlush();
}
```



Zoom

- Modificar el zoom
- Aumentar el factor de zoom al presionar la “Z”
- Disminuir el factor de zoom al presionar la “z”

```
float zoom_factor=1.0;
GLsizei current_w, current_h;

void CALLBACK funcionDePintado (void)
{
    //color de fondo
    glClearColor(0.5, 0.5, 0.5, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    glShadeModel(GL_SMOOTH);
    glBegin(GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0);
    glVertex3f(-1.0, 0.0, 0.0);
    glColor3f (0.0, 1.0, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glColor3f (0.0, 0.0, 1.0);
    glVertex3f(0.0, 1.0, 0.0);
    glEnd();

    glFlush();
}
```

```
void CALLBACK funcionDeReescalado(GLsizei w, GLsizei h)
{

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(90.0 * zoom_factor, (float)w/(float)h, 2.0, 7.0);

    glViewport (0, 0, w, h);

    current_w=w;
    current_h=h;
    glMatrixMode (GL_MODELVIEW);
    /* restaurar la matriz de modelo-vista como activa*/
}
```

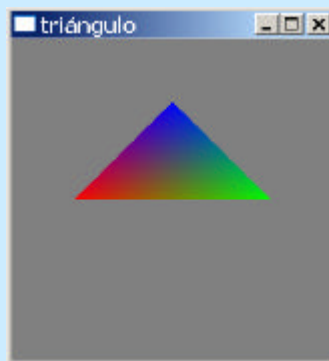
```
void CALLBACK aumentarZoom(){
    zoom_factor-=0.15;
    funcionDeReescalado(current_w,
        current_h);
}

void CALLBACK disminuirZoom(){
    zoom_factor+=0.15;
    funcionDeReescalado(current_w,
        current_h);
}
```

```
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE |
AUX_RGB);
    auxInitPosition (0, 0, 250, 250);
    auxInitWindow ("triángulo");
    auxKeyFunc(AUX_Z, aumentarZoom);
    auxKeyFunc(AUX_z, disminuirZoom);
    auxReshapeFunc(funcionDeReescalado);
    auxMainLoop(funcionDePintado);

    return(0);
}
```

Zoom



Test de profundidad

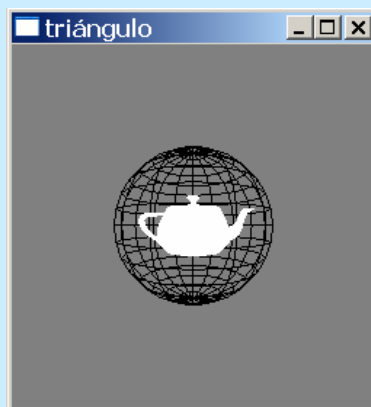
```
void CALLBACK funcionDePintado (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);

    glRotatef(90.0, 1.0, 0.0, 0.0);
    glColor3f(0.0, 0.0, 0.0);
    auxWireSphere(2);

    glRotatef(-90.0, 1.0, 0.0, 0.0);
    glColor3f(1.0, 1.0, 1.0);
    auxSolidTeapot(1);

    glFlush();
}
```

Sin test de profundidad

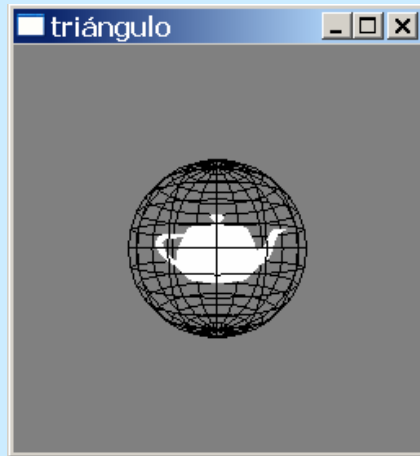


Test de profundidad

```
void inicializar(){  
  
    glClearColor(0.5, 0.5, 0.5, 1.0);  
    glColor3f(0.0, 0.0, 1.0);  
    glShadeModel(GL_FLAT);  
  
    glEnable(GL_DEPTH_TEST);  
}
```

```
void CALLBACK funcionDePintado (void)  
{  
  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
    glLoadIdentity();  
    glTranslatef(0.0, 0.0, -5.0);  
  
    glRotatef(90.0, 1.0, 0.0, 0.0);  
    glColor3f(0.0, 0.0, 0.0);  
    auxWireSphere(2);  
  
    glRotatef(-90.0, 1.0, 0.0, 0.0);  
    glColor3f(1.0, 1.0, 1.0);  
    auxSolidTeapot(1);  
  
    glFlush();  
}
```

Con test de profundidad



Material e iluminación

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>

GLsizei current_w, current_h;

void inicializar(){
    GLfloat mat_spec[]={1.0, 1.0,
        1.0, 1.0};
    GLfloat mat_shin[]={50.0};
    GLfloat light_pos[]={1.0,
        1.0, 1.0, 0.0};
    GLfloat light_dif[]={1.0,
        0.0, 0.0, 1.0};

    glClearColor (0.0, 0.0, 0.0,
        0.0);
    glShadeModel (GL_SMOOTH);

    glMaterialfv(GL_FRONT,
        GL_SPECULAR, mat_spec);
    glMaterialfv(GL_FRONT,
        GL_SHININESS, mat_shin);
    glLightfv(GL_LIGHT0,
        GL_POSITION, light_pos);
    glLightfv(GL_LIGHT0,
        GL_DIFFUSE, light_dif);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

}
```

```
void CALLBACK funcionDePintado
(void)

{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -5.0);
    auxSolidSphere(1);
    glFlush();
}
```

```
void CALLBACK funcionDeReescalado(GLsizei w, GLsizei h)
{

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluPerspective(45.0, (float)w/(float)h, 1.0, 10.0);

    glViewport (0, 0, w, h);

    current_w=w;
    current_h=h;
    glMatrixMode (GL_MODELVIEW);
}
```

```
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE |
    AUX_RGB);
    auxInitPosition (0, 0, 250, 250);
    auxInitWindow ("esfera");
    inicializar();
    auxReshapeFunc(funcionDeReescalado);
    auxMainLoop(funcionDePintado);

    return(0);
}
```

Material e iluminación



Prácticas

