

OpenGL.

Domingo Martín

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

Curso 2013-14

Índice

OpenGL.

- 1 Introducción
- 2 Glut
- 3 Primitivas gráficas
- 4 Trasformaciones
- 5 Cámara
- 6 Comportamiento de la visualización
- 7 Iluminación
- 8 Texturas
- 9 Selección

Introducción

- ▶ ¿Qué es OpenGL?
- ▶ Open Graphics Library
 - ▶ Una interfaz con el hardware gráfico
 - ▶ Arquitectura cliente/servidor
 - ▶ Independiente del hardware
 - ▶ Un conjunto de funciones que permiten definir, crear y modificar gráficos Orientado a gráficos 3D
 - ▶ Permite simplificar la obtención de imágenes a partir de modelos 3D

Introducción

- ▶ Es una máquina de estados
- ▶ Puede operar en modo
 - ▶ Inmediato
La información de los elementos gráficos no es almacenada por OpenGL
 - ▶ Retenido
La información de los elementos gráficos es almacenada por OpenGL
 - ▶ Estructuración
 - ▶ Jerarquización

Glut

- ▶ OpenGL sólo se encarga de la visaulización no de la interacción y manejo de ventanas
- ▶ Solución → glut
 - ▶ Permite la creación de aplicaciones interactivas orientadas a eventos
 - ▶ Objetos
 - ▶ Creación de ventanas, menús
 - ▶ Control de dispositivos

Glut

► Creando una ventana con glut

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

int main(&argc, argv)
{
    int XMinimoVentanaX,YMinimoVentanaX,XTamaniaVentanaX,YTamaniaVentanaX;

    int Ventana_x=50;
    int Ventana_y=50;
    int Ventana_ancho=300;
    int Ventana_alto=300;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(Ventana_x,Ventana_y);
    glutInitWindowSize(Ventana_ancho,Ventana_alto);
    glutCreateWindow("Practica 1");
    glutMainLoop();
    return(1);
}
```

Glut

- ▶ `glutInitDisplayMode`
 - ▶ Indica las propiedades para de la imagen
 - ▶ Las distintas posibilidades se combinan mediante un OR
 - ▶ `GLUT_SIMPLE`: sólo la memoria de imagen delantera izquierda
 - ▶ `GLUT_DOUBLE`: las memorias de imagen delantera izquierda y trasera izquierda
 - ▶ `GLUT_INDEX`: representación del color indirectamente
 - ▶ `GLUT_RGB`: representación del color con el modelo RVA
 - ▶ `GLUT_RGBA`: representación del color con el modelo RVA más el canal alfa (transparencia)
 - ▶ `GLUT_DEPTH`: memoria para valores de profundidad (eliminación de partes ocultas, z-buffer)
 - ▶ `GLUT_STENCIL`: memoria de estarcido

Glut

- ▶ `glutInitWindowPosition`
 - ▶ Indica las coordenadas de la esquina superior izquierda de la ventana
- ▶ `glutInitWindowSize`
 - ▶ Indica el tamaño de la ventana
- ▶ `glutCreateWindow`
 - ▶ Intenta crear una ventana con el nombre especificado
- ▶ `glutMainLoop`
 - ▶ Entra en el bucle de eventos. Comienza el programa. Se crea la ventana

Glut

- ▶ Para poder interactuar hay que hacer uso de los eventos (acciones del usuario o del programa)
- ▶ En glut se implementan mediante callbacks
 - ▶ `glutDisplayFunc(Función)`: función para dibujar la imagen
 - ▶ `glutReshapeFunc(Función)`: función para cuando se ha producido un cambio en el tamaño de la ventana
 - ▶ `glutKeyboardFunc(Función)`: función para cuando se ha pulsado una tecla normal
 - ▶ `glutSpecialFunc(Función)`: función para cuando se ha pulsado una tecla especial
 - ▶ `glutMouseFunc(Función)`: función para cuando se ha pulsado o soltado una tecla del ratón
 - ▶ `glutMotionFunc(Función)`: función para cuando se mueve el ratón con una tecla del ratón pulsada

Glut

- ▶
 - ▶ `glutPassiveMotionFunc(Función)`: función para cuando se mueve el ratón sin una tecla del ratón pulsada
 - ▶ `glutVisibilityFunc(Función)`: función para cuando la ventana cambia de estado de visibilidad
 - ▶ `glutEntryFunc(Función)`: función para cuando el cursor entra o sale en la ventana
 - ▶ `glutIdleFunc(Función)`: función para cuando glut está desocupado
 - ▶ `glutTimerFunc(Función)`: función para cuando un temporizador llega al final de la cuenta

Primitivas gráficas

- Para poder definir y dibujar cualquier primitiva gráfica OpenGL se usa la estructura:

```
glBegin(tipo_primitiva)
vertice
vertice
vertice...
glEnd()
```

Primitivas gráficas

- ▶ Los TIPOS permitidos son:
 - ▶ GL_POINTS: los vértices se representan como puntos individuales
 - ▶ GL_LINES: cada pareja de vertices representa una línea
 - ▶ GL_LINE_STRIP: los vertices forman una secuencia abierta de líneas
 - ▶ GL_LINE_LOOP: los vertices forman una secuencia cerrada de líneas
 - ▶ GL_POLYGON: los vertices definen un polígono
 - ▶ GL_QUADS: los vertices definen cuadriláteros
 - ▶ GL_QUAD_STRIP: los vertices definen una tira de cuadriláteros
 - ▶ GL_TRIANGLES: los vertices definen triángulos
 - ▶ GL_TRIANGLE_STRIP: los vertices definen una tira de triángulos
 - ▶ GL_TRIANGLE_FAN: los vertices definen un abanico de triángulos

Primitivas gráficas

- ▶ Los vértices se representan mediante `glVertex{dimension}{tipo}{vector}`
- ▶ `glVertex{2,3,4}{b,s,i,f,d,ub,us,ui}{v}`
 - ▶ `b` → 8 bits → entero con signo → `GLbyte`
 - ▶ `s` → 16 bits → entero con signo → `GLshort`
 - ▶ `i` → 32 bits → entero con signo → `GLint`
 - ▶ `f` → 32 bits → real → `GLfloat`, `GLclampf`
 - ▶ `d` → 64 bits → doble → `Gldouble`, `GLclampd`
 - ▶ `ub` → 8 bits → entero sin signo → `GLubyte`, `GLboolean`
 - ▶ `us` → 16 bits → entero sin signo → `GLuint`, `GLenum`
 - ▶ `ui` → 32 bits → entero sin signo → `GLuint`, `GLbitfield`
 - ▶ `v` → se manda una dirección

Primitivas gráficas

► Ejemplos

```
glVertex2i(1,2)  
glVertex3i(1,2,4)  
glVertex3f(1.,2.2,43.56)
```

```
glVertex4f(1.0,0.5,3.4,5)  
  
GLfloat vector[]={1.0,0.5,3.4,5}  
glVertex4fv(vector)
```

Primitivas gráficas

► Ejemplos

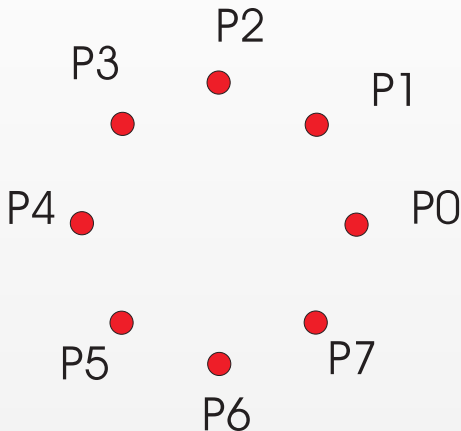
```
glBegin(GL_POINTS)  
glVertex3f(0,0,0)  
glVertex3f(1,1,1)  
glEnd()
```

```
glBegin(GL_LINES)  
glVertex3f(0,0,0)  
glVertex3f(1,1,1)  
glEnd()
```

```
glBegin(GL_TRIANGLES)  
glVertex3f(0,0,0)  
glVertex3f(1,1,1)  
glVertex3f(0.5,1,0)  
glEnd()
```

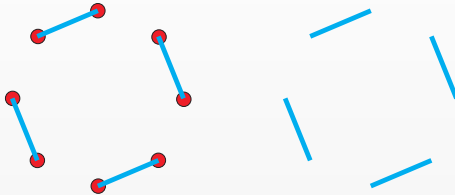
Primitivas gráficas

► GL_POINTS



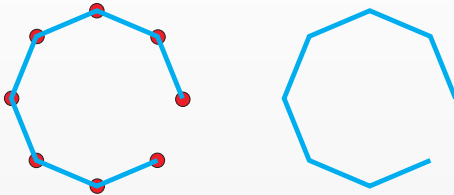
Primitivas gráficas

► GL_LINES



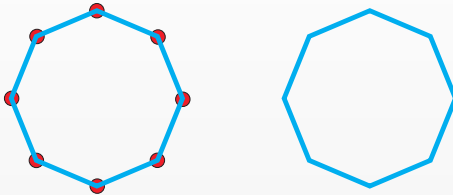
Primitivas gráficas

► GL_LINE_STRIP



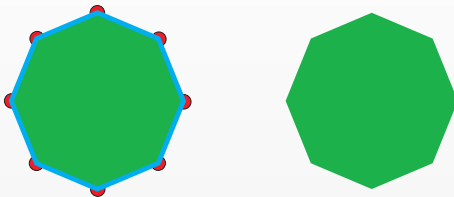
Primitivas gráficas

► GL_LINE_LOOP



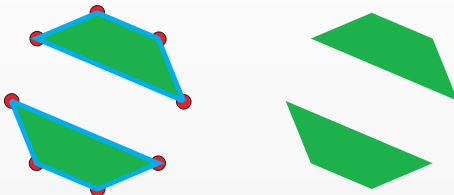
Primitivas gráficas

► GL_POLYGON



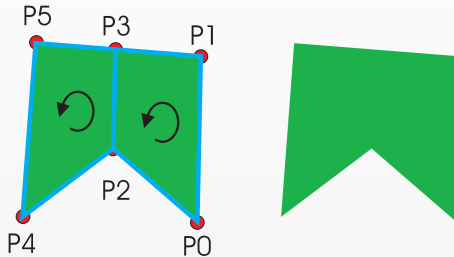
Primitivas gráficas

► GL_QUADS



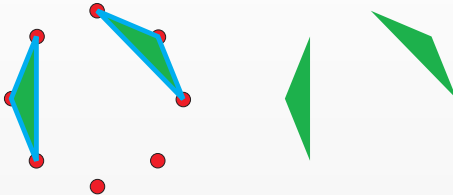
Primitivas gráficas

► GL_QUAD_STRIP



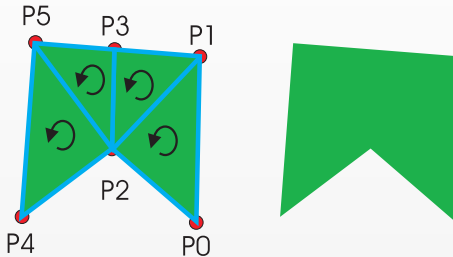
Primitivas gráficas

► GL_TRIANGLES



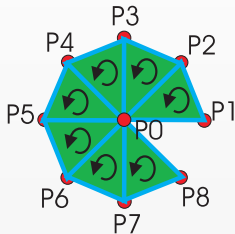
Primitivas gráficas

► GL_TRIANGLE_STRIP



Primitivas gráficas

► GL_TRIANGLE_FAN



Primitivas gráficas

- ▶ Los polígonos pueden dibujarse de tres formas usando la misma geometría:
 - ▶ Vértices
 - ▶ Frontera
 - ▶ Interior
- ▶ `glPolygonMode(LADO_CARA, TIPO_ELEMENTO)`
 - ▶ `LADO_CARA`
 - ▶ `GL_FRONT`: Lado exterior del polígono
 - ▶ `GL_BACK`: Lado interior del polígono
 - ▶ `GL_FRONT_AND_BACK`
 - ▶ `TIPO_ELEMENTO`
 - ▶ `GL_POINT`: Sólo se dibujan los vértices
 - ▶ `GL_LINE`: Sólo se dibuja la frontera (líneas)
 - ▶ `GL_FILL`: Sólo se dibuja el interior (relleno)

Primitivas gráficas

► Ejemplo

```
glColor3f(1,0,0);  
glPolygonMode(GL_FRONT, GL_POINT);  
glBegin(GL_POLYGON);  
dibujarListaPuntos(NumPuntosCirculo, Lista2);  
glEnd();  
glColor3f(0,0,1);  
glPolygonMode(GL_FRONT, GL_LINE);  
glBegin(GL_POLYGON);  
dibujarListaPuntos(NumPuntosCirculo, Lista2);  
glEnd();  
glColor3f(0,1,0);  
glPolygonMode(GL_FRONT, GL_FILL);  
glBegin(GL_POLYGON);  
dibujarListaPuntos(NumPuntosCirculo, Lista2);  
glEnd();
```

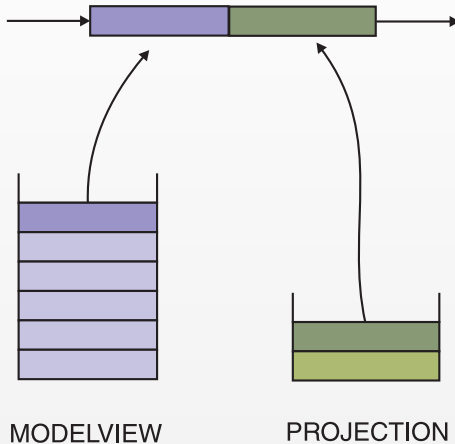
Transformaciones

- ▶ Cada vértice que es definido sufre una serie de transformaciones
- ▶ Las transformaciones se definen con matrices de 4x4
- ▶ Se distinguen 3 tipos de transformaciones
- ▶ Se guardan en pilas (LIFO)
 - ▶ GL_PROJECTION: Pila para la matriz con la transformación de proyección
 - ▶ GL_MODELVIEW: Pila para la matriz con la transformación de modelado y vista
 - ▶ GL_TEXTURE: Pila para la matriz con la transformación sobre las texturas
- ▶ Todos los elementos gráficos sufren las transformaciones que se encuentran en lo alto (top) de las pilas
- ▶ El orden de aplicación de las transformaciones es:

$$V \rightarrow \text{Modelview} \rightarrow \text{Projection} \rightarrow V'$$

Transformaciones

► Orden de las transformaciones

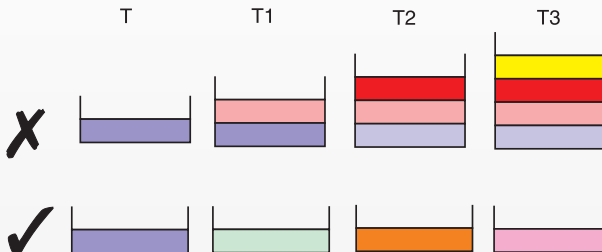


Transformaciones

- ▶ Operaciones sobre la pila
 - ▶ `glMatrixMode(PILA)`
 - ▶ Indica con que pila se va a operar, haciéndola activa
 - ▶ `glLoadIdentity()`
 - ▶ Carga la matriz identidad en lo alto de la pila
 - ▶ `glLoadMatrix{fd}(M)`
 - ▶ Carga la matriz M en lo alto de la pila
 - ▶ `glMultMatrix{fd}(M)`
 - ▶ Multiplica la matriz que está en lo alto de la pila con M
 - ▶ `glPushMatrix()`
 - ▶ Aumenta el puntero del tope de la pila activada
 - ▶ Copia en el tope la matriz de la posición anterior
 - ▶ `glPopMatrix()`
 - ▶ Decrementa el puntero del tope de la pila activada

Transformaciones

- Dada una pila activa, todas las transformaciones que se definan se van acumulando a la que se encuentra en el tope de la misma

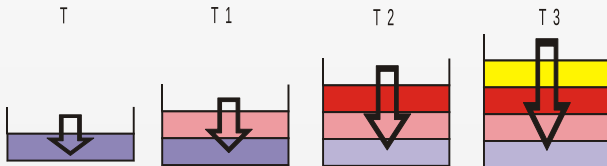


Transformaciones

- ▶ El orden es importante
- ▶ La multiplicación de matrices no es conmutativa
- ▶ Al utilizar una pila →

¡El orden de aplicación es el inverso al orden de escritura!

- ▶ Rotacion() → T1
- ▶ Traslacion() → T2
- ▶ Escalado() → T3



Transformaciones

- ▶ Transformaciones
 - ▶ `glTranslate{fd}($\delta_x, \delta_y, \delta_z$)`
 - ▶ Aplica una traslación
 - ▶ `glScale{fd}(F_x, F_y, F_z)`
 - ▶ Aplica un escalado
 - ▶ `glRotate{fd}(α, x, y, z)`
 - ▶ Aplica una rotación de α grados sexagesimales con respecto al eje definido por las tres coordenadas

Proyección

- ▶ Al implementar la cámara virtual tenemos que distinguir entre:
 - ▶ Proyección
 - ▶ Transformación que permite pasar de 3 dimensiones a 2 dimensiones
 - ▶ Se implementa mediante una transformación que se guarda en la pila `GL_PROJECTION`
 - ▶ Localización
 - ▶ Transformación que permite posicionar y orientar la cámara
 - ▶ Se implementa mediante una secuencia de transformaciones que se guardan en la pila `GL_MODELVIEW`

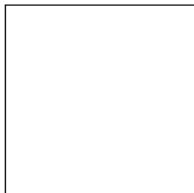
Proyección

- ▶ Proyección de perspectiva
 - ▶ `glFrustum(XMinimoVentanaMundo, XMaximoVentanaMundo, YMinimoVentanaMundo, YMaximoVentanaMundo, PlanoDelantero>0, PlanoTrasero>PlanoDelantero)`
 - ▶ Define los parametros que permiten calcular la matriz que realiza una proyección de perspectiva
 - ▶ Los planos delantero y trasero se dan como distancias, siendo positivas
- ▶ Proyección paralela
 - ▶ `glOrtho(XMinimoVentanaMundo, XMaximoVentanaMundo, YMinimoVentanaMundo, YMaximoVentanaMundo, PlanoDelantero, PlanoTrasero)`
 - ▶ Define los parametros que permiten calcular la matriz que realiza una proyección paralela
 - ▶ Los planos delantero y trasero se dan como distancias, con respecto al observador que mira en la dirección negativa de las zetas

Proyección

► Paralela

► Perspectiva



Localización

- Ejemplo de código para proyección de perspectiva

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glFrustum(-1, 1, -1, 1, 1, 1000);
```

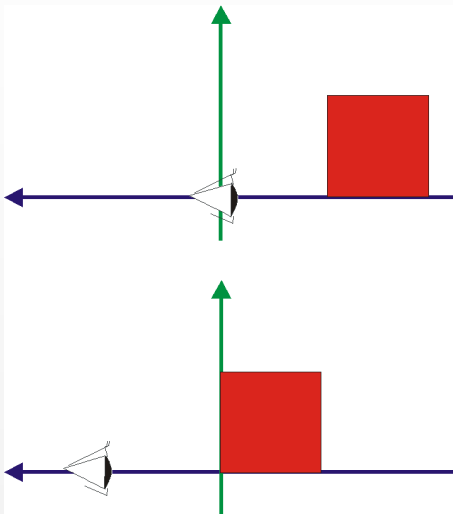
- Ejemplo de código para proyección paralela

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(-1, 1, -1, 1, -1, 1);
```

Localización

- ▶ Las transformaciones de localización se pueden calcular de dos maneras dependiendo del sistema de coordenadas que se considere fijo
 - ▶ Cámara fija en el origen
 - ▶ Se desplaza el objeto para que entre en la zona de visibilidad de la cámara
 - ▶ Objeto fijo en el origen
 - ▶ Se desplaza la cámara para que entre en la zona de visibilidad de la cámara
- ▶ En OpenGL la cámara está fija →
¡Mover la cámara → transformar los objetos!
- ▶ Las transformaciones de localización deben aplicarse antes que las transformaciones de modelado

Localización



Localización

- ▶ Ejemplo de código
- ▶ La traslación de -5 en el eje z de un objeto es equivalente a una traslación de 5 en el eje z de la cámara

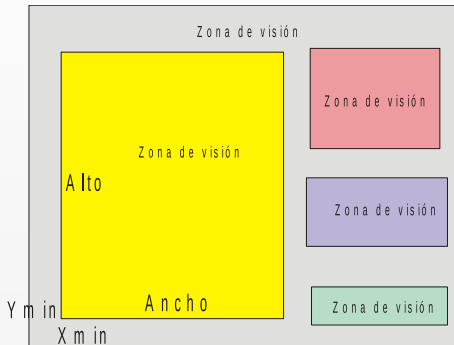
```
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity();  
glTranslate (0, 0, -5);
```


Viewport

- ▶ En general, la ventana no está ocupada únicamente por una aplicación
- ▶ La ventana se divide en zonas de visión (viewports)
- ▶ El sistema coordenado es derecho
- ▶ Sólo hay una zona de visión activa

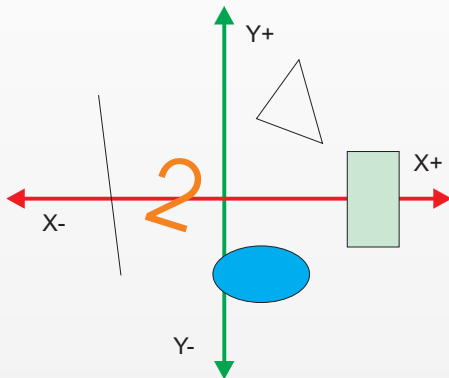
Viewport

- ▶ `glViewport(Xminimo,Yminimo,Ancho,Alto)`



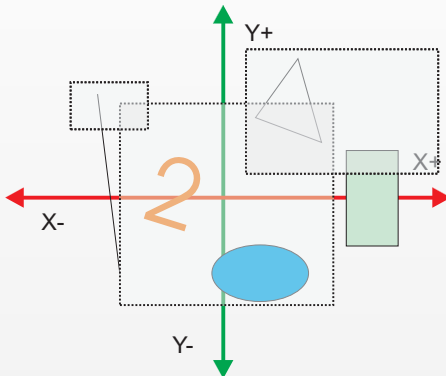
Viewport

- Vista del mundo



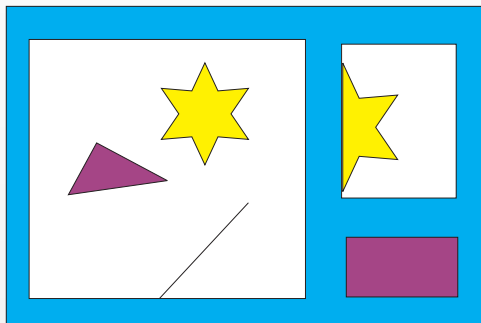
Viewport

- Vista del mundo con ventanas



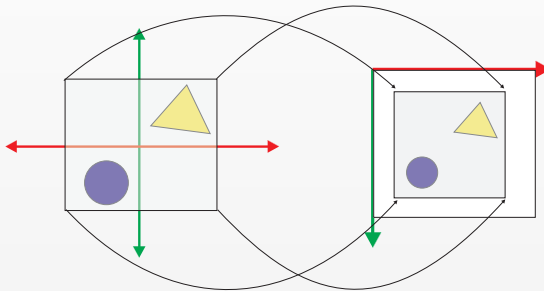
Viewport

- Vista en los viewports



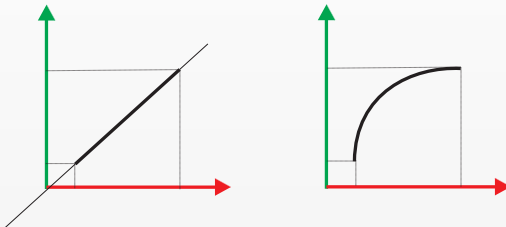
Viewport

- Transformación de visualización



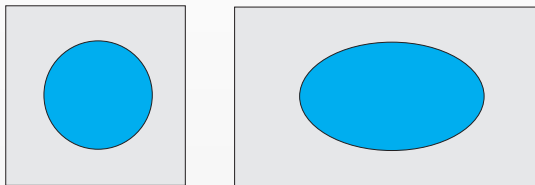
Viewport

- Tipo de transformación



Viewport

► Deformación



Eliminación de partes ocultas

- ▶ Las caras del modelo se dibujan en un orden arbitrario
- ▶ Problema → el resultado puede ser incorrecto
- ▶ Solución → eliminación de partes ocultas

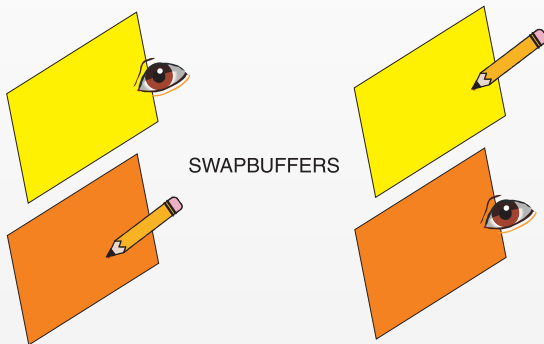
Z-buffer

Eliminación de partes ocultas

- ▶ `glutInitDisplayMode(... GLUT_DEPTH)`
 - ▶ Habilitar el uso del z-buffer
- ▶ `glClearDepth(1.0)`
 - ▶ Valor de inicialización
- ▶ `glClear(GL_DEPTH_BUFFER_BIT)`
 - ▶ Inicialización
- ▶ `glEnable(GL_DEPTH_TEST)`
 - ▶ Activación

Intercambio de zonas de memoria

- Para la mejora en la animación se recurre a la técnica del doble buffer



Iluminación

- ▶ Para poder iluminar hay que:
 - ▶ Usar las normales de caras o vértices dependiendo del modo de suavizado
 - ▶ Definir las fuentes de luz y sus propiedades
 - ▶ Definir las propiedades de los materiales de los objetos
 - ▶ Definir el modo de suavizado
 - ▶ Definir el modelo de iluminación

Normales

- ▶ Se usa la instrucción `glNormal3f{bsidf}{v}`
- ▶ Suavizado plano → 1 normal por cara
- ▶ La normal se replica en los vértices

```
glBegin(GL_TRIANGLES);  
glNormal3f(nx1,ny1,nz1);  
glVertex3f(x1,y1,z1);  
glVertex3f(x2,y2,z2);  
glVertex3f(x3,y3,z3);  
glEnd();
```

- ▶ Suavizado smooth (Gouraud) → 1 normal por vértice

```
glBegin(GL_POLYGON);  
glNormal3f(nx1,ny1,nz1);  
glVertex3f(x1,y1,z1);  
glNormal3f(nx2,ny2,nz2);  
glVertex3f(x2,y2,z2);  
glNormal3f(nx3,ny3,nz3);  
glVertex3f(x3,y3,z3);  
glEnd();
```

Luces

- ▶ Por defecto se pueden definir y utilizar 8 luces
- ▶ Existen varios parámetros, siendo los más importantes la posición y el color
- ▶ `glLight{fd}{v}(LUZ,ATRIBUTO,VALOR)`
 - ▶ LUZ
 - ▶ `GL_LIGHT0-GL_LIGHT7`
 - ▶ Luz local $\rightarrow (x, y, z, w \neq 0)$
 - ▶ Luz en el infinito $\rightarrow (x, y, z, w = 0)$
 - ▶ ATRIBUTO
 - ▶ `GL_POSITION`
 - ▶ `GL_AMBIENT`
 - ▶ `GL_DIFFUSE`
 - ▶ `GL_SPECULAR`

Materiales

- ▶ Se describen las características reflectivas del material
- ▶ `glMaterial{fd}{v}(LADO_CARA,COMPONENTE,VALOR)`
 - ▶ LADO_CARA
 - ▶ GL_FRONT
 - ▶ GL_BACK
 - ▶ GL_FRONT_AND_BACK
 - ▶ COMPONENTE
 - ▶ GL_AMBIENT
 - ▶ GL_DIFFUSE
 - ▶ GL_SPECULAR
 - ▶ GL_SHININESS

Modos de suavizado

- ▶ Los modelos suelen estar representados por caras planas
 - ▶ Si el objeto es poligonal → correcto
 - ▶ Si el objeto es curvo → incorrecto
- ▶ Solución → suavizado
- ▶ A partir de la variación del color producir la sensación de que el objeto es curvo
 - ▶ `glShadeModel(MODO)`
 - ▶ `GL_FLAT`: suavizado plano (no suavizado)
 - ▶ `GL_SMOOTH`: con suavizado de Gouraud

Modelo de iluminación

- ▶ Se pueden modificar los parámetros del modelo de iluminación y cómo se realizan ciertos cálculos
- ▶ `glMaterial{if}{v}(PARAMETRO, VALOR)`
 - ▶ `GL_LIGHT_MODEL_AMBIENT`
 - ▶ Intensidad de la componen global de luz ambiente
 - ▶ `GL_LIGHT_MODEL_LOCAL_VIEWER`
 - ▶ Cómo se calcula el ángulo de reflexión especular
 - ▶ `GL_LIGHT_MODEL_TWO_SIDE`
 - ▶ Si se ilumina una o las dos caras de los polígonos
 - ▶ `GL_LIGHT_MODEL_COLOR_CONTROL`
 - ▶ Si la componente especular se calcula aparte de la ambiente y difusa

General

- ▶ Para que se efectuen los cálculos de iluminación
 - ▶ glEnable(GL_LIGHTING)
- ▶ glEnable(LUZ)
 - ▶ LUZ
 - ▶ GL_LIGHT0-GL_LIGHT7

Ejemplo

```
GLfloat Luz0[4]={1,1,1,0};
GLfloat Ambiente[4]={1,1,1,1};
GLfloat Blanco[4]={1,1,1,1};
GLfloat Color[4]={1,0,0,1};
GLfloat Color1[4]={0.5,0,0,1};

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,Ambiente);
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE);
glMaterialfv(GL_FRONT,GL_AMBIENT,(GLfloat *) &Color1);
glMaterialfv(GL_FRONT,GL_DIFFUSE,(GLfloat *) &Color);
glMaterialfv(GL_FRONT,GL_SPECULAR,(GLfloat *) &Blanco);
glMaterialf(GL_FRONT,GL_SHININESS,10);
glLightfv(GL_LIGHT0,GL_POSITION,Luz0);
glShadeModel(GL_SMOOTH);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

...
```

General

- ▶ Las texturas son imagenes que se hacen corresponder con primitivas gráficas
- ▶ Hay texturas 1D, 2D y 3D
- ▶ El proceso para poder utilizar una textura consiste:
 - ▶ Cargar una imagen en un formato utilizable (RGB, RGBA, etc.)
 - ▶ Inicializar parámetros para usar la textura
 - ▶ Asignar la imagen a la textura de OpenGL
 - ▶ Habilitar el uso de la textura
 - ▶ Dibujar los objetos
 - ▶ Se deben definir unas coordenadas de textura para cada vértice

Parámetros

- ▶ `glTexParameter{if}{v}(OBJETIVO,PARAMETRO,VALOR)`
 - ▶ OBJETIVO
 - ▶ `GL_TEXTURE_1D`
 - ▶ `GL_TEXTURE_2D`
 - ▶ `GL_TEXTURE_3D`
 - ▶ ...
 - ▶ PARAMETRO
 - ▶ `GL_TEXTURE_MIN_FILTER`
 - ▶ `GL_TEXTURE_MAG_FILTER`
 - ▶ `GL_TEXTURE_WRAP_S`
 - ▶ `GL_TEXTURE_WRAP_T`
 - ▶ ...

Parámetros

- ▶ `glTexEnv{if}{v}(OBJETIVO,PARAMETRO,VALOR)`
 - ▶ OBJETIVO
 - ▶ `GL_TEXTURE_ENV`
 - ▶ `GL_TEXTURE_FILTER_CONTROL`
 - ▶ `GL_POINT_SPRITE`
 - ▶ PARAMETRO
 - ▶ `GL_TEXTURE_ENV_MODE`
 - ▶ `GL_TEXTURE_LOD_BIAS`
 - ▶ `GL_COMBINE_RGB`
 - ▶ ...
 - ▶ VALOR
 - ▶ `GL_MODULATE`
 - ▶ `GL_DECAL`
 - ▶ `GL_BLEND`
 - ▶ `GL_REPLAC`
 - ▶ ...

Asignación

- ▶ `glTexImage2D(OBJETIVO, NIVEL, FORMATO_INTERNO, ANCHO, ALTO, BORDE, FORMATO, TIPO, PÍXELES)`
 - ▶ OBJETIVO
 - ▶ `GL_TEXTURE_2D`
 - ▶ NIVEL
 - ▶ Level of detail ≥ 0
 - ▶ FORMATO_INTERNO
 - ▶ `GL_ALPHA`
 - ▶ `GL_RGB`
 - ▶ `GL_RGBA`
 - ▶ `GL_LUMINANCE`
 - ▶ `GL_LUMINANCE_ALPHA`.
 - ▶ BORDE
 - ▶ Debe ser 0
 - ▶ FORMATO=FORMATO_INTERNO
 - ▶ TIPO
 - ▶ `GL_UNSIGNED_BYTE`
 - ▶ `GL_UNSIGNED_SHORT_5_6_5`
 - ▶ `GL_UNSIGNED_SHORT_4_4_4_4`
 - ▶ `GL_UNSIGNED_SHORT_5_5_5_1`

Ejemplo

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// parametros de aplicacion de la textura
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

// asignacion de la textura
glTexImage2D(GL_TEXTURE_2D, 0, 4, Ancho, Alto, 0, GL_RGBA, GL_UNSIGNED_BYTE, &
    Textura);

glEnable(GL_TEXTURE_2D);
glBegin(GL_QUADS);
glTexCoord2f(0, 0);
glVertex3f(0, 0, 0);
glTexCoord2f(1, 0);
glVertex3f(10, 0, 0);
glTexCoord2f(1, 1);
glVertex3f(10, 10, 0);
glTexCoord2f(0, 1);
glVertex3f(0, 10, 0);
glEnd();
glDisable(GL_TEXTURE_2D);
```


General

- ▶ Hay diferentes procedimientos para realizar una selección:
 - ▶ Asignar un identificador a cada objeto o primitiva y observar a través de una pequeña ventana cual es el que se dibuja más cercano al observador
 - ▶ Lanzar un rayo desde el observador que pase por el pixel indicado e intersecciones los objetos y primitivas de la escena
 - ▶ Asignar un identificador a cada objeto o primitiva en forma de color, dibujar la escena con el z-buffer activado y comprobar en la posición deseada el color-identificador que hay
- ▶ OpenGL usa la primera forma

Procedimiento

- ▶ El procedimiento general es el siguiente:
 - 1** Obtener las coordenadas de la posición que marca la selección
 - 2** Crear el buffer de datos
 - 3** Cambiar al modo selección
 - 4** Redefinir el volumen de visión de tal manera que haya una pequeña ventana alrededor de la marca de selección (puede ser 1 pixel)
 - 5** Dibujar la escena metiendo en la pila de nombres aquellos que nos interesan
 - 6** Salir del modo selección y lectura de vector de datos

Procedimiento

- ▶ Crear el buffer de datos:
 - ▶ OpenGL devuelve los datos de la selección en un buffer
 - ▶ Hay que crearlo previamente a entrar al modo selección
 - ▶ Es un vector de enteros

```
GLuint Selection_buffer[BUFFER_SIZE];  
glSelectBuffer (BUFFER_SIZE,Selection_buffer);
```

Procedimiento

- ▶ Cambiar al modo selección:
 - ▶ OpenGL tiene tres modos de funcionamiento:
 - ▶ RENDER: Modo para dibujar (activado por defecto)
 - ▶ SELECT: Modo para seleccionar
 - ▶ FEEDBACK: Modo para obtener información geométrica de lo que se dibuja
 - ▶ Se cambia de modo con la instrucción *glRenderMode(MODO)*
 - ▶ MODO: GL_RENDER, GL_SELECT, GL_FEEDBACK

```
glRenderMode (GL_SELECT) ;
```

Procedimiento

- ▶ Redefinir el volumen de visión:
 - ▶ Consiste en redefinir el volumen de visión de tal manera que ocupe una zona pequeña alrededor de la posición marcada
 - ▶ Hay que hacer un paso de coordenadas de dispositivo a coordenadas de vista
 - ▶ Hay que conocer la relación entre el tamaño de un pixel y la ventana
 - ▶ Se usa una función auxiliar que simplifica el proceso: *gluPickMatrix(X, Y, ANCHO, ALTO, VIEWPORT)*
 - ▶ X: La coordenada x del dispositivo
 - ▶ Y: La coordenada y del dispositivo
 - ▶ ANCHO: Número de píxeles de ancho
 - ▶ ALTO: Número de píxeles de alto
 - ▶ VIEWPORT: Puntero a los datos del viewport
 - ▶ Para poder obtener los datos del viewport

```
GLint Viewport[4];  
glGetIntegerv (GL_VIEWPORT, Viewport);
```

Procedimiento

- ▶ Dibujado de la escena:
 - ▶ para identificar a cada objeto o primitiva se usa una pila de identificadores
 - ▶ Estos identificadores son enteros
 - ▶ Las instrucciones sobre la pila de nombres son:

```
glInitNames(): Inicializa la pila de nombres  
glLoadName(ID): Pone en el top de la pila a ID  
glPushName(ID): Aumenta el top de la pila y pone a ID  
glPopName(): Disminuye el top de la pila
```

Procedimiento

- Dibujado a nivel de objetos

```
glInitNames();  
  
for (i=0; i<NUM_OBJETOS; i++) {  
    glLoadName(i);  
    Draw_object(i);  
}
```

- Dibujado a nivel de primitiva

```
glInitNames();  
  
for (i=0; i<NUM_TRIANGLES; i++) {  
    glLoadName(i);  
    Draw_triangle(i);  
}
```

Procedimiento

- Dibujado con varios niveles

```
glInitNames();  
  
for (i=0; i<NUM_OBJETOS; i++) {  
    glPushName(i);  
    for (j=0; j<NUM_TRIANGLES; j++) {  
        glLoadName(j);  
        Draw_triangle(j);  
    }  
    glPopName();  
}
```


Procedimiento

► Ejemplo completo

```
void pick(unsigned int x, unsigned int y, unsigned int Width, unsigned
        int Height)
{
    GLuint Selection_buffer[BUFFER_SIZE];
    GLint Hits, Viewport[4];

    glGetIntegerv (GL_VIEWPORT, Viewport);
    glSelectBuffer (BUFFER_SIZE, Selection_buffer);

    glRenderMode (GL_SELECT);
    glInitNames();
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    gluPickMatrix ( x, Viewport[3] - y, Width, Height, Viewport);
    glFrustum(Min_x,Max_y,Min_y,Max_y,Front_plane,Back_plane);
    draw ();
    Hits = glRenderMode (GL_RENDER);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    glFrustum(Min_x,Max_y,Min_y,Max_y,Front_plane,Back_plane);
    ...
    Obtener informacion
}
```

Obtener los datos

- ▶ Para identificar los objetos y/o primitivas seleccionadas hay que interpretar los datos devueltos en el vector.
- ▶ Por cada selección se incluye la siguiente información
 - ▶ Número de nombres en la pila cuando se hace la selección
 - ▶ Valor mínimo de la profundidad (entero)
 - ▶ Valor máximo de la profundidad (entero)
 - ▶ Identificadores (habrá tantos como indique el primer valor)

Obtener los datos

- ▶ Para identificar los objetos y/o primitivas seleccionadas hay que interpretar los datos devueltos en el vector.
- ▶ Por cada selección se incluye la siguiente información
 - ▶ Número de nombres en la pila cuando se hace la selección
 - ▶ Valor mínimo de la profundidad (entero)
 - ▶ Valor máximo de la profundidad (entero)
 - ▶ Identificadores (habrá tantos como indique el primer valor)

N	Z min	Z max	Id
---	----------	----------	----

1	Z min	Z max	Id	3	Z min	Z max	Id	Id	Id	...
---	----------	----------	----	---	----------	----------	----	----	----	-----