

Introducción a OpenGL

1. Preámbulo.
2. Sintaxis de las funciones.
3. Primitivas gráficas.
4. GLUT.
5. Transformaciones geométricas.
6. Visualización.
7. Iluminación.
8. Texturas.
9. Selección
10. Lenguaje de Shading.

1

Introducción a OpenGL

1. Preámbulo

- OpenGL (**Graphics Library**) es una biblioteca gráfica que permite realizar aplicaciones gráficas interactivas tanto 2D como 3D.
- Interfaz (API) con el Hardware gráfico
- Existen APIs para distintos lenguajes de programación
- Existen implementaciones para diversas plataformas (Windows, Apple, Linux).
- Utiliza las capacidades de aceleración de las tarjetas gráficas (GPUs)
- Arquitectura cliente/servidor.
- Es portable (no contiene funciones para entornos de ventanas).



2

Introducción a OpenGL

1. Preámbulo

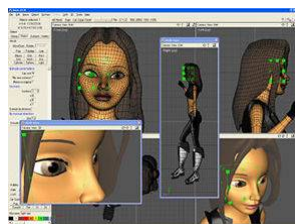
- Funciona como una máquina de estados.
- Bibliotecas asociadas:
 - GLU (OpenGL Utility Library) (texturas, proyecciones, NURBS)
 - GLUT (GLU Utility Toolkit) (entorno de ventanas)
 - Open Inventor (basada en objetos)
 - GLX (Extensión de OpenGL para X Windows)
 - OpenGL ES (OpenGL for Embedded Systems)
 - WebGL (inclusión de gráficos en 3D en un navegador)
- Versión 4.5 (especificación de Agosto de 2014). Operativas las versiones 2, 3 y 4 (4.4 NVIDIA GeForce GTX 750-780, 4.5 NVIDIA GeForce QUADRO)
- OpenGL Web site:
<http://www.opengl.org>



Introducción a OpenGL

1. Preámbulo

- Algunas aplicaciones que usan OpenGL
 - ArtCAM <http://www.artcam.com>
 - 3D Doctor <http://www.3d-doctor.com/>
 - AC3D <http://www.inivis.com/>
 - Medal of Honor <http://www.ea.com/medal-of-honor-warfighter>
 - Photo Modeler Pro <http://www.photomodeler.com/>
 - Sketchup <http://www.sketchup.com/>



Introducción a OpenGL

1. Preámbulo

- Librerías que usan OpenGL
 - OpenSceneGraph <http://www.openscenegraph.org/>
 - Apocalyx <http://apocalyx.sourceforge.net/>



Introducción a OpenGL

1. Preámbulo

- Vulkan, una nueva librería gráfica
- Especificación 1.0 febrero de 2016



Introducción a OpenGL

2. Sintaxis de las funciones

- Prefijo **gl** y letras mayúsculas en las iniciales de cada palabra que forma la orden:

glLineStipple

- Las constantes en mayúsculas, con palabras separadas por **underscores**:

GL_POLYGON

- Sufijos con dos partes. La primera indica el número de argumentos de la función (2 ó 3) y la segunda el tipo de dato:

glColor3f (1.0, 1.0, 1.0)

- En algunas órdenes puede aparecer una **v** final que indica un puntero a un vector:

glColor3fv (color_array)



8

Introducción a OpenGL

2. Sintaxis de las funciones

Tipo de dato	Tipo en lenguaje C	Tipo definido en OpenGL
b entero 8 b.	signed char	GLbyte
s entero 16 b.	short	GLshort
i entero 32 b.	long	GLint, GLsizei
f real 32 b.	Float	GLfloat, GLclampf
d real 64 b.	double	GLdouble, GLclampd
ub ent. sin signo 8 b.	unsigned char	GLubyte, GLboolean
us ent. sin signo 16 b.	unsigned short	GLushort
ui ent. sin signo 32 b.	unsigned long	GLuint, GLenum, GLbitfield



9

Introducción a OpenGL

3. Primitivas gráficas

- Se especifican dando un conjunto de vértices con `glVertex*()` delimitado por las llamadas a las funciones `glBegin` (tipo de función de dibujo) y `glEnd ()`.

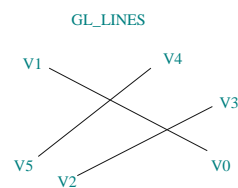
```
glBegin (GL_POINTS);      glBegin (GL_POINTS);
glVertex2f (0.0, 0.0);    glVertex3f (0.0, 0.0, 1.0);
glVertex2f (1.0, 0.0);    glVertex3f (1.0, 0.0, 0.0);
glVertex2f (2.0, 3.0);    glVertex3f (2.0, 3.0, 0.7);
...                       ...
glEnd ( );                glEnd ( );
```

- Argumentos de `glBegin`:

`glBegin (GLenum mode)`

`GL_POINTS` puntos individuales.

`GL_LINES` segmentos de líneas individuales (sin conexión).



10

Introducción a OpenGL

3. Primitivas gráficas

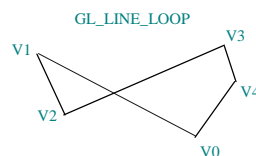
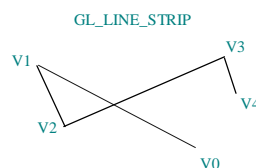
- Argumentos de `glBegin` (continuación):

`glBegin (GLenum mode)`

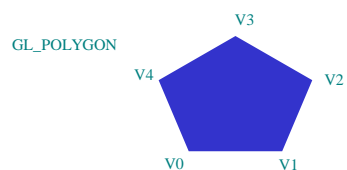
`GL_LINE_STRIP` dibuja una polilínea.

`GL_LINE_LOOP` dibuja una polilínea cerrada.

`GL_POLYGON` polígono convexo.



Polígono cóncavo



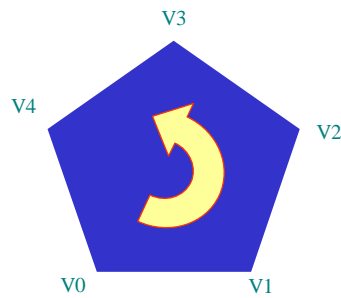
11

Introducción a OpenGL

3. Primitivas gráficas

- Polígono sentido positivo. Muy importante en el sentido del vector normal a la cara

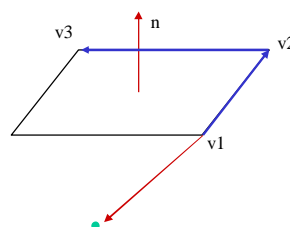
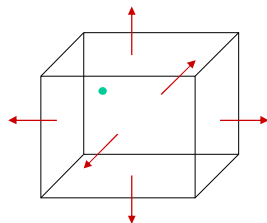
```
glBegin (GL_POLYGON);  
    glVertex3f (V0_x, V0_y,0.0);  
    glVertex3f (V1_x, V1_y,0.0);  
    glVertex3f (V2_x, V2_y,0.0);  
    glVertex3f (V3_x, V3_y,0.0);  
    glVertex3f (V4_x, V4_y,0.0);  
glEnd();
```



Introducción a OpenGL

3. Primitivas gráficas

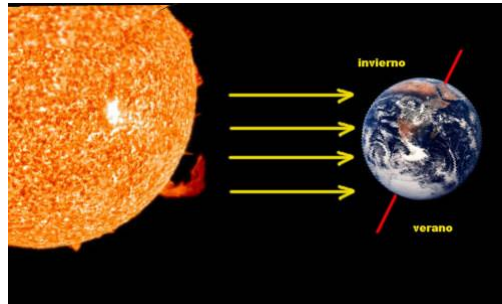
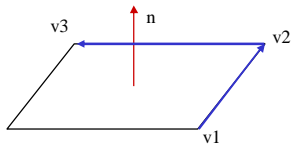
- Usos de la normal a una cara: contención de un punto en un objeto cerrado formado por caras convexas.



Introducción a OpenGL

3. Primitivas gráficas

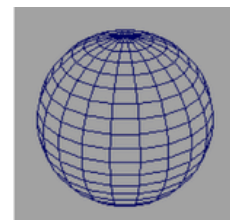
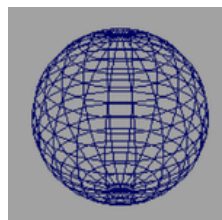
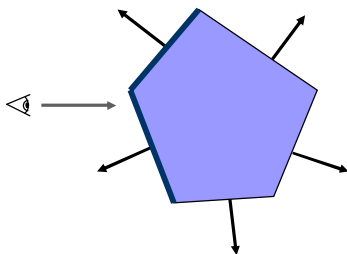
- Usos de la normal a una cara: iluminación (color) de un polígono



Introducción a OpenGL

3. Primitivas gráficas

- Usos de la normal a una cara: optimización mediante eliminación de la cara trasera en la visualización (**BackFace Culling**)



Introducción a OpenGL

3. Primitivas gráficas

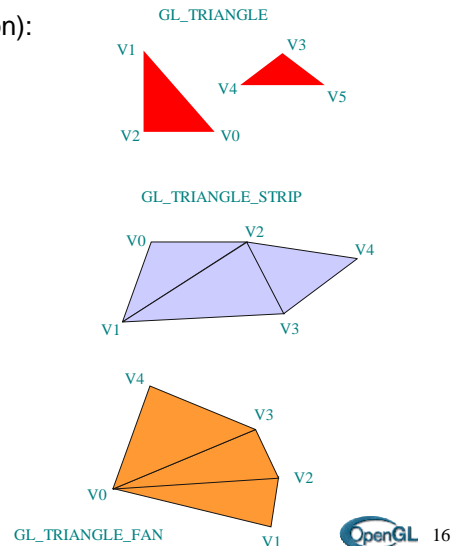
- Argumentos de `glBegin` (continuación):

`glBegin (GLenum mode)`

`GL_TRIANGLES` tripletas de vértices interpretadas como triángulos.

`GL_TRIANGLE_STRIP` dibuja triángulos encadenados en la misma orientación de modo que formen parte de una misma superficie.

`GL_TRIANGLE_FAN` enlaza triángulos en abanico.



Introducción a OpenGL

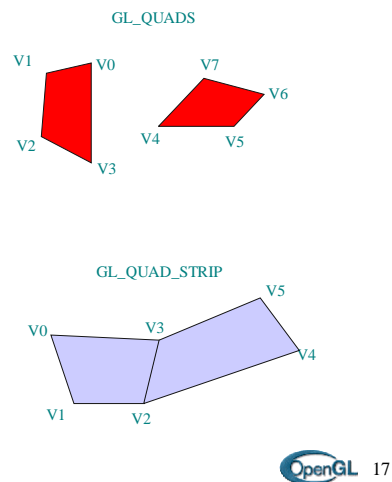
3. Primitivas gráficas

- Argumentos de `glBegin` (continuación):

`glBegin (GLenum mode)`

`GL_QUADS` Cuádruplas de vértices interpretadas como polígonos de cuatro lados.

`GL_QUAD_STRIP` dibuja cuadriláteros encadenados en la misma orientación de modo que formen parte de una misma superficie.



Introducción a OpenGL

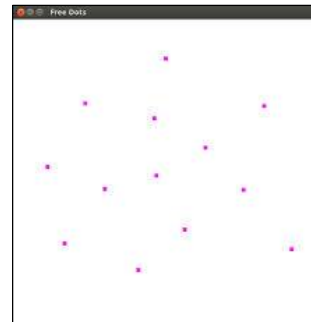
3. Primitivas gráficas

- Atributos: han de estar fuera de un bloque definido por `glBegin(funcion_salida)` y `glEnd ()`, excepto el atributo de color
- Atributo de punto.

`void glPointSize (GLfloat size)` fija el tamaño del punto en pixeles.

```
glBegin (GL_POINTS);  
glPointSize(10.0);  
glVertex3f (...);  
glVertex3f (...);  
glVertex3f (...);  
...  
glEnd ( );
```

```
glPointSize(10.0);  
glBegin (GL_POINTS);  
glVertex3f (...);  
glVertex3f (...);  
glVertex3f (...);  
...  
glEnd ( );
```



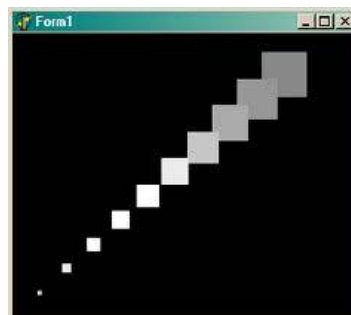
Introducción a OpenGL

3. Primitivas gráficas

- Atributo de punto.

`void glPointSize (GLfloat size)` fija el tamaño del punto en pixeles.

```
glPointSize(4.0);  
glBegin (GL_POINTS);  
glVertex3f (...);  
glEnd ( );  
  
glPointSize(8.0);  
glBegin (GL_POINTS);  
glVertex3f (...);  
glEnd ( );  
...
```



Introducción a OpenGL

3. Primitivas gráficas

- Atributos de línea.

`void glLineWidth (GLfloat width)` ancho en pixeles.

`void glLineStipple (GLint factor, GLushort pattern)` fija el estilo de una línea. Pattern es una serie de 16-bits (ceros o unos) que definen un patrón. Factor indica el número de repeticiones de la trama.

`void glEnable (GL_LINE_STIPPLE)` permite activar el estilo de líneas. El estilo solo se cambia después de usar esta función.

`void glDisable (GL_LINE_STIPPLE)` desactiva el estilo de líneas

PATTERN	FACTOR
0x00FF	1
0x00FF	2
0x0C0F	1
0x0C0F	3
0xAAAA	1
0xAAAA	2
0xAAAA	3
0xAAAA	4



Introducción a OpenGL

3. Primitivas gráficas

- Atributos de línea.

```
glColor3f(1.0, 1.0, 1.0);  
  
glEnable (GL_LINE_STIPPLE);  
glLineStipple (1, 0x00FF);  
glBegin( GL_LINES );  
    glVertex2f( -1.0, 0.0,0.0 );  
    glVertex2f( 0.5, 0.0,0.0 );  
glEnd();  
glDisable (GL_LINE_STIPPLE );
```



Introducción a OpenGL

3. Primitivas gráficas

- Atributos de polígono.

`void glPolygonMode (GLenum face, GLenum mode)` controla el estilo de dibujar un polígono tanto en su cara de atrás `GL_BACK`, cara delantera `GL_FRONT` y ambas `GL_FRONT_AND_BACK`. Los modos son `GL_POINT`, `GL_LINE` y `GL_FILL`.

`void glPolygonStipple (const GLubyte *mask)` permite definir un patrón de relleno de un polígono donde `mask` es un puntero a una matriz bidimensional de 32x32 bits.

`void glEnable (GL_POLYGON_STIPPLE)`

`void glDisable (GL_POLYGON_STIPPLE)` activación y desactivación del patrón.

`void glEdgeFlag (GLboolean flag)` se indica si las aristas han de dibujarse con el flag a `GL_TRUE` (por defecto) o no se dibujan con `GL_FALSE`.

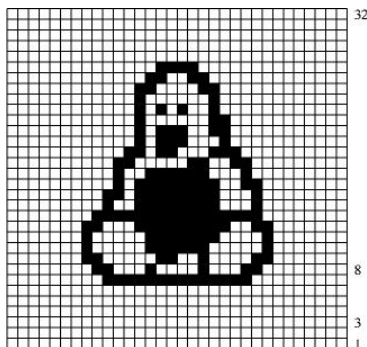


22

Introducción a OpenGL

3. Primitivas gráficas

- Atributos de polígono.



```
GLubyte tux[] = { 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
                  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
                  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
                  0x0, 0x7f, 0xfe, 0x0, 0x0, 0xc4, 0x23, 0x0,
                  0x1, 0x83, 0x21, 0x80, 0x1, 0x7, 0xe0, 0x80,
                  0x1, 0x7, 0xf0, 0x80, 0x1, 0x8f, 0xf9, 0x80,
                  0x0, 0xff, 0xff, 0x0, 0x0, 0x4f, 0xf1, 0x0,
                  0x0, 0x6f, 0xf1, 0x0, 0x0, 0x2f, 0xf3, 0x0,
                  0x0, 0x27, 0xe2, 0x0, 0x0, 0x30, 0x66, 0x0,
                  0x0, 0x1b, 0x1c, 0x0, 0x0, 0xb, 0x88, 0x0,
                  0x0, 0xb, 0x98, 0x0, 0x0, 0x8, 0x18, 0x0,
                  0x0, 0xa, 0x90, 0x0, 0x0, 0x8, 0x10, 0x0,
                  0x0, 0xc, 0x30, 0x0, 0x0, 0x6, 0x60, 0x0,
                  0x0, 0x3, 0xc0, 0x0, 0x0, 0x0, 0x0, 0x0,
                  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
                  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 };
```

```
...
glPolygonStipple(tux);
glBegin(GL_POLYGON);
...
glEnd();
.
```



23

Introducción a OpenGL

3. Primitivas gráficas

- Atributo color.

```
void glColor3f(float red, float green, float blue)
```

Ejemplo `glColor3f(1.0,0.0,0.0)` color rojo puro.

- Funciones para manejo de la memoria de imagen o **frame buffer**.

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

```
void glClear(GLbitfield {GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT}).
```

Ejemplo

```
glClearColor(0.0,0.0,1.0,0.0);
```

```
glClear(GL_COLOR_BUFFER_BIT);
```

borra la ventana de salida y la dibuja en color azul puro.

- Funciones de sincronización.

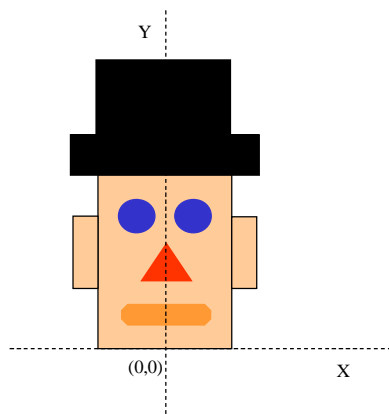
```
void glFlush(), void glFinish()
```



Introducción a OpenGL

3. Primitivas gráficas

- Ejemplo básico



```
glColor3f(0.0,0.0,0.0); // copa
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glBegin (GL_POLYGON);
glVertex3f (...);
glVertex3f (...);
...
glEnd ();
```

```
glBegin (GL_POLYGON); // ala
glVertex3f (...);
glVertex3f (...);
...
glEnd ();
```

```
glColor3f(1.0,0.8,0.6); // cara
glBegin (GL_POLYGON);
glVertex3f (...);
glVertex3f (...);
...
glEnd ();
```

```
glColor3f(0.0,0.0,0.0); // borde de cara
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glLineWidth(2);
glBegin (GL_POLYGON);
glVertex3f (...);
glVertex3f (...);
...
glEnd ();
```



3. Primitivas gráficas

- Ejemplo básico

```
void Circle (GLfloat radio, GLfloat cx, GLfloat cy, GLint n, GLenum modo)
{
    int i;

    if (modo==GL_LINE) glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else if (modo==GL_FILL) glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    else glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);

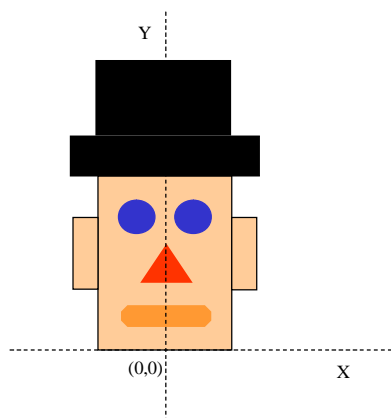
    glBegin( GL_POLYGON );
    for (i=0;i<n;i++)
        glVertex2f( cx+radio*cos(2.0*M_PI*i/n), cy+radio*sin(2.0*M_PI*i/n));
    glEnd();
}

....

glColor3f(1.0,0.0,1.0);
Circle(0.9,0.1,0.0,20,GL_FILL);
```

3. Primitivas gráficas

- Ejemplo básico.
Separando construcción del modelo
de visualización del modelo



// Funciones para visualizar un polígono de distintas formas

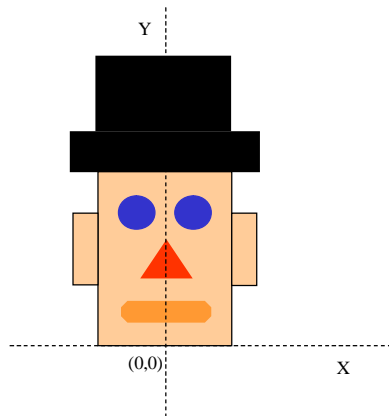
```
void Poligono_Solido (float v[][3], int n)
{
    int i;
    glPolygonMode(GL_FRONT, GL_FILL);
    glBegin(GL_POLYGON);
    for (i=0;i<n;i++) glVertex3f(v[i][0],v[i][1],v[i][2]);
    glEnd();
}
```

```
void Poligono_Hueco (float v[][3], int n)
{
    int i;
    glPolygonMode(GL_FRONT, GL_LINE);
    glBegin(GL_POLYGON);
    for (i=0;i<n;i++) glVertex3f(v[i][0],v[i][1],v[i][2]);
    glEnd();
}
...
```

Introducción a OpenGL

3. Primitivas gráficas

- Ejemplo básico.
Separando construcción del modelo
de visualización del modelo



```
// Modelo monigote
float copa_sombrero[4][3]={{-0.2,0.7,0.0},{0.2,0.7,0.0},
                             {0.2,0.95,0.0},{-0.2,0.95,0.0}};
float ala_sombrero[4][3]={{-0.35,0.55,0.0},{0.35,0.55,0.0},
                             {0.35,0.7,0.0},{-0.35,0.7,0.0}}
float cara_monigote[4][3]={{-0.2,0.0,0.0},{0.2,0.0,0.0},
                             {0.2,0.55,0.0},{-0.2,0.55,0.0}};
...

void Dibujar_Monigote ()
{
    // copa sombrero
    glColor3f(0.0,0.0,0.0);
    Poligono_Solido(copa_sombrero,4);

    // ala sombrero
    Poligono_Solido(ala_sombrero,4);

    // cara monigote
    glColor3f(1.0,0.8,0.6);
    Poligono_Solido(cara_monigote,4);

    // borde de cara monigote
    glColor3f(0.0,0.0,0.0);
    Poligono_hueco(cara_monigote,4);
    ...
}
```

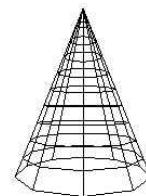


Introducción a OpenGL

3. Primitivas gráficas

- Objetos tridimensionales básicos mediante GLUT

```
void glutWireSphere (GLdouble radio, GLint n_lados_longitudinales, GLint n_lados_transversales)
void glutWireCube (GLdouble lado)
void glutWireTorus (GLdouble radio_interior, GLdouble radio_exterior, GLint n_lados, GLint n_anillos)
void glutWireIcosahedron ( void )      Radio fijado a 1.0.
void glutWireOctahedron ( void )       Radio fijado a 1.0.
void glutWireTetrahedron ( void )      Radio fijado a 1.0.
void glutWireDodecahedron ( void )     Radio fijado a 1.0.
void glutWireCone (GLdouble radio, GLdouble altura, GLint n_lados_longitudinales,
                  GLint n_lados_transversales)
void glutWireTeapot (GLdouble tamaño)
```



Introducción a OpenGL

3. Primitivas gráficas

- Objetos tridimensionales básicos mediante GLU

```
void gluCylinder (GLUquadric * objeto, GLdouble base, GLdouble tapa, GLdouble altura,  
                 GLint n_lados_longitudinales, GLint n_lados_transversales)
```

```
void gluDisk (GLUquadric * objeto, GLdouble radio_interior, GLdouble radio_exterior, GLint n_sectores,  
             GLint n_pistas)
```

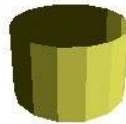
```
void gluSphere (GLUquadric * objeto, GLdouble radio, GLint n_lados_longitudinales,  
               GLint n_lados_transversales)
```

```
GLUquadric * gluNewQuadric (void)
```

```
void gluDeleteQuadric (GLUquadric * objeto)
```

```
void gluQuadricDrawStyle (GLUquadric * objeto, GLenum estilo) donde estilo es GLU_FILL, GLU_LINE,  
GLU_SILHOUETTE y GLU_POINT
```

```
...  
GLUquadricObj *qobj;  
..  
qobj = gluNewQuadric();  
gluQuadricDrawStyle(qobj, GLU_FILL);  
gluCylinder(qobj, 5.0, 5.0, 3.0, 14, 2);  
..
```



30

Introducción a OpenGL

3. Primitivas gráficas

- Un **vertex array** permite enviar el mayor número de información posible optimizando el tiempo de síntesis o **rendering**

```
void glEnableClientState (GLenum array) donde array puede ser, entre otras opciones  
GL_VERTEX_ARRAY, GL_COLOR_ARRAY, GL_NORMAL_ARRAY o  
GL_TEXTURE_COORD_ARRAY.
```

```
void glVertexPointer (GLint size, GLenum type, GLsizei stride, const GLvoid *pointer)  
size indica cuantos componentes tiene cada elemento del array, type indica el tipo de  
dato (GL_FLOAT o GL_DOUBLE), stride se usa para arrays que se solapen y pointer  
apunta al array de datos.
```

```
void glNormalPointer (...), void glTexCoordPointer (...)
```

```
void glColorPointer (...) como las funciones anteriores salvo que type es GL_FLOAT,  
GL_DOUBLE o GL_UNSIGNED_BYTE
```

```
void glDrawArrays (GLenum mode, GLint first, GLsizei count) permite dibujar el array de  
datos, donde mode es la primitiva de salida a mostrar (GL_POLYGON, GL_POINTS,  
GL_TRIANGLES, etc.), first es el índice inicial del array y count el número total de  
elementos a dibujar del array.
```



31

Introducción a OpenGL

3. Primitivas gráficas

- Ejemplo **vertex array**.

```
int n_puntos=5;
float *puntos, *colores;
puntos=(float *)malloc(3*n_puntos*sizeof(float));
colores=(float *)malloc(3*n_puntos*sizeof(float));
...
glPolygonMode (GL_FRONT_AND_BACK, GL_FILL);
puntos[0]=1.0; puntos[1]=1.0; puntos[2]=0.0;
puntos[3]=3.0; puntos[4]=1.0; puntos[5]=0.0;
puntos[6]=3.5; puntos[7]=2.4; puntos[8]=0.0;
puntos[9]=2.0; puntos[10]=3.5; puntos[11]=0.0;
puntos[12]=0.5; puntos[13]=2.4; puntos[14]=0.0;
colores[0]=0.0; colores[1]=0.0; colores[2]=1.0;
colores[3]=0.0; colores[4]=0.0; colores[5]=1.0;
colores[6]=0.5; colores[7]=0.0; colores[8]=1.0;
colores[9]=1.0; colores[10]=0.0; colores[11]=1.0;
colores[12]=0.5; colores[13]=0.0; colores[14]=1.0;

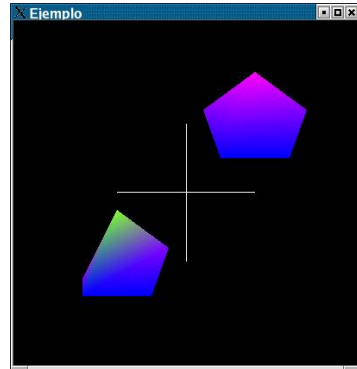
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, &(puntos[0]));

glEnableClientState(GL_COLOR_ARRAY);
glColorPointer(3, GL_FLOAT, 0, &(colores[0]));
```

```
glDrawArrays(GL_POLYGON, 0, n_puntos);

puntos[12]=1.0; puntos[13]=1.5; puntos[14]=0.0;
colores[9]=0.5; colores[10]=1.0; colores[11]=0.2;

glTranslatef(-4.0, -4.0, 0.0);
glDrawArrays(GL_POLYGON, 0, n_puntos);
...
```



OpenGL 32

Introducción a OpenGL

3. Primitivas gráficas

- Funciones de dibujado con **vertex array** y STL.

```
// Visualizando en modo puntos
void _puntos3D::draw_puntos(float r, float g, float b, int grosor)
{
    glPointSize(grosor);
    glColor3f(r,g,b);
    glBegin( GL_POINTS );
    for(int i=0 ; i < Vertices.size() ; i++)
        glVertex3f( vertices[i].x, vertices[i].y, vertices[i].z );
    glEnd() ;
}
```

```
// alternativa con vertex array
void _puntos3D::draw_puntos(float r, float g, float b, int grosor)
{
    glPointSize(grosor);
    glColor3f(r,g,b);
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, &vertices[0]);
    glDrawArrays(GL_POINTS, 0, vertices.size());
}
```

OpenGL 33

3. Primitivas gráficas

- Funciones de dibujo con **vertex array** y STL.

```
// Visualizando en modo línea
...
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_TRIANGLES);
for (int i=0; i<caras.size(); i++){
    glVertex3fv((GLfloat *) &vertices[caras[i]._0]);
    glVertex3fv((GLfloat *) &vertices[caras[i]._1]);
    glVertex3fv((GLfloat *) &vertices[caras[i]._2]);
}
glEnd();
...

// alternativa con vertex array
...
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, &vertices[0]);
➔ glDrawElements(GL_TRIANGLES, caras.size()*3, GL_UNSIGNED_INT, &caras[0]);
...
```



4. GLUT

- Librería **GLUT** (OpenGL Utility Toolkit)
 - Gestión de ventanas (única y múltiples)
 - Gestión de eventos (callbacks)
 - Gestión de menús pop-up
 - Objetos gráficos más complejos (cubo, cono, esfera, ...)
 - Fuentes bitmap
 - Doble buffer
- Versión **GLUT 3.7**
- <http://www.opengl.org/resources/libraries/glut/>



4. GLUT

- Creación de una ventana.

```
#include <GL/glut.h>
...
int main(int argc, char **argv)
{
    int W_x=50, W_y=50, W_width=400, W_high=400;

    // inicialización de glut
    glutInit(&argc, argv);

    // se indican las características que se desean para la visualización con OpenGL
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);

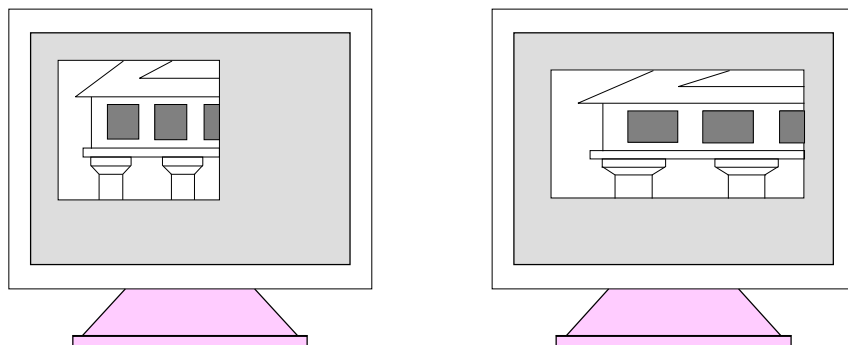
    // posición de la esquina inferior izquierda de la ventana
    glutInitWindowPosition(W_x, W_y);

    // tamaño de la ventana (ancho y alto)
    glutInitWindowSize(W_width, W_high);

    // llamada para crear la ventana, indicando el título
    glutCreateWindow("IG: Practicas");
    ...
}
```

4. GLUT (creación de una ventana)

- Tamaño de una ventana con `glutInitWindowSize`: cuadrada y no cuadrada



Introducción a OpenGL

4. GLUT (creación de una ventana)

- La función `glutInitDisplayMode` permite fijar las características que se desean para la visualización con OpenGL
 - GLUT_SIMPLE: sólo la memoria de video
 - GLUT_DOUBLE: memoria de video y memoria virtual
 - 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 (plantilla para recortar)
- Se combinan mediante OR,
`glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);`



Introducción a OpenGL

4. GLUT (creación de una ventana)

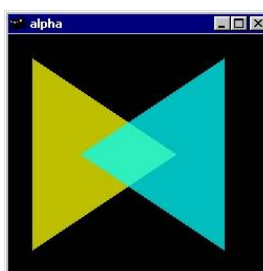
- Ejemplos de `glutInitDisplayMode`



Sombras con estarcido



Sin activar z-buffer



Transparencia



4. GLUT

- Gestión de eventos. Se implementan mediante Callbacks
- Tipos de eventos:
 - Redibujado de la ventana:
`glutDisplayFunc(Función)`
 - Cambio de tamaño de la ventana:
`glutReshapeFunc(Función)`
 - Teclado:
`glutKeyboardFunc(Función)` cuando se ha pulsado una tecla normal
`glutSpecialFunc(Función)` para cuando se ha pulsado una tecla especial
`glutKeyboardUpFunc(Función)` y `glutSpecialUpFunc(Función)`

4. GLUT (eventos)

- Tipos de eventos (continuación):
 - Ratón:
`glutMouseFunc(Función)` para cuando se ha pulsado o soltado una tecla del ratón
`glutMotionFunc(Función)` para cuando se mueve el ratón pulsando una tecla
`glutPassiveMotionFunc(Función)`: función para cuando se mueve el ratón sin pulsar
`glutEntryFunc(Función)`: función para detectar cuando el cursor entra o sale de una ventana
 - Sistema:
`glutIdleFunc(Función)` GLUT desocupado. No hay eventos
`glutTimerFunc(Función)` para cuando un temporizador llega al final de la cuenta

Introducción a OpenGL

4. GLUT (eventos)

- Código para la gestión de eventos:

```
....
int main(int argc, char **argv)
{
    int W_x=50, W_y=50, W_width=400, W_high=400;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowPosition(W_x,W_y);
    glutInitWindowSize(W_width,W_high);
    glutCreateWindow("IG: Practicas");

    glutDisplayFunc(dibujar); // asignación de la función "dibujar" para evento redibujado

    glutReshapeFunc(tam_ventana); // asignación de la función "tam_ventana" al evento correspondiente

    glutKeyboardFunc(tecla_normal); // asignación de la función "tecla normal" al evento correspondiente

    initialize(); // aquí se puede crear el modelo
    // inicio del bucle de eventos
    glutMainLoop();
    return 0;
}
```



Introducción a OpenGL

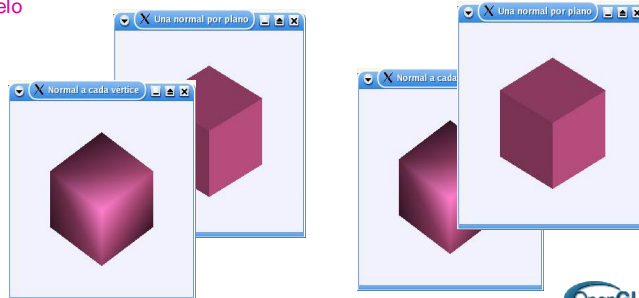
4. GLUT (eventos)

- Código para la gestión de eventos:

```
....
void dibujar(void)
{
    glClearColor(0.0,1.0,0.0,0.0); // fija el color de borrado (color de fondo)

    // borra la memoria de video y de profundidad
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    ....
    // Aquí se dibuja el modelo
    ....
    glFlush();
}
```

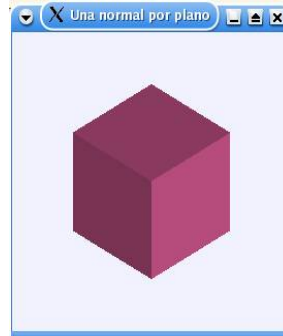
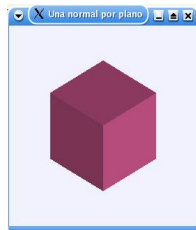


Introducción a OpenGL

4. GLUT (eventos)

- Código para la gestión de eventos:

```
...  
void tam_ventana(int Ancho, int Alto)  
{  
    float Size_y;  
    Size_y=(float) Alto/(float) Ancho;  
    glOrtho (-1.0, 1.0,-Size_y, Size_y,10.0,-10.0);  
    // inicialmente Ancho y Alto vienen fijados por la llamada a la función glutInitWindowSize()  
    glViewport(0,0,Ancho,Alto);  
    glutPostRedisplay();  
}
```



44

Introducción a OpenGL

4. GLUT (eventos)

- Código para la gestión de eventos:

```
...  
void tecla_normal (unsigned char Tecla, int x, int y)  
{  
    switch (toupper(Tecla)){  
        case 'Q': exit(0);  
            break;  
        case 27: exit(0);  
            break;  
        case 'otra tecla':  
            hacer algo  
            puede ser necesario llamar a "glutPostRedisplay()"  
    }  
}
```

45

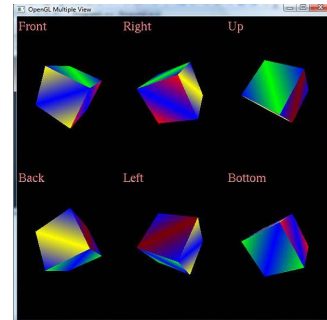
Introducción a OpenGL

4. GLUT

- Letras bitmap con Glut.

```
void glutBitmapCharacter(GLUTenum fuente, char texto)
```

```
char text[10]="...";  
...  
glRasterPosition2i (x, y);  
for (k = 0; k < 10; k++)  
    glutBitmapCharacter (GLUT_BITMAP_ROMAN_24, text [k]);  
....  
  
fuentes  
GLUT_BITMAP_8_BY_13, GLUT_BITMAP_9_BY_15,  
GLUT_BITMAP_TIMES_ROMAN_10, GLUT_BITMAP_HELVETICA_10
```

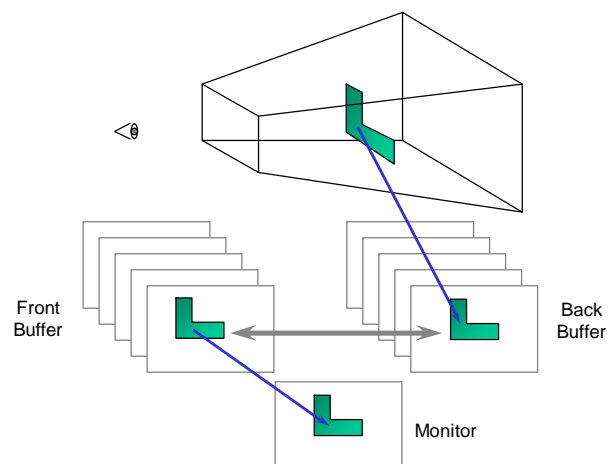


Introducción a OpenGL

4. GLUT.

- Uso del doble buffer.

```
glutSwapBuffers(); en lugar de glFlush()
```



5. Transformaciones geométricas

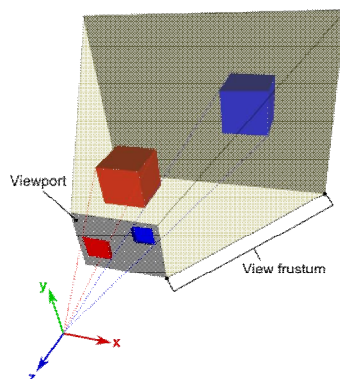
- En OpenGL las matrices de transformación 4x4 como se definen como un vector de 16 elementos (vértices como vectores columna):

$$M = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix}$$

- Las matrices se guardan en pilas:
 - **GL_PROJECTION** pila para la transformaciones de proyección y vista
 - **GL_MODELVIEW** pila para las transformaciones de modelado

5. Transformaciones geométricas

- Orden de aplicación de las transformaciones
Objeto → pila **GL_ModelView** → pila **GL_PROJECTION** → Imagen



Con `glMatrixMode()` se indica la pila a usar(**GL_PROJECTION** como argumento o bien **GL_MODELVIEW**)

5. Transformaciones geométricas

- Funciones de transformación generales.

`void glLoadIdentity (void)` inicialización de la matriz de transformación.

`void glLoadMatrix{fd} (TYPE *m)` construcción de una matriz 4x4 cualquiera.

```
float m[16]={12.0, .....};  
.....  
glLoadMatrixf (m);
```

`void glMultMatrix{fd} (TYPE *m)` compone una matriz con la matriz actual. Si **M** es la matriz actual y **C** es la que se crea con `glMultMatrix`, la matriz resultante de la composición es **MC**.

5. Transformaciones geométricas

- Funciones de transformación generales.

`void glLoadTransposeMatrix{fd} (TYPE *m)` construcción de una matriz 4x4 cualquiera.

```
float m[16]={12.0, .....};  
.....  
glLoadTransposeMatrixf (m);
```

$$M = \begin{pmatrix} m_1 & m_2 & m_3 & m_4 \\ m_5 & m_6 & m_7 & m_8 \\ m_9 & m_{10} & m_{11} & m_{12} \\ m_{13} & m_{14} & m_{15} & m_{16} \end{pmatrix}$$

`void glMultTransposeMatrix{fd} (TYPE *m)` compone la transpuesta de una matriz con la matriz actual. Si **M** es la matriz actual y **C** es la que se crea con `glMultTransposeMatrix`, la matriz resultante de la composición es **MCT**.

5. Transformaciones geométricas

- Funciones de transformaciones geométricas.

`void glTranslate(f,d) (TYPE x, TYPE y, TYPE z)` traslación.

`void glRotate(f,d) (TYPE angulo, TYPE x, TYPE y, TYPE z)` rotación.

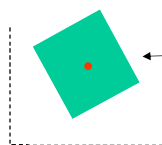
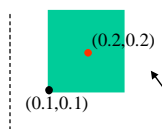
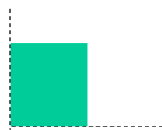
`glRotatef (45.0, 0, 0, 1)` rotación en z, equivalente a una rotación bidimensional.

`void glScale(f,d) (TYPE x, TYPE y, TYPE z)` escalado.

El uso sucesivo de dos o más transformaciones hace que éstas se concatenen.

5. Transformaciones geométricas

- Ejemplo



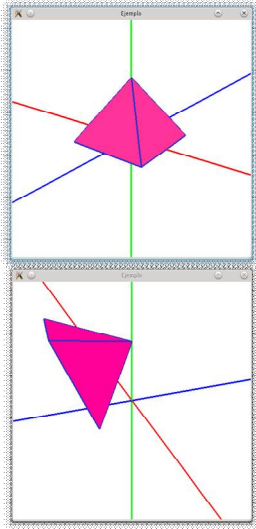
```
void Objeto ( )
{glPolygonMode(GL_FRONT_AND_BACK,
GL_FILL);
glBegin( GL_POLYGON );
glColor3f (0.0, 0.8, 0.6);
glVertex2f( 0.0, 0.0); glVertex2f( 0.2, 0.0);
glVertex2f( 0.2, 0.2); glVertex2f( 0.0, 0.2);
glEnd();
}
```

```
....
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
```

```
glTranslatef(0.2,0.2,0.0);
glRotatef(39.0,0,0,1);
glTranslatef(-0.2,-0.2,0.0);
glTranslatef(0.1,0.1,0.0);
Objeto( );
....
```

5. Transformaciones geométricas

- Ejemplo



```
void visualizar_meshVT (meshVT *malla, float, r, float g, float b,  
                        GLenum modo)
```

```
{  
...  
}
```

```
....
```

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity( );
```

```
....
```

```
glTranslatef(0.0,0.75,0.0); // 0.75 es la altura de la pirámide
```

```
glRotatef(-70,1,0,0);
```

```
glTranslatef(0.0,-0.75,0.0);
```

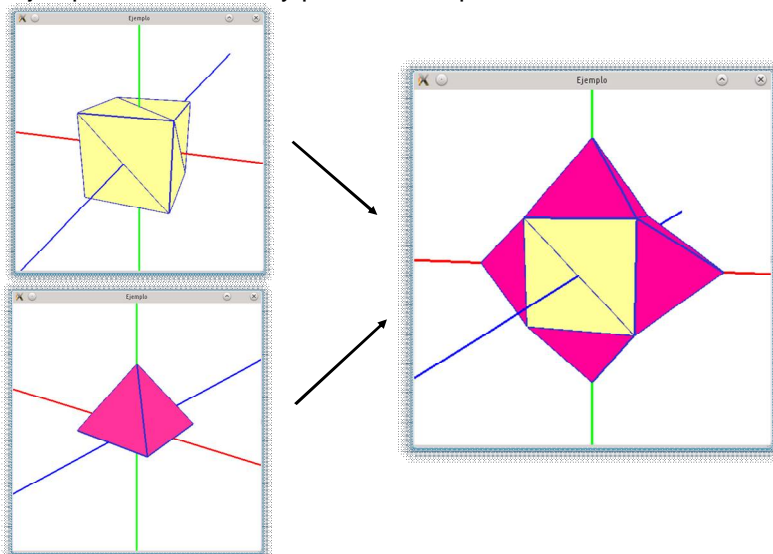
```
visualizar_meshVT (piramide, 0.2, 0.2, 0.8, GL_LINE);
```

```
visualizar_meshVT (piramide, 1.0, 0.0, 0.6, GL_FILL);
```

```
...
```

5. Transformaciones geométricas

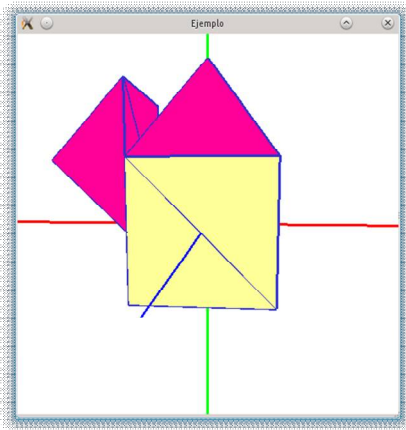
- Ejemplo. Con un cubo y pirámides se pretende crear una estrella



Introducción a OpenGL

5. Transformaciones geométricas

- Ejemplo. Apilamiento erróneo de las transformaciones geométricas



```
....  
// 1 es el tamaño del cubo  
visualizar_meshVT(cubo,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(cubo,1.0,1.0,0.6,GL_FILL);  
  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
....
```

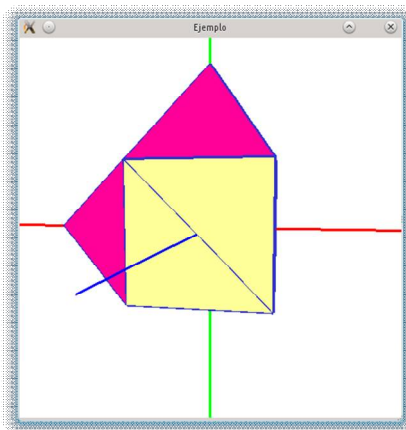


56

Introducción a OpenGL

5. Transformaciones geométricas

- Ejemplo. Apilamiento correcto de las transformaciones geométricas



```
....  
// 0.5 es el tamaño del cubo  
visualizar_meshVT(cubo,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(cubo,1.0,1.0,0.6,GL_FILL);  
  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0); ←  
  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
....
```

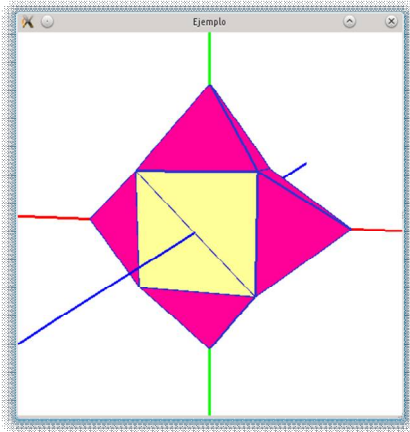


57

Introducción a OpenGL

5. Transformaciones geométricas

- Ejemplo. Apilamiento correcto de las transformaciones geométricas



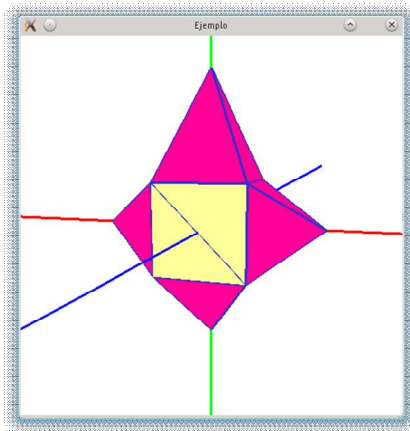
```
....  
// 0.5 es el tamaño del cubo  
visualizar_meshVT(cubo,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(cubo,1.0,1.0,0.6,GL_FILL);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide,1.0,0.0,0.6,GL_FILL);  
...
```

58

Introducción a OpenGL

5. Transformaciones geométricas

- Ejemplo. Referencia y copia

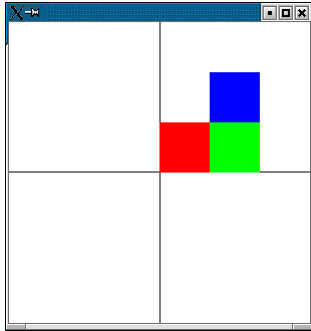


```
....  
// 0.5 es el tamaño del cubo  
visualizar_meshVT(cubo,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(cubo,1.0,1.0,0.6,GL_FILL);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide1,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide1,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide2,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide2,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide3,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide3,1.0,0.0,0.6,GL_FILL);  
glTranslatef(0.0,-0.5,0.0);  
glRotatef(90,0,0,1);  
glTranslatef(0.0,0.5,0.0);  
visualizar_meshVT(piramide4,0.2,0.2,0.8,GL_LINE);  
visualizar_meshVT(piramide4,1.0,0.0,0.6,GL_FILL);  
...
```

59

5. Transformaciones geométricas

- Uso de `glPushMatrix()` y `glPopMatrix()`. Ejemplo



```
void Pieza ( )
{glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
 glBegin( GL_POLYGON );
 ....
 glEnd( );
 }
 ....
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity( );
 ...

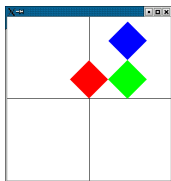
 glColor3f(1.0,0.0,0.0);
 Pieza();

 glColor3f(0.0,1.0,0.0);
 glTranslatef(1.0,0.0,0.0);
 Pieza

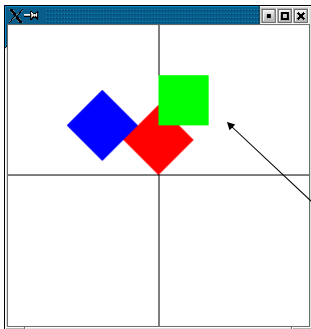
 glColor3f(0.0,0.0,1.0);
 glTranslatef(0.0,1.0,0.0);
 Pieza();
```

5. Transformaciones geométricas

- Uso de `glPushMatrix()` and `glPopMatrix()`. Ejemplo



Lo que queremos ver



Lo que vemos

```
void Pieza ( )
{glRotatef(45.0,0.0,0.0,1.0); ←
 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
 glBegin( GL_POLYGON );
 ....
 glEnd( );
 }
 ....
 glColor3f(1.0,0.0,0.0);
 Pieza();

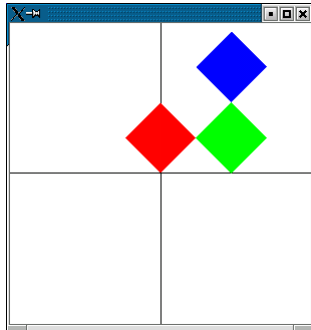
 glColor3f(0.0,1.0,0.0);
 glTranslatef(sqrt(2.0),0.0,0.0);
 Pieza();

 glColor3f(0.0,0.0,1.0);
 glTranslatef(0.0,sqrt(2.0),0.0);
 Pieza();
```

Introducción a OpenGL

5. Transformaciones geométricas

- Uso de `glPushMatrix()` y `glPopMatrix()`. Ejemplo



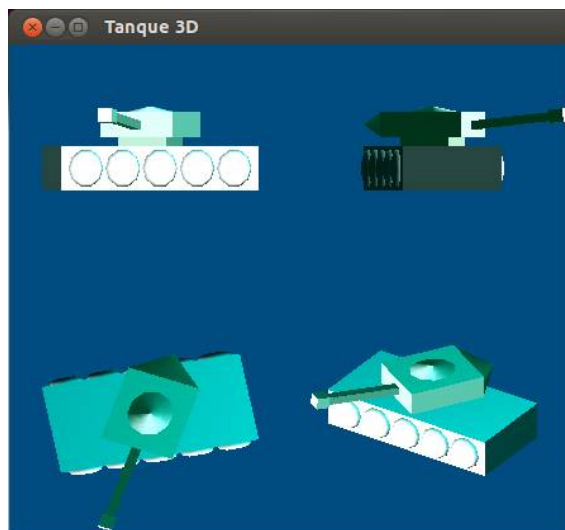
```
void Pieza ( )  
{glPushMatrix( ); ←  
  glRotatef(45.0,0.0,0.0,1.0);  
  glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);  
  glBegin( GL_POLYGON );  
  ....  
  glEnd( );  
  glPopMatrix( ); ←  
}  
....  
glColor3f(1.0,0.0,0.0);  
Pieza();  
  
glColor3f(0.0,1.0,0.0);  
glTranslatef(sqrt(2.0),0.0,0.0);  
Pieza();  
  
glColor3f(0.0,0.0,1.0);  
glTranslatef(0.0,sqrt(2.0),0.0);  
Pieza();
```



Introducción a OpenGL

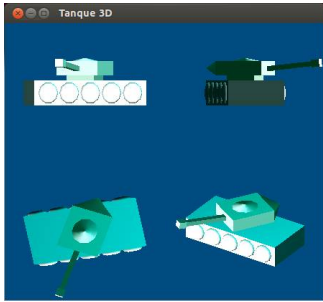
5. Transformaciones geométricas

- Ejemplo de modelo jerárquico



5. Transformaciones geométricas

- Ejemplo de modelo jerárquico

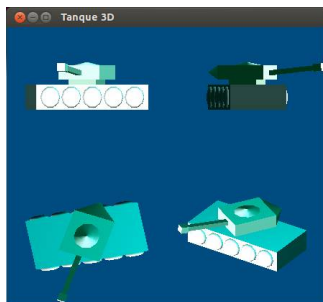


// En este ejemplo solo se puede visualizar un tanque

```
....
visualizar_tanque(float tx, float ty, float tz, float an1, float an2 )
{
    glTranslatef(tx,ty,tz);
    visualizar_chasis();
    glTranslatef(...);           // Traslación para situar la torreta
    glRotatef(an1,0,1,0);        // Rotación de la torreta
    visualizar_torreta();
    glTranslatef(...);           // Traslación para situar el cañón
    glRotatef(an2,0,0,1);        // Rotación del la cañón
    visualizar_canon();
}
```

5. Transformaciones geométricas

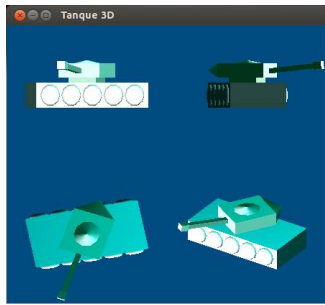
- Ejemplo de modelo jerárquico



```
visualizar_canon()
{
    glPushMatrix();
    glTranslatef(...);
    glScalef(...);               // Bocacha del cañón
    visualizar_meshVT(cubo,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(...);
    glScalef(...);               // Caña del cañón
    visualizar_meshVT(cubo,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
}
```


5. Transformaciones geométricas

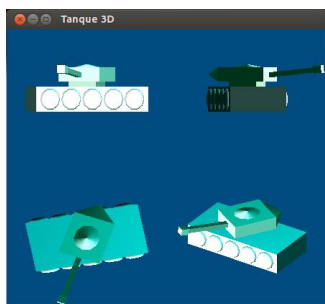
- Ejemplo de modelo jerárquico



```
visualizar_torreta()
{
    glPushMatrix();
    // Anillo
    glPushMatrix();
    glTranslatef(...);
    glScalef(...);
    visualizar_meshVT(cubo,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
    // Parte superior Torreta
    glTranslatef(...);
    glPushMatrix();
    glScalef(...);
    visualizar_meshVT(cubo,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(...);
    glRotatef(...)
    glScalef(...);
    visualizar_meshVT(piramide,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
    glPopMatrix();
}
```

5. Transformaciones geométricas

- Ejemplo de modelo jerárquico



```
visualizar_rodamiento()
{
    glPushMatrix();
    glTranslatef(...);
    glRotatef(...)
    glScalef(...);
    visualizar_meshVT(cilindro,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
}

visualizar_chasis()
{
    glPushMatrix();
    glScalef(...);
    visualizar_meshVT(cubo,0.1,0.2,0.9,GL_FILL);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(...);
    visualizar_rodamiento();
    glPopMatrix();
    ... // cuatro llamadas más a visualizar_rodamiento
}
```

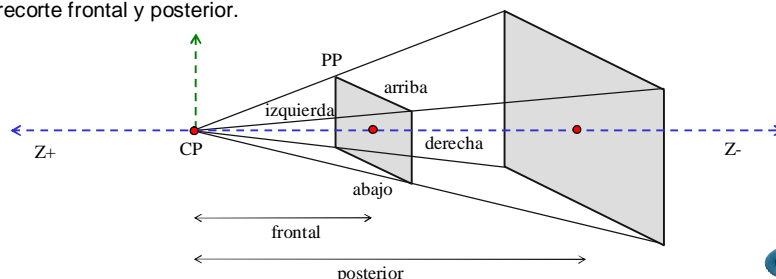
6. Visualización

- Antes de fijar la transformación de vista se usa la función `glMatrixMode()` con el argumento `GL_PROJECTION` para conmutar a la pila de matrices para realizar la proyección a un espacio 2D.
- Se tiene que especificar un `viewport` para indicar el rectángulo del dispositivo para mostrar la imagen
- `void glViewport (GLint x, GLint y, GLint width, GLint height)` el `viewport` se especifica en coordenadas de dispositivo (OpenGL no usa coordenadas de dispositivo normalizadas), con el origen en la esquina inferior izquierda del dispositivo
 - `x` e `y` representan el origen del `viewport`; `width`, `height` son el tamaño del mismo
- Los objetos tridimensionales se pueden proyectar en perspectiva y ortográficamente.

6. Visualización

- Proyección en perspectiva.
 - Un solo punto de fuga en el eje Z-
 - Centro de proyección en el origen.
 - El plano de recorte frontal y el plano de proyección coinciden

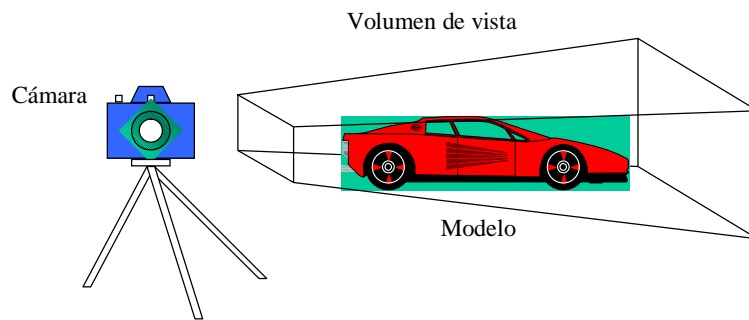
`void glFrustum (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)` define un volumen de vista simétrico en perspectiva. Los parámetros `left`, `right`, `bottom` y `top` se expresan en coordenadas mundiales y corresponden a la ventana del mundo en el plano de proyección. `Near` y `far` se expresan en coordenadas de vista con valores positivos y definen la situación de los planos recorte frontal y posterior.



Introducción a OpenGL

6. Visualización

- Atención a donde se colocan los objetos y el volumen de vista.



Introducción a OpenGL

6. Visualización

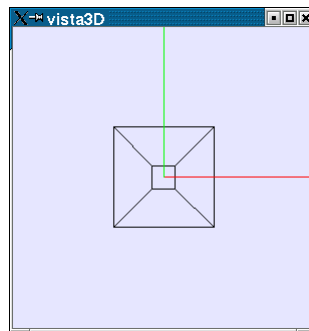
- Ejemplo 1.

....

```
glMatrixMode (GL_PROJECTION);  
glLoadIdentity ( );  
glViewport (0, 0, width, height);  
glFrustrum (-1.0, 1.0, -1.0, 1.0, 0.2, 40.0 );  
glTranslatef(0.0, 0.0, -0.8); ←
```

....

```
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity ( );  
  
ejes ( );  
visualizar_meshVT(cubo,0.0,0.0,0.0,0.0,GL_LINE);  
  
glFlush( );
```



6. Visualización

- Ejemplo 2. (viendo dos puntos de fuga)

....

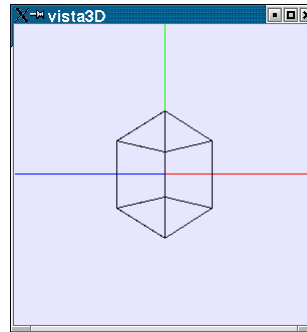
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, width, height);
glFrustrum(-0.3, 0.3, -0.3, 0.3, 0.2, 40.0);
glTranslate(0.0, 0.0, -1.5);
glRotatef(-45.0, 0.0, 1.0, 0.0);
```

....

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

ejes();
visualizar_meshVT(cubo,0.0,0.0,0.0,0.0,GL_LINE);

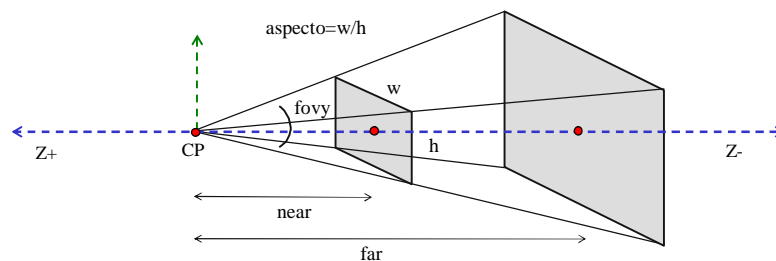
glFlush();
```



6. Visualización

- Proyección en perspectiva (librería glu).

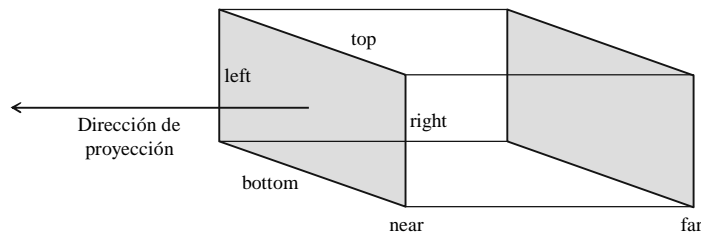
`void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far)` crea también un volumen de vista pero especificando el ángulo del campo de visión en el plano X-Y y la proporción entre el ancho y el alto ($\text{aspect} = w/h$).



6. Visualización

- Proyección ortográfica.

`void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)` delimita un volumen de vista para una proyección ortográfica en alzado, no centrada. `left`, `right`, `bottom`, `top`, `near` y `far` definen los planos de recorte de dicho volumen. La dirección de proyección es paralela al eje `Z`.



6. Visualización

Ejemplo (varias vistas y viewports).

```

.....
glMatrixMode(GL_PROJECTION); // perfil
glLoadIdentity(...);
glViewport(ancho/2,alto/2,ancho/2,alto/2);
glOrtho(-5.0,5.0,-5.0,5.0,-15.0,15.0);
glRotatef(90.0,0.0,1.0,0.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
visualizar_tanque();

glMatrixMode(GL_PROJECTION); // perspectiva
glLoadIdentity();
glViewport(ancho/2,0,ancho/2,alto/2);
glFrustum (-1.0, 1.0, -1.0, 0.5,2.0, 80.0);
glTranslatef(0.0, 0.0, -2.0);
glRotatef(-35.0, 1, 0, 0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
visualizar_tanque(...);
.....
                    
```

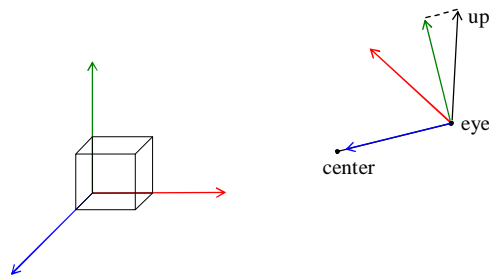
Introducción a OpenGL

6. Visualización

- Cambiando la posición de la cámara (librería glu).

`void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz)`

se puede cambiar el punto de vista o del observador, el punto de línea mira y la orientación con esta función. El punto de vista viene dado por `eye`; el punto de mira por `center`; y el vector hacia arriba por `up`. También `gluLookAt` se puede usar con una proyección ortográfica.



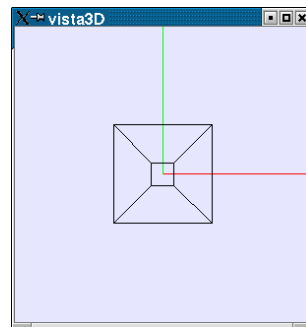
Introducción a OpenGL

6. Visualización

- Ejemplo 1.

```
....
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
glViewport (0, 0, width, height);
glFrustum (-1.0, 1.0, -1.0, 1.0, 0.2, 40.0 );
....

glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );
gluLookAt(0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); ←
...
ejes( );
visualizar_meshVT(cubo,0.0,0.0,0.0,GL_LINE);
...
glFlush( );
```

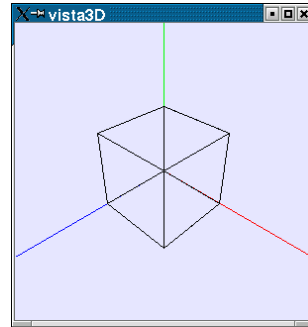


6. Visualización

- Ejemplo 2.

```
....
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
glViewport (0, 0, width, height);
glFrustrum (-0.1, 0.1, -0.1, 0.1, 0.2, 40.0 );
....

glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );
gluLookAt(2.0, 2.0, 2.0, 0.0, 0.0, 0.0, 1.0, 0.0);
...
ejes( );
visualizar_meshVT(cubo,0.0,0.0,0.0,GL_LINE);
...
glFlush( );
```

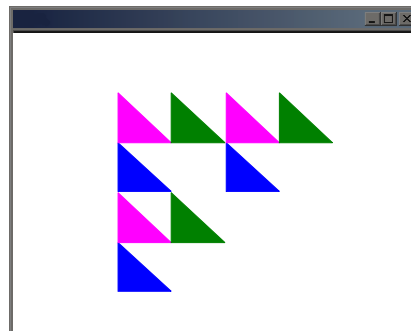


6. Visualización

- Para objetos en 2D, podemos visualizarlos usando una proyección ortográfica.
- Los parámetros **near** y **far** se fijan a -1 y 1 respectivamente

```
....
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
glViewport (0, 0, w, h);
glOrtho (-5.0, 5.0, -5.0, 5.0, -1.0, 1.0 );
...

glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );
dibujar_modelo_2D ( );
```



6. Visualización

- Vista 2D con GLU

`void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)`
`left`, `right`, `bottom` y `top` hacen las veces de `wxmin`, `wxmax`, `wymin` y `wymax` respectivamente.

```
....  
glMatrixMode (GL_PROJECTION);  
glLoadIdentity ( );  
glViewport (a, b, c, d);  
gluOrtho2D (-5.0, 5.0, -5.0, 5.0 );  
...  
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity ( );  
dibujar_modelo( );  
...
```

