OpenGL como máquina de estados

Así, no será lo mismo dibujar un triángulo y activar una textura, que activar una textura y dibujar un triángulo... en OpenGL, el orden de las acciones resulta crítico en la mayoría de las ocasiones... de igual manera, no será lo mismo trasladar y rotar algo, que rotarlo y trasladarlo, como veremos más adelante, en lo referente a las transformaciones...

De este modo, de manera muy abstracta, la manera de dibujar algo en OpenGL suele ser la siguiente:

1. Activar todas las opciones que van a ser persistentes a la escena (ponemos la cámara, activamos la iluminación global ...)

2. Activar las opciones que establecen el estado de un objeto específico (su posición en el espacio, su textura ...)

3. Dibujar el objeto.

4. Desactivar las opciones propias de ese objeto (volver a la posición original, desactivar su textura)

5. Volver al punto 2 hasta haber dibujado todos los objetos.

Esto, evidentemente, es un esquema sencillo... como se verá más adelante, estas operaciones pueden agruparse en jerarquías, lo que proporciona una gran potencia y flexibilidad a la hora de programar.

El espacio 3D

Si bien es cierto que OpenGL proporciona acceso a funciones de dibujado 2D, en este curso nos vamos a centrar en el espacio 3D... OpenGL trabaja, a grandes rasgos, en un espacio de tres dimensiones, aunque veremos que realmente, trabaja con coordenadas homogéneas (de cuatro dimensiones).

Las tres dimensiones que nos interesan ahora son las especificadas por un sistema 3D ortonormal. Es decir, sus ejes son perpendiculares, y cada unidad en uno de ellos está representada por un vector de módulo 1 (si nos alejamos una unidad, nos alejamos la misma distancia del eje de coordenadas, da igual la dirección).

La cuarta coordenada se utiliza entre otras razones, para representar la perspectiva, pero no nos meteremos con ello en este momento... de este modo, el sistema de coordenadas inicial de un sistema OpenGL puede representarse con esta matriz:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Finalmente, es recomendable desempolvar nuestros algebraicos básicos (vectores, normales, etc.), porque como veremos, nos van a resultar de gran utilidad a la hora de programar en 3D...

La situación de los ejes de coordenadas se refleja en la matriz de transformación. Esta matriz representa la transformación que se aplicará a todos los vértices que se dibujen mientras ella esté activa.

Como se ha comentado, OpenGL es una máquina de estados: activamos y desactivamos opciones que afectan al dibujado. Habitualmente, las opciones se activan y desactivan con

glEnable(<OPTION>) y
glDisable(<OPTION>).

Las matrices y OpenGL Anteriormente hemos visto que OpenGL guarda la transformación de los objetos en una matriz. A esta matriz se le denomina matriz de visualización/modelado, porque se emplea para estas dos funciones.

Al realizar operaciones que modifiquen alguna de estas dos matrices, tendremos que cambiar el "modo de matriz", para que las operaciones afecten a la matriz que nos interesa.

Para ello utilizaremos las funciones

glMatrixMode(GL_MODELVIEW) o
glMatrixMode(GL_PROJECTION).

Además, existen dos funciones que permiten guardar y restaurar los valores de la matriz activa en una pila.

La función

glPushMatrix()	guarda una matriz en la cima de la pila, y
glPopMatrix()	la saca, y la restaura. Esto lo podemos utilizar para dibujar un objeto y, antes de dibujar el siguiente, restauramos la transformación inicial.

Dibujado en OpenGL

Para dibujar en OpenGL, tenemos que habilitar el modo de dibujado, establecer las opciones de dibujado de cada vértice, y dibujar cada uno de ellos. Al terminar de dibujar una figura, finalizamos el modo de dibujado.

Para comenzar a dibujar, utilizaremos el comando

glBegin(<MODO_DE_DIBUJADO>),

dónde el modo de dibujado vendrá dado por una constante:

Parámetro	**Descripción**
GL_POINTS	Se dibujan vertices separados
GL_LINES	Cada par de vértices se interpreta como una línea
GL_POLYGON	Los vértices describen el contorno de un polígono
GL_TRIANGLES	Cada triplete de vértices de interpreta como un triángulo
GL_QUADS	Cada cuarteto de vértices se interpreta como un cuadrilátero
GL_LINE_STRIP	Líneas conectadas
GL_LINE_LOOP	Líneas conectadas, con unión entre el primer y último vértice
GL_TRIANGLE_STRIP	Se dibuja un triángulo, y cada nuevo vértice se interpreta con un triángulo entre los dos anteriores vértices y el nuevo
GL_TRIANGLE_FAN	Se dibujan triángulos con un vértice común
GL_QUAD_STRIP	Igual que el TRIANGLE_STRIP, con cuadriláteros

Orientación. Un polígono tiene dos caras, delantera y trasera. La manera de saber qué cara es la delantera, y cual la trasera, es que, si miramos la delantera, los vértices se habrán dibujado en orden antihorario.
Por ejemplo, tenemos la siguiente figura:

1 2

3 4

Si dibujamos los vertices en el orden 1,3,4,2, estaremos dibujando la cara delantera mirando hacia nosotros, pero si el orden es, por ejemplo, 1,2,4,3, estaremos mirando la cara trasera del polígono.

El programa de “Hola Mundo”.

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

void display(void);

int main(int argc, char *argv[])
{
	glutInit(&argc, argv);			// Inicializa la libreria auxiliar GLUT
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);		// Inicializa el modo de visualizacion
	glutInitWindowPosition(20,20);
	glutInitWindowSize(500,500);
	glutCreateWindow(argv[0]);
	glutDisplayFunc(display);
	glutMainLoop();				//Comienza el bucle de dibujo y proceso de eventos.
	return 0;
}

void display(void)
{
	glClearColor(0.0,0.0,0.0,0.0);			// Color de fondo: negro
	glClear(GL_COLOR_BUFFER_BIT);			// Boramos la pantalla
	glMatrixMode(GL_PROJECTION);		// Modo proyección
	glLoadIdentity();				//Cargamos la matriz identidad
	glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);	// Proyección ortográfica, dentro del cubo señalado
	glMatrixMode(GL_MODELVIEW);		// Modo de modelado
	glBegin(GL_TRIANGLES);			// Dibujamos un triángulo
	glColor3f(1.0,0.0,0.0);			// Color del primer vértice: rojo
	glVertex3f(0.0,0.8,0.0);			// Coordenadas del primer vértice
	glColor3f(0.0,1.0,0.0);			// Color del segundo vértice: verde
	glVertex3f(-0.6,-0.2,0.0);			// Coordenadas del  segundo vértice
	glColor3f(0.0,0.0,1.0);			// Color del tercer vértice: azúl
	glVertex3f(0.6,-0.2,0.0);			// Coordenadas del  tercer vértice
	glEnd();				// Terminamos de dibujar
	glFlush();				// Forzamos el dibujado
	sleep(10);				// Esperamos 10 segundos
	exit(0);				// Salimos del programa
}
```

b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Command Suffixes and Argument Data Types