Lenguaje de Programación: C++ GLUT

José Luis Alonzo Velázquez

Universidad de Guanajuato

Noviembre 2010

GLUT

Se trata de una librería diseñada para facilitar el manejo de ventanas y eventos en aplicaciones que pretenden mostrar una escena mediante OpenGL. GLUT (The OpenGL Utility Toolkit) es una interface de programa disponible en ANSI C y FORTRAN, pensado para permitir la portabilidad de las aplicaciones desarrolladas sobre GLUT a distintos sistemas operativos.

Nota curiosa

GLUT permite la entrada de dispositivos como joystick u otros dispositivos analógicos de una manera sencilla de manejar, otra funcionalidad que aporta es la posibilidad de cambiar el modo de pantalla y disponer de la ventana a pantalla completa.

GLUT

La librería soporta las siguientes funcionalidades:

- Múltiples ventanas de renderización OpenGL
- Proceso de eventos mediante llamadas callback(display e idle)
- Dispositivos de entrada sofisticados
- Control de timers y rutina idle
- Permite un sencillo sistema de menús desplegables
- Rutinas para generar varios objetos sólidos y alámbricos
- Soporte para renderizar fuentes mediante trazados o bitmaps
- Funciones para el manejo de ventanas

Tipos de datos OpenGL

OpenGL para permitir una mayor portabilidad entre plataformas define sus propios tipos de datos, eso si, haciendo una correspondencia con los tipos ya existentes en C:

Tipo en OpenGL	Representación interna	Definición tipo C	Sufijo Literal
GLbyte	Entero de 8 bits	char	b
GLshort	Entero de 16 bits	short	s
GLint, GLsizei	Entero de 32 bits	long	l
GLfloat, GLclampf	Coma flotante de 32 bits	float	f
GLdouble, GLclampd	Coma flotante de 64 bits	double	d
GLubyte, GLboolean	Entero de 8 bits sin signo	unsigned char	ub
GLushort	Entero de 16 bits sin signo	unsigned short	us
GLuint, GLenum, GLbitfiled	Entero de 32 bits sin signo	unsigned long	ul

Para realizar la denominación de funciones OpenGL sigue el siguiente esquema para todas las funciones, la funcionalidad que ofrece el procedimiento o función viene indicado por la raíz:<Librería><Comando Raíz><Cuenta Opcional de Argumentos><Tipo de Argumentos Opcional>

Ejemplo

Un ejemplo es la función para indicar el color a utilizar, en este caso lo indicamos con tres componentes (red, green, blue) en formato de floats : glColor3f(...)

- gl - Librería
- Color - Comando Raíz
- 3 - Número de argumentos
- f - Tipo de argumento

Borrando pantalla y renderización

Generalmente a la hora de generar una imagen el primer paso que se debe dar es borrar el contenido actual de la pantalla para pasar después a dibujar la escena. Para borrar la pantalla, o mejor dicho, el buffer de salida disponemos de la función void glClear(GLbitfield mask) que le pasamos como parámetro que buffer deseamos borrar, en nuestro caso la llamada será:

```
glClear(GL_COLOR_BUFFER_BIT);
```

Otros Buffers

Existe la posibilidad de borrar otros buffers, OpenGL dispone de cuatro buffers:

Buffer	Parámetro	Almacena...
Buffer de color	GL_COlOR_BUFFER_BIT	la imagen generada
Buffer de profundidad	GL_DEPTH_BUFFER_BIT	la profundidad a que se halla cada pixel
Buffer de acumulación	GL_ACCUM_BUFFER_BIT	varias imágenes acumuladas
Buffer de plantilla	GL_STENCIL_BUFFER_BIT	una máscara para restringir la zona de dibujado

Podemos especificar con que color deseamos que borre el buffer de color, esto lo haremos con la instrucción glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha), por defecto es (0, 0, 0, 0) o sea negro.

Para los otros bufferes disponemos de funciones similares: glClearDepth, glClearAccum y glClearStencil.

Detalles al terminar la escena

Una vez hayamos dibujado la escena, aplicando todo lo necesario debemos indicarle a OpenGL que es hora de mostrar el resultado, para ello tenemos dos funciones:

- void glFlush(void): Fuerza a OpenGL a ejecutar los comandos que le hemos dado, esto nos garantiza que se realizará y completará en un tiempo finito.
- void glFinish(void): Fuerza a OpenGL a ejecutar los comandos que le hemos dado, este comando pero retorna cuando se han completado todas las operaciones, lo que diferencia de la anterior función.

Nos aseguramos así que ya han sido realizados los comandos OpenGL.

Dibujo

En OpenGL disponemos de diez funciones básicas de dibujo, estas funciones las podemos clasificar en tres grupos, según dibujen puntos, líneas o superficies. Para dibujar una función primero indicamos que función deseamos dibujar con la instrucción void glBegin (GLenum mode) a continuación vamos especificando las coordenadas 3D que forman la primitiva. Y por último indicamos que hemos terminado de definir los vértices. A cada vértice se le pueden dar unas propiedades que veremos posteriormente como son el color, la orientación y la coordenada de la textura si empleamos texturas.

Ejemplo

```
glBegin(GL_POINTS);
glVertex2f(0.0, 0.0);
glVertex2f(0.0, 1.0);
glVertex2f(1.0, 0.2);
glVertex2f(0.5, 0.1);
glVertex2f(-0.2, 1.4);
glEnd();
```

Funciones Básicas

En la instrucción glBegin especificamos que función deseamos, las diferentes posibilidades son:

Valor	Primitiva
GL_POINTS	Puntos individuales
GL_LINES	Cada par de vértices se interpreta como una línea
GL_POLYGON	Un polígono convexo
GL_TRIANGLES	Tripletas de vértices que forman triángulos
GL_QUADS	Cuádruples de vértices formando cuadrilateros
GL_LINE_STRIP	Series de líneas conectadas
GL_LINE_LOOP	Mismo que el anterior, pero el último y el primer vértice cierran en bucle
GL_TRIANGLE_STRIP	Series de triángulos
GL_TRIANGLE_FAN	Abanico de triángulos
GL_QUAD_STRIP	Serie de triángulos

Hay que tener en cuenta que cuando dibujamos un polígono, este no puede ser cóncavo, si lo fuese el resultado no esta especificado en OpenGL por lo que será inesperado. Además todos los puntos que lo forman deben ser coplanares.

Observaciones

Hay que tener en cuenta que cuando dibujamos un polígono, este no puede ser cóncavo, si lo fuese el resultado no esta especificado en OpenGL por lo que será inesperado. Además todos los puntos que lo forman deben ser coplanares, es decir, estar en un mismo plano, esto no es aplicable a las opciones en que se generán triángulos (GL_TRIANGLES, GL_TRIANGLE_STRIP y GL_TRIANGLE_FAN) ya que tres puntos siempre son coplanares al definir ellos mismo un único plano.

En realidad cada polígono tiene dos caras, la cara frontal y la cara posterior, por convenio la cara frontal es aquella en que los vértices están definidos en el sentido de las agujas del reloj, aunque esto lo podemos modificar con la función void glFrontFace(GLenum mode), el parámetro puede ser GL_CCW, por defecto o GL_CW, donde el sentido de las agujas del reloj corresponde a la cara posterior.

Funciones de color

Funciones para modificar el color son:

```
glColord(red, green, blue[, alpha]) |
glColord((red, green, blue[, alpha]))
glColor3d(red, green, blue)
```

Cuyos parámetros son:

- red, green, blue - Especifican la cantidad de rojo, verde azul.
- alpha - Especifica la intensidad de color.

Rotaciones

Rota en x,y,z por ángulo angle.

```
void glRotate( GLdouble angle,GLdouble x,GLdouble y,GLdouble z )
void glRotated( GLdouble angle,GLdouble x,GLdouble y,GLdouble z
void glRotatef( GLfloat angle,GLfloat x,GLfloat y,GLfloat z )
```

Cuyos parámetros son:

- angle - Especifica el ángulo de rotación.
- x,y,z - Especifica las coordenadas del vector.

Traslaciones

Traslada por vector x,y,z.

```
void glTranslate( GLdouble x,GLdouble y,GLdouble z)
void glTranslated( GLdouble x,GLdouble y,GLdouble z)
void glTranslatef( GLfloat x,GLfloat y,GLfloat z )
```

Cuyos parámetros son:

- x,y,z - Especifica las coordenadas del vector.

Double Buffer. No Parpadeo

La mejor técnica sin duda para mejorar un juego o cualquier aplicación que utilice animación es el intercambio de páginas. El intercambio de páginas usa dos o más superficies para asegurarse de que las operaciones de dibujo nunca se realizan sobre la superficie que el usuario está viendo. Por el contrario, las nuevas imágenes se dibujan sobre superficies no visibles, o bufferes secundarios, que se intercambian después con toda rapidez para hacerse visibles.
Además de intercambiar las superficies, nos interesa el momento en que se realiza este cambio. El monitor muestra la imagen recorriendo los pixeles de horizontalmente y de arriba abajo, una vez ha llegado abajo retorna a arriba, este tiempo se denomina intervalo de retorno vertical. Lo más apropiado es que se realice en este intervalo y así nos aseguramos que cada imagen se muestra al completo, si no fuese se daría el caso que el buffer cambiase a medio refresco del monitor y tendríamos media imagen actual y media imagen anterior.

Double Buffer. No Parpadeo

GLUT nos da la posibilidad de disponer de doble buffer y de realizar el cambio de buffer, todo esto mediante comandos.
En cambio, para especificar que el cambio de buffer se realice en el intervalo de retorno vertical debemos hacerlo configurando el driver de OpenGL (ICD) que dependerá de cada tarjeta de vídeo y las posibilidades que nos ofrezca.
Para disponer de doble buffer lo indicaremos en el momento de crear la ventana.
glutInitDisplayMode (GLUT_RGB — GLUT_DOUBLE); Para cambiar el buffer utilizamos la función void glutSwapBuffers(void), esta función ya realiza la llamada a glFlush(), por lo que no es necesario que la llamemos.

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org

 (O'Reilly) Practical C Programming (3rd Edition)

 http://www.cplusplus.com

 http://es.wikipedia.org/wiki/GLUT