

Práctica 1

Introducción a OpenGL y GLUT

Objetivo

En esta primera práctica se explican al alumno los fundamentos sobre los que se apoyará el desarrollo de todas las demás, persiguiendo con todas ellas introducir al alumno en el manejo de la librería OpenGL. Esta librería no proporciona, sin embargo, función alguna para la creación de una ventana sobre la que pintar, ni para gestionar la entrada por teclado o ratón, para lo cual sería necesario dedicar buena parte del código a llamadas del sistema operativo que distraen del objetivo principal. Gracias a la librería GLUT esto no será necesario, permitiendo al alumno la realización de las prácticas con independencia del sistema operativo utilizado, ya sea este MS Windows o Linux.

1. Antes de empezar

Los ordenadores del laboratorio donde se realizan las prácticas tienen instalados los sistemas operativos MS Windows y Linux. El alumno podrá elegir cualquiera de ellos para realizar las prácticas sin que ello suponga diferencias en el código que deberá escribir para completar cada práctica, pudiendo además portar dicho código de una a otra plataforma sin problemas. Las diferencias se encontrarán, eso sí, en el manejo propio del sistema operativo y las herramientas que se emplearán en uno y otro, dándose las oportunas indicaciones a lo largo de esta práctica.

Así, tras encender el ordenador del laboratorio aparecerá un menú en el que, de entre las opciones disponibles, seleccionaremos el sistema operativo MS Windows o Linux a utilizar para el desarrollo de la práctica. Normalmente, la elección por defecto es MS Windows, opción que será seleccionada automáticamente al cabo de unos segundos, por lo que aquellos alumnos que deseen trabajar en Linux deberán estar especialmente atentos a ese menú. En algunos ordenadores también es posible que sea necesario pulsar la tecla “Inicio” para acceder a ese menú, lo cual se indica con un apropiado mensaje en la pantalla.

En la Web de la asignatura se han colgado archivos con todo lo necesario para la realización de la práctica, por lo que una vez arrancado el sistema operativo preferido, se recomienda lanzar un navegador Web para acceder a la siguiente dirección:

<http://www.dsi.uclm.es/asignaturas/42538/practicas.htm>

En el caso de MS Windows, el archivo a descargar es `holamundo.zip`, tan sólo hay que bajarlo al disco local y descomprimirlo con una utilidad como, por ejemplo, el popular WinZip.

En la carpeta creada se encontrará, entre otros ficheros, uno que tiene por nombre `holamundo.cpp` y que contiene el código que se proporciona para esta práctica y que habrá que completar. Para verlo se puede utilizar cualquier editor de texto, pero dado que en MS Windows emplearemos el entorno de programación MS Visual C++ 2005, se recomienda abrirlo desde este.



Una vez lanzado entonces ese entorno, desplegamos el menú *Archivo*, seleccionamos la opción *Abrir* y tras ella *Proyecto o solución...*. Buscamos el directorio que se creó al descomprimir el archivo anterior y seleccionamos el archivo de solución o proyecto para abrir el entorno de trabajo correspondiente a la presente práctica.

En el caso de Linux, cuando el sistema nos pida nuestro login escribiremos `usuario`, y como password introduciremos `generico`:

```
login: usuario
password: generico
```



Entraremos en el entorno de ventanas X Windows introduciendo la siguiente orden en la línea de comandos:

```
$ startx
```

Para editar los ficheros fuentes podemos utilizar cualquiera de los editores de texto que proporciona el sistema operativo, como por ejemplo el `joe`, que podemos ejecutar desde una ventana de terminal (una vez arrancado este editor, podéis ver la pantalla de ayuda pulsando `Ctrl.+ H`), aunque se recomienda utilizar otro editor que explote mejor las características gráficas del entorno de ventanas.

El archivo a descargar de la Web es, en este caso, `holamundo.tgz`. Para descomprimirlo escribiremos en la línea de comandos esta otra orden:

```
$ tar -xzvf holamundo.tgz
```

Obtendremos un directorio `holamundo` en el que podemos encontrar también el código fuente `holamundo.cpp` que deberá ser completado, pero ahora los ficheros que antes definían el entorno de trabajo han sido sustituidos por un archivo `Makefile` que nos facilitará aquí la compilación.

En cualquier caso, el apagado de los ordenadores del laboratorio debe realizarse siempre de forma segura, lo que en Linux significa introducir la orden `shutdown` o `halt` para un correcto apagado.

2. ¿Qué es lo que hay que hacer en esta práctica?

Tomarse el tiempo que haga falta para leer el enunciado de esta práctica. Revisar con atención el código fuente que se proporciona, en lenguaje C++, y completarlo allá donde se indique. Compilar el programa y ejecutarlo. Comprobar que el resultado coincide con el ejecutable que se proporciona a modo de muestra. Realizar cambios en el código inicial para profundizar en el funcionamiento de las librerías OpenGL y GLUT.

3. Una breve introducción a OpenGL

OpenGL (*Open Graphics Library*) es una interfaz de programación o API para gráficos tridimensionales que, en tan sólo unos pocos años, se convirtió en una norma industrial de enorme aceptación, y considerado por muchos como un lenguaje ensamblador para gráficos.



OpenGL es el resultado de los esfuerzos de Silicon Graphics (Sgi) para mejorar la portabilidad de IRIS GL, la primera API de programación para las estaciones gráficas IRIS de la misma compañía. En 1992 se presentó la versión 1.0 de la especificación OpenGL, con el respaldo de importantes compañías como Digital, IBM, Intel y Microsoft –esta lo abandonaría una década después en pos de su propia tecnología DirectX-.

OpenGL es un lenguaje funcional más que un lenguaje declarativo. En lugar de diseñar la escena y cómo debe aparecer, el programador describe los pasos necesarios para conseguir una determinada apariencia final. OpenGL puede verse como una caja oscura, una máquina de estados cuyas transiciones son provocadas a través de llamadas a una API que actualmente incluye más de 250 funciones. Estas se emplean para dibujar primitivas gráficas como puntos, líneas y polígonos en tres dimensiones, soportando además Z-buffer, sombreado e iluminación, texturas y otros efectos especiales.

Además, si bien hasta la versión 2.0 cada etapa de la pipeline realizaba una función fija configurable sólo dentro de unos límites dados, a partir de esa versión es posible programar muchos otros efectos visuales usando el lenguaje GLSL (*OpenGL Shading Language*).

OpenGL está pensado para trabajar en sistemas con aceleración por hardware, y en este sentido podría definirse como una interfaz software para gráficos por hardware. Los fabricantes de hardware para gráficos para PC añaden entonces soporte OpenGL a

sus productos implementando las funciones descritas en la especificación. También existen implementaciones por software, siendo una de ellas la popular librería Mesa.

En cualquier caso, las aplicaciones OpenGL escritas adecuadamente no deben diferenciar entre generación acelerada por hardware y la generación puramente software de una implementación genérica.

Con todo, OpenGL no incluye ninguna función para la gestión de ventanas, interactividad con el usuario ni manejo de ficheros. Cada entorno particular (MS Windows, X Windows, etc...) posee sus propias funciones para este propósito y es responsable de facilitar a OpenGL la facultad de dibujar en una ventana o en un mapa de bits.

Por ejemplo, en el entorno de ventanas X Windows, el servidor X proporciona soporte OpenGL a través de la extensión GLX, y los pasos que deben seguirse para crear una ventana sobre la cual poder dibujar con OpenGL incluyen llamadas tanto a procedimientos de esa extensión como de la librería Xlib de Linux. Aunque el código OpenGL resulta totalmente portable, el empleo explícito de la librería Xlib elimina tal posibilidad.

Gracias a la librería GLUT podremos crear una ventana para dibujo y gestionar las entradas empleando el mismo conjunto de funciones tanto en MS Windows como en X Windows. De todos modos, quien desee encontrar información acerca de cómo preparar un entorno concreto de ventanas para trabajar con OpenGL podrá encontrarla en libro de Mark J. Kilgard “OpenGL Programming for the X Window System” y en las páginas Web de los tutoriales de Jeff Molofee:

<http://nehe.gamedev.net/lesson.asp?index=01>

4. La librería GLUT

La librería GLUT (*OpenGL Utility Toolkit*) es una librería escrita por Mark Kilgard que nos ofrece una interfaz de programación a través de la cual realizar, de manera sencilla, muchas de las tareas que conlleva la programación con OpenGL en un entorno de ventanas, pero sin preocuparnos del gestor concreto de ventanas, ya sea este MS Windows o X Windows.

Esto no significa, por ejemplo, que prescindamos de la librería Xlib al programar sobre X Windows, sino que la versión para X Windows de la librería GLUT ocultará al programador las llamadas a rutinas Xlib. Nuestro código será portable a cualquier entorno de ventanas para el que exista una versión de la librería GLUT.

Para poder usar la librería GLUT en MS Windows necesitaremos la librería dinámica `glut32.dll`, la cual puede descargarse desde la página Web de la asignatura. Una vez bajada al disco local, esta deberá copiarse en el mismo directorio donde se ejecute nuestro programa –en esta primera práctica `holamundo.exe`– o bien copiarla en un directorio donde el sistema busca esta clase de librerías, como por

ejemplo el directorio `Windows\System`, y así estará disponible para las siguientes prácticas y, en general, para cualquier otra aplicación OpenGL que haga uso de GLUT.

En cambio, la librería GLUT ya está instalada en el sistema operativo Linux de los ordenadores del laboratorio, por lo que el alumno no tendrá que hacer nada más. En cualquier caso, desde la misma página Web también puede descargarse el código fuente de esta librería (en formato tar de Unix) preparado para ser compilado en diferentes plataformas. Igualmente, puede obtenerse desde la siguiente dirección:

<http://www.opengl.org/resources/libraries/glut/>

La última versión disponible es la 3.7, que es la escogida para la realización de las prácticas, si bien aún podemos descargar también versiones anteriores, como la 3.6. En el anexo B de este documento se detallan los pasos seguidos para la instalación de la librería.

Desafortunadamente, la librería GLUT parece haber sido abandonada por su autor, pues desde la publicación en agosto de 1998 de la versión 3.7 no ha aparecido ninguna nueva revisión de la misma. El hecho de que el autor mantenga sus derechos sobre la librería y que no permita a nadie distribuir código modificado de la misma, impide que otras personas lleven a cabo el necesario proceso de puesta al día de esta librería.

Esta obsolescencia de la librería GLUT es fuente de problemas, sobre todo con las más recientes distribuciones del sistema operativo Linux, problemas que no se deben al software de NVIDIA o de XFree86. De hecho, RedHat dejó por ello de incluir esta librería en sus distribuciones.

Existen, sin embargo, alternativas al uso de la librería GLUT original. Una de estas alternativas es la librería **freeglut**, que con una implementación completamente nueva nos ofrece la misma interfaz de programación de GLUT, e incluso la mejora. Podemos encontrarla en la siguiente dirección:

<http://freeglut.sourceforge.net/>

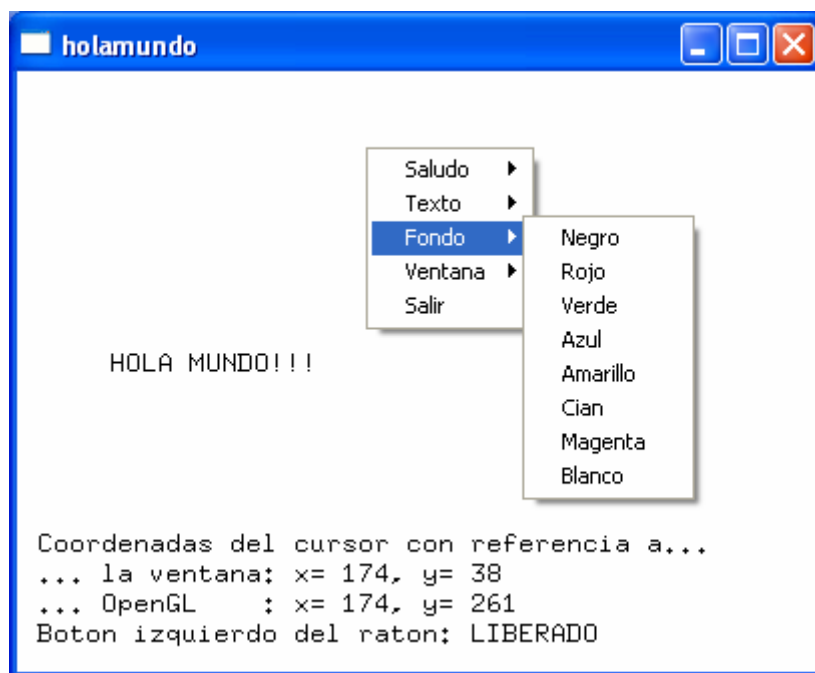
5. Un primer programa con OpenGL y GLUT

Una vez terminado el programa que vamos a escribir en esta práctica, el resultado visible será una ventana mostrando el clásico saludo “¡Hola Mundo!” al que recurren tantos libros de programación para presentar un primer ejemplo.

El texto de ese saludo exhibirá además una sencilla animación, deslizándose desde la parte izquierda a la derecha de la ventana. El usuario podrá parar y reanudar dicha animación a través de un menú que emergerá al pulsar el botón derecho del ratón, y desde el que también podrá cambiar el color del texto y del fondo, así como la presentación a pantalla completa o en ventana. Para completar el ejemplo, en la parte baja de la ventana se incluirá un bloque de texto que informará en todo momento de las

coordenadas actuales del cursor –en dos sistemas de referencia, el de la ventana y el empleado con OpenGL para dibujar el texto-, y el estado del botón izquierdo del ratón.

La siguiente captura se incluye con la pretensión de ilustrar lo anterior, si bien el alumno también encontrará en la Web un ejecutable de muestra que sirve mejor a ese propósito:



En los siguientes apartados se dará una explicación del código fuente que se proporciona al alumno –escrito en lenguaje C++, muy utilizado en gráficos por ordenador-, así como de aquellas partes que debe completar. Pero, para que pueda hacerse ya una primera idea, se introduce aquí el esquema básico del mismo.

Así, la función `main` comienza con la creación de una ventana sobre la cual poder dibujar con OpenGL. A continuación se invoca una función que hemos llamado `init`, y que utilizaremos para fijar las condiciones de inicio para los gráficos que dibujemos con OpenGL.

Siguiendo con la función `main`, hacia el final de la misma el programa llama a la función `glutMainLoop`, de la librería GLUT, para entrar en un bucle de espera de eventos desde el cual llamar, a su vez, a las funciones que nosotros hayamos programado para atender cada tipo de evento. Se sigue así el modelo “*don't call me, I'll call you*”, también conocido como modelo de Hollywood.

Esas funciones son registradas también en la función `main`, si bien como funciones escritas por el programador –nosotros- las encontraremos repartidas a lo largo del archivo del código fuente que se proporciona para esta práctica y las siguientes. Sus nombres pueden además variar según el gusto de cada programador, aunque en estas prácticas el alumno encontrará normalmente una función `redraw` para repintar la ventana, una función `reshape` a la que llamar si se redimensiona la ventana, una

función `keyboard` si se pulsa una tecla, las funciones `passive`, `mouse` o `motion` si se mueve el puntero con el ratón o se actúa sobre los botones de este, y una función `idle` que ejecuta cuando la CPU esté ociosa.

Con todo, el esquema básico del código de esta práctica y las que le seguirán, usando la librería GLUT, será entonces similar al siguiente:

```
void
init(void)
{
    /* ... */
}

void
redraw(void)
{
    /* ... */
}

void
reshape(int w, int h)
{
    /* ... */
}

void
keyboard(unsigned char key,
          int x, int y)
{
    /* ... */
}

void
passive(int x, int y)
{
    /* ... */
}

void
mouse(int button, int state,
       int x, int y)
{
    /* ... */
}

void
motion(int x, int y)
{
    /* ... */
}

void
idle(void)
{
    /* ... */
}

void
main(int argc, char* argv[])
{
    /* ... */

    glutCreateWindow("...");

    init();

    /* ... */

    glutMainLoop();
}
```

5.1. La función `main`

Si no utilizásemos la librería GLUT, el código de la función `main` tendría una longitud considerable debido al gran número de llamadas a rutinas del sistema operativo para crear la ventana, más el código perteneciente al bucle de espera de eventos. Gracias a la librería GLUT podremos reducir todo ese código a un pequeño conjunto de llamadas de más alto nivel.

Así, la función `main` comienza procesando la línea de comandos, estableciendo una conexión con el servidor y comprobando que el servidor soporta OpenGL. La función `glutInit` hará todo eso por nosotros, tan sólo hay que pasarle los dos argumentos de la función `main`:

```
glutInit(&argc, argv);
```

Después invocaremos a la función `glutInitDisplayMode` para indicar las características que deseamos que tenga la ventana que crearemos posteriormente, en este caso tan sólo doble buffer (`GLUT_DOUBLE`) y modelo de color RGBA (`GLUT_RGBA`). Como argumento le pasaremos un “OR” a nivel de bit de esas dos opciones:

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
```

Ahora podemos crear la ventana para dibujar sobre ella con OpenGL. La función que invocaremos será `glutCreateWindow`. Esta función seleccionará el visual y el mapa de colores más ajustado a nuestros deseos, y creará un contexto OpenGL asociado a la ventana. Tan sólo hay que pasarle como argumento una cadena de caracteres con el título de la ventana:

```
glutCreateWindow("holamundo");
```

Creada la ventana, es el momento para preparar el contexto OpenGL fijando su estado inicial, para lo que llamaremos a nuestra función `init`.

La parte final de la función `main` es ocupada por el bucle de eventos. No es necesario, sin embargo, escribir el bucle con una sentencia `while...do`, sino que entrar en el bucle de eventos se reduce a llamar a la función `glutMainLoop`, que no requiere ningún argumento.

Eso sí, si deseamos que nuestro programa responda a los eventos que llegan, es preciso entonces escribir las apropiadas funciones gestoras de eventos y registrarlas como tales antes de entrar en el bucle. Por ejemplo, la función `reshape` será invocada siempre que se cambien las dimensiones de la ventana. Para ello, debemos entonces registrarla como función que gestionará ese evento, empleando a su vez otra función de la librería GLUT:

```
glutReshapeFunc(reshape);
```

Cada función gestora deberá tener una firma concreta según el evento que desee gestionar, esto es, deberá contar con un número y tipo de argumentos preestablecidos. Además, para cada tipo de evento tendremos una función distinta para registrar la función gestora.

La siguiente tabla recoge el nombre de cada una de esas funciones de la librería GLUT, junto a los nombres de las rutinas que debemos registrar y el tipo de evento que atenderán:

Tipo de evento	Rutina para registrar la función gestora	Nuestra función gestora
<i>El contenido de la ventana debe ser redibujado.</i>	glutDisplayFunc	redraw
<i>La ventana es movida o redimensionada.</i>	glutReshapeFunc	reshape
<i>Una tecla que genera un carácter ASCII ha sido pulsada.</i>	glutKeyboardFunc	keyboard
<i>El puntero es desplazado moviendo el ratón sin actuar sobre ninguno de sus botones.</i>	glutPassiveMotionFunc	passive
<i>Un botón del ratón ha sido pulsado o soltado.</i>	glutMouseFunc	mouse
<i>El cursor es desplazado moviendo el ratón mientras uno o más de sus botones se mantienen pulsados.</i>	glutMotionFunc	motion
<i>Ninguna otra tarea pendiente.</i>	glutIdleFunc	idle

5.2. La función `init`

OpenGL proporciona un conjunto de primitivas con las que poder modificar o consultar el estado de esa caja oscura que representa el motor de *rendering*. Antes de dibujar nada con OpenGL querremos inicializar ese motor apropiadamente, y esta es la misión de la función `init`, contener todo el código necesario para llevar a cabo la puesta a punto del entorno de dibujo OpenGL.

Así, podemos comenzar especificando el color de fondo con la función `glClearColor`, la cual toma cuatro números en coma flotante como argumentos.

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

Los tres primeros corresponden a cada uno de los tres colores primarios que forman el modelo RGB: rojo (*Red*), verde (*Green*) y azul (*Blue*). Se trata de un sistema aditivo, esto es, el color final es el resultado de la suma de las contribuciones individuales de cada primario. Para cada uno de ellos se especifica entonces un valor entre 0 y 1, indicando así su contribución al color final, de menos a más. La tabla

siguiente recoge los valores para esas tres componentes de algunos de los colores más conocidos:

Color	R	G	B
Negro	0.0	0.0	0.0
Rojo	1.0	0.0	0.0
Verde	0.0	1.0	0.0
Azul	0.0	0.0	1.0
Amarillo	1.0	1.0	0.0
Cian	0.0	1.0	1.0
Magenta	1.0	0.0	1.0
Blanco	1.0	1.0	1.0



En esta ocasión emplearemos como color de fondo el negro, cuyas componentes son (0.0, 0.0, 0.0). El último parámetro de esta función corresponde a la componente *alpha*, componente que no explicaremos en esta práctica y que dejaremos a cero:

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

Para especificar, en cambio, el color con el que se dibuja una primitiva geométrica, como por ejemplo el texto que queremos mostrar en esta práctica, haremos uso de otra función, `glColor3f`. Esta función pertenece a la familia de funciones `glColor*`, con diferencias en el número de argumentos (3 ó 4) y el tipo de los mismos. En este caso, su sufijo nos indica que toma 3 valores en coma flotante de simple precisión.

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
```

Ahora, como ese texto que queremos mostrar en nuestro programa será todo del mismo color, también podemos fijar su color inicial en esta función `init`. Ese color será el blanco:

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

Es imprescindible tener claro que OpenGL funciona como una máquina de estados y que cualquier cambio en su configuración sólo tendrá efecto en las siguientes acciones que se realicen. Así, el color que se especifica con `glColor3f` se aplicará a todas las primitivas geométricas que se definan a partir de ese instante y hasta que se realice una nueva llamada a `glColor3f`, en consonancia con esa filosofía de máquina de estados adoptada para OpenGL.

5.3. La función `redraw`

Esta función tiene la tarea de actualizar el contenido de la ventana, concretándola en tres acciones: primero, limpiar el contenido del frame buffer; después,

dibujar la figura a representar en el mismo; y, finalmente, actualizar la ventana con la nueva imagen.

Borraremos el contenido del frame buffer llamando a la función `glClear`. Esta también permite limpiar el contenido de otros buffers, y de hecho en posteriores prácticas la utilizaremos para limpiar además el contenido del buffer que guarda los bits de profundidad, o Z-buffer, si bien aquí sólo borraremos los bits de color, sobrescribiéndolos con el color de fondo.

```
void glClear(GLbitfield mask);
```

Como argumento, esta función recibe una máscara con la que se indica qué buffers deben ser limpiados. Como sólo queremos limpiar los bits de color, el argumento a pasar aquí es `GL_COLOR_BUFFER_BIT`:

```
glClear(GL_COLOR_BUFFER_BIT);
```

Con el frame buffer limpio, se dibuja la nueva imagen describiendo la forma y apariencia de los objetos que aparecen en ella con llamadas a funciones OpenGL. En esta primera práctica no prestaremos mucha atención a esas funciones, limitándonos a dibujar cadenas de texto a partir de una posición que indicaremos con un par de coordenadas x e y .

La impresión de caracteres es sencilla, usando para ello un buffer auxiliar, la función estándar `sprintf` y otra función propia que hemos llamado `output`. Por ejemplo, en el código que se muestra a continuación la función `sprintf` añade al texto que se observa como segundo parámetro el valor –como números de tipo entero– de los parámetros que le siguen, y lo copia como una cadena de caracteres en la variable global `buffer`:

```
sprintf(buffer, "... la ventana: x=%d, y=%d",  
        mouseX, mouseY);  
output(10, 45, buffer);
```

Nuestra función `output` tomará entonces la cadena de caracteres que se le pasa y la imprimirá en la ventana carácter por carácter, partiendo de las coordenadas que se le indican en los dos primeros parámetros:

```
void output(GLfloat x, GLfloat y, char* text)  
{  
    char *p;  
  
    glRasterPos2f(x, y);  
    for (p = text; *p; p++)  
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);  
}
```

La tercera y última acción de `redraw` es actualizar el contenido de la ventana. Al hacer uso de doble buffer, la nueva imagen aún no es visible en la ventana del programa, pues ha sido dibujada en un buffer de trabajo. La imagen que se muestra procede de un segundo buffer, que guarda la imagen anterior. Estos dos buffers deben ahora intercambiarse los papeles, lo que la librería GLUT nos facilita con una única llamada a la función `glutSwapBuffers`, sin argumentos:

```
glutSwapBuffers();
```

5.4. La función `reshape`

Cuando se modifica el tamaño de la ventana del programa, bien por la acción del usuario o por una orden del propio programa, se invoca esta función `reshape`, quien recibe las nuevas dimensiones –en pixels– de esa ventana. El contexto de dibujo OpenGL también debe tener en cuenta este cambio, para lo que se recurre a la función `glViewport`.

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

Los dos primeros argumentos indican el origen de la región disponible dentro de la ventana, y los dos siguientes la anchura y altura de esa región. Aquí, los dos primeros serán 0, y para los dos siguientes se empleará la nueva anchura y altura de la ventana, definiendo así toda la ventana como puerto de vista:

```
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
```

Ahora que conocemos las dimensiones de la ventana, también podríamos especificar la forma de proyectar los objetos. En esta primera práctica, sin embargo, no proyectaremos ningún objeto, sino que configuraremos nuestro contexto OpenGL para el dibujo en dos dimensiones. El código que hace posible esto es el siguiente:

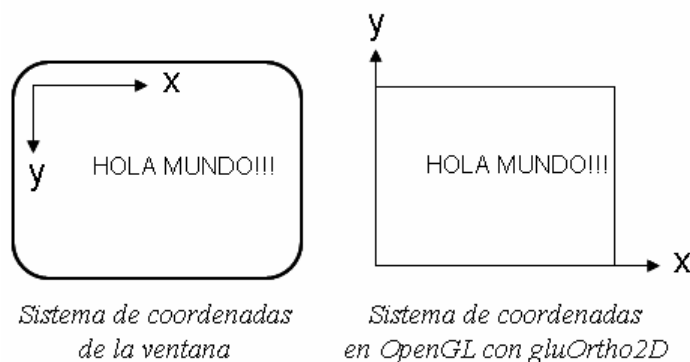
```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0, (GLdouble) w-1, 0, (GLdouble) h-1);  
glMatrixMode(GL_MODELVIEW);
```

De las anteriores funciones sólo se comentará `gluOrtho2D`, las demás se explicarán en posteriores prácticas.

```
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
```

Esta función define los límites de un rectángulo, indicando las coordenadas mínima y máxima en horizontal –*left* y *right*– y en vertical –*bottom* y *top*–. En nuestro caso esos límites coinciden, de nuevo, con las dimensiones de la ventana, de forma que cada unidad de área de ese rectángulo se corresponde con un píxel de la ventana.

Sin embargo, el sistema de coordenadas en el que se basa `gluOrtho2D`, y que se ha empleado en la función `redraw` al indicar la posición de las cadenas de texto, no es el de la ventana, ya que en esta última el origen se sitúa en la esquina superior izquierda –con el eje x apuntando hacia la derecha y el eje y apuntando hacia abajo–, y aquí en cambio el origen se encuentra en la esquina inferior izquierda –con el eje y apuntando ahora hacia arriba–.



La función `gluOrtho2D` no está incluida en la colección básica de funciones OpenGL, sino que pertenece a la librería GLU (*OpenGL Utility Library*). Como habrá podido observarse, el prefijo de cada una de las funciones que empleamos delata la librería a la que pertenece. Así, el prefijo `gl-` indica que la función se incluye en la librería OpenGL básica, `glu-` en la librería GLU y `glut-` en la librería GLUT.

5.5. La función `keyboard`

La función `keyboard` gestiona los eventos de entrada por teclado. Cada vez que una tecla es pulsada, esta función es invocada pasándole el valor ASCII de esa tecla. En esta práctica deseamos que el programa acabe si el usuario pulsa la tecla “Esc” (valor ASCII 27 en decimal). El programa terminará con una llamada `exit(0)`.

5.6. Las funciones `passive`, `mouse` y `motion`

Estas tres funciones se encargarán de atender los eventos del ratón, recibiendo todas ellas como argumentos las coordenadas actuales (x , y) del puntero que el ratón desplaza por la ventana. Sin embargo, no se llama a las tres a la vez, sino sólo a una de ellas, según el estado del ratón.

Así, la función `passive` será llamada siempre que no se actúe sobre ningún botón del ratón mientras se desplaza este para mover el puntero. Cuando el usuario pulsa uno de los botones o lo libera, entonces se llama a `mouse`. Y, mientras uno de los botones se encuentre pulsado, la función en ser llamada será `motion`.

Como en esta práctica se pretende mostrar por pantalla las coordenadas del puntero, todas ellas servirán para guardar, cada vez que sean llamadas, las últimas

coordenadas que se tengan del mismo. Para ello se han introducido en el código dos variables globales: `mouseX` y `mouseY`.

Esas coordenadas son después recuperadas desde la función `redraw` para ser mostradas por pantalla. Pero no debe invocarse la función `redraw` directamente. En su lugar, debe invocarse la función `glutPostRedisplay`, sin argumentos, cuya misión es la de marcar que el contenido de la ventana debe ser actualizado. Tan pronto sea posible, el sistema llamará entonces a la función registrada para llevar a cabo esa tarea, en nuestro caso la función `redraw`.

5.7. La función `idle`

Una función que se ejecute cuando nuestro programa no haga nada resulta bastante útil para crear animaciones. Nuestra función para ese cometido se llama `idle`, y como sabemos debe ser registrada usando la función `glutIdleFunc`.

El código de la función `idle` puede reducirse a una simple operación de incremento de una cierta variable de estado de los modelos que se dibujan en pantalla, para lo cual será necesario que esa variable sea visible tanto para esta función como para la que se encarga del redibujado, esto es, debe ser una variable global.

Además, cada vez que modifiquemos el valor de esa variable de estado, será necesario actualizar el contenido de la ventana para reflejar el cambio, lo que debe hacerse no llamando directamente a `redraw` sino, como se ha explicado en el apartado anterior, invocando la función `glutPostRedisplay`.

En esta práctica, la animación que se desea crear es bien sencilla, tan sólo desplazar el texto del saludo de un lado a otro de la ventana, para lo que se define una variable global llamada `helloX` en la que guardar el desplazamiento en horizontal. Cuando el texto alcance el lado opuesto de la ventana, el valor de esa variable debe volver a cero.

Para detener la animación, basta con desregistrar la función `idle`, pasando el argumento `NULL` a la función `glutIdleFunc`. De esa manera, el sistema no vuelve a invocarla y la animación no continúa. Para reanudarla, se registra de nuevo.

5.8. Un menú emergente para nuestro programa

La librería GLUT proporciona también un sencillo conjunto de rutinas para crear menús emergentes en cascada. Los menús pueden ser creados, configurados y asociados a un botón del ratón. Cuando se pulsa ese botón del ratón y se selecciona un ítem, el valor asociado a ese ítem se pasa a la función gestora del menú.

Por ejemplo:

```
glutCreateMenu(popupmenu);  
glutAddMenuEntry("Salir", 666);  
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

El resultado es un menú emergente con una única opción, que es la de terminar el programa. El menú se asocia a la ventana actual y será desplegado cuando se pulse el botón derecho dentro de esa ventana. La selección del ítem de este menú llamará a la función gestora `popupmenu`, que tendría el siguiente aspecto:

```
void popupmenu(int value)  
{  
    if (value == 666)  
        exit(0);  
}
```

Para poder añadir un submenú necesitamos una variable que guarde su identificador. El siguiente código muestra cómo extender el menú anterior para crear un menú en cascada:

```
int windowMenu;  
  
windowMenu = glutCreateMenu(windowSelect);  
glutAddMenuEntry("Pantalla completa", 1);  
glutAddMenuEntry("Restaurar", 2);  
  
glutCreateMenu(popupmenu);  
glutAddSubMenu("Ventana", windowMenu);  
glutAddMenuEntry("Salir", 666);  
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

En esta ocasión el menú emergente permite seleccionar el ítem “Salir” o bien desplegar el submenú “Ventana”. Si el usuario despliega este submenú y selecciona una de sus entradas, se invocará entonces a la función `windowSelect`, función que podríamos completar así:

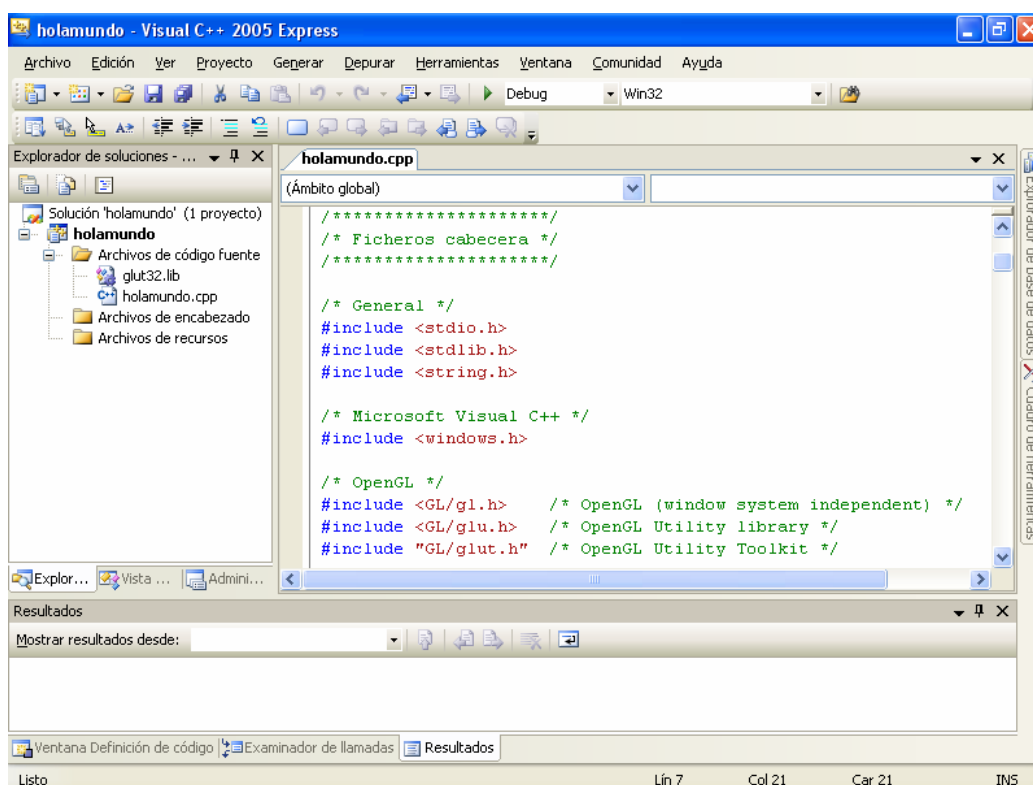
```
void windowSelect(int value)  
{  
    if (value == 1) glutFullScreen();  
    if (value == 2) glutReshapeWindow(640, 480);  
    glutPostRedisplay();  
}
```

Se propone entonces que el alumno cree un menú emergente que, además de las opciones anteriores, pueda seleccionar también el color del texto y del fondo de la ventana de entre ese conjunto de colores predefinidos, y que pueda parar o reanudar la animación del saludo. Una vez más, se recuerda que al modificar algún parámetro de la escena es aconsejable actualizar la ventana pero, en lugar de hacerlo de forma explícita,

solicitamos que se redibuje tan pronto como sea posible llamando a la función `glutPostRedisplay`.

6. Compilando en Windows con MS Visual C++ 2005

La siguiente imagen corresponde a una captura de la ventana de MS Visual C++ 2005 Express Edition una vez abierto ese entorno de trabajo. Los menús y botones de acceso directo se encuentran en la parte de arriba. A la izquierda se listan los ficheros que componen nuestro proyecto y las clases de objetos del mismo. En la parte de abajo se muestra el progreso y resultado de la compilación, entre otras informaciones.



Una vez abierto el entorno de trabajo que se proporciona, compilar el código de esta práctica es muy sencillo. Basta con desplegar el menú *Generar* y seleccionar la opción *Generar solución* o pulsar la tecla F7. Si no surge ningún problema durante la compilación, podremos lanzar el ejecutable resultante desplegando el menú *Depurar* y seleccionando esta vez *Iniciar sin depurar*, o bien pulsando la combinación de teclas Ctrl.+ F5.

Tras la compilación, el fichero ejecutable se encuentra bajo el subdirectorio Debug de nuestro directorio de trabajo. Allí podemos copiar la librería `glut32.dll` o bien, como se ha indicado previamente, dejarla en un directorio como `Windows\System`. Si la librería no es accesible el programa dará error en la ejecución.

7. Compilando en Linux

Para compilar el programa bastará con escribir la siguiente orden en la línea de comandos:

```
$ make
```

o bien,

```
$ make holamundo
```

El programa make encontrará las instrucciones necesarias para llevar a cabo la compilación en el fichero `Makefile`. El resultado es el mismo que si invocáramos directamente al compilador con la siguiente orden:

```
$ g++ -o holamundo holamundo.cpp -lGL -lGLU -lglut
```

que indica que se compile el fichero `holamundo.cpp` y que se enlace el código con las librerías `libGL` (OpenGL), `libGLU` (*OpenGL Utility Library*) y `GLUT`.

Sin embargo, si bien la orden anterior debería bastar para obtener el ejecutable, en el caso de las distribuciones Linux más recientes es necesario enlazar también con las librerías `X11lib`, `Xlib` y `Xmllib` para evitar ciertos errores en la compilación derivados - como se ha comentado antes- de la obsolescencia de la librería `GLUT`. El fichero `Makefile` que se distribuye con la práctica incorpora estas modificaciones.

Para compilar usaremos `cc` o `g++` según programemos en C o C++. El lenguaje C es suficiente para trabajar con OpenGL, pero también podemos llamar a las mismas rutinas programando en C++. Una vez obtenido el ejecutable, podemos hacerlo correr de la siguiente manera:

```
$ ./holamundo &
```

Para limpiar nuestro entorno de trabajo ejecutaremos la siguiente orden, que borra los ficheros resultantes de compilaciones anteriores:

```
$ make clean
```

8. El fichero `holamundo.cpp`

Las únicas diferencias existentes entre el código que se proporciona para realizar la práctica en MS Windows –el que aquí se muestra- y el que el alumno usará en Linux son la directiva `#include <windows.h>` y el tipo que devuelve la función `main`, que en Linux es `int`.

Práctica 1 de Informática Gráfica: Introducción a OpenGL y GLUT

```
/* holamundo.cpp
 *
 * Informatica Grafica.
 * Curso 2006/2007.
 *
 * Practica 1.
 * Introduccion a OpenGL y GLUT.
 *
 * Jose Pascual Molina Masso.
 * Escuela Politecnica Superior de Albacete.
 */

/*****/
/* Ficheros cabecera */
/*****/

/* General */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Microsoft Visual C++ */
#include <windows.h>

/* OpenGL */
#include <GL/gl.h> /* OpenGL (window system independent) */
#include <GL/glu.h> /* OpenGL Utility library */
#include "GL/glut.h" /* OpenGL Utility Toolkit */

/*****/
/* Variables globales */
/*****/

/* Dimensiones de la ventana */
int width; // Ancho
int height; // Alto

/* Guardan la posicion del raton y el estado del
   boton izquierdo */
int mouseX = 0, mouseY = 0;
bool isLeftButtonPressed = false;

/* Guarda el desplazamiento en horizontal del texto */
float helloX = 0.0;

/* Buffer auxiliar de caracteres */
char buffer[256];

/*****/
/* Prototipos de funciones */
/*****/

void init(void);
void output(GLfloat x, GLfloat y, char* text);
void redraw(void);
void reshape(int w, int h);
void keyboard(unsigned char key, int x, int y);
void passive(int x, int y);
void mouse(int button, int state, int x, int y);
void motion(int x, int y);
void idle(void);
/* ... completar ... */

//*****/

void
init(void)
{
    /* Establecer como color de fondo el negro, RGB(0,0,0) */
    glClearColor(0.0, 0.0, 0.0, 0.0);
}
```

Práctica 1 de Informática Gráfica: Introducción a OpenGL y GLUT

```
/* Establecer como color para pintar el blanco, RGB(1,1,1) */
glColor3f(1.0, 1.0, 1.0);
}

/*****

void
output(GLfloat x, GLfloat y, char* text)
{
    char *p;

    glRasterPos2f(x, y);
    for (p = text; *p; p++)
        glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
}

/*****

void
redraw(void)
{
    /* Limpiar el contenido del buffer */
    glClear(GL_COLOR_BUFFER_BIT);

    /* Dibujar texto 2D */
    sprintf(buffer, "HOLA MUNDO!!!");
    output(helloX, height/2, buffer);

    sprintf(buffer, "Coordenadas del cursor con referencia a...");
    output(10, 60, buffer);
    sprintf(buffer, "... la ventana: x= %d, y= %d", mouseX, mouseY);
    output(10, 45, buffer);
    sprintf(buffer, "... OpenGL      : x= %d, y= %d", mouseX, height-mouseY-1);
    output(10, 30, buffer);

    if (isLeftButtonPressed)
        sprintf(buffer, "Boton izquierdo del raton: PULSADO");
    else
        sprintf(buffer, "Boton izquierdo del raton: LIBERADO");
    output(10, 15, buffer);

    /* Volcar el contenido del buffer sobre la memoria de video */
    glutSwapBuffers();
}

/*****

void
reshape(int w, int h)
{
    /* Todo el area disponible en la ventana se utilizara
       como puerto de vista */
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);

    /* Se define un rectangulo de dibujo de forma que cada
       unidad de area del mismo se corresponda con un pixel*/
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, (GLdouble) w-1, 0, (GLdouble) h-1);
    glMatrixMode(GL_MODELVIEW);

    /* Guardamos las dimensiones actuales de la ventana */
    width = w;
    height = h;
}

/*****

void
keyboard(unsigned char key, int x, int y)
{
    if (key == 27) /* Tecla de escape */
        exit(0);
}

/*****/
```

Práctica 1 de Informática Gráfica: Introducción a OpenGL y GLUT

```

/*****
void
passive(int x, int y)
{
    /* ... completar ... */
}

/*****

void
mouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON) { // Si el boton izquierdo...
        if (state == GLUT_DOWN) { // ...ha sido pulsado
            isLeftButtonPressed = true;
        }
        else {
            if (state == GLUT_UP) { // ...ha sido liberado
                isLeftButtonPressed = false;
            }
        }
        mouseX = x;
        mouseY = y;
        glutPostRedisplay();
    }
}

/*****

void
motion(int x, int y)
{
    if (isLeftButtonPressed) { // Si el boton izquierdo esta pulsado
        mouseX = x;
        mouseY = y;
        glutPostRedisplay();
    }
}

/*****

void
idle(void)
{
    /* ... completar ... */
}

/*****

/* ... completar ... */

/*****

void
main(int argc, char* argv[])
{
    /* Ids para los submenus del menu emergente */

    /* ... completar ... */

    /* Procesa la linea de comandos y negocia el inicio de una sesion
       OpenGL con el sistema de ventanas */
    glutInit(&argc, argv);

    /* Indicamos las características que deseamos que tenga la ventana que
       crearemos posteriormente: doble buffer y modelo de color RGBA */
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

    /* Crear una ventana para dibujo con OpenGL: selecciona el visual y el
       colormap apropiados, y crea un contexto OpenGL asociado a esa ventana */
    glutInitWindowSize(640, 480);
    glutCreateWindow("holamundo");
}

```

```
/* Inicializar el contexto OpenGL */
init();

/* Creamos un menu emergente con varios submenus */

/* ... completar ... */

/* Registramos las funciones que manejaran los eventos */

/* ... completar ... */

/* Esperar eventos */
glutMainLoop();
}
```

9. Bibliografía y recursos

- 📖 Kilgard, Mark J. “OpenGL Programming for the X Window System”. Addison-Wesley. 1996.
- 📖 Woo, M., Neider, J., Davis, T., y Shreiner, D. “OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2”. Addison-Wesley Developers Press. 1999.
- 📖 Wright, R.S., y Lipchak, B. “OpenGL”. Anaya Multimedia, 2005.
- 🔗 Edición on-line del libro “OpenGL Programming Guide”:
<http://fly.srk.fer.hr/~unreal/theredbook/>
- 🔗 Librería GLUT: <http://www.opengl.org/resources/libraries/glut/>
- 🔗 Tutoriales de Jeff Molofee: <http://nehe.gamedev.net/>
- 🔗 Tutoriales de Nate Robins: <http://www.xmission.com/~nate/opengl.html>

Anexo A. Instalación del driver Nvidia

Los ordenadores del laboratorio poseen una tarjeta gráfica con GPU Nvidia que proporciona aceleración por hardware para gráficos OpenGL. En particular, los ordenadores del laboratorio Software 2 están equipados con el modelo GeForce 6600. Para un correcto funcionamiento, tanto en MS Windows como en Linux se ha instalado también el correspondiente driver Nvidia, que puede descargarse desde la Web de esta compañía:

<http://www.nvidia.com/>

Esto es especialmente importante en el caso de las distribuciones Linux, pues aunque durante la instalación seleccionemos correctamente el modelo de nuestra tarjeta, es probable que el driver que se instale sea uno genérico que no nos permita hacer uso del hardware para OpenGL de la tarjeta. Para comprobarlo basta con echar un ojo al

fichero de configuración del entorno X Windows, `/etc/X11/xorg.conf` en los ordenadores del laboratorio, y si en `Section "Device"` encontramos `Driver "nv"` significará que se emplea ese driver genérico. Si fuera el driver acelerado de la tarjeta, esa línea debería ser `Driver "nvidia"`.

La solución, como se ha apuntado antes, consiste en bajarse e instalar ese driver acelerado para Linux. En la propia página de descargas de Nvidia se indican los pasos a seguir para su instalación, si bien en las últimas versiones resulta muy sencilla. Tan sólo hay que descargar el archivo indicado, por ejemplo `NVIDIA-Linux-x86_64-1.0-xxxx-pkg2.run`, y fuera del entorno X Windows lanzamos la siguiente orden:

```
# sh NVIDIA-Linux-x86_64-1.0-xxxx-pkg2.run
```

A través de una serie de pantallas completaremos la instalación. Podemos decir que esta ha tenido éxito si al arrancar el entorno de ventanas aparece, por instante, la imagen de marca de la compañía Nvidia. También podemos revisar el anterior archivo de configuración para comprobar que, efectivamente, se hace referencia al driver acelerado, y si a pesar de la instalación esto no es así, editaremos entonces esa línea del archivo.

Anexo B. Instalación de la librería GLUT en Linux

En primer lugar, debemos acceder a la siguiente dirección para descargarnos el archivo que contiene el código fuente de la librería:

<http://www.opengl.org/resources/libraries/glut/>

Esta explicación supondrá que hemos descargado la versión 3.7 (archivo `glut-3.7.tar.gz`), aunque los pasos son los mismos si, por ejemplo, decidimos descargar, compilar e instalar una versión anterior, como la 3.6.

Una vez en nuestro disco local, podemos descomprimir el archivo `.tar.gz` descargado ejecutando la siguiente orden en la línea de comandos:

```
# tar -xzf glut-3.7.tar.gz
```

Bajamos al directorio `glut-3.7`. Encontraremos un directorio `linux` y dentro de él un fichero `README` que detalla los pasos a seguir para compilar e instalar con éxito la librería. Lo primero que nos dice es que hay que copiar el fichero `Glut.cf` al directorio padre:

```
# cd glut-3.7
```

```
Leer el fichero: README.linux
```

```
# cd linux
```

```
Leer el fichero: README
```

```
# cp Glut.cf ..  
# cd ..
```

Se recomienda editar el fichero `Glut.cf` eliminando cualquier referencia a la librería Mesa. En particular, deben reemplazarse las siguientes dos líneas:

```
OPENGL = $(TOP)/../lib/libMesaGL.so  
GLU = $(TOP)/../lib/libMesaGLU.so
```

por:

```
OPENGL = -lGL  
GLU = -lGLU
```

A continuación, ejecutamos `mkmkfiles.imake` para preparar los ficheros `Makefile` necesarios para la compilación:

```
# ./mkmkfiles.imake
```

Seguidamente, bajamos al directorio `lib/glut` y reemplazamos el fichero `Makefile` por el que se encuentra en el directorio `linux`:

```
# cd lib/glut  
# cp ../../linux/Makefile .
```

De nuevo, se recomienda sustituir cualquier referencia a la librería Mesa que encontremos en el fichero `Makefile`:

```
OPENGL = $(TOP)/../lib/libMesaGL.so  
GLU = $(TOP)/../lib/libMesaGLU.so
```

por:

```
OPENGL = -lGL  
GLU = -lGLU
```

Finalmente, ejecutamos `make` para obtener la librería, creamos un par de ligaduras a la misma y lo copiamos todo al directorio `/usr/lib`:

```
# make  
# ln -s libglut.so.3.7 libglut.so  
# ln -s libglut.so.3.7 libglut.so.3  
# cp -d libglut.* /usr/lib
```

Anexo C. Usando OpenGL y GLUT con Visual C++ 2005

El entorno de trabajo que se proporciona para la realización de esta práctica en MS Windows ya está preparado para compilar y ejecutar el programa, pero para la información del alumno se describirá, a continuación, los pasos que se han seguido para crear un proyecto en MS Visual C++ 2005 para programar usando las librerías OpenGL y GLUT.

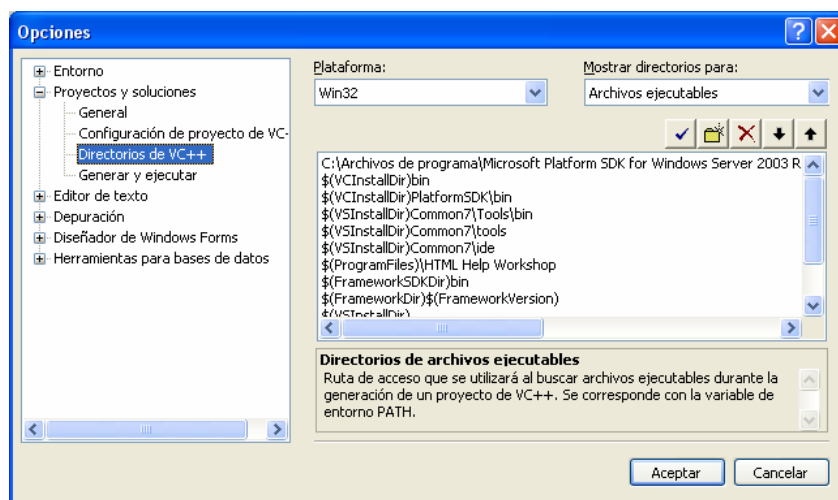
Como es de suponer, lo primero que necesitamos es el entorno de programación MS Visual C++, en su versión 8.0 (2005). Si no contamos con él, Microsoft ofrece de forma gratuita la versión Express Edition, que puede descargarse desde la siguiente dirección:

<http://msdn.microsoft.com/vstudio/express/visualc/default.aspx>

El tipo de aplicación que queremos crear no será .NET, ya que ni la librería OpenGL ni GLUT están disponibles para esa plataforma. En su lugar, generaremos código nativo de Windows, para lo que debemos comprobar que contamos también con el llamado Windows Platform SDK. Si no es así, éste puede descargarse también desde la Web de Microsoft.

Además, nuestro entorno VC++ debe estar configurado para trabajar con el anterior SDK. En el menú *Herramientas*, seleccionamos *Opciones...*, y en el diálogo que aparece desplegamos la rama *Proyectos y soluciones* y *soluciones*. Ahí deben aparecer las siguientes rutas en los respectivos apartados:

- Archivos ejecutables: C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Bin
- Archivos de inclusión: C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Include
- Archivos de biblioteca: C:\Archivos de programa\Microsoft Platform SDK for Windows Server 2003 R2\Lib

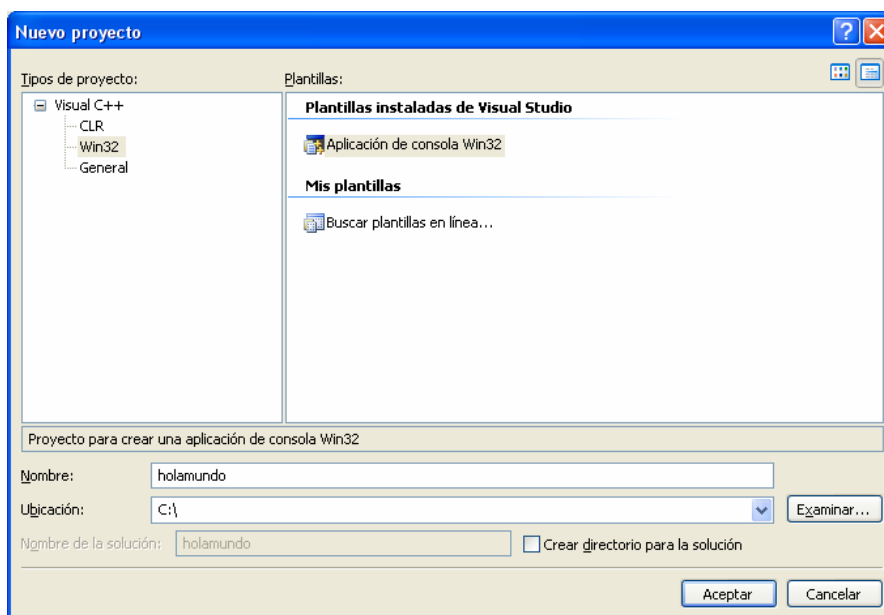


A continuación, necesitamos las propias librerías OpenGL y GLUT. En un ordenador que tenga instalado una tarjeta gráfica compatible con OpenGL, como los que encontramos en el laboratorio, la librería OpenGL se hallará en el subdirectorio `Windows\System32` en la forma de dos archivos, `opengl32.dll` y `glu32.dll`, los cuales representan la librería de órdenes básicas de OpenGL y la librería GLU. Sólo nos falta la librería GLUT que, como ya sabemos, podemos descargar de nuevo desde la dirección:

<http://www.opengl.org/resources/libraries/glut/>

El archivo que nos interesa es `glutdlls37beta.zip`. Aunque en estas prácticas trabajaremos con la versión 3.7 de la librería GLUT, también es posible descargar una versión anterior, como por ejemplo la 3.6. En cualquier caso, si descomprimos ese archivo encontraremos, entre otros, los siguientes ficheros: `glut.h`, `glut32.lib` y `glut32.dll`.

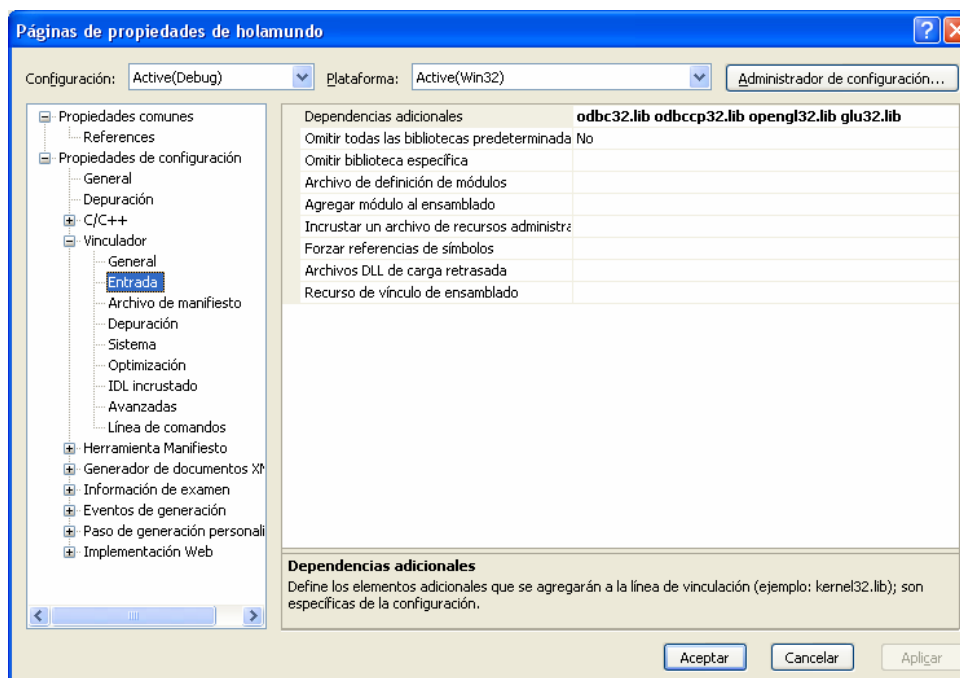
Con todo lo anterior, ya podemos crear nuestro proyecto en VC++ 2005. Para ello desplegamos el menú *Archivo* y seleccionamos *Nuevo*, y a continuación *Proyecto o solución...* Aparece un diálogo en el que seleccionamos, como tipo de aplicación, *Win32*, y como plantilla, *Aplicación de consola Win32*. Escribimos el nombre del proyecto y pulsamos *Aceptar*.



En los siguientes diálogos del asistente para aplicaciones Win32 seleccionamos la opción *Aplicación de consola* y marcamos la de *Proyecto vacío*. Tras pulsar el botón *Aceptar* obtendremos un entorno de trabajo limpio para empezar a trabajar.

A este proyecto recién creado agregamos un archivo C++ (`.cpp`) en el que incluir nuestro código. Para usar las librerías OpenGL y GLU en él se incluyen, como hemos visto en el archivo `holamundo.cpp`, los archivos de cabecera de estas.

Además, debemos añadir esas librerías al vinculador, para lo que desplegamos el menú *Proyecto* y seleccionamos *Propiedades de...* En el diálogo que aparece, desplegamos la rama *Propiedades de configuración* y a continuación *Vinculador*. Entonces seleccionamos el apartado *Entrada*, y en el diálogo de la derecha añadimos `opengl32.lib` y `glu32.lib` en *Dependencias adicionales*.



En el caso de la librería GLUT, copiaremos el archivo `glut32.lib` en el directorio de nuestro proyecto, crearemos también un subdirectorio `GL` y copiaremos en este el archivo de cabecera `glut.h`. El archivo `glut32.lib` lo agregamos después a la rama *Archivos de código* de nuestro proyecto, y el archivo `glut.h` lo incluimos en nuestro código con la siguiente directiva:

```
#include "GL\glut.h"
```

Se emplean comillas porque el fichero que se incluye se encuentra en el directorio de trabajo de nuestro proyecto, los ficheros cabeceras de librerías estándar se encierran entre los símbolos `<` y `>`. Por ejemplo, un nuevo fichero que debe incluirse es `windows.h`, lo que se consigue con la siguiente directiva:

```
#include <windows.h>
```

Por último, es necesario copiar también la librería `glut32.dll` en el mismo directorio donde se ejecute nuestro programa, tal vez el subdirectorio `Debug` o `Release` de nuestro proyecto, o bien copiarlo en un directorio donde suelen buscarse las librerías DLL, como el directorio `Windows\System`.