



# Entorno de desarrollo

*Seminario ISGI (S1)*



R. Vivó

# Entorno de desarrollo

## Seminario ISGI (S1)

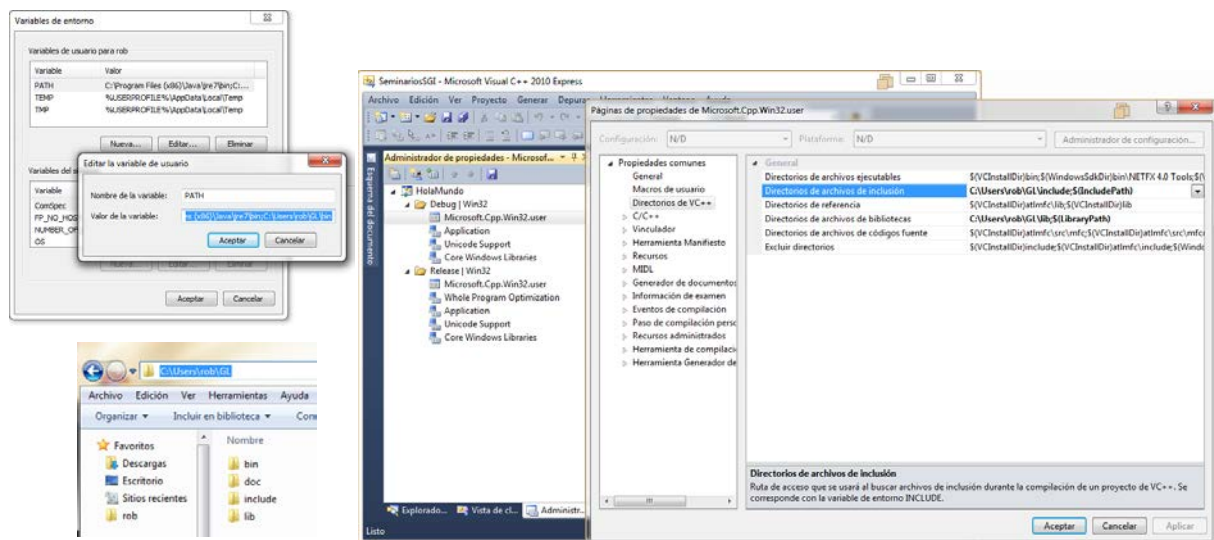
Este seminario explica cómo instalar y poner en funcionamiento el entorno de desarrollo de software gráfico utilizado en la asignatura de Introducción a los Sistemas Gráficos Interactivos del grado en Ingeniería Informática de la ETSINF de la UPV.

## Lenguaje y entorno de desarrollo

Se usa el lenguaje C++ bajo el entorno de desarrollo Visual C++ de Microsoft como herramientas sobre las que se va a dar soporte académico. Los laboratorios tienen ya instalado el MS Visual Studio.

Para el trabajo particular fuera de los laboratorios en ordenadores personales, se recomienda la instalación del Visual C++ Express Edition (alternativamente puede instalarse la versión gratuita del Visual Studio desde la web de Microsoft) . Se puede descargar gratuitamente desde el sitio de software para alumnos UPV [1]. Los pasos a seguir para la instalación una vez descargado son los siguientes:

1. Ejecutar el instalador del Visual C++ con permisos de administrador
2. Construir el directorio GL (bajar desde poliformat y descomprimir)
3. Añadir a la variable de entorno de usuario *PATH* la carpeta *GL\bin*
4. En el Visual C++ añadir al *path* de *ficheros de inclusión* de la hoja de propiedades del usuario la carpeta *GL\include*
5. En el Visual C++ añadir al *path* de *bibliotecas* de la hoja de propiedades del usuario la carpeta *GL\lib*



Leer la documentación y familiarizarse con el entorno. Una fuente de consulta útil sobre Visual C++ es el MSDN de Microsoft [2]. Sobre asuntos del lenguaje C++ y librerías asociadas puede consultarse también [3].

Es interesante usar la Standard Library para C++ para el manejo de entrada/salida, *strings*, *templates* para estructuras de datos dinámicas, y las librerías clásicas de C.

Es importante incluir antes los ficheros cabecera de la Standard Library como el `cstdio` que los que hacen referencia a OpenGL como `glut.h`

Se puede seguir una explicación para Visual C++ 2008 de este punto en el polimedia [4]. Téngase en cuenta que el manejo de propiedades cambia algo en la versión 2010.

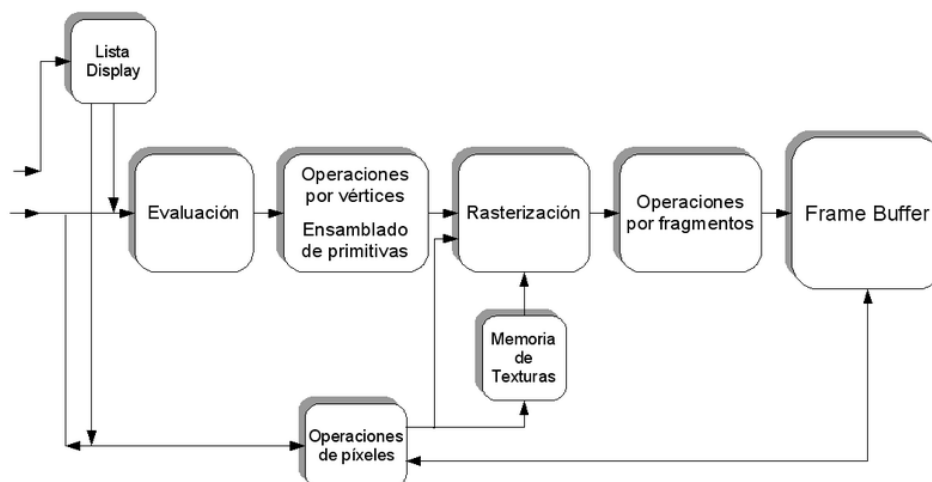
## Qué hace y qué no OpenGL

OpenGL es una especificación de la interfaz entre la aplicación de gráficos 3D y el hardware gráfico (API). Consiste en una máquina de estados y algoritmos de procesamiento gráfico que son accesibles a través de decenas de funciones. OpenGL realiza todo el procesamiento gráfico usando el hardware existente para representar en pantalla primitivas gráficas -puntos, líneas y polígonos- definidas en el espacio 3D. Mediante OpenGL es posible configurar el tamaño, posición y orientación de las primitivas, fijar el punto de observación y las condiciones de iluminación, y decidir que aspecto tendrán variando su apariencia.

OpenGL es independiente del hardware y la plataforma usada. Por ello no incluye ni funciones para gestionar ventanas ni funciones para gestionar los eventos de usuario. Estas tareas se encomiendan a librerías, como GLUT, construidas encima de OpenGL y específicas para cada plataforma usada. Así mismo, OpenGL tampoco proporciona funciones para la creación, gestión o interrelación de objetos gráficos 3D más allá de algunas funciones básicas para el modelado de superficies cuadradas y cúbicas incluidas en una librería anexa a OpenGL, la GLU. Para la gestión de escenas (colecciones ordenadas de objetos) se recomiendan herramientas construidas sobre OpenGL como OpenSceneGraph.

## Tubería gráfica en OpenGL

Llamamos tubería gráfica al orden en que se aplican las operaciones que transforman un modelo matemático (informático) de una primitiva 3D en una imagen en pantalla (conjunto de píxeles). El orden de operaciones en OpenGL es éste:



**Figura 1. Orden de operaciones en OpenGL**

Hay dos entidades gráficas principales, los vértices y los fragmentos. Los vértices hacen referencia a un punto en 3D y sus atributos (color, normal, coordenada de textura, etc.), mientras que los fragmentos son elementos pre-píxel, es decir, son posibles píxeles a los que se les asocia información de color, profundidad y transparencia, entre otros. Los fragmentos se obtienen en el proceso de discretización

(conversión al *raster*) de las primitivas. Obviamente puede haber dos vértices con las mismas coordenadas pero con distintos atributos. Lo mismo vale para los fragmentos.

En la tubería fija (figura 1) las operaciones por vértice y fragmento son siempre las mismas y se aplican o no dependiendo de si están habilitadas o deshabilitadas.

Actualmente es posible programar el comportamiento de cada fase de la tubería (excepto la de discretización) mediante lo que se ha dado en llamar “*shaders*”. Este tipo de tubería se denomina dinámica.

## Librerías relacionadas con OpenGL

---

Las librerías imprescindibles que nos van a permitir trabajar con gráficos son las siguientes:

1. OpenGL: Que incluye `gl.h` `glu.h` como ficheros de cabecera y `opengl32.lib` y `opengl32.dll` como librerías. En la instalación del entorno de desarrollo se incluyen todos estos ficheros, por lo que no hay que preocuparse. La librería dinámica `opengl32.dll` se instala en el System32 habitualmente. Los ficheros de cabecera suelen estar bajo un SDK de Microsoft en archivos de programa. No obstante puede habilitarse un directorio particular para direccionar todo esto por si no está en el sistema.
2. GLUT<sup>1</sup> (*freeglut*): Que incluye `freeglut.h`, `freeglut32.lib` y `freeglut32.dll`. Los ficheros de ubicarán en directorios convenientes para que sean accesibles tanto en montaje como en ejecución. Por ejemplo se pueden usar los mismos directorios que para OpenGL. Una fuente desde donde bajarse la librería GLUT es [4]. Es muy recomendable usar *freeglut*, una implementación más reciente, desde el repositorio [5]. Si fuera necesario, es posible generar las librerías usando el *make* apropiado. Otra posibilidad es OpenGLUT que puede encontrarse en [6]. Aunque cualquiera de ellos sirve para la programación básica de gráficos sobre Win32, sólo se dará soporte para *freeglut*.

Con la instalación de estas dos tecnologías es suficiente para generar gráficos por computador en el paradigma de tubería fija y con interactividad limitada al uso del ratón, teclas y menús de pop-up sin otros controles más complejos como botones, cajas de texto, etc., típicos de cualquier interfaz. Si se quiere ampliar las capacidades de programación, por ejemplo usando modo retenido o tubería dinámica, o se quiere mejorar la interfaz con controles mejores, los paquetes que se recomienda instalar son:

1. GLEW: Esta librería hace un envoltorio a la `gl.h` para poder usar las extensiones más allá de la versión 1.2 de OpenGL que las tarjetas gráficas incorporan hoy día. Para instalar la GLEW haremos lo mismo. Descargamos el paquete desde [7] para Win 32, copiamos los ficheros de cabecera a nuestro directorio de *includes* (`include/GL`), los ficheros *lib* al directorio `LIB` y la *dll* al *path* de ejecución. GLEW ofrece dos librerías la `glew32.lib` y la `glew32s.lib` que corresponden a la versión dinámica y estática respectivamente.
2. GLUI: Nos ofrece controles avanzados contruidos sobre OpenGL, por lo que su integración es fácil. Se puede descargar desde [8].

Existen otras posibilidades para la construcción de aplicaciones interactivas con una interfaz más cuidada como son las basadas en la tecnología Qt que exceden el ámbito de conocimiento de este seminario.

Hay documentación abundante sobre cada una de las librerías anteriores en las direcciones de descarga. Se recomienda consultarla.

---

<sup>1</sup> En adelante, cuando se haga referencia a GLUT, nos estaremos refiriendo a la implementación *freeglut*

## Sintaxis en OpenGL y GLUT

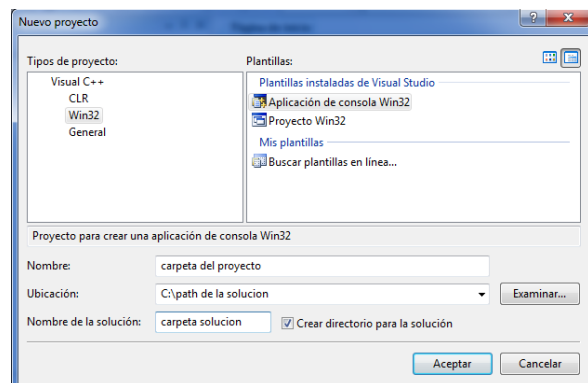
OpenGL y GLUT son librerías cuyo API consiste en:

1. Funciones que comienzan con `gl`, `glu`, `glut` según correspondan al API de OpenGL, GLU o GLUT respectivamente. El resto del nombre la función sigue el convenio de iniciales por ejemplo, `glutInitDisplayMode()`.
2. Tipos predefinidos, por ejemplo `GLfloat`.
3. Constantes que comienzan por `GL_` o `GLUT_` según la librería. El nombre de la constante es en mayúsculas y se separa por guiones bajos si consta de más de una palabra, por ejemplo `GL_COLOR_BUFFER_BIT`.
4. El API de OpenGL está orientado al lenguaje C. Por ello no existe polimorfismo en las funciones según el número o tipo de parámetros. Para subsanar esto, la misma función con diferente tipo o número en los parámetros tiene un sufijo diferente, por ejemplo para situar un punto puede usarse según el caso `glVertex2f(float x, float y)`, `glVertex3f(float x, float y, float z)` o `glVertex3i(int x, int y, int z)` entre otras.

## Hola mundo

Vamos a generar la primera aplicación de gráficos con OpenGL, GLUT y Visual C++. Para ello se seguirán los siguientes pasos previos:


1. Abrir Visual C++
2. Archivo->Nuevo->Proyecto
3. Win32, aplicación de consola. Ubicar la solución y el proyecto
4. Seleccionar “Configuración de la aplicación” y marcar “Proyecto vacío”. Con esto no se generará ningún tipo de plantilla. A continuación pulsar “Aceptar”.
5. Debe haberse generado una solución y un proyecto dentro de ella con los nombres del punto 3.
6. Agregar un nuevo archivo `HolaMundo.cpp` en la carpeta de “Archivos de código fuente”



A partir de aquí, pasaremos a editar el nuevo archivo creado como programa principal. Debe tener necesariamente una función `main` y sólo una. Los argumentos de la función `main` se adquieren cuando el programa se lanza con argumentos, por ejemplo desde línea de órdenes.

Para ello escribiremos el código siguiente:

```
#include <iostream>
#include <gl\freeglut.h>
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
void reshape(GLint w, GLint h)
{
}
void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(200,200);
    glutCreateWindow("Hola Mundo");
    std::cout << "Hola Mundo running" << std::endl;
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}
```

Este es el código más sencillo que podemos escribir y que nos servirá de plantilla para todos los programas sobre OpenGL. Para generar el ejecutable hacemos “Generar->Generar HolaMundo”. Esto debería compilar y montar sin errores a menos que no se hayan incluido bien las librerías. Para ejecutar, pulsar sobre el triángulo verde . La ejecución produce dos ventanas, la **consola** por donde salen los mensajes “cout” y la **ventana de dibujo** en negro pues no se ha dibujado nada. Observar las dimensiones de la ventana de dibujo y su título.

Observar el orden de inclusión de los ficheros de cabecera así como la función que cumplen.

Las funciones `display()` y `reshape(int,int)` se conocen como **callbacks** y son las que se ejecutan cuando se dispara un evento de esa clase. Es necesario registrarlas en GLUT antes del inicio del bucle de atención a eventos. El manejador de eventos “MainLoop” es el encargado de recoger los eventos de usuario y llamar a la función que corresponda a su tratamiento.

## Nuevas funciones y constantes

Sigue una pequeña referencia de las funciones y constantes utilizadas. Mientras no se diga lo contrario todas las funciones son void.

### GLUT

#### *glutInit()*

Sirve para inicializar la GLUT y debe llamarse antes de cualquier llamada. Puede procesar los parámetros del main pero es dependiente de la plataforma. Lo normal es llamarla sin parámetros.

#### *glutInitDisplayMode(unsigned int formatoBuffers)*

Configura la memoria gráfica según los buffers que se vayan a utilizar. Por defecto, si no se llama, la configuración es de un buffer de color con canal alfa en modo RGB. Corresponde a usar como parámetros `GLUT_RGBA|GLUT_SINGLE`. El parámetro es una máscara de bits por combinación or de constantes predefinidas. Se puede usar `GLUT_DOUBLE` en vez de `GLUT_SINGLE` para tener un buffer trasero además

del buffer frontal (*front y back buffers*). GLUT\_DEPTH para el uso del *Z-buffer*, GLUT\_STENCIL para el uso del *stencil buffer*, entre otros. La llamada debe hacerse antes de la creación de la ventana.

#### *glutInitWindowSize(int ancho, int alto)*

Determina el tamaño de la ventana de dibujo que se va a crear. Los parámetros indican el alto y el ancho en píxeles.

#### *int glutCreateWindow(char\* titulo)*

Crea la ventana, sin mostrarla todavía, y configura todas sus características como tamaño, posición y formato de buffers. El parámetro hace referencia al título que aparece en la barra superior de la ventana (barra de título) si es que existe con el manejador de ventanas usado. No se puede dibujar todavía sobre la ventana hasta que no se ponga en marcha el bucle de eventos con glutMainLoop(). La función devuelve un identificador único (manejador de la ventana). El identificador sirve para referirse a esa ventana en concreto, pasarle el foco por ejemplo, cuando se usa multiventana. Cada ventana mantiene su propio contexto de dibujo que ha quedado configurado con la llamada a esta función.

#### *glutDisplayFunc()*

Permite registrar la función que atenderá el evento de “*display*”. Este evento se produce cuando el contenido de la ventana de dibujo cambia por cualquier razón. La causa más frecuente es que se fuerce desde dentro de la aplicación el redibujo porque algo ha cambiado (por ejemplo en una animación). La forma de registrar la función de atención, llamada *callback*, es pasar su nombre como parámetro p.e. glutDisplayFunc(onDisplay) siendo onDisplay() la función que se activará cuando se procese el evento de redibujo.

#### *glutReshapeFunc()*

Como la anterior permite registrar la *callback* que atiende al evento de redimensión de la ventana de dibujo, por ejemplo cuando “estiramos” la ventana con el ratón. El evento de redimensión también se lanza cuando se crea la ventana.

#### *glutMainLoop()*

Esta función activa el bucle de atención de eventos. Cada evento que se origine en el sistema (ratón, teclado, etc.), o sea generado por la propia aplicación (p.e. redibujo), llega a una cola de eventos que se procesa en secuencia. Todos los eventos que tengan asociada una *callback* por el registro anterior son atendidos por el manejador activando la *callback*, a la que se le pasan los parámetros del evento si éste los tuviera.

#### *GLUT\_SINGLE*

Constante que indica la selección de un único *buffer* de dibujo.

#### *GLUT\_RGB*

Constante que indica la selección del modo de color rojo, verde, azul.

#### OpenGL

##### *glClear()*

Orden de borrado de cualquier buffer. Pueden indicarse varios simultáneamente mediante constantes que actúan como máscaras OR.

##### *glFlush()*

Orden de volcado a la tarjeta gráfica de todas las ordenes de dibujo pendientes de emitir. Con esto nos aseguramos que el dibujo se ha completado antes de proseguir.

### *GL\_COLOR\_BUFFER\_BIT*

Constante que se refiere al *buffer* de dibujo y que se inserta como parámetro en la función anterior `glClear()`.

## Bibliografía

---

- [1] «Visual C++ Express Edition 2010 y 2012,» [En línea, otoño 2020]. Available: <http://software.upv.es> (necesita identificación UPV)
- [2] «MSDN,» [En línea]. Available: <http://msdn.microsoft.com/en-us/library/3bstk3k5>.
- [3] «CPLUSPLUS,» [En línea]. Available: <http://www.cplusplus.com/>.
- [4] «Visual Studio con OpenGL y GLUT,» [En línea]. Available: <https://polimedia.upv.es/visor/?id=d73df12b-45b7-7a44-9ed9-1ba0de2a5c41>.
- [5] «GLUT de Nate Robins,» [En línea]. Available: <http://user.xmission.com/~nate/glut.html>.
- [6] «Freeglut,» [En línea]. Available: <http://sourceforge.net/projects/freeglut/>.
- [7] «OpenGLUT,» [En línea]. Available: <http://sourceforge.net/projects/openglut>.
- [8] «GLEW: OpenGL Extension Wrangler Library,» [En línea]. Available: <http://glew.sourceforge.net/>.
- [9] «GLUI: User Interface Library,» [En línea]. Available: <http://glui.sourceforge.net/>.



```
/******  
ISGI::Hola Mundo  
Roberto Vivo', 2012 (v1.0)  
  
Esqueleto basico de un programa en OpenGL  
  
Dependencias:  
+GLUT  
*****/  
#define PROYECTO "ISGI::S1E01::Hola Mundo"  
  
#include <iostream> // Biblioteca de entrada salida  
#include <gl\freeglut.h> // Biblioteca grafica  
  
void display()  
// Funcion de atencion al dibujo  
{  
    glClear(GL_COLOR_BUFFER_BIT);  
    glFlush();  
}  
  
void reshape(GLint w, GLint h)  
// Funcion de atencion al redimensionamiento  
{  
}  
  
void main(int argc, char** argv)  
// Programa principal  
{  
    glutInit(&argc, argv); // Inicializacion de GLUT  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // Alta de buffers a usar  
    glutInitWindowSize(200,200); // Tamanyo inicial de la ventana  
    glutCreateWindow(PROYECTO); // Creacion de la ventana con su titulo  
    std::cout << PROYECTO << " running" << std::endl; // Mensaje por consola  
    glutDisplayFunc(display); // Alta de la funcion de atencion a display  
    glutReshapeFunc(reshape); // Alta de la funcion de atencion a reshape  
    glutMainLoop(); // Puesta en marcha del programa  
}
```