Práctica 1. Primeros pasos en las IU basadas en OpenGL

Creando Interfaces de Usuario

Grado en Ingeniería Informática. Mención Computación
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Contenidos

- Configurar el Entorno.
- Conocer las librerías e includes.
- Dibujar triángulos
- Verificar la instalación y el prototipo de main()

Primer paso …

- En esta primera práctica programaremos y ejecutaremos una aplicación sencilla OpenGL en la que podremos visualizar la estructura esencial de GLUT. También aparecen múltiples elementos de GL, que no podemos comprender de momento. No es problema, no es necesario que los comprendamos, solamente los intuiremos.

- Aprenderemos la estructura general y los elementos de configurar la compilación y creación de la aplicación.

Algunas verificaciones

- Ejecutar Visual Studio 2013. Verificar que está instalado Qt.
- Verificar que podemos ejecutar Qt Designer.
- Verificar que está instalado OpenGL en la carpeta C:/OpenGL
- Verificar que está instalado MinGW
- Instalar el pluging de C/C++ de Netbeans
- Configurar las opciones de C/C++

Entorno de Trabajo

EL SDK de Visual Studio instala el soporte para utilizar las librerías GL y GLU, pero el resto de la programación, por el ejemplo para utilizar las ventanas y la interacción, debe realizarse con las clases de Microsoft.

Para trabajar en un modo abierto debemos descargar manualmente las distribuciones de las librerías necesarias, por ejemplo GLUT. Existen muchas formas de descargar e instalar las librerías necesarias para utilizar OpenGL. Para Windows hemos utilizado como referencia el magnifico Tutorial de OpenGL:

OGLdev: http://ogldev.atspace.co.uk/

Los tutoriales y ejemplos son muy ilustrativos. Al descargarlos también se incluyen el set completo de librerías y cabeceras necesarias que hemos instalado en la carpeta:

C:\OpenGL

Que incluye tres subcarpetas: include, lib y bin. En esta última deben estar las DLL. El Path del SO debe incluir este carpeta. Verificar la instalación.

Funciones básicas

- **Main()** contiene la creación de la ventana de la aplicación y definición de
- **Init()** para definir el espacio de trabajo y el color del fondo
- **Display()** En cada aplicación se programa el contenido del espacio de dibujo
- **Resize()** No la utilizaremos en esta práctica, se utiliza para repintar el contenido en función de los cambios de tamaño de la ventana que realice el usuario.

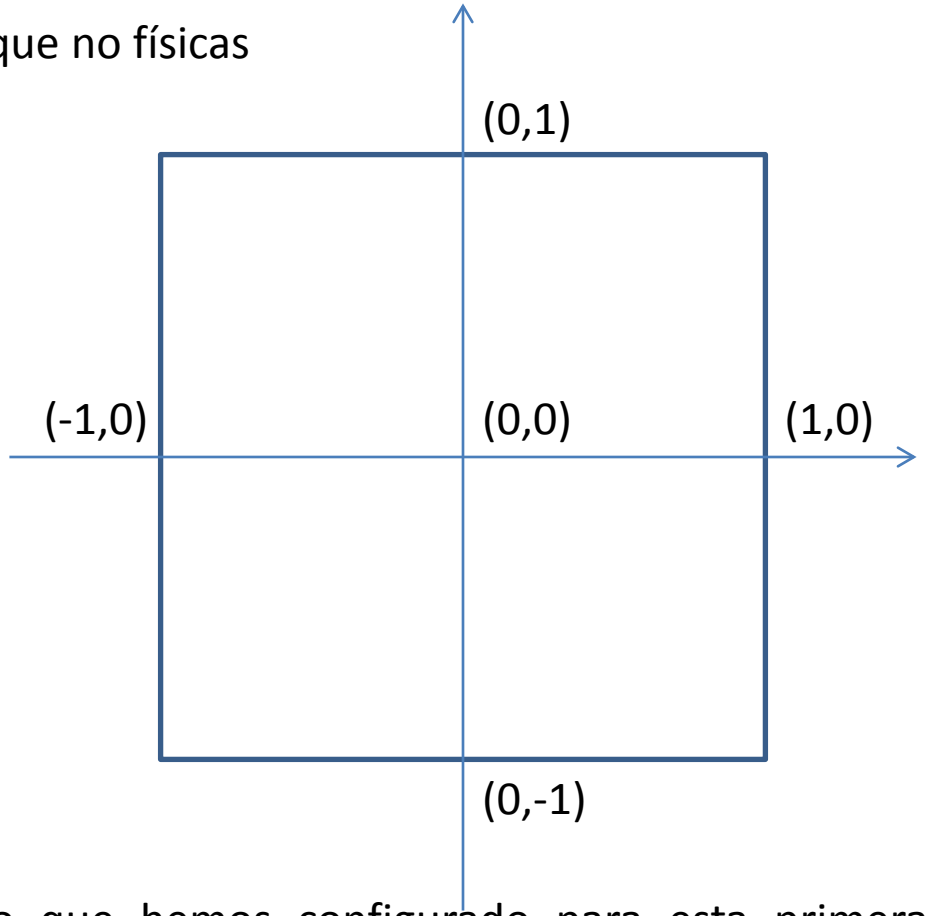
Main() prototipo

Función de inicialización sencilla con un fondo negro y unas dimensiones del espacio de trabajo de [-1,1] en ambas direcciones x e y.

```
/*
Prototipo de main.c
ULPGC, EII, Creando Interfaces de Usuario

Para aplicaciones sin animación
*/
#include <stdio.h>

#include <GL\freeglut.h>

void Init(){

    glClearColor(0.0, 0.0, 0.0, 0.0); // fondo negro
    // TO DO

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 0.0f); // espacio de trabajo

}
```

Espacio de Trabajo

Coordenadas lógicas que no físicas



El espacio de trabajo que hemos configurado para esta primera práctica tiene estas dimensiones. Al cambiar el tamaño de la ventana, se conservan las dimensiones, pero la relación de aspecto cambia. No es físicamente cuadrado en la ventana final.

Main() prototipo

Funciones prototipos sencillas para pintar (está vacía) y principal

```
void Display(){

    glClear(GL_COLOR_BUFFER_BIT); // borra todo lo existente en el framebuffer
    // TO DO


    glFlush(); // actualiza el framebuffer
}


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

    glutInit(&argc, argv);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(300,200);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutCreateWindow("OpenGL Practica xxx");
    Init();
    glutDisplayFunc(Display); // define la función de rendering


    glutMainLoop(); // bucle principal

    return 0;
}
```

Callback Display() ahora vacio.

Tamaño y posición de la ventana. Buffer sencillo de color RGA+transparencia, título de la ventana, inicialización y definición del callback de dibujo.

Entrada en un bucle de interacción.

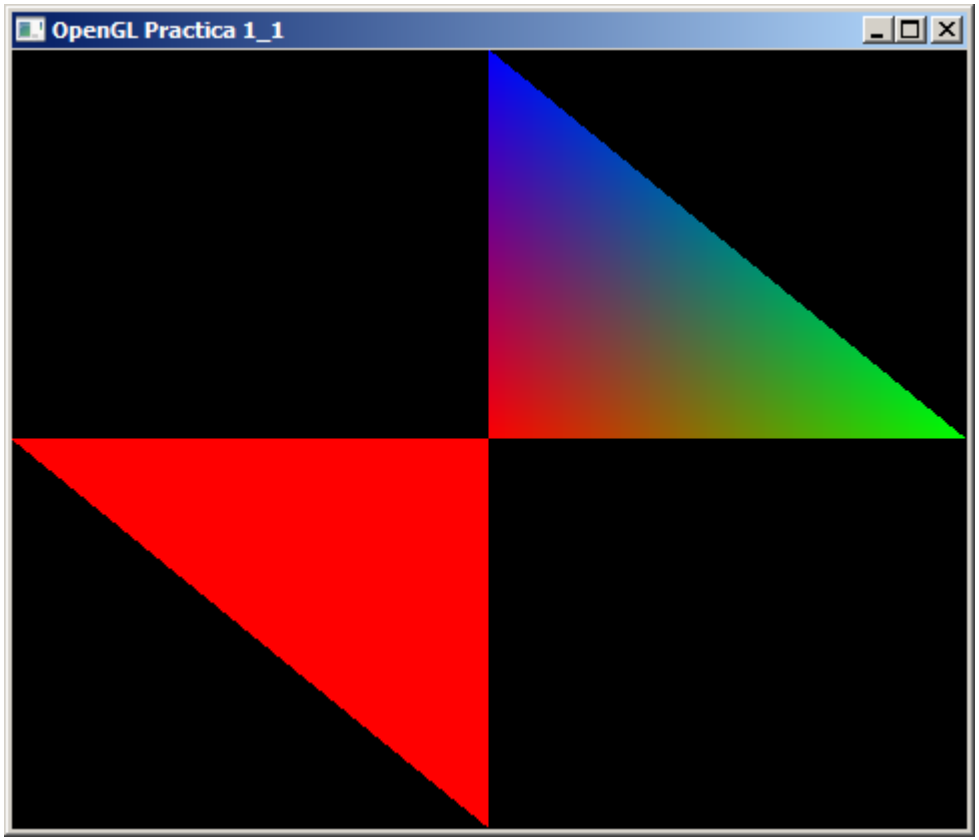
Tarea 1.

```
int main(int argc, char *argv[]){

    glutInit(&argc, argv);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(300,200);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutCreateWindow("OpenGL Practica 1_1");
    Init();
    glutDisplayFunc(Display);


    glutMainLoop();

    return 0;
}
```

Dibujo de triángulos

Callback para esta primera tarea.

```
void Display(){

    glClear(GL_COLOR_BUFFER_BIT);
    // TO DO

    glBegin(GL_TRIANGLES);
        // primer triangulo con interpolación de color
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glColor3f(0.0f, 1.0f, 0.0f);
        glVertex2f(1.0f, 0.0f);
        glColor3f(0.0f, 0.0f, 1.0f);
        glVertex2f(0.0f, 1.0f);

        // segundo triangulo con color uniforme
        glColor3f(1.0f, 0.0f, 0.0f);
        glVertex2f(0.0f, 0.0f);
        glVertex2f(-1.0f, 0.0f);
        glVertex2f(0.0f, -1.0f);

    glEnd();

    glFlush();
}
```



Funciones básicas, glxxxx

glColor3f(): define el color mediante 3 coordenadas float

glVertex2f(): coordenadas de un punto/vértice mediante 2 coordenadas float.

Una vez definido un color, todos los puntos/vértice subsiguientes conservan ese color. El color se asigna al punto. En el caso de un triangulo con colores diferentes, todo los puntos del triangulo resultan de la interpolación dependiente de los tres vértices.

Cada propiedad R(x,y), G(x,y) y B(x,y) se obtiene a partir de las correspondientes en los tres vértice mediante interpolación. Esta se puede realizar, por ejemplo, a partir de las coordenadas baricéntricas.

GL Command Syntax [2.3]

GL commands are formed from a return type, a name, and optionally up to 4 characters (or character pairs) from the Command Letters table (above), as shown by the prototype below:

return-type **Name**{1234}{b s i f d ub us ui}{v} (*[args ,] T arg1 , . . . , T argN [, args]*);

The arguments enclosed in brackets ([args ,] and [, args]) may or may not be present.
The argument type T and the number N of arguments may be indicated by the command name suffixes. N is 1, 2, 3, or 4 if present, or else corresponds to the type letters from the Command Table (above). If "v" is present, an array of N items are passed by a pointer.

For brevity, the OpenGL documentation and this reference may omit the standard prefixes. The actual names are of the forms: glFunctionName(), GL_CONSTANT, GLtype

Vertex Specification

Begin and End [2.6.1, 2.6.3]

Enclose coordinate sets between Begin/End pairs to construct geometric objects.

void Begin(enum *mode*);
void End(void);

mode: POINTS, LINE_STRIP, LINE_LOOP, LINES, POLYGON, QUAD_STRIP, QUADS, TRIANGLE_STRIP, TRIANGLE_FAN, TRIANGLES, LINES_ADJACENCY, LINE_STRIP_ADJACENCY, TRIANGLES_ADJACENCY, TRIANGLE_STRIP_ADJACENCY

Polygon Edges [2.6.2]

Flag each edge of polygon primitives as either boundary or non-boundary.

void EdgeFlag(boolean *flag*);
void EdgeFlagv(boolean **flag*);

Vertex Specification [2.7]

Vertices have two, three, or four coordinates, and optionally a current normal, multiple current texture coordinate sets, multiple current generic vertex attributes, current color, current secondary color, and current fog coordinates.

void Vertex{234}{sifd}(T *coords*);
void Vertex{234}{sifd}v(T *coords*);
void TexCoord{1234}{sifd}(T *coords*);
void TexCoord{1234}{sifd}v(T *coords*);
void MultiTexCoord{1234}{sifd}(enum *texture*, T *coords*)
void MultiTexCoord{1234}{sifd}v(enum *texture*, T *coords*)

texture: TEXTURE*i* (where *i* is [0, MAX_TEXTURE_COORDS - 1])

void Normal3{bsifd}(T *coords*);
void Normal3{bsifd}v(T *coords*);
void FogCoord{fd}(T *coord*);
void FogCoord{fd}v(T *coord*);
void Color{34}{bsifd ubusui}(T *components*);
void Color{34}{bsifd ubusui}v(T *components*);
void SecondaryColor3{bsifd ubusui}(T *components*);
void SecondaryColor3{bsifd ubusui}v(T *components*);
void Index{sifd ub}(T *index*);
void Index{sifd ub}v(T *index*);
void VertexAttrib{1234}{sfd}(uint *index*, T *values*);
void VertexAttrib{123}{sfd}v(uint *index*, T *values*);
void VertexAttrib4{bsifd ub T *values*);
void VertexAttrib4Nub(uint
void VertexAttrib4N{bsi ub T *values*);
void VertexAttribI{1234}{i u
void VertexAttribI{1234}{i u T *values*);
void VertexAttribI4{bs ubus

OpenGL Operation

Floating-Point Numbers [2.1.2]

16-Bit	1-bit sign 5-bit exponent 10-bit mantissa
Unsigned 11-Bit	no sign bit 5-bit exponent 6-bit mantissa
Unsigned 10-Bit	no sign bit 5-bit exponent 5-bit mantissa

Command Letters [Table 2.1]

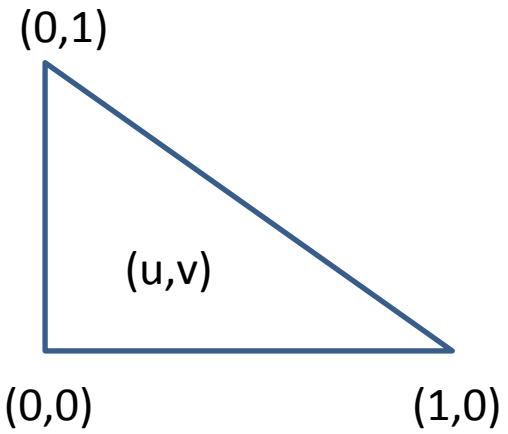
Letters are used in commands to denote types as shown below.

b - byte (8 bits)	**ub** - ubyte (8 bits)
s - short (16 bits)	**us** - ushort (16 bits)
i - int (32 bits)	**ui** - uint (32 bits)
f - float (32 bits)	**d** - double (64 bits)

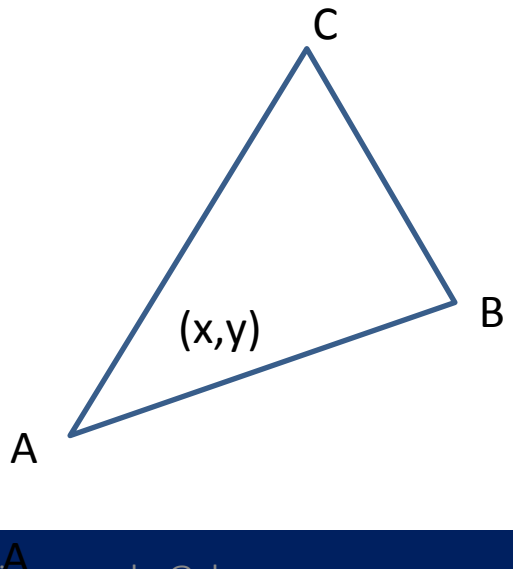
OpenGL 3.2 API Quick Reference Card

Geometría y coordenadas baricéntricas del triangulo

- Para calcular propiedades numéricas sobre los triángulos reales se introduce un modelo de triangulo rectángulo ideal referido a un sistema de coordenadas denominadas baricéntricas, tal que simplifica las operaciones numéricas.
- Coordenadas del triángulo real: (x,y)
- Coordenadas baricéntricas: (u,v)
- El triángulo real se caracteriza por tres puntos. A,B y C



$$A \leftrightarrow (0,0)$$
$$B \leftrightarrow (1,0)$$
$$C \leftrightarrow (0,1)$$



Coordenadas baricéntricas

Una interpolación lineal sobre cualquier propiedad f(x,y) se puede computar como:

$$f(u,v) = W_A(u,v)\, f_A + W_B(u,v)\, f_B + W_C(u,v)\, f_C$$

$$W_A(u,v) = 1-u-v$$

$$W_B(u,v) = u$$

$$W_C(u,v) = v$$

Las coordenadas se transforman como:

$$P(u,v) = W_A(u,v)P_A + W_B(u,v)P_B + W_C(u,v)P_C$$

$$x = (1-u-v)x_A + ux_B + vx_C$$

$$y = (1-u-v)y_A + uy_B + vy_C$$

$$\begin{pmatrix} x-x_A \\ y-y_A \end{pmatrix} = \begin{pmatrix} x_B-x_A & x_C-x_A \\ y_B-y_A & y_C-y_A \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

Coordenadas baricéntricas

Utilizando una transformación matricial entre coordenadas (x,y) y (u,v)

$$A = \begin{pmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \end{pmatrix}$$

$$\begin{pmatrix} x - x_A \\ y - y_A \end{pmatrix} = A \begin{pmatrix} u \\ v \end{pmatrix}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = A^{-1} \begin{pmatrix} x - x_A \\ y - y_A \end{pmatrix}$$

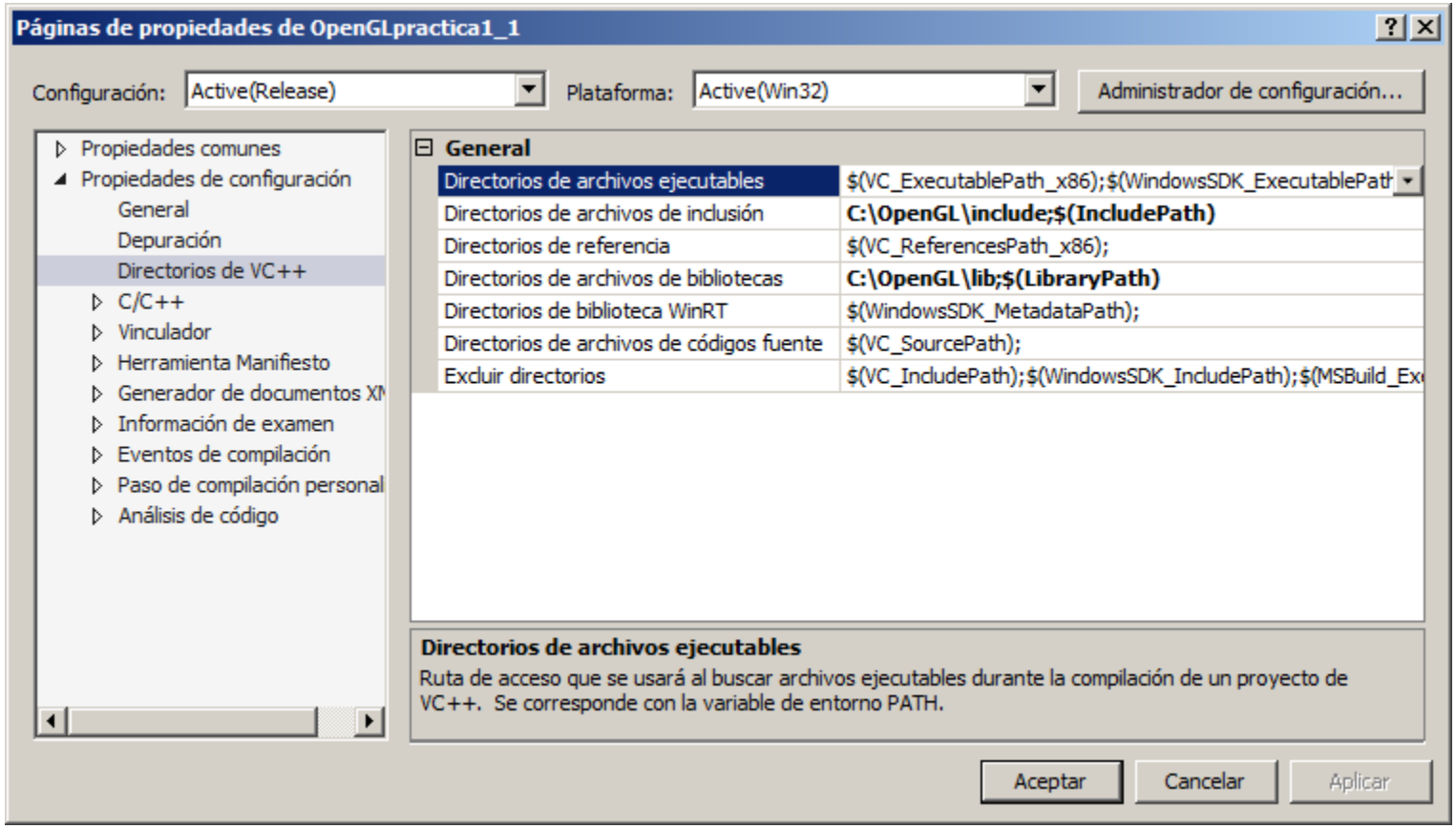
De hecho el determinante de A es dos veces el área del triángulo real.

Configurar Visual Studio 2013

Puntos

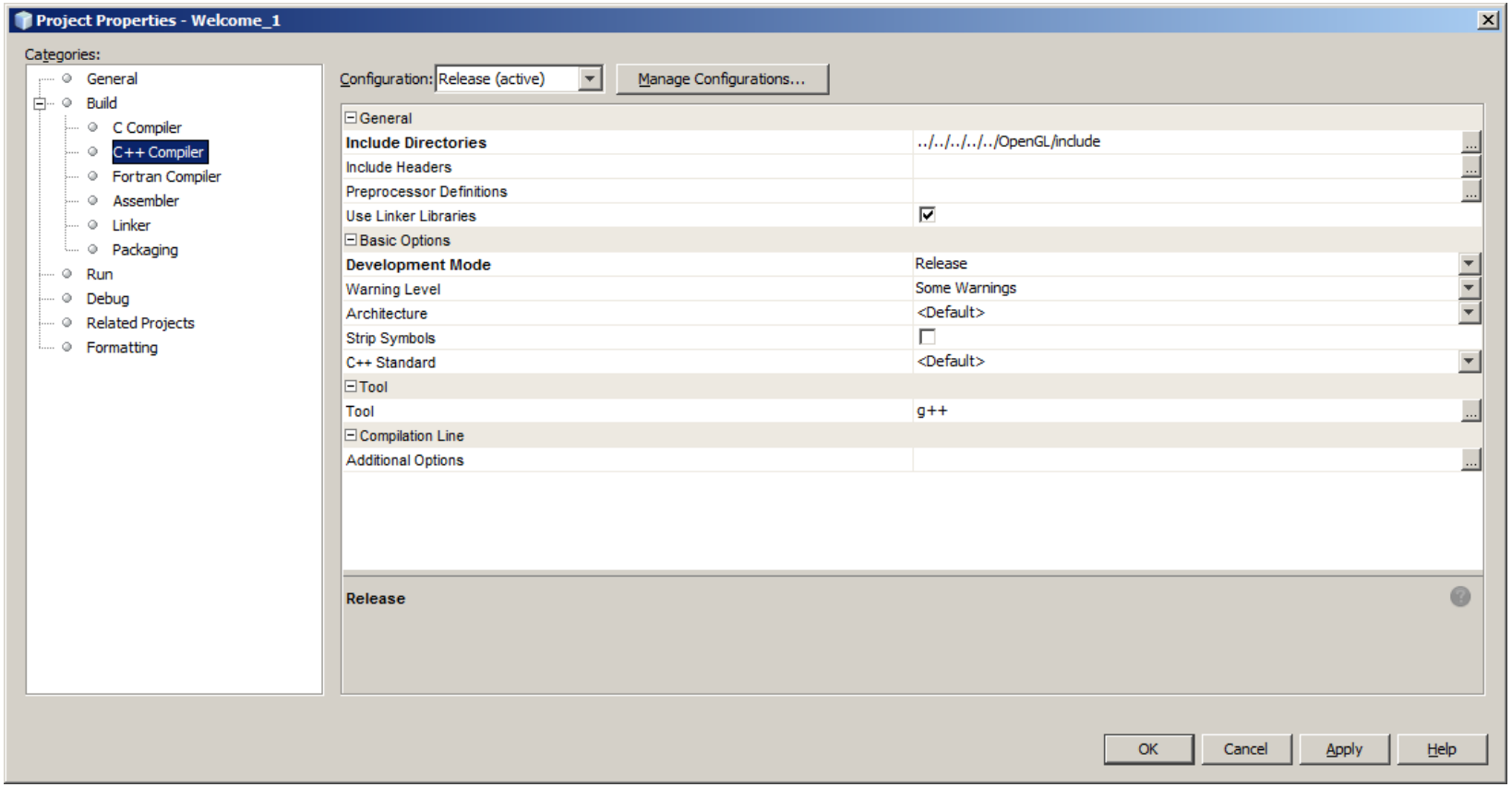
1. Localizar la distribución de cabeceras y librerías de OpenGL instaladas para ser utilizadas en la prácticas de CIU. Deberán estar en la carpeta C:\OpenGL\
2. Proyecto vacio y Configuración de Win32, Release
3. Incorporar el código prototipo y modificarlo según el ejemplo.
4. Modificar en carpeta de includes:
5. Modificar la carpeta de librerías adicionales:
6. Incluir la librería freeglut.lib
7. Compilar y ejecutar.
8. Comprender la escala del usuario con respecto al resize de la ventana.

Configuración de VS



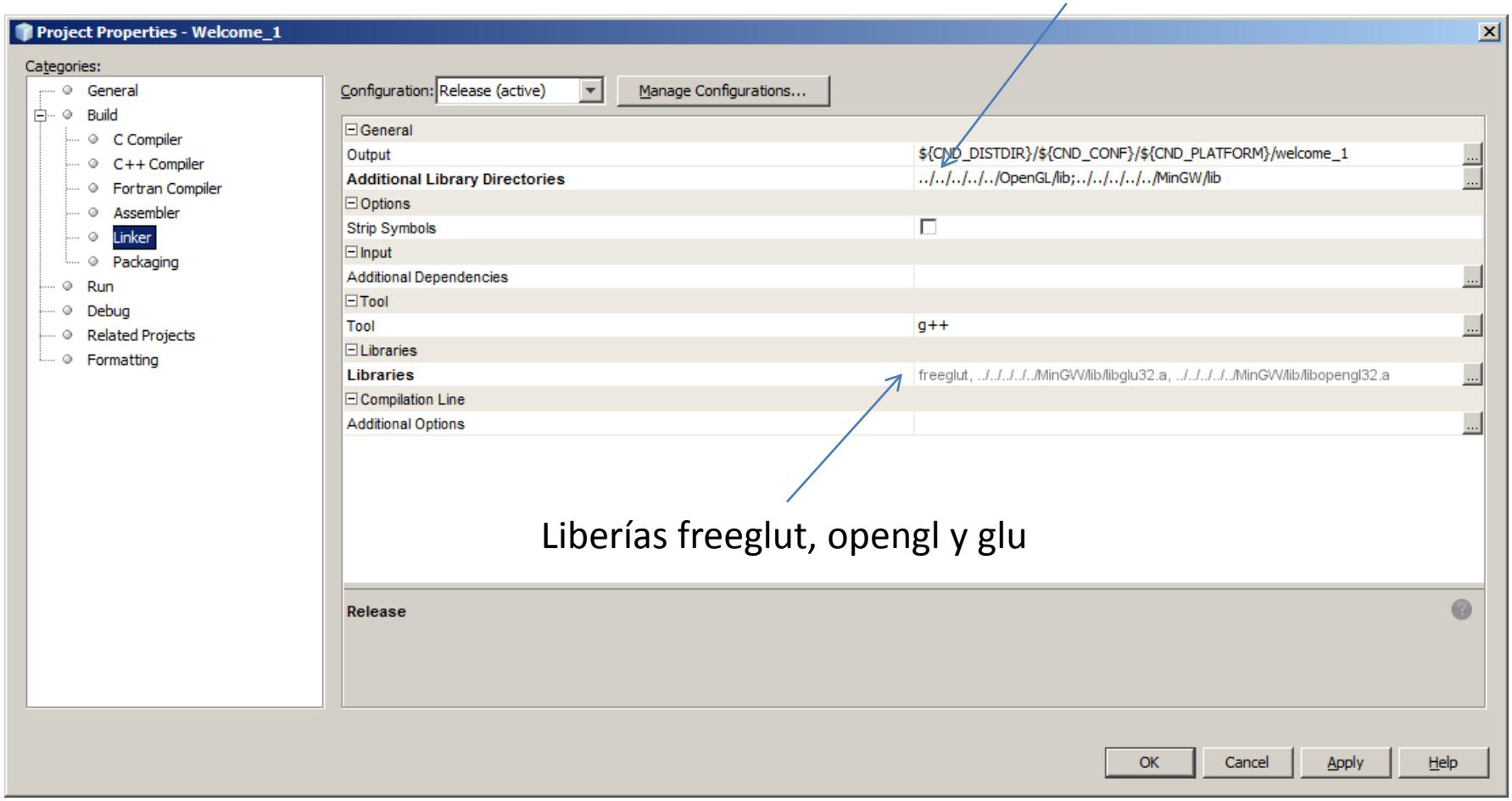
Utilización de Netbeans

También podemos utilizar Netbeans mediante la instalación de l plugin de C++, que utilizará las herramientas de compilación de gcc de MinGw. Posteriormente debemos indicar el directorio de include



Configuración Netbeans

Directorio de Librerías de OpenGL y MinGW



Liberías freeglut, opengl y glu

Que debe entregar el alumno?

- Cada alumno entregará en el Campus Virtual una memoria en PDF en la que estará contenida una descripción del trabajo realizado, incluyendo descripción, el listado C/C++ de la actividad realizada y la captura de pantalla de las gráficas o imágenes generadas.
- Para autentificar las imágenes cuando sea posible el alumno incluirá su nombre en cada ventana en el título.
- En principio la tarea quedará abierta para su entrega hasta cierta fecha que se indicará.
- Se puede trabajar en grupo en el Laboratorio, pero la memoria elaborada y entregado será individual.

Bibliografía

Reference Card: https://www.khronos.org/files/opengl-quick-reference-card.pdf

Colección Canónica de OpenGL en: https://www.opengl.org/documentation/books/

Computer Graphics through OpenGL:
http://www.sumantaguha.com/files/materials/Experimenter.pdf