# Introducción al manejo de gráficos utilizando OpenGL y Qt

Eric Orellana-Romero, Cristian Duran-Faundez
Departamento de Ingeniería Eléctrica y Electrónica, Facultad de Ingeniería
Universidad del Bío-Bío, Concepción, Chile
Email: eorellan@alumnos.ubiobio.cl, crduran@ubiobio.cl

Document Version: 1.0.0 – Oct-2012

**Resumen**

En el presente trabajo se explora la API OpenGL para el manejo de gráficos en aplicaciones informáticas. Puesto que al diseñar interfaces humano-máquina, en ocasiones, un "dibujo" o "animación" puede ser más ejemplificador para el usuario, es que se hace necesario contar con herramientas que nos permitan construirlos aprovechando las capacidades hardware de nuestro equipo. Dado esto, OpenGL entrega un adecuado nivel de abstracción para programarlas y permite generar imágenes de excelente calidad.

## 1. Introducción

Antiguamente los computadores se "comunicaban" con los usuarios mediante intérpretes de comandos, siendo utilizados en tareas específicas y por unos pocos usuarios calificados. Producto de esto, y con el fin de hacer accesible la utilización de un computador a usuarios comunes, a partir de 1980, comienza a desarrollarse la computación gráfica, generándose, en 1982, el concepto *"graphics machine"* en la universidad de Stanford, el que fue soportado por las librerías IRIS GL desarrolladas por Sillicon Graphics Inc, las que entregaban soporte de periféricos (mouse, teclado, joystick) y manejo de ventanas (fue desarrollada antes de los gestores de ventanas). Dado lo anterior, las empresas de hardware comenzaron a desarrollar los primeros circuitos específicos para gráficos, y puesto que este tipo de tecnología se encontraba en su génesis, no existía consenso respecto a como realizarlo, por lo que cada fabricante construía tarjetas gráficas según sus directivas, lo que se convirtió en un verdadero desafío para los programadores de software que pretendían desarrollar aplicaciones compatibles con hardware de diferentes empresas, siendo necesario escribir un driver para cada una, lo que implicaba una práctica costosa e ineficiente (código redundante). Debido a esta limitante, en la década del '90 comienza el desarrollo de una librería gráfica estándar abierta para el manejo exclusivo de hardware gráfico, por parte de empresas tales como Silicon Graphics Inc., Microsoft, IBM Corporation, Sun Microsystems, Digital Equipment Corporation (DEC), Hewlett-Packard Corporation, Intel e Intergraph Corporation, utilizando como base IRIS GL.

El resultado fue la publicación de la versión 1.0 de OpenGL en Enero de 1992. Dicha librería permitió estandarizar el acceso al hardware, traspasando la responsabilidad de generar una interfaz (capa intermedia) a los fabricantes, y delegar el manejo de ventanas al sistema operativo. Además, al ser abierta, permitía la adaptación a diferentes plataformas de manera sencilla, facultando a los programadores para generar aplicaciones en un lenguaje común, independiente al hardware utilizado. Ese mismo año se funda el *" OpenGL Architecture Review Board"* (ARB), organismo formado por diferentes empresas del rubro que revisaban las especificaciones de la API. En 2003 Microsoft abandona el ARB, y en 2006 es controlado por el grupo *Khronos*[1] [1], [2]. Actualmente, la API OpenGL se encuentra disponible en la versión 4.3, y se distribuye bajo licencia GPL (Licencia Pública General), siendo soportada en sistemas tales como GNU/Linux, MacOS X, Windows, PlayStation 3 y Unix. En cuanto al control y desarrollo, en la actualidad es manejada por el grupo *Khronos*, formando parte de los miembros activos Adobe, EA, Google, Hp, IBM, Mitsubishi Electronic, Mozilla, Sony, entre muchos otros. Cabe destacar que las especificaciones de OpenGL permite el manejo de gráficos, pero no incluye funciones para la gestión de

[1] `http://www.khronos.org/`

ventanas [3], por lo que esta tarea es administrada por el gestor de ventanas del sistema operativo sobre el que se esté trabajando o a través de API's desarrolladas con este fin, como por ejemplo Qt. Además, es compatible con lenguajes de programación tales como C, C++, Visual Basic, Visual Fortran y Java [4], convirtiéndola en una API poderosa y multiplataforma, lo que sumado a su gratuidad, la convierten en una de las herramientas para manejo de gráficos más utilizada, solo igualada por DirectX de Microsoft.

## 2. Principales características de OpenGL

OpenGL es una API de programación que a partir de de primitivas geométricas simples, tales como puntos, líneas y polígonos, permite dibujar escenas tridimensionales complejas. Para ello oculta al programador las características de cada tarjeta gráfica, presentando una interfaz común, pudiendo emular hardware mediante software en el caso de que este no soporte algún tipo de función. OpenGL es considerada una *máquina de estados*, por lo que cualquier características configurada será aplicada hasta que se especifique lo contrario. Por ejemplo, al crear una línea de color blanco sobre un fondo negro, cualquier otra figura creada, posterior a esto, será de color blanco sobre un fondo negro. Cada estado tiene un valor por defecto y cuenta con funciones para obtener su valor actual, habilitarlos, deshabilitarlos, etc. [2],[3]. Además de la librería GL se cuenta con librerías auxiliares que permiten el manejo de rutinas tediosas, entre ellas se encuentran [1]:

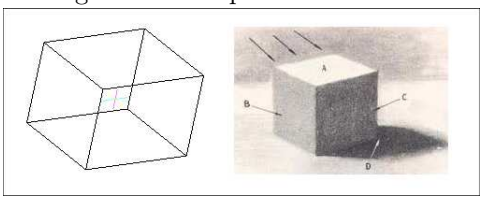
- GLU: Incluye funciones más complejas que GL, tales como definición de un cilindro o un disco con un solo comando y operaciones con matrices. Utiliza como base GL y generalmente se distribuyen de manera conjunta.
- GLUT: Es independiente de GL y permite el control de ventanas, mouse y teclado de manera transparente a la plataforma utilizada. Además incluye funciones para la creación de conos, tazas de té, etc.
- GLAUX: Implementa las mismas funcionalidades que GLUT, pero es exclusiva para plataformas Windows.
- GLX: Es utilizada en sistemas X-Windows (Linux) para redenderizar.

Por otro lado, existe la librería *MESA*, que implementa funciones equivalentes a OpenGL, pero que no puede llamarse "derivada" puesto que no a obtenido la licencia.

## 3. Objetos 3D sobre un plano 2D

Una imagen en tres dimensiones mostrada mediante un computador es realmente un conjunto de figuras dibujadas en un plano bidimensional, la pantalla. Para obtener la "ilusión" de tres dimensiones se utilizan antiguos conceptos utilizados por dibujantes, tales como la perspectiva, colores, luces y sombras. Así, a través de ellos es posible obtener la percepción de volumen, como se muestra en la figura 1.

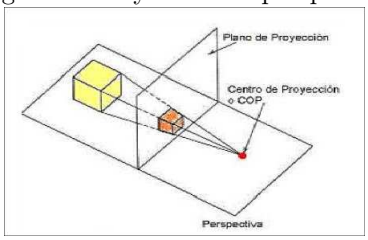
Figura 1: Percepción de volumen




Uno de los conceptos internalizados por OpenGL es la perspectiva, la que consiste en mostrar "profundidad" en la imagen mediante la modificación del tamaño de los objetos en la escena. Para ello, se utiliza un "mapeo" del plano cartesiano que muestra las ubicaciones de los objetos en la pantalla (coordenadas

$x$ e $y$), a una representación de espacio (coordenadas $x$, $y$ y $z$) a través de manipulación de matrices, concepto de *proyección en perspectiva* mostrado en la figura 2 [3].

Figura 2: Proyección en perspectiva




# 4. Instalación de OpenGL en sistemas GNU/Linux

La instalación de la API en el sistema se puede llevar a cabo de dos maneras. La primera es descargando las librerías directamente desde su página[2] o a través de los repositorios de Debian. La primera opción suele ser complicada y difícil de ejecutar, ya que no se cuenta con instaladores y deben ser compiladas total o parcialmente, mientras que al instalarlas desde los repositorios es posible utilizar *aptitude*, simplificando la complejidad. Dado esto, se utiliza la segunda opción, instalado los paquetes necesarios mediante la siguiente instrucción (notar que para instalar paquetes se debe tener permiso de administrador).

```
# aptitude install freeglut3-dev libgl1-mesa-dev libglu1-mesa-dev libice-dev libpthread-
    stubs0 libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev
     libxext-dev libxt-dev mesa-common-dev x11proto-core-dev x11proto-input-dev x11proto-
    kb-dev x11proto-xext-dev xtrans-dev
```

**NOTA:** Al realizar la instalación mediante *aptitude* es posible solo especificar el paquete *freeglut3-dev* para ser instalado, los demás son marcados como dependencias e instalados automáticamente por el gestor.

# 5. "Hola Mundo" en OpenGL mediante código

Al utilizar las librerías de OpenGL, el programa más simple que puede crearse es un triangulo contenido en una ventana. Es por esto que es nombrado como el *Hola Mundo* de la API. Por ser el primer ejemplo en ser explicado, se lleva a cabo utilizando un editor de texto, que en este caso será *gedit* y se compila manualmente desde consola. La primera función a implementar es la encargada de dibujar el triangulo, llamada *display()*, y se define según el código a continuación.

```
1 void display()
2 {
3 glClear(GL_COLOR_BUFFER_BIT);
4 glBegin(GL_TRIANGLES);
5   glColor3f(0,0,0);
6   glVertex3f(0,1,0);
7   glVertex3f(-1,-1,0);
8   glVertex3f(1,-1,0);
9 glEnd();
10 glFlush();
11 }
```

Para dibujar cualquier polígono, las funciones que indican las características de estos, deben estar contenidas entre las instrucciones *glBegin(polígono )* (indicando que tipo de polígono se utilizará) y *glEnd()*,

[2] http://www.opengl.org/resources/libraries/glut/glut_downloads.php#2

como se muestra en las líneas 5 y 10 del código anterior. Existen diversos tipos de polígonos, entre los más utilizados *GL_POINTS, GL_LINES, GL_POLYGON, GL_QUADS, GL_TRIANGLES, GL_TRIANGLES_STRIP* y *GL_QUAD_STRIP*, una explicación detallada de cada uno puede ser encontrada en [5, p. 14]. Para el ejemplo dado se utiliza *GL_TRIANGLES*, cuya función es unir cada tres vértices declarados, formando un triangulo. De este modo, mediante las instrucciones en las líneas 6, 7 y 8 se definen los vértices superior, izquierdo y derecho, respectivamente. Como se menciona anteriormente, el espacio de trabajo utilizado por OpenGL es referenciado según un sistema coordenado, con 2 o 3 dimensiones dependiendo de la aplicación. En este caso, se utiliza un plano cartesiano cuyo origen se encuentra al centro de la ventana creada. La función en la línea 6 indica el color que será utilizado en adelante, en este caso, negro. *glColor3f(Rojo,Verde,Azul)* genera un color a partir de la paleta RGB, debiendo indicarse la cantidad que debe ser agregada de cada uno, en un intervalo de 0 (nada) a 1 (todo). Un aspecto importante es que OpenGL considera que los polígonos son dibujados creando sus vértices en sentido anti-horario, por lo que según esa convención se define el lado frontal del mismo (lado a dibujar). Mediante la instrucción en la línea 3 se libera el buffer que almacena el polígono mientras se crea (para evitar "parpadeos"). Por último, mediante la instrucción en la línea 10 se ejecuta cualquier operación pendiente en la pila. De este modo, la figura a dibujar está completamente especificada. A continuación, ya que no se trabaja con un gestor de ventanas externo, se muestra la función encargada de gestionar el reescalado de la ventana que contiene al espacio de trabajo.

```
1 void reshape(int width, int height)
2 {
3 glViewport(0,0,width,height);
4 glMatrixMode(GL_PROJECTION);
5 glLoadIdentity();
6 glMatrixMode(GL_MODELVIEW);
7 glLoadIdentity();
8 }
```

Esta función es accedida cada vez que se reescala la ventana que contiene los objetos creados mediante OpenGL, de esta forma, es redibujada cada vez que sea necesario. La función recibe 2 atributos como entrada, los que indican cual será el nuevo tamaño de la ventana. Las instrucciones restantes indican características del espacio de trabajo. Así, mediante la instrucción en la tercera línea, se especifica la porción del plano en el que puede ser utilizado por OpenGL, en este caso todo (desde 0 hasta 'ancho' y desde 0 hasta 'alto'). Mediante las instrucciones en las líneas 4 y 5, se carga la matriz identidad (elemento neutro en la multiplicación) a la matriz de proyección, y a través de las instrucciones en las líneas 6 y 7 se carga la matriz identidad a la matriz de transformación, de esta manera nos aseguramos de no generar transformaciones al polígono en construcción. Cualquier punto creado en el plano es multiplicado por estas matrices antes de ser dibujado, por lo que mediante su modificación varía la proyección (perspectiva) de un objeto, matriz de proyección, o se define alguna transformación (escalado, traslación, etc.), mediante la matriz de transformaciones [5]. Para concluir el ejemplo, solo resta crear la ventana y dibujar el triangulo que propusimos como objetivo, el código asociado a esto es mostrado a continuación.

```
1 #include <GL/glu.h>
2
3 int main(int argc, char **argv)
4 {
5 glutInit(&argc, argv);
6 glutInitWindowPosition(50,50);
7 glutInitWindowSize(200,200);
8 glutCreateWindow("I Hello OpenGL");
9 glClearColor(1,1,1,0);
10 glutDisplayFunc(display);
11 glutReshapeFunc(reshape);
12 glutMainLoop();
13 return 0;
14 }
```

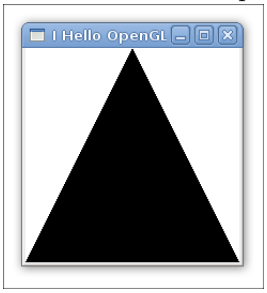
Así, mediante la primera instrucción es agregada la librería necesaria, GLU. A partir de la línea 3 hasta la 9 se indican las características de la ventana a crear, las que resultan bastante intuitivas. Es importante mostrar que mediante la línea 7 se define el color base de la ventana, en este caso blanco, y que a través de las líneas 8 y 9 se indican a la librería las funciones creadas anteriormente. Una vez realizado esto, mediante la función *glutMainLoop()*, se entrega el control de la aplicación a la librería y se dibuja el objeto creado.

Para compilar este proyecto, se utiliza *gcc* y es necesario indicar la librería utilizada, como se muestra en la instrucción a continuación.

```
$ gcc triangulo.c .o triangulo -lglut
```

Una vez compilado se genera un archivo binario llamado *triangulo*, el que la ser ejecutado produce la ventana mostrada en la figura 3.

Figura 3: Hola Mundo OpenGL




# 6. Integración de OpenGL a Qt Creator

Como se menciona anteriormente, la librería GL del API de programación OpenGL entrega soporte para el diseño de objetos a partir primitivas básicas, pero no implementa funciones para el manejo de ventanas que las contengan, notar que en el ejemplo anterior todas las instrucciones utilizadas para manejo de ventanas comienzan con el prefijo *glut*. Dado esto, puede ser utilizada la librería GL en conjunto con Qt para generar interfaces gráficas de mejor calidad. Es una buena práctica utilizar el framework Qt en vez de la librería GLUT, ya que GLUT al ser una librería de propósito general genera ventanas independientes, Qt, por otro lado, integra los gráficos generados por GL en las interfaces desarrolladas. Para explicar los pasos necesarios en esta tarea se utiliza el mismo ejemplo anterior, incluso se reutiliza el código generado, pero esta vez utilizando los IDE Qt Creator y Qt Designer. Para ello se debe generar un proyecto, como se explica en el tutorial anterior [6], pero esta vez, se debe agregar el módulo *QtOpenGL*. En este caso, se utilizará el nombre `Triangulo` para la clase principal. Una vez finalizado el proceso se habrán generado los archivos *triangulo-Qt.pro* (dentro del cual puede verificarse que se agregó el módulo de OpenGL), *triangulo.ui, triangulo.h, triangulo.cpp* y *main.cpp*. Luego, se eliminan los widgets innecesarios de *triangulo.ui*, y se agrega un objeto de la clase (widget) *QWidget*, el que más tarde alojará las figuras creadas mediante la librería GL. Para continuar, se crea una nueva clase llamada `VentanaOGL`, que se utilizará para crear las figuras con OpenGL. Dicha clase hereda las características de `QGLWidget`. Para esto, mediante el menú *File→New File or Project*, se crea una nueva clase C++ (*C++ class*), la que es llamada `VentanaOGL` y se indica que tiene como base la clase `QGLWidget`[3] (seleccionar en *Type information* `inherits QObject`). Al finaliza este proceso, se habrán creado dos nuevos archivos llamados *ventanaogl.h* y *ventanaogl.cpp*, los que deben ser modificados como se muestra en los cuadros 1 y 2.

Una vez escritos ambos ficheros, en *triangulo.ui* se debe seleccionar el widget `QWidget` antes creado y haciendo click derecho sobre él y seleccionar la opción *promoted to ...*, una vez desplegada la ventana, se debe seleccionar en el campo *Base class name*, `QWidget` y en *Promoted class name*, `VentanaOGL`, de este modo se actualiza automáticamente el campo *Header file*. Luego, se presiona el botón *Add* y *Promote*.

[3] `http://qt-project.org/doc/qt-4.8/QGLWidget.html`

Cuadro 1: Definición clase VentanaOGL, ventanaogl.h

```
#ifndef VENTANAOGL_H
#define VENTANAOGL_H

#include <QGLWidget>

class VentanaOGL : public QGLWidget
{
public:
    explicit VentanaOGL(QWidget *parent = 0);

    void initializeGL();
    void paintGL();
    void resizeGL(int w, int h);
};

#endif // VENTANAOGL_H
```

Cuadro 2: Implementación clase VentanaOGL, ventanaogl.cpp

```
#include "ventanaogl.h"

VentanaOGL::VentanaOGL(QWidget *parent) :
    QGLWidget(parent)
{
}

void VentanaOGL::initializeGL()
{
    glClearColor(1,1,1,0);
}

void VentanaOGL::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
      glColor3f(0,0,0);
      glVertex3f(0,1,0);
      glVertex3f(-1,-1,0);
      glVertex3f(1,-1,0);
    glEnd();
    glFlush();
}

void VentanaOGL::resizeGL(int w, int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

Siguiendo estos pasos se crea el objeto de `QWidget` como una instancia de `VentanaOGL`. Para finalizar, se compila el proyecto y se obtiene la ventana mostrada en la figura 3, obteniendo el mismo resultado que en el apartado anterior.

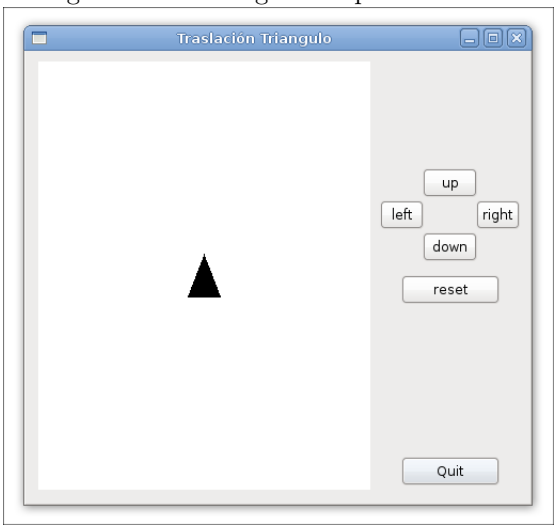
Cabe destacar que siguiendo esta metodología no es necesario utilizar la librería GLUT, no se utiliza

ninguna función con el prefijo *glut*, ya que esta tarea es realizada por la API Qt.

# 7. Aplicación final

En esta sección se utilizarán los conceptos adquiridos para crear una interfaz gráfica que maneje una figura creada con Qt. Puesto que ya se implementó un triangulo en los ejemplos anteriores, se agregará funcionalidad a la aplicación creada en la sección 6, para ello es necesario generar una ventana como la mostrada en la figura 4. A través de ella, se pretende dibujar un triangulo al centro del área manejada por OpenGL y trasladarlo mediante los botones *up, down, right, left* y *reset*.

Figura 4: Interfaz gráfica aplicación final




Para realizar este cometido, se utilizan las funciones *glTranslatef(float x, float y, float z)* y *updateGL()*. Estas permiten manejar la posición en la que se dibuja un objeto y redibujarlo, respectivamente. Al dibujar cualquier objeto mediante la librería GL, este es posicionado al centro del espacio de trabajo, por lo que para ubicarlo en otro lugar es necesario agregar un *offset* en la dirección requerida, lo que es logrado a través de la modificación de la matriz de transformaciones (GL_MODELVIEW). Para ello, GL implementa la función *glTranslatef(...)*, la que modifica esta matriz en función de los parámetros requeridos, *offset* en el eje *x, y* y *z*. De esta forma, se modifica el código mostrado en los cuadros 1 y 2, agregando una método para manejar el traslado del objeto y los slot requeridos para generar la funcionalidad de los botones.

Como puede observarse en el cuadro 4 se genera un método llamado *translate*, el que utiliza dos atributos para especificar el *offset* requerido en la dirección deseada. Así, este método modifica la matriz de transformaciones y redibuja el triangulo en su nueva posición. Además, se genera un slot para cada dirección y para resetear la ubicación. Los slots en las líneas 9, 10, 11 y 12 utilizan el método antes explicado para modificar la posición del triangulo en la dirección necesaria, mientras que *reset* "carga" la matriz identidad en la matriz de transformaciones y redibuja, ubicando nuevamente al triangulo al centro del espacio de trabajo. Cabe destacar que los cambios de posición intoducidos mediante *glTranslatef(...)* son acumulativos, lo que permite generar el movimiento deseado.

Una vez generados los métodos que modifican la figura creada con OpenGL, es necesario definir las conexiones entre los botones y los widgets. Estas definiciones se agregan al constructor de la clase *Triangulo*, recordar que es la base de esta aplicación (contiene a todos los widgets agregados). Así, en la línea 8 del cuadro 5 se "conecta" el botón de salida (quit) con la clase *Triangulo*, cerrando la aplicación cuando este es presionado [6]. Mediante los restantes se conectan los diferentes botones de dirección y reset al widget *widget* (contiene el espacio de trabajo para dibujar con OpenGL) ejecutando los slots definidos anteriormente al presionarlos.

Cuadro 3: Modificación ventanaogl.h

```
...
class VentanaOGL : public QGLWidget
{
    Q_OBJECT
...
protected:
    void translate(float x, float y);

public slots:
    void go_up();
    void go_down();
    void go_right();
    void go_left();
    void reset();
...
```

Cuadro 4: Modificación ventanaogl.cpp

```
...
void VentanaOGL::translate(float x, float y)
{
    glTranslatef(x, y, 0);
    updateGL();
}

// slots
void VentanaOGL::go_up() { translate(0,0.1); }
void VentanaOGL::go_down() { translate(0,-0.1); }
void VentanaOGL::go_right() { translate(0.1,0); }
void VentanaOGL::go_left() { translate(-0.1,0); }
void VentanaOGL::reset()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    updateGL();
}
```

Una vez realizados estos cambios al código y compilada la aplicación, esta es completamente funcional, modificando la posición del triangulo al presionar los botones de dirección, y cerrando la aplicación al presionar Quit.

# Referencias

[1] F. S. García. Introducción a opengl. website. Revisada el 15 Octubre de 2012. [Online]. Available: http://usuarios.multimania.es/andromeda_studios/paginas/tutoriales/tutgl001.htm

[2] Wikipedia. Opengl. Revisada el 15 de Octubre de 2012. [Online]. Available: http://es.wikipedia.org/wiki/OpenGL

[3] D. H. G. Centanaro, *Integración de OpenGL en Qt*, Universidad Carlos III de Madrid, Noviembre 2011, trabajo de Título.

[4] J. D. de la Calle and A. J. Villanueva, *Una aproximación a OpenGL*.

Cuadro 5: Conexión entre widgets, triangulo.cpp

```
...
Triangulo::Triangulo(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Triangulo)
{
    ui->setupUi(this);
    ui->quit->setFocus();
    QObject::connect(ui->quit,SIGNAL(clicked()),this,SLOT(close()));
    QObject::connect(ui->up,SIGNAL(clicked()),ui->widget,SLOT(go_up()));
    QObject::connect(ui->down,SIGNAL(clicked()),ui->widget,SLOT(go_down()));
    QObject::connect(ui->right,SIGNAL(clicked()),ui->widget,SLOT(go_right()));
    QObject::connect(ui->left,SIGNAL(clicked()),ui->widget,SLOT(go_left()));
    QObject::connect(ui->reset,SIGNAL(clicked()),ui->widget,SLOT(reset()));
}
...
```

[5] O. García and A. Guevara, *Introducción a la Programación Gráfica con OpenGL*, Escola Tècnica Superior d'Enginyeria Electrònica i Informàtica La Salle, Enero 2004.

[6] E. Orellana-Romero and C. Duran-Faundez, *White Paper: Introducción al Desarrollo de GUI's mediante Qt*, Departamento de Ingeniería Eléctrica y Electrónica, Facultad de Ingeniería, Universidad del Bío-Bío, Concepción, Chile, Octubre 2012.