Mesh and Computational Geometry
Master 2 Informatique ID3D et ECL – 2023-24
Raphaëlle Chaine

TP0 – Getting started with QT Creator

Qt is a set of cross-platform libraries with a W.O.R.A. objective (Write Once, Run Anywhere). Therefore, apart from a few small modifications, you should be able to compile and execute your code in the main operating systems (desktop and mobile).

Although Qt can be used in a wide range of applications, it is mainly used for programming software with GUI (Graphical User Interface), because of the simplicity offered by its IDE, Qt Creator, to create and edit windows and other graphics elements.

Preamble: Today you will get an example tarball for your project and install Qt5 or Qt6. The provided files were developed using qt5 but you can upgrade them to qt6 following the 8 tags “UpgradeQt6” in the files Mesh_Computational_Geometry.pro and gldisplaywidget.h.cpp and mesh.h.

Installation (Qt + Qt Creator) - Linux, Windows, Mac (Open Source)

http://www.qt.io/download-open-source/
Qt is huge. Don’t install everything, only Design tools et Environnement Bureau.

For those who prefer to use *apt-get* on Linux :

- apt-get install qt5-default
- apt-get install libqt5opengl5-dev
- apt-get install qtcreator

If you install qt6, the apt-get commands are similar.
Make sure you have a C++ compiler (Gnu gcc).

Installation of qt and qtcreator under windows :
Check that you have a C++ compiler (with MinGW compiler or with Visual Studio msvc2019_64)

For those who prefer to use *brew* to install packages on Mac OS (after making sure that usr/local/Cellar and /usr/local/Caskroom are not restricted to admin installation) :

- brew install qt5
- brew cask install qt-creator

If you install qt6, the commands are similar.

Example project

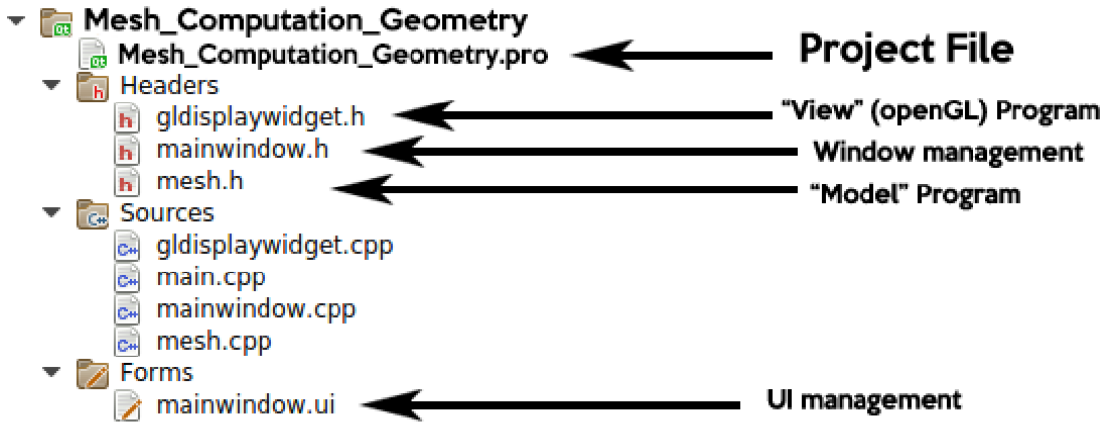
Get the example tarball and open Mesh_Computational_Geometry.pro with QtCreator.

- Comment/uncomment some lines in the file Mesh_Computational_Geometry.pro to correct the information specific to the machine and compiler, depending if you work on MacOS, Linux or Windows.
- Click on configure, then build and execute the program.
- When configuring the project, you may have to choose one kit.
- If the C++ compiler was not found automatically, check that you have a g++ compiler and add it into qt Creator using manage kits (click on Projets in the left menu to find this option)
- The deployment folder (Built) should be distinct of the Code folder.

Compile and run the project.

- If there is an error when compiling programs using the provided archive: You may have to replace #include <glu.h> by #include <GL/glu.h> in gldisplaywidget.h

Project structure



mainwindow : Description of the events (slot) that can occur in the main window, possibly in association with signals that are being received by widgets (all the widgets should be descendant of mainwindow)

ui : xml file describing Qwidgets. Modifying this file using Qt designer in Qt creator adds new features to the *mainwindow* module. It is also possible to edit this xml file manually.

During compilation, Qt transforms the Qt C++ code into another source code (located in *Built*), in which Qt macros have been processed. Thus, all widgets that have been defined as *MainWindow* descendants are also available via the *ui* pointer corresponding to an attribute of the window. *Connect* is a static function of the window.

Where to add your changes?

In this project you will handle meshes of 3D object surfaces or meshes 2D domains that will be displayed using (old fashion) OpenGL.

1) The geometric data structures and operations are to be added in ***mesh.h*** et ***mesh.cpp*** (we will work on those files during the lessons). For now, the mesh module only contains the class GeometricWorld in which you will add instances of meshes.

2) Regarding OpenGL and the construction of an image of the scene, your changes can be added in ***glDisplayWidget.h*** et ***glDisplayWidget.cpp***

Note: The class *GLDisplayWidget* offered by the *glDisplayWidget* module contains a data member of type GeometricWorld and the function *GLDisplayWidget::paintGL* invokes the function *draw* of that GeometricWorld. The area delimiting the scene to be displayed is specified in *GLDisplayWidget::resizeGL*.

3) Graphical Interface: ***mainwindow.ui***, ***mainwindow.h***, ***mainwindow.cpp***

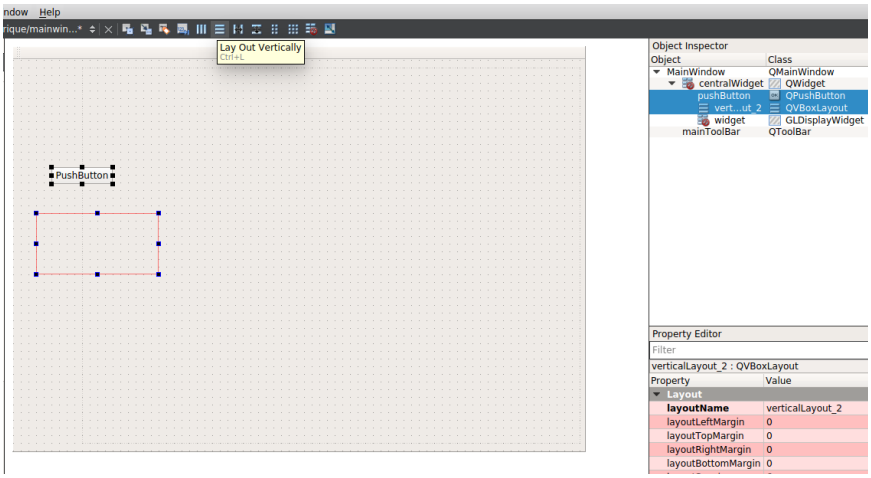
You should know that in Qt, objects are not only equipped with attributes and member functions or methods, but also with signals and slots. Slots are methods connected to signals. Thus, it will be possible to connect a slot of one object to the signal of another.

Customizing your interface with buttons

1) Edit ***mainwindow.ui*** in qt-creator.

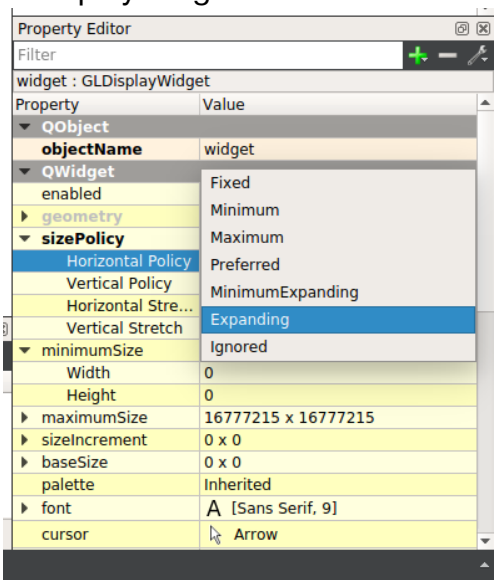
2) Creating an area for the interface:

- Add a Vertical Layout and a PushButton, select both and click at the top of your screen on the Vertical Layout icon like this:



- Click on CentralWidget in the tree on the right and select a horizontal layout. CentralWidget is parent of both the interface and display widgets.

- Then change Horizontal Policy from sizePolicy to Expanding for the GlDisplayWidget.



-

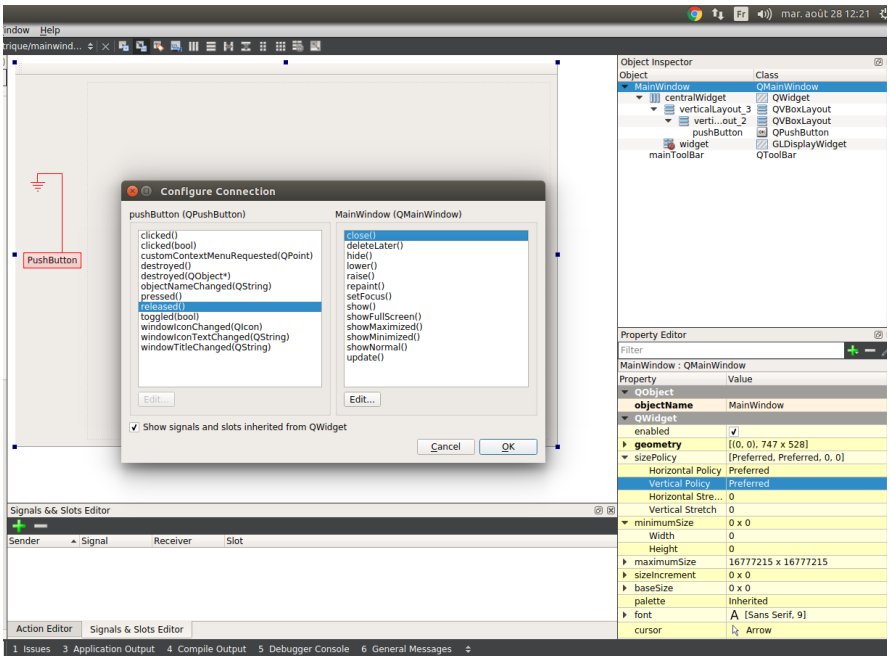
3) Customizing the pushButton :

- Double click on the button to change its text to "exit".

You can use three different approaches to associate slots (events) when the button is pressed.

Method A:

- Click on "Edit Signals/Slot" at the top and select the exit button as if you were moving it above (see picture).
- Check ''show signals'' then select *released()* in association with *close()*. This way, the *released()* signal of the pushButton is connected to the closed() slot of the *MainWindow*.



In order to return to the edit mode click on "edit widget" at the top.

Method B:

After creating and selecting a pushButton (or a Checkbox), use goToSlot (right click to access it) to associate one of its signals to a slot.

You then define the desired action in the generated code skeleton for the slot (here in mainWindow class).

```
// mainwindow.cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

// Code generated by goToSlot
void MainWindow::on_checkBox_clicked(bool checked)
{
    ... code of the slot
    for example ui->widget-> _geomWorld … ()
// where widget is the GLDisplayWidget which manages the display in
//the main Window and that has an access to the GeometricWorld
}
```

Method 3 :

- If you prefer, you can also create new slots and connect them to signals by editing the files by yourself.

Add a **public slot** to a Qt object (here in the mainWindow class).

```
// mainwindow.h
...
public slots:
        void onButton();
...
```

```
// mainwindow.cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

void MainWindow::onButton()
{
        ... code of the action to be performed
        For example ui->pushButton->setText("Released");

 }
```

Connection of the **signal** to the **slot**.

```
MainWindow::MainWindow(QWidget *parent) :
        QMainWindow(parent),
        ui(new Ui::MainWindow)
{
        ui->setupUi(this);
        connect(ui->pushButton, SIGNAL(released()), this, SLOT(onButton()));
}

// ui is an object that contains all the widgets of the form
```

Now you are ready for creating a new button to select a display mode for your GeometricWorld (and your meshes). This will trigger the use of _geomWorld.draw() **or** _geomWorld.drawWireFrame() **in** GLDisplayWidget::***paintGL***()**. You can also create a button to load the data structure (instead of doing it in** GLDisplayWidget::***initializeGL***()).

Keyboard shortcuts

- Ctrl + r : Run
- Ctrl + b : Build
- Ctrl + space : to complete a variable or function name
- F4 : Switching from .hpp to .cpp

Note: Some OpenGL instructions used in this archive correspond to a version of OpenGL that has become obsolete since shader programming changed tremendously. However, the example has the advantage of being simple to understand, and you will be able to make it evolve in Image Synthesis courses. Since Qt4, the QtOpenGL module provides the class QGLWidget and its derivatives (all prefixed by QGL).