Mesh and Computational Geometry
Master 2 Informatique ID3D – 2021-22
Raphaëlle Chaine

TP1 – Getting started with QT Creator, Meshes and Data Structures

Qt is a set of cross-platform libraries. This type of library meets a W.O.R.A. objective (Write Once, Run Anywhere). This means that 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 tested 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/

Installation of qt and qtcreator under linux :

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

If you install qt6, the apt-get commands should be similar.

Installation of qt and qtcreator under windows (with MinGW compiler or with Visual Studio)

- Follow the instruction on qt website. No need to install all the packages.

Installation of qt and qtcreator under Mac

- Use brew and brew cask (brew install qt5 et brew cask install qt-creator)

- Make sure that usr/local/Cellar and /usr/local/Caskroom are not restricted to admin installation!
- If you install qt6, the commands should be similar.

Example project

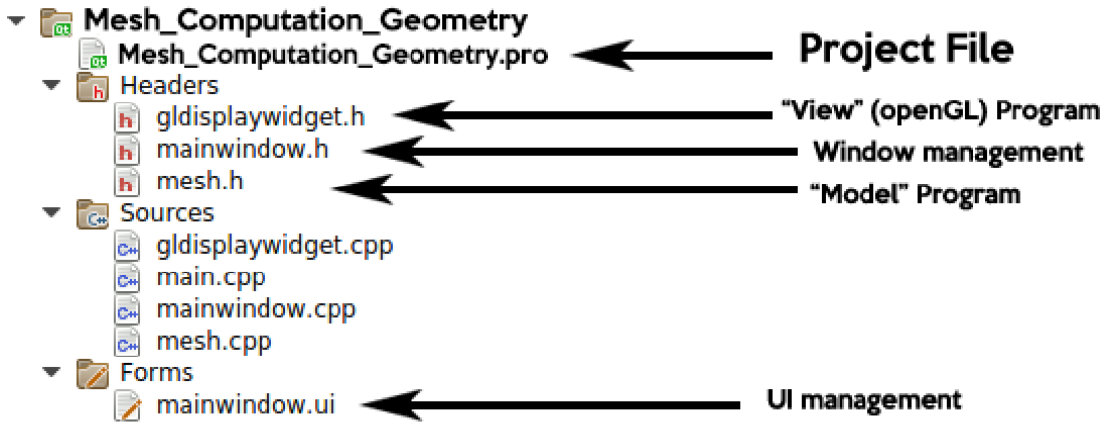
Get the example tarball and open the project example from the .pro file.

- 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.
- Try to configure the project with the proposed kits.
- 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)
- 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 associated with signals that are being received by widgets (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 module 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 build meshes of 3D/2D objects that will be displayed using OpenGL.

1) Regarding OpenGL and the construction of an image of the scene: ***glDisplayWidget.h*** et ***glDisplayWidget.cpp***

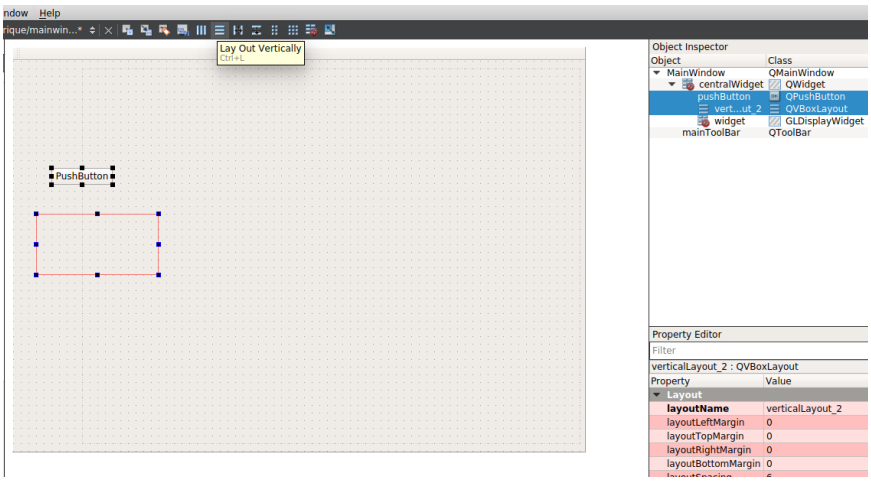
2) Definitions of geometric and graphic data: ***mesh.h*** et ***mesh.cpp***. The class *GLDisplayWidget* offered by the module *glDisplayWidget* contains a data member of type GeometricWorld and the function *GLDisplayWidget::paintGL* invokes the function *draw* of that GeometricWorld. The area delimiting 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

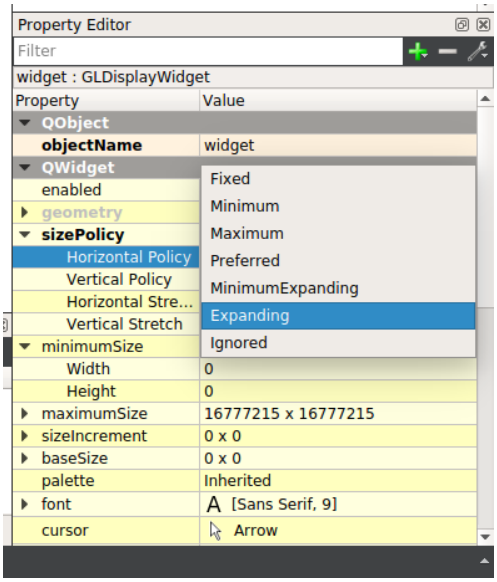
1) Edit ***mainwindow.ui*** in qt-creator.

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:



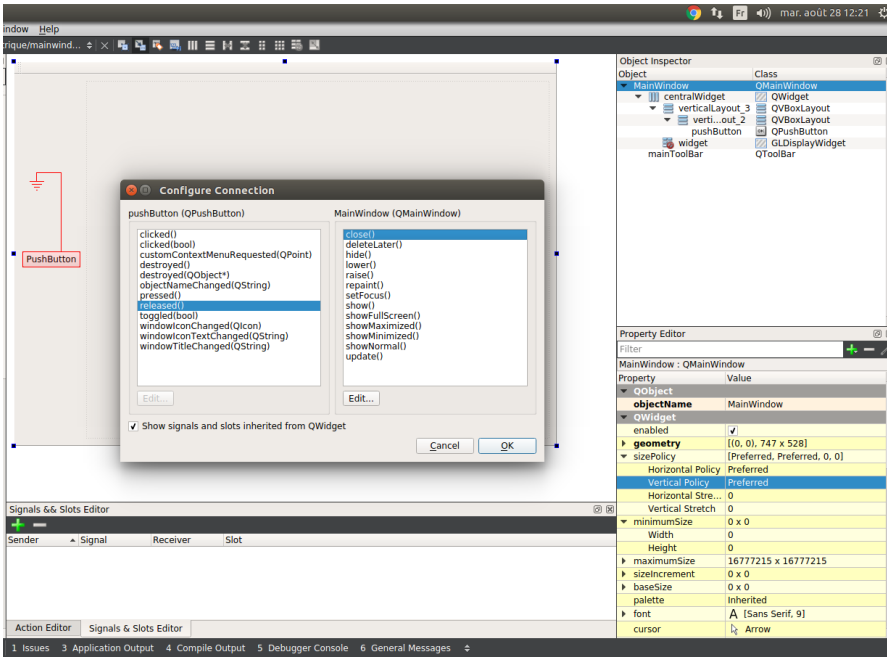
- Click on CentralWidget in the tree on the right and select a horizontal layout. CentralWidget is parent of both the interface and display widgets.

- 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".
- 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.

Note :

- You can create and select a pushButton (or a Checkbox)... then use goToSlot to associate one of its signals to a slot : **you will further need it to create a button to select a display mode for your mesh for example**. 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-> … ()
// where widget is the GLDisplayWidget which manages the display in
//the main Window
}
```

- 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

Keyboard shortcuts

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

Creating or loading a topological mesh

1) Write the code of the triangulated mesh data structure with geometry and connectivity information.
   - First, we will model a mesh by a vector of its vertices and a vector of its faces. In this first model, the faces are not attached together. They are simply represented by the indexes of their 3 vertices.
   - Then set up the data structure corresponding to the topological model based on vertices and faces that we saw in class. In this model, the faces are attached together.

2) Construction of elementary meshes to test your data structures.
   - A tetrahedron (be careful when attaching the faces together)
   - A pyramid with a square base
   - A 2D bounding box (composed of 2 triangles) whose edges are connected to an artificial “infinite” vertex at the back.

3) Set up a menu with some buttons to change the display mode (wireframe, plain face).

4) Facultatif : Write a routine to read and load, in your mesh data structure, a triangulated mesh written in an OFF file (queen.off is a nice example of such a mesh made by an artist here)
Format OFF :
   - Number of vertices s
   - Number of faces c
   - Description of the faces (sequence of the indices of the vertices of the face, preceded by its number of vertices).

Ensure that you implement a careful software approach.

Note: Some OpenGL instructions used in the 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. In Qt4 and Qt5, the QtOpenGL module provides the class QGLWidget and its derivatives (all prefixed by QGL).