

Pràctica número 0

Tema: Introducció als gràfics

Objectiu: Visualització de models poligonals amb Qt i OpenGL utilitzant shaders

La pràctica té per objectiu que us familiaritzeu amb el software de pràctiques escrit en C++: **QtCreator**, **Qt**, **OpenGL** i **shaders** i que aprengueu l'arquitectura d'una aplicació gràfica visualitzant un model poligonal utilitzant **QtCreator** i programant amb **OpenGL** i **C++**.

La pràctica es compon de 3 exercicis:

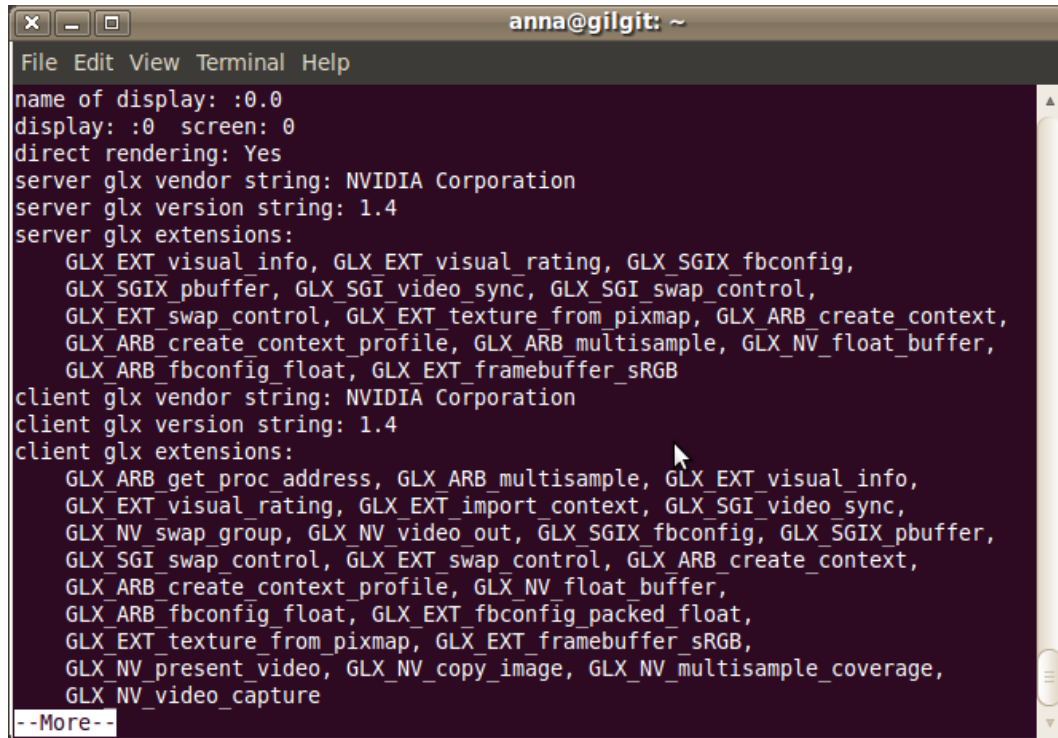
- familiarització amb Qt,
- modelat d'un model poligonal amb OpenGL
- visualització del model mitjançant shaders (glsl)

En aquest document primer s'indica com detectar la instal·lació bàsica de OpenGL en entorns Linux i les versions que estan actualment instal·lades a l'Aula IA. A més a més, s'explica com fer una aplicació Qt. Després s'explica com generar un primer exemple guiat (HelloQt). Després, es detalla com crear una nova aplicació amb un widget que permeti visualitzar GL, basant-nos en l'exemple anterior. Finalment, es donen les indicacions per a incloure el model poligonal a visualitzar en l'entorn i com visualitzar-lo amb OpenGL directament, mitjançant l'arquitectura clàssica de les aplicacions gràfiques) o bé amb shaders (programant amb el llenguatge glsl, segons l'arquitectura de les noves aplicacions gràfiques). Com a pràctica, heu de desenvolupar la visualització d'un model d'esfera i la seva trajectòria en l'espai.

Nota: Intenteu respondre les preguntes que es fan a cada apartat per a validar que heu entès tota l'arquitectura de les aplicacions.

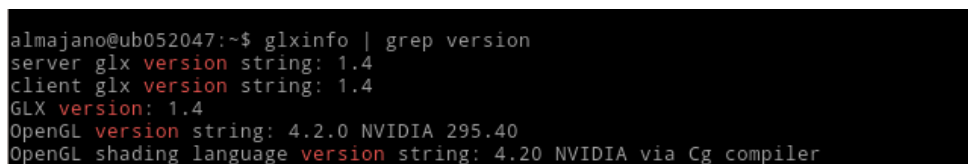
1 Instal·lació bàsica

- Instal·lació de OpenGL. Reviseu si ho teníeu instal·lat en el vostre computador, per exemple, des de linux fent la comanda `glxinfo`. Si teniu una targeta Nvidia, us hauria de sortir un missatge com:



```
anna@gilgit: ~
File Edit View Terminal Help
name of display: :0.0
display: :0 screen: 0
direct rendering: Yes
server glx vendor string: NVIDIA Corporation
server glx version string: 1.4
server glx extensions:
    GLX_EXT_visual_info, GLX_EXT_visual_rating, GLX_SGIX_fbconfig,
    GLX_SGIX_pbuffer, GLX_SGI_video_sync, GLX_SGI_swap_control,
    GLX_EXT_swap_control, GLX_EXT_texture_from_pixmap, GLX_ARB_create_context,
    GLX_ARB_create_context_profile, GLX_ARB_multisample, GLX_NV_float_buffer,
    GLX_ARB_fbconfig_float, GLX_EXT_framebuffer_sRGB
client glx vendor string: NVIDIA Corporation
client glx version string: 1.4
client glx extensions:
    GLX_ARB_get_proc_address, GLX_ARB_multisample, GLX_EXT_visual_info,
    GLX_EXT_visual_rating, GLX_EXT_import_context, GLX_SGI_video_sync,
    GLX_NV_swap_group, GLX_NV_video_out, GLX_SGIX_fbconfig, GLX_SGIX_pbuffer,
    GLX_SGI_swap_control, GLX_EXT_swap_control, GLX_ARB_create_context,
    GLX_ARB_create_context_profile, GLX_NV_float_buffer,
    GLX_ARB_fbconfig_float, GLX_EXT_fbconfig_packed_float,
    GLX_EXT_texture_from_pixmap, GLX_EXT_framebuffer_sRGB,
    GLX_NV_present_video, GLX_NV_copy_image, GLX_NV_multisample_coverage,
    GLX_NV_video_capture
--More--
```

Per a veure les versions instal·lades de GL i dels shaders, feu la comanda `glxinfo | grep version` i us hauria de donar un missatge com:



```
almajano@ub052047:~$ glxinfo | grep version
server glx version string: 1.4
client glx version string: 1.4
GLX version: 1.4
OpenGL version string: 4.2.0 NVIDIA 295.40
OpenGL shading language version string: 4.20 NVIDIA via Cg compiler
```

Si no ho teniu instal·lat, aneu a la plana http://www.opengl.org/wiki/Getting_started on hi han les instal·lacions per a les diferents plataformes i targetes.

2 Una aplicació Qt des de zero

Obriu el QtCreator des del menu d'inici, desenvolupament. Començarem donant el nom del projecte nou, que l'anomenem Carreres. I seleccionem realitzar tant la versió de Release com la de Debug de Qt.

En la pantalla de sumari, ens explica quines classes i fitxers es crearan. El fitxer Carreres.pro conté l'equivalent a un Makefile de C o de C++ convencional. Accionem el botó de **Done** i així crearà el primer projecte.

Ara apareixeran els fitxers creats en el mode Edit, si tot ha anat bé:

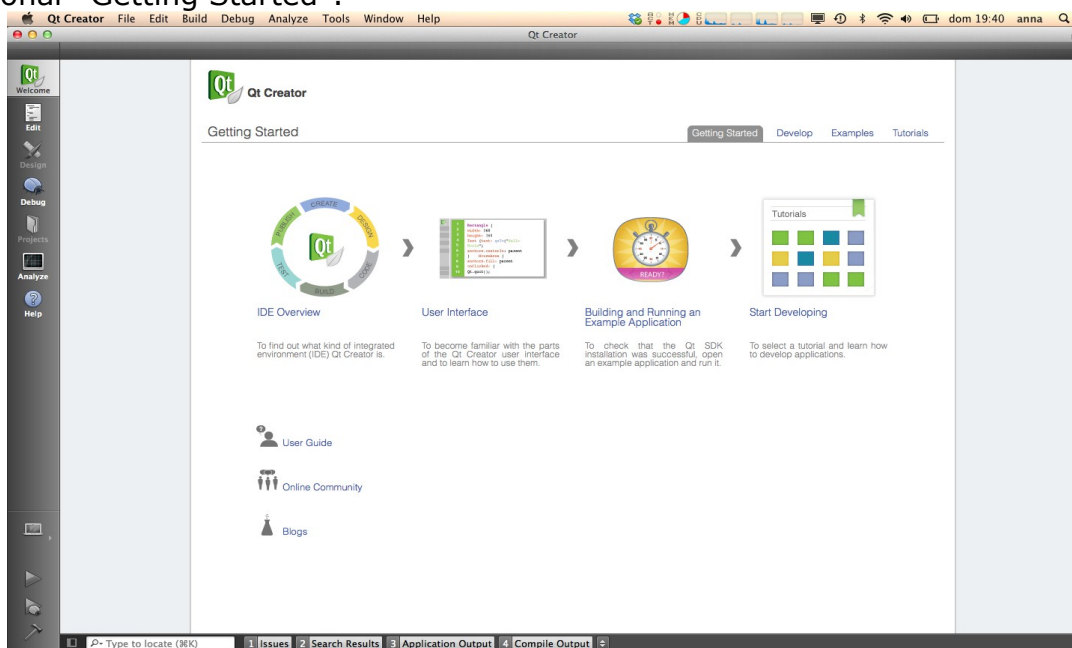
- el fitxer **main.cpp** conté el programa principal. Fixeu-vos que conté només la instanciació de la finestra principal i que directament executar un fil d'execució (o thread) i dona el control dels events al motor de Qt.

```
main.cpp
1  #include "mainwindow.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      MainWindow w;
8      w.show();
9
10     return a.exec();
11 }
12
```

- La classe **mainwindow.cpp** serà l'encarregada de fer una aplicació amb tot el disseny dels menús, widgets específics, etc. Ara no conté res, però si executem el projecte creat amb el Play, veurem que surt una finestra principal buida.

```
mainwindow.cpp
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  MainWindow::MainWindow(QWidget *parent) :
5      QMainWindow(parent),
6      ui(new Ui::MainWindow)
7  {
8      ui->setupUi(this);
9  }
10
11 MainWindow::~MainWindow()
12 {
13     delete ui;
14 }
15
```

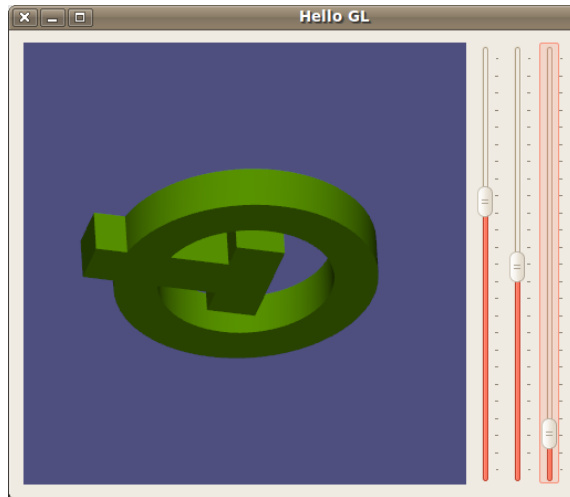
Per a conèixer més detalls de l'IDE QtCreator podeu picar el botó de **Welcome** i seleccionar "Getting Started":



Tanquem aquesta primera execució i carreguem un exemple ja creat que posa un widget de GL com a àrea de visualització.

3 Hello Qt

En aquest apartat acabarem tenint una aplicació en QT i OpenGL que visualitza el logo de Qt en 3D. S'interacciona amb els "sliders" de l'esquerra.



Passos a seguir:

1. Baixeu-vos del campus virtual de l'assignatura, l'aplicació HelloQt de l'enllaç: <https://campusvirtual2.ub.edu/mod/resource/view.php?id=36495>
2. Descomprimiu-vos-la en la vostra carpeta de pràctiques
3. Obriu el projecte des del QtCreator (menú File->Open Project)

Aquesta primera aplicació és un exemple que està inclòs dins del QtCreator i que permet visualitzar el logo de Qt en tres dimensions, permetent rotar la visualització.

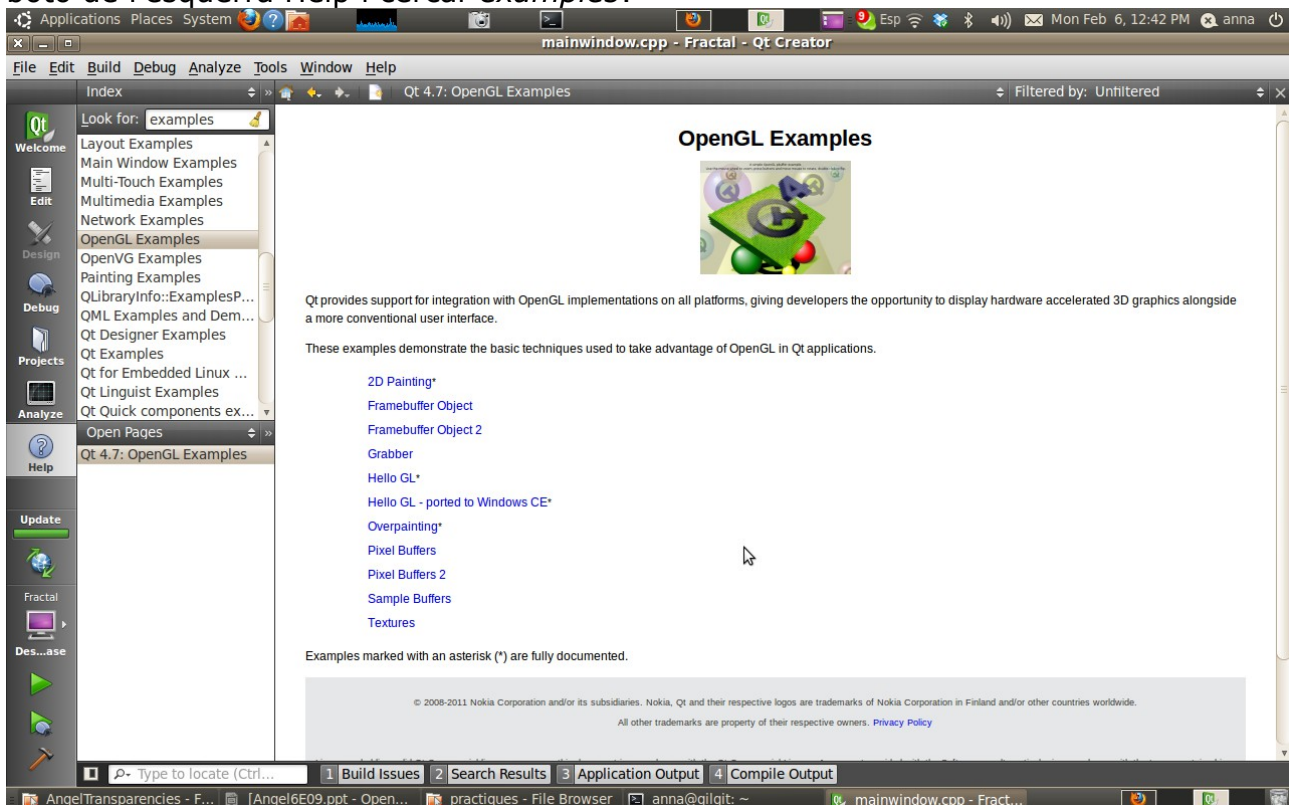
Podeu accedir a l'exemple per executar-lo i per veure un tutorial explicatiu teclejant en la consola la comanda:

```
$ qtdemo
```

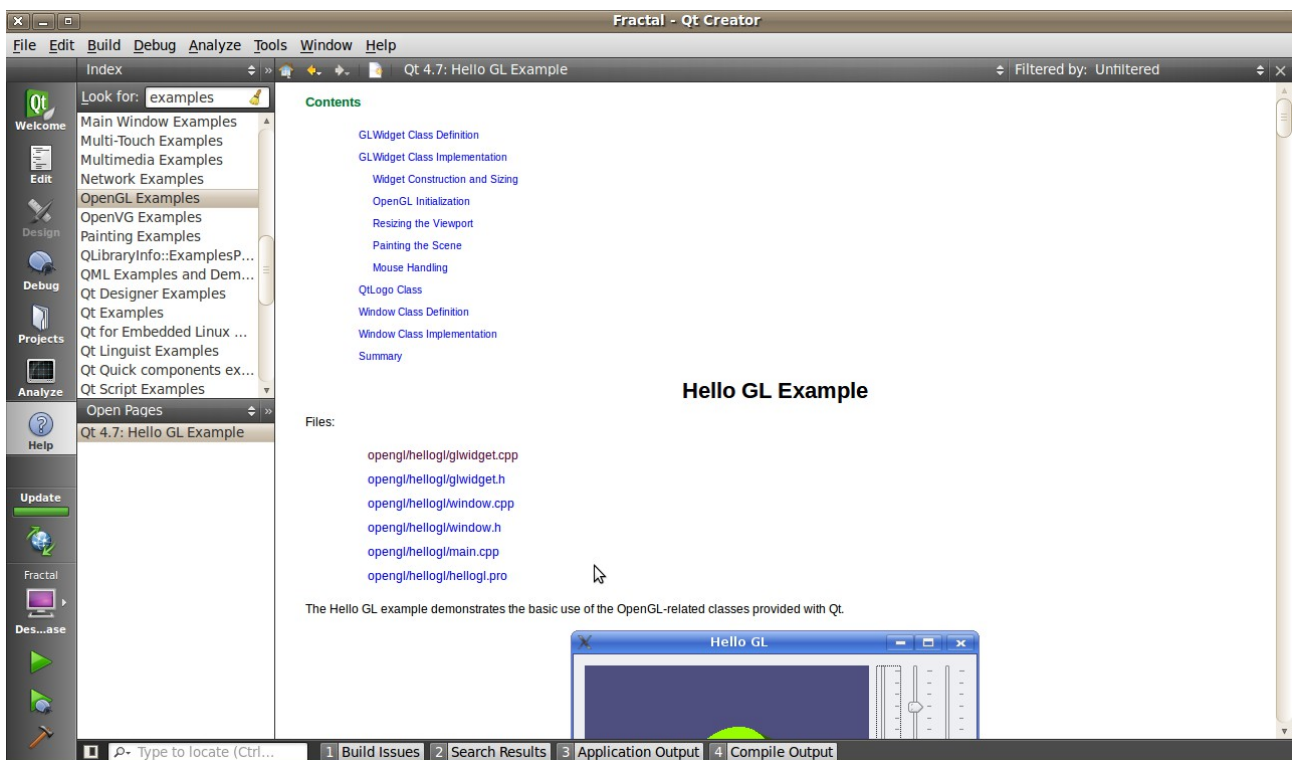
i seleccionant les demostracions de OpenGL. Si seleccioneu Hello GL, us sortirà l'exemple concret.



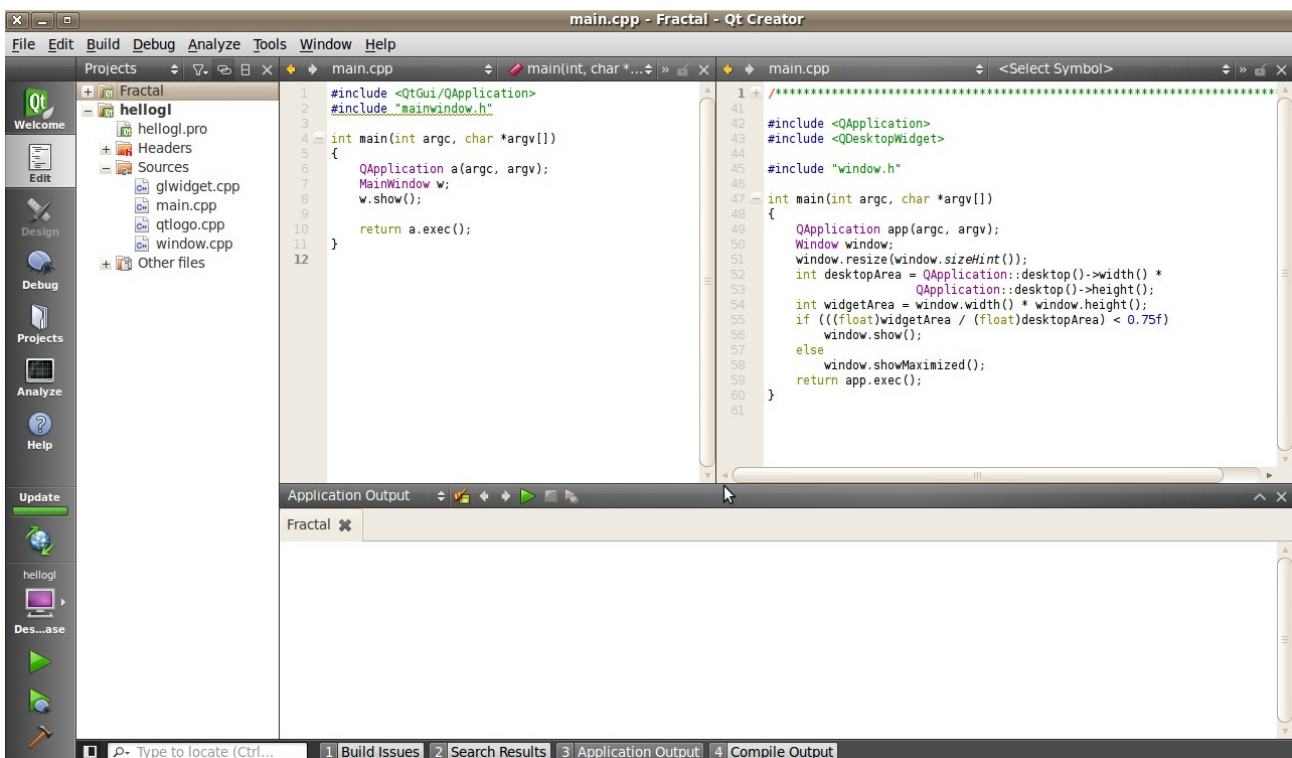
Per a veure el tutorial i el codi des de dins de l'entorn del QtCreator podeu anar al botó de l'esquerra Help i cercar *examples*:



Aquí, seleccioneu HelloGL i us sortirà tota la informació disponible:



Si anem explorant cadascuna de les classes, veurem que, a diferència de l'aplicació que hem creat anteriorment, la finestra principal es de tipus QWidget i no de tipus MainWindow. En principi, es poden usar de forma molt similar. A la captura següent es pot veure la diferència de codis generats en els programes principals de cadascuna de les aplicacions.



Les principals classes d'aquesta aplicació són:

- **main**: conté el programa principal
- **window**: conté el disseny de la interfície
- **glwidget**: conté el widget que permetrà dibuixar amb OpenGL en el seu interior. Els mètodes principals a destacar en aquest widget són mètodes que es deriven de la classe abstracte QGLWidget, que són:
 - *initializeGL()*: es crida només el primer cop que s'instancia el widget
 - *paintGL()*: mètode que es crida cada cop que la finestra es refresca, ja sigui per que una altra finestra la tapa, o per què hi ha un canvi de refresc, per exemple per que es canvia un angle de visió.
 - *resizeGL()*: mètode que es crida cada vegada que hi ha un canvi en el tamany del widget, per exemple quan s'amplia la finestra.
- **logo**: classe que conté la informació de la geometria 3D, topologia i els colors del logo Qt. D'aquesta classe només cal remarcar, per ara, que és important realitzar el mètode *draw()* i el mètode que permetrà pintar tota la geometria en el widget de GL. Els mètodes per a construir la geometria es tractaran més endavant en l'assignatura. Per ara, només ens fixarem que és necessari saber els punts, la seva connexió (o topologia) i els colors associats a cadascun d'ells.

Observeu que en el fitxer .pro (o makefile) hi ha una línia que inclou la utilització de OpenGL. A vegades, si feu un projecte nou, cal editar especialment aquest fitxer per afegir aquesta opció:

QT += opengl

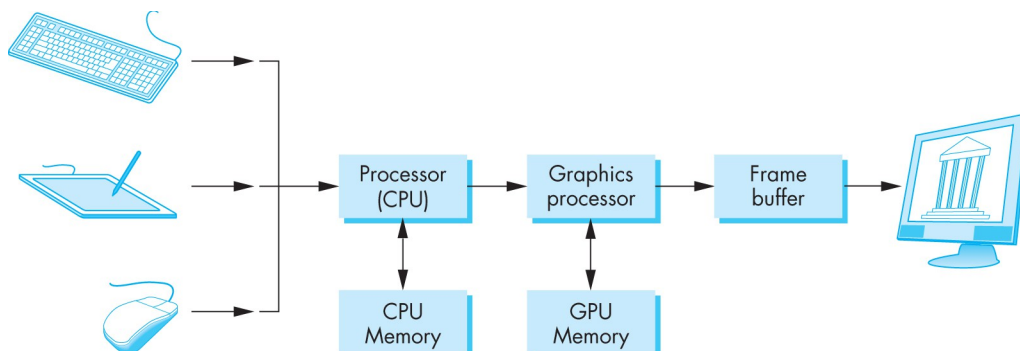
Sense ella, no és possible compilar amb els widgets de gl dins de Qt.

Veureu que en aquest primer exemple també es sincronitzen els canvis que hi ha en els scrollbars de la dreta amb els angles que es rota la imatge central. En Qt aquesta notificació es fa mitjançant el mecanisme de *signal* i *slot*. Podeu trobar més informació del seu funcionament a l'enllaç <http://qt-project.org/doc/qt-4.8/signalsandslots.html>

Per a veure en detall aquest exemple, segueix les explicacions del tutorial associat al HelloQT. És important que entenguis el funcionament d'aquesta aplicació abans de passar al següent punt.

4 La primera aplicació GL: visualització d'un model poligonal

L'arquitectura en la què es basa tota aplicació gràfica és la següent:



OpenGL és una llibreria que treballa en la memòria de la CPU i encapsula tot l'enviament de dades i gestions a la GPU. El traspàs a la GPU de les dades gràfiques (vèrtexs i colors) pot ser transparent al programador d'OpenGL. Les aplicacions gràfiques clàssiques funcionen d'aquesta manera i, per accelerar l'execució de les aplicacions, transfereixen les dades a la GPU utilitzant **display lists**. Aquestes estructures guarden els vèrtexs 3D i colors pre-calculats a la GPU, de forma que quan se'ls hi fan transformacions geomètriques, com per exemple rotacions, no cal tornar-los a enviar des de la CPU a la GPU, sinó que es recalcula la nova imatge final a la GPU. Tot i així, quan hi ha algun canvi a les llums, per exemple, cal tornar a recalcular les noves display lists i tornar a enviar les dades a la GPU.

En els darrers anys, s'ha obert la possibilitat de treballar i programar directament en la GPU mitjançant el llenguatge GLSL. Això significa la possibilitat d'una reprogramació directa d'algunes funcions que segueix oferint OpenGL, però tenint constantment la geometria i els colors a la GPU, evitant les transferències de memòria entre els dos processadors (CPU i GPU).

A continuació es veuran tres maneres de programar aplicacions gràfiques:

- Dins de l'arquitectura clàssica d'OpenGL:
 1. Usant directament OpenGL sense optimitzacions
 2. Usant display lists d'OpenGL
- Reprogramant la GPU:
 3. Usant directament shaders.

Per a realitzar aquesta comparativa, visualitzarem un model poligonal, concretament un cub.

4.1 Arquitectura clàssica

OpenGL ens ofereix diferents possibilitats per a visualitzar dades. Existeixen diferents funcions que permeten dibuixar punts, rectes, triangles, etc.



Les crides a aquests mètodes (o Function calls) es fan des de la CPU i les dades s'emmagatzemen en la memòria de la CPU. Quan es fa la crida a les funcions d'OpenGL de *display()*, es realitza la transferència a la GPU.

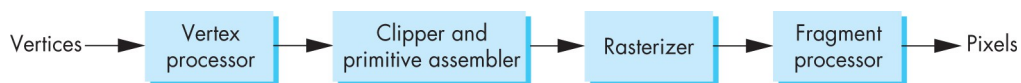
1. Baixa l'aplicació del campus CubGL.tar.gz de l'enllaç <https://campusvirtual2.ub.edu/mod/resource/view.php?id=36496>, descomprimeix-la i obre un nou projecte amb l'IDE QtCreator. Analitza el mètode *draw()*:
 - Des d'on es crida? Per què?
 - Dins de quina classe està definit?
 - Quina diferència hi ha entre el constructor i el mètode *draw()*? On es defineix la geometria? On es dibuixa? Qui ho envia a la GPU?
 - Fixa't que el cub gira sense parar. Per què? Quin mecanisme s'utilitza per a cridar cada cert temps el canvi d'angle de gir? Mira com es connecten els signals i els slots per activar les rotacions.

2. Baixa ara l'aplicació del campus `CubGLDisplayLists.tar.gz` de l'enllaç , <https://campusvirtual2.ub.edu/mod/resource/view.php?id=36497>. Analitza el mètode `draw()`:

- Des d'on es crida? Per què?
- Dins de quina classe està definit?
- Quina diferència hi ha entre el constructor i el `draw()`? On es defineix la geometria? On es dibuixa? Qui ho envia a la GPU?
- Prova de canviar els colors dels vèrtexs de cub quan es canvia la rotació en X. Canvia la visualització? Què has de fer per a poder veure els resultats?

4.2 Reprogramant la GPU(shaders)

Una vegada es tenen els vèrtexs, es poden realitzar les transformacions geomètriques el processador de vèrtexs. Aquest programa s'anomena *vertex shader*. Després de rasteritzar els vèrtexs en píxels, es pot programar el càlcul del color de cada píxel en el processador de fragments (el programa s'anomena *fragment shader*).



3. Baixa ara l'aplicació `CubGPU.tgz`(de l'enllaç del Campus Virtual <https://campusvirtual2.ub.edu/mod/resource/view.php?id=36498>), descomprimeix el fitxer i obre un nou projecte amb l'IDE QtCreator. En la versió per a Qt 4.8 s'utilitzen unes estructures auxiliars per a passar informació a la GPU que són els **Vertex Arrays**. Amb aquestes estructures es permet passar els vèrtexs i la informació associada a cada vèrtex, com per exemple el color, la normal, etc. Per a ser usada en el vertex shader. Que fa el mètode `draw()`?

- Des d'un es crida? Per què?
- Dins de quina classe està definit?
- Quina diferència hi ha entre el constructor i el mètode `draw()`? On es defineix la geometria? On es dibuixa? Qui ho envia a la GPU?