

Laboratori de VIG

Sessió 1

Maria J. Blesa

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya.

Visualització i Interacció Gràfica (VIG)

Professora

- ▶ Maria J. Blesa
ALBCOM Research Group
Dept. Llenguatges i Sistemes Informàtics
UPC
<http://www.lsi.upc.edu/~mjblesa>

Consultes

- ▶ Despatx Ω – 213, Campus Nord
- ▶ Cita prèvia per mail a mjblesa@lsi.upc.edu

Visualització i Interacció Gràfica (VIG)

Organització del curs

- ▶ Qt i QtDesigner + pràctica 0 (P0)
- ▶ Introducció a OpenGL
- ▶ OpenGL (cont.) + pràctica 1 (P1)
- ▶ OpenGL (cont.) + pràctica 2 (P2)
- ▶ OpenGL (cont.) + pràctica 3 (P3)

Visualització i Interacció Gràfica (VIG)

Metodologia de treball

- ▶ Per a les sessions introductòries disposeu de documents que us guiaran en l'aprenentatge de Qt i OpenGL. Heu de seguir-les, i fer els exemples i exercicis **pel vostre compte**. No són obligatòries.
- ▶ A la resta de sessions, heu d'anar elaborant les dues pràctiques puntuables restants.
- ▶ Les pràctiques són **individuals**.
- ▶ És aconsellable que hagueu llegit els documents prèviament.
- ▶ Les pràctiques s'han de poder **compilar i executar en aquesta aula**

Visualització i Interacció Gràfica (VIG)

Avaluació

- ▶ La nota de pràctiques (**NP**) suposa un 25% de la nota final de VIG. La nota de pràctiques es calcula a partir de la nota de les 4 pràctiques puntuables que es realitzen a laboratori, de la següent forma:

$$NP = 0.10 \cdot P0 + 0.30 \cdot P1 + 0.30 \cdot P2 + 0.30 \cdot P3$$

Visualització i Interacció Gràfica (VIG)

Recursos

- ▶ **Qt:**
www.trolltech.com/products/qt/,
"Programming with Qt" (O'Reilly)
- ▶ **OpenGL:**
www.opengl.org,
www.xmission.org/~nate/tutors.html
"OpenGL Programming Guide" (red book), online
- ▶ **C++:**
infinat de tutorials online
Introducció al C++ per a programadors en Java

Llibreria Qt

Llibreria Qt

Què és?

- ▶ Una llibreria en C++ per a dissenyar interfícies gràfiques d'usuari (GUI) en diferents plataformes.
- ▶ Va ser creada i és actualment mantinguda per Trolltech.
- ▶ És la base del popular entorn d'escriptori KDE en Linux, i una component estàndard de la major part de distribucions Linux

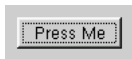
Disponibilitat

- ▶ Descarregable (lliure per a usos no comercials) a:
<http://trolltech.com/downloads/opensource>
<ftp://ftp.trolltech.com/qt/source>
- ▶ Per a plataformes MS Windows, Unix/X11, Macintosh

Llibreria Qt

Widgets

Qt proporciona diversos components atòmics configurables que anomena **Widgets** (windows + gadgets).



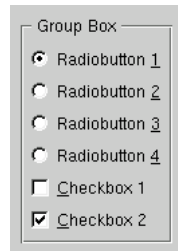
QPushButton



QCheckBox



QListBox

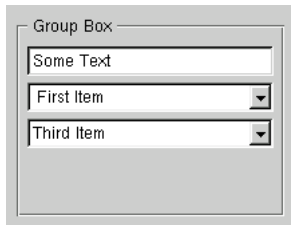


QButtonGroup

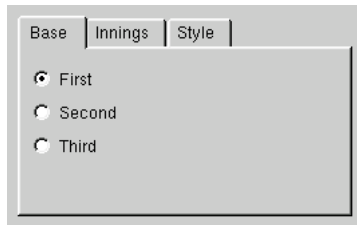
Llibreria Qt

Widgets

Qt proporciona diversos components atòmics configurables que anomena **Widgets** (windows + gadgets).



QGroupBox

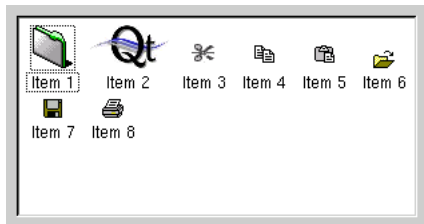


QTabWidget

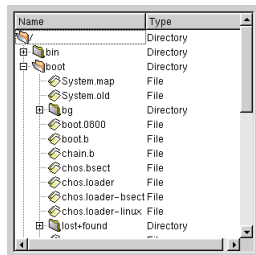
Llibreria Qt

Widgets

Qt proporciona diversos components atòmics configurables que anomena **Widgets** (windows + gadgets).



QIconView



QListView

Llibreria Qt

Widgets

Cada Widget està implementat per una classe de la llibreria Qt.

- ▶ Per utilitzar-los hem de crear un objecte de la classe

```
QPushButton boto1(0);  
QPushButton* boto2 = new QPushButton(0);
```

- ▶ Per consultar de quins mètodes disposa cada classe hem de consultar el manual de referència, ja sigui executant l'assistent del sistema (**assistant &**), o a la plana web <http://doc.trolltech.com/4.0/classes.html>

Llibreria Qt

Widgets

Mitjançant els mètodes de cadascuna d'aquestes classes, podem:

- ▶ Definir o modificar l'aspecte o estat del Widget

```
boto1.setPaletteBackgroundColor(Qt::black);  
boto2->resize(100,50);
```

- ▶ Afegir o consultar informació

```
boto1.setText("OK");  
boto2->setText("Quit");
```

```
boto1.isOn();  
boto2->isDown();
```

Llibreria Qt

Widgets

Com que s'implementen en classes C++, podem:

- ▶ Heretar-les per a definir nous Widgets personalitzats

```
class ElMeuBoto: public QPushButton {  
public:  
    ElMeuBoto(QWidget *parent=0);  
    ~ElMeuBoto();  
    void canviarTamany(int w, int h);  
}
```

- ▶ I després utilitzar els propis Widgets

```
ElMeuBoto boto(0);  
boto.canviarTamany(10,5);
```

Llibreria Qt

Estructura bàsica d'un programa

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char **argv)
{
    QApplication a(argc,argv);
    ...
    return a.exec();
}
```

Llibreria Qt

Estructura bàsica d'un programa

exemple-1.cpp

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char **argv)
{
    QApplication a(argc,argv);
    QPushButton hello("Hello Qt!",0);
    hello.resize(100,30);
    a.setMainWidget(&hello);
    hello.show();
    return a.exec();
}
```

Llibreria Qt

Estructura bàsica d'un programa

Compilem i enllacem exemple-1.cpp

```
g++ -c exemple-1.cpp -I$QTDIR/include  
g++ -o exemple-1 exemple-1.o -L$QTDIR/lib -lqt-mt
```

(o bé tot d'una)

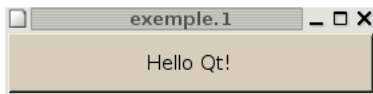
```
g++ exemple-1.cpp -o exemple-1  
-I$QTDIR/include -L$QTDIR/lib -lqt-mt
```

Però ja veurem més endavant que podem compilar d'una forma molt més senzilla fent servir fitxers de projecte.

Llibreria Qt

Estructura bàsica d'un programa

Executem exemple-1...



Llibreria Qt

Estructura bàsica d'un programa

exemple-2.cpp

```
#include <QApplication>
#include <QPushButton>
#include <QFrame>
#include <QLayout>
#include <QLineEdit>

int main(int argc, char **argv)
{
    QApplication a(argc,argv);

    QString fontFamily = "Arial";
    a.setFont(fontFamily);
```

Llibreria Qt

Estructura bàsica d'un programa

exemple-2.cpp (cont.)

```
QFrame F(0, NULL);
```

```
QHBoxLayout cH(&F);
```

```
QLineEdit le(&F);  
cH.addWidget(&le);
```

```
QSpacerItem *sp = new QSpacerItem(100, 20);  
cH.addItem(sp);
```

Llibreria Qt

Estructura bàsica d'un programa

exemple-2.cpp (cont.)

```
QPushButton ok("D'acord", &F);  
cH.addWidget(&ok);  
  
QPushButton surt("Surt", &F);  
cH.addWidget(&surt);  
  
a.setMainWidget(&F);  
F.show();  
  
return a.exec();  
  
}
```

Llibreria Qt

Layouts

Els **layout** (disposicions) permeten organitzar els components visuals dintre de formularis i quadres de diàleg.



Horitzontal
(QHBoxLayout)



Vertical
(QVBoxLayout)



En graella
(QGridLayout)

Llibreria Qt

Estructura bàsica d'un programa

Compilem i enllacem exemple-2...

```
g++ -c exemple-2.cpp -I$QTDIR/include  
g++ -o exemple-2 exemple-2.o -L$QTDIR/lib -lqt-mt
```

Executem exemple-2...



Llibreria Qt

Signals i Slots

Per tal de connectar la interfície gràfica que dissenyem amb la nostra aplicació, caldrà connectar els elements gràfics Qt al nostre codi C++.

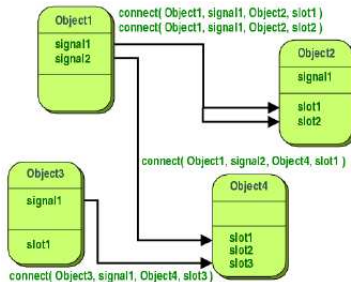
Les connexions poden ser:

- ▶ Alt nivell: associades als components
- ▶ Baix nivell: events bàsics del computador

Per exemple, voldrem que quan es clica un determinat botó, canviï el punt de vista d'una figura 3D que dibuixem.

Llibreria Qt

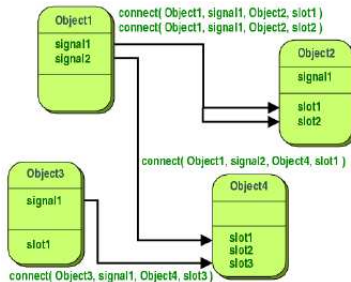
Signals i Slots



Els senyals (**signals**) i els mètodes forat (**slots**) s'utilitzen per a la comunicació entre objectes. Qualsevol classe que hereti de `QObject` (o de les seves subclasses), pot contenir senyals i forats.

Llibreria Qt

Signals i Slots

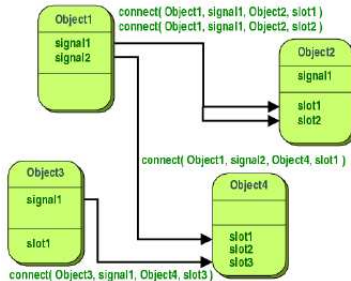


Signals

Els senyals són emesos per un objecte quan el seu estat canvia d'alguna forma que pot ser interessant per a d'altres objectes.

Llibreria Qt

Signals i Slots



Slots

Un slot és cridat quan s'emeti un senyal què és connectat a ell.

Llibreria Qt

Signals i Slots

Com es connecten Signals i Slots?

```
bool QObject::connect(  
    const QObject* sender, const char* signal,  
    const QObject* receiver, const char* member) [static]
```

Exemple

```
QLabel label;  
QScrollBar scroll;  
QObject::connect(&scroll, SIGNAL(valueChanged(int)),  
                &label, SLOT(setNum(int)));
```

Llibreria Qt

Signals i Slots

exemple-3.cpp

```
#include <QApplication>
#include <QFrame>
#include <QPushButton>
#include <QLayout>
#include <QLabel>

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

    QApplication app(argc, argv);
    QFrame wnd;
    wnd.setCaption("Qt");
    QVBoxLayout layout(&wnd);
```

Llibreria Qt

Signals i Slots

exemple-3.cpp (cont.)

```
QLabel label("Hello World!", &wnd);
QPushButton button("Exit", &wnd);

layout.addWidget(&label,0,Qt::AlignHCenter|Qt::AlignVCenter);
layout.addWidget(&button,0,Qt::AlignHCenter|Qt::AlignVCenter);

app.connect(&button, SIGNAL(clicked()), SLOT(quit()));

wnd.resize(150,70);
wnd.show();
return app.exec();
}
```

Llibreria Qt

Signals i Slots

Compilem i enllacem exemple-3...

```
g++ -c exemple-3.cpp -I$QTDIR/include  
g++ -o exemple-3 exemple-3.o -L$QTDIR/lib -lqt-mt
```

Executem exemple-3...



Llibreria Qt

Signals i Slots

Teniu un exemple més complicat a

</assig/vig/sessions/S1.1/lab0.cpp>

Llibreria Qt

Interfícies més complexes

Tots els components de Qt deriven de la classe `QObject`. Si volem crear un nou component amb noves funcionalitats o que s'hagi de comunicar amb components Qt, haurem de fer-ho mitjançant derivació (directa o indirecta) d'aquesta classe.

En general, per construir interfícies més complexes i útils caldrà poder **connectar els signals i slots dels widgets Qt amb codi nostre**. Per a fer-ho, dissenyarem classes que deriven de:

- ▶ `QObject` (per a objectes no gràfics)
- ▶ `QWidget` (per a dissenyar nous components gràfics)
- ▶ Algun altre component de la llibreria

Llibreria Qt

Interfícies més complexes

Exemple d'estructura (MevaClasse.h)

```
class MevaClasse : public QObject
{
    Q_OBJECT
public:
    MevaClasse();
public slots:
    void setValue( int );
signals:                                     // No s'implementen
    void valueChanged( int );
private:
    ...
};
```

Llibreria Qt

Interfícies més complexes

Exemple d'estructura (MevaClasse.cpp)

```
#include "MevaClasse.h"
...

MevaClasse::MevaClasse(){
    ...
}

MevaClasse::setValue( int v ){
    ...
    emit valueChanged(v); // s'executen els slots connectats
    ...
}
```

Llibreria Qt

Interfícies més complexes

Noteu que s'introdueix sintaxi que no és purament C++. Això farà que necessitem fer un processament previ abans de la compilació normal. Aquest preprocessament es fa mitjançant el **Meta-object compiler** (**moc**). En teniu una descripció al document del laboratori.

És molt més senzill, però, utilitzar el **qmake** per a crear un Makefile que ja crei les instruccions necessàries per a una correcta compilació (amb el preprocessament del **moc** inclòs). En general, el **qmake** també ens facilita la compilació de projectes més complexos que requereixen la compilació de diversos fitxers/classes.

Llibreria Qt

Interfícies més complexes

Compilació amb qmake

1. Heu de crear un **fitxer de projecte** `<nomprojecte>.pro` (com ara el de `/assig/vig/sessions/S1.1/ver0.pro`; més detalls de com fer-ho, a continuació).
2. Feu `qmake <nomprojecte>.pro -o Makefile`
Ara ja disposeu del fitxer Makefile
3. Feu `make` per a compilar
Ara ja disposeu de l'executable

Llibreria Qt

Interfícies més complexes

Compilació amb qmake: fitxer de projecte

```
TEMPLATE = app
CONFIG += qt warn_on release
CONFIG += moc
INCLUDEPATH += .

SOURCES += <main>.cpp, <classe1>.cpp, <classe2>.cpp, ...
HEADERS += <classe1>.h, <classe2>.h, ...
```