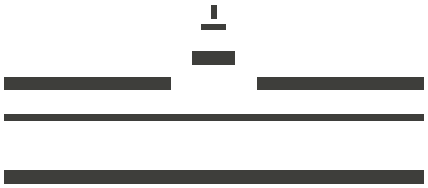
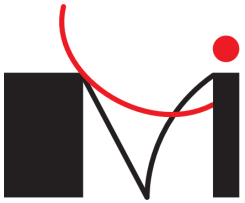


Westfälische Wilhelms-Universität Münster



Applied Mathematics Münster

Interaktive Simulationen

Lektion 3/3: Grafische Ausgabe

living.knowledge
WWU Münster

Prof. Dr. Christian Engwer
& Andreas Buhr

30. April 2014

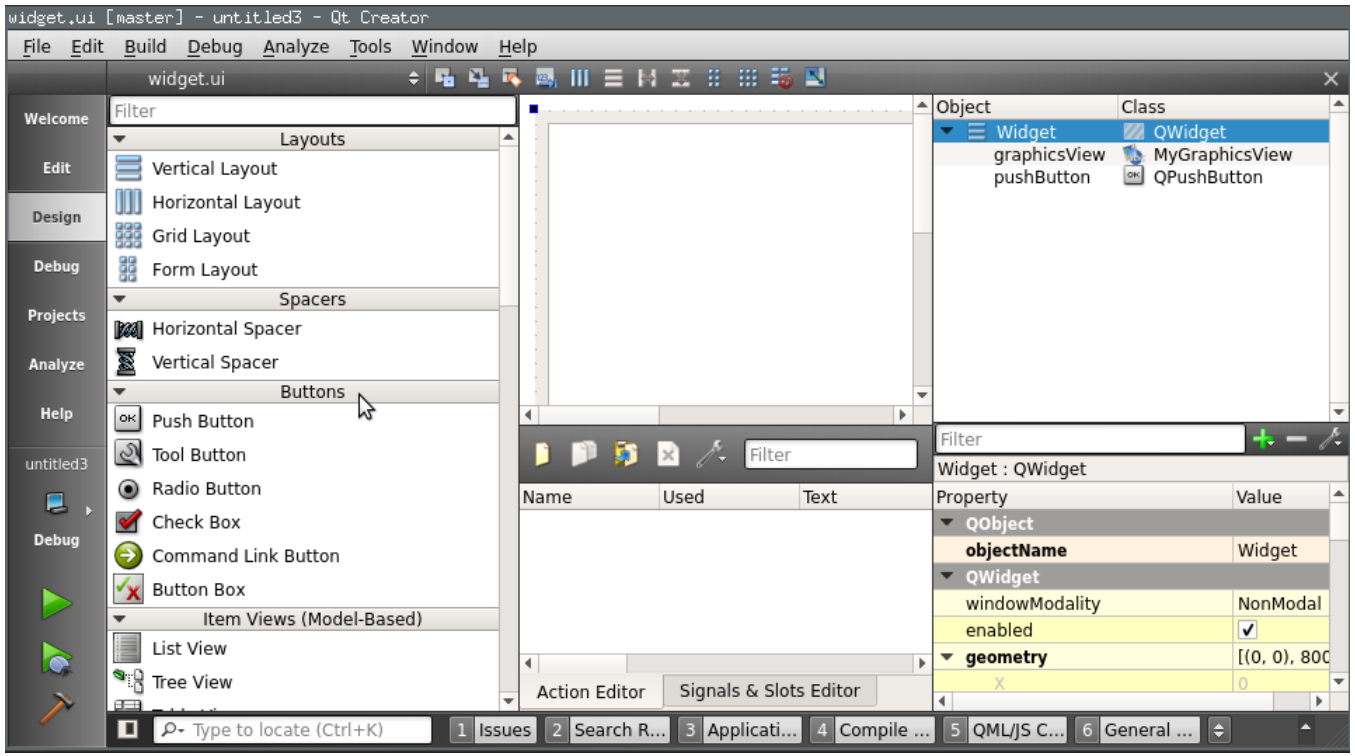
Was ist ein Widget?

Ein QWidget ist ein QObject, welches auf dem Bildschirm sichtbar ist.

Qt Documentation

The QWidget class is the base class of all user interface objects.

Widgets im QtCreator



QWidgets sind QObjects

- Sie können Signale senden und empfangen.
- Man kann sie aber nicht aus dem Hauptthread heraus bewegen:

Fehlermeldung

QObject::moveToThread: Widgets cannot be moved to a new thread

- Widgets erhalten “Events”

Was ist ein “Event”?

Events werden durch Userinteraktion ausgelöst:

- Maus kommt über Widget (enterEvent)
- Maus bewegt sich (mouseMoveEvent)
- Maustaste wird gedrückt (mousePressEvent)
- Maustaste wird losgelassen (mouseReleaseEvent)
- Fenstergröße wird geändert (resizeEvent)
- Taste auf Tastatur wird gedrückt (keyPressEvent)
- viel mehr ...

Was heißt “ein Event erhalten”?

Einen Event erhalten heißt, dass die entsprechende Memberfunktion aufgerufen wird.

Beispiel:

Maus bewegt sich $\rightarrow$
“mouseMoveEvent” des darunter liegenden Widgets wird aufgerufen.

Wie kommen wir an die Events

Wir müssen eine eigene Klasse schreiben, die von dem Widget, welches wir benutzen, ableitet.

Livedemo

Livedemo: Klasse erstellen, Event Handler überschreiben, benutzen.

QGraphicsView und QGraphicsScene

QGraphicsView und QGraphicsScene sind das “Schweizer Messer” zur Visualisierung.

QGraphicsView und QGraphicsScene

QGraphicsView und QGraphicsScene sind das “Schweizer Messer” zur Visualisierung.

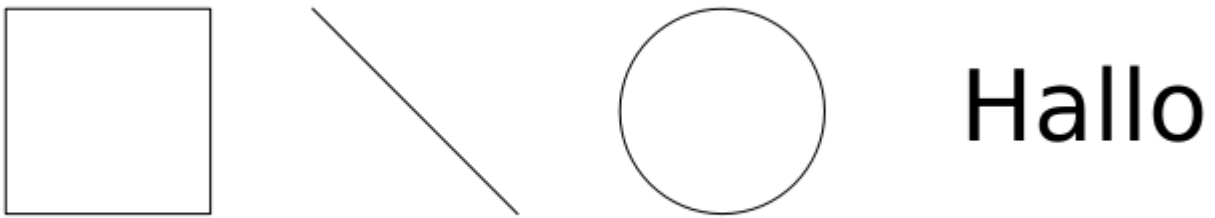
- *QGraphicsScene* enthält “Items”, die visualisiert werden.
- *QGraphicsView* verhält sich wie ein Fenster, durch das man die Scene betrachtet

Items in der QGraphicsScene

Items in der QGraphicsScene können sein:

- Linien (QGraphicsLineItem)
- Ellipsen (QGraphicsEllipseItem)
- Rechtecke (QGraphicsRectItem)
- Bilder (QGraphicsPixmapItem)
- ... viel mehr

Eine Scene mit ein paar Items:



Wer ist für was zuständig?

QGraphicsView

- Zoomen
- Scrollen

QGraphicsScene

- Items hinzufügen
- Items bewegen
- Items entfernen

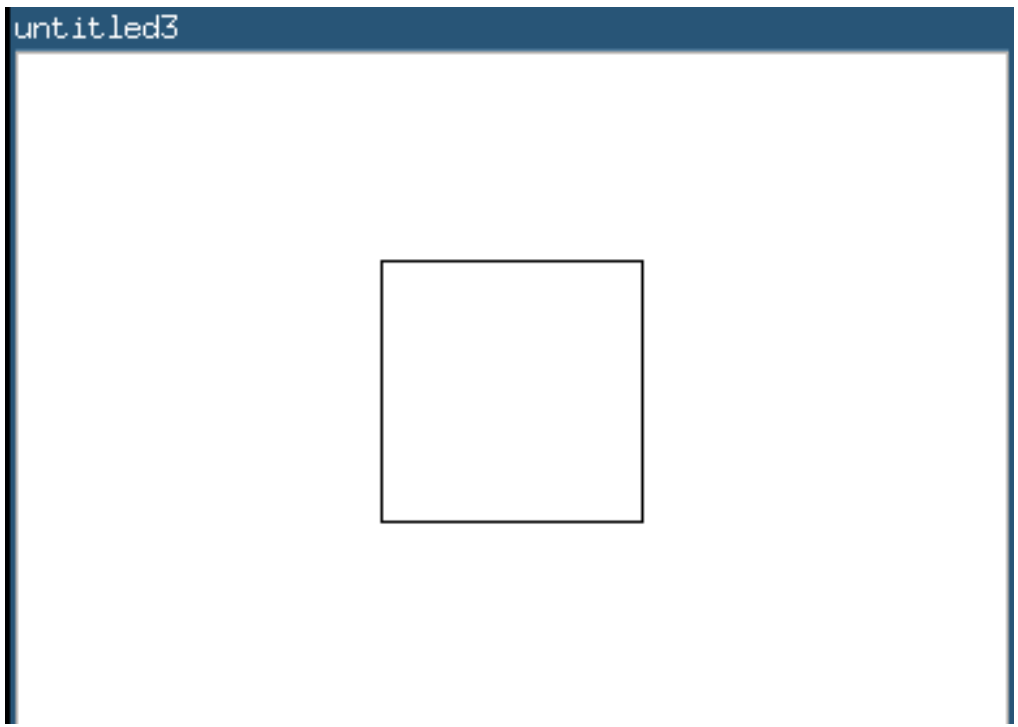
Quick HowTo

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

    QGraphicsView view;
    view.show();
    QGraphicsScene scene;
    view.setScene(&scene);
    scene.addRect(0,0,100,100);

    return a.exec();
}
```

Quick HowTo: Screenshot



Wir wollen QGraphicsView und Events

Lösung:

- Im QtCreator einen QGraphicsView hinzufügen.
- Eine neue Klasse anlegen, die von QGraphicsView ableitet, z.B. “MyGraphicsView”
- Den graphicsView zu “MyGraphicsView” promoten: Rechtsklick, dann “Promote to ...”

Der Weg eines Events

Ein mouseEvent kann an drei Stellen behandelt werden:

- Im QGraphicsView:
  Dort kann man die Position auf dem Bildschirm abfragen.
- Im QGraphicsScene:
  Dort kann man die Position in Scene-Koordinaten abfragen
- Im QGraphicsItem:
  Dort kann man die Position in Item-Koordinaten abfragen

MousePressEvent im QGraphicsView

```
protected:
  virtual void mousePressEvent ( QMouseEvent * event );
```

Man bekommt einen QMouseEvent. Dieser kennt

```
const QPoint &globalPos () const
const QPoint &pos () const
```

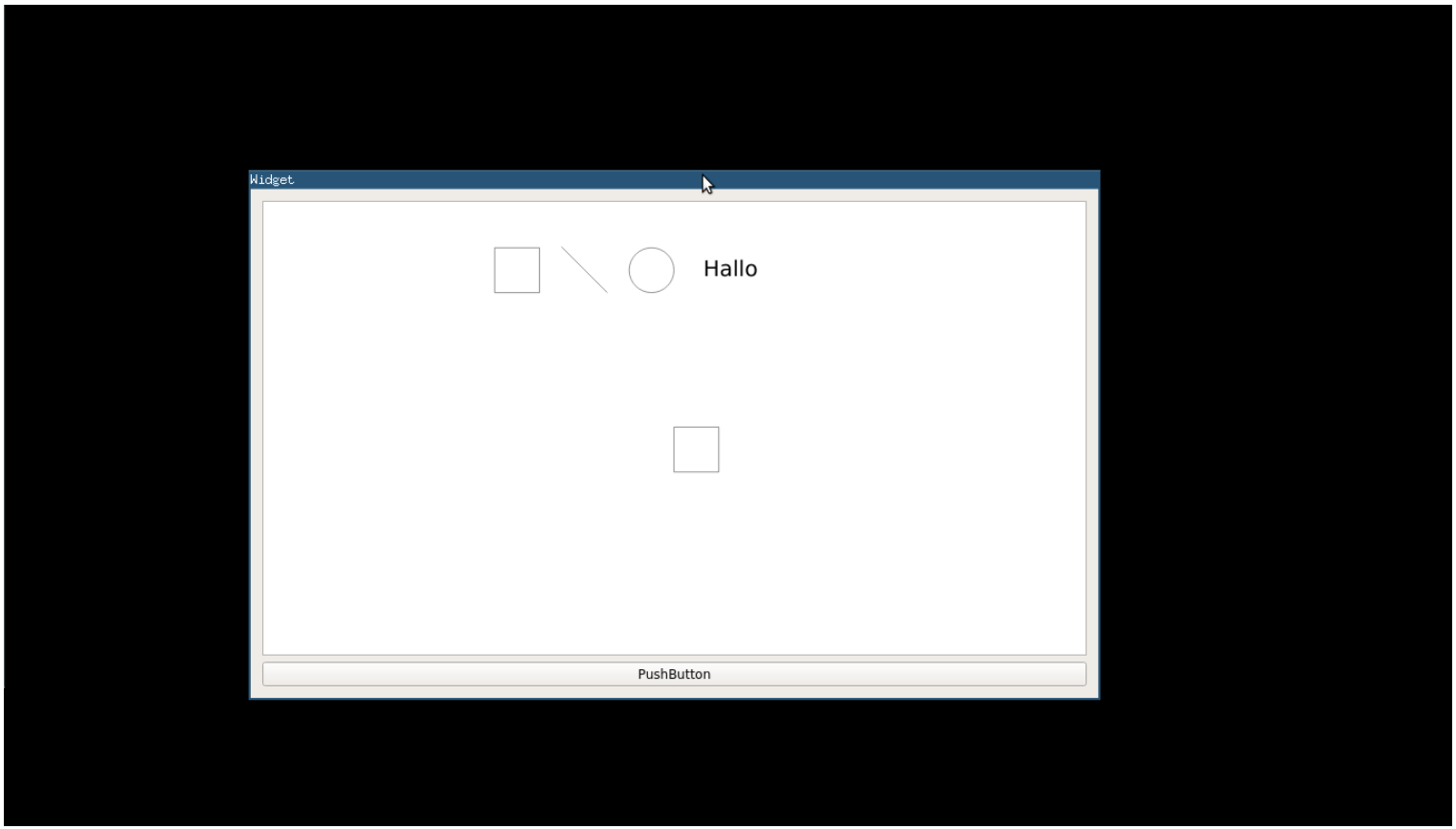
MousePressEvent in QGraphicsScene

```
protected:
    virtual void mousePressEvent(
        QGraphicsSceneMouseEvent *event);
```

Man bekommt einen QGraphicsSceneMouseEvent. Dieser kennt

```
QPointF pos () const
QPointF scenePos () const
QPoint  screenPos () const
```

vier Koordinatensysteme



vier Koordinatensysteme

- Bildschirm
- QGraphicsView
- Scene-System
- Item-System

Item Koordinatensystem

Items arbeiten immer in ihrem eigenen Koordinatensystem:

```
QGraphicsRectItem* item =
    new QGraphicsRectItem(500,500,100,100);
item.setPos(100,100);
```

Quizfrage: Wo ist das Item jetzt?

Item Koordinatensystem

Items arbeiten immer in ihrem eigenen Koordinatensystem:

```
QGraphicsRectItem* item =
    new QGraphicsRectItem(500,500,100,100);
item.setPos(100,100);
```

Quizfrage: Wo ist das Item jetzt?

Tipp

Items immer an der Stelle (0,0) erstellen und dann mit “setPos” verschieben.
So liegen alle Koordinatensysteme aufeinander.

Gelernt

- Man nehme einen QGraphicsView...
- ... verbinde ihn mit einer QGraphicsScene ...
- ... und füge QGraphicsItems hinzu ....

... fertig ist die Visualisierung.

Für die Particle- / Schwarm-Simulation

Beispiel 1

1000 Partikel könnte man als 1000 QGraphicsEllipseItem visualisieren

Beispiel 2

Fische in einer Schwarmsimulation könnte man durch Bilder (QGraphicsPixmapItem) visualisieren

Für die 2D-Grid Simulation

Beispiel

Aus 2D-Daten ein Bild (QImage) erzeugen, das in ein QPixmap konvertieren und daraus ein QGraphicsPixmapItem machen. (Beispielcode auf der Vorlesungswebseite)

Bilder zu Projekt hinzufügen

- Bild in Projektverzeichnis legen
- Dem Projekt ein “Qt Resource File” hinzufügen
- Dem “Qt Resource File” das Bild hinzufügen
- Beim Öffnen der Datei (im Code) vor den Pfad ein Doppelpunkt setzen, e.g. ":/meinbild.jpg"