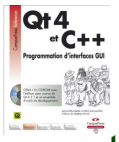
Qt dans l'Enseignement

Dessiner

digia

Références bibliographiques

- Qt4 et C++ : Programmation d'interfaces GUI
  - Jasmin Blanchette
  - Mark Summerfield

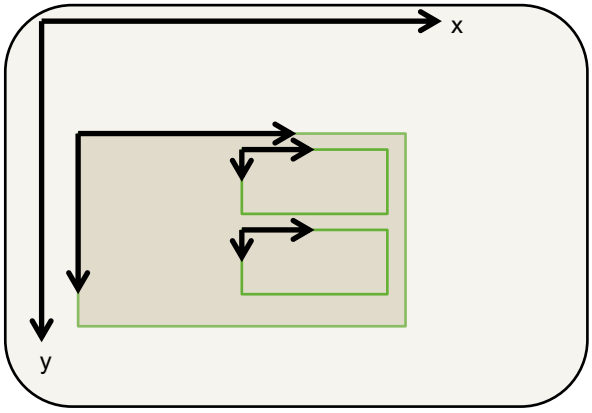


- Traduction et adaptation supports enseignement
- Site : http://www.bogotobogo.com/index.php#menu

http://www.qt.io/training-materials/
http://www.qt.io/qt-essentials-widget-edition/

Système de coordonnées

- Les coordonnées peuvent être
  - globales
  - locales (à un widget)



digia

Système de coordonnées

- Qt fournit des classes spécifiques
  - `QPoint` ⇨ un point (x, y)
  - `QSize` ⇨ une taille (largeur, hauteur)
  - `QRect` ⇨ un point et une taille (x, y, largeur, hauteur)
  - `QPointF/QSizeF/QRectF` ⇨ version en réels
- Et des fonctions
  - `topLeft`, `topRight`
  - `bottomLeft`, `bottomRight`
  - `size`

digia

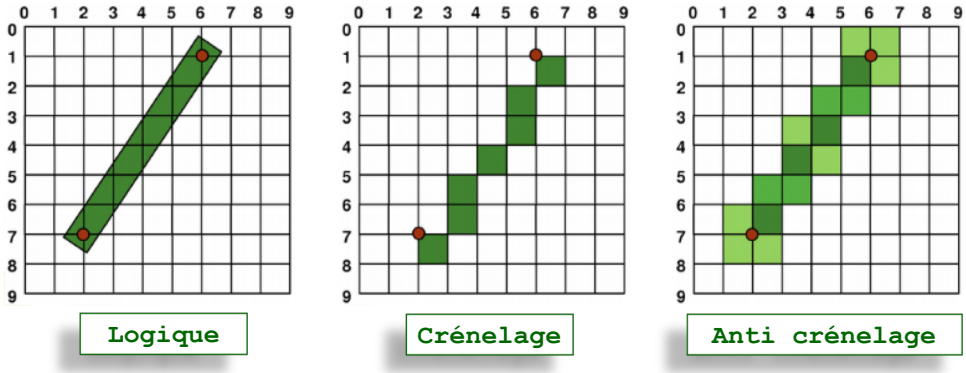
QPainter : base du dessin

- Rôle
  - Trace formes géométriques, pixmap, images et texte
  - Gère fonctionnalités avancées
    - Anticrénelage, mélange alpha, dégradé, tracés de vecteurs
    - Transformations 2D
  - Dessine sur périphériques de dessin `QPaintDevice`
    - `QWidget`
    - `QImage` ⇨ indépendant matériel, pour modification
    - `QPixmap` ⇨ en mémoire, pour afficher à l'écran
    - `QPicture` ⇨ enregistre et exécute commandes de dessin
    - `QSvgGenerator` ⇨ enregistre commandes dessin + .svg
    - `QPrinter` ⇨pour imprimer et générer .pdf

digia

QPainter : base du dessin

- Types de rendu

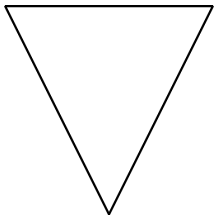


digia

QPainter : ses outils

- Contour affiché avec stylo `QPen`

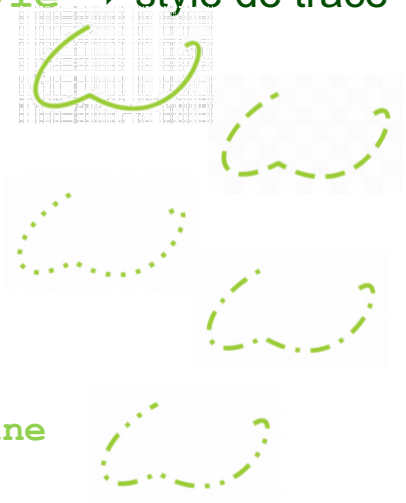
```
QPainter p( ... );
QPen pen(Qt::black, 5);
p.setPen(pen);
p.drawPolygon(polygon);
```



digia

QPainter : ses outils

- void `QPen::setStyle` ⇨ style de tracé
  - `Qt::SolidLine`
  - `Qt::DashLine`
  - `Qt::DotLine`
  - `Qt::DashDotLine`
  - `Qt::DashDotDotLine`
  - `Qt::CustomDashLine` ⇨ controllé par `dashPattern`

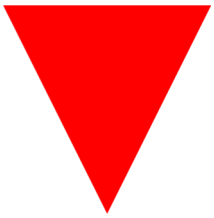


digia

QPainter : ses outils

- Intérieur rempli avec pinceau **QBrush**

```
QPainter p( ... );
p.setPen(Qt::NoPen);
p.setBrush(Qt::red);
p.drawPolygon(polygon);
```



- Plusieurs types de pinceaux disponibles
  - Solide

```
QBrush red(Qt::red);
QBrush odd(QColor(55, 128, 97));
```

  - Motifs
  - Texturé

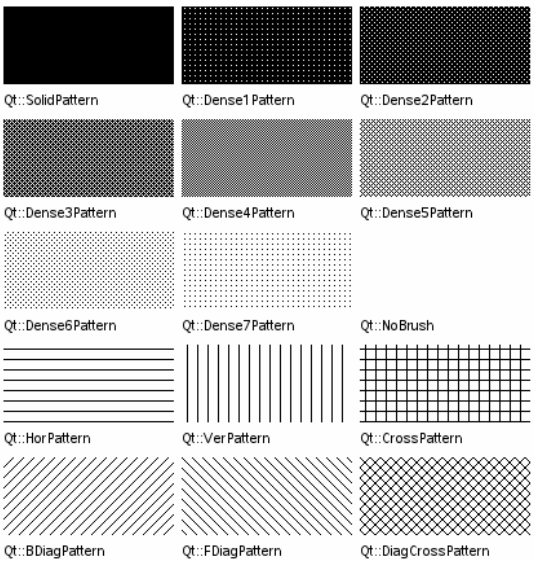
```
QBrush( const QColor &color, Qt::BrushStyle style )
```

  - Gradients

digia

QPainter : ses outils

- Les styles de motif



digia 10

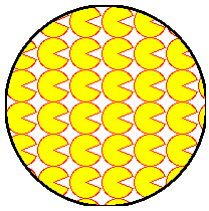
QPainter : ses outils

- Pinceau texturé ⇨ texture = `QPixmap`

```
QBrush( const QPixmap &pixmap )
```

- Exemple

```
QPixmap pacPixmap("pacman.png");

painter.setPen(QPen(Qt::black, 3));
painter.setBrush(pacPixmap);
painter.drawEllipse(rect());
```



- Texture monochrome ⇨ couleur pinceau utilisé

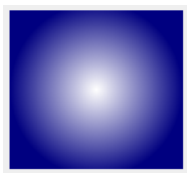
digia

QPainter : ses outils

- Pinceau dégradé ⇨ objet QGradient

```
QBrush b = QBrush( QRadialGradient( ... ) );
```

QLinearGradient



QRadialGradient

QConicalGradient

digia

QPainter : ses outils

- Exemple de gradient linéaire

```
setColorAt(0.0,
  QColor(0, 0, 0))
```

```
setColorAt(0.7,
  QColor(255, 0, 0))
```

```
setColorAt(1.0,
  QColor(255, 255, 0))
```

digia

QPainter : ses outils

- Possibilité d'ajouter un pinceau à un stylo
  - Pour générer un texte dégradé

digia

QPainter : ses outils

- Penser à supprimer stylo et pinceau

```
QPainter p;
p.setPen(Qt::NoPen);
p.setBrush(Qt::NoBrush);
```

- Changer de stylo et de pinceau a un coût
  - Plannifier le dessin pour gagner en performance

digia

QPainter : ses outils

- `QColor` ⇨ pour définir des couleurs

```
QColor c = QColor(red, green, blue, alpha=255);
```

  - Valeurs ∈ [0, 255]
  - alpha pour gérer la transparence
    - 255 ⇨ couleur opaque
    - 0 ⇨couleur transparente
  - Couleurs prédéfinies dans `enum Qt::GlobalColor`
    - `Qt::red`, `Qt::white`, etc.

digia

QPainter : ses outils

- `QColor` ⇨ utilise différents modèles de couleur
  - RGB
  - HSV
  - CMYK

```
QColor(255,0,0) // RGB
QColor::fromHsv(h,s,v) // HSV
QColor::fromCmyk(c,m,y,k) // CMYK
```

digia

QPainter : ses outils

- `QFont` ⇨ pour le texte
  - Famille
  - Taille
  - Gras / Italique / souligné / barré / etc.

Hello Qt!

Hello Qt!

Hello Qt!

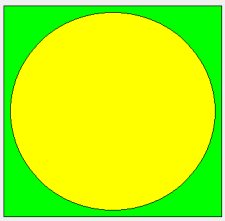
~~Hello Qt!~~

Hello Qt!

Hello Qt!

digia

Affichage de base



- Dans `paintEvent` ⇨ pour persistance
  - Stylo `QPen` par défaut ⇨ noir

```
void RectWithCircle::paintEvent(QPaintEvent *ev)
{
    QPainter p(this);

    p.setBrush(Qt::green);
    p.drawRect(10, 10, width()-20, height()-20);
    p.setBrush(Qt::yellow);
    p.drawEllipse(20, 20, width()-40, height()-40);
}
```

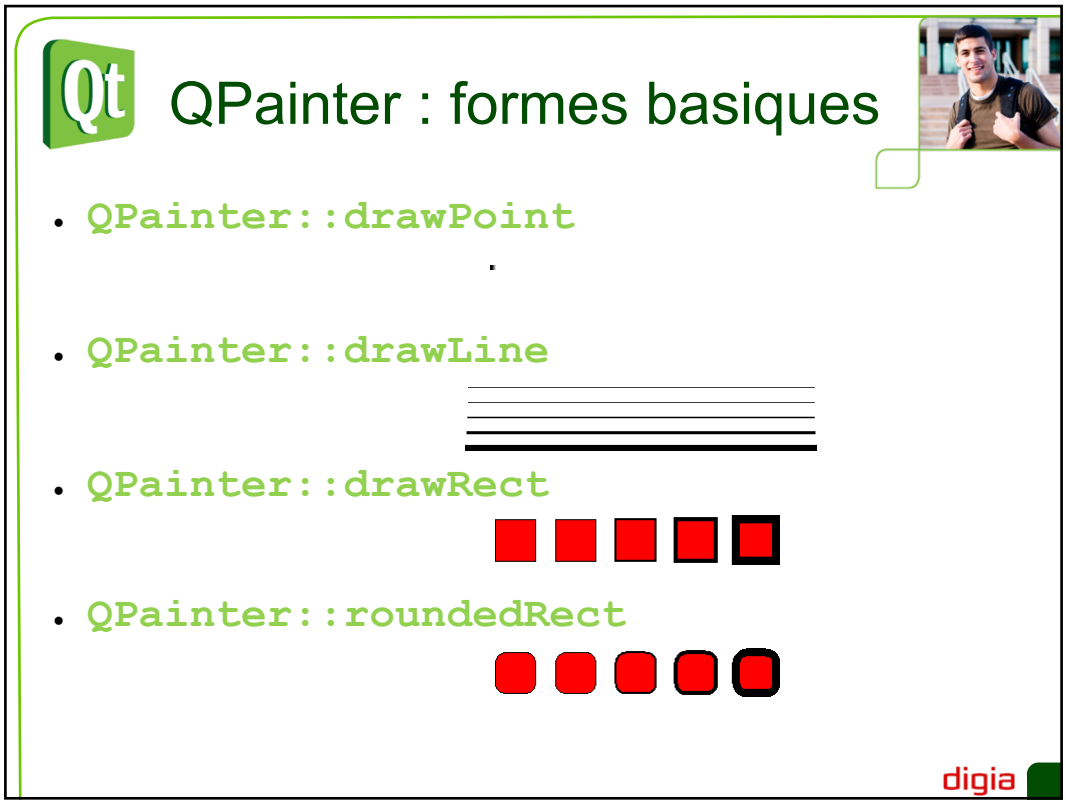
- Planifier l'affichage si en dehors de paintEvent
  - `update()` ⇨ dans prochain paintEvent
  - `repaint()` ⇨ immédiatement

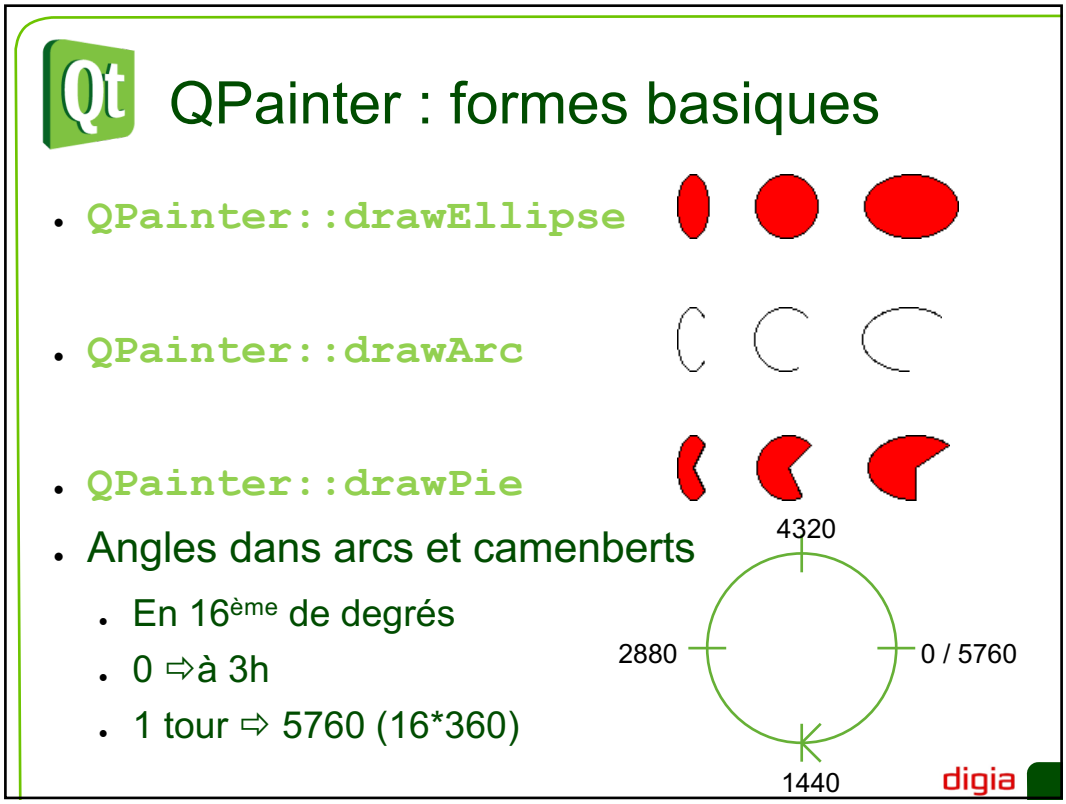
digia

Affichage sur widget : attention !

- Dans `paintEvent` ⇨ OBLIGATOIRE
- QPainter ne peut pas être utilisé ailleurs
- Que faire si tracés à faire en dehors?
  - Mettre à jour dans un modèle ou dans une image
  - Forcer le redessin ⇨ `repaint()`
    - `paintEvent` appelé
    - Tracer dans le widget : `QPainter p(this)`
      - En parcourant le modèle
      - Ou en affichant l'image : `p.DrawPixmap(img)`

digia


QPainter : formes basiques
QPainter::drawPoint
QPainter::drawLine
QPainter::drawRect
QPainter::roundedRect
digia


QPainter : formes basiques
QPainter::drawEllipse
QPainter::drawArc
QPainter::drawPie
Angles dans arcs et camenberts
En 16ème de degrés
0 ⇨à 3h
1 tour ⇨ 5760 (16*360)
4320
2880
0 / 5760
1440
digia

QPainter : texte

Hello World!
Hello World!
Hello World!
Hello World!

- `QPainter::drawText`

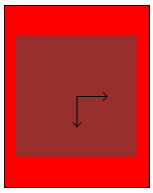
```
QPainter p(this);

QFont font("Helvetica");
p.setFont(font);
p.drawText(20, 20, 120, 20, 0, "Hello World!");

font.setPixelSize(10);
p.setFont(font);
p.drawText(20, 40, 120, 20, 0, "Hello World!");

font.setPixelSize(20);
p.setFont(font);
p.drawText(20, 60, 120, 20, 0, "Hello World!");

QRect r;
font.setPixelSize(20);
p.setFont(font);
p.setPen(Qt::red);
p.drawText(20, 80, 120, 20, 0, "Hello World!", &r);
```

Rectangle englobant le texte

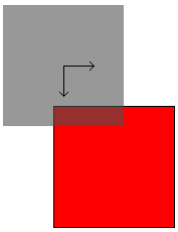
digia

QPainter : transformations

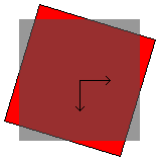
- `QPainter::scale`



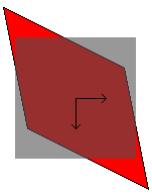
- `QPainter::translate`



- `QPainter::rotate`
  - `Angle ⇨ sens horaire`



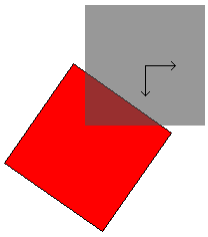
- `QPainter::shear`



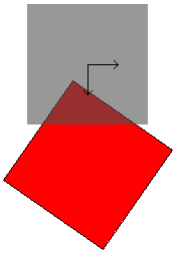
digia

QPainter : transformations

- Attention à
  - l'ordre des transformations
  - l'origine pour échelle, rotation et étirement



```
p.translate(0, 100);
p.rotate(35);

p.drawRect(-60, -60, 120,
120);
```



```
p.rotate(35);
p.translate(0, 100);

p.drawRect(-60, -60, 120,
120);
```

digia

QPainter : transformations

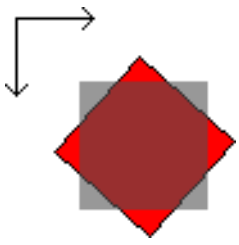
- `save` ⇨ stocke dans pile les transformations
- `restore` ⇨ annule les transformations de la pile
- Exemple : rotation autour d'un point quelconque

```
QPoint rotCenter(50, 50);
qreal angle = 42;
p.save();
p.translate(rotCenter);
p.rotate(angle);
p.translate(-rotCenter);
p.setBrush(Qt::red);
p.setPen(Qt::black);
p.drawRect(25,25, 50, 50);
p.restore();
p.setPen(Qt::NoPen);
p.setBrush(QColor(80, 80, 80, 150));
p.drawRect(25,25, 50, 50);
```

Transformations appliquées

Rectangle rouge

Rectangle gris

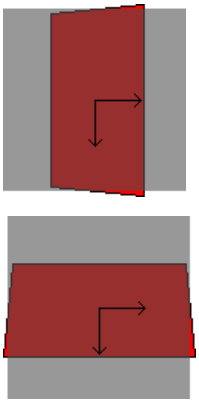


digia

QPainter : transformations 2.5D

- Rotation selon un axe ⇨ crée effet 3D

```
QTransform t;
t.rotate(60, Qt::YAxis);

painter.setTransform(t, true);
```



digia

QPainterPath

- Conteneur pour des opérations de dessin
  - Permet de construire et réutiliser des formes

```
QPainterPath path;
path.addRect(20, 20, 60, 60);

path.moveTo(0, 0);
path.cubicTo(99, 0,  50, 50,  99, 99);
path.cubicTo(0, 99,  50, 50,  0, 0);

QPainter painter(this);
painter.fillRect(0, 0, 100, 100, Qt::white);
painter.setPen(QPen(QColor(79, 106, 25), 1, Qt::SolidLine,
                    Qt::FlatCap, Qt::MiterJoin));
painter.setBrush(QColor(122, 163, 39));

painter.drawPath(path);
```

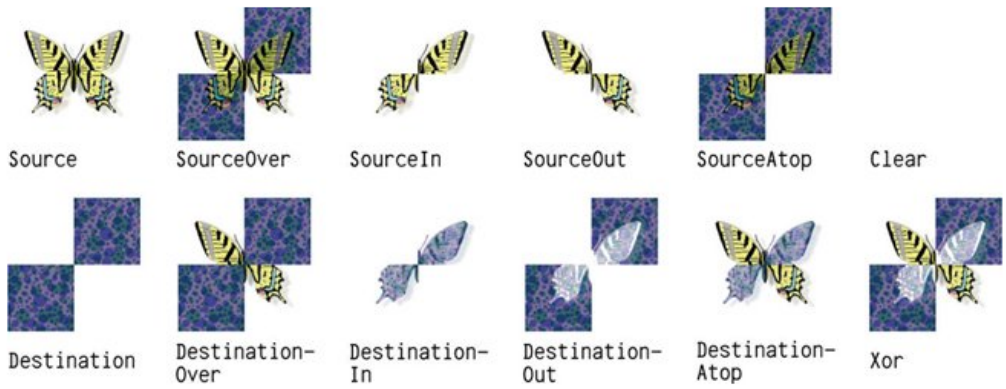
digia

QImage / QPixmap

- Compromis entre vitesse et précision
  - Efficacité ⇨ moteur de dessin natif plateforme
    - `QWidget`
    - `QPixmap` ⇨ optimisé pour afficher image sur écran
  - Précision ⇨ moteur interne Qt, multiplateforme
    - `QImage` ⇨ optimisé pour la manipulation
    - Prise en charge des modes de composition
      - Définit fusion entre pixel source et destination
      - Méthode `QPainter::setCompositionMode()`

QImage / QPixmap

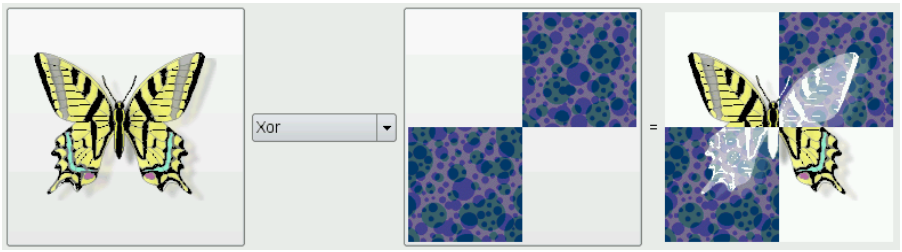
- Les différents modes de composition



QImage / QPixmap

- Exemple : papillon XOR damier

```
QImage resultimage = checkerPatternImage;
QPainter painter(&resultImage);
Painter.setCompositionMode(QPainter::CompositionMode_Xor);
Painter.drawImage(0, 0, butterflyImage);
```



QImage / QPixmap

- Conversion QImage ⇔ QPixmap

```
QImage QPixmap::toImage();

QPixmap QPixmap::fromImage( const QImage& );
```

- Chargement et sauvegarde

```
QPixmap pixmap( "image.png" );
pixmap.save( "image.jpeg" );
```

```
QImage image( "image.png" );
image.save( "image.jpeg" );
```

QImage / QPixmap

- Afficher dans QImage ⇨ avec QPainter

```
QImage image( 100, 100, QImage::Format_ARGB32 );
QPainter painter(&image);

painter.setBrush(Qt::red);

painter.fillRect(image.rect(), Qt::white );
painter.drawRect(image.rect().adjusted( 20, 20, -20, -20 ) );
```

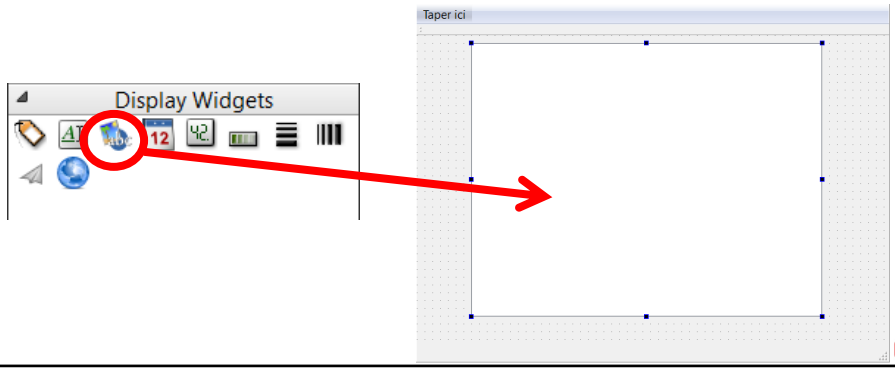
- QPixmap optimisé pour être affiché à l'écran

```
void MyWidget::imageChanged( const QImage &image )
{   pixmap = QPixmap::fromImage( image );
    update();
}

void MyWidget::paintEvent( QPaintEvent* )
{    QPainter painter( this );
    painter.drawPixmap( 10, 20, pixmap );
}
```

digia

La vue ⇨ QGraphicsView

- Surface pour gérer et interagir avec une scène 2D
  - Zoom
  - Rotation
  - Barre de défilement si scène trop grande



digia 34

La scène ⇨ QGraphicsScene

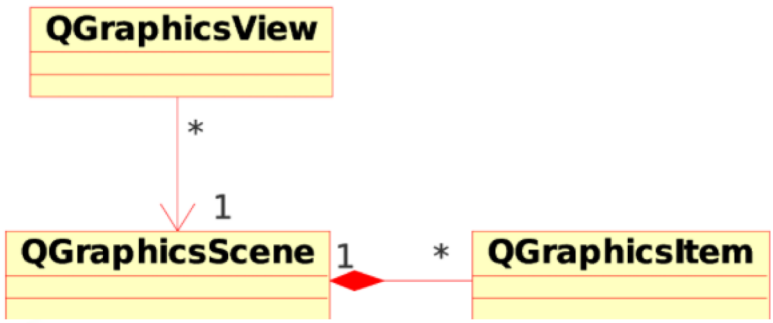
- La scène gère des éléments QGraphicsItems
  - Interface rapide pour un grand nombre d'éléments
  - Propage les événements à ses éléments
    - Souris, clavier, déplacement
  - Gère l'état des éléments (sélection, focus)
  - Fournit des fonctionnalités de rendu
- Association Scene ⇔ Vue

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),ui(new Ui::MainWindow)
{
   ui->setupUi(this);
    scene = new QGraphicsScene(this);
    ui->graphicsView->setScene(scene);
…}
```

digia 35

La scène ⇨ QGraphicsScene

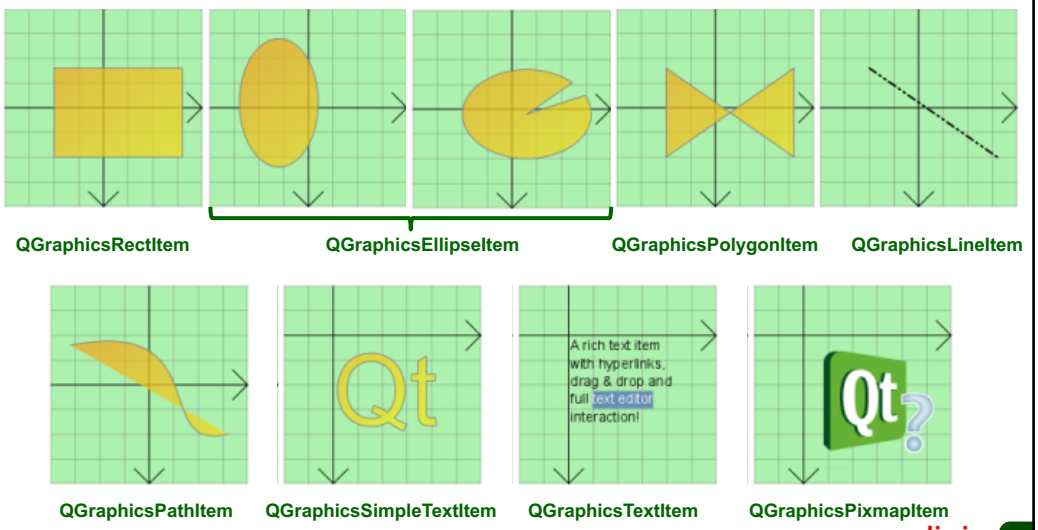
- QGraphicsScene est
  - Un modèle pour QGraphicsView
  - Un conteneur pour QGraphicsItems



digia 36

Les éléments ⇨ QGraphicsItem

- Différents éléments possibles

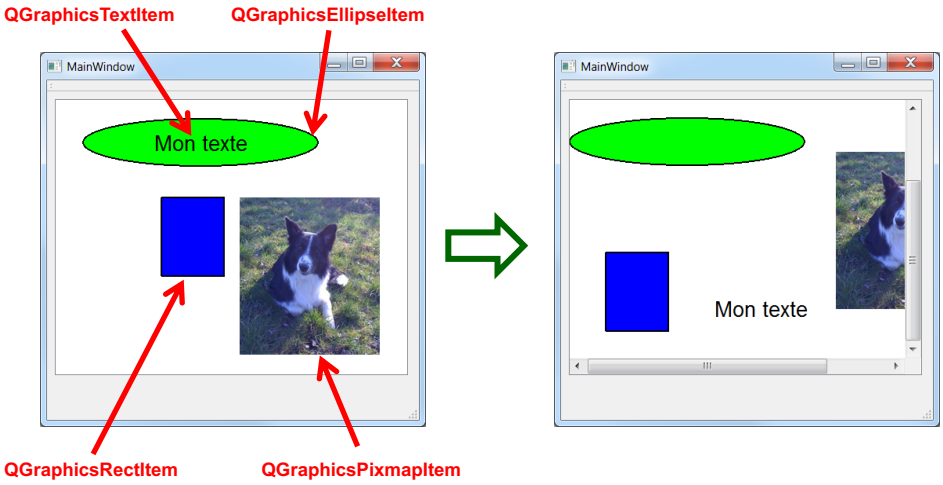


Les éléments ⇨ QGraphicsItem

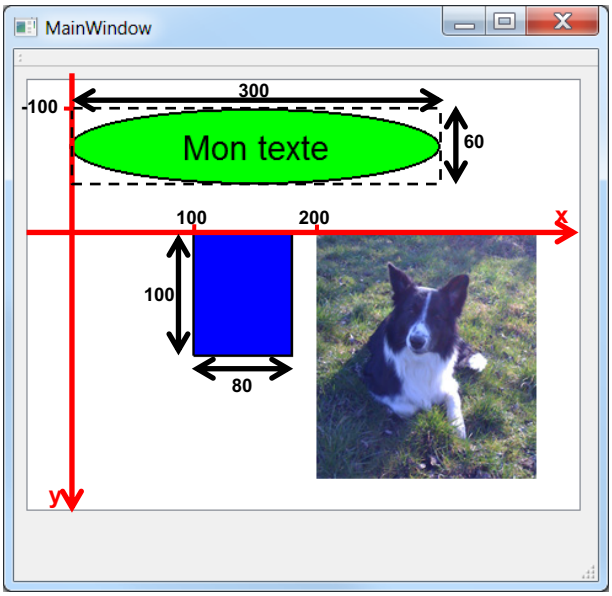
- Un élément ne peut appartenir qu'à une scène
- Ajouter un élément dans la scène
  - Méthode QGraphicScene::Add<XXX>(…)
    - Spécifique à l'élément
- Caractéristiques de l'élément
  - Méthode QGraphicsItem::setFlags(…)
    - QGraphicsItem::ItemIsMovable ⇨ élément déplaçable
    - QGraphicsItem::ItemIsSelectable ⇨ élément sélectionnable
    - Etc.

QGraphicsView : exemple

- Une scène avec 4 éléments dont 3 déplaçables



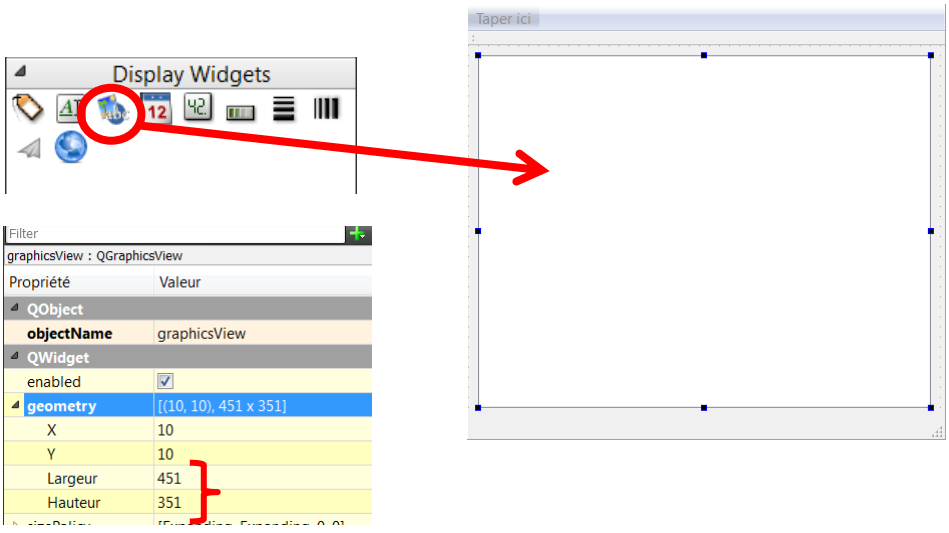
QGraphicsView : exemple



QGraphicsView : exemple

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
    QGraphicsEllipseItem *ellipse;
    QGraphicsRectItem *rectangle;
    QGraphicsTextItem *texte;
    QGraphicsPixmapItem *image;

};
```


QGraphicsView : exemple



QGraphicsView : exemple

- Création de la scène dans la vue

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    scene = new QGraphicsScene(this);
    ui->graphicsView->setScene(scene);

    QBrush greenBrush(Qt::green);
    QBrush blueBrush(Qt::blue);
    QPen outlinePen(Qt::black);
    outlinePen.setWidth(2);
```

Outils pour dessiner

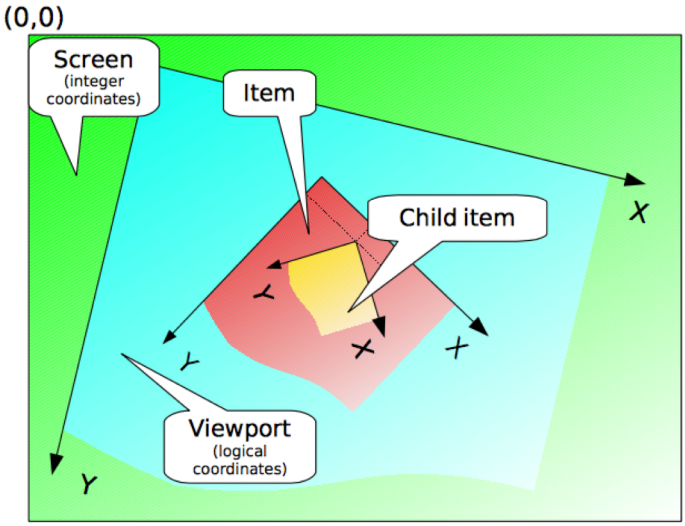
QGraphicsView : exemple

- Création des éléments dans la scène

```
rectangle = scene->addRect(100, 0, 80, 100, outlinePen, blueBrush);
rectangle->setFlag(QGraphicsItem::ItemIsMovable);

ellipse = scene->addEllipse(0, -100, 300, 60, outlinePen, greenBrush);

texte = scene->addText("Mon texte", QFont("Arial", 16) );
texte->setFlag(QGraphicsItem::ItemIsMovable);
QRectF boiteT = texte->boundingRect();
QRectF boiteE = ellipse->boundingRect();
texte->setPos((boiteE.width()-boiteT.width())/2,
              boiteE.y()+(boiteE.height()-boiteT.height())/2);

image = scene->addPixmap(QPixmap("C:\\Images\\Chien.jpg", "JPG"));
image->setPos(200, 0);
image->setFlag(QGraphicsItem::ItemIsMovable);}
```

QGraphicsView : repères

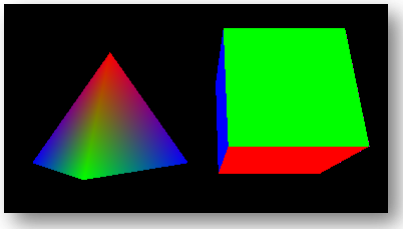
- Chaque vue/item a un repère local



QGraphicsView : repères

- Coordonnées locales à un élément
  - Coordonnées logiques, pas pixel
  - Point réel, pas entier
  - Sans transformation, 1 coord logique = 1 pixel
- Les éléments sont redessinés récursivement
  - Parent ⇨ enfant

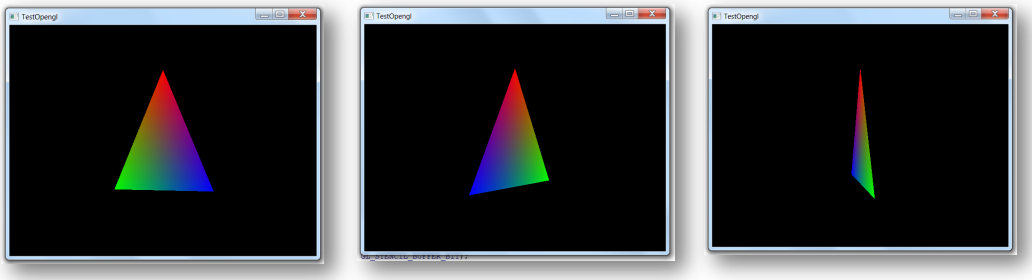
Qt et OpenGL



- Bibliothèque QtOpenGL
- Créer une classe dérivée de
  - QOpenGLFunctions
    - Indépendant OpenGL ou OpenGL ES 2.0
  - QWindow
- Lui indiquer d'utiliser le rendu OpenGL
  - Méthode setSurfaceType
  - Définir le type QSurface::OpenGLSurface
- Surcharger initialize()
  - Pour les initialisations OpenGL

Qt et OpenGL

- Surcharger les méthodes de redessin
  - Y définir les commandes OpenGL
- Exemple complet d'un triangle animé qui tourne
  - http://qt-project.org/doc/qt-5/qtgui-openglwindow-example.html



Qt et OpenGL : QGLWidget

- Pour utiliser OpenGL dans application QT
- Créer une classe dérivée de QGLWidget
- Y surcharger
  - paintGL() ⇨ afficher la scène OpenGL
    - Appelé dès raffraichissement ou appel à updateGL()
  - resizeGL() ⇨ mise à jour si widget redimensionnée
    - Zone d'affichage ⇨ glViewport(…)
    - Projection, …
  - initializeGL() ⇨ initialiser OpenGL
    - Contexte
    - Liste d'affichage

Qt et OpenGL : QGLWidget

```
class MyGLDrawer : public QGLWidget
{
    Q_OBJECT        // must include this if you use Qt signals/slots

public:
    MyGLDrawer(QWidget *parent)
        : QGLWidget(parent) {}

protected:

    void initializeGL()
    {
        // Set up the rendering context, define display lists etc.:
        ...
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glEnable(GL_DEPTH_TEST);
        ...
    }
```

Qt et OpenGL : QGLWidget

```
    void resizeGL(int w, int h)
    {
        // setup viewport, projection etc.:
        glViewport(0, 0, (GLint)w, (GLint)h);
        ...
        glFrustum(...);
        ...
    }

    void paintGL()
    {
        // draw the scene:
        ...
        glRotatef(...);
        glMaterialfv(...);
        glBegin(GL_QUADS);
        glVertex3f(...);
        glVertex3f(...);
        ...
        glEnd();
        ...
    }

};
```


Qt et OpenGL : QGLWidget

- Un exemple

http://www.bogotobogo.com/Qt/Qt5_OpenGL_QGLWidget.php