Forum Nokia

Qt – External Interaction and Graphics

Andreas Jakl

Senior Technical Consultant
Forum Nokia

20 September, 2010
v3.0.0

Contents

- Events
- Low Level Painting
- Graphics View Framework
- Optimizing Images

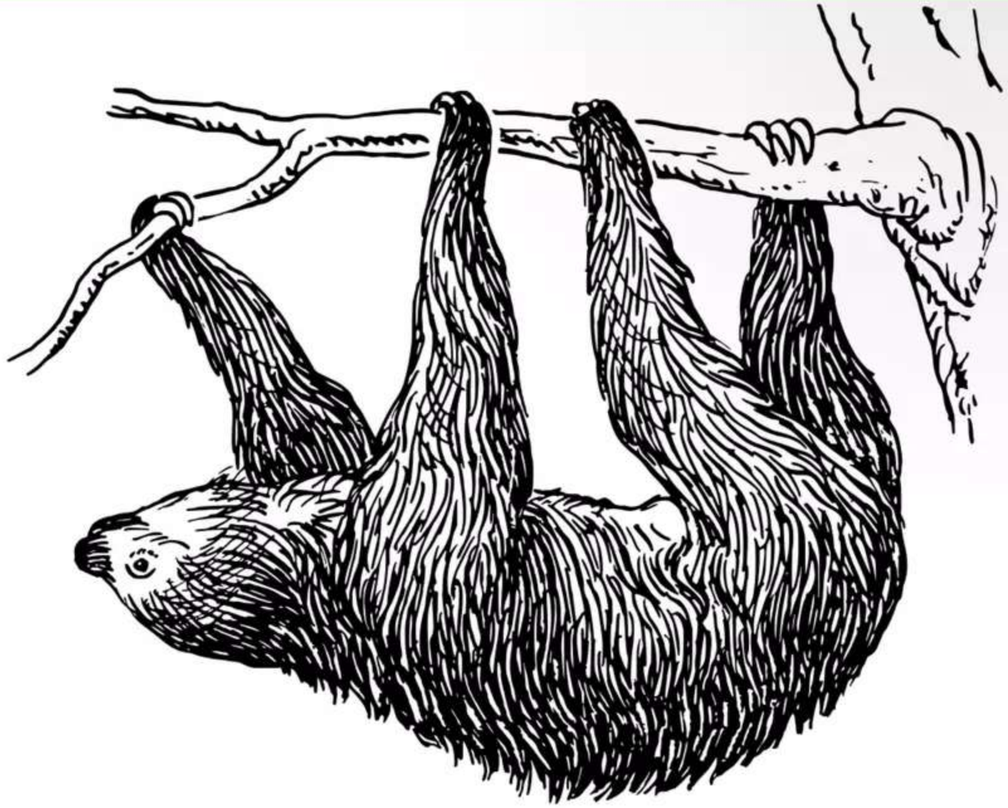
Events

Events

- **UI applications are event-based**
  - Wait for user to interact
  - Then start working on the requested task
- **Alternative**
  - Polling for status changes

    "Is the user pressing a button?"

→ Events save processing time, thus battery life



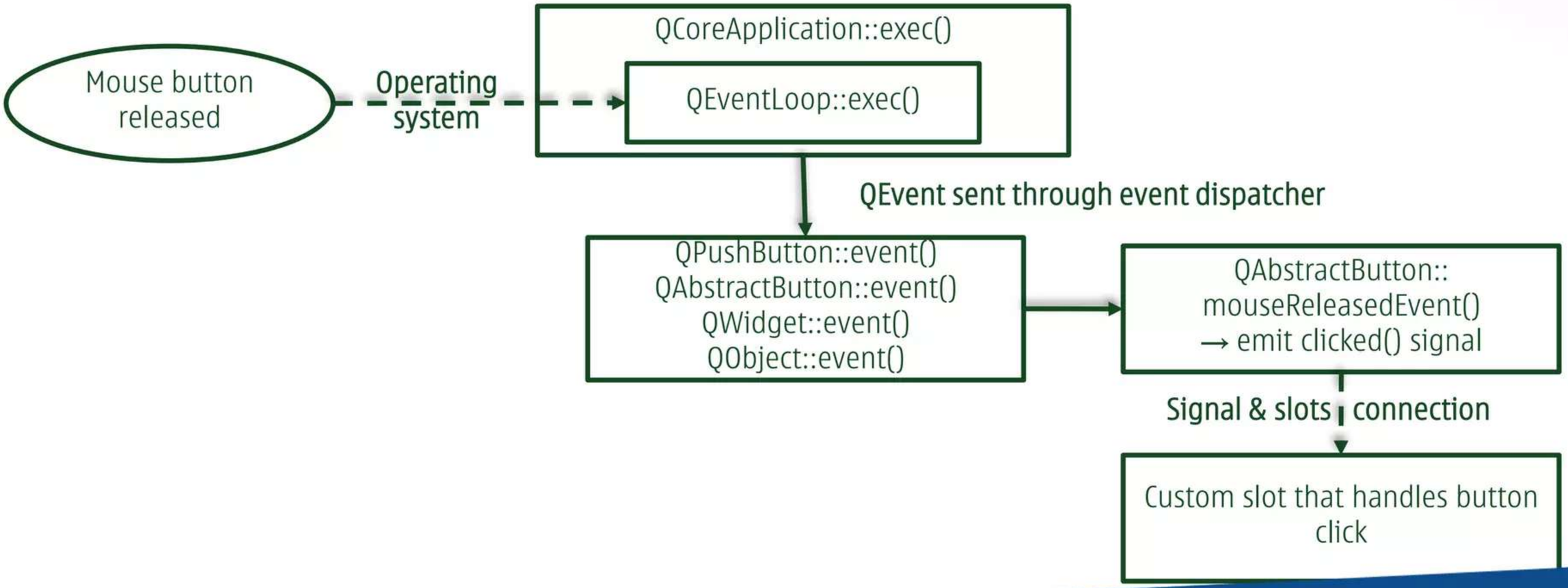
Picture credits: Pearson Scott Foresman
Public domain

Events vs. Signals?

- **Signals**
  - For **using widgets**
  - e.g., push button should return clicked() signal, not low-level mouse or key events
- **Events**
  - For **implementing widgets**
    - Modifying existing widgets
    - New widgets
  - e.g., implement own push button: handle mouse and key events and emit clicked() signal when necessary

Events and Signals

- **Mouse button's way to become a clicked() signal**



Event Loop

- **Threads in Qt can have an event loop**
  - Initial thread starts its loop through QCoreApplication::exec()
  - Other threads: QThread::exec()
- **Events**
  - **Derived from QEvent**
  - Source:
    - **Window system** (QMouseEvent, QKeyEvent)
    - **Generated by Qt itself** (QTimerEvent)
  - Handled by **QObject subclass** (especially widgets)

QEvent

- **Get type of event through QEvent::type()**
  - 100+ events defined through enum values in Qt
  - Own events can be defined
- **Event is an instance of a QEvent subclass**
  - **Adds additional information** about event
  - e.g., QMouseEvent: buttons, position, ...

Handling Events in Widgets

- Events delivered to event() function of QObject
- Calls appropriate virtual event handler functions

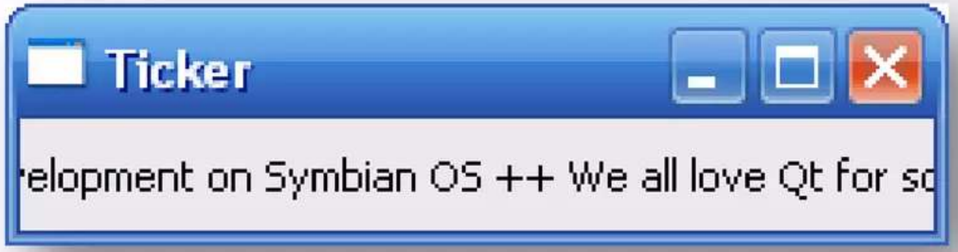
qwidget.cpp

```
bool QWidget::event(QEvent *event)
{
    Q_D(QWidget);
    // ...
    switch (event->type()) {
    case QEvent::MouseMove:
        mouseMoveEvent((QMouseEvent*)event);
        break;
    case QEvent::Paint:
        paintEvent((QPaintEvent*)event);
        break;
    case QEvent::Close:
        closeEvent((QCloseEvent*)event);
        break;
        // ...
    default:
        return QObject::event(event);
    }
    return true;
}
```

You only need to override virtual, pre-defined handler method from QWidget base class

Example: Ticker

- **Scrolling text**
  - 1 Pixel / 30 ms
  - Demonstrates Timer events
- **Handles 4 events**
  - **Paint event:** manual drawing of the text
  - **Timer event:** regular call-backs
  - **Show event:** start timer when widget is shown
  - **Hide event:** end timer when widget becomes invisible



Ticker – General Definition

ticker.h

```
#ifndef TICKER_H
#define TICKER_H

#include <QtGui>

class Ticker : public QWidget
{
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText)

public:
    Ticker(QWidget* parent = 0);
    void setText(const QString &newText);
    QString text() const { return m_text; }
    QSize sizeHint() const;

protected:
    void paintEvent(QPaintEvent* event);
    void timerEvent(QTimerEvent* event);
    void showEvent(QShowEvent* event);
    void hideEvent(QHideEvent* event);

private:
    QString m_text;
    int m_offset;
    int m_timerId;
};

#endif // TICKER_H
```

ticker.cpp

```
#include "ticker.h"

Ticker::Ticker(QWidget* parent)
        : QWidget(parent)
{
    m_offset = 0;
    m_timerId = 0;
}

void Ticker::setText(const QString &newText)
{
    m_text = newText;
    update();
    updateGeometry();
}

// Return recommended size of this widget
QSize Ticker::sizeHint() const
{
    // Determine size required by the text
    // First argument isn't needed here -> 0
    return fontMetrics().size(0, text());
}
```

Ticker – Event Handling

ticker.cpp

```
void Ticker::paintEvent(QPaintEvent* /*event*/)
{
    QPainter painter(this);

    // Get required width of the text
    int textWidth = fontMetrics().width(text());
    if (textWidth < 1)
        return;

    // Draw the text as many times as necessary
    // to fill the entire width of the widget
    // (taking offset into account)
    int x = -m_offset;
    while (x < width())
    {
        painter.drawText(x, 0, textWidth, height(),
            Qt::AlignLeft | Qt::AlignVCenter, text());
        x += textWidth;
    }
}

void Ticker::showEvent(QShowEvent* /*event*/)
{
    // Starts a timer. Returned ID number can be
    // used to identify timer later
    m_timerId = startTimer(30);
}
```

ticker.cpp

```
void Ticker::timerEvent(QTimerEvent* event)
{
    // Called by the system at intervals
    if (event->timerId() == m_timerId)
    {
        // Increase offset by 1 to simulate movement
        ++m_offset;
        if (m_offset >= fontMetrics().width(text()))
            m_offset = 0;
        // Call to QWidget::scroll(), moves existing
        // pixels on-screen and only paints new area.
        // Also possible: call update() to redraw
        // the whole widget.
        scroll(-1, 0);
    } else {
        // Event not from the timer we are
        // interested in -> pass to base class
        QWidget::timerEvent(event);
    }
}

void Ticker::hideEvent(QHideEvent* /*event*/)
{
    // Stop the timer
    killTimer(m_timerId);
    m_timerId = 0;
}
```

Event Filters

- **Look at / intercept events delivered to other object**
  - e.g., dialogs filter key presses for widgets to modify Return-key handling
  - Set up through `QObject::installEventFilter()`
  - **Target object gets events before monitored object**
  - **Accepts or rejects events:** allow / deny further event processing

```
monitoredObj->installEventFilter(targetObj);
```

Ways to Influence Event Handling I

1. **Incoming event first goes to virtual function QCoreApplication::notify()**
   - → Override notify()
   - Get all events before any event filters can intercept
   - Only one subclass of QCoreApplication can be active at a time
2. **Install an event filter on QCoreApplication::instance()**
   - → implement eventFilter()
   - Get events after sent out by notify()
   - As powerful as option 1, also allows multiple application-global event filters

Ways to Influence Event Handling II

3. **Event filter on target object**
   - → implement eventFilter()
   - Gets all events (except Tab, Shift+Tab) before actual target object
4. **Event handler of target object**
   - → override QObject::event()
   - Gets events after event filter
   - Don't forget to call event handler of base class for unhandled events!
5. **Re-implement event handling functions**
   - → implement paintEvent(), mousePressEvent()
   - Easiest, **most common,** but least powerful

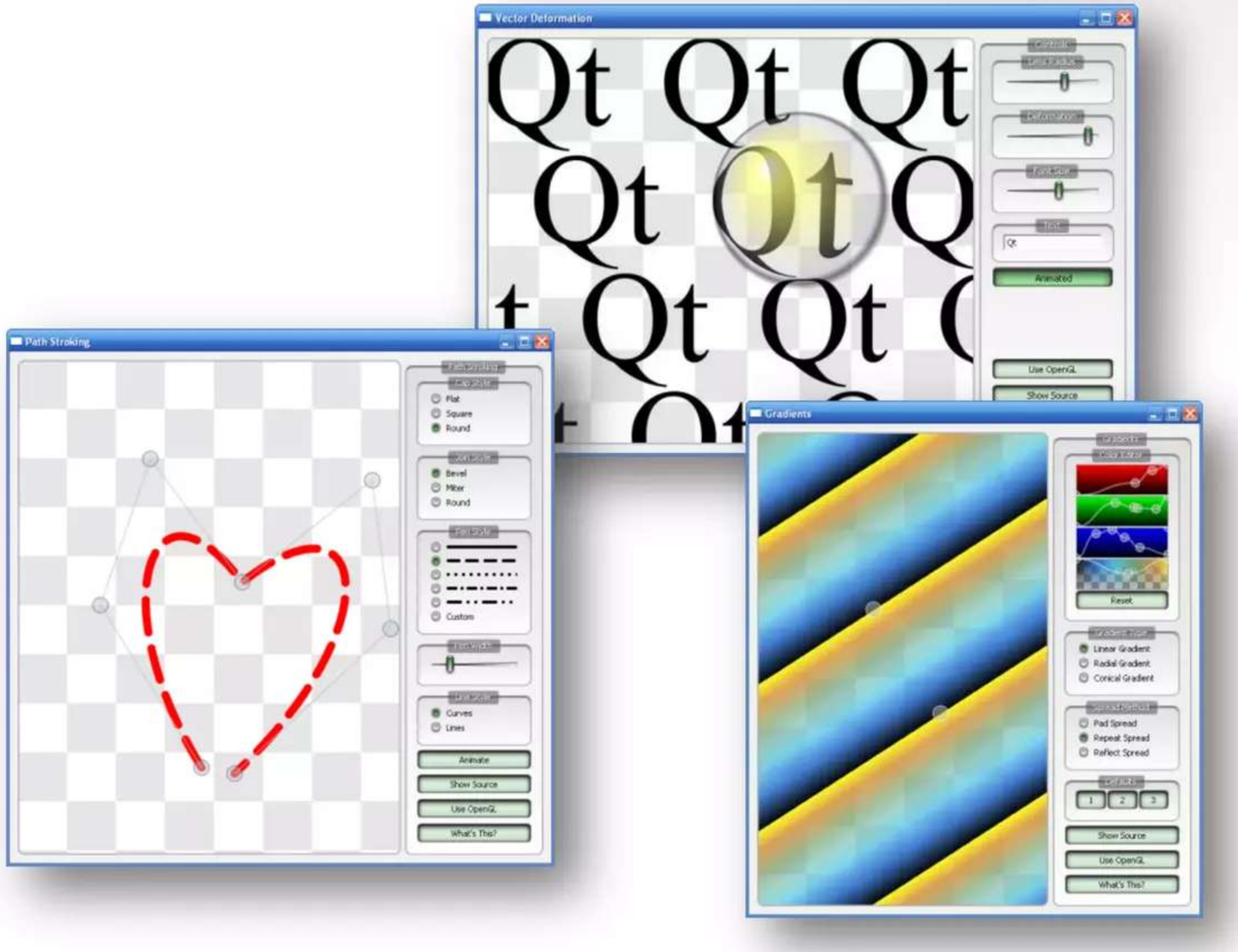
Forum Nokia

Painting

- **Low-level drawing handled through QPainter**
- **Draws on paint devices (QPaintDevice):**
  - **QWidget:** most common use case
  - **QImage:** optimized for I/O and direct pixel manipulation
  - **QPixmap:** optimized for showing images on screen
    - **QBitmap:** convenience for monochrome QPixmap – e.g., QCursor, QBrush, QRegion
  - **QPicture:** records and replays QPainter commands
  - **QPrinter:** paint on a printer
  - ...

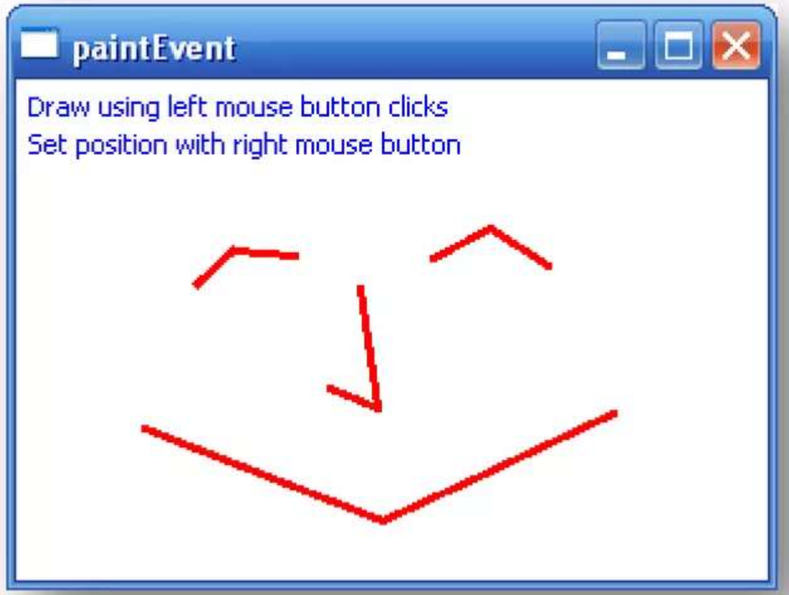
QPainter

- **Can draw:**
  - Geometric shapes with pen/brush
  - Fonts
  - Bezier curves
  - Images
- **Features:**
  - Anti-aliasing
  - Alpha blending
  - Gradient filling
  - Vector paths
  - Linear transformations



Example: Drawing

- Draws line between coordinates of two mouse clicks
- Uses image to store drawing
- Text written directly on Widget surface



mainWidget.cpp

```
MainWidget::MainWidget() {
    // Use standard colors of system
    setBackgroundRole(QPalette::Base);
    lastX = -1;
    lastY = -1;
    resize( 300, 200 );
    bgImg = new QPixmap(300, 200);
    bgImg->fill(QColor(255, 255, 255));
}

void MainWidget::mousePressEvent(QMouseEvent* event) {
    if (event->button() == Qt::LeftButton && lastX > -1 && lastY > -1) {
        // Create painter object for our image
        QPainter painter(bgImg);
        // Draw a line from previous pos to new position
        painter.setPen(QPen(Qt::red, 3));
        painter.drawLine(lastX, lastY, event->x(), event->y());
        update();
    }
    lastX = event->x();
    lastY = event->y();
}

void MainWidget::paintEvent(QPaintEvent* event) {
    // Paint directly on the widget surface
    QPainter painter(this);
    // Draw the image to the widget
    painter.drawPixmap(0, 0, *bgImg);
    // Draw text on top of the image to the widget
    painter.setPen(QPen(Qt::blue, 1));
    painter.drawText(5, 15, tr("Draw using left mouse button clicks"));
    painter.drawText(5, 30, tr("Set position with right mouse button"));
}
```

mainWidget.h

```
class MainWidget : public QWidget {
Q_OBJECT
public:
    MainWidget();
protected:
    void mousePressEvent(QMouseEvent* event);
    void paintEvent(QPaintEvent* event);
private:
    int lastX;
    int lastY;
    QPixmap* bgImg;
};
```

The Graphics View Framework

Forum Nokia

GraphicsView

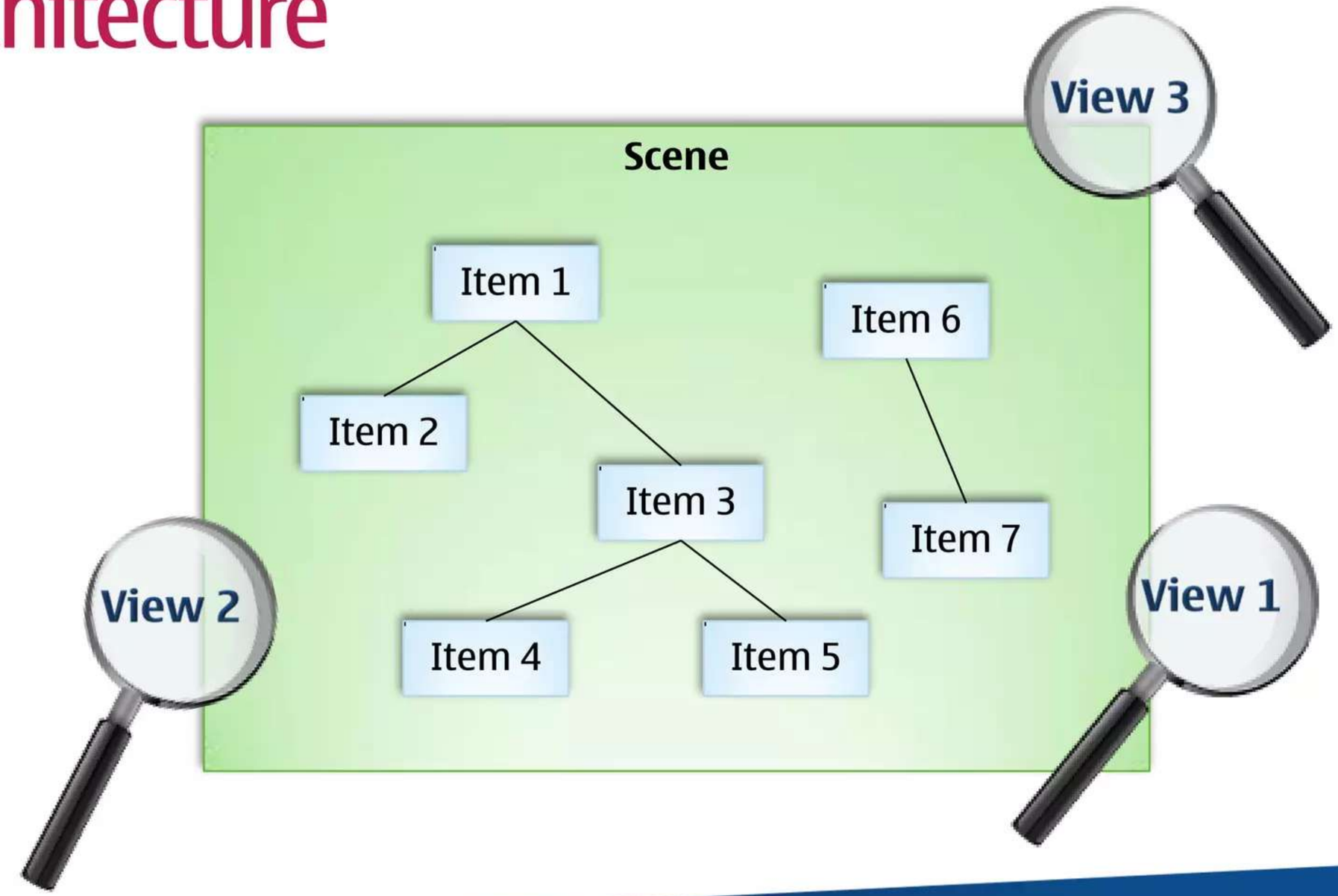
QPainter

- Good for individual graphics & widgets
- Static graphics, manual control

QGraphicsView

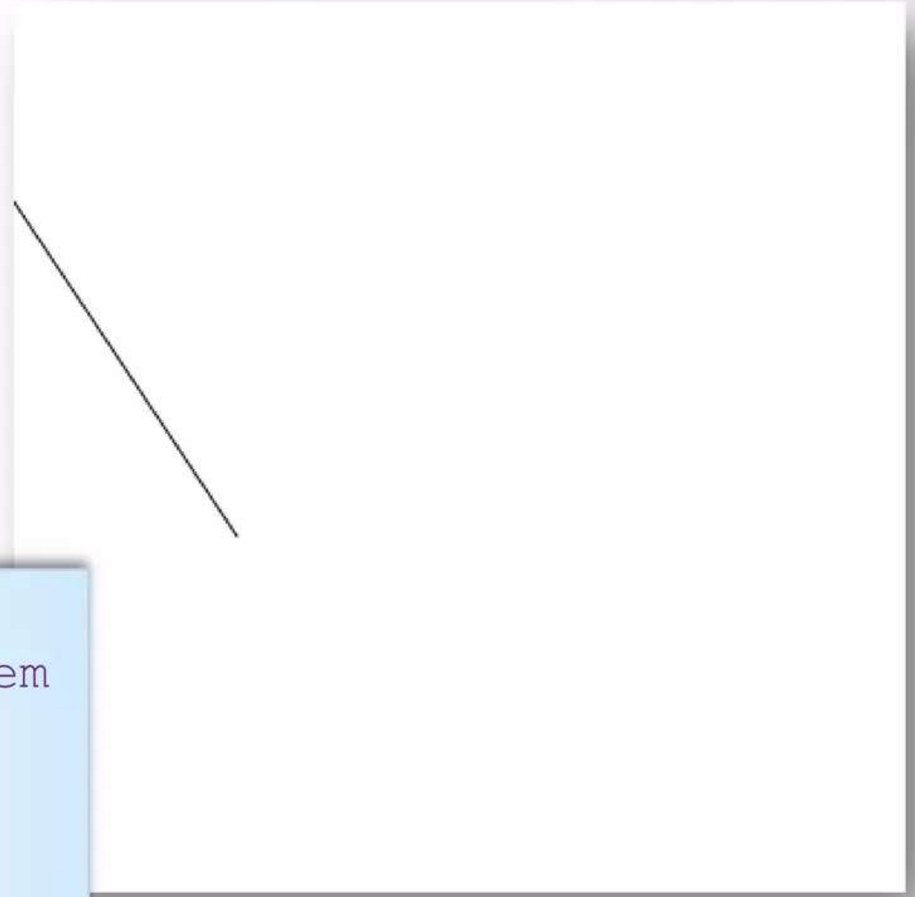
- *Manages multiple items (BSP-tree)*
- *Support for user interaction*
- *Animation*

Architecture



Graphics View Framework – Example

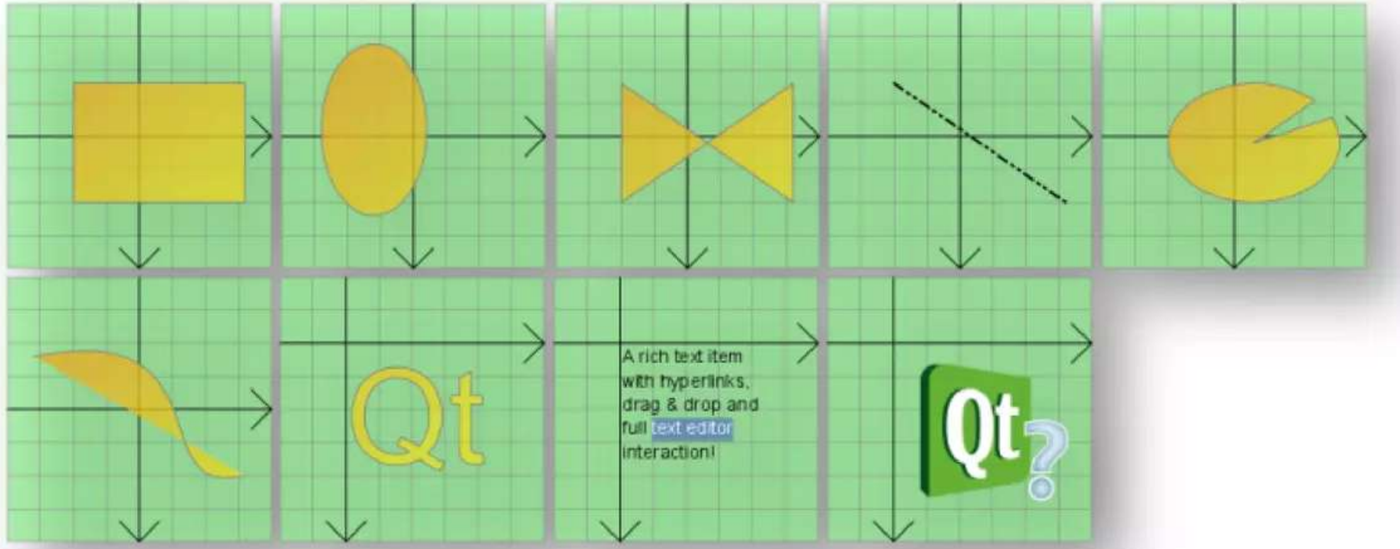
- **Setting up a simple scene**



```
scene = new QGraphicsScene(0, 0, 400, 400);
QGraphicsLineItem *wall = new QGraphicsLineItem
    (QLineF(0.0, 200.0, 100.0, 200.0));
scene->addItem(wall);
view = new QGraphicsView(scene);
view->setRenderHint(QPainter::Antialiasing);
```

Items

- **QGraphicsItem subclasses**
  - Pre-defined shapes (line, rectangle, polygon, path, etc.)
  - Text
  - Images
  - Widgets
  - Custom items
- **Supports**
  - Painting
  - Transformations
  - Collision Detection
  - Interaction through event handlers



Using Widgets as Items?

- **Possible through two ways:**
  - **QGraphicsWidget**
    - New subclass for widgets in `QGraphicsScene`s
    - Resembles `QWidget`, only few limitations
  - **QGraphicsProxyWidget**
    - Embeds a standard `QWidget` into `QGraphicsScene`
    - Translates coordinate systems
    - Automatically manages child widgets
    - Not suitable for high performance scenarios

Proxy Widget – Example


```
#include <QtGui>

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

    QPushButton *testButton = new QPushButton("Hello World");

    QGraphicsScene scene;
    QGraphicsProxyWidget *proxy = scene.addWidget(testButton);
    proxy->rotate(-20.0);

    QGraphicsView view(&scene);
    view.show();

    return app.exec();
}
```

Scene: QGraphicsScene

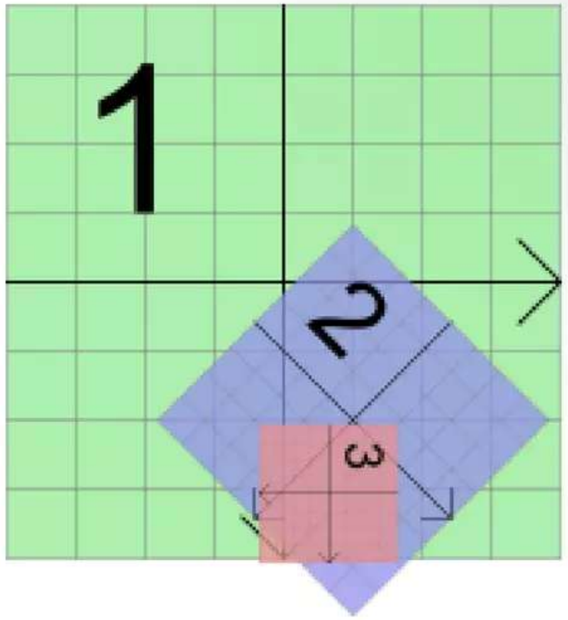
- **Surface for managing items on a 2D surface**
  - **3 layers:** background, item layer, foreground
  - Scene can be **parent of items**, or items are children of other items
  - **Manages items:** position & transformation, visibility, collision, locate items
  - Propagates **events** to items
  - **Indexes items** with BSP tree by default
    - Suitable for large scenes with relatively static items

View: QGraphicsView

- **Widget that presents a scene**
  - Multiple views for a single scene possible
  - Supports transformations (zooming, rotation, etc.)
  - Provides scroll bars if necessary
  - By default uses 2D paint engine, possible to use OpenGL

Image Credit: Ricardo J. Reyes
Public Domain

Coordinate Systems

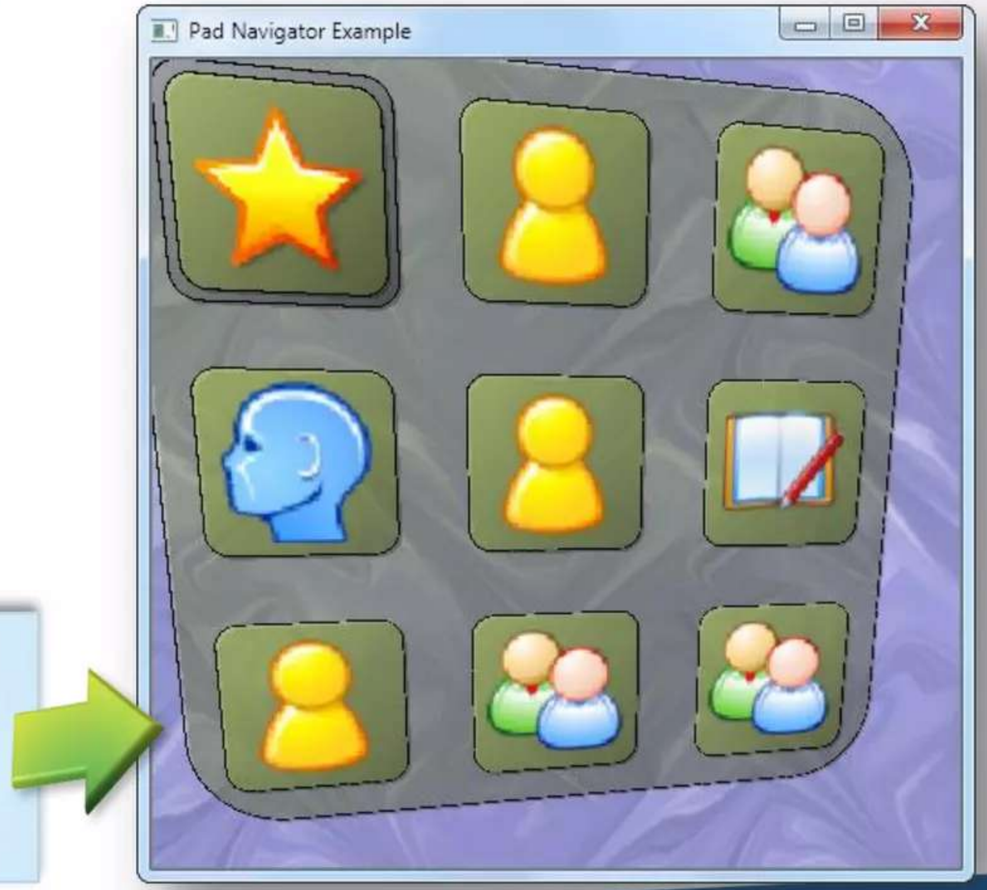


- **Item Coordinates**
  - Each item has its own coordinate system
  - Usually centered around its center point (0,0)
  - Relative to parent's coordinates
- **Scene Coordinates**
  - Base coordinate system for all items
  - Describes position for all top-level items
- **View Coordinates**
  - Coordinates relative to the widget

Coordinate Mappings / Transformations

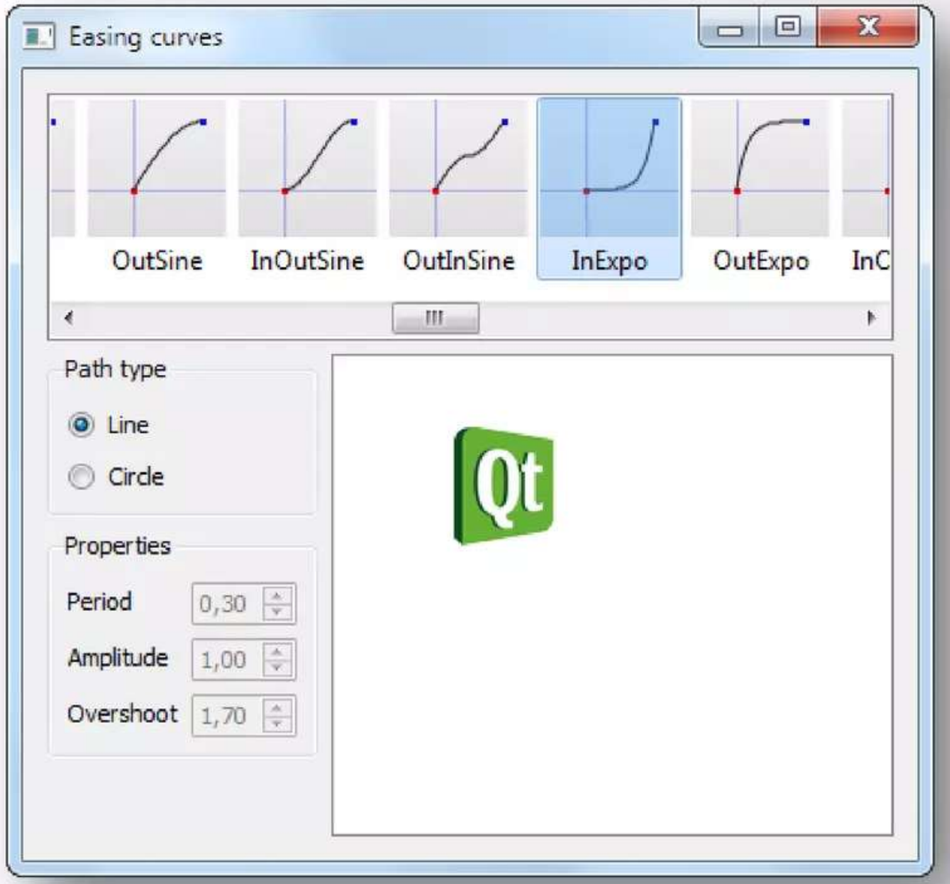
- **Coordinate mappings with utility functions**
  - `mapToScene()` / `mapFromScene()`
  - Available in view / scene / item classes
- **Affine Transformations**
  - Extra methods (`rotate()`, etc.)
  - `QMatrix`

```
QTransform transform;
transform.rotate(newPos.x() / 6.0, Qt::YAxis);
transform.rotate(newPos.y() / 6.0, Qt::XAxis);
baseItem->setTransform(transform);
```



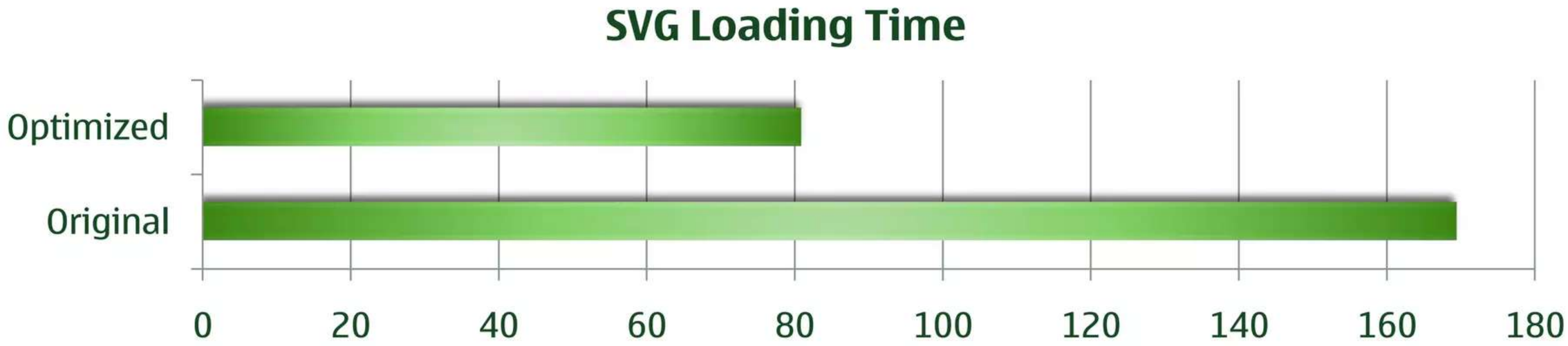
Animation Framework

- **Part of Qt Kinetic project (→ Qt 4.6)**
  - Animate Qt properties of widgets and `QObject`s
  - Can be used on its own or with state machine
  - Supports easing curves, animation groups



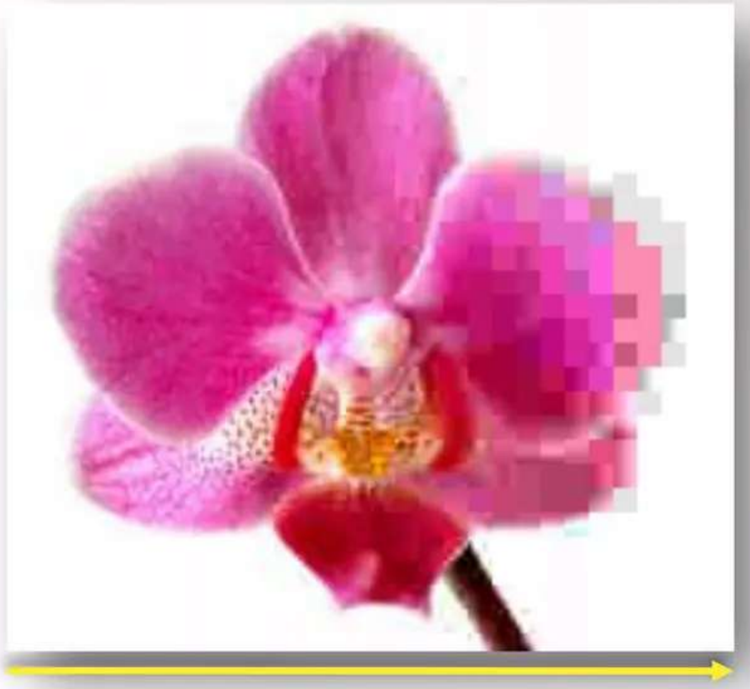
Optimizing SVG

- **Free tools**
  - http://code.google.com/p/svgmin/
  - http://codedread.com/scour/



Pixel Images

- **Most common pixel-based formats:**
  - **.png (Portable Network Graphics)**
    - Similar to .gif (which was patented until 2003 because of its LZW compression)
    - **Compression:** works well for graphics, not so well for photos
    - **Transparency:** support depends on device – 1 bit transparency or full alpha-channel.
  - **.jpg (Joint Photographic Experts Group)**
    - **Compression:** for photos (wavelength), not for graphics
    - **Transparency:** not supported

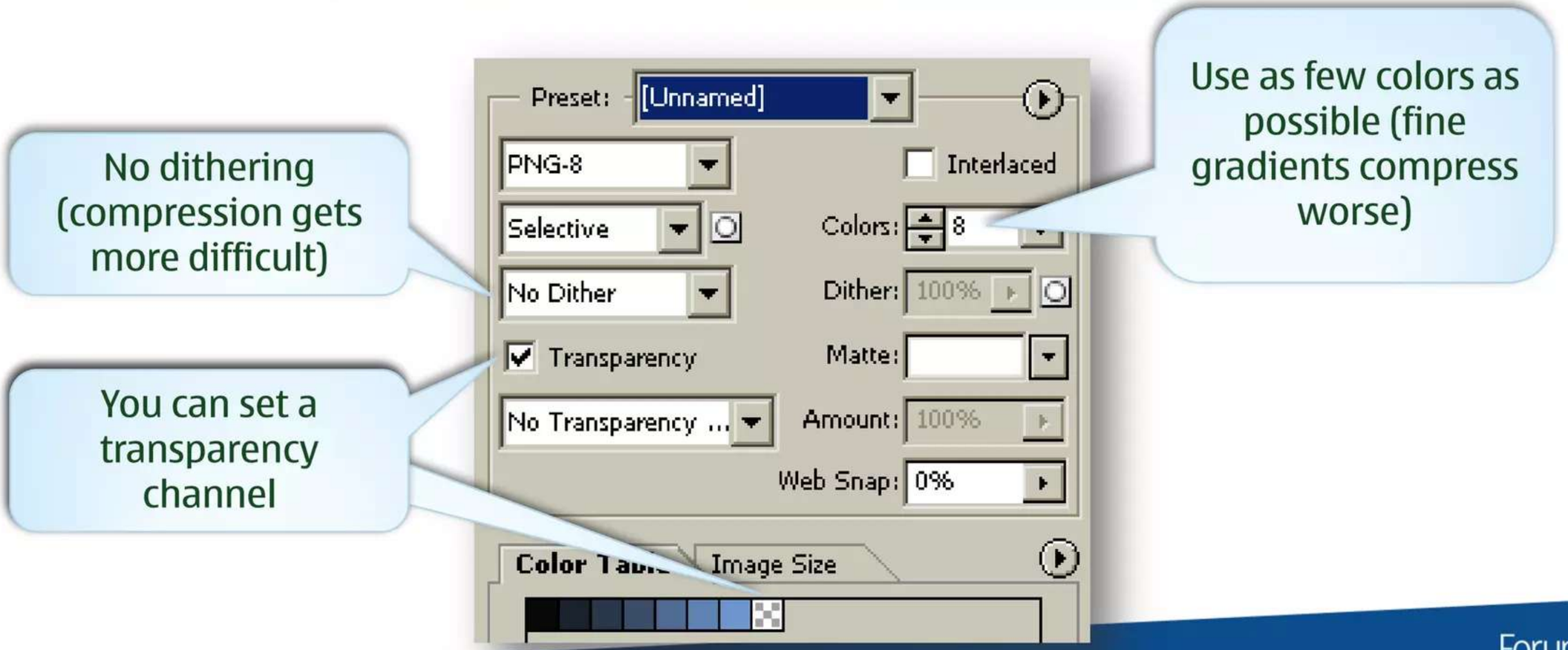


increasing JPEG compression

Image Credit: Ilmari_Karonen / Brianski
Creative Commons

Save PNGs – File Size Reduction

1. Optimized export – Photoshop: Save for Web
2. Further optimization – Pngcrush: http://pmt.sourceforge.net/pngcrush/



PNG Optimization – Example: 472 x 472 px

256 colours, no dither
30,0 kB

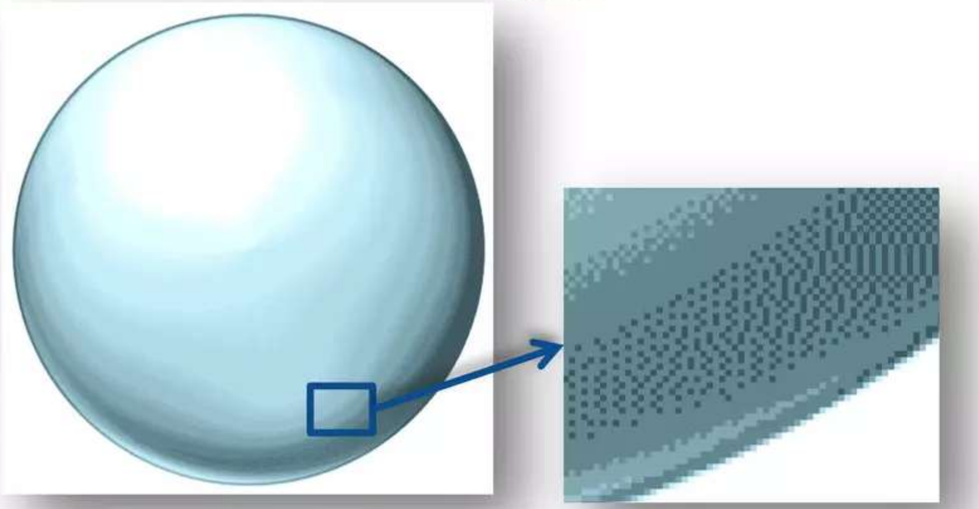
64 colours, no dither
16,3 kB

8 colours, no dither
6,3 kB

Results:

- Not much difference between 256 and 64 colours (especially on a device display), but only half of the file size
- Limits of optimization: 8 colours not enough
- Dithering at 8 colours: same file size as visually better 64 colours image

→ often, dithering is problematic!



8 colours, Diffusion dither
15,9 kB

Forum Nokia

Thank You.