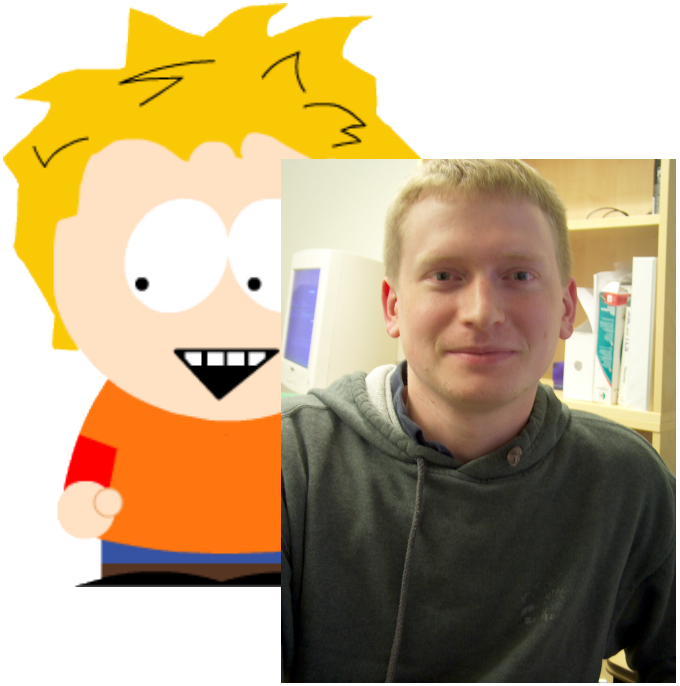
Graphics View

Andreas Aardal Hanssen

About the Presenter

- M. Sc. Andreas Aardal Hanssen / Bitto / bibr
- Senior Software Engineer / Team Lead Widgets
- In Trolltech since: 2003
- Graphics View architect

- Background
  - Open Source (Binc IMAP)
  - PC Demo Graphics (1996-2000)
  - Network Programming (2000-2003)



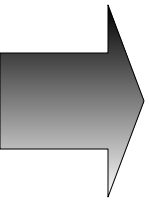
TROLLTECH DEV DAYS '07

Agenda

- Background
  - Why all modern toolkits need a powerful canvas
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all works
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

TROLLTECH DEV DAYS '07

Agenda



- **Background**
  - **Why all modern toolkits need a powerful canvas**
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all fits together
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

TROLLTECH DEV DAYS '07

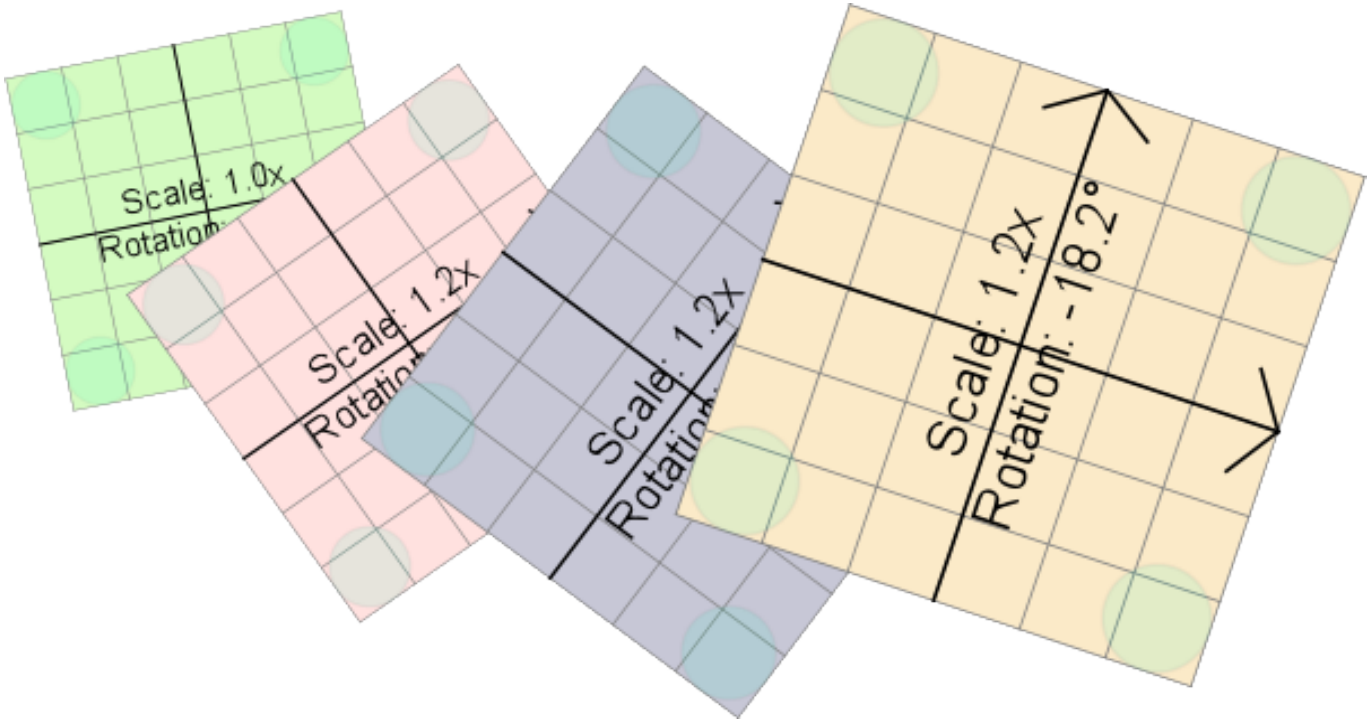
Background

- “Why do all modern toolkits need a canvas API?”

Background

- An artist's canvas
  - Cotton fiber, wooden frame, support for painting
- A canvas API
  - Abstract surface, high-level API, powerful painter

Background



Everybody loves graphics!

Background

Low Level or High Level?

```
glBegin(GL_LINES);
glVertex3f(6.0, 4.0, 2.0);
glVertex3f(2.0, -4.0, 3.3);
glVertex3f(5.0, 8.0, 8.0);
glVertex3f(-4.7, 5.0, -3.0);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(6.0, -1.0, -7.0);
glEnd();
```

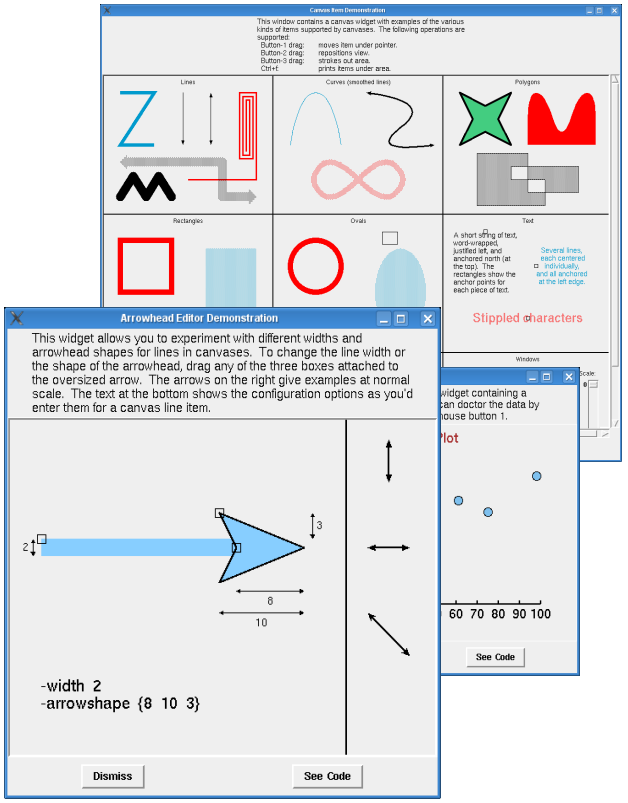
Background

- Low Level Graphics:
  - “Add ints to a pixel buffer”
  - Draw point, draw line, draw trigon strip, draw pixmap
  - Transform, translate, apply colors & brushes
  - Interface with hardware, upload subroutines, tailor graphics to fit with the graphics hardware
- Symptoms of Low Level APIs:
  - Hard to learn, hard to use?
  - Exposes limitations & benefits of the platform & hardware
  - “Very fast & lean”

Background

- High Level Graphics:
  - Create objects, assign textures, apply filters.
  - Move items around, stack them, group them together.
  - Connect graphical elements with other high-level (multimedia) compontents
- Symptoms of High Level APIs:
  - Easy to learn, easy to use?
  - Hides platform-specific details
  - “Slow & bloated”

Background

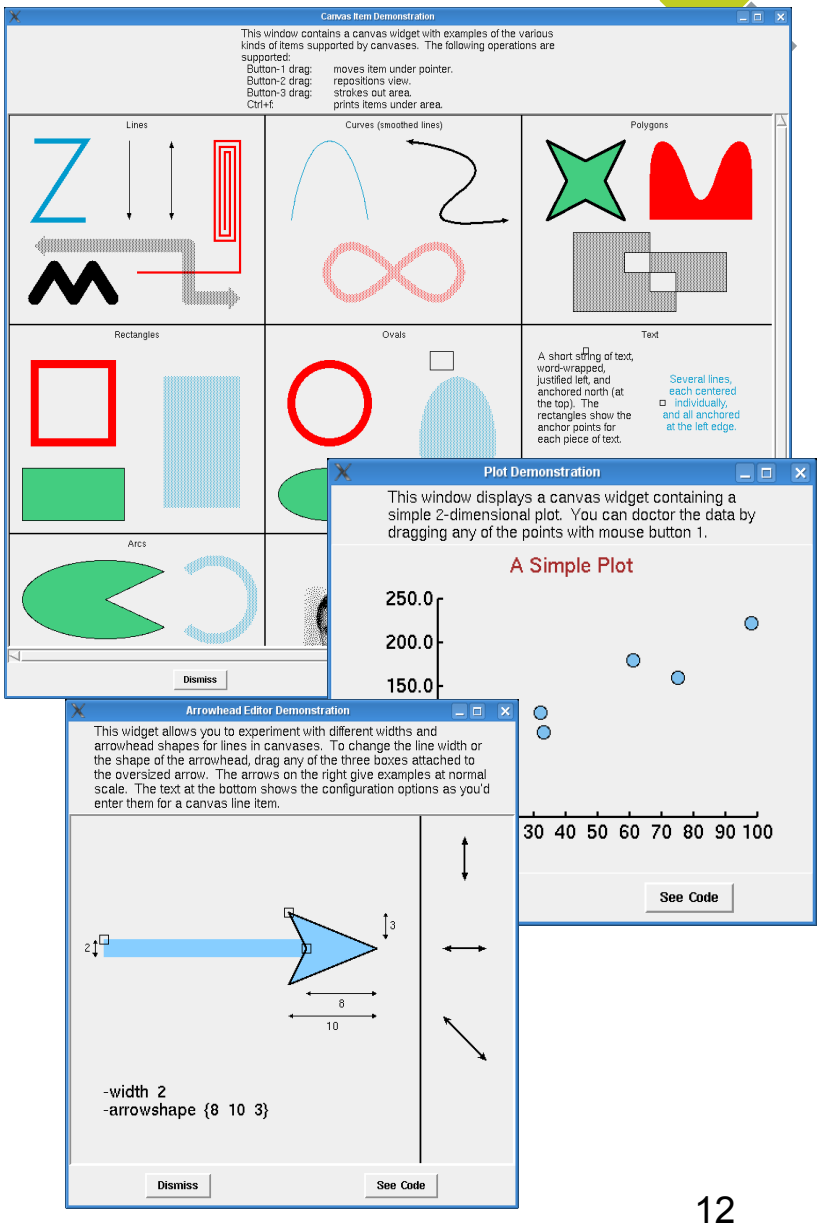


- Tcl/Tk
  - The mother of all canvases
- GnomeCanvas
  - Based partly on the Tcl/Tk canvas
- QCanvas
  - Based on QwSpriteField
- Silverlight (FKA “WPF/E”), Adobe Flash & Air
  - Rich internet applications
- HTML5 canvas
  - “Scriptable rendering”

Background

- Tcl/Tk Canvas
  - API: Tcl, Perl, Python ++
  - No item groups
  - Items receive events
  - Single buffering
  - No antialiasing / alpha
  - No transformations
  - Embedded widget support

  - Allows plugin-items written in C or C++ through extensions
  - “What if I just want to plot a pixel?”



Qt NetHack

OK

Game Gear Action Magic Info Help

You cannot drop something you are wearing.
You drop 11 uncursed apples.
You drop 11 uncursed carrots.
You drop 2 apples.
You drop a scroll labeled PRIRUTSENIE.
b – a +2 spear.
a – a +0 long sword.
i – 2 apples.
h – 11 uncursed carrots.
g – 11 uncursed apples.
j – a scroll labeled PRIRUTSENIE.

Warwick the Gallant
The Dungeons of Doom, level 1

STR:13	DEX:9	CON:11	INT:9	WIS:16	CHA:17
Au:0	HP:16/16	Pow:5/5	AC:3	Level:1	Exp:16

Lawful

Background: QCanvas

- Warwick Allison's QwSpriteField became QCanvas
  - QCanvas officially released with Qt 2.2 in 2000
  - *The* canvas for Qt applications
- The problem changed
  - While based on games, used in *all kinds* of applications
    - Traffic control, maps and charting, audio studio apps
    - ...ways that we had never thought about

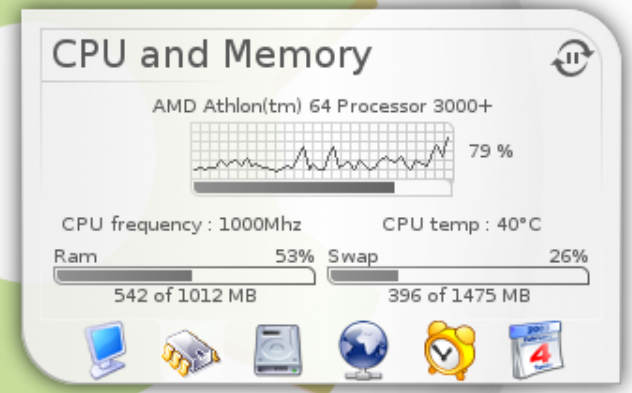
Background

- Qt 4: QPainter improved...

Background

- Qt 4: QPainter improved...
- August 2006: **Graphics View**
  - (feels so long ago!)

Configure Desktop



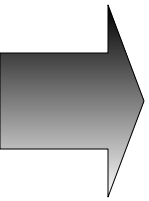
Background

- “Why do all modern toolkits need a canvas API?”

Background

- “Why do all modern toolkits need a canvas API?”
  - Answer: Because *you are an artist!*

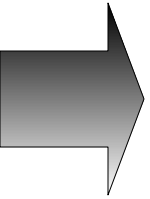
Agenda



- **Background**
  - **Why all modern toolkits need a powerful canvas**
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all fits together
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

TROLLTECH DEV DAYS '07

Agenda



- Background
  - Why all modern toolkits need a powerful canvas
- **Understanding the Graphics View framework**
  - **The Item, Scene and View**
  - **How it all fits together**
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

The Framework

- Canvas framework
  - Diagrams, graphs, vector graphics, board games
- Modular, compositional design
  - Easy to learn, and easy to master!
- High-level API for structured graphics

The Framework: Speed

- It's pretty *fast*:
  - Hit tests: logarithmic
    - 4 mill item scene: 20000 random lookups/sec
  - Collision detection: O(n log(n))
  - OpenGL
  - Tune & adjust (coming up!)

The Framework: Memory

- It's lean on memory usage (for being item-based):
  - QGraphicsItem: 84 bytes *(Qt3: 80 bytes, QObject: 104 bytes)*
  - QGraphicsRectItem: 156 bytes (pen, brush, rect)
  - QGraphicsSimpleTextItem: 333 bytes (text, font, cache)
- Including the overhead of item management:
  - 1000000 plain items: 80 MB of memory
  - 1000000 rects: 148 MB of memory

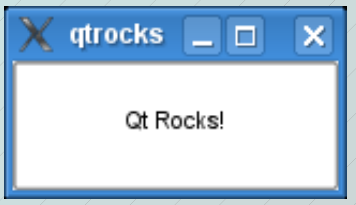
The Framework: API

- It's easy to use:
  - Genuine Qt 4 quality API
    - Suits beginners just as well as advanced users.
  - Works as a tool
    - Ready-made items
  - Works as a framework
    - Custom Item
    - Custom Scene
    - Custom View

TROLLTECH DEV DAYS '07

The Framework: Show me some code!

```
#include <QtGui>

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

    QGraphicsScene scene;
    scene.addText("Qt Rocks!");

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

    return app.exec();
}
```



TROLLTECH DEV DAYS '07

The Framework: Popularity

- It's very popular:
  - Customer Survey 2007
    - 30% use Graphics View
    - 46% use “Canvas Module”
  - QtCentre.org
    - 1 / 20 threads in “Qt Programming” forum
  - Qt Commercial Support
    - 1 / 50 emails

The Framework: Popularity

- What are you waiting for? ;-)

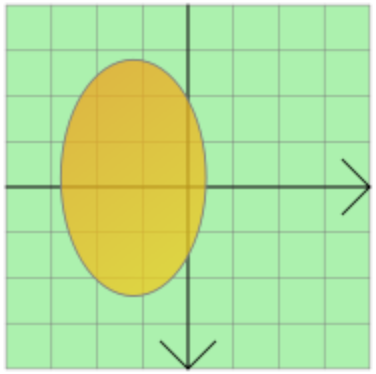
TROLLTECH DEV DAYS '07

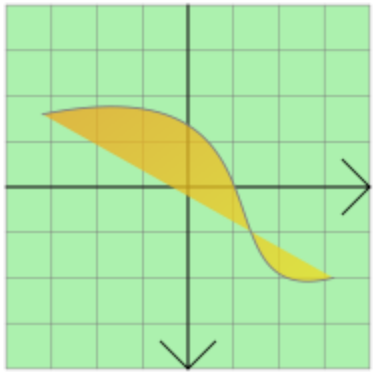
The Framework: Design

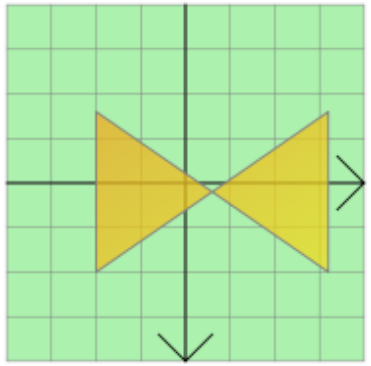
- Item
  - Basic element
- Scene
  - The “world”
- View
  - A viewport widget

The Framework: QGraphicsItem

- The basic building stone in Graphics View
  - Similar to QWidget; handles geometry, rendering, and interaction (keyboard and mouse)
  - No background of its own; by default “invisible”
  - Arbitrary shape: rectangular/elliptic/curved/non-contiguous
  - Simple, light-weight and flexible

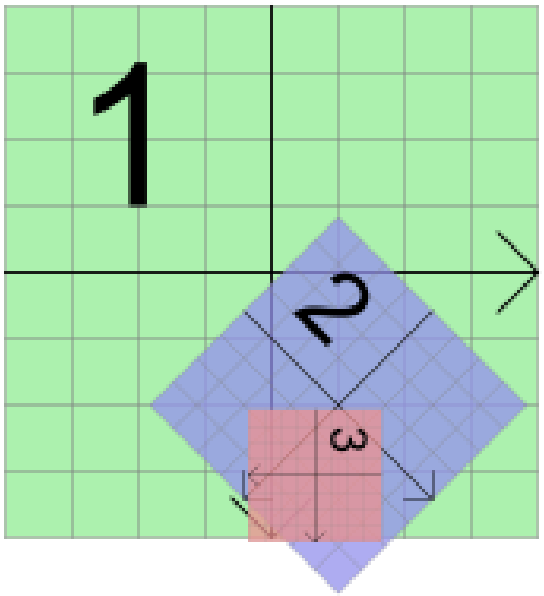






The Framework: QGraphicsItem

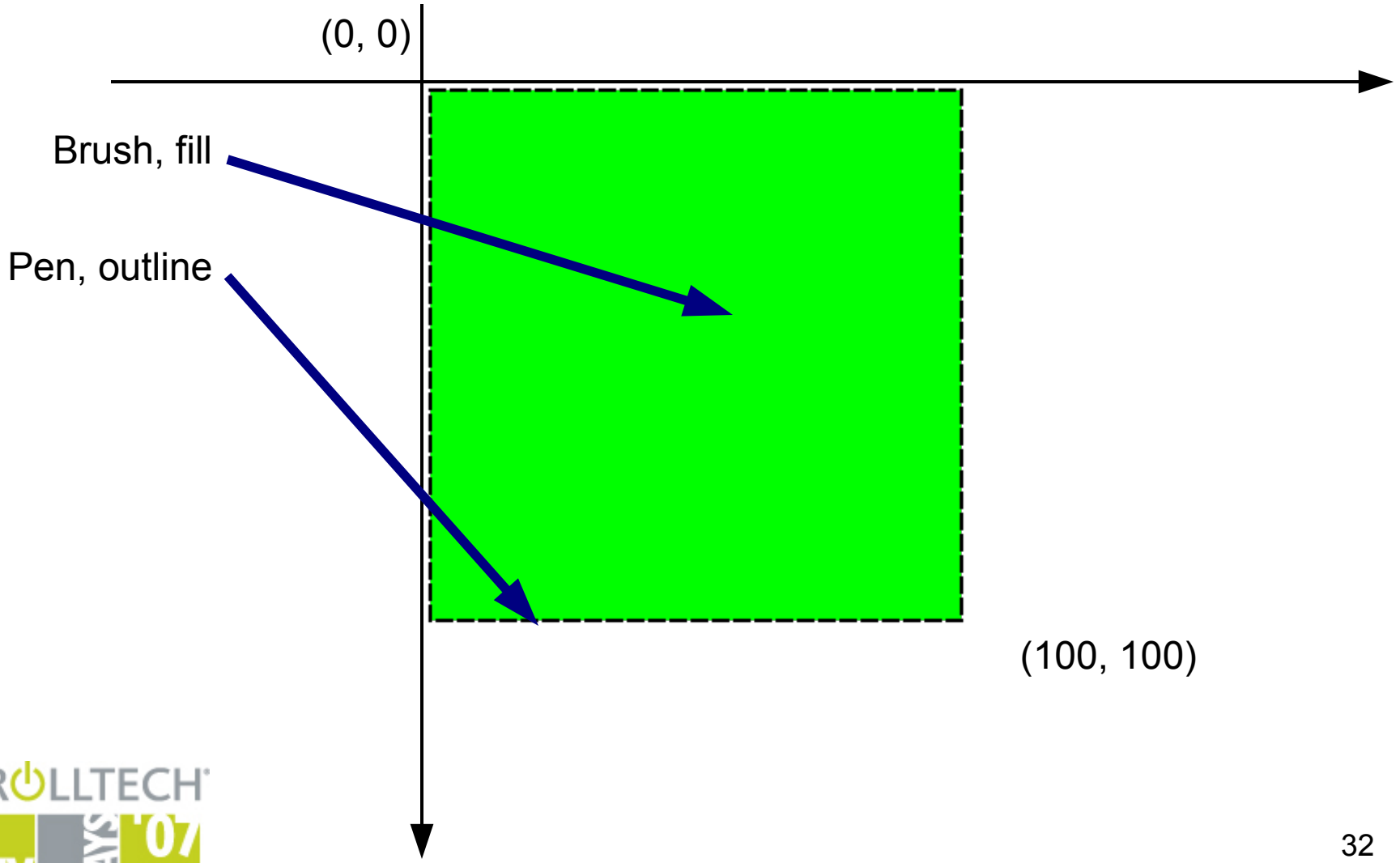
- All coordinates are item-local
- Its position is given relative to its parent
- Items can be stacked
  - Parent-child or sibling-sibling
  - Ideal for composite / “complex” items
- Items can be transformed
  - Rotate/scale/shear/translate
  - Transformations propagate
- Several basic items are provided
  - Rectangles, ellipses, polygons, text



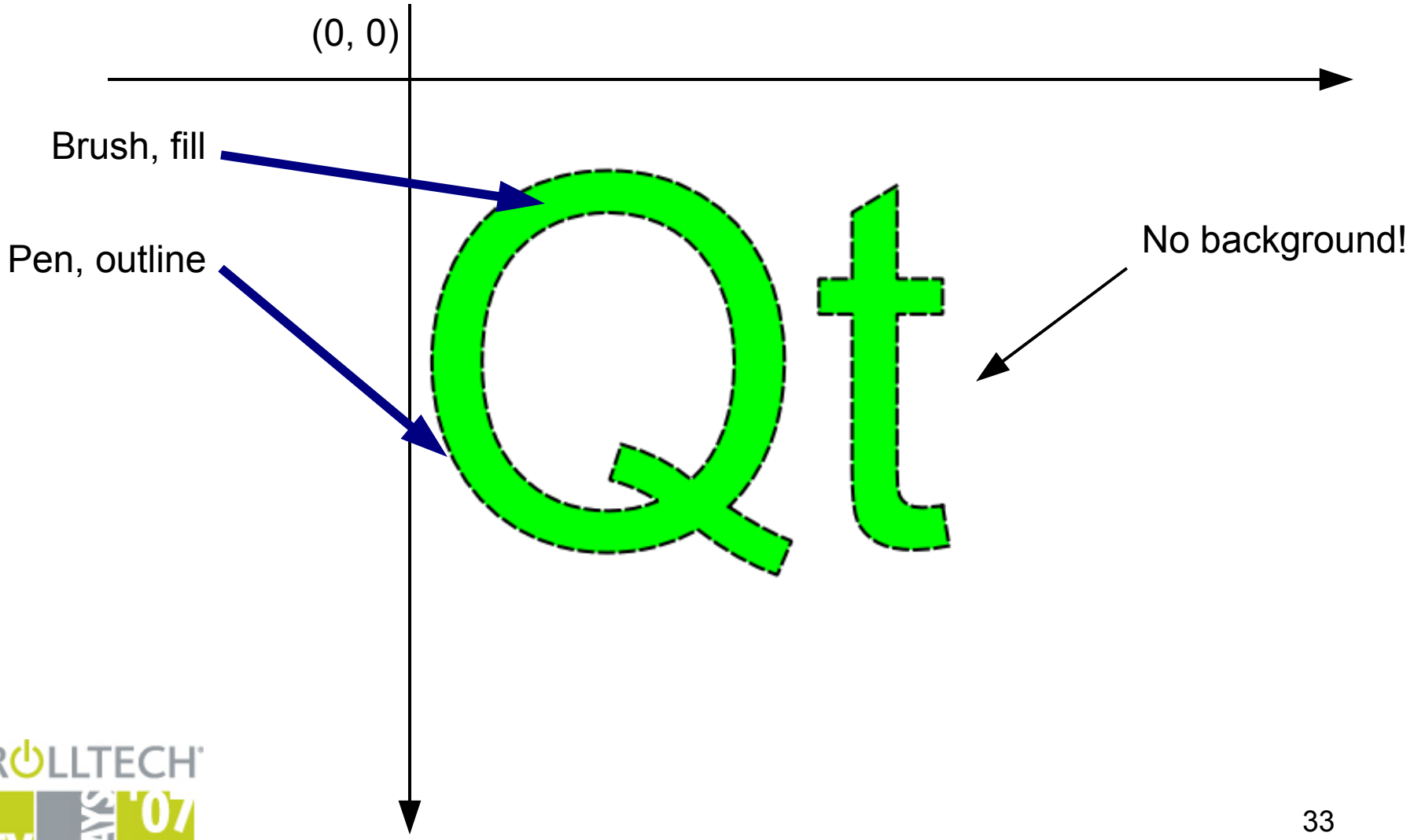
TROLLTECH DEV DAYS '07

Demo!

The Framework: QGraphicsItem

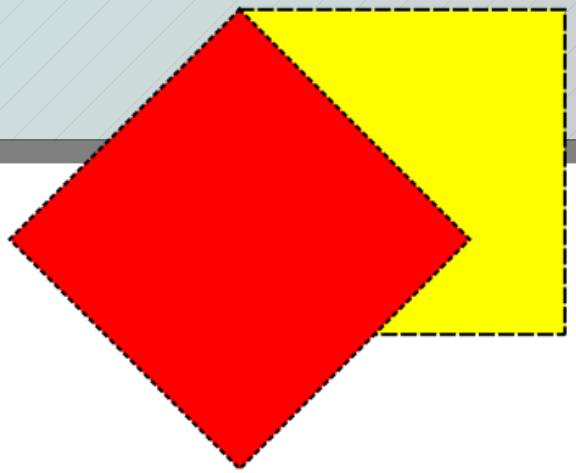


The Framework: QGraphicsItem



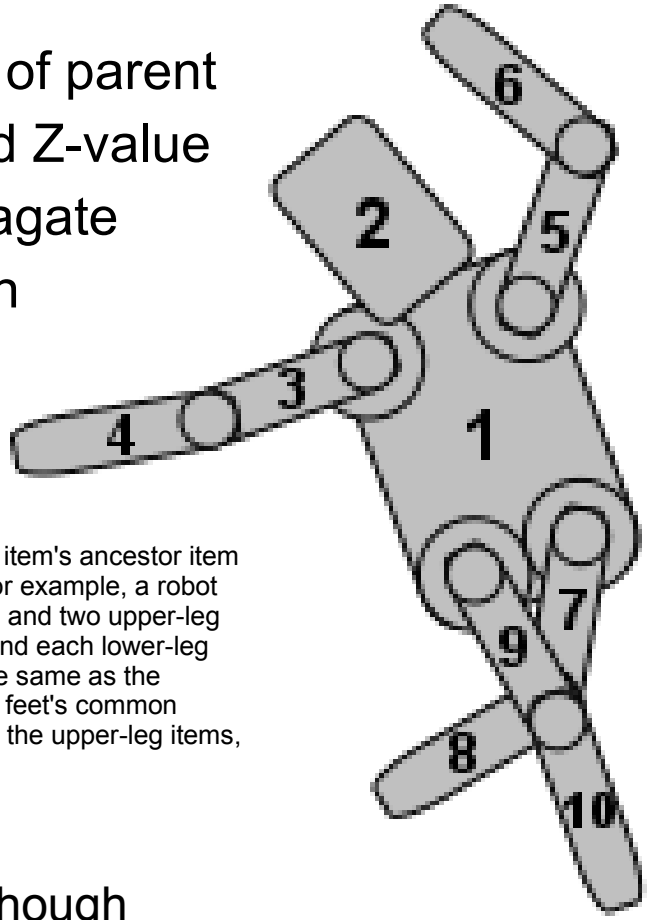
The Framework: QGraphicsItem

```
QGraphicsRectItem *rect = new QGraphicsRectItem;
rect->setRect(0, 0, 200, 200);
rect->setPen(QPen(Qt::black, 2, Qt::DashLine));
rect->setBrush(QBrush(Qt::yellow));
rect->setPos(10, 0);

QGraphicsRectItem *rect2 = new QGraphicsRectItem(rect);
rect2->setRect(0, 0, 200, 200);
rect2->setPen(QPen(Qt::black, 2, Qt::DotLine));
rect2->setBrush(QBrush(Qt::red));
rect2->rotate(45);
```



The Framework: QGraphicsItem

- Composition:
  - Children are always stacked on top of parent
  - Siblings are ordered by creation and Z-value
  - Positions and transformations propagate
  - Allows for complex item composition
  - Parents do not *clip* children[*]

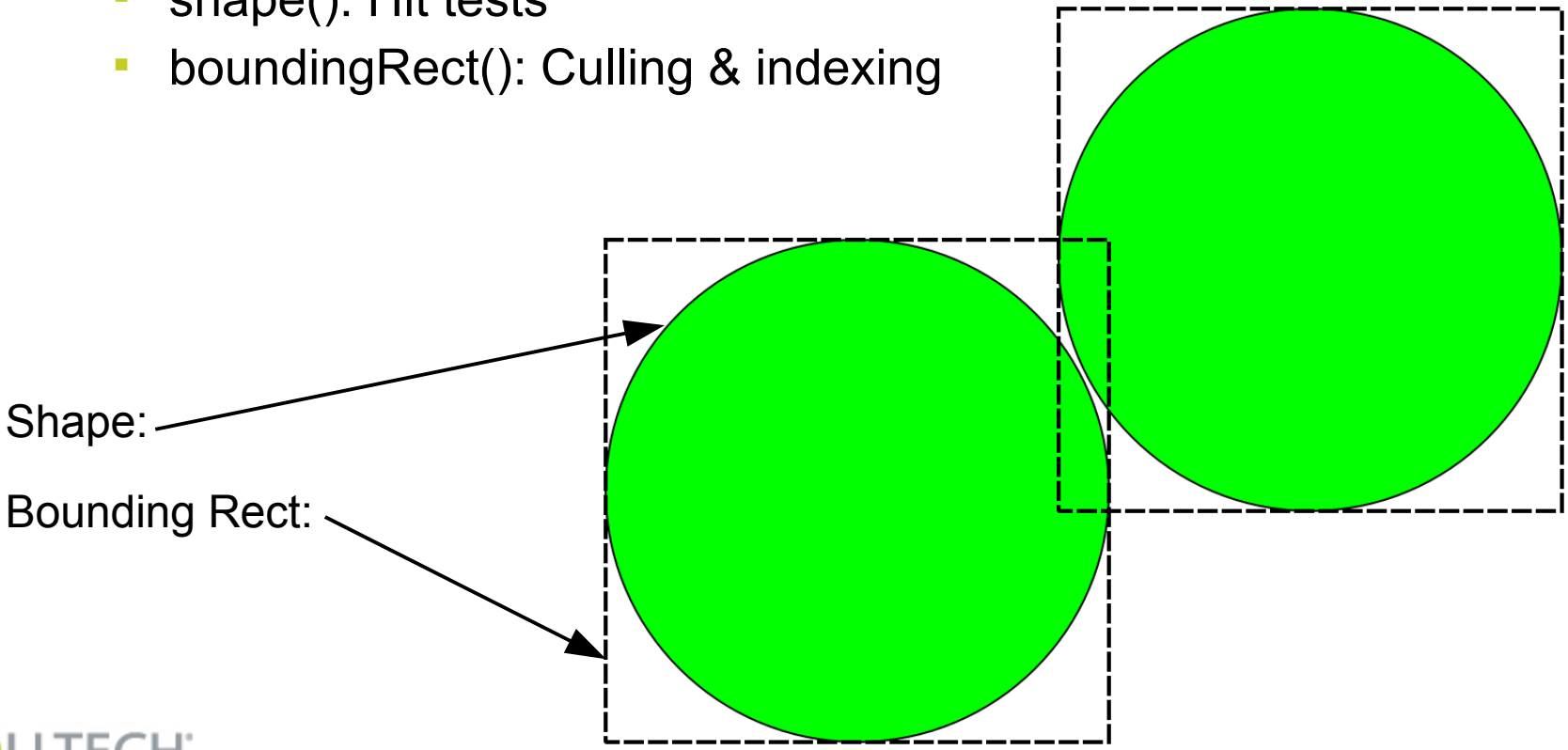


"Children of different parents are stacked according to the Z-value of each item's ancestor item which is an immediate child of the two items' closest common ancestor. For example, a robot item might define a torso item as the parent of a head item, two arm items, and two upper-leg items. The upper-leg items would each be parents of one lower-leg item, and each lower-leg item would be parents of one foot item. The stacking order of the feet is the same as the stacking order of each foot's ancestor that is an immediate child of the two feet's common ancestor (i.e., the torso item); so the feet are stacked in the same order as the upper-leg items, regardless of each foot's Z-value."

[*] can be enabled with flags though

The Framework: QGraphicsItem

- Geometry: Item Shape & Bounding Rectangle
  - shape(): Hit tests
  - boundingRect(): Culling & indexing



The Framework: QGraphicsItem

- Features 1/2:
  - Mouse & wheel events (press, release, click, doubleclick)
  - Hover events (“mouse tracking”)
  - Focus & keyboard input
  - High-resolution, *fast* collision detection (ideal for games)
  - Drag & drop to and from regular widgets, and other items

The Framework: QGraphicsItem

- Features 2/2:
  - Obscurity testing / opaque area management
  - Cursors, Tooltips, Custom Data (“dynamic string properties”)
  - Visibility checking & “ensure visible” functionality
  - Some basic standard interaction (e.g., selection & moving)
  - *Consistent use of Local Coordinates makes it easy to implement your own item*

The Framework: QGraphicsItem

```
class HoverItem : public QGraphicsItem
{
public:
    HoverItem(QGraphicsItem *parent = 0)
        : QGraphicsItem(parent)
    {
        setAcceptsHoverEvents(true);
    }

    QRectF boundingRect() const;

    void paint(QPainter *painter,
               const QStyleOptionGraphicsItem *option,
               QWidget *viewport);
};
```

The Framework: QGraphicsItem

```
QRectF HoverItem::boundingRect() const
{
    QRectF rect(0, 0, 50, 50);
    return rect.adjusted(-0.5, -0.5, 0.5, 0.5);
}

void HoverItem::paint(QPainter *painter,
                const QStyleOptionGraphicsItem *option,
                QWidget *viewport)
{
    painter->setPen(QPen(Qt::black, 1));
    if (option->state & QStyle::State_MouseOver)
        painter->setBrush(QColor(Qt::white));
    else
        painter->setBrush(QColor(Qt::green));
    painter->drawRoundRect(0, 0, 50, 50);
}
```

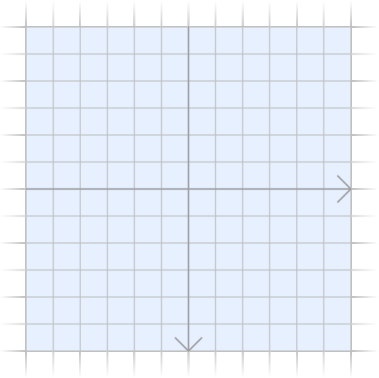
The Framework: QGraphicsItem

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

    QGraphicsScene scene;
    for (int i = 0; i < 100000; ++i) {
        HoverItem *item = new HoverItem;
        item->setPos(-5000 + qrand() % 1000,
                     -5000 + qrand() % 1000);
        scene.addItem(item);
    }

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

    return app.exec();
}
```


Demo!

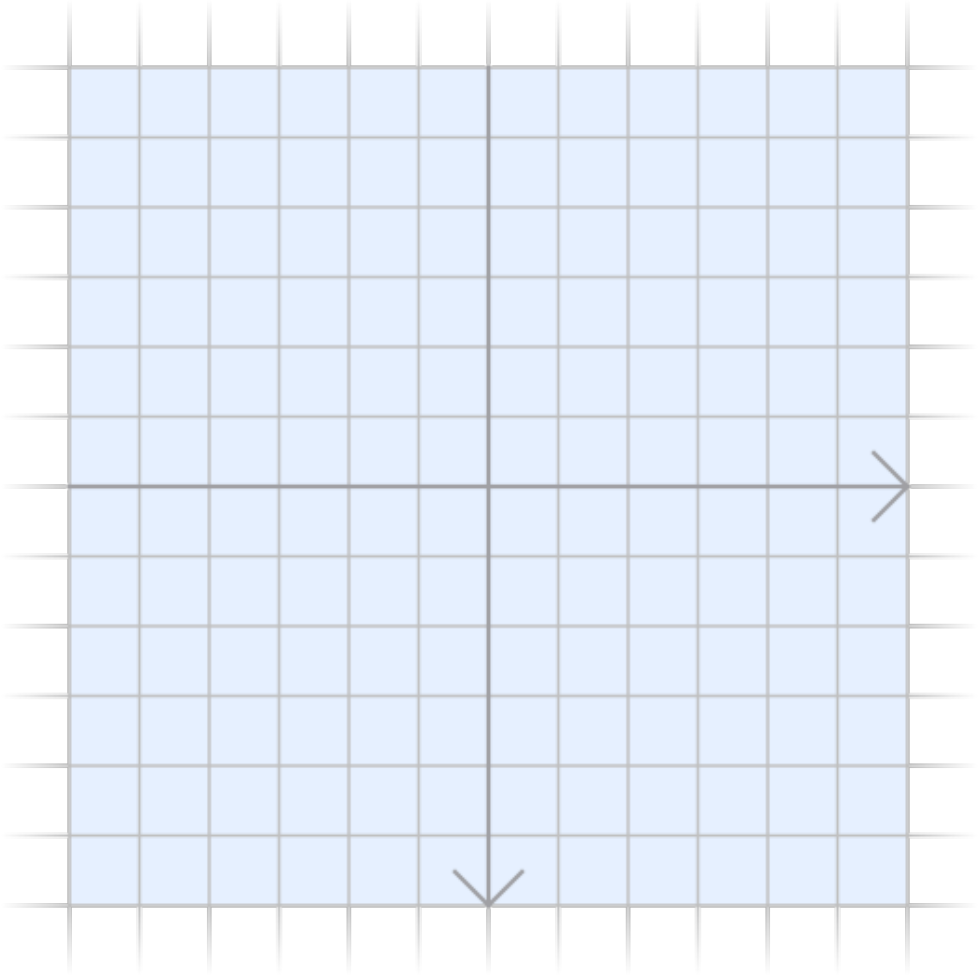
The Framework: QGraphicsScene

- The canvas / “world” object / scene graph
  - Manages positions and transformations for all items
  - Can easily store millions of items (or just a few)
  - Provides instant item location using space partitioning
  - Tracks and notifies the view of changes to items
  - Does most of the “smart stuff” in Graphics View
    - Indexing
    - Selection management
    - Focus management
    - Event translation, delivery and propagation
    - Collision detection



TROLLTECH DEV DAYS '07

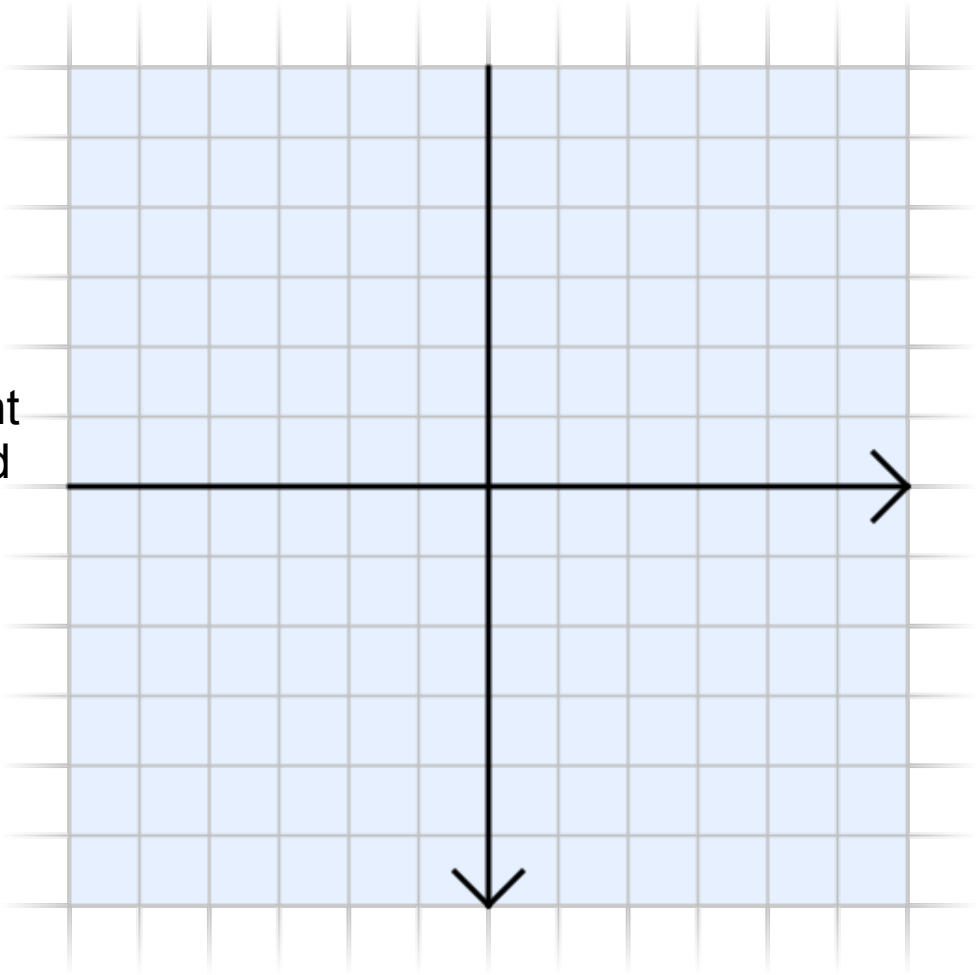
The Framework: QGraphicsScene



The Framework: QGraphicsScene

- Fundamental properties
  - Real 2D Cartesian coordinate system
  - Three layers
    - BackgroundLayer (brush or custom)
    - ItemLayer
    - ForegroundLayer (brush or custom)
  - A scene rect (bounds)
  - *No visual appearance of its own*

The Framework: QGraphicsScene

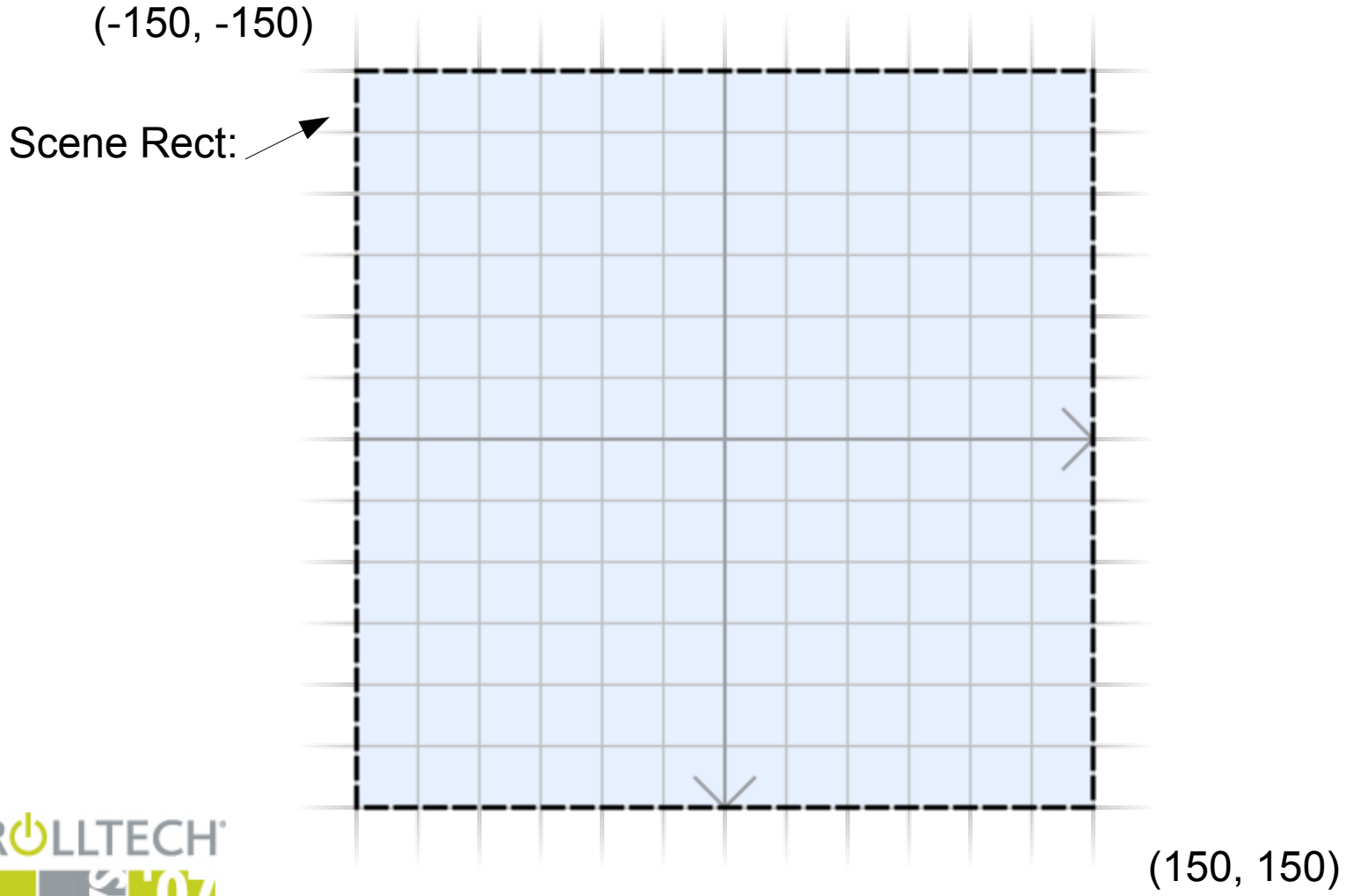


Note:

X grows to the right
Y grows downward

TROLLTECH DEV DAYS '07

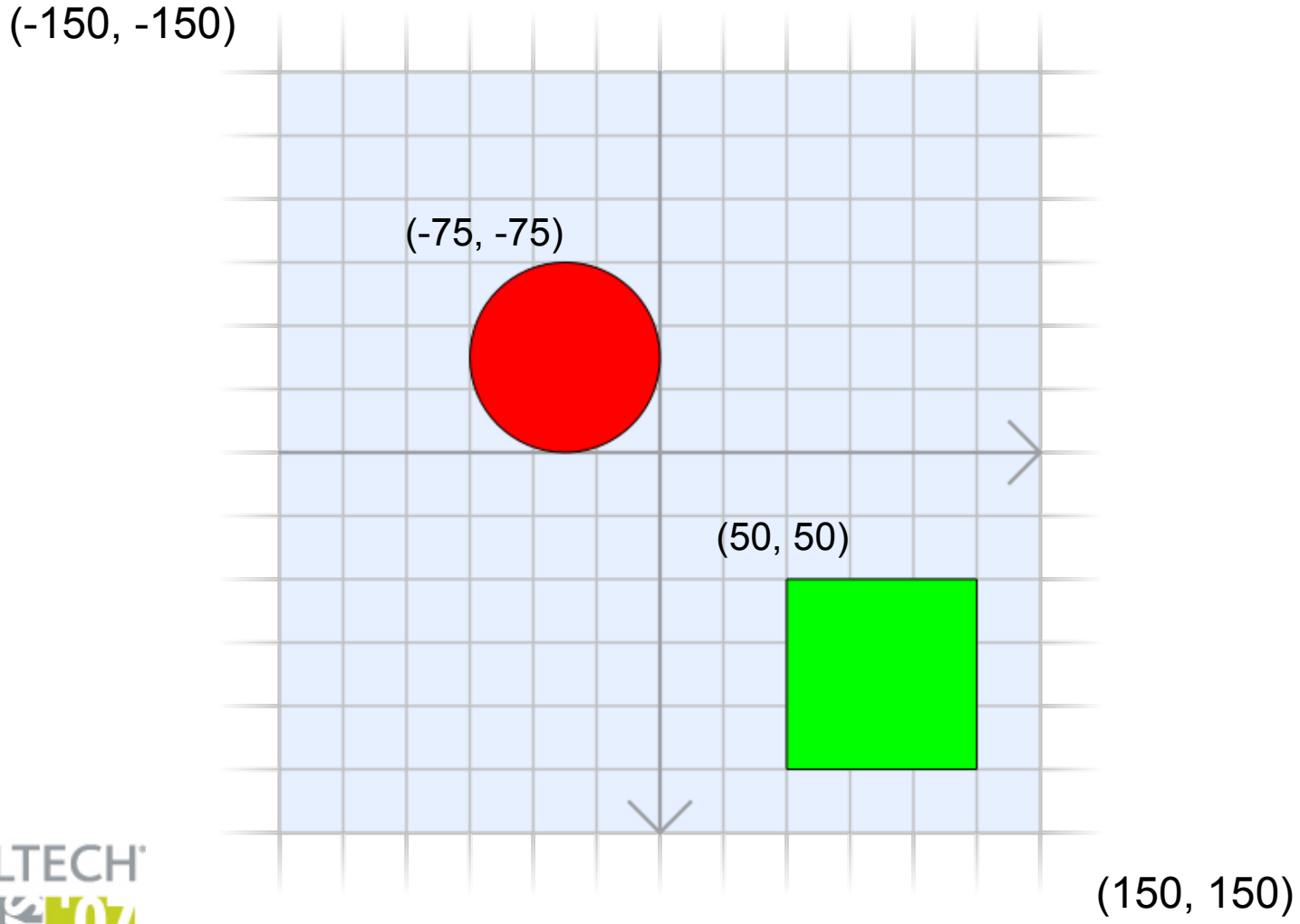
The Framework: QGraphicsScene



The Framework: QGraphicsScene

```
#include <QtGui>

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

    QGraphicsScene scene(-150, -150, 300, 300);
    QGraphicsItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    ellipse->setBrush(Qt::red);
    rect->setPos(50, 50);
    rect->setBrush(Qt::green);

    return app.exec();
}
```

The Framework: QGraphicsScene



The Framework: QGraphicsScene

- Hit tests / culling / collision detection
  - Detect collision with *any vector path*
  - 1) Index lookup
    - Generates what may or may not intersect the shape.
  - 2) Bounding rectangle intersection
    - Roughly eliminates items that cannot possibly collide.
  - 3) Shape intersection
    - QPainterPath::intersects() - O(n log(n)) + cutoffs

The Framework: QGraphicsScene

```
#include <QtGui>

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

    QGraphicsScene scene(-150, -150, 300, 300);
    QGraphicsEllipseItem *ellipse
        = scene.addEllipse(0,0,75,75);
    QGraphicsRectItem *rect
        = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    ellipse->setBrush(Qt::red);
    rect->setPos(50, 50);
    rect->setBrush(Qt::green);
    ...
```

The Framework: QGraphicsScene

```
    ...
    QImage image(1600, 1200,
                 QImage::Format_ARGB32_Premultiplied);
    image.fill(0);

    QPainter painter(&image);
    scene.render(&painter);
    painter.end();

    image.save("scene.png");
    return 0;
}
```

Resolution independent:
You choose the size!

TROLLTECH DEV DAYS '07

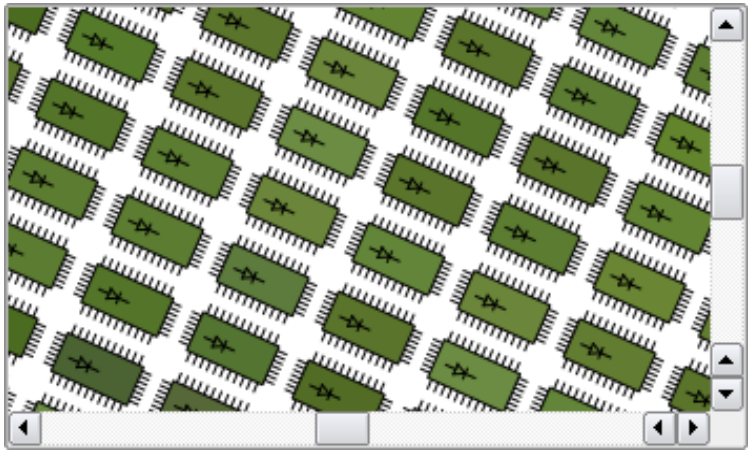
The Framework: QGraphicsScene

```
    ...
    QSvgGenerator svg;
    svg.setFileName("scene.svg");

    QPainter painter(&svg);
    scene.render(&painter);

    return 0;
}
```

Output: SVG file

The Framework: QGraphicsView

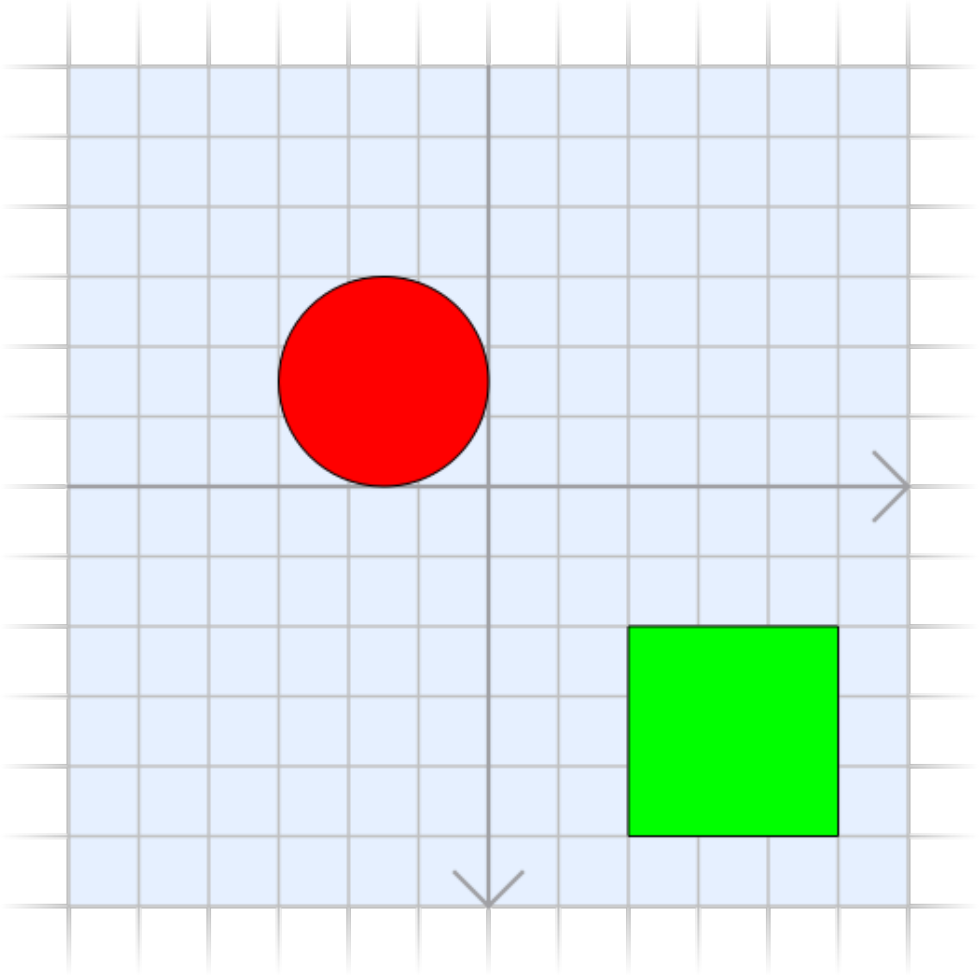
- The widget / viewport / GUI
  - Renders a section of the scene into a widget
  - Provides scrollbars for navigating the scene
  - Supports zooming, rotation, and projection in “3D”
  - Allows scene interaction using mouse & keyboard
  - Provides **level-of-detail** information to help simplify graphics when zoomed out



TROLLTECH DEV DAYS '07

Demo!

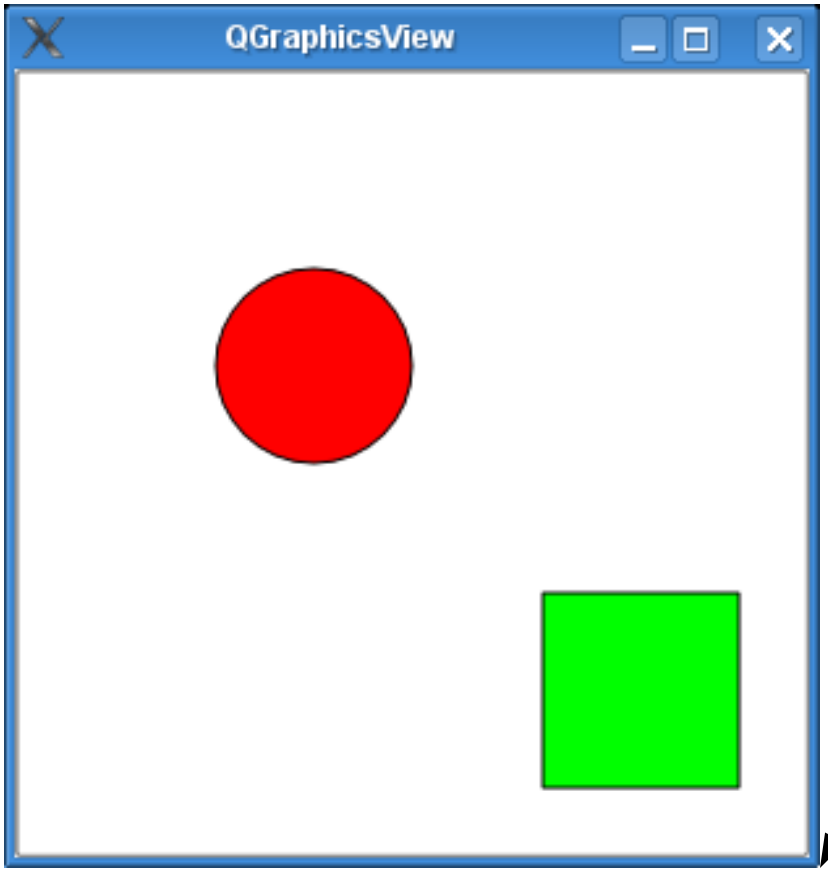
The Framework: QGraphicsView



The Framework: QGraphicsView

```
#include <QtGui>

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

    QGraphicsScene scene(-150, -150, 300, 300);
    QGraphicsEllipseItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsRectItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    ellipse->setBrush(Qt::red);
    rect->setPos(50, 50);
    rect->setBrush(Qt::green);

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

    return app.exec();
}
```

The Framework: QGraphicsView



Size of widget matches size of scene.

Demo!

The Framework: QGraphicsView

- Features 1/2:
  - Hardware accelerated rendering using QGLWidget
  - Scroll-hand dragging
  - Rubberband selection
  - Dirty region handling
  - Configurable viewport update mode
    - (More on this later)

The Framework: QGraphicsView

- Features 2/2:
  - Transformations
    - Zoom / rotate X, Y and Z / scale
    - Stretching scene contents to fit inside the view
  - Maps coordinates between the viewport and the scene
    - Can also map rectangles, ellipses and other shapes
      - view rect -> scene polygon
      - scene polygon -> view polygon

The Framework: QGraphicsView

TROLLTECH DEV DAYS '07

Demo!

The Framework: QGraphicsView

```
#include <QtOpenGL>

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

    QGraphicsScene scene(-500, -500, 1000, 1000);
    QGraphicsItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    rect->setPos(50, 50);

    QGraphicsView view(&scene);
    view.setViewport(new QGLWidget);
    view.show();

    return app.exec();
}
```

Rendering onto an OpenGL viewport.

The Framework: QGraphicsView

```
#include <QtGui>

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

    QGraphicsScene scene(-500, -500, 1000, 1000);
    QGraphicsItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    rect->setPos(50, 50);

    QGraphicsView view(&scene);
    view.setDragMode(QGraphicsView::ScrollHandDrag);
    view.show();

    return app.exec();
}
```

1 line to enable scroll-hand dragging.

The Framework: QGraphicsView

```
#include <QtGui>

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

    QGraphicsScene scene(-500, -500, 1000, 1000);
    QGraphicsItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    rect->setPos(50, 50);

    QGraphicsView view(&scene);
    view.setDragMode(QGraphicsView::RubberBandDrag);
    view.show();

    return app.exec();
}
```

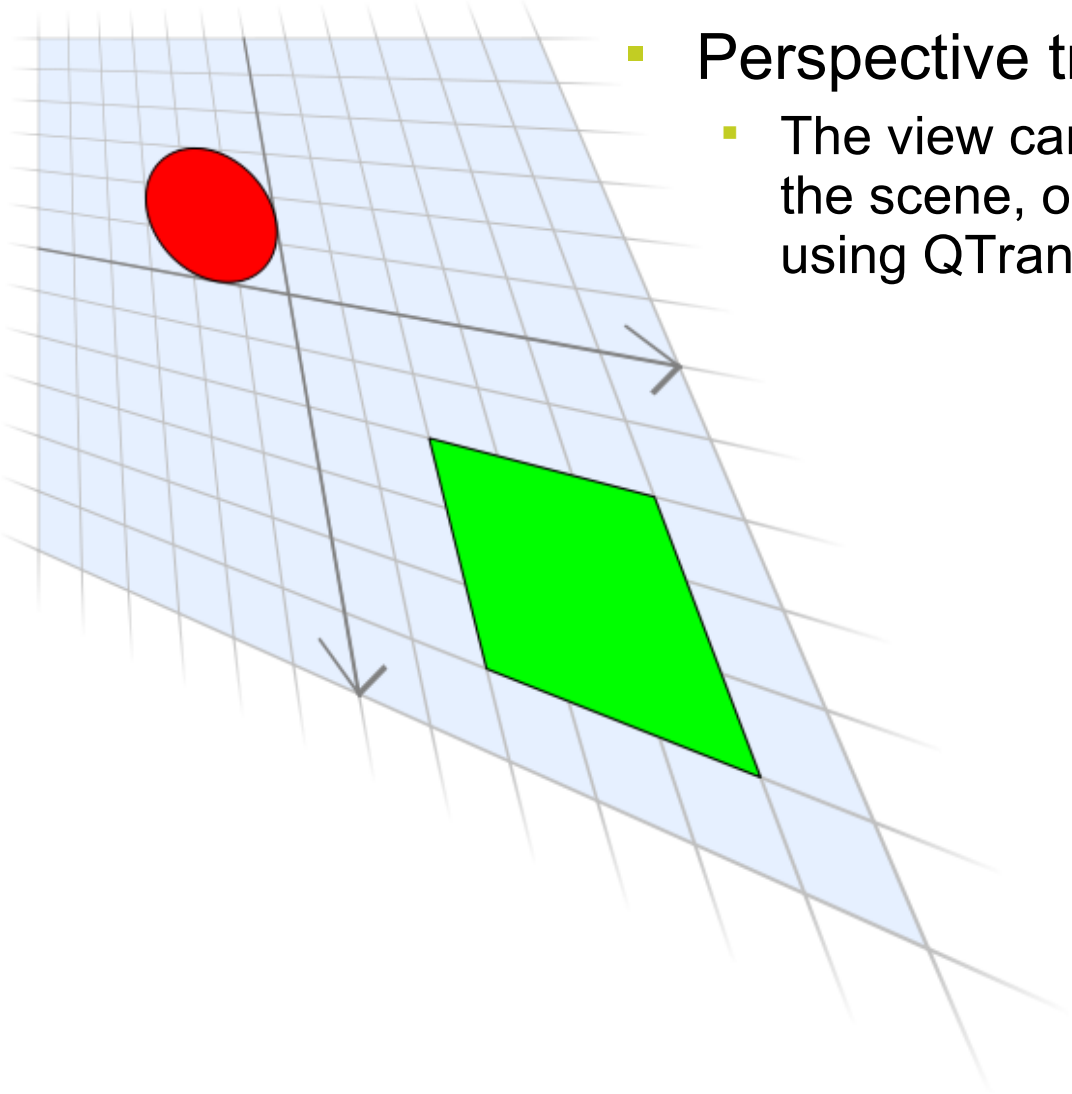
1 line to enable rubber-band selection

The Framework: QGraphicsView

```
#include <QtGui>

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

    QGraphicsScene scene(-500, -500, 1000, 1000);
    QGraphicsItem *ellipse = scene.addEllipse(0,0,75,75);
    QGraphicsItem *rect = scene.addRect(0,0,75,75);

    ellipse->setPos(-75, -75);
    rect->setPos(50, 50);

    QGraphicsView view(&scene);
    view.setTransform(QTransform().rotate(45, Qt::XAxis));
    view.show();

    return app.exec();
}
```

1 line to rotate in 3D

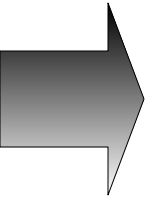
The Framework: QGraphicsView



- Perspective transform
  - The view can transform the scene, or any item, using QTransform (4.3)

Demo!

Agenda



- Background
  - Why all modern toolkits need a powerful canvas
- **Understanding the Graphics View framework**
  - **The Item, Scene and View**
  - **How it all fits together**
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

Agenda

- Background
  - Why all modern toolkits need a powerful canvas
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all works
- **Let's write some code!**
  - **Writing your own Graphics Item**
  - **Zooming, scrolling, taking screenshots, and printing**
  - **Implementing support for Drag & Drop**
  - **Optimization tricks / how to make it FAST**
  - **Widgets On The Canvas**

TROLLTECH DEV DAYS '07

Let's write some code!

- Writing a custom item
- Implementing scroll-wheel zooming & printing
- Dragging & dropping items
- Optimization tricks
- Widgets on the Canvas

TROLLTECH DEV DAYS '07

Examples: Writing a custom item

- Define a size and unit for your item
  - “Natural size”: Use pixels
  - “Logical size”: Use unity coordinates (-1, +1)
- Decide on what shape your item should have
- Start implementing the base functions
  - boundingRect()
  - paint() (typically you start with just a rectangle)
- Extend
  - Spend some time to improve the drawing code
  - Add mouse / hover effects as you go

Examples: Writing a custom item

Examples: Writing a custom item

```
class Item : public QGraphicsItem
{
public:
    Item(QGraphicsItem *parent = 0)
        : QGraphicsItem(parent)
    { }

    QRectF boundingRect() const
    {
        return QRectF(0, 0, 100, 100);
    }

    void paint(QPainter *painter,
               const QStyleOptionGraphicsItem *option,
               QWidget *viewport)
    {
        painter->drawEllipse(0, 0, 100, 100);
    }
};
```

Examples: Writing a custom item

```
#include <QtGui>

class Item
...

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

    QGraphicsScene scene;
    scene.addItem(new Item);

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

    return app.exec();
}
```

TROLLTECH DEV DAYS '07

Demo!

Examples: Writing a custom item

- Handle mouse events:
  - mousePressEvent()
  - mouseMoveEvent()
  - mouseReleaseEvent()
  - mouseDoubleClickEvent()
  - wheelEvent()
  - contextMenuEvent()
- Handle hover events:
  - hoverEnterEvent()
  - hoverMoveEvent()
  - hoverLeaveEvent()

TROLLTECH DEV DAYS '07

Examples: Writing a custom item

- QGraphicsSceneMouseEvent
  - Events in local, scene or screen (desktop) coordinates:
    - pos()
    - scenePos()
    - screenPos()
  - The last position is provided as part of the event
    - lastPos(), lastScenePos(), lastScreenPos()
  - You can also find where the buttons were pressed:
    - buttonDownPos(Qt::MouseButton), ...ScenePos(), ...ScreenPos()
  - But: Beware when items move!

Examples: Writing a custom item

```
class Item : public QGraphicsItem
{
    ...
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event)
    {
        if (event->pos().x() > 0)
            event->ignore(); // propagate

        qDebug() << “Press at local pos” << event->pos();
        qDebug() << “  ....at scene pos” << event->scenePos();
        qDebug() << “  ....at screen pos” << event->screenPos()
    }
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event)
    {
        qDebug() << “Move at local pos” << event->pos();
        qDebug() << “  distance from last pos:”
                 << QLineF(event->lastPos(), event->pos()).length();
    }
};
```

Examples: Writing a custom item

- New mouse event propagation scheme
  - Events are ignored by default (items are “transparent”)
  - If you reimplement mousePressEvent(), the event will be accepted by default
  - Ignoring the event will pass it on to the item underneath
  - If an item accepts the first mouse press event, it will become the *mouse grabber*, and receives all subsequent mouse events.
  - Convenience: setAcceptedMouseButtons(Qt::MouseButtons)

Examples: Writing a custom item

```
class Item : public QGraphicsItem
{
    ...
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event)
    {
        if (!down) {
            down = true;
            update();
        }
    }
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
    {
        if (event->buttons() == 0) {
            down = false;
            update();
        }
    }
};
```

Examples: Writing a custom item

```
class Item : public QGraphicsItem
{
    ...
    void paint(QPainter *painter, ...
    {
        painter->setBrush(QBrush(down ? Qt::blue : Qt::green));
        painter->drawEllipse(0, 0, 100, 100);
    }
```

TROLLTECH DEV DAYS '07

Demo!

Examples: Writing a custom item

```
class Item : public QGraphicsItem
{
    ...
    QPainterPath shape() const
    {
        QPainterPath path;
        path.addEllipse(0, 0, 100, 100);
        return path;
    }
```

TROLLTECH DEV DAYS '07

Demo!

Examples: Writing a custom item

- When it comes to Computer Graphics....

Examples: Writing a custom item

- When it comes to Computer Graphics....
  - “Never underestimate the power of trial & error!”

Examples: Zoom with the scroll wheel

```
class ZoomView : public QGraphicsView
{
public:
    ZoomView(QGraphicsScene *scene, QWidget *parent = 0)
        : QGraphicsView(scene, parent) { }

protected:
    void keyPressEvent(QKeyEvent *event)
    {
        if (event->key() == Qt::Key_Plus) scale(1.2, 1.2);
    }
    void keyReleaseEvent(QKeyEvent *event)
    {
        if (event->key() == Qt::Key_Minus) scale(1/1.2, 1/1.2);
    }
    void wheelEvent(QWheelEvent *event)
    {
        qreal scaleFactor = ::pow(2, -event->delta() / (2 * 120.0));
        scale(scaleFactor, scaleFactor);
    }
};
```

Examples: Printing

- The scene can print, and the view can print
- You can render the whole scene, or parts of the scene
- You can also render with a transformation
  - (Which is what QGraphicsView does, really)

Examples: Printing

```
class ZoomView : public QGraphicsView
{
public:
    ZoomView(QGraphicsScene *scene, QWidget *parent = 0)
        : QGraphicsView(scene, parent) { }

protected:
    void printViewport()
    {
        QPrinter printer;
        if (QPrintDialog(&printer, this).exec()) {
            QPainter painter(&printer);
            painter.setRenderHint(QPainter::Antialiasing);
            render(&painter);
        }
    }
};
```

TROLLTECH DEV DAYS '07

Examples: Drag & drop

- Graphics View uses an improved DnD model
  - All DnD classes compressed into one
  - Just as powerful as QWidget's DnD
- The scene translates view events into individual drag & drop events for items that accept drops
  - You can handle drag & drop for the view
  - You can handle drag & drop for the scene
  - You can handle drag & drop for individual items

Demo!

Examples: Drag & drop

```
class Scene : public QGraphicsScene
{
    ...
protected:
    void dragEnterEvent(QGraphicsSceneDragDropEvent *event);
    void dragMoveEvent(QGraphicsSceneDragDropEvent *event);
    void dragLeaveEvent(QGraphicsSceneDragDropEvent *event);
    void dropEvent(QGraphicsSceneDragDropEvent *event);
private:
    QGraphicsItem *dragItem;
};
```

Examples: Drag & drop

```
void Scene::dragEnterEvent(QGraphicsSceneDragDropEvent *event)
{
    event->acceptProposedAction();
}

void Scene::dragLeaveEvent(QGraphicsSceneDragDropEvent *event)
{
    if (dragItem) {
        delete dragItem;
        dragItem = 0;
    }
}
```

Examples: Drag & drop

```
void Scene::dragMoveEvent(QGraphicsSceneDragDropEvent *event)
{
    if (!event->mimeData()->hasFormat("image/x-item"))
        return;
    if (!dragItem) {
        dragItem = new Item;
        dragItem->setZValue(1);
        addItem(dragItem);
    }
    dragItem->setPos(event->scenePos());
}

void Scene::dropEvent(QGraphicsSceneDragDropEvent *event)
{
    if (dragItem) {
        dragItem->setZValue(0);
        dragItem = 0;
    }
}
```

Examples: Drag & drop

```
class Item : public QGraphicsItem
{
    ...
protected:
    void dragEnterEvent(QGraphicsSceneDragDropEvent *event);
    void dragLeaveEvent(QGraphicsSceneDragDropEvent *event);
    void dragMoveEvent(QGraphicsSceneDragDropEvent *event);
    void dropEvent(QGraphicsSceneDragDropEvent *event);
```

- Item DnD works the same way as for the scene
  - Propagation & handling is the same as QWidget

Examples: Drag & drop

```
void Item::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    QDrag *drag = new QDrag(event->widget());
    QMimeData *mime = new QMimeData;

    QImage image(":/images/head.png");
    mime->setImageData(image);

    drag->setMimeData(mime);
    drag->exec();
}
```

- Starting a drag
  - Just like QWidget
  - See also the “Drag and Drop Robot” example

TROLLTECH DEV DAYS '07

Demo!

Examples: Optimization Tricks

- Speed is of the essence

Examples: Optimization Tricks

- Speed
  - Choosing the right update mode
  - Rendering as fast as possible
  - Fast Culling & Hit-tests & Collision Detection
  - Caching

Examples: Optimization Tricks

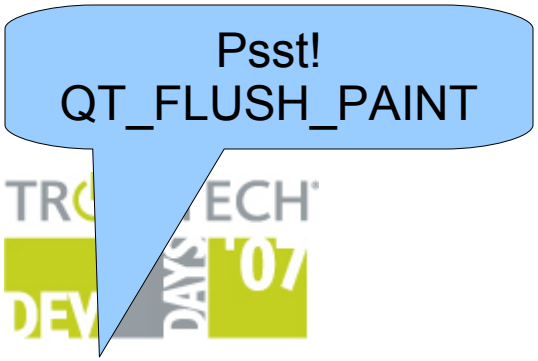
- General: This is Graphics :-)
  - If it's slow, use a good profiling tool to find out why:

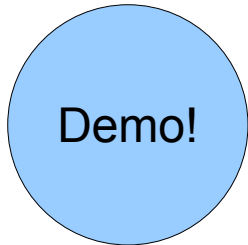
    valgrind –tool=callgrind && kcachegrind

  - However: Profilers can only tell you so much...

Examples: Optimization Tricks

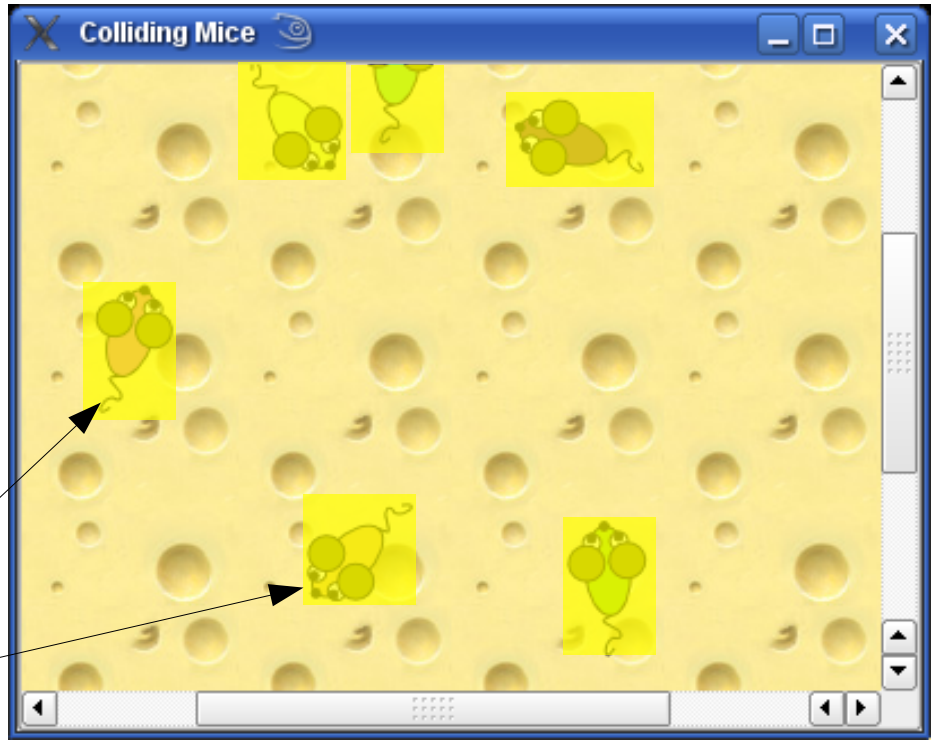
- Choosing the right update mode
  - QGraphicsView::viewportUpdateMode (4.3)
  - **MinimalViewportUpdate**
  - **SmartViewportUpdate**
  - **FullViewportUpdate**
  - **NoViewportUpdate**
  - **BoundingRectViewportUpdate (4.4)**





Examples: Optimization Tricks

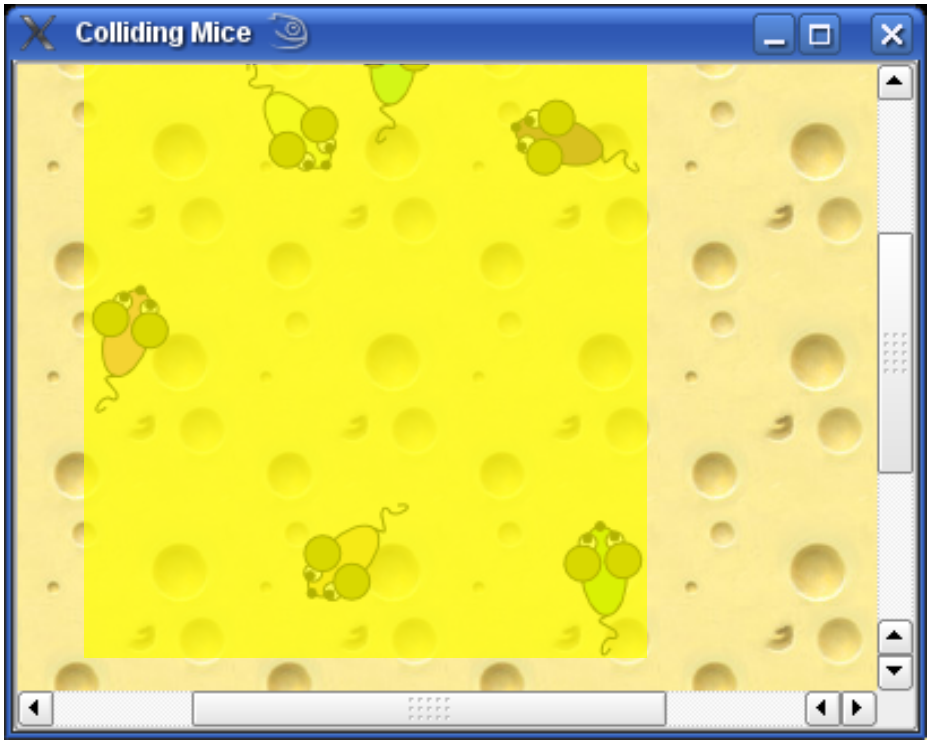
- MinimalViewportUpdate (Default)
  - Minimal rendering
  - Maximum culling



Yellow fields
are redrawn!

Examples: Optimization Tricks

- SmartViewportUpdate
  - Variable rendering
  - Variable culling



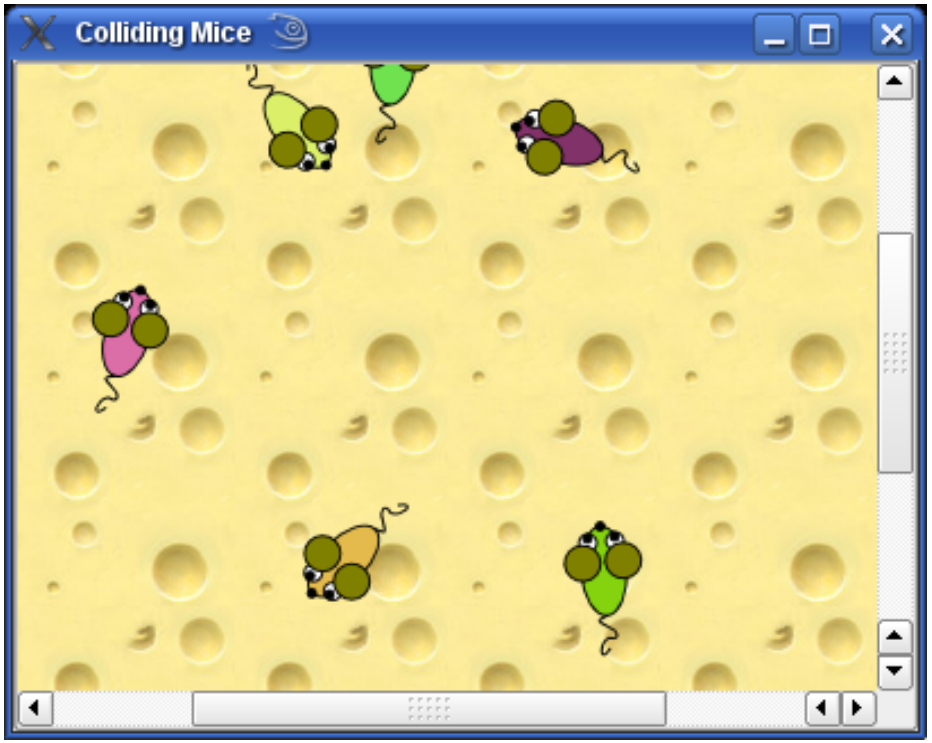
TROLLTECH DEV DAYS '07

Examples: Optimization Tricks

- FullViewportUpdate
  - Maximum rendering
  - No culling
  - Perfect for OpenGL

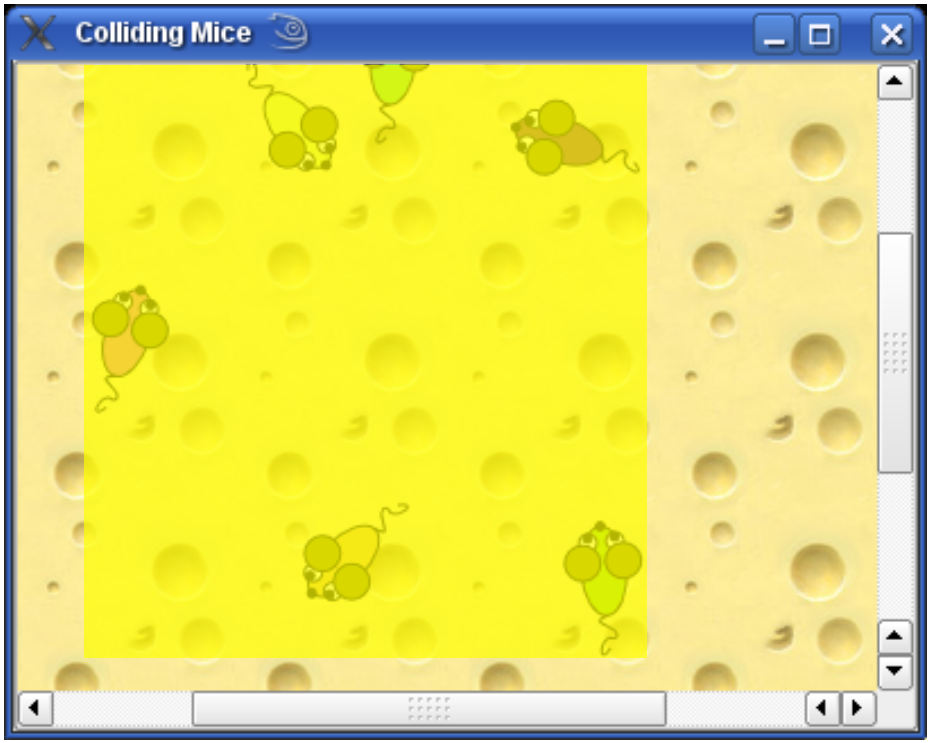
Examples: Optimization Tricks

- NoViewportUpdate
  - No rendering
  - No culling
  - “Do it yourself!”



Examples: Optimization Tricks

- BoundingRectViewportUpdate (4.4)
  - Medium rendering
  - Minimal culling
  - “QCanvas mode”



TROLLTECH DEV DAYS '07

Examples: Optimization Tricks

- Rendering as Fast As Possible
  - Use Level-Of-Detail (LOD) to simplify drawing
  - Don't draw what you can't see
  - Prepare to sacrifice “good looks” for speed

Examples: Optimization Tricks

```
void MyItem::paint(QPainter *painter, const QStyleOption...
{
    int lod = option->levelOfDetail;
    if (lod >= 1) {
        // 1:1 or closer
        painter->drawRoundRect(0, 0, 100, 100);
    } else if (lod >= 0.5) {
        // 1:2 to 1:1
        painter->drawRect(0, 0, 100, 100);
    } else if (lod >= 0.25) {
        // 1:4 to 1:2
        painter->fillRect(0, 0, 100, 100);
    } else {
        // Don't draw at all
    }
}
```

TROLLTECH DEV DAYS '07

Demo!

Examples: Optimization Tricks

- Rendering as Fast As Possible
  - Speed depends on the paint engine
  - The paint engines are different for each platform
  - Trial and error!

Examples: Optimization Tricks

- Rendering – avoid common pitfalls:
  - Curves are expensive
  - Dashed lines are expensive
  - Gradients are expensive
  - Alpha blending is expensive (antialiasing)
  - Transformed pixmaps are expensive

Examples: Optimization Tricks

- Rendering – stick to the fast API!
  - No antialiasing is usually faster
  - QPainter::CompositionMode_Source
  - Prefer int-functions in QPainter
    - painter->drawRect(QRect(0, 0, 100, 100)) - faster!
    - painter->drawLine(QLine(0, 0, 100, 100)) - faster!
  - Use *clever tricks*, e.g., simplify while moving
    - Remember: Perceived Speed is more important

Examples: Optimization Tricks

- Rendering – use OptimizationFlags (4.3)
  - Disable features you don't need
  - DontClipPainter – if your items never need to be clipped
  - DontSavePainterState – if you're a “good QPainter citizen”
  - DontAdjustForAntialiasing – if you're not using it

Examples: Optimization Tricks

- Rendering – consider OpenGL for everything
  - You can use raw OpenGL code in paint()
  - ...or in QGraphicsScene::drawItems()
  - ...or in QGraphicsView::drawItems()
  - Clever use of OpenGL unleashes unimaginable speed

Examples: Optimization Tricks

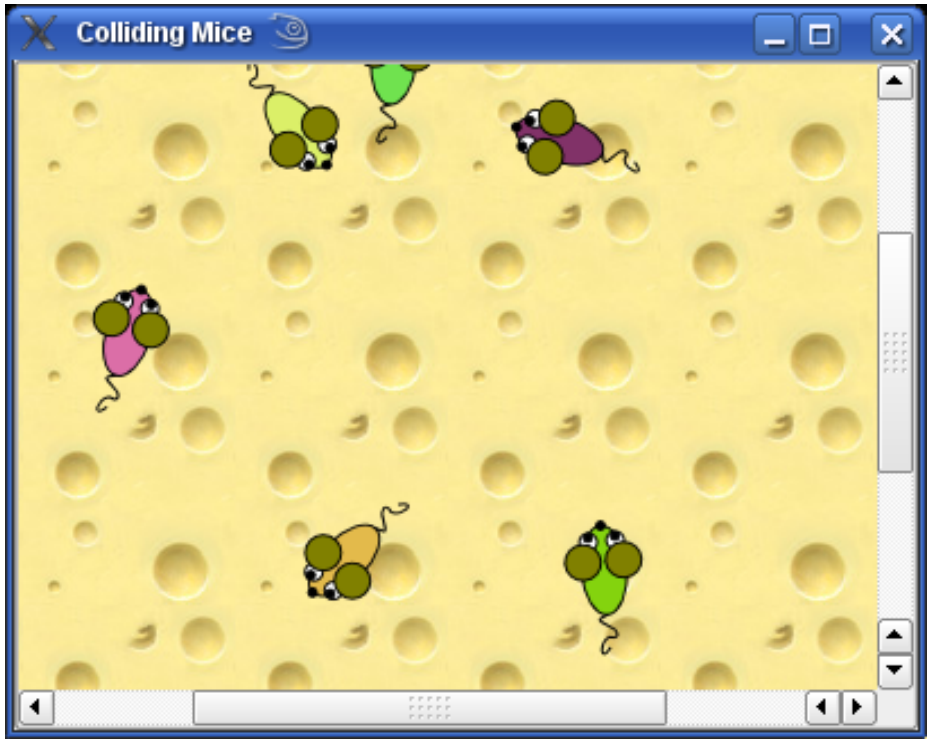
- Fast Culling & Hit-tests & Collision Detection
  - Write a FAST implementation of
    - boundingRect() - cache it!
    - shape() - cache it!
  - Return as few elements as possible in shape()
  - Make sure your boundingRect() is small enough
    - QT_FLUSH_PAINT is worth its weight in Gold
  - Choose the right BSP tree depth (if QGV guesses wrong)
    - QGraphicsScene::bspTreeDepth()

Examples: Optimization Tricks

- Caching
  - Cache the viewport background (CacheBackground)
  - Cache the item painting into a QPixmap
  - Qt 4.4
    - ItemCoordinateCaching
    - DeviceCoordinateCaching

Examples: Optimization Tricks

- Conclusion
  - Graphics View can be tuned to perform very well!



TROLLTECH DEV DAYS '07

Demo!

Examples: Widgets On Canvas

- Latest research
  - Providing drop-in support for QWidget-based widgets
  - Writing your own “QGraphicsWidget”
  - Support for
    - Layouts
    - Style
    - Palette
    - Font
    - Layout directions

TROLLTECH DEV DAYS '07

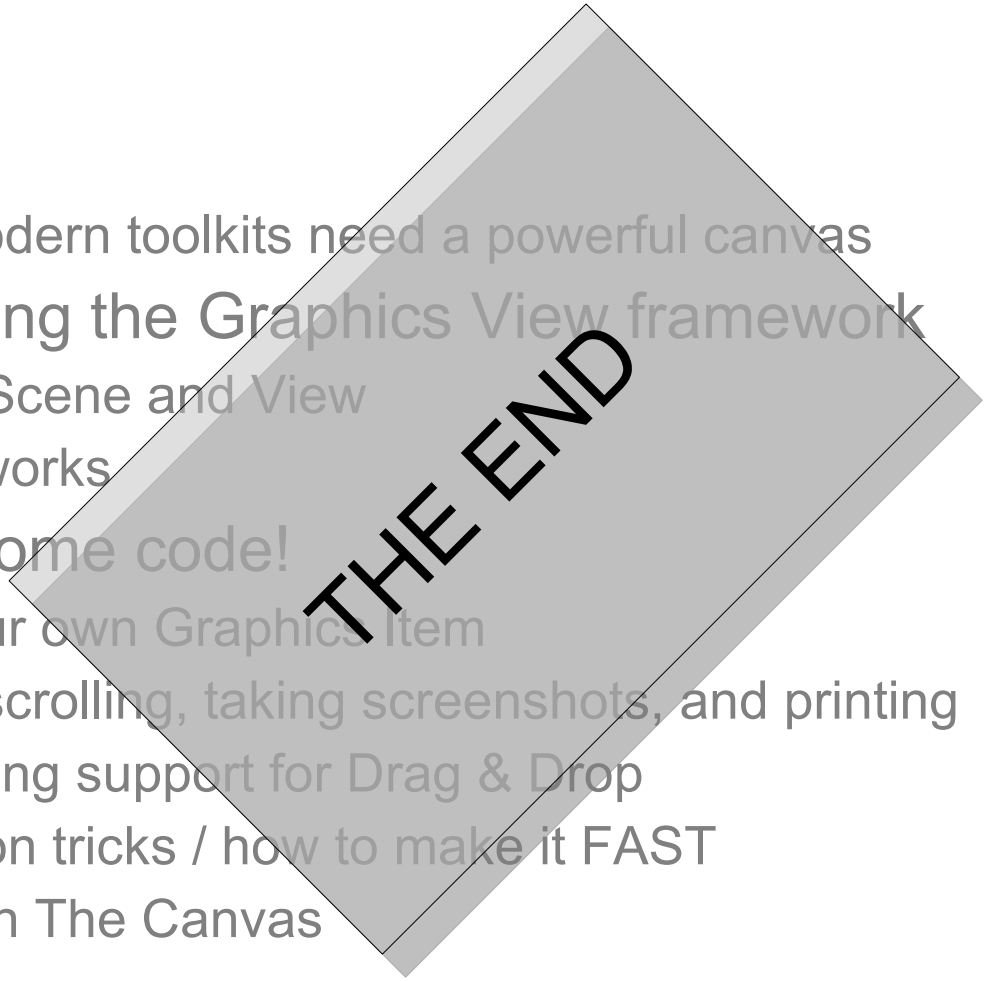
Demo!

Agenda

- Background
  - Why all modern toolkits need a powerful canvas
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all fits together
- **Let's write some code!**
  - **Writing your own Graphics Item**
  - **Zooming, scrolling, taking screenshots, and printing**
  - **Implementing support for Drag & Drop**
  - **Optimization tricks / how to make it FAST**
  - **Widgets On The Canvas**

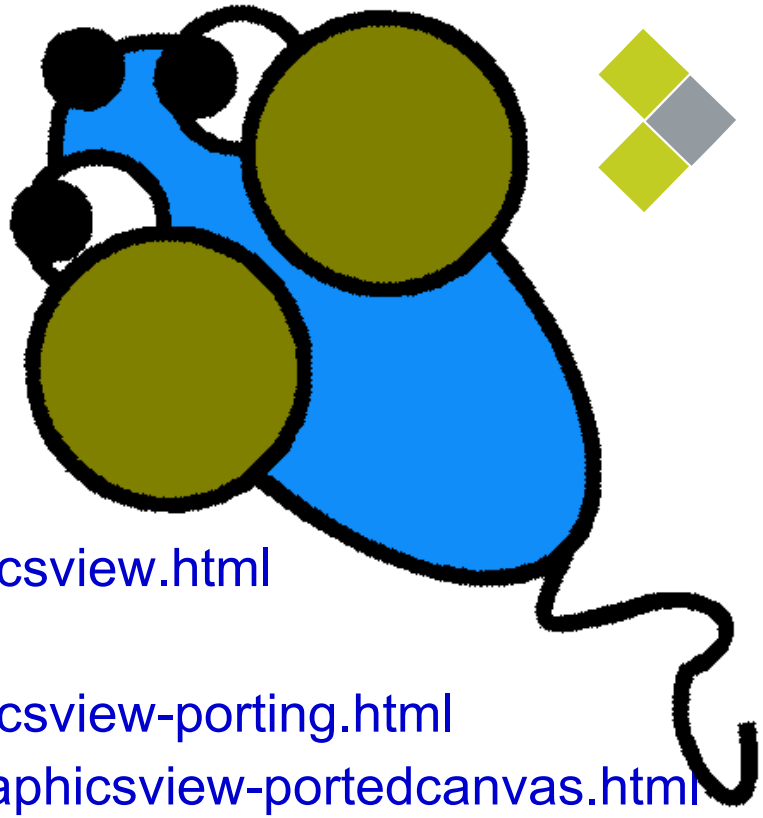
TROLLTECH DEV DAYS '07

Agenda



- Background
  - Why all modern toolkits need a powerful canvas
- Understanding the Graphics View framework
  - The Item, Scene and View
  - How it all works
- Let's write some code!
  - Writing your own Graphics Item
  - Zooming, scrolling, taking screenshots, and printing
  - Implementing support for Drag & Drop
  - Optimization tricks / how to make it FAST
  - Widgets On The Canvas

Graphics View



- Thank you for attending!
- Documentation:
  - Framework docs:
    - http://doc.trolltech.com/graphicsview.html
  - Porting from QCanvas:
    - http://doc.trolltech.com/graphicsview-porting.html
    - http://doc.trolltech.com/4.3/graphicsview-portedcanvas.html
- Blogs & early research:
  - http://labs.trolltech.com/