Code less.
Create more.
Deploy everywhere.

使用 Qt 和 OpenGL® 创建跨平台可视化 UI

概要

科学可视化、医学成像、飞行模拟、流程建模、动画、游戏和视觉效果应用程序都需要大量使用高性能 2D 和 3D 图形。标准的图形 API（如 OpenGL® ）非常适合渲染复杂的图形，但对用户界面编程却没有提供太多支持，无法满足多样化的市场。

在本文中，我们将详述可视化软件开发人员遇到的常见难题，并提供可轻易将高级 2D 和 3D 图形集成至本地高性能应用程序的技术方法。我们将借助 Qt® 应用框架，演示这些方法如何加快可视化开发，以及如何使用单一的代码库为多个操作系统进行开发部署。

NOKIA

介绍

大多数应用程序以表格、列表、图像、图形、地图和动画方式显示信息。这些方式当中，比较简单的可视化方法并不需要大量的图形处理，它们只是将数据映射到堆栈 widget。例如，使用表格 widget 显示数据库内容，使用列表视图显示日志文件内容。不过，较高级的可视化方法常常要使用一些低级的渲染技术（如 OpenGL®）和高级的 API （如 Scene graph、Open Inventor 和 Visualization ToolKit (VTK)）。这些高级方法通常直接控制图形处理器 (GPU)，以获得所需的图形处理能力。

所有的 widget 工具包都通过一套标准的 GUI widget 支持简单的可视化方法，但对高级可视化方法的支持程度却各不相同。在本白皮书中，我们会详述 Qt 是如何为高级可视化方法提供支持的，包括对 2D 和 3D 渲染以及与本地 widget 紧密集成提供直接支持。

常见用途

在本白皮书中，***高级可视化方法***指以下领域中需要使用的可视化方法：

- 医学成像 — 显示心电图 (ECG)、核磁共振成像 (MRI)、CT 扫描
- 地形成像 — 显示地理、天气和热量图
- 流程可视化 — 显示生产过程
- 数据挖掘 — 可视化显示历史记录、日志文件
- 航空和国防 — 雷达图像
- 汽车 — 车辆 3D 建模、CAD

标准成像 API

以下是两个用于高性能可视化的主要 API：

1. **OpenGL** — OpenGL 是与设备无关的跨平台图形 API，用于渲染 2D 和 3D 图形。
2. **Direct3D** — Direct3D 是图形加速 API，是Microsoft® DirectX 套件的组成部分，仅可用于 Windows®。

OpenGL 和 Direct3D 都是低级图形 API。应用程序无论使用哪一个，都必须以非常原生的方式告诉计算机如何绘制画面，如点、线和多边形。例如，要绘制一个立方体，就必须要指定立方体的八个边。实在不敢想象使用这样初级的 API 渲染诸如餐桌这样复杂的事物会是什么样。此外，这些 API 还必须按顺序调用函数，这就会非常容易出错。因此，支持面向对象的 3D 可视化的高级 API 就应运而生。有了高级 API，程序员就可以定义一个场景（场景图形 API），然后在场景中放置对象。

对于 2D 可视化，场景图形没有标准可言。功能和性能因工具包而异。在以下章节中，我们会详述 Qt 的 2D 场景图形 Graphics View 以及如何轻松地使用 OpenGL 渲染 2D 场景。

对于 3D 可视化，诸如 Open Inventor 和 VTK 等 API 被视为标准。我们将详述其与 Qt 应用程序的集成。

可视化开发的常见难题

以下列出可视化程序开发过程中一些最常见的难题：

- **支持本地用户界面**
  可视化 API（如 OpenGL）用于在计算机屏幕上的某个区域内快速渲染对象。它们并不创建窗口和 widget，也不会捕捉鼠标、键盘或窗口系统事件。这就需要通过“粘贴代码”的方式将 OpenGL 渲染的区域嵌入到窗口中。

- **支持多个平台**
  OpenGL 可跨平台，但是在选择了用于封装 OpenGL 场景的 widget 工具包后，可能会不必要的限制其在某些特定平台上的应用。

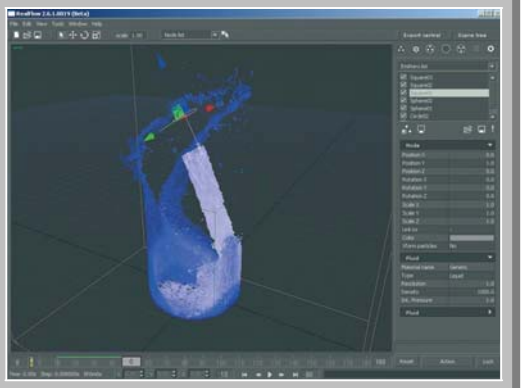
- **支持 2D 图形**
  大多数应用程序需要支持基本的 2D 绘图，包括绘制多边形、变形体、路径、梯度和图像控制。此外，有些应用程序需要用到在基本 2D 绘图基础上构建的功能，如绘制图表。除了这些基本的，无状态的 2D 绘图之外，还有些应用程序需要在这些基础上增加 2D 保留模式场景图形的功能。

- **实用的支持工具**
  可视化应用程序主要用来渲染高级图形，但仍需要附加的 API 来打印文档、解析 XML、读取数据库、使用线程、连接网络等等。理想情况下，工具包应支持这些附加的工具，并且可跨平台使用。

- **创建叠置**
  通常需要在可视化应用程序设置“控制器”按钮，例如用于缩放。在 OpenGL 区域中叠置本地 widget 的功能是 widget 工具包的一个特性。

在以下章节中，我们将了解 Qt 如何处理这些难题。

Qt in Use



公司：Next Limit Technologies
应用程序：RealFlow 4

当包括迪斯尼和 Pixar 在内的制片厂和视觉效果工作室需要生成水花飞溅、喷涌和漩涡效果时，他们常常求助于 Next Limit Technologies 及其 RealFlow 模拟软件。Next Limit 于 2007 年因其出色的工作赢得了奥斯卡技术成就奖，这一成就让电影工作者可以逼真地绘制出水和其他液体的流动。

RealFlow 最初是基于 Windows 开发的应用程序，但却受到倾向于 Linux 的潜在客户的热捧。因此 Next Limit 没有将 Win32 代码移植到 Linux，而是使用 Qt 重新构建了软件。

为了测试 Qt，Next Limit 设计了一个简便的应用程序，覆盖了大多数在 RealFlow 需要注意到的敏感问题，包括流畅处理 OpenGL 的能力以及在用户控制 GUI 时使用计算线程的能力。测试非常成功， 结果 Next Limit 开发人员只用了两个月就将 RealFlow 移植到 Qt。

Next Limit RealFlow 产品技术主管 Angel Tena 说“实在是太容易了。Qt 类设计的很好并易于使用。我们甚至常常不必去看文档。我们只要想出函数的名称，就会发现有一堆的函数可用。”

http://www.realflow.com/

Qt 简介

Qt 是用于桌面系统和嵌入式开发的跨平台应用程序框架，包括直观的 API 和丰富 C++ 类库、用于 GUI 开发和国际化的集成工具，支持使用 Java™ 和 C++ 语言进行开发。

Qt 通过 QPainter 和 QGraphicsView API 对高级 2D 图形提供了支持。此外，Qt 通过其 OpenGL 模块支持 3D 图形。在以下章节中对这些技术一一进行详细说明。

高级可视化解决方案

在本章节中，我们会详述 Qt 所提供的用于简化高级可视化应用程序开发的技术。

基本 2D 图形：QPainter

QPainter 提供了一个全面的 2D 绘图框架。除了渲染多边形、绘图路径、访射和非访射变形体这些基本功能之外，它还支持平滑处理、渐变画刷和 alpha 混合。使用 QPainter 实在是太方便了：

```
void Widget::paintEvent(QPaintEvent *event)
{
            QPainter painter(this);
            painter.setBrush(Qt::red);
            painter.drawRect(rect());
    // create a pen from any brush!
            painter.setPen(QPen(QBrush(“texture.png”), 1);
            painter.drawLine(10, 30, 40, 503);
}
```

QPainter 可用于所有 QpaintDevice，包括 widget (QWidget)、图像 (QImage)、打印机 (QPrinter) 和像素图 (QPixmap)。请注意，QPainter 只是用于 2D 渲染的 API。实际绘图是通过应用了 QPaintEngine 的后台程序完成的。Qt 提供了绘图引擎，可使用 Raster Graphics (Windows)、XRender (X11)、 OpenGL（所有平台）和 PDF（所有平台）。也就是说，上述的 painter.drawLine() 函数调用会自动转换为 OpenGL 命令或 PDF 命令，只需为 QPainter 提供合适的绘图引擎和采用支持该引擎的 QPaintDevice。我们将在后续章节中了解如何使用 QPainter 通过 OpenGL 绘制加速的 2D 图形。

2D 场景图形：Qt Graphics View

Qt Graphics View 提供了 2D 场景图形 API。它带有高级 API，用于在场景中放置对象（形体）并在多个视图中显示该场景。Graphics View 的优势就在于它为解决复杂难题提供了简单的 API。

要开始使用 Graphics View，只需创建包含所需 QGraphicsItems（形体）的 QGraphicsScene（场景）。然后，就可以使用 QGraphicsView（视图）可视化呈现该场景。

```
QGraphicsScene scene;
    scene.addRect(QRectF(0, 0, 100, 100));
    // populate more items in the scene

    QGraphicsView *view = new QGraphicsView;
    view->setScene(&scene);
    view->show();
```

通过继承 QGraphicsItem 可创建自定义条目。Graphics View 不要求条目一定是矩形。例如，我们可能需要一个不接受圆圈外鼠标点击事件的圆形条目。在这种情况下，让 shape() 返回条目的形状。Graphics View 通过 boundingRect() 返回的矩形来判断是否需要重新绘制条目。paint() 函数则执行实际的绘制。

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

   QPainterPath CircularItem::shape() const
   {
       QPainterPath path;
       path.addEllipse(boundingRect());
       return path;
   }

   void CircularItem::paint(QPainter *painter,
                        const QStyleOptionGraphicsItem *option,
                        QWidget *Widget = 0)
   {
       painter->drawEllipse(QRectF(0, 0, 100, 100));
   }
```

Graphics View 提供了可处理输入的完整框架，就如同使用了 widget。每个条目都可以接受鼠标和键盘事件。事件要么被处理，要么被忽略，忽略的话则会向上传递其父级、视图，并最终传递至场景。Graphics View 条目还支持拖拽、定制光标、提示信息和动画，这与 widget 类似。此外，Graphics View 条目可进行分组（可按组移动/选择条目）并支持碰撞检测。

请注意，QStyleOptionGraphicsItem 参数传递到上述 CircularItem 类的 paint() 函数。选项结构包含了一个有趣的字段，叫做 levelOfDetail，可为条目的缩放状态提供提示。当条目充分放大时，可跳过绘制细节。这样可优化复杂条目的渲染速度。

Graphics View 一个主要特性就是直接支持变换。可对单个条目，也可对整个场景进行变换。要为场景增加缩放功能，只需调用 scale()。在下例中，按 '+' 放大或按 '-' 缩小。代码编写如下：

```
class ZoomableView : public QGraphicsView
{
public:
    ZoomableView(QWidget *parent = 0) : QGraphicsView(parent) { }

protected:
    void keyPressEvent(QKeyEvent *event) {
        if (event->key() == Qt::Key_Plus) {
            scale(1.5, 1.5);
        } else if (event->key() == Qt::Key_Minus) {
            scale(1/1.5, 1/1.5);
        } else {
            event->ignore();
        }
    }
};
```

如果复杂场景带有上百万个条目，就需要考虑渲染的处理能力。Graphics View 支持通过一行 OpenGL 代码就可渲染场景：

```
  view->setViewport(new QGLWidget(QGLFormat(QGL::SampleBuffers))); // SampleBuffers for
antialiasing
```

在上述 CircularItem 的情况下，实际上是通过 OpenGL 绘制的圆圈。我们接着会在“OpenGL 绘图引擎”章节中看到，无需编写 OpenGL 代码，也是可实现这样不可思议的 OpenGL 渲染的，这完全要归功于 Qt 的可扩展绘图系统。

Qt 附带了芯片演示，使用 QGraphicsView 可视化 400000 个芯片。它使用了 Raster 引擎和 OpenGL 演示了芯片的可视化*（屏幕截图在下一页）*。

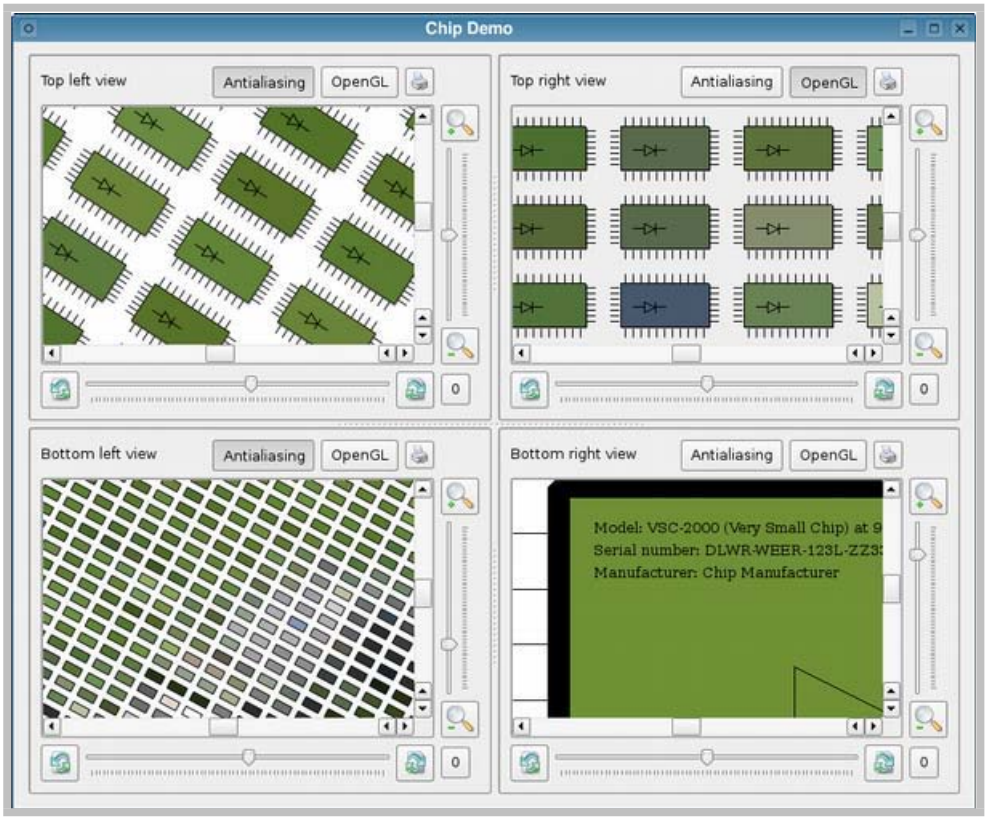


图 1：400000 个芯片以四个视图显示。每个视图可以有不同的变换，并可使用 OpenGL 进行渲染。

更高级的 2D 图形：画布中的 widget

在最新版 Qt (Qt 4.4) 中，Graphics View 具有在场景中放置本地 widget 功能。画布中的 widget 与普通对象操作类似，甚至可以进行变换。与本地 widgets 不同，画布中的 widget 在多个视图中同时实时显示。画布 widgets 是使用 QGraphicsProxyWidget 创建的：

```
QComboBox *modeCombo = new QComboBox;
    modeCombo->addItem("Landscape");
    QGraphicsProxyWidget *proxyWidget = new QGraphicsProxyWidget;
    proxyWidget->setWidget(modeCombo);
    scene.addItem(proxyWidget);
```

从 Qt 4.4 开始，Graphics View 还包括了用于 QGraphicsItems 的布局系统。要在 Graphics View 布局 (QGraphicsLayout) 中排列条目，条目通过继承 QGraphicsLayoutItem 类。然后，就可以使用 QGraphicsLinearLayout 或 QGraphicsGridLayout 线性（水平或垂直）或以网格排列条目。

通常，将 widget 放置在画布上来实现“控制器”widget，例如缩放按钮。Widget 条目与其他画布条目没有什么区别。当画布发生变换时，它们也会随之变换。但这不是我们想得到的效果。在放大画布时，控制器按钮不应该也随之放大。如果采用下列代码，就可避免此种情况发生：

```
// don't transform this widget with the canvas
  widgetItem->setFlags(QGraphicsItem::ItemIgnoresTransformations);
```

支持图像格式

Qt 的 QImage 支持标准的文件格式。它内置了对加载 PNG 和 XPM 文件的支持。由于图像通常以为某个特定领域优化的方式储存，大多数可视化应用程序需要支持加载定制文件格式。QImage 支持插件架构，可为新文件格式提供完整的 Qt 支持。通过继承 QimageIOPlugin 可以实现支持 JPG、GIF 和 MNG 的插件。请注意，插件可根据需要静态编译。

Qt 支持使用 Qt SVG 模块加载 SVG 1.2 Tiny 文件。它提供了 QsvgRenderer，可用来加载和绘制 SVG 文件>

```
class Widget : public QWidget
{
public:
    Widget(QWidget *parent = 0)
    {
        renderer = new QSvgRenderer(this);
        renderer->load(QString("sample.svg"));
        QObject::connect(renderer, SIGNAL(repaintNeeded()),
                         this, SLOT(update())); // animated SVG
    }

protected:
    void paintEvent(QPaintEvent *event)
    {
        QPainter painter(this);
        renderer->render(&painter);
    }

    QSvgRenderer *renderer;
};
```

在实际应用中，可以使用 QSvgWidget，利用 QSvgRenderer 来显示 SVG 文件。（如上例所示)

3D 图形：Qt OpenGL 模块

Qt 开发人员可使用 OpenGL 在 GUI 应用程序中绘制 3D 图形。Qt 提供了单独的 Qt OpenGL 模块，可将 OpenGL 图像与本地窗口系统集成在一起。

要将 OpenGL 配合 Qt 使用，要继承 QGLWidget。下面代码中演示的 QGLWidget 的主要功能如下：

- 调用 initializeGL() 来初始化 GL 环境。Qt 提供了易用的函数（如 qglClearColor 和 qglColor），可将 QColor 转换为 gl color。
- 在重新调整 widget 大小时调用 resizeGL()。可以调整视角，然后进行相应变换。
- 在 OpenGL widget 需要更新时调用 paintGL()。

```
class GLWidget : public QGLWidget
{
public:
    GLWidget(QWidget *parent = 0) : QGLWidget(parent)
    {
        setFormat(QGL::DoubleBuffer | QGL::DepthBuffer);
    }

protected:
    void initializeGL()
    {
        qglClearColor(Qt::white);
    }

    void resizeGL(int w, int h)
    {
        glViewport(0, 0, w, h);
        // setup project matrix
    }

    void paintGL()
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        // draw using gl
    }
};
```

QGLWidget 的使用方式与普通的 QWidget 相似。可创建多个 QGLWidget 并将其以指定大小放置在布局中。鼠标和键盘事件可通过 mousePressEvent() 和 keyPressEvent() 进行处理（如同 QWidget）。要创建动画，只需启动 QTimer，然后调用 updateGL()。

QGLFrameBufferObject 和 QGLPixelBuffer 分别提供了对 GL 帧缓冲对象和 GL 像素缓冲的支持。

Qt 提供加载图像并将其与纹理绑定的易用功能。例如，以下代码加载了图像并将其与纹理绑定>

```
texture = bindTexture(QPixmap(QString("image.png”)), GL_TEXTURE_2D);
```

Qt 在内部对已与纹理绑定的像素图/图像进行跟踪，这样就可在使用相同的图像文件或像素图时重新使用纹理。

可使用 renderPixmap() 将当前场景渲染为像素图。帧缓冲的内容可使用 grabFrameBuffer() 进行捕获。renderPixmap() 和 grabFrameBuffer() 的主要差别就是使用 renderPixmap() 时不捕捉叠置。

Qt 嵌入式（Qt 用于嵌入式 Linux 和 Windows CE 的产品）支持 OpenGL ES（OpenGL 的嵌入式版本）。使用 Qt 嵌入式，可以利用 OpenGL 通过 QWSGLWindowSurface 渲染整个窗口。

OpenGL 绘图引擎 — 使用 OpenGL 绘制 2D 图形

请记住，QPainter 只是个 API，实际的渲染是由 QPaintEngine 来执行的。Qt 带有 QGLPaintEngine，为 QPainter 提供了 OpenGL 后台。在 QGLWidget 基础上创建的 QPainter，自动将 QGLPaintEngine 作为绘画引擎。这样，任何使用此 QPainter 类型的 2D 图形都会自动解析至 OpenGL。

```
void myGLWidget::paintEvent(QPaintEvent *event)
{
            // First, create a QPainter on a QGLWidget
            QPainter painter(this);
            // That's it! All commands issued to QPainter are now
    // rendered using OpenGL!
            painter.setRenderHint(Qt::AntiAliasing);
            painter.drawLine(10, 203, 40, 30);
            painter.drawRect(QRectF(100, 100, 50, 50));
}
```

使用 QPainter 而不是 OpenGL 命令的优势是，可轻易将渲染在平台默认引擎和 OpenGL 之间切换。程序员无需具备 OpenGL 的知识就可进行操作。上述示例代码说明了如何利用这个小窍门通过 OpenGL 渲染 QGraphicsView。

同使用 OpenGL 相比，QPainter 确实对性能有轻微的影响，因为每次调用 QPainter 都会为虚拟函数调用产生附加的系统开销。但是如果应用程序并不要求很高的图形性能，这一点可忽略不计。最佳的解决方法就是动手尝试。

叠置支持

叠置是在场景上绘制的画面。这些画面一般不受场景变换的影响，通常可以认为它们与模型无关。

Qt 有三种机制 可以在 QGLWidget 上绘制叠置：

1. **使用 QPainter**

   我们可使用 QPainter 绘制 2D 叠置。由于 QGLWidget 是 QWidget，它会在 widget 要求更新时接收 paintEvent()。我们可以在 paintEvent() 中构造 QPainter，然后按以下方式绘制叠置：

```
GLWidget::GLWidget()
            {
                        // QPainter automatically calls glClear() unless
         // autoFillBackground is false.
                        setAutoFillBackground(false);
            }

            void GLWidget::paintEvent(QPaintEvent *event)
    {
                        makeCurrent(); // set the context for this widget
         paintGl();
                        QPainter painter(this);
                        drawOverlay(&painter);
                        // remember to swap buffers if double buffered!
            }
    void GLWidget::drawOverlay(QPainter *painter)
    {
        painter.drawRect(QRectF(50, 50, 100, 100),
                          QColor(255, 0, 0, 140 /* alpha */));
    }
```

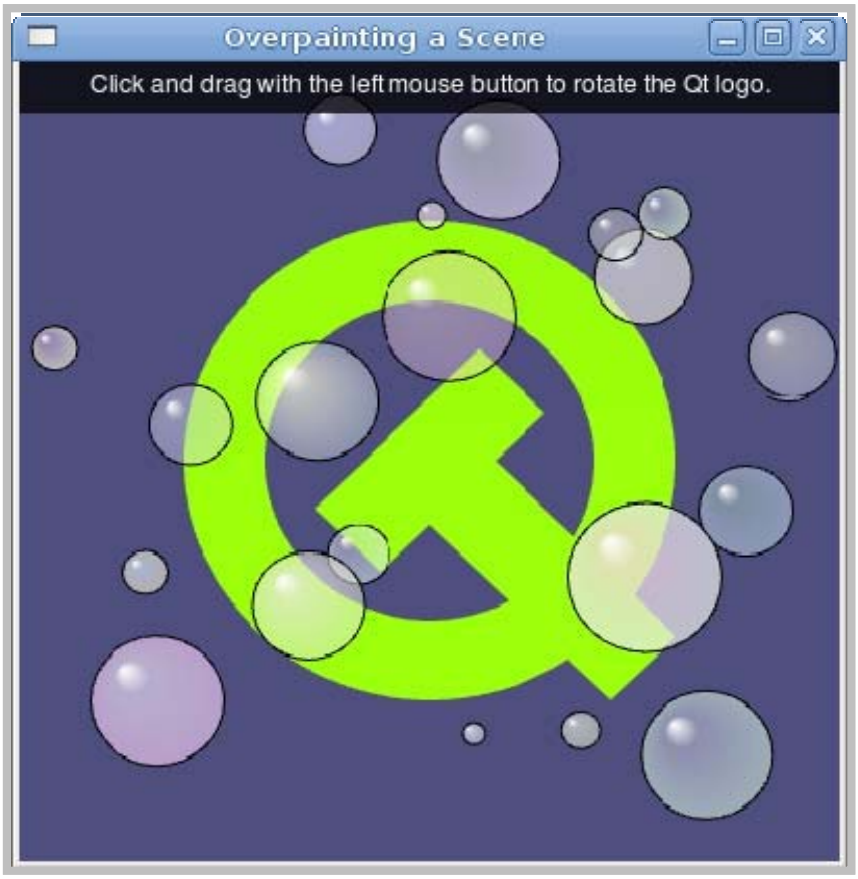


图 2: Qt 中包含了使用 QPainter 绘制叠置的示例。在主平面中绘制了“Qt”文字，而气泡则是使用 QPainter 绘制的叠置。

OpenGL 命令和 QPainter 还可交叉使用，前提条件是我们在发布 QPainter 命令（通过 glPushAttrib）之前保存 OpenGL 状态并在发布命令后重新进行恢复。由于场景必须重新进行绘制以便更新叠置，所以此方法仅在叠置未频繁更新时才奏效。

2. 使用帧缓冲对象

有时需要频繁更新叠置。例如，如果选择了橡皮筋作为叠置，那么在拖动鼠标时就需要重新绘制橡皮筋。如果采用 QPainter 方式，必须在每次重新绘制叠置时也重新绘制场景。

一个解决方案就是缓存当前场景为帧缓冲对象的。Qt 提供了 QGLFramebufferObject，可创建离屏表面。您可以使用 QGLFramebufferObject::bind() 在该表面上开始绘图，并使用 GLFramebufferObject::release() 结束绘图。一旦屏幕渲染为帧缓冲对象，就可用作场景区域的纹理。每次更新该场景时，Framebuffer 对象就会随之更新。使用 QPainter 或简单的 OpenGL 命令就可在场景区域中绘制叠置。

3. 使用 OpenGL 叠置

OpenGL 叠置是窗口系统界面的特性，有助于绑定 OpenGL 和窗口系统。基于 Windows 的 WGL，基于 X11 的 GLX 都支持 OpenGL 叠置。

Qt 提供创建叠置的跨平台接口。QGLFormat::hasOpenGLOverlays() 可用来在运行时检测平台是否支持叠置。

要创建带有叠置的 QGLWidget，需在 QGLFormat 指定。QGLWidget 提供了回调函数，可绘制与主平面类似的叠置：initializeOverlayGL()、resizeOverlayGL()、paintOverlayGL() 和 updateOverlayGL()。makeOverlayCurrent() 函数可以用于将叠置标记为当前上下文对象。

```
class GLWidget
{
      GLWidget(QWidget *parent)
             : QGLWidget(QGLFormat(QGL::HasOverlay)))
  {
  }

protected:
    void initializeOverlayGL()
    {
        // initialize the overlay
    }

    void resizeOverlayGL(int w, int h)
    {
        // overlay was resized
    }

    void paintGL()
    {
        // draw the overlay
    }
};
```

实用支持工具

Qt 不仅仅用于跨平台用户界面，它还带有大多数程序所需的一整套跨平台实用工具。例如：

- **Qt XML 模块** — 提供了针对 XML 的 DOM、SAX 流解析器。XmlPatterns 模块提供了 XQuery 和 XPath 支持。
- **Qt 多媒体模块** — 支持播放音频和视频文件。
- **Qt 数据库模块** — 支持使用插件式数据库驱动器进行 SQL 查询
- **Qt 网络模块** — 支持套接字和通用网络协议应用

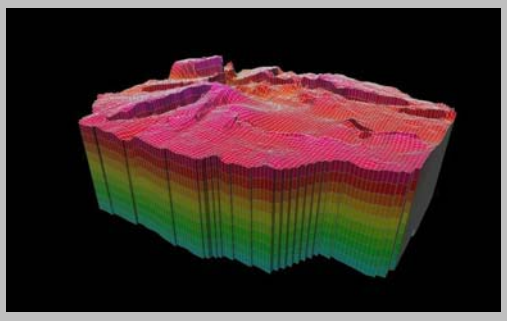
此处仅列举了一小部分而已。Qt 还支持打印（打印机或生成 PDF）、线程、自定义外观和更多内容。有关详细信息，请参见 Qt 文档 (http://doc.trolltech.com)。

第三方软件

OpenGL 是低级的 API，难以用来直接构建复杂可视化图形。因此，在 OpenGL 的基础上诞生了许多面向 3D 对象的 API：

- **Coin3D 和 Quarter** — Coin3D 是 Systems in Motion 的产品，是可保持模式的高级工具包，可用于高效 3D 图形开发。该产品是 Open Inventor API 的替代。使用 Coin 的应用程序可使用 SoQt 库与 Qt 集成。Quarter 提供了 QuarterWidget（从 QGLWidget 衍生而来），具有渲染 Coin 场景图形和将 QEvent 转化为 SoEvent 的功能。
- **VTK** — VTK 是与平台无关的图形引擎，支持并行渲染。可对 VTK 进行编译，通过 VTK_USE_GUISUPPORT 和 VTK_USE_QVTK 选项使用 Qt。有关详情，请参见 http://wiki.qtcentre.org/index.php?title=VTK_with_Qt。

Qt in Use



公司： Midland Valley Exploration Ltd.
应用程序： 4DVista

Midland Valley Exploration 团队曾一直计划着对其产品套件做一次大的改动，该套件是一套完善的构造地质应用程序，以 3D 可视化方式呈现了对地表下的勘探。他们希望新的应用软件能够在 Linux、Windows 以及传统的 Unix 系统上运行，并且能与其图形工具包 Coin 3D 无缝集成。使用了 Qt 后，团队简化了 UI 开发过程，并加快了编码速度。最终获得的结果实在令人惊叹，全新的应用程序没有经过任何代码调整，就可以在所有平台上自然的运行。此外， Qt 与Midland Valley 已有的图形工具包 Coin 3D 配合可以说是天衣无缝。MVE 团队还充分利用了 Qt 的 widget 和第三方附加软件和附带的库。

Midland Valley Exploration Ltd. 首席软件工程师 Colin Dunlop 说“再清楚不过了，Qt 是一个成熟的平台，它所提供的成熟的 OpenGL 模块支持和经过验证的 C++ 解决方案都是我们绝对需要的。”

http://www.mve.com/

总结

可视化应用程序需要使用与底层窗口系统的 widget 充分集成的 2D 和 3D API。对于 2D 图形，Qt 提供了完善的 Graphics View 框架，可在场景中放置对象并将该场景显示在多个视图中。对于使用 OpenGL 的 3D 图形，Qt 提供了 Qt OpenGL 模块，可将本地 widget 与 OpenGL 场景集成在一起。由于 Qt 可完全跨平台，使用 Qt OpenGL 的应用程序因此也可跨平台使用，也就是说，只需单一的源代码库就可对应用程序进行维护。

更多详情和下载

要了解 Qt 跨平台应用框架的更多详情以及更多客户案例和基于 Qt 的应用，请访问 Qt 网站：http://www.qtsoftware.com/。

免费评估版 Qt 适用于 Windows、Mac、Linux、嵌入式 Linux 和 Windows CE 平台。评估版不但带有源代码，我们还会在您进行评估期间提供技术支持。要下载 Qt，请访问 http://www.qtsoftware.com/downloads。

Qt Software 简介

t Software（前身为 Trolltech）是跨平台应用框架的全球领先者。诺基亚于 2008 年 6 月收购了 Trolltech，并其作为诺基亚内部的一个部门重新命名为 Qt Software。借助 Qt，开源和商业用户不用过多编写代码，却可创更多应用并可随时随地进行开发。开发人员使用 Qt 可一次性构建创新服务和应用程序，无需重新编写代码就可所有主要桌面系统、移动和其他嵌入式平台部署全新开发。众多领先的消费性电子厂商也可使用 Qt 创建用于nux 设备的高级用户界面。Qt 为诺基亚开发一系列用户倾慕的产品和体验的战略提供了稳固的基础。

诺基亚简介

诺基亚是全球移动行业的领先者，一直推动着互联网和通信行业融合的变革与发展。我们制造一系列移动设备，并提供各种服务和软件，使人们可以享受到音乐、导航、视频、电视、图像、游戏、和移动商务等诸多便利。拓展互联网服务以及企业解决方案和软件是我们的业务核心。我们还通过诺基亚西门子网络公司为通信网络提供设备、解决方案和服务。

Code less.
Create more.
Deploy everywhere.

NOKIA

挪威，奥斯陆 +47 21 60 48 00 美国加州，红木城 +1 650-551-1676 中国，北京 +86 10 6588 8836 qt-info@nokia.com www.qtsoftware.com