News from the Graphics Stack:
Improvements to the core OpenGL enablers in Qt 5.3 & 5.4

Laszlo Agocs
The Qt Company

© 2014 The Qt Company
Who am I?

• Senior software engineer at The Qt Company (formerly Digia) in Oslo, Norway

• OpenGL, graphics, windowing systems, platform integration

• Previously at ARM and Nokia
Why are we here?

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
The long road to QQuickWidget
class Widget : public QGLWidget, protected QGLFunctions
{
  void initializeGL() { initializeGLFunctions(); ... }
  void resizeGL(int w, int h) { ... }
  void paintGL() {
    // Render directly via GL or open a QPainter on 'this'
  }
};
Qt 4

Native windows
Qt 5.0

- QWindow, QOpenGLContext, QSurfaceFormat
- QQuickWindow, QQuickView
- QGL*
Qt 5.1

- `QWidget::createWindowContainer()`
Qt 5.3

• New concept: Avoid native windows, render offscreen and composite in the widget stack

• Essential for embedded

• QQuickWidget
QQuickWidget

This is a semi-transparent overlay widget which is placed on top of the Quick content.

Hello Qt is always better than

Hello Qt
black has alpha == 0
And it just works. Even on embedded.

A QWidget with two QOpenGLWidgets running on i.MX6 Sabre LITE on eglfs

A Qt Quick 2 scene embedded into a widget UI using QQuickWidget
Composition

- Does this mean all widget apps require OpenGL from now on? No.
- Does this support multisampling? Yes.
- Qt::WA_AlwaysStackOnTop
QQuickWidget API

- Mirrors QQuickView
- Except that it is a QWidget
QQuickRenderControl

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
QQuickRenderControl

Qt Quick in a texture
QQuickRenderControl

- No on-screen QQuickWindow.
- The rendering of the scene is redirected into a FBO.
- Texture can be used in arbitrary ways. No costly readbacks.
- Need to send input to Quick? QCoreApplication::sendEvent()
- Integrate Quick into external engines, not the other way round.
QOpenGLWidget

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
• Back to widgets. No more boring QML stuff.

• QOpenGLWidget is here

• R.I.P. QGLWidget

10 QOpenGLWidget instances in one top-level window
QOpenGLWidget

• Basic API is familiar
• Can open a QPainter on it
• Utilities like bindTexture() are gone → QOpenGLTexture

```cpp
class Widget : public QOpenGLWidget, protected QOpenGLFunctions
{
public:
    void initializeGL() { initializeOpenGLFunctions(); ... }
    void resizeGL(int w, int h) { ... }
    void paintGL() { ... }
};
```
One visible native window only, unlike QGLWidget.
Animation

- QTimer::singleShot(16, m_legacyQGLWidget, SLOT(updateGL()))
- Swap interval defaults to 1 since Qt 5.3
- QSurfaceFormat::setSwapInterval()
- Schedule repaints with update() and rely on vsync
- Timers with small intervals can be useful, but know what you are doing
Formats and contexts

• QSurfaceFormat::setDefaultFormat()

• Easy to request OpenGL 3+ or core profile for everything in the app

• The new compositing architecture relies heavily on multiple contexts and resource sharing.

• QOpenGLWidget comes with documentation. Use it.
Window 1 with two QOpenGLWidgets

TLW context 1

Context 1  Context 2

Window 2 with two QOpenGLWidgets

TLW context 2

Context 3  Context 4
Formats and contexts, cont.

- `Qt::AA_ShareOpenGLContexts`

![Diagram showing contexts and sharing]

Window 1 with two `QOpenGLWidgets`

- TLW context 1
  - Context 1
  - Context 2

Global share context

Window 2 with two `QOpenGLWidgets`

- TLW context 2
  - Context 3
  - Context 4
High DPI screens

- OpenGL operates in pixel dimensions on retina screens.
- \texttt{new QOpenGLFramebufferObject(w->devicePixelRatio() * w->size())}
- Qt takes care of all the internal framebuffers and viewports. The rest is up to you.
- Watch out for screen changes
  - \texttt{screenChanged(QScreen*)} signal for QWindow
  - \texttt{ScreenChangeInternal} event for QWidget
QOpenGLWindow

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
QWindow

- Enough of widgets, I don't want your push buttons.
- QWindow is fine for you then.
- Lightweight and powerful, but...
class MyWindow : public QOpenGLWindow {
    void resizeGL(int w, int h) { ... }
    void paintGL() {
        QOpenGLFunctions *f = context()->functions();
        f->glClearColor(GL_COLOR_BIT | GL_DEPTH_BUFFER_BIT);
        // issue some native OpenGL commands
        ...
        QPainter p(this);
        // draw using QPainter
        ...
        // animate continuously: assume blocking swap and just schedule an update
        update();
    }
};
QRasterWindow

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
QRasterWindow

- QOpenGLWidget has a little brother: QRasterWindow
- Exactly what the name suggests. Nothing more, nothing less.

```cpp
class HelloWindow : public QRasterWindow
{
    void paintEvent(QPaintEvent *) {
        QPainter painter(this);
        painter.fillRect(0, 0, width(), height(), Qt::white);
        painter->drawText(QRectF(0, 0, width(), height()), Qt::AlignCenter, QStringLiteral("Hello world"));
    }
};
```
Summary

- QQuickWidget
- QQuickRenderControl
- QOpenGLWidget
- QOpenGLWindow
- QRasterWindow
<table>
<thead>
<tr>
<th>Qt Quick 2 application</th>
<th>QQuickView or QQmlApplicationEngine</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL without any need for Qt widgets</td>
<td>QOpenGLWindow</td>
</tr>
<tr>
<td>Custom Q Painter-based drawing without Qt widgets</td>
<td>QRasterWindow</td>
</tr>
<tr>
<td>OpenGL content in a widget-based application</td>
<td>QOpenGLWidget</td>
</tr>
<tr>
<td>Qt Quick 2 content in a widget-based application</td>
<td>QQuickWidget</td>
</tr>
<tr>
<td>Qt Quick 2 content used as a texture in a custom OpenGL renderer</td>
<td>QOpenGLWindow + QQuickRenderControl</td>
</tr>
</tbody>
</table>
What was left out

- Dynamic OpenGL implementation loading on Windows
- Adoption of existing native OpenGL contexts
Thank you!

www.qt.io
www.qt.io

See you there!