Qt and Multithreaded OpenGL in Cross-Platform Applications

Matthew Hunt
Senior Software Design Engineer
Agenda

- Navico and Qt
- Product and Design Overview
- QPA with Windowed OpenGL
- Multithreaded OpenGL in Qt
Who are we?

• Navico is the world’s largest marine electronics company
• Navico designs, develops, and manufactures a wide range of products for leisure and professional marine markets
What do we make?
What do we make?
What do we make?
What do we make?

View your chartplotter anywhere onboard... using your iPad or iPhone... then take control from anywhere onboard.
Agenda

• Navico and Qt
• Product and Design Overview
• QPA with Windowed OpenGL
• Multithreaded OpenGL in Qt
Navico and Qt

• Began investigating Qt in 2006
• Needed to invest in development of a new codebase that could support many platforms
  – Existing codebase had reached its limits (reusability vs. speed)
  – Easier driver / BSP support in Linux
  – Need to maximize GUI code reuse
Navico and Qt

• Started development with Qt v4.4
• Development to support three platforms simultaneously
  – Windows
  – X11 Linux
  – Embedded Linux with QWS
Navico and Qt

• Initial OpenGL Support
  – Windows: Modern OpenGL (2.0+)
  – X11 Linux: OpenGL 1.5
  – Embedded Linux: OpenGL ES 1.1
  – Embedded Linux: OpenGL ES 2.0
Agenda

• Navico and Qt
• Product and Design Overview
• QPA with Windowed OpenGL
• Multithreaded OpenGL in Qt
Multi-Function Devices (MFDs)
Multi-Function Devices (MFDs)
Design Constraints

- Common codebase, but different OpenGL versions
- Third Party closed-source modules
- Multiple independent OpenGL windows
- Windowed, multithreaded OpenGL
Product Demos
Agenda

• Navico and Qt
• Product and Design Overview
• QPA with Windowed OpenGL
• Multithreaded OpenGL in Qt
Implementation Challenges

• The Dark Times: Achieving windowed, threaded OpenGL support in Qt v4.4
A New Direction: Lighthouse

- The Lighthouse project was realized as the Qt Platform Abstraction (QPA) in Qt 4.8
- Replaced the QWS / Qt for Embedded Linux gfx-plugin approach
- Navico migrated completely to Qt 4.8 to take advantage of the QPA on embedded devices
QPA in Qt 4.8

• QPA Plugin Creation Challenges
  – Little documentation
  – No guarantee of API forward compatibility

• Only for embedded platforms
  – Windows and X11 still had their own platform-specific code in QtGui
QPA in Qt5

• No more platform specific code in QtGui!
• QPA now used for all platforms, Windows, Linux X11, Embedded Linux, Android, etc.

// // WARNING
// --------------
// This file is part of the QPA API and is not meant to be used
// in applications. Usage of this API may make your code
// source and binary incompatible with future versions of Qt.
// //
QPA in Qt5

• The bad news
  – Still lacking good documentation
  – Still no guarantee of API forward-compatibility

• The good news
  – Changes in QPA between Qt v4.8 and Qt5 have been mostly additions and not modifications or removals
QPA Basic Classes

• QPlatformIntegration
  – Provides the basic links between Qt and the underlying windowing system
  – Manages the physical display(s)
  – Creates native windows and/or surfaces for QWidgets
  – Creates native pixel storage for QPixmaps
  – Handles font abstractions
QPA Basic Classes

• QPlatformScreen
  – Represents the physical display
  – Manages geometry and format for the screen
QPA Basic Classes

• QPlatformWindow
  – Represents a window within the screen
  – Does not include the surface / content
    • Only represents the window
  – Provides access to the OpenGL context for the window (if available)
QPA Basic Classes

- QWindowSurface (Qt 4.8)
- QPlatformBackingStore (Qt 5)
  - Defines the surface / content for a window
  - Manages painting and flushing to the surface
QPA Basic Classes

- QPlatformGLContext (Qt 4.8)
- QPlatformOpenGLContext (Qt 5)
  - Abstraction for OpenGL rendering context
  - Provides basic makeCurrent() / doneCurrent() / swapBuffers() APIs
QPA Class Relationship

**QPA in Qt 4.8**
- QPlatformIntegration
- QPlatformScreen
- QPlatformWindow
- QWindowSurface
- QPlatformGLContext

**QPA in Qt 5**
- QPlatformIntegration
- QPlatformScreen
- QPlatformWindow
- QPlatformBackingStore
- QPlatformOpenGLContext
– What Is It?

• Provides the abstraction between rendering APIs and the underlying windowing system
• Similar in purpose to the QPA
QPA / EGL Relationship

- Application
- Qt GUI Classes (QWidget, QPixmap, etc.)
- QPA Platform Plugin
- Native Windowing System
- EGL
EGL – Basic Components

• EGLDisplay
  – Represents the abstract display
  – Equivalent to QPlatformScreen

• EGLSurface
  – Represents a rendering surface
EGL – Basic Components

• EGLContext
  – State machine for the rendering API
  – Corresponds to QPlatformGLContext / QPlatformOpenGLContext
  – On the EGL side, a context can be “made current” on any valid EGLSurface associated with the EGLDisplay
Agenda

• Navico and Qt
• Product and Design Overview
• QPA with Windowed OpenGL
• Multithreaded OpenGL in Qt
Threaded OpenGL

• Why have multiple threads?
  – Need to keep the GUI thread responsive
  – Texture uploads can be slow
  – Buffer swapping can be slow
Mobile Architecture Challenges

• Memory Bandwidth
  — Necessarily restricted
  — Shared with CPU

• Cache Limitations

• Architectural Differences
  — Mobile GPUs are not just scaled-down desktop GPUs
The “QGL” Classes

• QGLWidget
  – Encapsulates both a window and an OpenGL context
  – Not really designed for threaded rendering
    • All default implementations will attempt to re-bind context to GUI thread

• QGLContext
  – Encapsulates an OpenGL context
The “QGL” Classes

• Other Helpful Classes
  – QGLBuffer
  – QGLShader
  – QGLShaderProgram
  – QGLFrameBufferObject
Threaded QGL 4.8 Example
OpenGL without QGLWidget?

• All we really need is a Window and an OpenGL Context
• QGLWidget default implementation may not be helpful
• Alternative: Use QWidget and QGLContext
QWidget OpenGL Example
Threaded QGL in Qt5

• Support for QGL-classes in Qt 5: What changed?
  – The QWidget approach no longer works so easily
  – QGLContext now keeps track of its thread affinity via QThread
    • makeCurrent()/doneCurrent() isn’t sufficient
ANGLE in Qt5

- Qt5 can be configured to use desktop OpenGL or ANGLE
- ANGLE provides transparent cross-platform development with OpenGL ES 2.0
- Unfortunately, the current ANGLE implementation is not fully thread safe, so multiple contexts may not work.
Threaded QGL 5.0 Example
QOpenGL

• New with Qt5
• Intended as a replacement for “QGL”
  – But, can be used alongside existing QGL classes
• Recommended for all new OpenGL development
• Can be integrated into existing widget-based applications
QOpenGL Classes

• QWindow
  – Effectively replaces QGLWidget
  – Back-end equivalent is QPlatformWindow

• QOpenGLContext
  – Replaces QGLContext
  – Back-end equivalent is QPlatformOpenGLContext
QOpenGL Classes

• Other helpful classes
  – QOpenGLBuffer
  – QOpenGLShaderProgram
  – QOpenGLFrameBufferObject

• These replace their QGL-equivalents
Integrating QOpenGL

- `QWidget::createWindowContainer()` can be used to insert QWindows into QWidget-based applications
Threaded QOpenGL Example
Questions?