Qt Quick Graphics State of the Union

László Agócs & Andy Nichols
@alpqr / lagocs & @nezticle / nezticle
Senior Software Engineers, Graphics/Multimedia
The Qt Company, Oslo, Norway

QtCon – Berlin - 3 September 2016

Agenda

- Qt Quick modularization in Qt 5.8
- Integrated software backend
- Experimental Direct3D 12 backend
- Benefits and limitations
- Consequences for shaders and custom Qt Quick items
- Future plans

Qt Quick and OpenGL

- Qt Quick 2 in Qt 5.0
  - Scene graph designed for OpenGL ES 2.0
  - Moved away from the QPainter world
  - Initial focus on mobile & embedded
  - Assumptions for OpenGL (2.0 + FBO or ES 2.0) being available everywhere

OpenGL on Desktop

- The latter was a bit too much to wish for.
  - For example, old Windows machines with no or outdated drivers.
  - Remoting (RDP, X forwarding, etc.)
- To ease the pain on Windows, Qt 5.0 shipped with ANGLE, the EGL/GLES to DXGI/D3D9/11 translator also used by Chromium and friends.
- Later added Mesa llvmpipe as an option (full software rasterizer).

OpenGL on Low-end Embedded Linux Boards

- No GPU.
- Software rasterizers not an option for the low-powered embedded world.
- Initially no solution for Qt Quick – stick with QWidget.
- Later: Qt Quick 2D Renderer. Commercial only. First targeting directfb so that at least some operations like blits get accelerated.
- Turns out it’s pretty ok for linuxfb as well, as long as the screen resolution is small.

OpenGL on Low-end Embedded Linux Boards

- The low-cost embedded segment is more important than many think.
- Two obvious improvement candidates surfaced around Qt 5.7:
  - Partial updates by tracking dirty areas in order to help apps with little animation.
  - Proper –no-opengl support in qtdeclarative. No dummy EGL/GLES libs like before with the separate 2D Renderer module.

New Graphics APIs

- And then came Metal, Direct3D 12 and Vulkan.
- Does not mean Qt (and Qt Quick) has to support all of them.
- But: should have the ability and means to do so, in case it becomes beneficial.
- Hence the need to modularize a bit.

New Graphics APIs

- The new, low-level APIs present various research and improvement opportunities:
  - Some, unlike OpenGL, are the platform in question's primary, vendor-supported API, with potentially better tooling and system integration.
  - Threading. Parallel command submission, texture management, etc.
  - Different approaches to shaders. More pre-compilation. (D3D shader bytecode, SPIR-V)

Pre-work: QD3D12Window

- End of 2015: integrate D3D12 into QWindow apps (or embed into QWidget)
  - https://blog.qt.io/blog/2016/01/28/qt-and-direct3d-12-first-encounter/
- Vulkan not released at the time.
- DXGI+D3D is interesting anyway:
  - Long-term plans for removing ANGLE (in the far far future…)
  - Can we handle special situations (device reset, driver update, etc.) better, while having fewer driver issues?
  - Graphics debugger in VS, single set of tools (e.g. run fxc from a qmake rule and be done with it, no vendor-specific mess)
  - WARP

Post-work ;)

- Something similar for Vulkan, just with a lot fewer examples:

  https://github.com/alpqr/qtvulkan

Qt 5.8

- So what about Qt Quick?
- Qt 5.8 modularizes: no strict OpenGL requirement anymore.
- Builds on the existing concept of scenegraph plugins.
  - Software backend built-in.
  - Experimental D3D12 backend as a plugin.
- https://doc-snapshots.qt.io/qt5-dev/qtquick-visualcanvas-adaptations.html

Qt 5.8 – Software Backend

- All candidate features implemented:
  - Built-in, LGPLv3 + Commercial license like the rest of qtdeclarative.
  - Partial updates!
  - No more GL lib requirement: Qt Quick no longer skipped when OpenGL is disabled in the Qt configuration.
  - Force with QT_QUICK_BACKEND=software or QQuickWindow::setSceneGraphBackend(QSGRendererInterface::Software)

Qt 5.8 – Software Backend

- This is not the same as a software OpenGL rasterizer.
- Much more lightweight. Goes through the scene and makes ordinary QPainter calls.
- Suitable for the low-end.
- Less complete, obviously, no shader effects and particles for instance.

Qt 5.8 – Software Backend

Demo time!

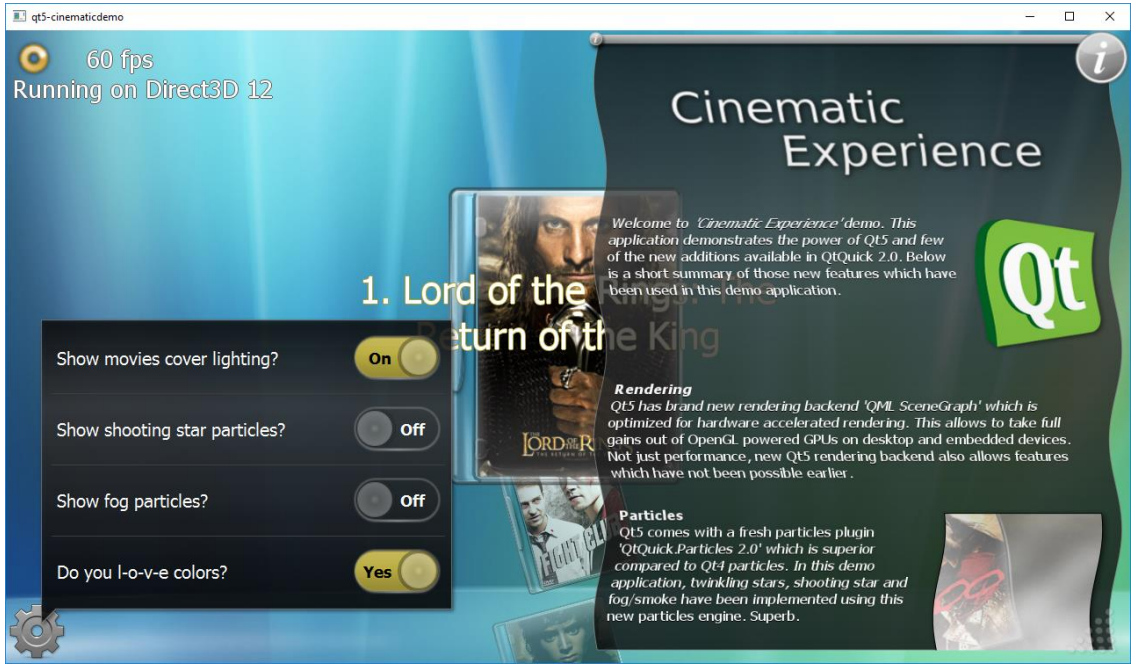
Qt 5.8 – D3D12 Backend

- When building qtdeclarative on Windows 10, the new plugin will be built as well.
- Supported in practice on MSVC2015 only.
- Not just traditional Windows, but UWP (WinRT) too!
- OpenGL is always the default (except in –no-opengl builds). D3D12 must always be explicitly requested: set QT_QUICK_BACKEND=d3d12 or via QSGRendererInterface::Direct3D12.

Qt 5.8 – D3D12 Backend

- Recommended reading:

  https://doc-snapshots.qt.io/qt5-dev/qtquick-visualcanvas-adaptations-d3d12.html



Qt 5.8 – D3D12 Backend – Renderer

- The default OpenGL renderer of scenegraph batches since Qt 5.2.
  - Helps many cases, but may have no benefit for others. Increased CPU usage.

- When doing a new backend for any new graphics API, it is likely better to get started with a pre-5.2 style renderer: one item – one draw call.

- D3D12 follows this. So no batching for now.
  - State changes are less of a problem anyway.
  - Batching is still a goal long-term.

Qt 5.8 – D3D12 Backend – Render Loops

- Implemented both a threaded and a non-threaded render loop.
- That is, the equivalents of the “threaded” and “windows” render loops.
- http://doc.qt.io/qt-5/qtquick-visualcanvas-scenegraph.html#scene-graph-and-rendering mostly applies to D3D12 as well.
- There’s a catch:
  - DXGI calls may require the gui thread to be able to deliver events.
  - This is impossible with the threaded scenegraph design where gui blocks.
  - So the non-threaded loop is the default for D3D12. ☹

Qt 5.8 – D3D12 Backend – Robustness

- What happens when the GPU is lost, due to driver update for instance?
- OpenGL-based Qt apps typically crash.
  - ANGLE may survive, not sure.
- Not anymore with the D3D12. Device losses are (should be) handled correctly from day one.

Qt 5.8 – D3D12 Backend – Shaders

- All the built-in materials’ shaders are HLSL 5.0
- Pre-compiled to D3D shader bytecode at build time via fxc
- Simple and fast
- Material system is different from GL – no uniforms, only one constant buffer.

Qt 5.8 – D3D12 Backend – HLSL in Applications

- What about applications using the ShaderEffect item?
- https://doc-snapshots.qt.io/qt5-dev/qml-qtquick-shadereffect.html#direct3d-and-hlsl
- Options:
  - Shader source strings: GLSL, HLSL
  - Shader source files: GLSL, HLSL
  - Shader bytecode files: HLSL (fxc)

Qt 5.8 – D3D12 Backend – HLSL in Applications

- Branch via GraphicsInfo (new, replacing OpenGLInfo)
- Or yet better: file selectors!
  - Resource structure:
    - */*
      - **shaders**
        - **effect.frag** -> GLSL source code
        - **+hlsl**
          - **effect.frag** -> HLSL source code OR pre-compiled bytecode
  - The rest is magic:
    ShaderEffect { … fragmentShader: “qrc:shaders/effect.frag” }

Qt 5.8 – D3D12 Backend – HLSL in Applications

- +glslcore selector also supported to select OpenGL core profile context compatible shaders.
- Some vague long-term ideas for a more unified shading language solution.
  - However, Qt is no Unity/Unreal/CryEngine so don’t hold your breath.
  - Porting/adding multiple shader variants is pretty painless in our experience, when it comes to the typical GUI use cases.

Qt 5.8 – D3D12 Backend – Multisampling

- As expected, a QSurfaceFormat with samples 4, 8, … will request a multisample render target.
- It’s all done in the backend though, not in the driver. Flip swapchains have no multisample support built-in.
  - The backend will create extra multisample targets and resolve the samples before Present.

Qt 5.8 – D3D12 Backend – Semi-transparent Windows

- Request a non-zero alpha size in the QSurfaceFormat.
- Note: This is not free since it changes behavior heavily: will go through DirectComposition.
- Use only when you really need a window with (semi)transparent areas.

Qt 5.8 – D3D12 Backend - Mipmaps

- Has to be implemented in the backend, no driver support.
- Compute shader.
- We only handle power-of-two images for now.
  - Rest involves a scaling on the CPU, ouch.
  - This has to be fixed later.
- Should be painless, but may have glitches.
  - So do not do mipmap: true for Image unless really needed

Qt 5.8 – D3D12 Backend – Limitations

- What’s missing?
  - Particles
  - Distance field based text rendering
  - QQuickFramebufferObject / Canvas via FBO
  - QQuickWidget
  - Texture atlases
  - Point/lines with width != 1
  - Rendering via QSGEngine/QSGAbstractRenderer
- Will fail gracefully in most cases, e.g. particles are just not shown.

Qt 5.8 – Custom Items

- There’s a catch with non-OpenGL backends when it comes to custom QQuickItem implementations.
- You cannot do your own materials atm. QSGMaterialShader and most of QSGSimple* is tied to GL and won’t work.
- Options:
  - Use QSGRectangleNode and QSGImageNode via QQuickWindow to get cross-backend simple rectangle and image nodes.
  - Use QSGRenderNode to do custom rendering.

Qt 5.8 – QSGRenderNode

- http://code.qt.io/cgit/qt/qtdeclarative.git/tree/examples/quick/scenegraph/rendernode?h=5.8

- Previously internal (used by QtWebKit). Now enhanced and made public.

Qt 5.8 – QSGRendererInterface

- https://doc-snapshots.qt.io/qt5-dev/qquickwindow.html#rendererInterface

Qt 5.8 – Simple rect/image node

- https://doc-snapshots.qt.io/qt5-dev/qquickwindow.html#createRectangleNode

Qt 5.8 – Other Modules?

- Modules outside qtdeclarative are usually tied to OpenGL.
- Do not expect QtGraphicalEffects, QtWebEngine, QtCanvas3D, Qt3D, etc. to work through D3D12.
- QtLocation is partially functional with software and D3D12 as well.

Future – Qt 5.9 and beyond

- OpenGL not going anywhere.
  - ES 3.1/3.2 optimizations
  - Compressed textures
  - Pre-compiled shaders

- Printing. Software backend allows drawing scenes onto a QPaintDevice. Could be used for proper Qt Quick printing.

- **Path rendering. Shapes**, SVG, …
  - Bridge the gap between QOpenGLPaintEngine and Qt Quick
  - GL_NV_path_rendering

Thank You!

https://github.com/alpqr/qtcon2016