Qt World Summit 2017

October 10-12 | Berlin, Germany

Modern Graphics APIs in Qt: Vulkan and friends

László Agócs (lagocs / @alpqr)
Graphics & Multimedia
The Qt Company, Oslo, Norway

The Qt Company

Vulkan support as of Qt 5.10

Vulkan

- New generation graphics and compute API.
- Available at least on Windows, Linux, Android.
- Includes WSI bits too. No separate EGL/GLX/WGL.
  - Have fun with swapchains…

Vulkan in Qt

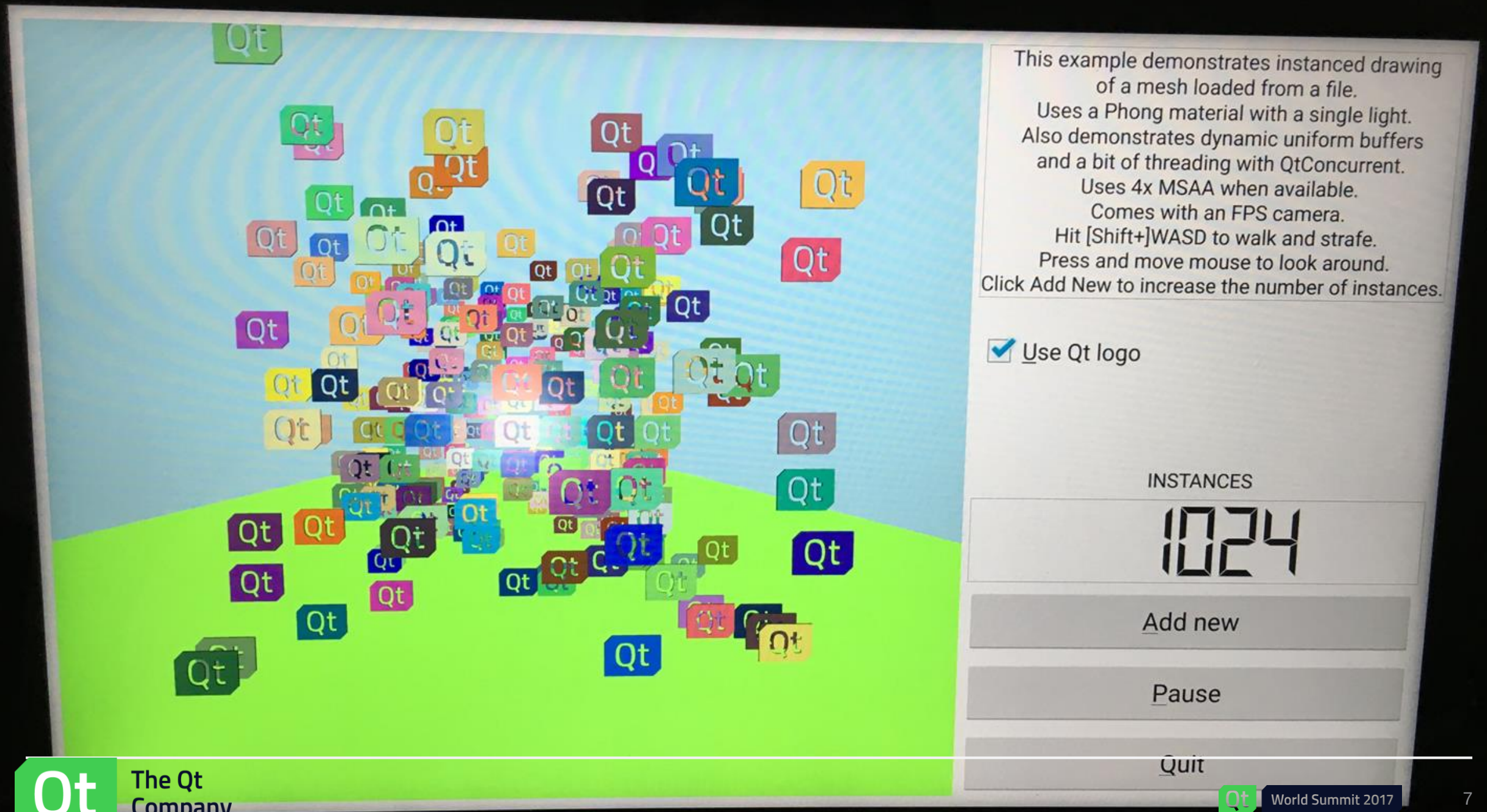
- Qt 5.10 introduces basic Vulkan support.
- Meaning Vulkan-based QWindows are now supported.
  - In the end similar to what GLFW or SDL provides.
- Can be embedded into QWidget UIs as usual. (QWidget::createWindowContainer())
- Enablers for the future.

Platforms

- Windows (desktop)
- Linux (xcb)
  - Others (wayland?) may follow later.
- Android (API level 24+)
  - Or 23 for the NVIDIA Shield Tablet, just copy the Vulkan headers in the NDK from level 24.
- Support in qtbase is enabled when (not too old) Vulkan headers are found during configure.

Platforms

- A Vulkan-capable QtGui is loadable on any system
  - since it does not directly depend on libvulkan or similar.
  - QVulkanInstance loads everything at runtime.
- Plan is to ship *some* of the 5.10 pre-built packages with Vulkan support.


This example demonstrates instanced drawing
of a mesh loaded from a file.
Uses a Phong material with a single light.
Also demonstrates dynamic uniform buffers
and a bit of threading with QtConcurrent.
Uses 4x MSAA when available.
Comes with an FPS camera.
Hit [Shift+]WASD to walk and strafe.
Press and move mouse to look around.
Click Add New to increase the number of instances.
Use Qt logo
INSTANCES
1024
Add new
Pause
Quit

What does it offer?

- Vulkan has platform-specific windowing system interface bits in it.
  - Like getting a VkSurfaceKHR from a native window.
- Perfect fit for a nice and handy abstraction via QPA and the platform plugins.

What does it offer?

```
#if defined(VK_USE_PLATFORM_WIN32_KHR)
    VkWin32SurfaceCreateInfoKHR createInfo;
    createInfo.sType = VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR;
    ...
    err = vkCreateWin32SurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
#elif defined(VK_USE_PLATFORM_XCB_KHR)
    VkXcbSurfaceCreateInfoKHR createInfo;
    createInfo.sType = VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR;
    ...
    err = vkCreateXcbSurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
#elif defined(VK_USE_PLATFORM_ANDROID_KHR)
    VkAndroidSurfaceCreateInfoKHR createInfo;
    ...
    err = vkCreateAndroidSurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
...
```

What does it offer?

```
#if defined(VK_USE_PLATFORM_WIN32_KHR)
    VkWin32SurfaceCreateInfoKHR createInfo;
    createInfo.sType = VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR;
    ...
    err = vkCreateWin32SurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
#elif defined(VK_USE_PLATFORM_XCB_KHR)
    VkXcbSurfaceCreateInfoKHR createInfo;
    createInfo.sType = VK_STRUCTURE_TYPE_XCB_SURFACE_CREATE_INFO_KHR;
    ...
    err = vkCreateXcbSurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
#elif defined(VK_USE_PLATFORM_ANDROID_KHR)
    VkAndroidSurfaceCreateInfoKHR createInfo;
    ...
    err = vkCreateAndroidSurfaceKHR(demo->inst, &createInfo, NULL, &demo->surface);
...
```

What does it offer?

```
QWindow *window;
…
VkSurfaceKHR surface = QVulkanInstance::surfaceForWindow(window);
```

What does it offer?

- Convenience.
  - Hide library loading.
    - No direct linking. Tries avoiding some of the mess we had/have with GL.
    - Gives you a getInstanceProcAddr and the core 1.0 API in a cross-platform manner conformant to section 3.1 of the Vulkan specification.
  - Expose layer and extension lists via the familiar Qt types.
  - Route debug output to qDebug.
  - A straightforward QWindow subclass with a simple, double-buffered swapchain can be helpful.

QVulkanInstance

- Represents a native Vulkan instance.
- Either creates its own or adopts an existing one.
- Layer and extension queries and requests.
- QPA-based
  - QPlatformVulkanInstance, implementation in platform plugins, etc.

QVulkanInstance

- Instance and device level core 1.0 Vulkan API exposed via QVulkan(Device)Functions
  - Use getInstanceProcAddr for the rest.
  - Applications can choose to do something else, like some C++ bindings, or link directly to libvulkan
- More: https://doc-snapshots.qt.io/qt5-5.10/qvulkaninstance.html

QWindow

- No surprises here:
  - There is a new **VulkanSurface** type.
  - Must be associated with a QVulkanInstance: **void setVulkanInstance(QVulkanInstance *inst)**
- Some pitfalls when going the hard route and working directly with QWindow instead of the convenience wrapper (QVulkanWindow).
  - See docs for QVulkanInstance and QVulkanWindow
  - http://blog.qt.io/blog/2017/07/03/vulkan-support-qt-5-10-part-3/

QVulkanWindow

- The optional convenience class.
- Manages a swapchain for you.
- Provides a Q(Open)GLWidget-like experience.
- MSAA, readbacks, special situations (device loss).
- https://doc-snapshots.qt.io/qt5-5.10/qvulkanwindow.html

What about shaders?

- Things are slightly different now. Goodbye runtime GLSL source strings.
  - Although some drivers accept these.
- Standard only mandates SPIR-V. No runtime compilation APIs.
  - Hence those pre-built .spv files in Qt's own Vulkan examples.
  - More complex projects will often want to do more than this
    - e.g. integrate an offline compiler in the build system, or even compile in glslang and friends
- Reflection (dynamically discover shader input/outputs, e.g. uniform blocks) needs additional tools.
- All this is not in scope for Qt in 5.10. However…

Some experiments

- …a fairly complete PoC for integrating glslang and SPIRV-Cross is available at https://github.com/alpqr/qtshaderstack17
- Provides a Qt module called shadertools.

Some experiments

- You then have
  - QSpirvCompiler (wrapping glslang) **to compile GLSL into SPIR-V**,
  - QSpirvShader and QShaderDescription to do **reflection** and to serialize/deserialize results,
    - Typical pipelines are expected to do as much as they can offline, including both compilation and reflection.
  - and an experimental QShaderBaker built on top of these.
    - Explores the idea of standardizing on a **single language**, compiling to SPIR-V, and then generating various GLSL (ES) versions, HLSL and MSL via SPIRV-Cross, baking them all into a single **serializable** entity (QBakedShader).

Interop

- Food for thought: Having Vulkan-based rendering in a Qt Quick UI, or having a Qt Quick scene embedded into a Vulkan-based engine do not strictly require that Qt Quick renders through Vulkan.
- Remember **https://github.com/alpqr/sw_quick_in_qvulkanwindow** ?

Interop

- Today we can do even better: **Vulkan – OpenGL (- D3D) interop**.
  - Initially some vendor-specific attempts (GL_NV_draw_vulkan_image)
  - Yet better: **VK_KHR_external_memory** and **GL_EXT_memory_object** family of extensions.
  - Room for research, no practical experience yet.
- Expected to become more common in the future.
  - As an alternative to expecting 100% support for every graphics API from every Qt module.

Let’s look at some code!

qtbase/examples/vulkan/*
qtbase/tests/manual/qvulkaninstance

Wait, wasn’t the title Vulkan **and friends**?

New graphics APIs

- Now we also have Metal, D3D12, …
- None of them truly available cross-platform
- No single solution that is the no. 1 API on all platforms
- Exciting new opportunities
  - easier threading, better performance, new approaches to shaders
- Hence the start of research in Qt 5.8.
  - Note that Qt Quick is not GPU bound.
    - and often not CPU bound either
  - Potential big gains are around the 3D offering, not the 2D (or 2.5D) UIs.

New graphics APIs

- Let’s try to replace OpenGL in Qt with something else.
- At minimum we then need to consider:
  - Windowing system interface implications, QWindow
  - The Qt Quick scenegraph
  - Various integration points (QOpenGL*, QQuickFBO, …)
  - The case of Qt Quick or custom OpenGL content composited with QWidgets via OpenGL
  - QPainter’s OpenGL backend
  - Qt 3D
  - Qt 3D Studio (with its very own runtime as of today)
  - Texture-based interop between these
  - Some modules are hopeless (Canvas 3D, WebGL streaming, …)
  - All sorts of implications in various places (Graphical Effects, Controls, Location, …)

Status

- Let’s try to replace OpenGL in Qt with something else.
- At minimum we then need to consider:
  - Windowing system interface implications, QWindow **-> Vulkan in 5.10**
  - The Qt Quick scenegraph **-> D3D12 since 5.8**
  - Various integration points (QOpenGL*, QQuickFBO, …)
  - The case of Qt Quick or custom OpenGL content composited with QWidgets via OpenGL
  - QPainter’s OpenGL backend
  - Qt 3D **-> Vulkan research on-going**
  - Qt 3D Studio (with its very own runtime as of today) **-> runtime likely based on Qt 3D in the future**
  - Texture-based interop between these
  - Some modules are hopeless (Canvas 3D, WebGL streaming, …)
  - All sorts of implications in various places (Graphical Effects, Controls, Location, …)

Thank You!

The Qt Company