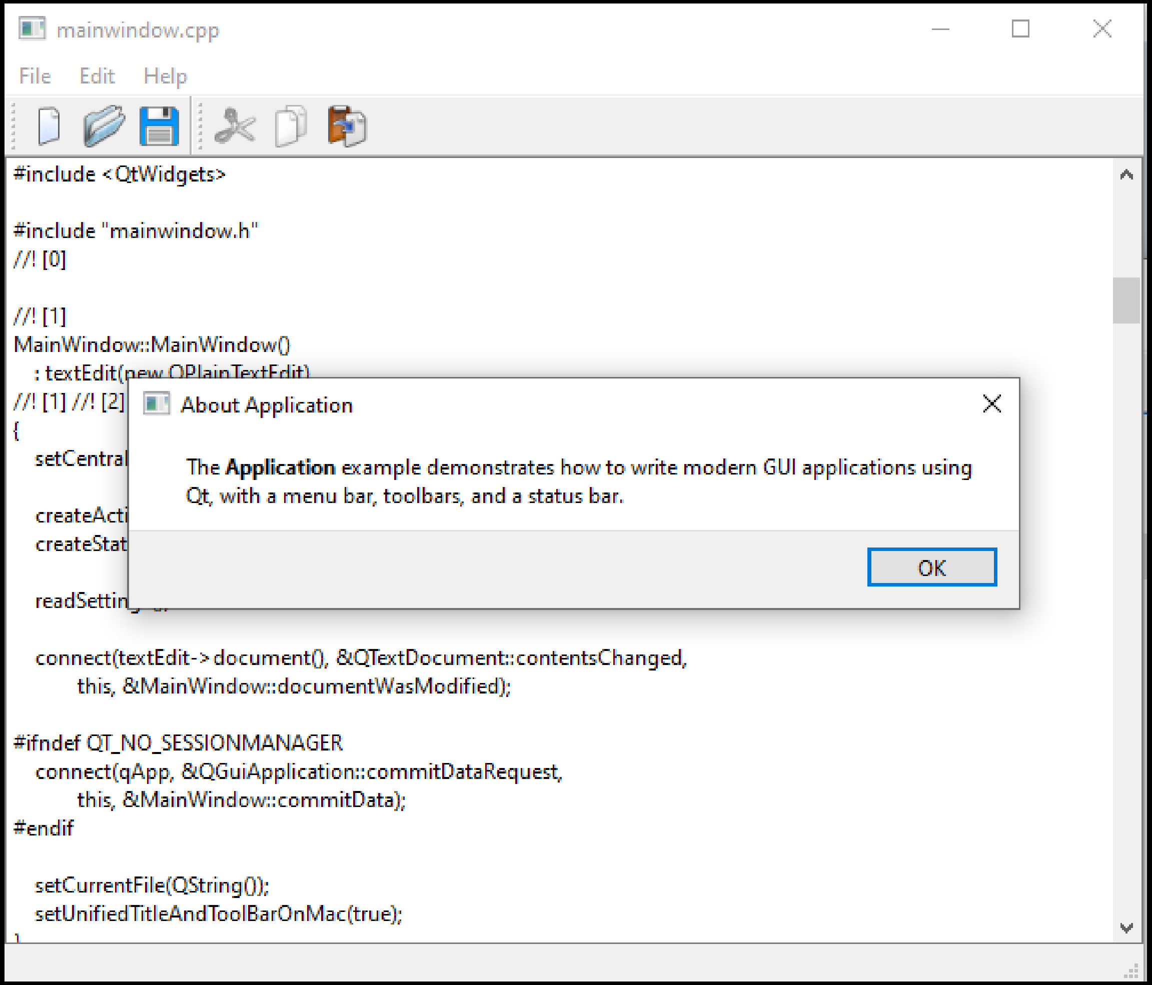
Accelerated Graphics in Qt 6.0

László Agócs
Principal Software Engineer
The Qt Company, Oslo, Norway

Contents

Qt UI Technologies
(some of them)

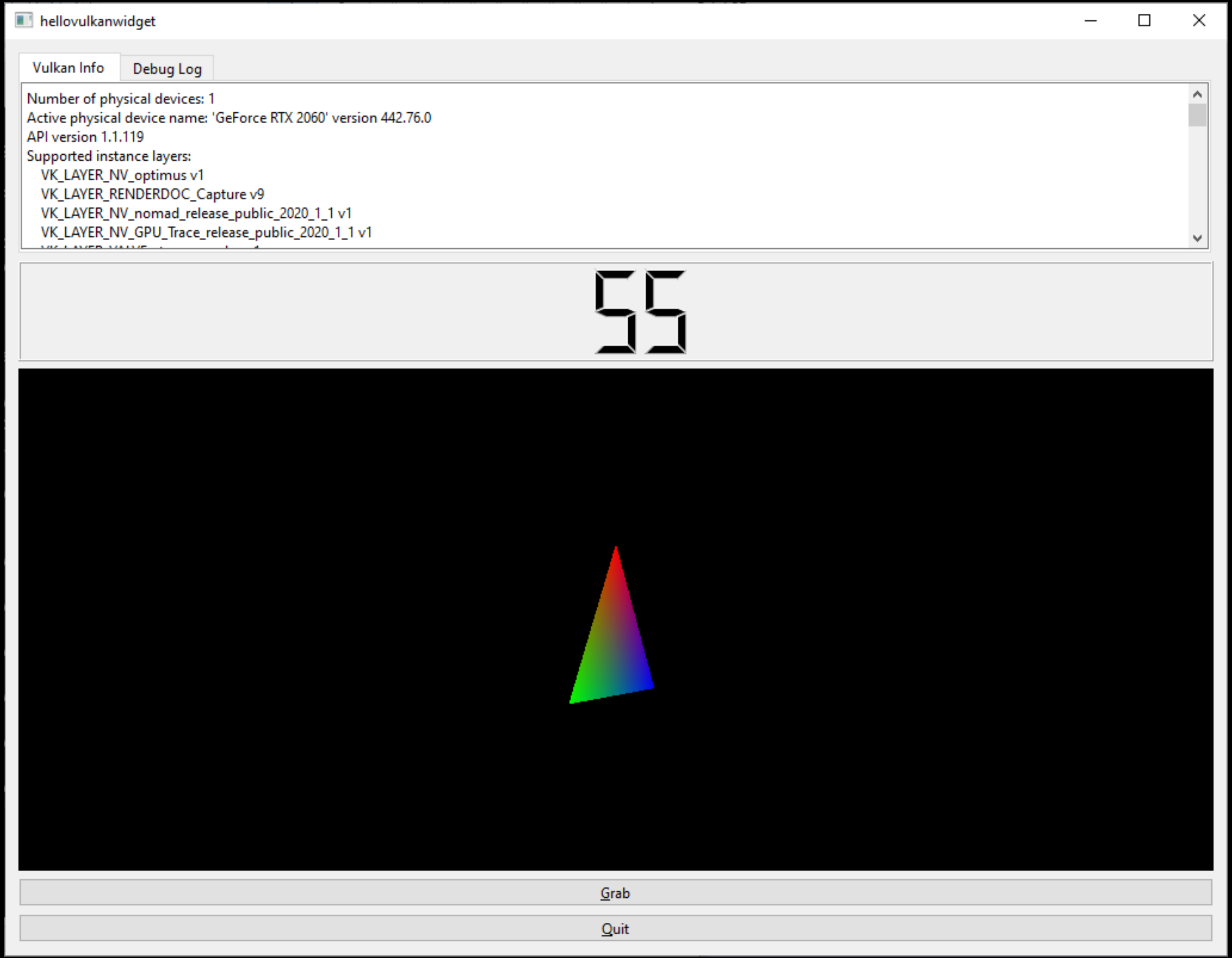
Qt Widgets



```
#include <QtWidgets>

#include "mainwindow.h"
//! [0]

//! [1]
MainWindow::MainWindow()
   : textEdit(new QPlainTextEdit)
//! [1] //! [2]
{
   setCentral
   
   createActi
   createStat

   readSettin

   connect(textEdit->document(), &QTextDocument::contentsChanged,
           this, &MainWindow::documentWasModified);

#ifndef QT_NO_SESSIONMANAGER
   connect(qApp, &QGuiApplication::commitDataRequest,
           this, &MainWindow::commitData);
#endif

   setCurrentFile(QString());
   setUnifiedTitleAndToolBarOnMac(true);
```

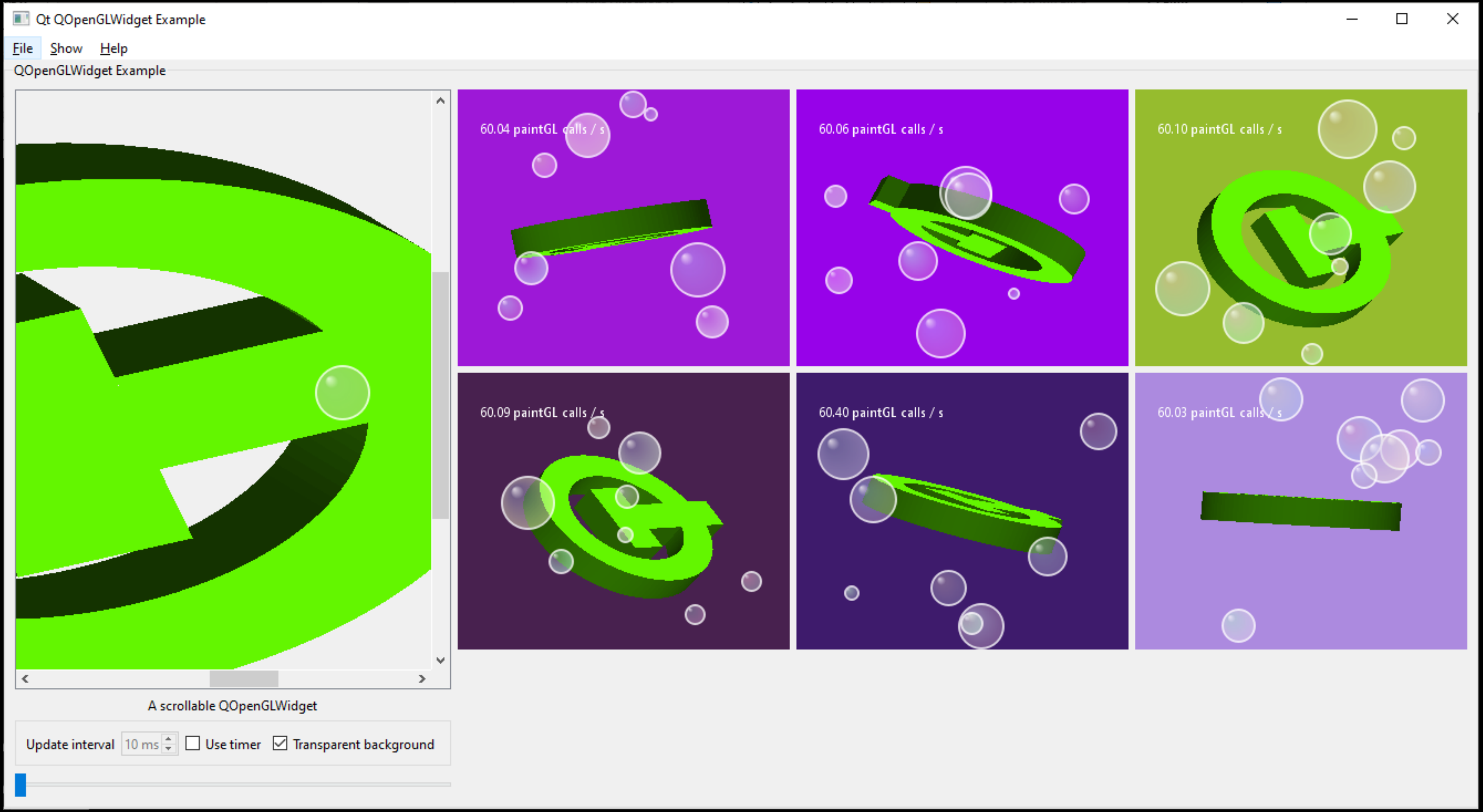
The **Application** example demonstrates how to write modern GUI applications using Qt, with a menu bar, toolbars, and a status bar.

Widgets + native (child) windows



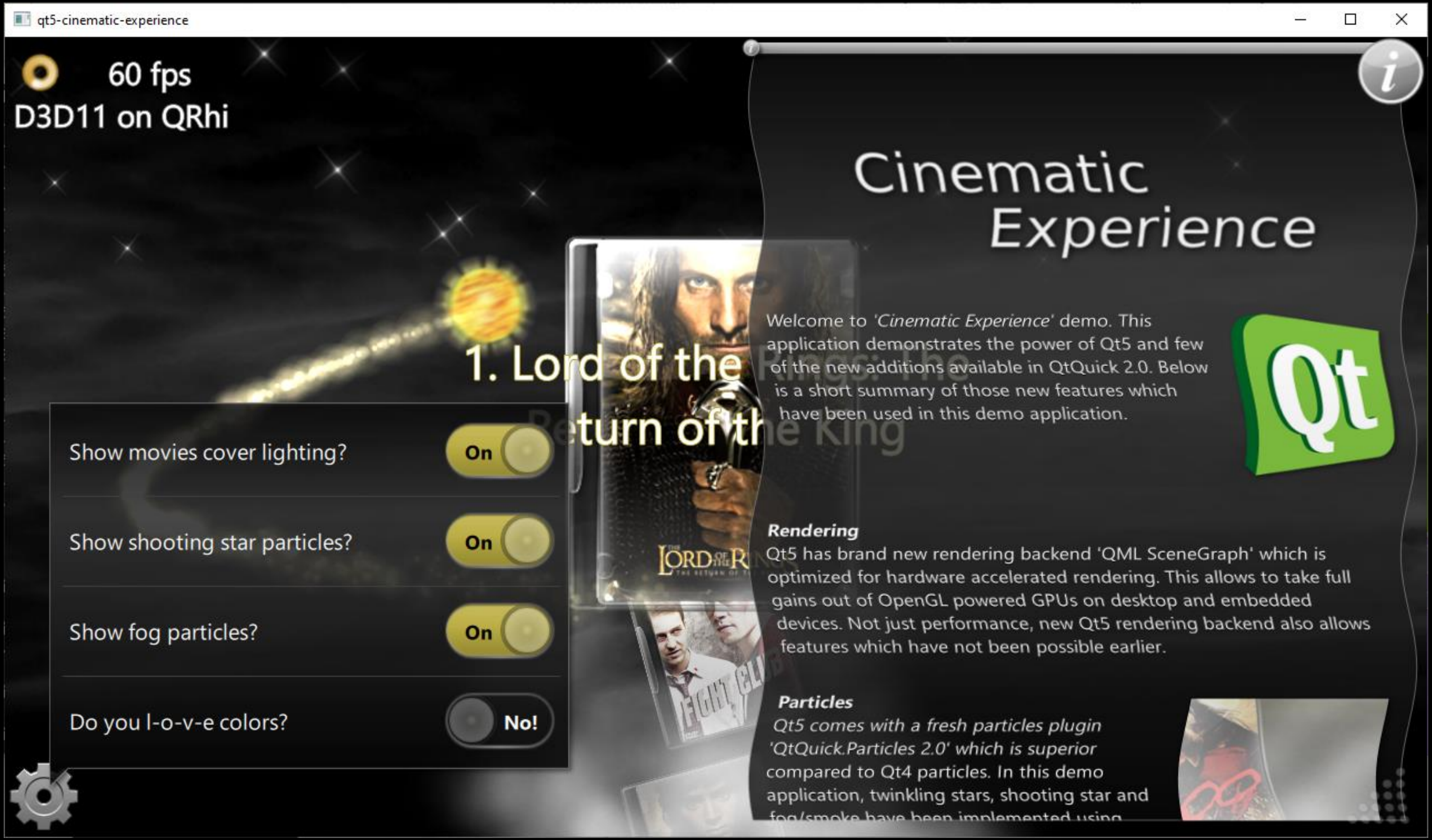
```
Number of physical devices: 1
Active physical device name: 'GeForce RTX 2060' version 442.76.0
API version 1.1.119
Supported instance layers:
   VK_LAYER_NV_optimus v1
   VK_LAYER_RENDERDOC_Capture v9
   VK_LAYER_NV_nomad_release_public_2020_1_1 v1
   VK_LAYER_NV_GPU_Trace_release_public_2020_1_1 v1
```

Widgets + QOpenGLWidget



Update interval 10 ms ☐ Use timer ☑ Transparent background

QML + Qt Quick



Show movies cover lighting? On

Show shooting star particles? On

Show fog particles? On

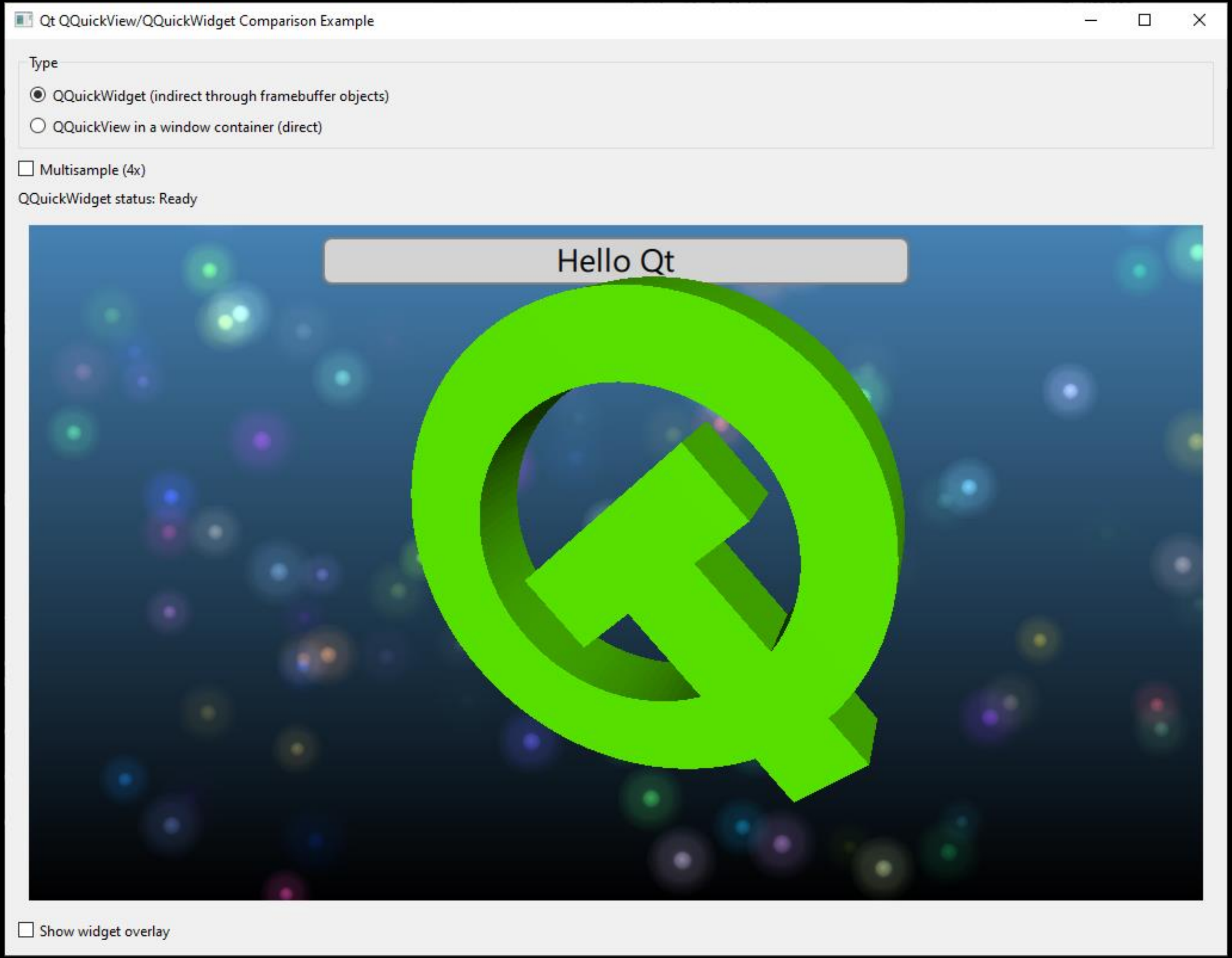
Do you l-o-v-e colors? No!

Welcome to *'Cinematic Experience'* demo. This application demonstrates the power of Qt5 and few of the new additions available in QtQuick 2.0. Below is a short summary of those new features which have been used in this demo application.

Rendering
Qt5 has brand new rendering backend 'QML SceneGraph' which is optimized for hardware accelerated rendering. This allows to take full gains out of OpenGL powered GPUs on desktop and embedded devices. Not just performance, new Qt5 rendering backend also allows features which have not been possible earlier.

Particles
Qt5 comes with a fresh particles plugin 'QtQuick.Particles 2.0' which is superior compared to Qt4 particles. In this demo application, twinkling stars, shooting star and fog/smoke have been implemented using

Widgets + QQuickWidget


Qt QQuickView/QQuickWidget Comparison Example

Type

(●) QQuickWidget (indirect through framebuffer objects)

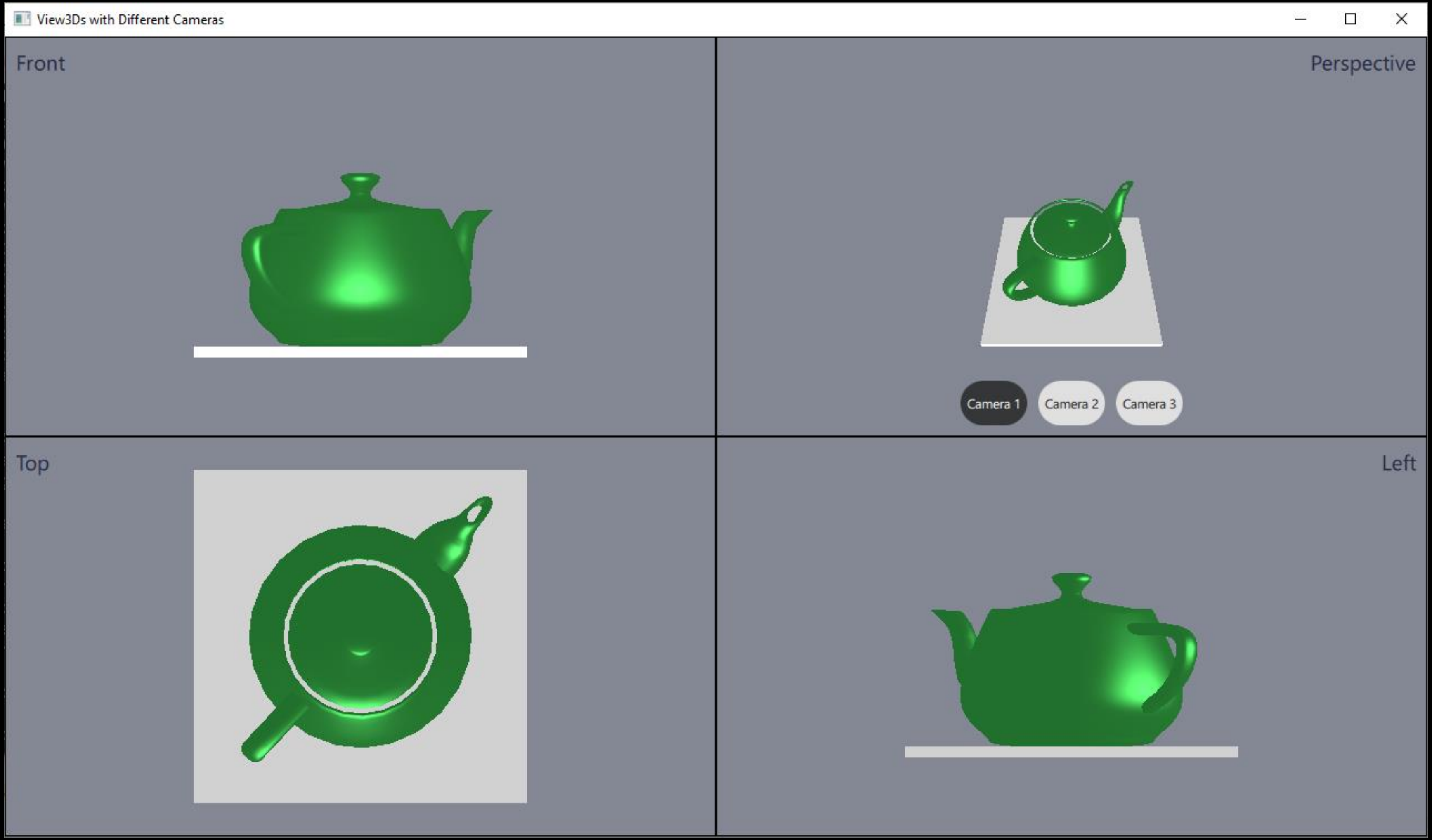
( ) QQuickView in a window container (direct)

☐ Multisample (4x)

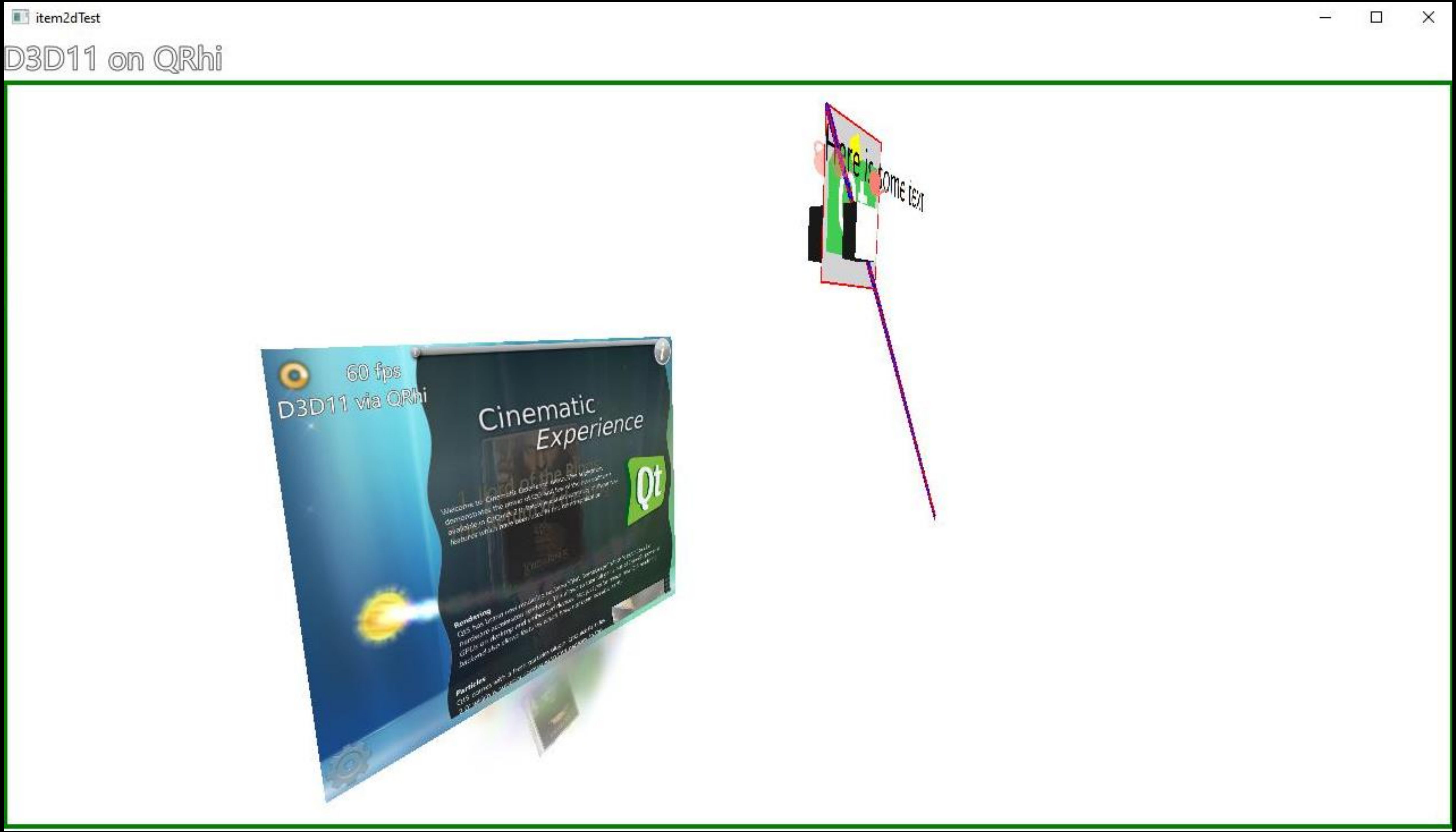
QQuickWidget status: Ready

☐ Show widget overlay

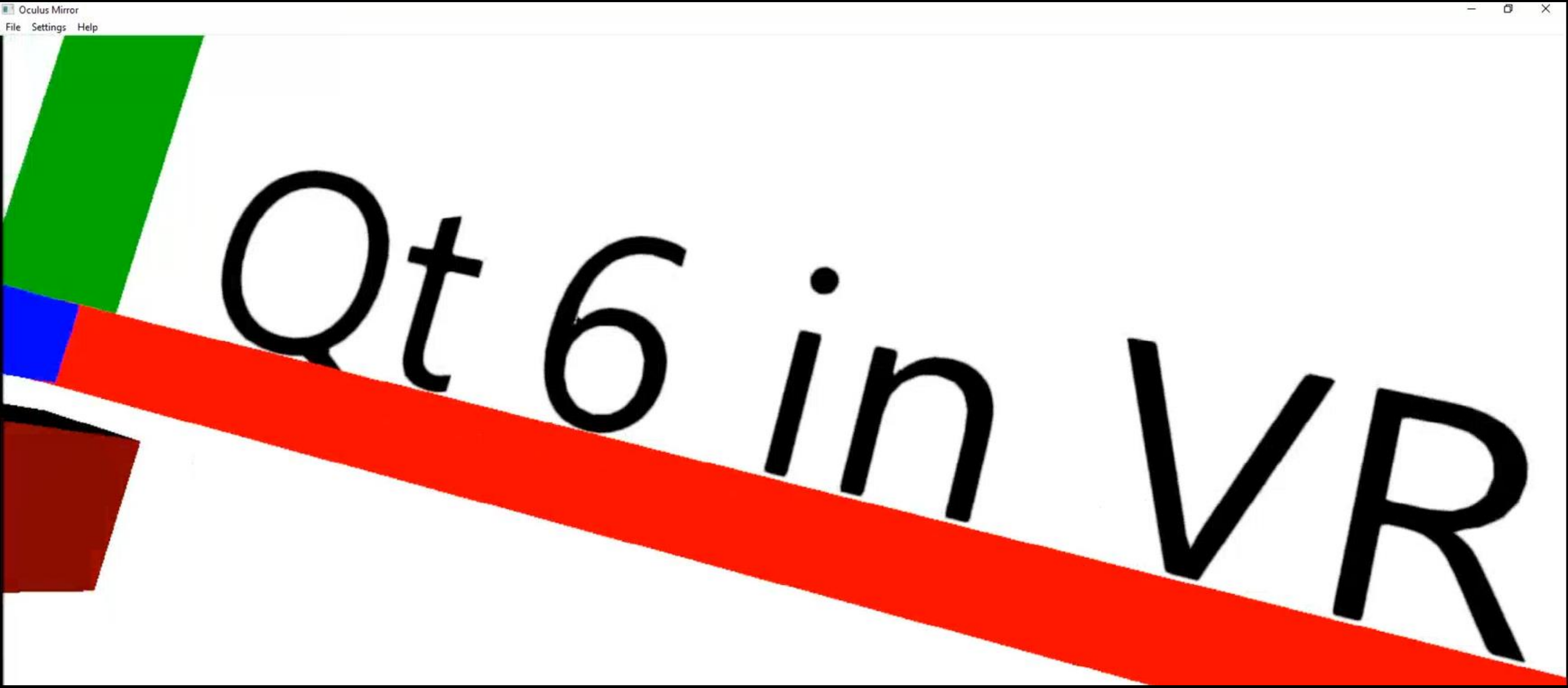
QML + Qt Quick + Qt Quick 3D



QML + Qt Quick + Qt Quick 3D + Qt Quick (no composition, new in Qt 6)



QML + Qt Quick + Qt Quick 3D + Qt Quick in VR (new enablers in Qt 6)



Accelerated 3D APIs in Qt 5 and Qt 6

Qt 5.0 – 5.15

OpenGL, OpenGL ES
WGL/GLX/EGL/drm/…

QWindow
QOpenGLContext
QOpenGLFunctions

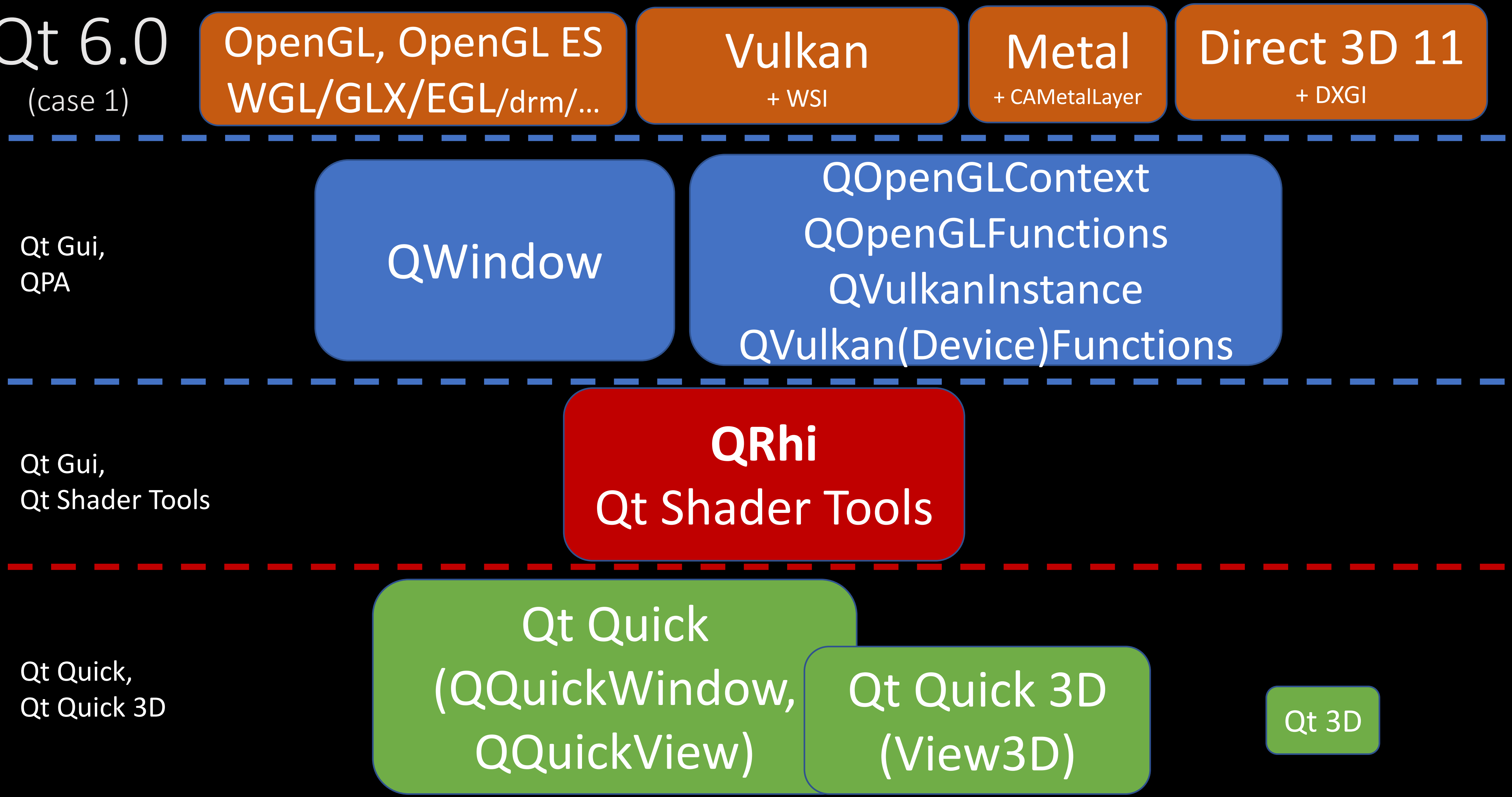
QQuickWindow
QQuickView
QQuickWidget

QOpenGLWidget
QPainter-on-OpenGL

Qt Web Engine / Chromium integration

Qt 3D

Qt 6.0
(case 1)


OpenGL, OpenGL ES
WGL/GLX/EGL/drm/…
Vulkan
+ WSI
Metal
+ CAMetalLayer
Direct 3D 11
+ DXGI
Qt Gui,
QPA
QWindow
QOpenGLContext
QOpenGLFunctions
QVulkanInstance
QVulkan(Device)Functions
Qt Gui,
Qt Shader Tools
QRhi
Qt Shader Tools
Qt Quick,
Qt Quick 3D
Qt Quick
(QQuickWindow,
QQuickView)
Qt Quick 3D
(View3D)
Qt 3D

Qt 6.0

(case 2)

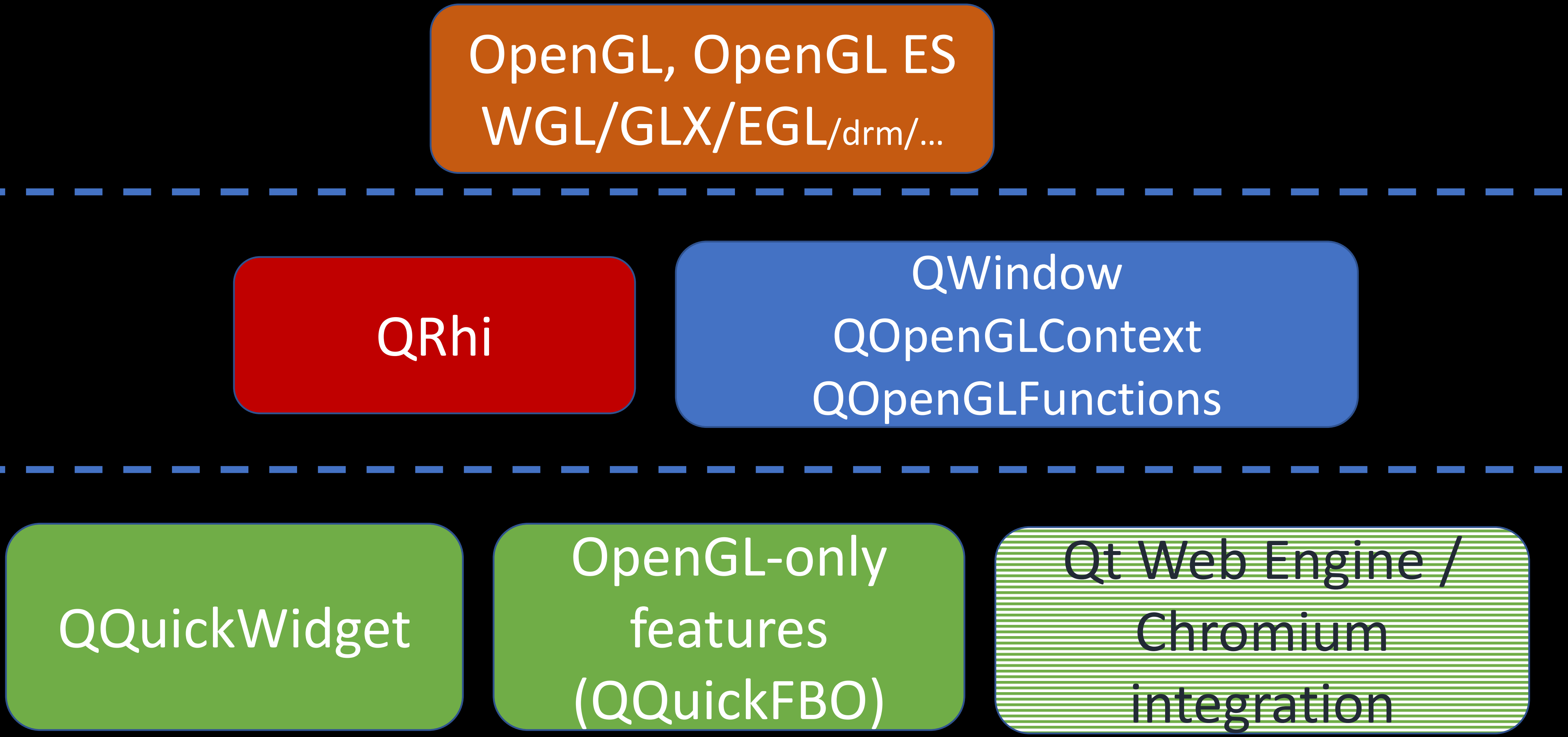
OpenGL, OpenGL ES
WGL/GLX/EGL/drm/…

QWindow
QOpenGLContext
QOpenGLFunctions

QOpenGLWidget
QPainter OpenGL paint engine

Qt 6.0

(case 3)


OpenGL, OpenGL ES
WGL/GLX/EGL/drm/…
QRhi
QWindow
QOpenGLContext
QOpenGLFunctions
QQuickWidget
OpenGL-only features (QQuickFBO)
Qt Web Engine / Chromium integration

Qt Quick in Qt 6.0

- Direct 3D 11.1 on Windows
- Vulkan 1.0+ on Windows, Linux (X11, Wayland), Android
- Metal 1.2+ on macOS, iOS
- OpenGL 2.1+
- OpenGL ES 2.0+
- (software renderer, no changes there compared to 5.15)

Qt Quick 3D in Qt 6.0

- Direct 3D 11.1 on Windows
- Vulkan 1.0+ on Windows, Linux (X11, Wayland), Android
- Metal 1.2+ on macOS, iOS
- OpenGL 3.0+
- OpenGL ES 3.0+
    - ES 2.0 technically but please don’t

Qt Widgets in Qt 6.0

- Same as in Qt 5.15 when it comes to graphics architecture
- Deprecated QGL* (QGLWidget) removed
- Some reorganization (opengl, openglwidgets modules)

Qt Widgets in Qt 6.0

- Early prototype for QPainter on QRhi
  - Will not be in 6.0. No commitment for 6.x.
- Research backing store scaling (high DPI) with QRhi
- QOpenGLWidget and QQuickWidget compositor works like in 5.x
  - directly with OpenGL

OpenGL on Windows in Qt 6.0

- One potentially impactful change for OpenGL-based applications on Windows:

  ANGLE *and ANGLE support* have been removed.

- Qt Quick prefers Direct 3D 11 by default on Windows in Qt 6.0.

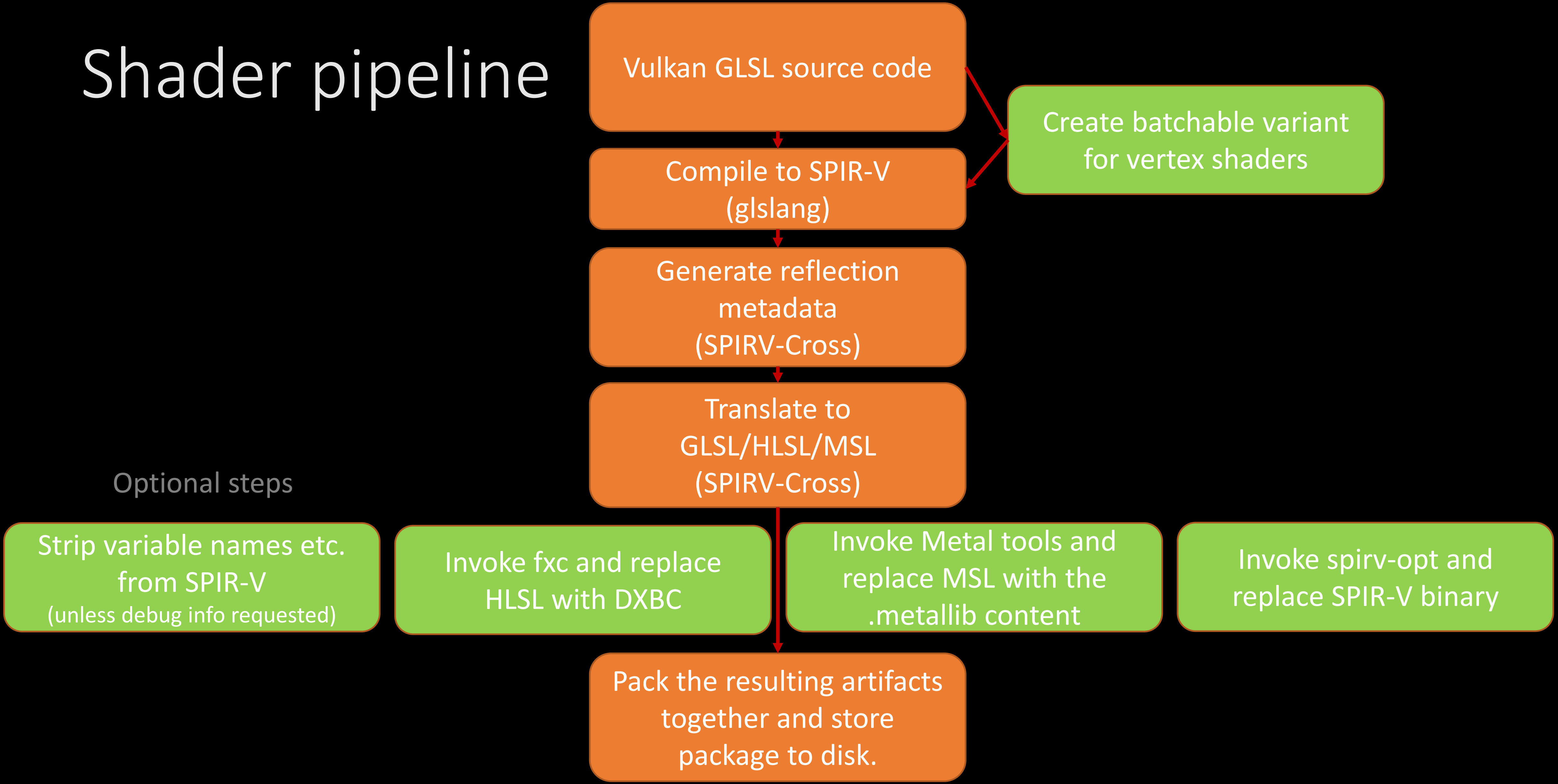
RHI and Shader Pipeline

Qt Rendering Hardware Interface

- QRhi and related classes
  - This is what Qt Quick and Quick 3D uses in Qt 6.0.
  - No direct OpenGL calls and QOpenGLContext.
- Private API in Qt 6.0.
  - To be decided if/when/how this changes in 6.x and beyond.

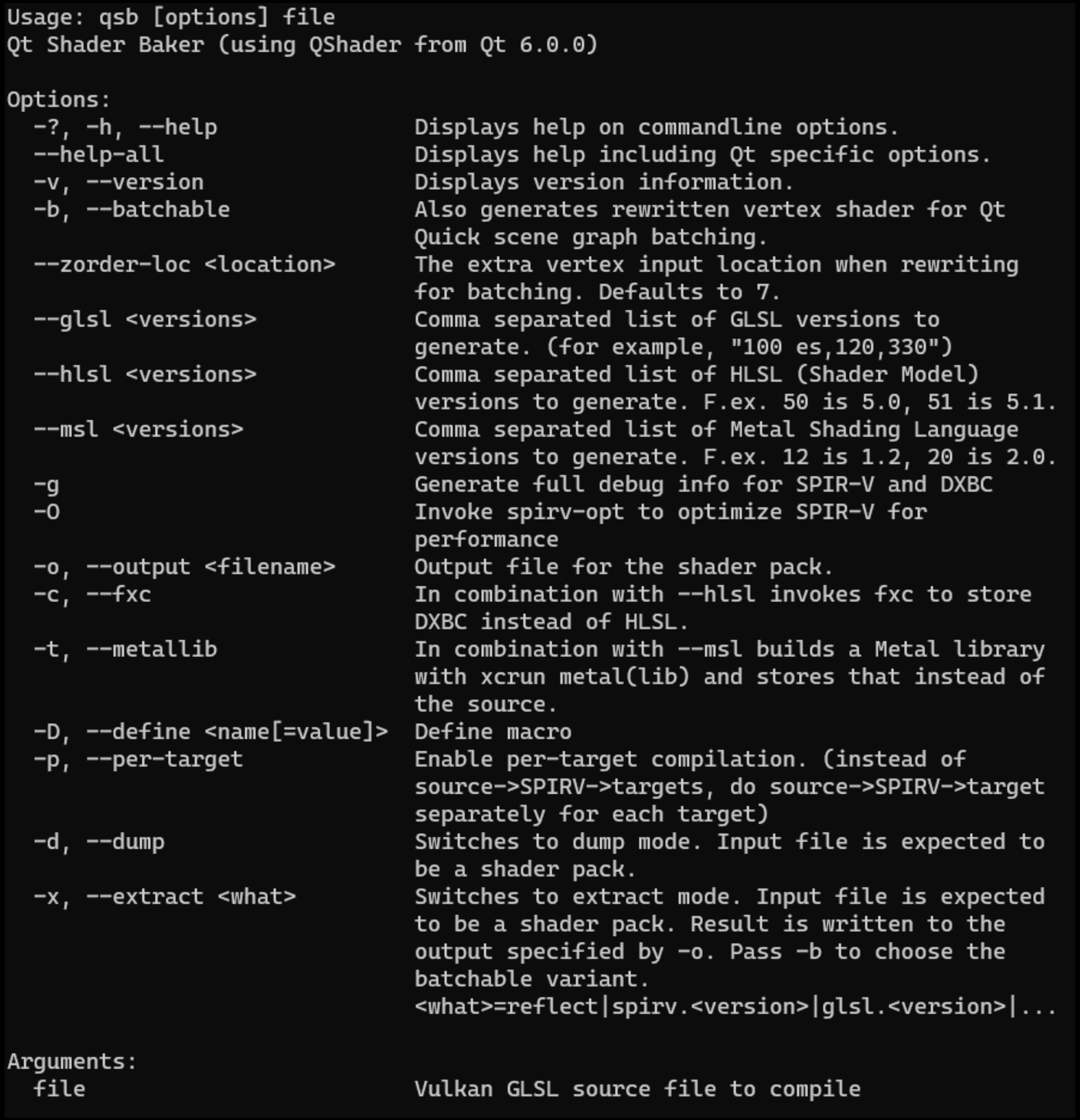
```cpp
class Q_GUI_EXPORT QRhiCommandBuffer : public QRhiResource
{
public:
    enum IndexFormat {
        IndexUInt16,
        IndexUInt32
    };

    QRhiResource::Type resourceType() const override;

    void resourceUpdate(QRhiResourceUpdateBatch *resourceUpdates);

    void beginPass(QRhiRenderTarget *rt,
                   const QColor &colorClearValue,
                   const QRhiDepthStencilClearValue &depthStencilClearValue,
                   QRhiResourceUpdateBatch *resourceUpdates = nullptr);
    void endPass(QRhiResourceUpdateBatch *resourceUpdates = nullptr);

    void setGraphicsPipeline(QRhiGraphicsPipeline *ps);
    using DynamicOffset = QPair<int, quint32>;
    void setShaderResources(QRhiShaderResourceBindings *srb = nullptr,
                            int dynamicOffsetCount = 0,
                            const DynamicOffset *dynamicOffsets = nullptr);
    using VertexInput = QPair<QRhiBuffer *, quint32>;
    void setVertexInput(int startBinding, int bindingCount, const VertexInput *bindings,
                        QRhiBuffer *indexBuf = nullptr, quint32 indexOffset = 0,
                        IndexFormat indexFormat = IndexUInt16);

    void setViewport(const QRhiViewport &viewport);
    void setScissor(const QRhiScissor &scissor);
    void setBlendConstants(const QColor &c);
    void setStencilRef(quint32 refValue);

    void draw(quint32 vertexCount,
              quint32 instanceCount = 1,
```

Shader pipeline

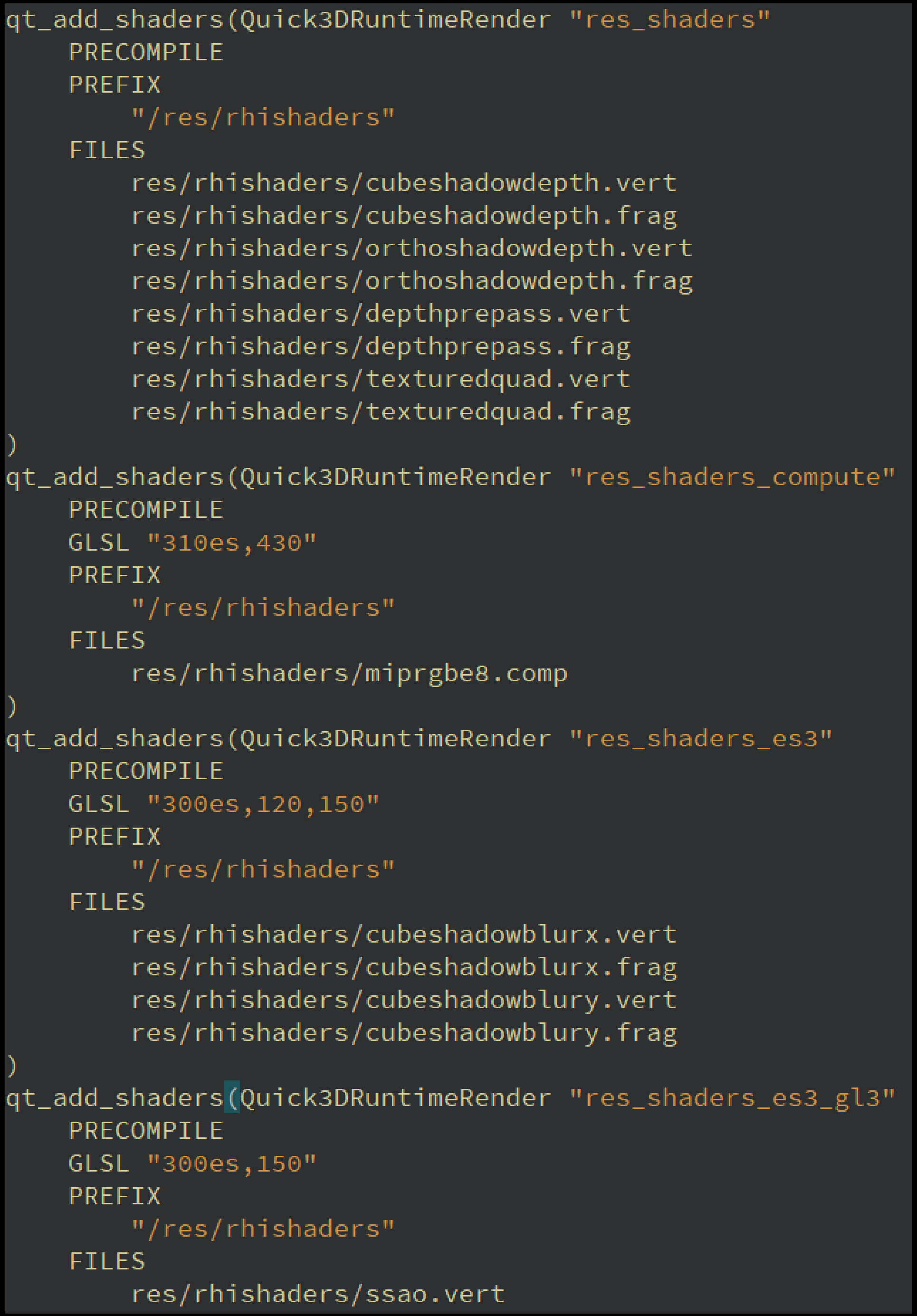

Vulkan GLSL source code
Create batchable variant for vertex shaders
Compile to SPIR-V (glslang)
Generate reflection metadata (SPIRV-Cross)
Translate to GLSL/HLSL/MSL (SPIRV-Cross)
Optional steps
Strip variable names etc. from SPIR-V (unless debug info requested)
Invoke fxc and replace HLSL with DXBC
Invoke Metal tools and replace MSL with the .metallib content
Invoke spirv-opt and replace SPIR-V binary
Pack the resulting artifacts together and store package to disk.

Shader pipeline

- No more GLSL 100 es, 120, etc. sprinkled all over the place
- Everything is Vulkan-compatible GLSL
- Reflecting and translating should happen offline or at build time
- Qt Quick enforces this
  - ShaderEffect and QSGMaterial only works with .qsb files
- Qt Quick 3D does not
  - Some scenes will have the option to do it offline / build time
  - Others will still do it at run time



```
Usage: qsb [options] file
Qt Shader Baker (using QShader from Qt 6.0.0)

Options:
  -?, -h, --help               Displays help on commandline options.
  --help-all                   Displays help including Qt specific options.
  -v, --version                Displays version information.
  -b, --batchable              Also generates rewritten vertex shader for Qt
                               Quick scene graph batching.
  --zorder-loc <location>      The extra vertex input location when rewriting
                               for batching. Defaults to 7.
  --glsl <versions>            Comma separated list of GLSL versions to
                               generate. (for example, "100 es,120,330")
  --hlsl <versions>            Comma separated list of HLSL (Shader Model)
                               versions to generate. F.ex. 50 is 5.0, 51 is 5.1.
  --msl <versions>             Comma separated list of Metal Shading Language
                               versions to generate. F.ex. 12 is 1.2, 20 is 2.0.
  -g                           Generate full debug info for SPIR-V and DXBC
  -O                           Invoke spirv-opt to optimize SPIR-V for
                               performance
  -o, --output <filename>      Output file for the shader pack.
  -c, --fxc                    In combination with --hlsl invokes fxc to store
                               DXBC instead of HLSL.
  -t, --metallib               In combination with --msl builds a Metal library
                               with xcrun metal(lib) and stores that instead of
                               the source.
  -D, --define <name[=value]>  Define macro
  -p, --per-target             Enable per-target compilation. (instead of
                               source->SPIRV->targets, do source->SPIRV->target
                               separately for each target)
  -d, --dump                   Switches to dump mode. Input file is expected to
                               be a shader pack.
  -x, --extract <what>         Switches to extract mode. Input file is expected
                               to be a shader pack. Result is written to the
                               output specified by -o. Pass -b to choose the
                               batchable variant.
                               <what>=reflect|spirv.<version>|glsl.<version>|...

Arguments:
  file                         Vulkan GLSL source file to compile
```



```
qt_add_shaders(Quick3DRuntimeRender "res_shaders"
    PRECOMPILE
    PREFIX
        "/res/rhishaders"
    FILES
        res/rhishaders/cubeshadowdepth.vert
        res/rhishaders/cubeshadowdepth.frag
        res/rhishaders/orthoshadowdepth.vert
        res/rhishaders/orthoshadowdepth.frag
        res/rhishaders/depthprepass.vert
        res/rhishaders/depthprepass.frag
        res/rhishaders/texturedquad.vert
        res/rhishaders/texturedquad.frag
)
qt_add_shaders(Quick3DRuntimeRender "res_shaders_compute"
    PRECOMPILE
    GLSL "310es,430"
    PREFIX
        "/res/rhishaders"
    FILES
        res/rhishaders/miprgbe8.comp
)
qt_add_shaders(Quick3DRuntimeRender "res_shaders_es3"
    PRECOMPILE
    GLSL "300es,120,150"
    PREFIX
        "/res/rhishaders"
    FILES
        res/rhishaders/cubeshadowblurx.vert
        res/rhishaders/cubeshadowblurx.frag
        res/rhishaders/cubeshadowblury.vert
        res/rhishaders/cubeshadowblury.frag
)
qt_add_shaders(Quick3DRuntimeRender "res_shaders_es3_gl3"
    PRECOMPILE
    GLSL "300es,150"
    PREFIX
        "/res/rhishaders"
    FILES
        res/rhishaders/ssao.vert
```

Some relevant API changes

Qt Quick: materials and textures

- QSGMaterialShader changes (a lot, but conceptually the same)
  - custom materials need minor porting work
- QSGTexture interface changes to some degree
  - relevant when working with materials, or when subclassing (rare)
- New ways to access and adopt native texture resources
  - QSGTexture::textureId() and QQuickWindow::createTextureFromId() have new alternatives

Qt Quick: redirecting

- QQuickRenderControl API extended
- QQuickWindow: new *render target* and *graphics device* concept
  - OpenGL-isms removed (openglContextCreated, setRenderTarget(GLuint fbo))
  - QQuickRenderTarget, QQuickGraphicsDevice, QQuickGraphicsConfiguration

Qt Quick: redirecting

- We can render Qt Quick content with an external graphics context/device, targeting an external texture, with all the supported graphics APIs.
- Proof: Qt Quick 3D in VR with OpenXR on D3D/Vulkan/OpenGL

```
QVector<XrSwapchainImageBaseHeader*> OpenXRGraphicsVulkan::allocateSwapchainImages(int count, XrSwapchain swapch
{
    QVector<XrSwapchainImageBaseHeader*> swapchainImages;
    QVector<XrSwapchainImageVulkanKHR> swapchainImageBuffer(count);
    for (XrSwapchainImageVulkanKHR& image : swapchainImageBuffer) {
        image.type = XR_TYPE_SWAPCHAIN_IMAGE_VULKAN_KHR;
        swapchainImages.push_back(reinterpret_cast<XrSwapchainImageBaseHeader*>(&image));
    }
    m_swapchainImageBuffer.insert(swapchain, swapchainImageBuffer);
    return swapchainImages;
}


QQuickRenderTarget OpenXRGraphicsVulkan::renderTarget(const XrCompositionLayerProjectionView &layerView, const X
{
    Q_UNUSED(swapchainFormat)
    VkImage colorTexture = reinterpret_cast<const XrSwapchainImageVulkanKHR*>(swapchainImage)->image;

    return QQuickRenderTarget::fromNativeTexture({ quint64(colorTexture), VK_IMAGE_LAYOUT_UNDEFINED},   !!
                                                 QSize(layerView.subImage.imageRect.extent.width,
                                                       layerView.subImage.imageRect.extent.height));
}
```

that's QQuickRenderTarget::fromVulkanImage(colorTexture, ...) in 6.0

```
void OpenXRGraphicsVulkan::setupWindow(QQuickWindow *quickWindow)
{
    quickWindow->setGraphicsDevice(QQuickGraphicsDevice::fromPhysicalDevice(m_vulkanPhysicalDevice));   !!
    quickWindow->setGraphicsConfiguration(m_graphicsConfiguration);
    quickWindow->setVulkanInstance(&m_vulkanInstance);
```

```
void OpenXRManager::doRender(const XrCompositionLayerProjectionView &layerView, const XrSwapchainImageBaseHeader *swa
{
    m_quickWindow->setRenderTarget(m_graphics->renderTarget(layerView, swapchainImage, swapchainFormat)); !!

    m_quickWindow->setGeometry(0, 0, layerView.subImage.imageRect.extent.width, layerView.subImage.imageRect.extent.h
    m_quickWindow->contentItem()->setSize(QSizeF(layerView.subImage.imageRect.extent.width, layerView.subImage.imageR

    m_renderControl->polishItems();
    m_renderControl->beginFrame();
    m_renderControl->sync();          !!
    m_renderControl->render();
    m_renderControl->endFrame();

}

void OpenXRManager::setupQuickScene()
{
    m_renderControl = new QQuickRenderControl;
    m_quickWindow = new QQuickWindow(m_renderControl);          !!
    m_graphics->setupWindow(m_quickWindow);
    m_animationDriver = new OpenXRAnimationDriver;
    m_animationDriver->install();

    const bool initSuccess = m_renderControl->initialize();
```

Qt Quick: integrating native rendering

- When integrating your own OpenGL/Vulkan/Metal/D3D rendering:
  - setClearBeforeRendering() is gone
  - beforeRenderPassRecording(), afterRenderPassRecording() signals
  - beforeFrameBegin(), afterFrameEnd() signals
  - beginExternalCommands(), endExternalCommands(), graphicsStateInfo()
  - QSGRendererInterface::getResource()

Qt Quick: changing the RHI backend

- setGraphicsApi()

```
QQuickWindow::setGraphicsApi(QSGRendererInterface::OpenGLRhi);
```

- Environment variables:
  - QSG_RHI_BACKEND={d3d11,vulkan,metal,opengl}
  - QSG_RHI_DEBUG_LAYER=1 for validation / debug layer on Vulkan / D3D11
  - QSG_RHI_PROFILE=1 to get debug markers in frame captures (RenderDoc) with D3D and Vulkan
  - as always, set QSG_INFO=1 to see what's going on at startup (which API, which render loop, etc.)

Thank you!