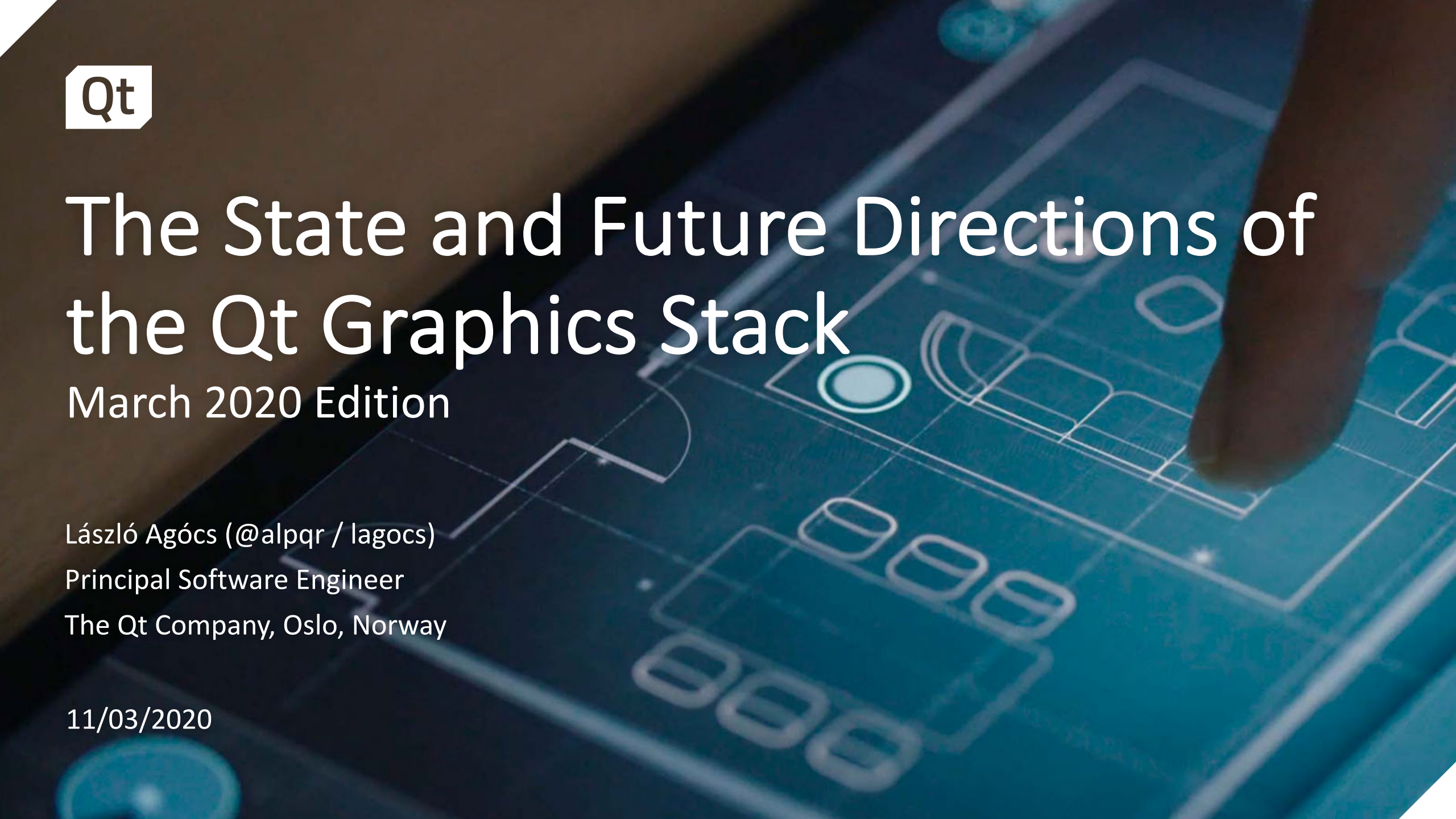
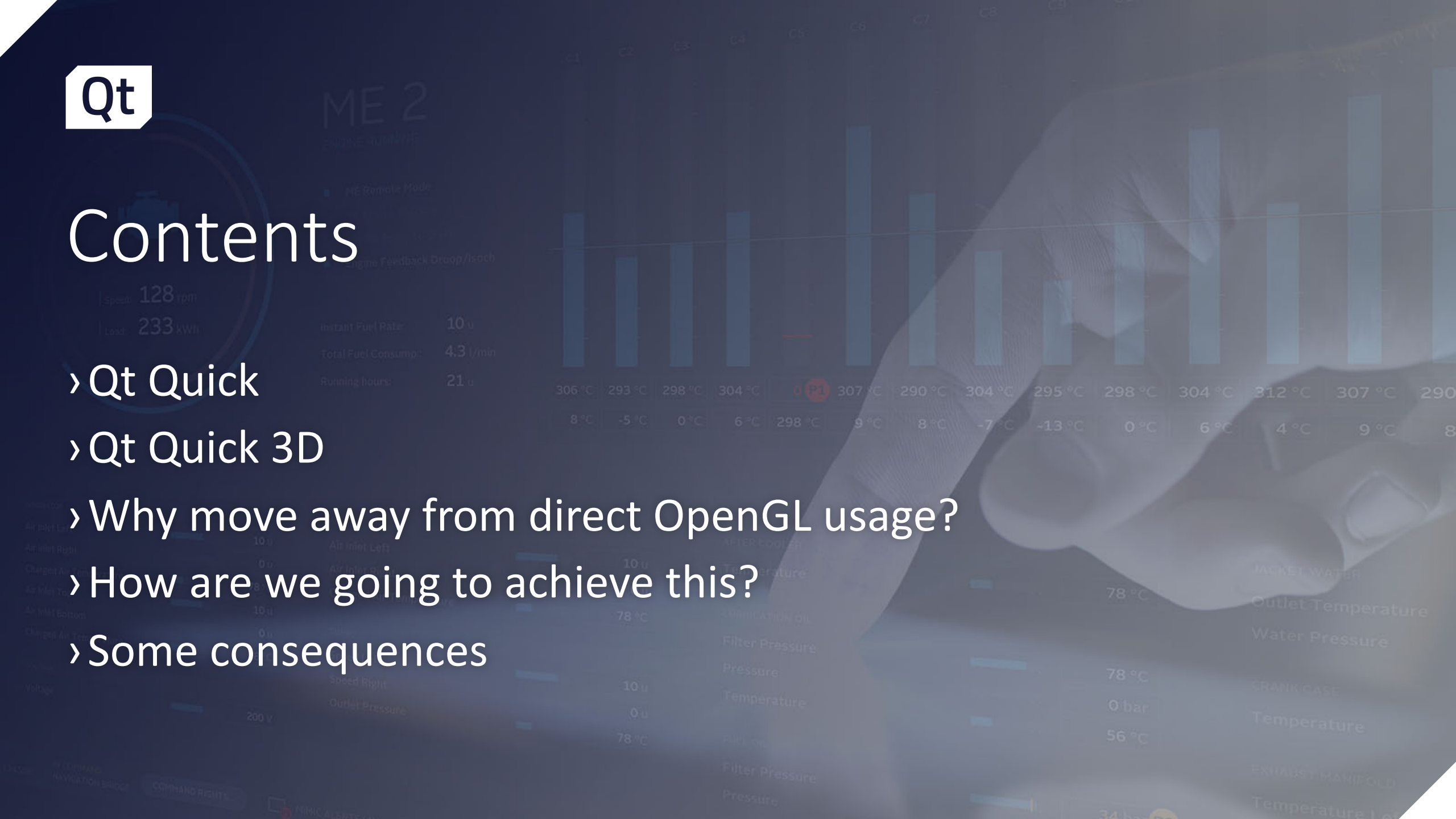


The State and Future Directions of the Qt Graphics Stack

March 2020 Edition

László Agócs (@alpqr / lagocs)
Principal Software Engineer
The Qt Company, Oslo, Norway

11/03/2020



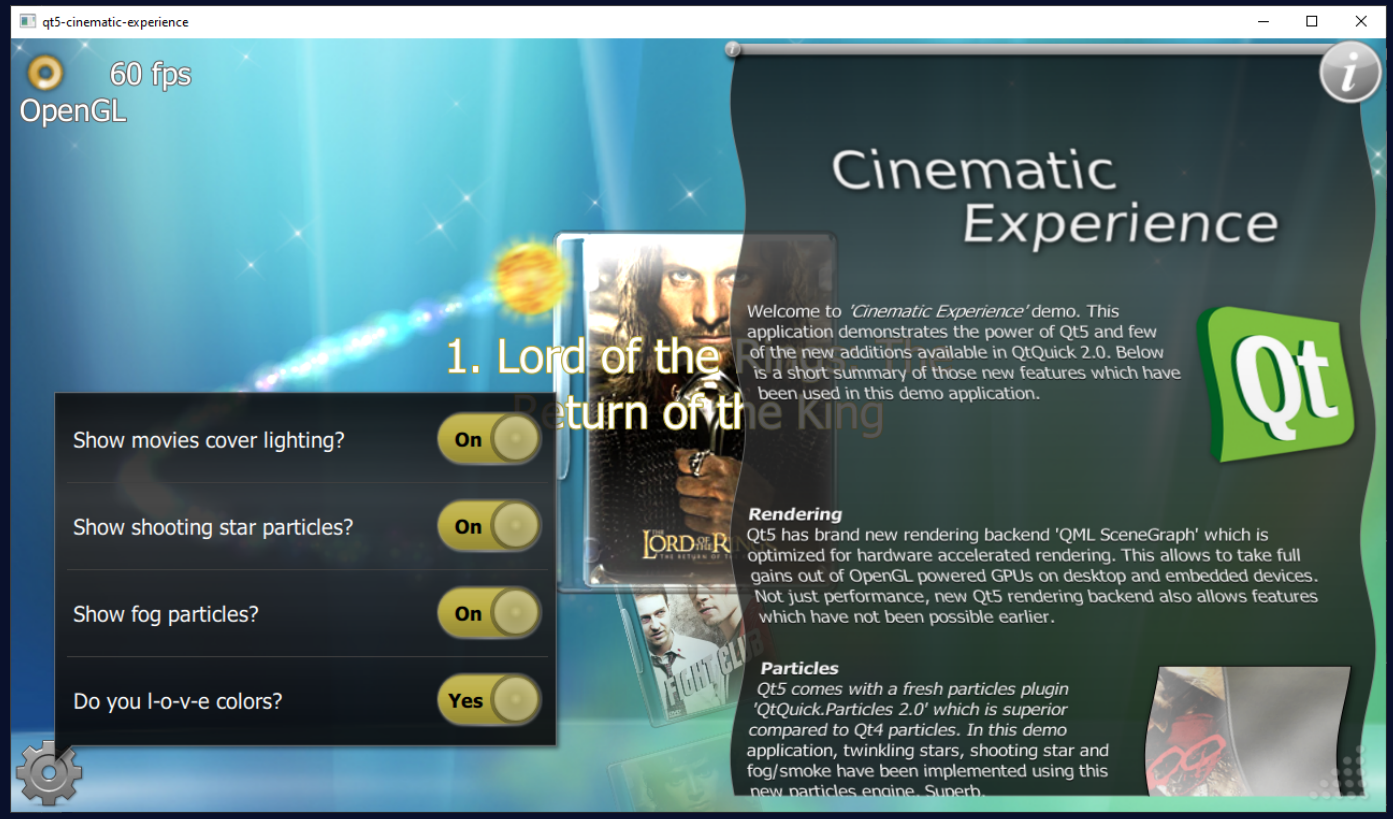
Contents

- Qt Quick
- Qt Quick 3D
- Why move away from direct OpenGL usage?
- How are we going to achieve this?
- Some consequences

Qt Quick

- QML types for creating 2D/2.5D user interfaces
- Backed by a scene graph based rendering engine
  - Uses OpenGL (ES) and GLSL shaders.
    - A limited, pure software rendering solution exists too; not in focus for us here – it is expected to stay as-is in Qt 6.

```
import QtQuick 2.0

Rectangle {
    Rectangle {
        width: 100
        height: 100
        anchors.centerIn: parent
        color: "red"
        NumberAnimation on rotation {
            from: 0; to: 360; duration: 2000; loops: Animation.Infinite;
        }
    }

    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
}
```

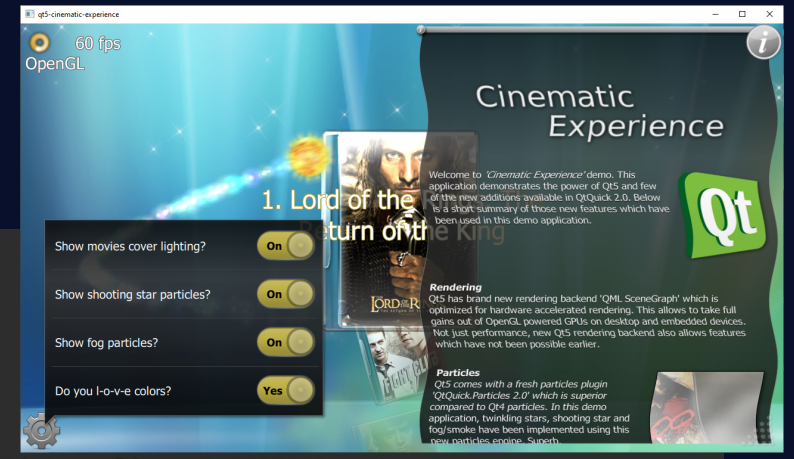


Qt Quick

- QQuickItem tree
  - described in QML
  - main thread
- QSGNode tree + material system
  - the “scene graph”
  - render thread
- OpenGL

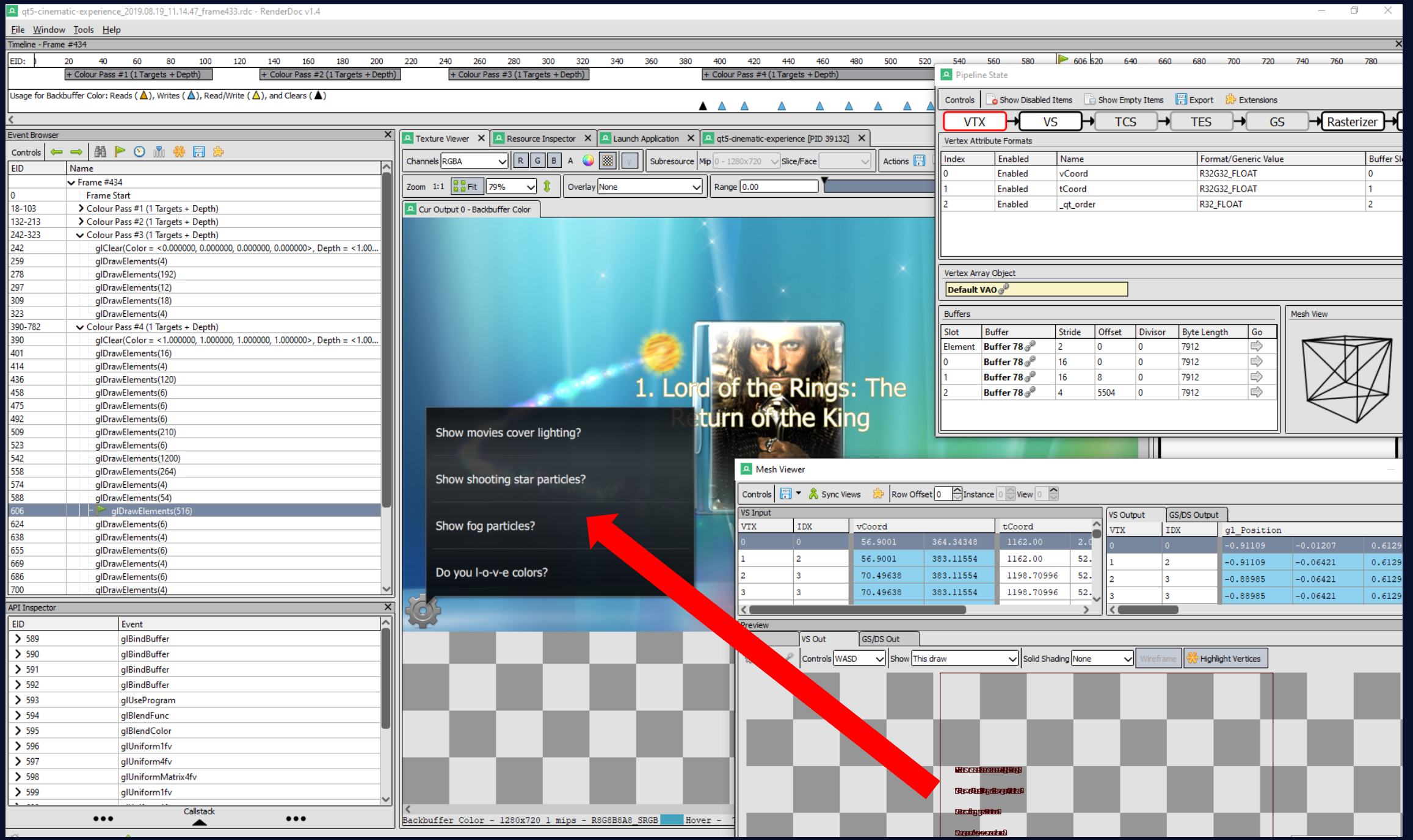
Qt

Batching, atlasing, ...

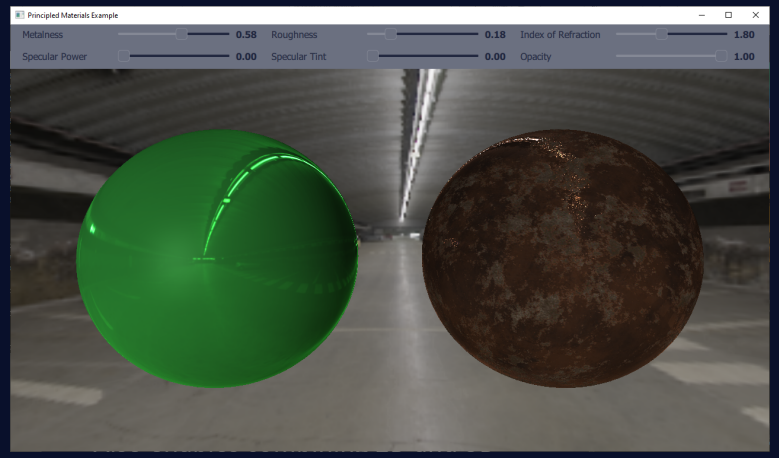


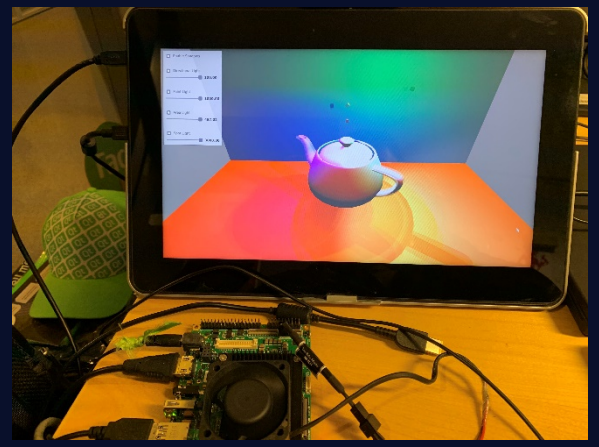

```
[ ] RootNode 0x7fe48800f310  "" ) 1
 [ ] TransformNode( 0x7fe48800f390 identity "QQuickItem(QQuickRootItem:)"  ) 0
  [ ] TransformNode( 0x7fe488015fc0 identity "QQuickItem(QQuickItem:)"  ) 0
   [ ] TransformNode( 0x7fe488016220 identity "QQuickItem(MainView_QMLTYPE_8:)"  ) 0
    [ ] TransformNode( 0x7fe48808f450 identity "QQuickItem(QQuickItem:)"  ) 0
     [ ] TransformNode( 0x7fe48808f620 identity "QQuickItem(QQuickItem:)"  ) 0
      [ ] TransformNode( 0x7fe4881b70e0 identity "QQuickItem(QQuickImage_QML_2:)"  ) 0
       [ ] GeometryNode( 0x7fe4881b7c20 strip #V: 4 #I: 0 x1= 0 y1= 0 x2= 1280 y2= 720 materialtype= 0x7fe4c166ad32 ) "internalimage" 1000001 order 0
      [ ] TransformNode( 0x7fe4881b7290 identity "QQuickItem(QQuickParticleSystem:)"  ) 0
       [ ] TransformNode( 0x7fe4881b7480 identity "QQuickItem(QQuickImageParticle:)"  ) 0
       [ ] TransformNode( 0x7fe4881b7620 identity "QQuickItem(QQuickParticleEmitter:)"  ) 0
      [ ] TransformNode( 0x7fe48808f730 identity "QQuickItem(QQuickListView_QML_9:)"  ) 0
       [ ] TransformNode( 0x7fe4881ad830 translate 0 220 0 "QQuickItem(QQuickItem:)"  ) 0
        [ ] TransformNode( 0x7fe4881ad910 det= 0.444444 "QQuickItem(DelegateItem_QMLTYPE_0:)"  ) 0
         [ ] OpacityNode( 0x7fe4881ada10 opacity= 0.5 combined= 1  "" ) 1
          [ ] TransformNode( 0x7fe4881adf20 identity "QQuickItem(QQuickMouseArea:)"  ) 0
          [ ] TransformNode( 0x7fe4881ae0b0 translate 512 0 0 "QQuickItem(QQuickImage:)"  ) 0
           [ ] RootNode 0x7fe4881b0b90  "" ) 1
            [ ] GeometryNode( 0x7fe4881b1de0 strip #V: 4 #I: 0 x1= 0 y1= 0 x2= 256 y2= 256 materialtype= 0x7fe4c166ad32 ) "internalimage" 1000001 order 0
[ ] TransformNode( 0x7fe4881adaf0 det= 0.444444 "QQuickItem(DelegateItem_QMLTYPE_0:)"  ) 0
acity= 0.7  combined= 1  "" ) 1
 identity  QQuickItem(QQuickMouseArea:)"  ) 0
 translate 512 0 0 "QQuickItem(QQuickImage:)"  ) 0
) 1
0 strip #V: 4 #I: 0 x1= 0 y1= 0 x2= 256 y2= 256 materialtype= 0x7fe4c166ad32 ) "internalimage" 1000001 order 0
entity "QQuickItem(QQuickItem:)"  ) 0
et= 0.0948417 "QQuickItem(DelegateItem_QMLTYPE_0:)"  ) 0
acity= 0.825694 combined= 1  "" ) 1
 identity "QQuickItem(QQuickMouseArea:)"  ) 0
 translate 512 0 0 "QQuickItem(QQuickImage:)"  ) 0
) 1
0 strip #V: 4 #I: 0 x1= 0 y1= 0 x2= 256 y2= 256 materialtype= 0x7fe4c166ad32 ) "internalimage" 1000001 order 0
nslate 390 267 0 "QQuickItem(QQuickText_QML_10:)"  ) 0
entity "text"  ) 1
iangles #V: 140 #I: 210 x1= 20.0244 y1= 8.79277 x2= 479.55 y2= 107.761 materialtype= 0x7fe4c166b051 ) "glyphs" 1000003 order 0
nslate 480 528 0 "QQuickItem(QQuickImage_QML_11:)"  ) 0
```

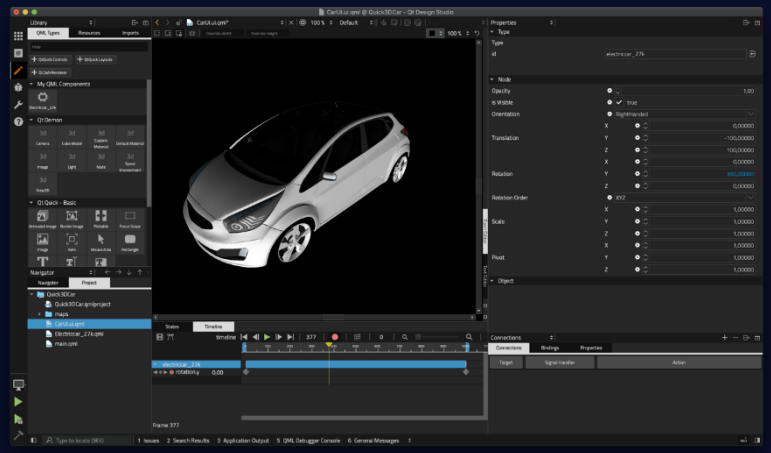
```
Item {
    id: mainViewArea
    anchors.fill: parent

    Background {
        id: background
    }

    ListView {
        id: listView

        property real globalLightPosX: lightImage.x / root.width
        property real globalLightPosY: lightImage.y / root.height

        // Normal-mapped cover shared among delegates
        ShaderEffectSource {
            id: coverNmapSource
            sourceItem: Image { source: "images/cover_nmap.png" }
            hideSource: true
            visible: false
        }

        anchors.fill: parent
        spacing: -60
        model: moviesModel
        delegate: DelegateItem {
            name: model.name
```

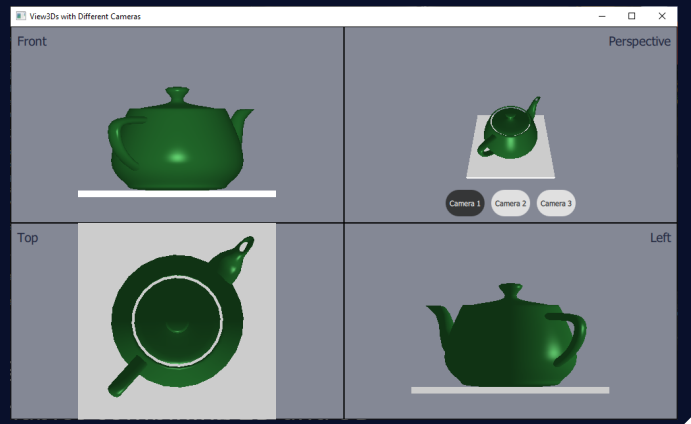

qt5-cinematic-experience_2019.08.19_11.14.47_frame433.rdc - RenderDoc v1.4
File Window Tools Help
Timeline - Frame #434
+ Colour Pass #1 (1 Targets + Depth)
+ Colour Pass #2 (1 Targets + Depth)
+ Colour Pass #3 (1 Targets + Depth)
+ Colour Pass #4 (1 Targets + Depth)
Usage for Backbuffer Color: Reads ( ), Writes ( ), Read/Write ( ), and Clears ( )
Event Browser
Controls
EID Name
Frame #434
Frame Start
Colour Pass #1 (1 Targets + Depth)
Colour Pass #2 (1 Targets + Depth)
Colour Pass #3 (1 Targets + Depth)
glClear(Color = <0.000000, 0.000000, 0.000000, 0.000000>, Depth = <1.00...
glDrawElements(4)
glDrawElements(192)
glDrawElements(12)
glDrawElements(18)
glDrawElements(4)
Colour Pass #4 (1 Targets + Depth)
glClear(Color = <1.000000, 1.000000, 1.000000, 1.000000>, Depth = <1.00...
glDrawElements(16)
glDrawElements(4)
glDrawElements(120)
glDrawElements(6)
glDrawElements(6)
glDrawElements(6)
glDrawElements(210)
glDrawElements(6)
glDrawElements(1200)
glDrawElements(264)
glDrawElements(4)
glDrawElements(54)
glDrawElements(516)
glDrawElements(6)
glDrawElements(4)
glDrawElements(6)
glDrawElements(4)
glDrawElements(6)
glDrawElements(4)
API Inspector
EID Event
glBindBuffer
glBindBuffer
glBindBuffer
glBindBuffer
glUseProgram
glBlendFunc
glBlendColor
glUniform1fv
glUniform4fv
glUniformMatrix4fv
glUniform1fv
Callstack
Texture Viewer
Resource Inspector
Launch Application
qt5-cinematic-experience [PID 39132]
Channels RGBA
Subresource
Slice/Face
Actions
Zoom 1:1 Fit 79%
Overlay None
Range 0.00
Cur Output 0 - Backbuffer Color
1. Lord of the Rings: The Return of the King
Show movies cover lighting?
Show shooting star particles?
Show fog particles?
Do you l-o-v-e colors?
Backbuffer Color - 1280x720 1 mips - R8G8B8A8_SRGB
Hover -
Pipeline State
Controls
Show Disabled Items
Show Empty Items
Export
Extensions
VTX
VS
TCS
TES
GS
Rasterizer
Vertex Attribute Formats
Index Enabled Name Format/Generic Value Buffer Sl
0 Enabled vCoord R32G32_FLOAT 0
1 Enabled tCoord R32G32_FLOAT 1
2 Enabled _qt_order R32_FLOAT 2
Vertex Array Object
Default VAO
Buffers
Slot Buffer Stride Offset Divisor Byte Length Go
Element Buffer 78 2 0 0 7912
0 Buffer 78 16 0 0 7912
1 Buffer 78 16 8 0 7912
2 Buffer 78 4 5504 0 7912
Mesh View
Mesh Viewer
Controls
Sync Views
Row Offset 0
Instance 0
View 0
VS Input
VTX IDX vCoord tCoord
0 0 56.9001 364.34348 1162.00
1 2 56.9001 383.11554 1162.00
2 3 70.49638 383.11554 1198.70996
3 3 70.49638 383.11554 1198.70996
VS Output
GS/DS Output
VTX IDX gl_Position
0 0 -0.91109 -0.01207
1 2 -0.91109 -0.06421
2 3 -0.88985 -0.06421
3 3 -0.88985 -0.06421
Preview
VS Out
GS/DS Out
Controls WASD
Show This draw
Solid Shading None
Wireframe
Highlight Vertices

Qt Quick 3D



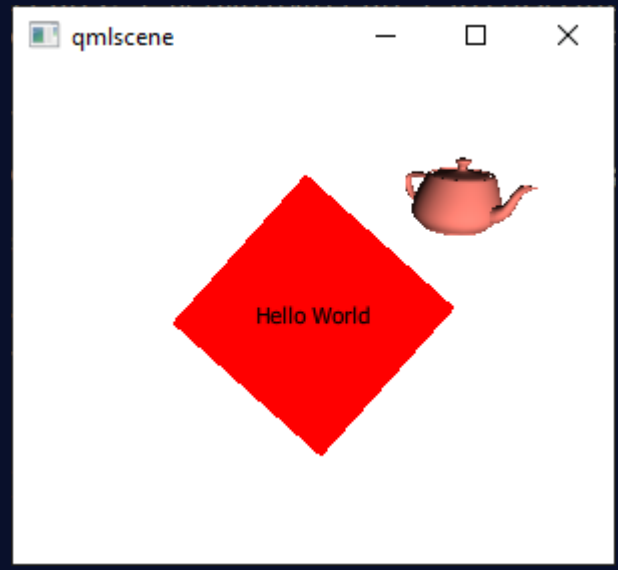






- Not to be confused with Qt 3D.
- QML types for 3D, providing a high-level API for creating 3D content or user interfaces with Qt Quick.
- Tooling! Qt Design Studio.
- Also enables combining 2D and 3D content.

```
import QtQuick 2.15
import QtQuick3D 1.15

Rectangle {
    View3D {
        anchors.fill: parent
        camera: camera
        Model {
            source: "teapot.mesh"
            x: 100
            y: 100
            scale: Qt.vector3d(15, 15, 15)
            materials: [ DefaultMaterial { diffuseColor: "salmon" } ]
            PropertyAnimation on eulerRotation.y { from: 0; to: 360; duration: 2000; loops: -1 }
        }
        DirectionalLight {
            id: dirLight
            ambientColor: Qt.rgba(0.1, 0.1, 0.1, 1.0);
        }
        PerspectiveCamera {
            id: camera
            position: Qt.vector3d(0, 200, 300)
            eulerRotation: Qt.vector3d(-30, 0, 0)
        }
    }

    Rectangle {
        width: 100
        height: 100
        anchors.centerIn: parent
        color: "red"
        NumberAnimation on rotation {
            from: 0; to: 360; duration: 2000; loops: Animation.Infinite;
        }
    }

    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
}
```



Qt Quick 3D

- Forward renderer
- Default/Principled material (PBR)
- Lights (directional, point, area, spot)
- Light probes
- Shadow mapping, SSAO, skybox
- Progressive, temporal AA
- Post-processing effects
- Custom materials
- Assets: import FBX, glTF2, etc.

Qt Quick 3D

- Uses OpenGL
  - ES 3.0+ or 3.3+
  - Feature limited GLES 2.0 level rendering path exists
- Material and effect systems all based on GLSL
- So same story, more or less, as with Qt Quick

How does this change in Qt 6?

Qt Quick, revisited

- QQuickItem tree
  - described in QML
  - main thread
- QSGNode tree + material system
  - the “scene graph”
  - render thread

~~OpenGL~~ OpenGL, Vulkan, Metal, Direct 3D, (WebGPU, …)

https://wiki.qt.io/New_Features_in_Qt_5.14

- Qt Quick
  - Added the first preview of the graphics API independent scenegraph renderer as an opt-in feature. This allows running qualifying Qt Quick applications on top of Vulkan, Metal, or Direct3D 11 instead of OpenGL. The supported platforms are currently Windows 10, Linux with X11 (xcb), macOS with MoltenVK, or Android 7.0+ for Vulkan, macOS for Metal, Windows 10 for D3D.

Why?

1. Qt Everywhere

2. Qt Quick(3D) application
+
external rendering

3. Shader pipeline renewal

4. Possible performance gains

+1. Enable new stuff

(for example, compute)

https://wiki.qt.io/New_Features_in_Qt_5.14

- Still tech preview in Qt 5.15
  - What’s new in 5.15:
    - Enabled Metal on iOS
    - Initial work to enable Vulkan on Wayland (clients)
    - Lots of fixes and improvements
      - In fact, all Qt 6 focused fixes are in the 5.15 branch atm.

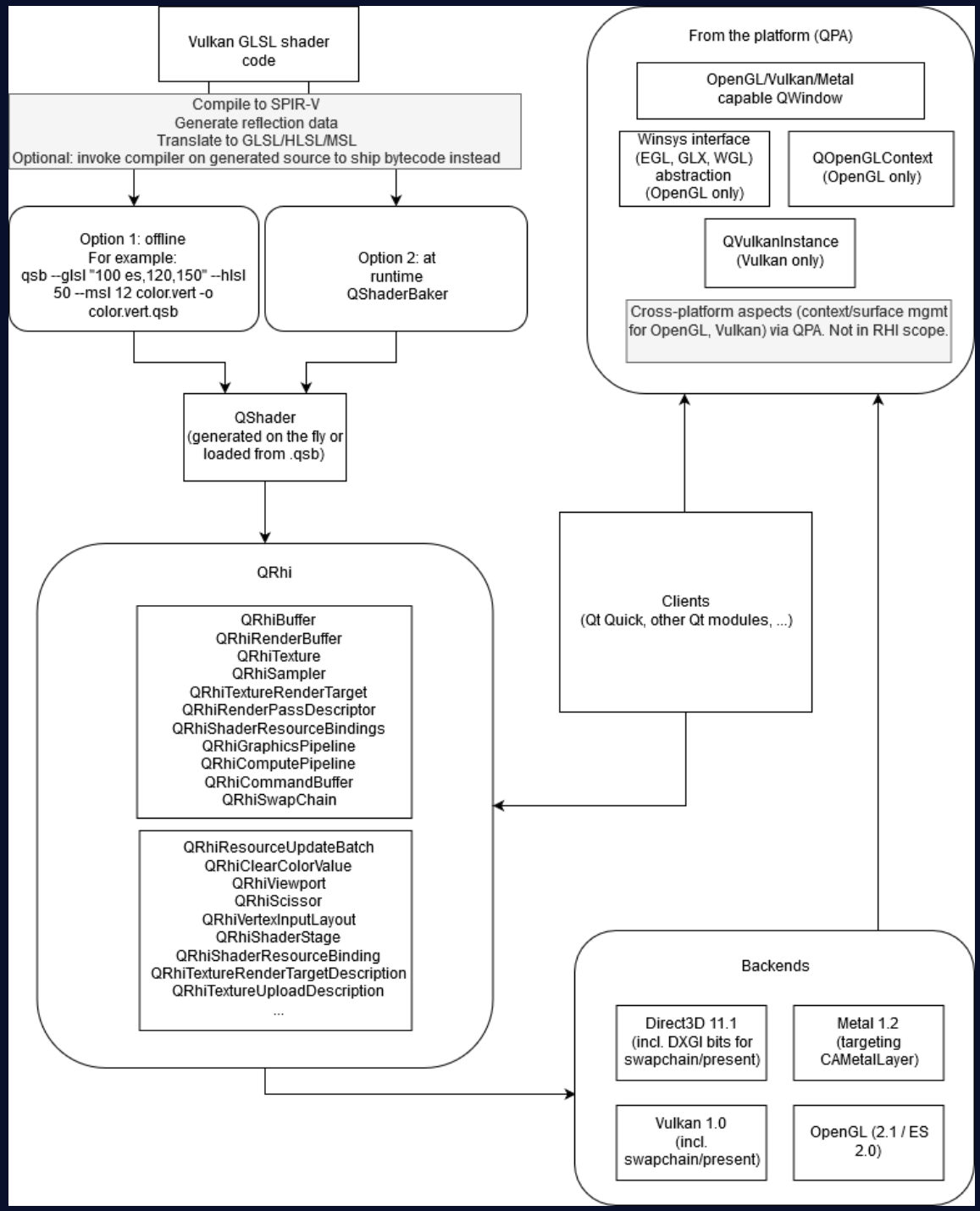
https://wiki.qt.io/New_Features_in_Qt_5.14

- The plan for Qt 6.0 is to make this the **default** and **only** rendering path for Qt Quick
  - not an opt-in feature, like in 5.14 and 5.15
- And the same for Qt Quick 3D!

How?

Components

- Qt RHI
  - Part of the QtGui module
  - Private API
- Qt Shader Tools
  - Own module
- Qt Quick (scenegraph) port
- Qt Quick 3D port



QML (Qt Quick) application

QtQuick + scenegraph

QtShaderTools

QtGui
(QGuiApplication, QWindow, QOpenGLContext, QPainter, …)

QtRHI

QRhi backends

QPA
(QPlatformWindow, QPlatformOpenGLContext, QPlatformVulkanInstance, …)

Platform plugin
(implementing QPlatform*)

Backends/plugins for the platform plugin
(if the platform has multiple windowing systems and related APIs)

The OS and platform:
Win32/xcb/Wayland/Cocoa/UIKit/Android, …
EGL/GLX/WGL, DRM+GBM/DRM+EGLStream,
Screen, OpenWF, proprietary stuff, …

OpenGL

Vulkan

Metal

Direct 3D

```
macs@DESKTOP-GIGNP5J
```

```
    bool useNewStyleShader = format.profile() == QSurfaceFormat::CoreProfile;
    // Try to handle 3.0 & 3.1 that do not have the core/compatibility profile concept 3.2+ has.
    // This may still fail since version 150 (3.2) is specified in the sources but it's worth a try.
    if (format.renderableType() == QSurfaceFormat::OpenGL && format.majorVersion() == 3 && format.minorVersion() <= 1)
        useNewStyleShader = !format.testOption(QSurfaceFormat::DeprecatedFunctions);
    if (m_forceGLSL110)
        useNewStyleShader = false;

    const char *vsrc = useNewStyleShader ? vertexShaderSource : vertexShaderSource110;
    const char *fsrc = useNewStyleShader ? fragmentShaderSource : fragmentShaderSource110;
    qDebug("Using version %s shader", useNewStyleShader ? "150" : "110");

    if (!m_program->addShaderFromSourceCode(QOpenGLShader::Vertex, vsrc)) {
        emit error(m_program->log());
        return;
    }
    if (!m_program->addShaderFromSourceCode(QOpenGLShader::Fragment, fsrc)) {
        emit error(m_program->log());
        return;
    }
    if (!m_program->link()) {
        emit error(m_program->log());
        return;
    }

    m_posAttr = m_program->attributeLocation("posAttr");
    m_colAttr = m_program->attributeLocation("colAttr");
    m_matrixUniform = m_program->uniformLocation("matrix");

    m_vbo.create();
    m_vbo.bind();
    m_vbo.allocate(vertices, sizeof(vertices) + sizeof(colors));
    m_vbo.write(sizeof(vertices), colors, sizeof(colors));
    m_vbo.release();

    QOpenGLVertexArrayObject::Binder vaoBinder(&m_vao);
    if (m_vao.isCreated()) // have VAO support, use it
        setupVertexAttribs();
}

void RenderWindow::setupVertexAttribs()
{
    m_vbo.bind();
    m_program->setAttributeBuffer(m_posAttr, GL_FLOAT, 0, 2);
    m_program->setAttributeBuffer(m_colAttr, GL_FLOAT, sizeof(vertices), 3);
    m_program->enableAttributeArray(m_posAttr);
    m_program->enableAttributeArray(m_colAttr);
    m_vbo.release();
}

bool RenderWindow::event(QEvent *ev)
```

```
-\-   renderwindow.cpp   57% (183,0)   Git-5.15   (C++//l Abbrev)
```

```
{
    if (ev->type() == QEvent::UpdateRequest)
        render();
    return QWindow::event(ev);
}

void RenderWindow::render()
{
    if (!m_context->makeCurrent(this)) {
        emit error(tr("makeCurrent() failed"));
        return;
    }

    QOpenGLFunctions *f = m_context->functions();
    if (!m_initialized) {
        m_initialized = true;
        f->glEnable(GL_DEPTH_TEST);
        f->glClearColor(0, 0, 0, 1);
        init();
        emit ready();
    }

    if (!m_vao.isCreated()) // init() failed, don't bother with trying to render
        return;

    const qreal retinaScale = devicePixelRatio();
    f->glViewport(0, 0, width() * retinaScale, height() * retinaScale);
    f->glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    m_program->bind();
    QMatrix4x4 matrix;
    matrix.perspective(60.0f, 4.0f / 3.0f, 0.1f, 100.0f);
    matrix.translate(0.0f, 0.0f, -2.0f);
    matrix.rotate(m_angle, 0.0f, 1.0f, 0.0f);
    m_program->setUniformValue(m_matrixUniform, matrix);

    if (m_vao.isCreated())
        m_vao.bind();
    else // no VAO support, set the vertex attribute arrays now
        setupVertexAttribs();

    f->glDrawArrays(GL_TRIANGLES, 0, 3);

    m_vao.release();
    m_program->release();

    // swapInterval is 1 by default which means that swapBuffers() will (hopefully) block
    // and wait for vsync.
    m_context->swapBuffers(this);

    m_angle += 1.0f;
```

```
-\---   renderwindow.cpp   82% (205,0)   Git-5.15   (C++//l Abbrev)
```

Close

emacs@DESKTOP-GIGNP5J

```
void HelloWindow::customInit()
{
    m_vbuf.reset(m_rhi->newBuffer(QRhiBuffer::Immutable, QRhiBuffer::VertexBuffer, sizeof(vertexData))
);
    m_vbuf->build();
    m_vbufReady = false;

    m_ubuf.reset(m_rhi->newBuffer(QRhiBuffer::Dynamic, QRhiBuffer::UniformBuffer, 68));
    m_ubuf->build();

    m_srb.reset(m_rhi->newShaderResourceBindings());
    m_srb->setBindings({
            QRhiShaderResourceBinding::uniformBuffer(0,
                                                     QRhiShaderResourceBinding::VertexStage
                                                     | QRhiShaderResourceBinding::FragmentStage,
                                                     m_ubuf.get())
    });
    m_srb->build();

    m_ps.reset(m_rhi->newGraphicsPipeline());

    QRhiGraphicsPipeline::TargetBlend premulAlphaBlend;
    premulAlphaBlend.enable = true;
    m_ps->setTargetBlends({ premulAlphaBlend });

    const QShader vs = getShader(QLatin1String(":/color.vert.qsb"));
    if (!vs.isValid())
        qFatal("Failed to load shader pack (vertex)");
    const QShader fs = getShader(QLatin1String(":/color.frag.qsb"));
    if (!fs.isValid())
        qFatal("Failed to load shader pack (fragment)");

    m_ps->setShaderStages({
        { QRhiShaderStage::Vertex, vs },
        { QRhiShaderStage::Fragment, fs }
    });

    QRhiVertexInputLayout inputLayout;
    inputLayout.setBindings({
        { 5 * sizeof(float) }
    });
    inputLayout.setAttributes({
        { 0, 0, QRhiVertexInputAttribute::Float2, 0 },
        { 0, 1, QRhiVertexInputAttribute::Float3, 2 * sizeof(float) }
    });

    m_ps->setVertexInputLayout(inputLayout);
    m_ps->setShaderResourceBindings(m_srb.get());
    m_ps->setRenderPassDescriptor(m_rp.get());

    m_ps->build();
}

// called once per frame
void HelloWindow::customRender()
{
    QRhiResourceUpdateBatch *u = m_rhi->nextResourceUpdateBatch();
    if (!m_vbufReady) {
        m_vbufReady = true;
        u->uploadStaticBuffer(m_vbuf.get(), vertexData);
    }
    m_rotation += 1.0f;
    QMatrix4x4 mvp = m_proj;
    mvp.rotate(m_rotation, 0, 1, 0);
    u->updateDynamicBuffer(m_ubuf.get(), 0, 64, mvp.constData());
    m_opacity += m_opacityDir * 0.005f;
    if (m_opacity < 0.0f || m_opacity > 1.0f) {
        m_opacityDir *= -1;
        m_opacity = qBound(0.0f, m_opacity, 1.0f);
    }
    u->updateDynamicBuffer(m_ubuf.get(), 64, 4, &m_opacity);

    QRhiCommandBuffer *cb = m_sc->currentFrameCommandBuffer();
    const QSize outputSizeInPixels = m_sc->currentPixelSize();

    cb->beginPass(m_sc->currentFrameRenderTarget(), QColor::fromRgbF(0.4f, 0.7f, 0.0f, 1.0f), { 1.0f, 0
}, u);

    cb->setGraphicsPipeline(m_ps.get());
    cb->setViewport({ 0, 0, float(outputSizeInPixels.width()), float(outputSizeInPixels.height()) });
    cb->setShaderResources();

    const QRhiCommandBuffer::VertexInput vbufBinding(m_vbuf.get(), 0);
    cb->setVertexInput(0, 1, &vbufBinding);
    cb->draw(3);

    cb->endPass();
}
```

-**- **hellowindow.cpp** 51% (125,18) Git-5.15 (C++//l Abbrev)

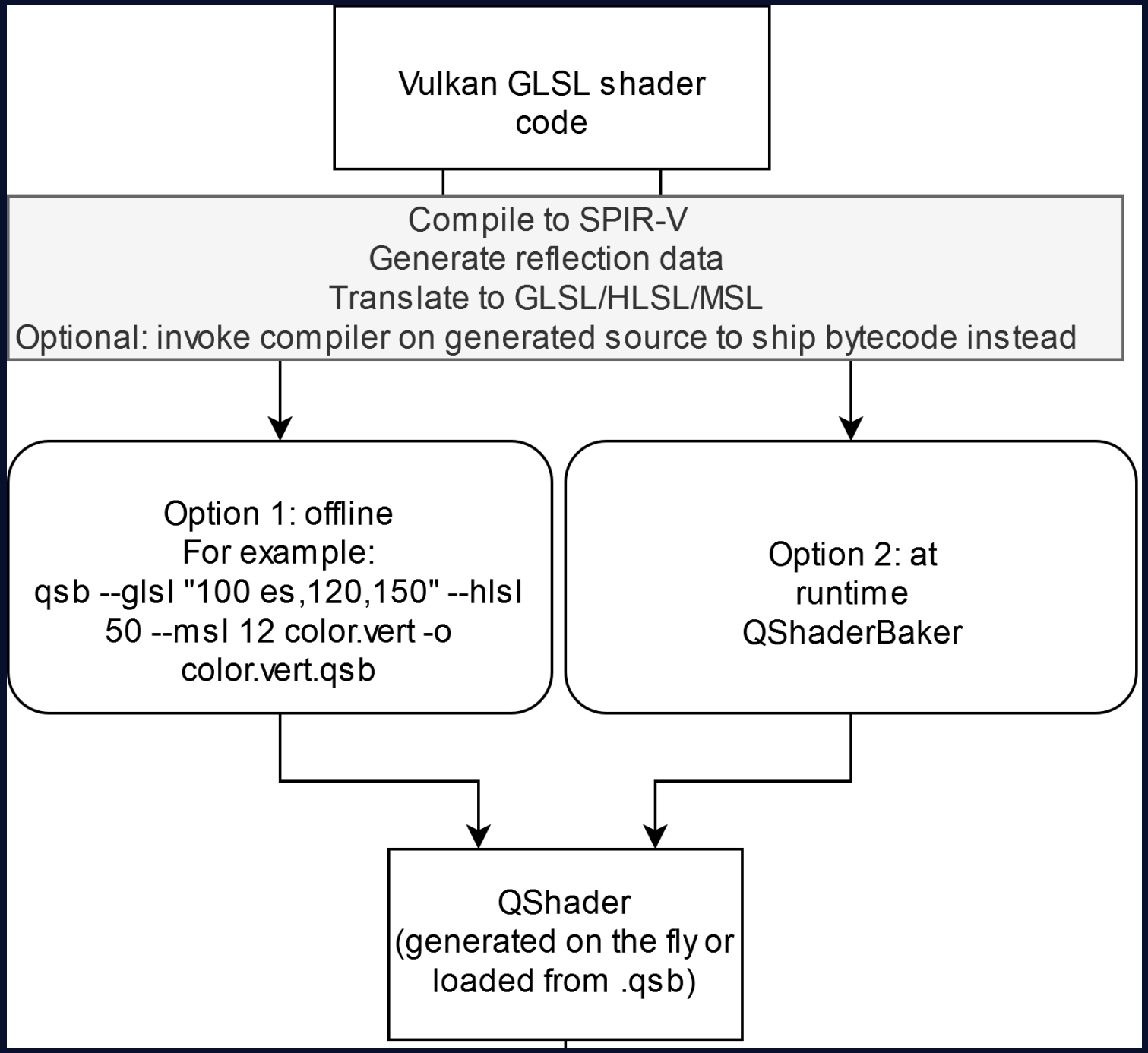
-**- **hellowindow.cpp** Bot (131,0) Git-5.15 (C++//l Abbrev)

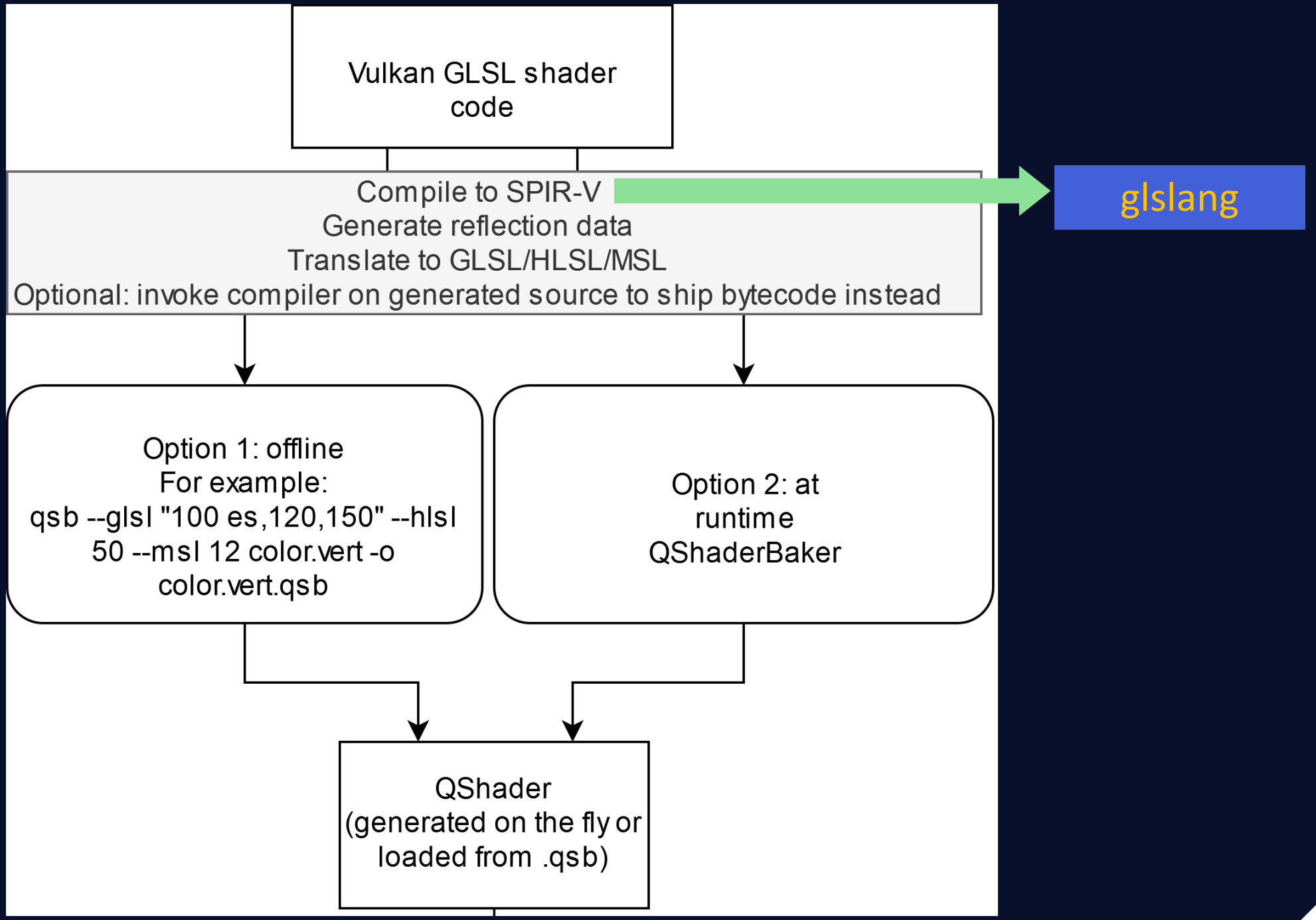
Qt Rendering Hardware Interface

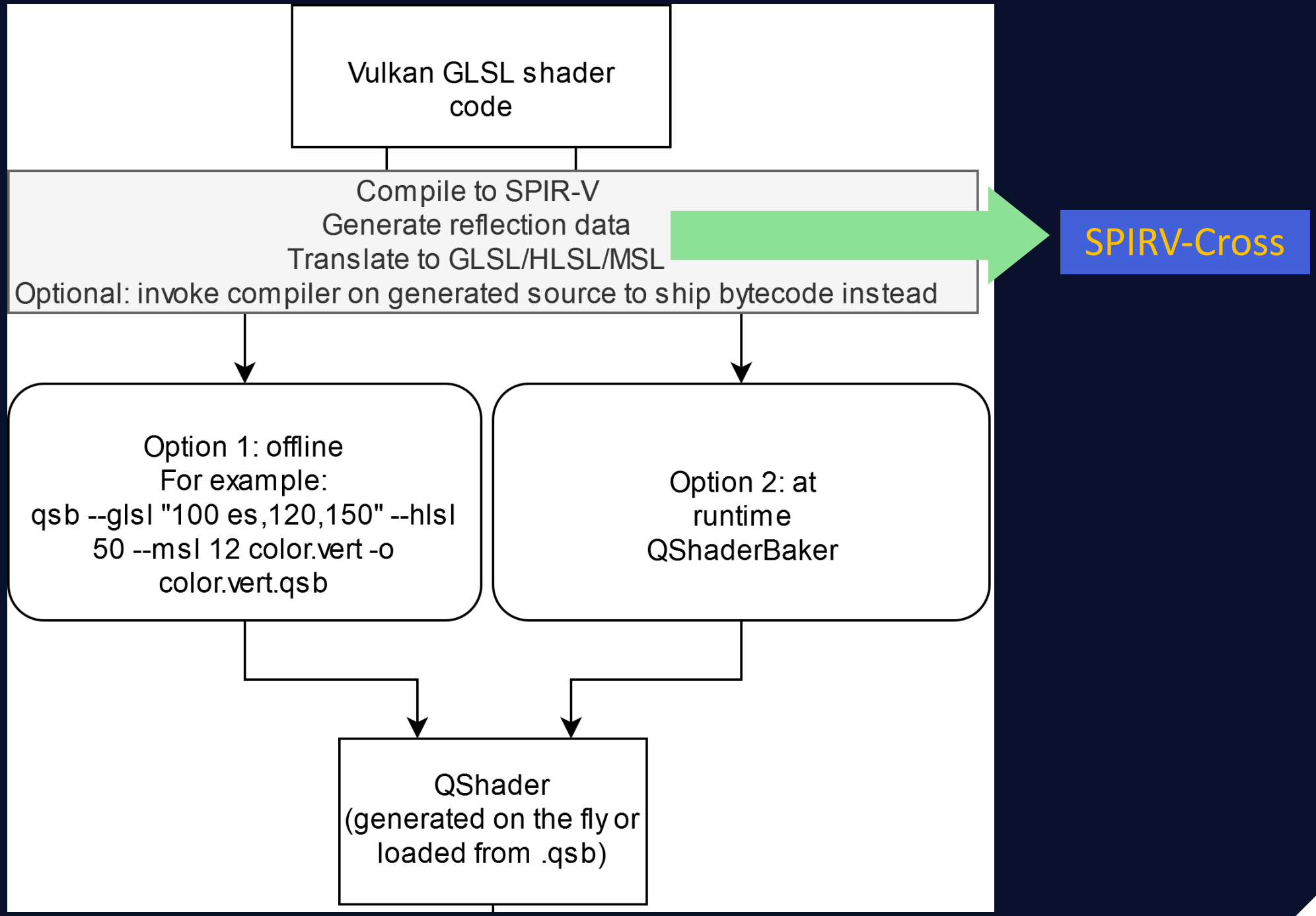
- QRhi and related structs is the main interface
  - This is what Qt Quick and Quick 3D will use instead of QOpenGLContext, the QOpenGL wrappers, and calling OpenGL functions directly.
- Has backends for
  - Vulkan (1.0) (shaders: SPIR-V 1.0)
  - D3D11 (11.1) (shaders: HLSL Shader Model 5.0)
  - Metal (1) (shaders: MSL 1.2 (or 2.0 for some things))
  - OpenGL (ES) 2/3 (needs 4.3/ES3.1 for compute) (shaders: various GLSL versions)

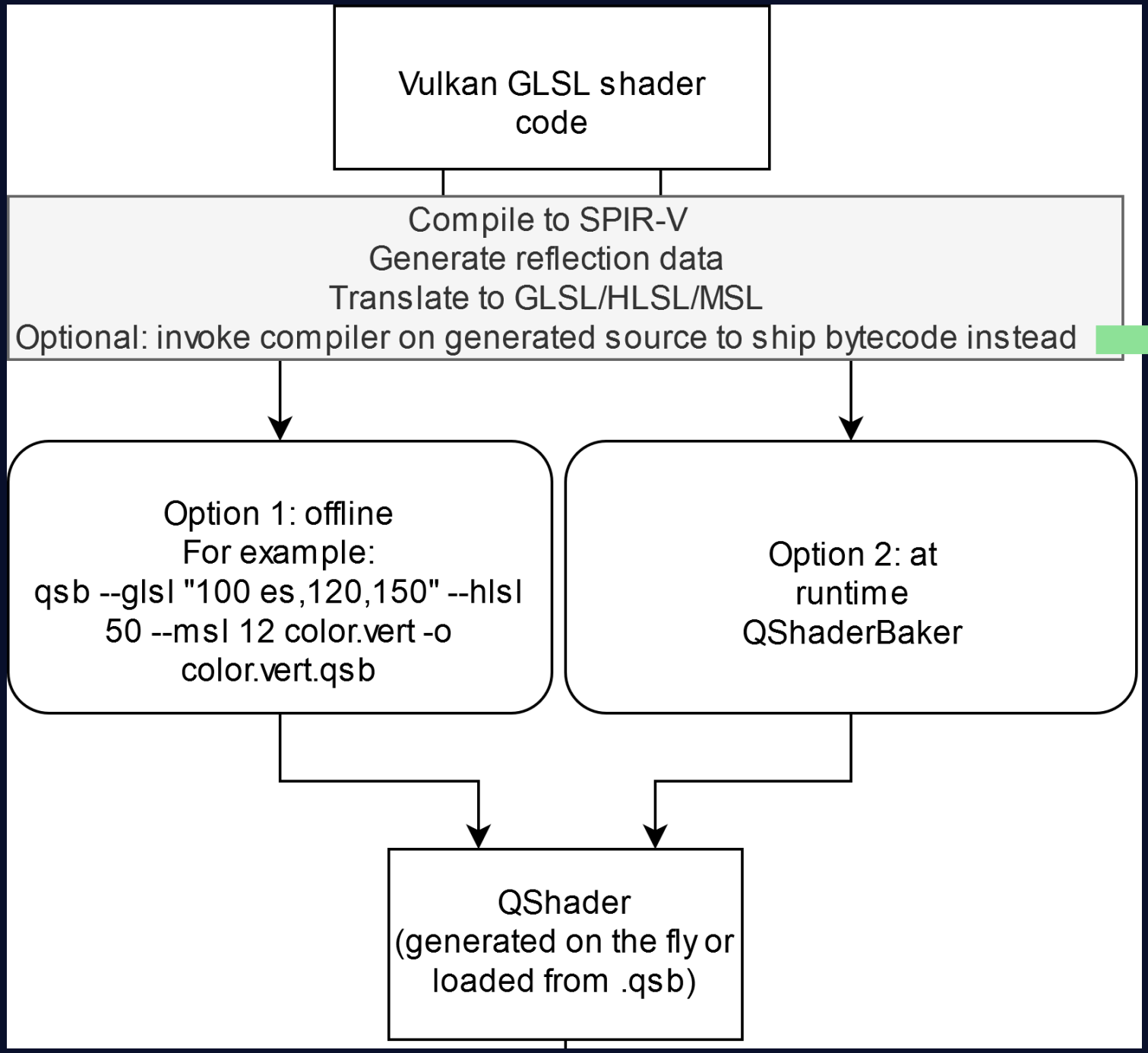
Qt Rendering Hardware Interface

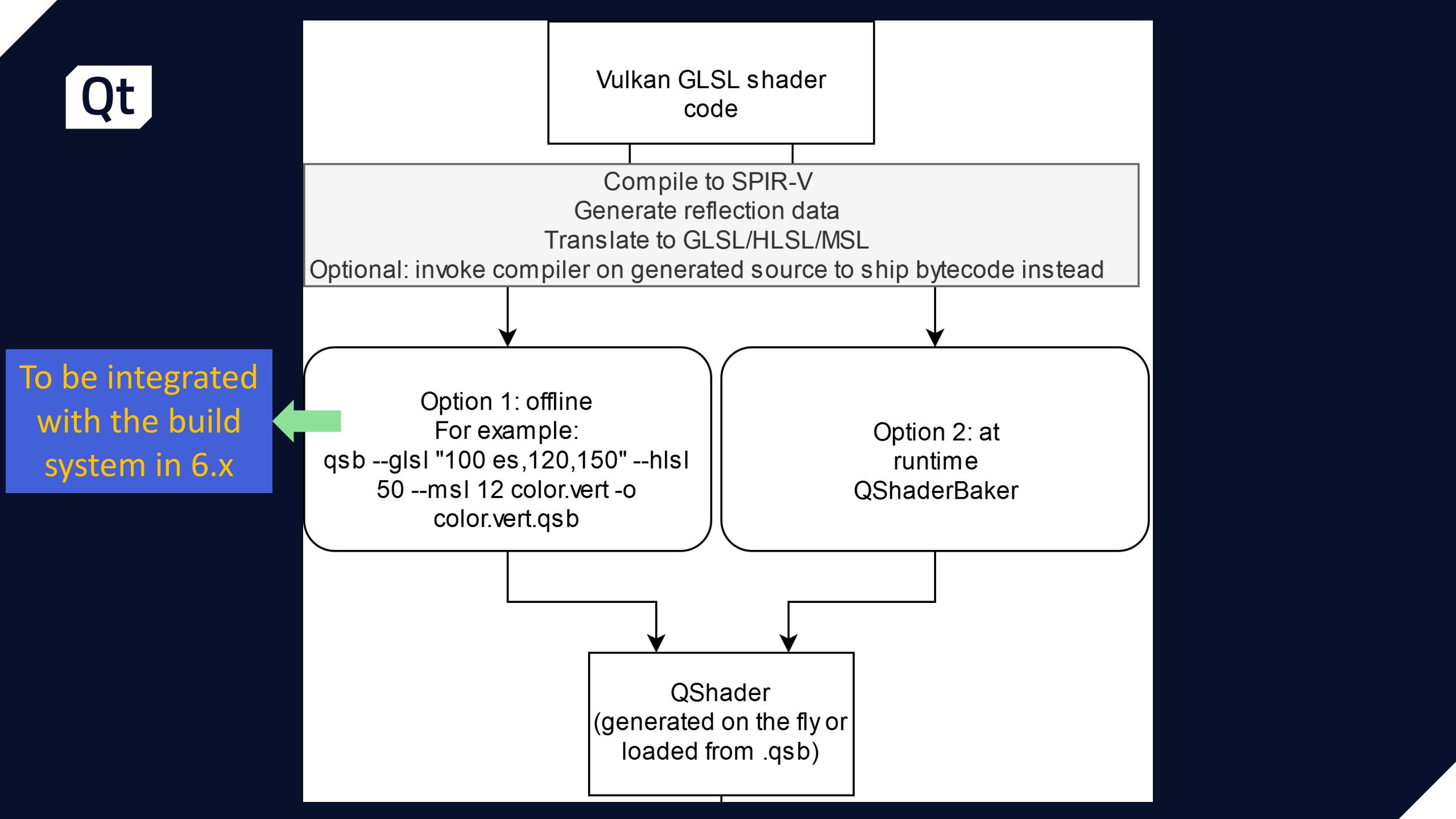
- Command buffer oriented API.
- Resource updates (upload, copy, readback) and render/compute pass recording are separate phases.
  - Think MTLBlitCommandEncoder vs. MTLRenderCommandEncoder or the Vulkan renderpass concept.
- Vertex, fragment, compute shader stages.
  - No geometry or tessellation
- Attempts to enable multiple frames in flight (reduce pipeline stalls)
  - No low-level operations such as map/unmap. Instead, one enqueues upload/update/copy operations on a *resource update batch*, that is then committed at latest when starting to record a render/compute pass.
  - Handle host visible buffers smarter than just a dumb 1:1 QRhiBuffer – VkBuffer/MTLBuffer/… mapping.
    - Some backends internally double/triple buffer such native objects, transparently to the API client.
  - No headache with resources being still in-use: a QRhi* C++ object can be destroyed safely once a frame has been submitted, even if the underlying native resource is still in use by an in-flight frame.
- No multi queue for now.


Vulkan GLSL shader code
Compile to SPIR-V
Generate reflection data
Translate to GLSL/HLSL/MSL
Optional: invoke compiler on generated source to ship bytecode instead
Option 1: offline
For example:
qsb --glsl "100 es,120,150" --hlsl 50 --msl 12 color.vert -o color.vert.qsb
Option 2: at runtime
QShaderBaker
QShader
(generated on the fly or loaded from .qsb)


Vulkan GLSL shader code
Compile to SPIR-V
Generate reflection data
Translate to GLSL/HLSL/MSL
Optional: invoke compiler on generated source to ship bytecode instead
glslang
Option 1: offline
For example:
qsb --glsl "100 es,120,150" --hlsl 50 --msl 12 color.vert -o color.vert.qsb
Option 2: at runtime
QShaderBaker
QShader
(generated on the fly or loaded from .qsb)


Vulkan GLSL shader code
Compile to SPIR-V
Generate reflection data
Translate to GLSL/HLSL/MSL
Optional: invoke compiler on generated source to ship bytecode instead
SPIRV-Cross
Option 1: offline
For example:
qsb --glsl "100 es,120,150" --hlsl 50 --msl 12 color.vert -o color.vert.qsb
Option 2: at runtime
QShaderBaker
QShader
(generated on the fly or loaded from .qsb)


Vulkan GLSL shader code
Compile to SPIR-V
Generate reflection data
Translate to GLSL/HLSL/MSL
Optional: invoke compiler on generated source to ship bytecode instead
fxc, dxc,
xcrun metal(lib),
...
Option 1: offline
For example:
qsb --glsl "100 es,120,150" --hlsl 50 --msl 12 color.vert -o color.vert.qsb
Option 2: at runtime
QShaderBaker
QShader
(generated on the fly or loaded from .qsb)


Vulkan GLSL shader code
Compile to SPIR-V
Generate reflection data
Translate to GLSL/HLSL/MSL
Optional: invoke compiler on generated source to ship bytecode instead
To be integrated with the build system in 6.x
Option 1: offline
For example:
qsb --glsl "100 es,120,150" --hlsl 50 --msl 12 color.vert -o color.vert.qsb
Option 2: at runtime
QShaderBaker
QShader
(generated on the fly or loaded from .qsb)

```
#version 440

layout(location = 0) in vec2 sampleCoord;
layout(location = 0) out vec4 fragColor;

layout(binding = 1) uniform sampler2D _qt_texture;

layout(std140, binding = 0) uniform buf {
    mat4 matrix;
    vec4 color;
    vec2 textureScale;
    float dpr;
} ubuf;

void main()
{
    vec4 glyph = texture(_qt_texture, sampleCoord);
    fragColor = vec4(glyph.rgb * ubuf.color.a, glyph.a);
}
```

qsb --glsl "150,120,100 es" --hlsl 50 --msl 12
-o textmask.frag.qsb textmask.frag

```
Stage: Fragment

Has 6 shaders: (unordered list)
  Shader 0: HLSL 50 [Standard]
  Shader 1: GLSL 150 [Standard]
  Shader 2: GLSL 100 es [Standard]
  Shader 3: GLSL 120 [Standard]
  Shader 4: SPIR-V 100 [Standard]
  Shader 5: MSL 12 [Standard]

Reflection info: {
    "combinedImageSamplers": [
        {
            "binding": 1,
            "name": "_qt_texture",
            "set": 0,
            "type": "sampler2D"
        }
    ],
    "inputs": [
        {
            "location": 0,
            "name": "sampleCoord",
            "type": "vec2"
        }
```

```
Shader 0: HLSL 50 [Standard]
Entry point: main
Contents:
cbuffer buf : register(b0)
{
    row_major float4x4 ubuf_matrix : packoffset(c0);
    float4 ubuf_color : packoffset(c4);
    float2 ubuf_textureScale : packoffset(c5);
    float ubuf_dpr : packoffset(c5.z);
};

Texture2D<float4> _qt_texture : register(t1);
SamplerState __qt_texture_sampler : register(s1);
```

```
    "uniformBlocks": [
        {
            "binding": 0,
            "blockName": "buf",
            "members": [
                {
                    "matrixStride": 16,
                    "name": "matrix",
                    "offset": 0,
                    "size": 64,
                    "type": "mat4"
                },
                {
                    "name": "color",
                    "offset": 64,
                    "size": 16,
                    "type": "vec4"
                },
                {
                    "name": "textureScale",
                    "offset": 80,
                    "size": 8,
                    "type": "vec2"
                },
                {
                    "name": "dpr",
                    "offset": 88,
                    "size": 4,
                    "type": "float"
                }
            ],
            "set": 0,
            "size": 92,
            "structName": "ubuf"
        }
```

...

Consequences for applications

- Many Quick/Quick3D applications will just work.
- Advanced use cases, often the ones that involve application-provided shader code, may need to do some migration.

ShaderEffect

```
import QtQuick 2.0

Rectangle {
    width: 200; height: 100
    Row {
        Image { id: img;
                sourceSize { width: 100; height: 100 } source: "qt-logo.png" }
        ShaderEffect {
            width: 100; height: 100
            property variant src: img
            vertexShader: "
                uniform highp mat4 qt_Matrix;
                attribute highp vec4 qt_Vertex;
                attribute highp vec2 qt_MultiTexCoord0;
                varying highp vec2 coord;
                void main() {
                    coord = qt_MultiTexCoord0;
                    gl_Position = qt_Matrix * qt_Vertex;
                }"
            fragmentShader: "
                varying highp vec2 coord;
                uniform sampler2D src;
                uniform lowp float qt_Opacity;
                void main() {
                    lowp vec4 tex = texture2D(src, coord);
                    gl_FragColor = vec4(vec3(dot(tex.rgb,
                                        vec3(0.344, 0.5, 0.156))),
                                             tex.a) * qt_Opacity;
                }"
        }
```

```
import QtQuick 2.0

Rectangle {
    width: 200; height: 100
    Row {
        Image { id: img;
                sourceSize { width: 100; height: 100 } source: "qt-logo.png" }
        ShaderEffect {
            width: 100; height: 100
            property variant src: img
            vertexShader: "
                uniform highp mat4 qt_Matrix;
                attribute highp vec4 qt_Vertex;
                attribute highp vec2 qt_MultiTexCoord0;
                varying highp vec2 coord;
                void main() {
                    coord = qt_MultiTexCoord0;
                    gl_Position = qt_Matrix * qt_Vertex;
                }"
            fragmentShader: "
                varying highp vec2 coord;
                uniform sampler2D src;
                uniform lowp float qt_Opacity;
                void main() {
                    lowp vec4 tex = texture2D(src, coord);
                    gl_FragColor = vec4(vec3(dot(tex.rgb,
                                        vec3(0.344, 0.5, 0.156))),
                                             tex.a) * qt_Opacity;
                }"
        }
```

???

```
ShaderEffect {
    width: 160
    height: 160
    property variant source: theSource
    property real amplitude: 0.04 * wobbleSlider.value
    property real frequency: 20
    property real time: 0
    NumberAnimation on time { loops: Animation.Infinite; from: 0; to: Math.PI * 2; duration: 600 }
    fragmentShader: "qrc:/wobble.frag.qsb"
```

- The vertexShader and fragmentShader properties also accept a URL (as in local file or qrc URL) since Qt 5.8.
- Dynamic shader strings are problematic with the new pipeline.
- Therefore, the default approach is to
  - use the QtShaderTools provided tools (or the build system, eventually) to condition shaders offline,
  - ship the resulting .qsb shader pack file with the application,
  - and reference that from ShaderEffect.

```
ShaderEffect {
    width: 160
    height: 160
    property variant source: theSource
    property real amplitude: 0.04 * wobbleSlider.value
    property real frequency: 20
    property real time: 0
    NumberAnimation on time { loops: Animation.Infinite; from: 0; to: Math.PI * 2; duration: 600 }
    fragmentShader: "qrc:/wobble.frag.qsb"
```

- It could be that some compatibility solutions will be provided just for OpenGL. TBD.
- To be evaluated if/to what extent runtime shader processing is promoted in Qt 6 and beyond.

Qt Quick Materials
(QSGMaterial and co.)

QSGMaterial creates a QSGMaterialShader
-> suitable for direct OpenGL

```
class Q_QUICK_EXPORT QSGMaterialShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
        inline bool isMatrixDirty() const { return m_dirty & DirtyMatrix; }
        ...
        QMatrix4x4 combinedMatrix() const;
        ...
    };

    virtual void updateState(const RenderState &state,
                             QSGMaterial *newMaterial,
                             QSGMaterial *oldMaterial);
    virtual char const *const *attributeNames() const = 0;

    virtual void initialize();
    virtual void activate();
    virtual void deactivate();

    void setShaderSourceFile(QOpenGLShader::ShaderType type, const QString &sourceFile);

    virtual const char *vertexShader() const;
    virtual const char *fragmentShader() const;
    virtual void compile();
```

```
class Q_QUICK_EXPORT QSGMaterialShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
        inline bool isMatrixDirty() const { return m_dirty & DirtyMatrix; }
        ...
        QMatrix4x4 combinedMatrix() const;
        ...
    };

    virtual void updateState(const RenderState &state,
                             QSGMaterial *newMaterial,
                             QSGMaterial *oldMaterial);
    virtual char const *const *attributeNames() const = 0;

    virtual void initialize();
    virtual void activate();
    virtual void deactivate();

    void setShaderSourceFile(QOpenGLShader::ShaderType type, const QString &sourceFile);

    virtual const char *vertexShader() const;
    virtual const char *fragmentShader() const;
    virtual void compile();
```

```
class Q_QUICK_EXPORT QSGMaterialShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
        inline bool isMatrixDirty() const { return m_dirty & DirtyMatrix; }
        ...
        QMatrix4x4 combinedMatrix() const;
```

```
void QSGVertexColorMaterialShader::updateState(const RenderState &state, QSGMaterial * , QSGMaterial *)
{
    if (state.isOpacityDirty())
        program()->setUniformValue(m_opacity_id, state.opacity());

    if (state.isMatrixDirty())
        program()->setUniformValue(m_matrix_id, state.combinedMatrix());
}
```

```
    virtual void activate();
    virtual void deactivate();

    void setShaderSourceFile(QOpenGLShader::ShaderType type, const QString &sourceFile);

    virtual const char *vertexShader() const;
    virtual const char *fragmentShader() const;
    virtual void compile();
```

QSGMaterial creates a QSGMaterialRhiShader
-> suitable for QRhi-based rendering

```
class Q_QUICK_EXPORT QSGMaterialRhiShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
        inline bool isMatrixDirty() const { ... }
        ...
        QMatrix4x4 combinedMatrix() const;
        QByteArray *uniformData();
        ...
    };
    enum Flag {
        UpdatesGraphicsPipelineState = 0x0001
    };
    enum Stage {
        VertexStage,
        FragmentStage,
    };

    virtual bool updateUniformData(RenderState &state,
                                   QSGMaterial *newMaterial, QSGMaterial *oldMaterial);

    virtual void updateSampledImage(RenderState &state, int binding, QSGTexture **texture,
                                    QSGMaterial *newMaterial, QSGMaterial *oldMaterial);

    virtual bool updateGraphicsPipelineState(RenderState &state, GraphicsPipelineState *ps,
                                             QSGMaterial *newMaterial,QSGMaterial *oldMaterial);

    void setFlag(Flags flags, bool on = true);

    // filename is for a file containing a serialized QShader.
    void setShaderFileName(Stage stage, const QString &filename);
```

```
class Q_QUICK_EXPORT QSGMaterialRhiShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
        inline bool isMatrixDirty() const { ... }
        ...
        QMatrix4x4 combinedMatrix() const;
        QByteArray *uniformData();
        ...
    };
    enum Flag {
        UpdatesGraphicsPipelineState = 0x0001
    };
```

A material (be it built-in or custom) provides data, and only data.
No graphics API access.

```
    virtual bool updateUniformData(RenderState &state,
                                   QSGMaterial *newMaterial, QSGMaterial *oldMaterial);

    virtual void updateSampledImage(RenderState &state, int binding, QSGTexture **texture,
                                    QSGMaterial *newMaterial, QSGMaterial *oldMaterial);

    virtual bool updateGraphicsPipelineState(RenderState &state, GraphicsPipelineState *ps,
                                             QSGMaterial *newMaterial,QSGMaterial *oldMaterial);

    void setFlag(Flags flags, bool on = true);

    // filename is for a file containing a serialized QShader.
    void setShaderFileName(Stage stage, const QString &filename);
```

```
class Q_QUICK_EXPORT QSGMaterialPhiShader
{
public:
    class Q_QUICK_E
    public:
        inline bool
        ...
        QMatrix4x4
        QByteArray
        ...
    };
    enum Flag {
        UpdatesGrap
    };
    enum Stage {
        VertexStage
        FragmentSta
    };

    virtual bool up

    virtual void up

    virtual bool up

    void setFlag(Fl

    // filename is
    void setShaderFileName(Stage stage, const QString &filename);
```

```
struct Q_QUICK_EXPORT GraphicsPipelineState {
    enum BlendFactor {
        Zero,
        One,
        SrcColor,
        ...
    };

    enum ColorMaskComponent {
        R = 1 << 0,
        G = 1 << 1,
        B = 1 << 2,
        A = 1 << 3
    };
    Q_DECLARE_FLAGS(ColorMask, ColorMaskComponent)

    enum CullMode {
        CullNone,
        CullFront,
        CullBack
    };

    bool blendEnable;
    BlendFactor srcColor;
    BlendFactor dstColor;
    ColorMask colorWrite;
    QColor blendConstant;
    CullMode cullMode;
```

```
class Q_QUICK_EXPORT QSGMaterialRhiShader
{
public:
    class Q_QUICK_EXPORT RenderState {
    public:
```

```
bool QSGVertexColorMaterialRhiShader::updateUniformData(RenderState &state, QSGMaterial *, QSGMaterial *)
{
    bool changed = false;
    QByteArray *buf = state.uniformData();

    if (state.isMatrixDirty()) {
        const QMatrix4x4 m = state.combinedMatrix();
        memcpy(buf->data(), m.constData(), 64);
        changed = true;
    }

    if (state.isOpacityDirty()) {
        const float opacity = state.opacity();
        memcpy(buf->data() + 64, &opacity, 4);
        changed = true;
    }

    return changed;
}
```

```
    void setFlag(Flags flags, bool on = true);

    // filename is for a file containing a serialized QShader.
    void setShaderFileName(Stage stage, const QString &filename);
```

Integrating custom Vulkan/Metal/D3D11/OpenGL rendering

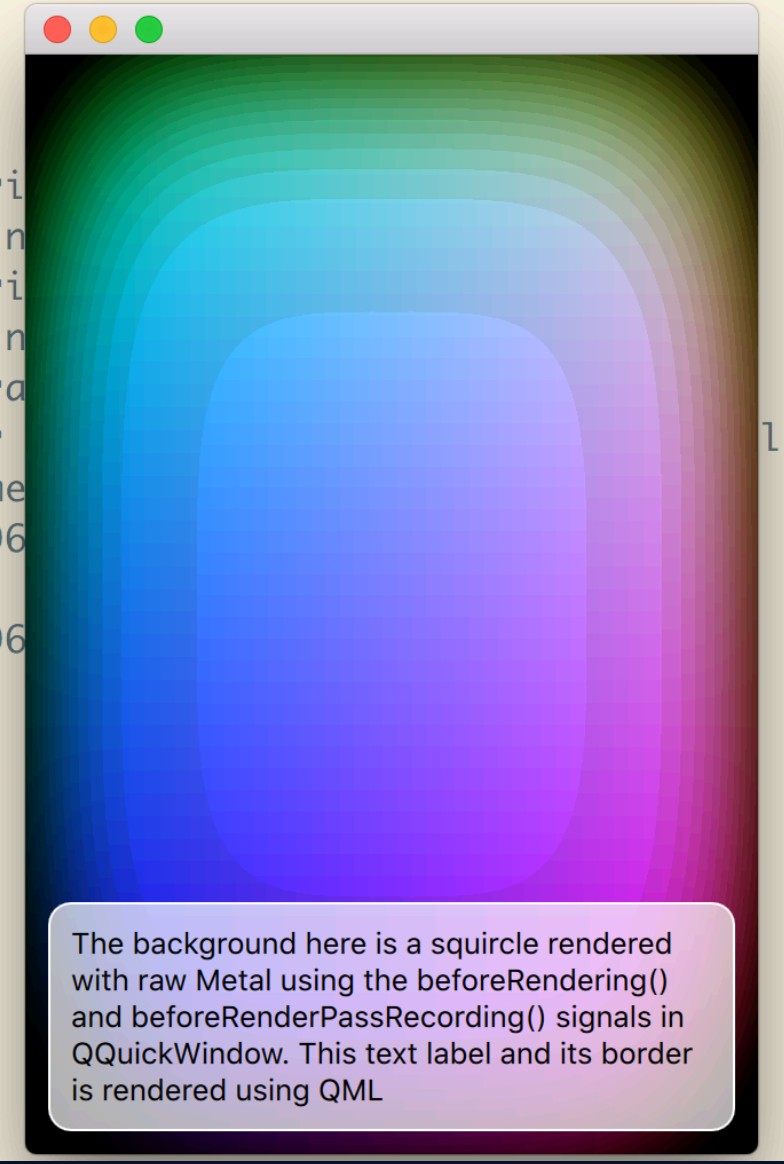
Underlay/overlay

Item with texture

rendering QSGRenderNode

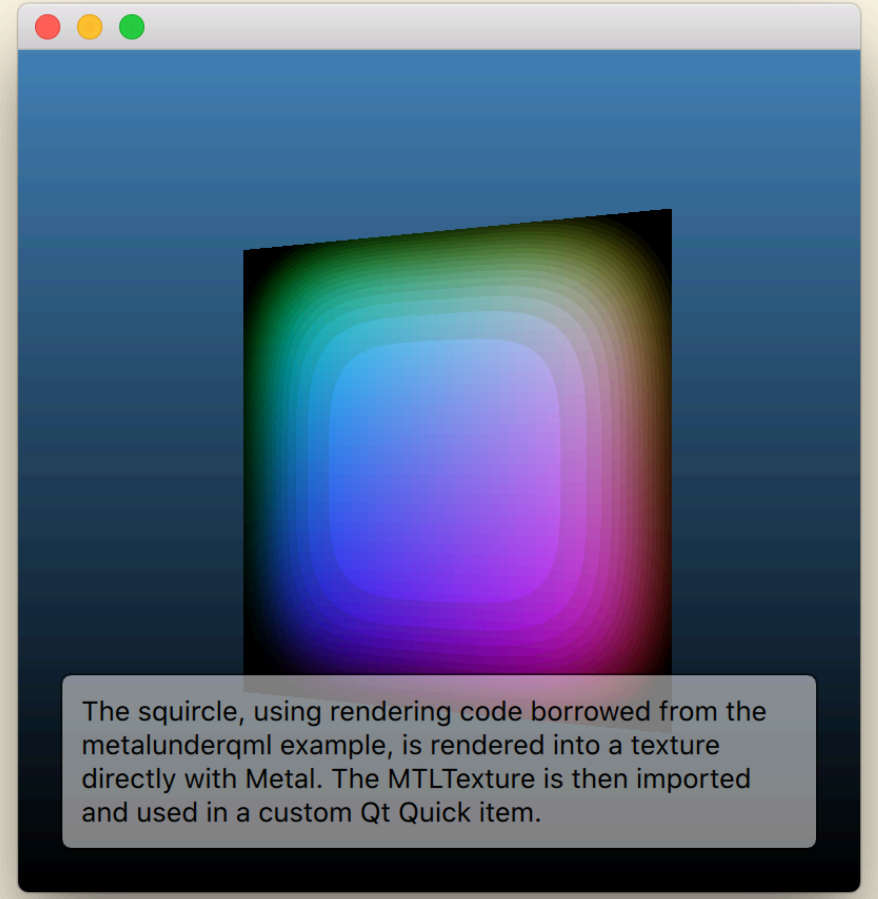
1. metalunderqml

```
~/qtdeclarative_dev/examples/quick/scenegraph/metalunderqml $ ./metalunderqml.app/Contents/MacOS/metalunderqml
qt.scenegraph.general: Using QRhi with backend Metal
  graphics API debug/validation layers: 0
  QRhi profiling and debug markers: 0
qt.scenegraph.general: threaded render loop
qt.scenegraph.general: Using sg animation dri
qt.scenegraph.general: Animation Driver: usin
qt.scenegraph.general: Using sg animation dri
qt.scenegraph.general: Animation Driver: usin
qt.rhi.general: Metal device: Intel(R) HD Gra
qt.scenegraph.general: MSAA sample count for                                  l requested = no.
qt.scenegraph.general: rhi texture atlas dime
qt.rhi.general: got CAMetalLayer, size 640x96
init
qt.rhi.general: got CAMetalLayer, size 640x96
```



1. metaltextureimpo

```
~/qtdeclarative_dev/examples/quick/scenegraph/metaltextureimport $ ./metaltextureimport.app/Contents/MacOS/metaltextureimport
renderer created
Got QSGTexture wrapper QSGPlainTexture(0x7f88c26587b0) for an MTLTexture of size QSize(800, 800)
resources initialized
```



```
QSGRendererInterface *rif = m_window->rendererInterface();
m_device = (id<MTLDevice>) rif->getResource(m_window, QSGRendererInterface::DeviceResource);
Q_ASSERT(m_device);

MTLTextureDescriptor *desc = [[MTLTextureDescriptor alloc] init];
desc.textureType = MTLTextureType2D;
desc.pixelFormat = MTLPixelFormatRGBA8Unorm;
desc.width = m_size.width();
desc.height = m_size.height();
desc.mipmapLevelCount = 1;
desc.resourceOptions = MTLResourceStorageModePrivate;
desc.storageMode = MTLStorageModePrivate;
desc.usage = MTLTextureUsageShaderRead | MTLTextureUsageRenderTarget;
m_texture = [m_device newTextureWithDescriptor: desc];
[desc release];

QSGTexture *wrapper = m_window->createTextureFromNativeObject(QQuickWindow::NativeObjectTexture,
                                                              &m_texture,
                                                              0,
                                                              m_size);

qDebug() << "Got QSGTexture wrapper" << wrapper << "for an MTLTexture of size" << m_size;
```

```
QSGRendererInterface *rif = m_window->rendererInterface();
m_device = (id<MTLDevice>) rif->getResource(m_window, QSGRendererInterface::DeviceResource);
Q_ASSERT(m_device);

MTLTextureDescriptor *desc = [[MTLTextureDescriptor alloc] init];
desc.textureType = MTLTextureType2D;
```

```
~/qtdeclarative_dev/examples/quick/scenegraph $ ls
customgeometry          openglunderqml          textureinthread
d3d11underqml           rendernode              threadedanimation
fboitem                 scenegraph.pro          twotextureproviders
graph                   sgengine                vulkantextureimport
metaltextureimport      shared                  vulkanunderqml
metalunderqml           simplematerial
```

```
[desc release];

QSGTexture *wrapper = m_window->createTextureFrom
                                                    m_size);

qDebug() << "Got QSGTexture wrapper" << wrapper << "for an MTLTexture of size" << m_size;
```

Qt 5.15 now has equivalent Vulkan examples for both cases

Thank You

- https://doc-snapshots.qt.io/qt5-dev/qtquick-visualcanvas-scenegraph-renderer.html#rendering-via-the-qt-rendering-hardware-interface
- https://www.qt.io/blog/qt-quick-on-vulkan-metal-direct3d
- https://www.qt.io/blog/qt-quick-on-vulkan-metal-and-direct3d-part-2
- https://www.qt.io/blog/qt-quick-on-vulkan-metal-and-direct3d-part-3