



# Qt Graphics Stack Evolution

## Qt Quick on Vulkan, Metal, Direct 3D, and OpenGL

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

05/11/2019

Qt World Summit 2019

Berlin, Germany

# Contents

› What?

› Why?

› How?

- 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.



- 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.

- 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.



# Demo

Qt

Why?

Qt

# 1. Qt Everywhere



# 2. Qt (Quick) application + external rendering

### 3. Shader pipeline renewal

## 4. Possible performance gains

+1. Enable new stuff  
(for example, compute)

Qt

How?



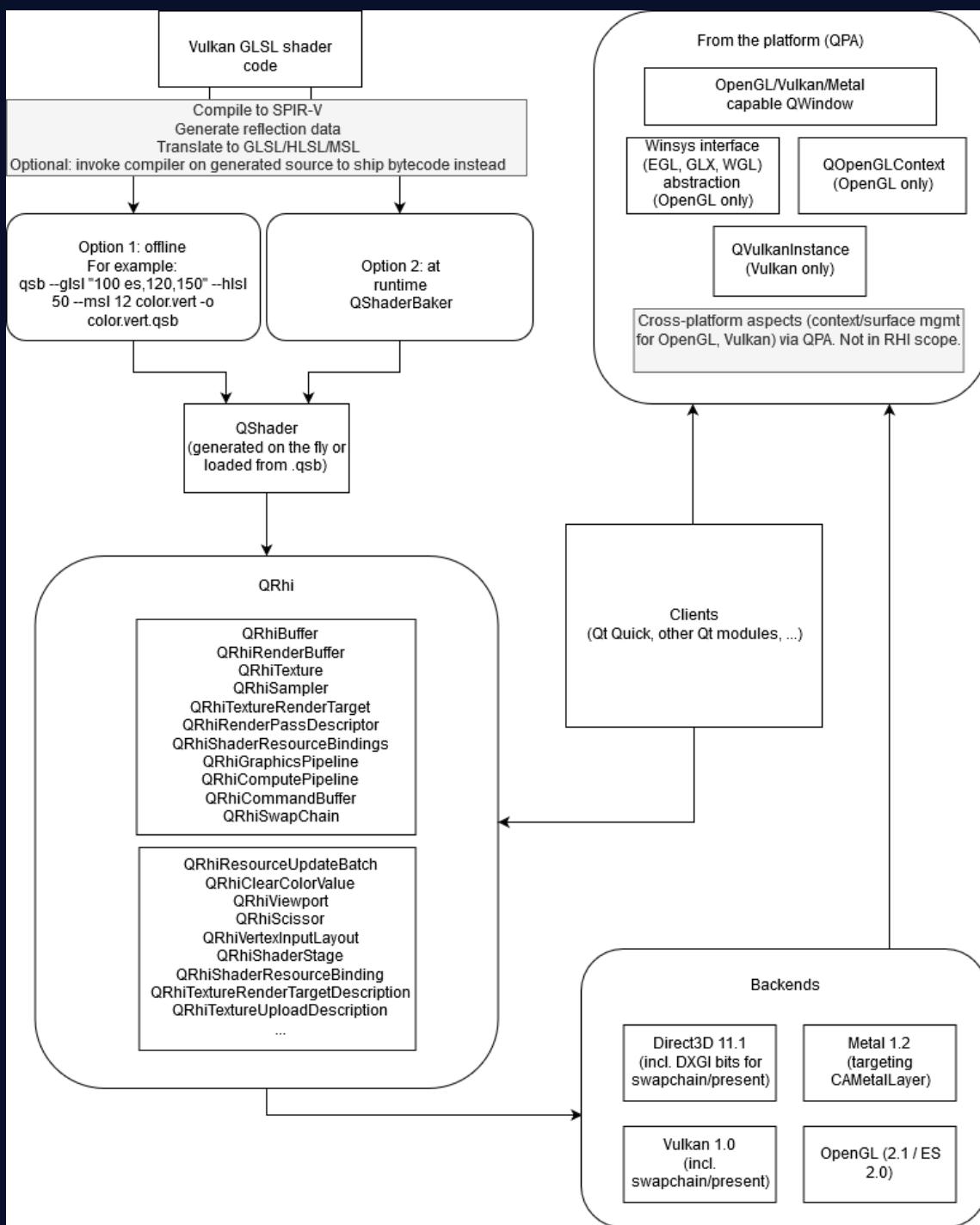
# Components

- › Qt RHI
  - › Part of the `QtGui` module
  - › **Private API**
- › Qt Shader Tools
  - › `qt-labs/qtshadertools`
- › Qt Quick port
  - › Side-by-side



# Components

- › Qt RHI
  - › Part of the `QtGui` module
  - › **Private API**
- › Qt Shader Tools
  - › [qt-labs/qtshadertools](#)
- › Qt Quick port
  - › Side-by-side



# QRhi

- QRhiBuffer
- QRhiRenderBuffer
- QRhiTexture
- QRhiSampler
- QRhiTextureRenderTarget
- QRhiRenderPassDescriptor
- QRhiShaderResourceBindings
- QRhiGraphicsPipeline
- QRhiComputePipeline
- QRhiCommandBuffer
- QRhiSwapChain

- QRhiResourceUpdateBatch
- QRhiClearColorValue
- QRhiViewport
- QRhiScissor
- QRhiVertexInputLayout
- QRhiShaderStage
- QRhiShaderResourceBinding
- QRhiTextureRenderTargetDescription
- QRhiTextureUploadDescription
- ...

## Backends

- Direct3D 11.1  
(incl. DXGI bits for swapchain/present)

- Metal 1.2  
(targeting CAMetalLayer)

- Vulkan 1.0  
(incl. swapchain/present)

- OpenGL (2.1 / ES 2.0)

Qt

From the platform (QPA)

OpenGL/Vulkan/Metal  
capable QWindow

Winsys interface  
(EGL, GLX, WGL)  
abstraction  
(OpenGL only)

QOpenGLContext  
(OpenGL only)

QVulkanInstance  
(Vulkan only)

Cross-platform aspects (context/surface mgmt  
for OpenGL, Vulkan) via QPA. Not in RHI scope.

# Qt

QtShaderTools

QML (Qt Quick) application

QtQuick + scenegraph

QtGui

(QGuiApplication, QWindow,  
QOpenGLContext, QPainter, ...)

QtRHI

QRhi backends

OpenGL

Vulkan

Metal

Direct 3D

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, ...

Qt

Can we do a triangle in 50 lines?

```
// window is an exposed QWindow
```

```
// vertexShaderPack and fragmentShaderPack are the contents of .qsb files
```

```
QRhiSwapChain *swapChain = rhi->newSwapChain();
swapChain->setWindow(window);
QRhiRenderPassDescriptor *rpDesc = swapChain->newCompatibleRenderPassDescriptor();
swapChain->setRenderPassDescriptor(rpDesc);
swapChain->buildOrResize();
```

```
QRhiResourceUpdateBatch *updates = rhi->nextResourceUpdateBatch();
```

```
static const float vertices[] = {
```

```
    -1.0f, -1.0f,
```

```
    1.0f, -1.0f,
```

```
    0.0f, 1.0f
```

```
};
```

```
QRhiBuffer *vbuf = rhi->newBuffer(QRhiBuffer::Immutable, QRhiBuffer::VertexBuffer, sizeof(vertices));
```

```
vbuf->build();
```

```
updates->uploadStaticBuffer(vbuf, vertices);
```

```
QHiShaderResourceBindings *srh = rhi->newShaderResourceBindings();
srh->build();

QHiGraphicsPipeline *pipeline = rhi->newGraphicsPipeline();
const QShader vs = QShader::fromSerialized(vertexShaderPack);
const QShader fs = QShader::fromSerialized(fragmentShaderPack);
pipeline->setShaderStages({ { QHiShaderStage::Vertex, vs }, { QHiShaderStage::Fragment, fs } });
QHiVertexInputLayout inputLayout;
inputLayout.setBindings({ { 2 * sizeof(float) } });
inputLayout.setAttributes({ { 0, 0, QHiVertexInputAttribute::Float2, 0 } });
pipeline->setVertexInputLayout(inputLayout);
pipeline->setShaderResourceBindings(srh);
pipeline->setRenderPassDescriptor(rpDesc);
pipeline->build();
```

```
rhi->beginFrame(swapChain);
QRhiCommandBuffer *cb = swapChain->currentFrameCommandBuffer();
QRhiRenderTarget *rt = swapChain->currentFrameRenderTarget();
const QSize outputSize = rt->pixelSize();

cb->beginPass(rt, Qt::green, { 1.0f, 0 }, updates);
cb->setGraphicsPipeline(pipeline);
cb->setViewport({ 0, 0, outputSize.width(), outputSize.height() });
QRhiCommandBuffer::VertexInput vbindings(vbuf, 0);
cb->setVertexInput(0, 1, &vbindings);
cb->draw(3);
cb->endPass();

rhi->endFrame(swapChain);
```



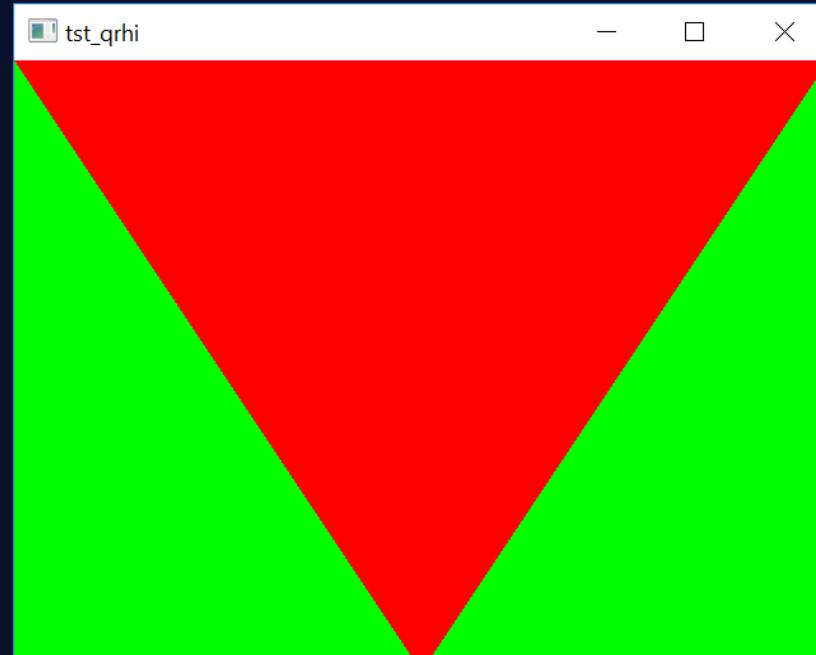
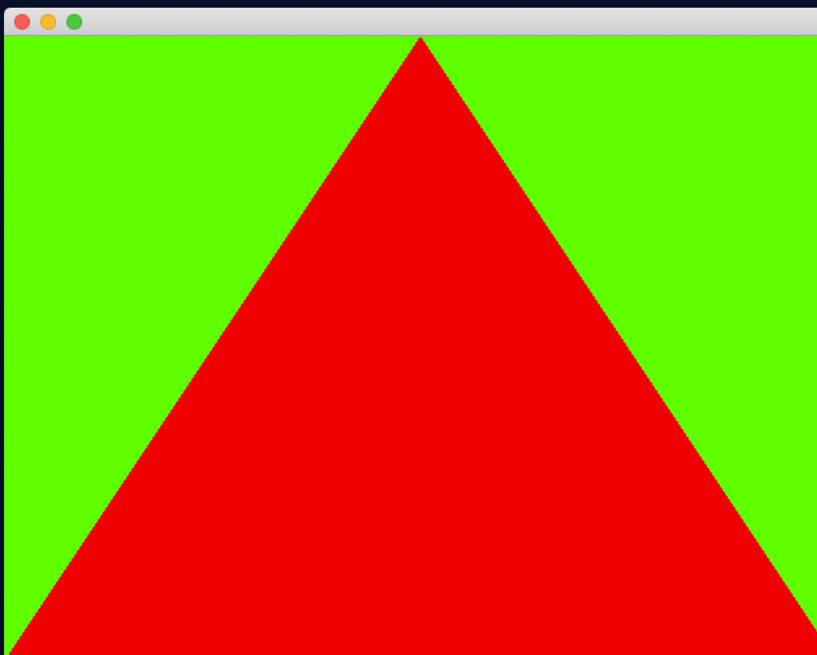
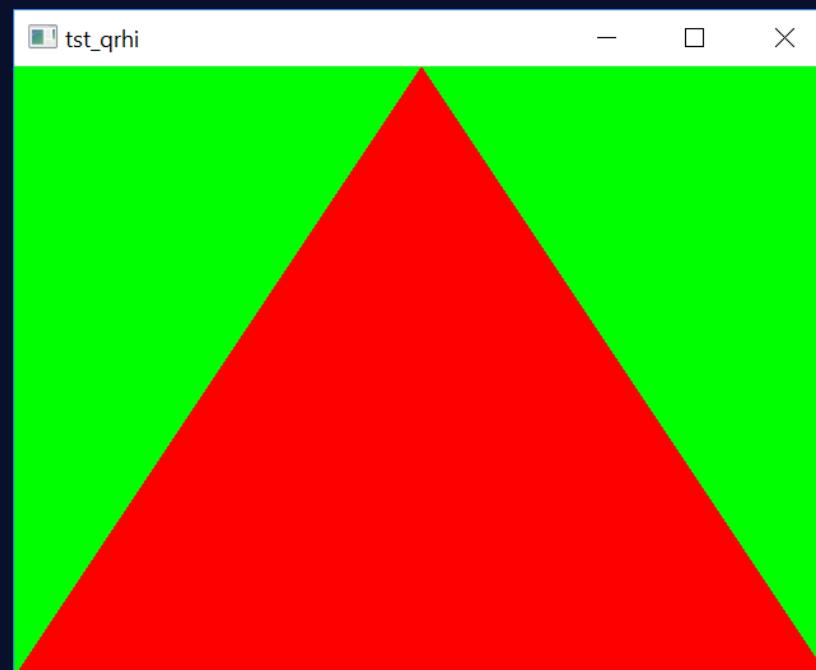
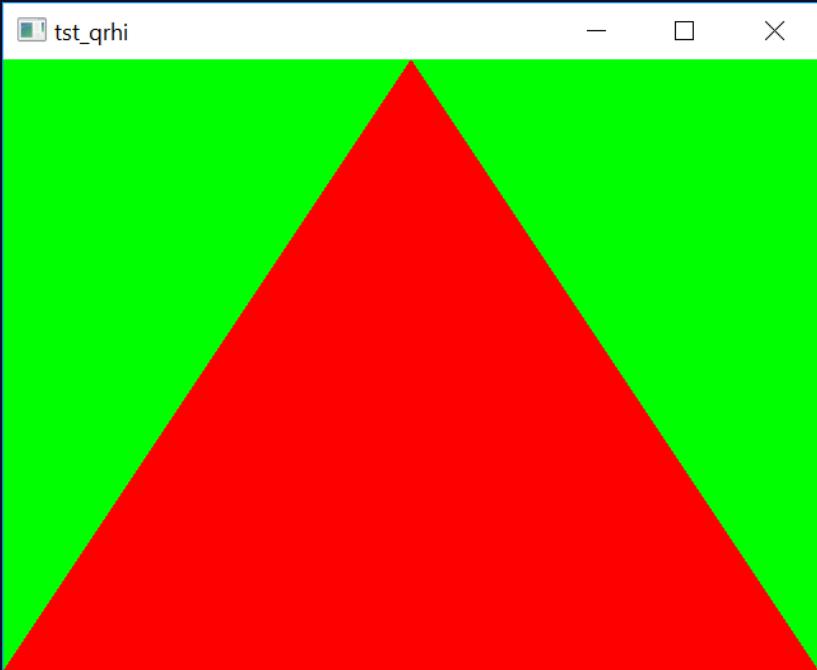
## Assuming these vertex and fragment shaders...

```
#version 440
layout(location = 0) in vec4 position;
out gl_PerVertex { vec4 gl_Position; };
void main()
{
    gl_Position = position;
}
```

```
#version 440
layout(location = 0) out vec4 fragColor;
void main()
{
    fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

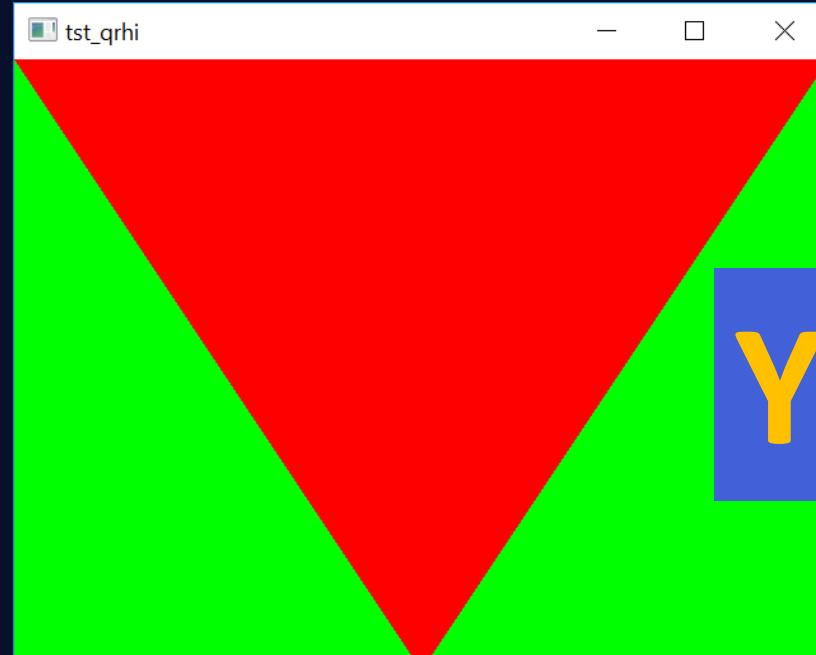
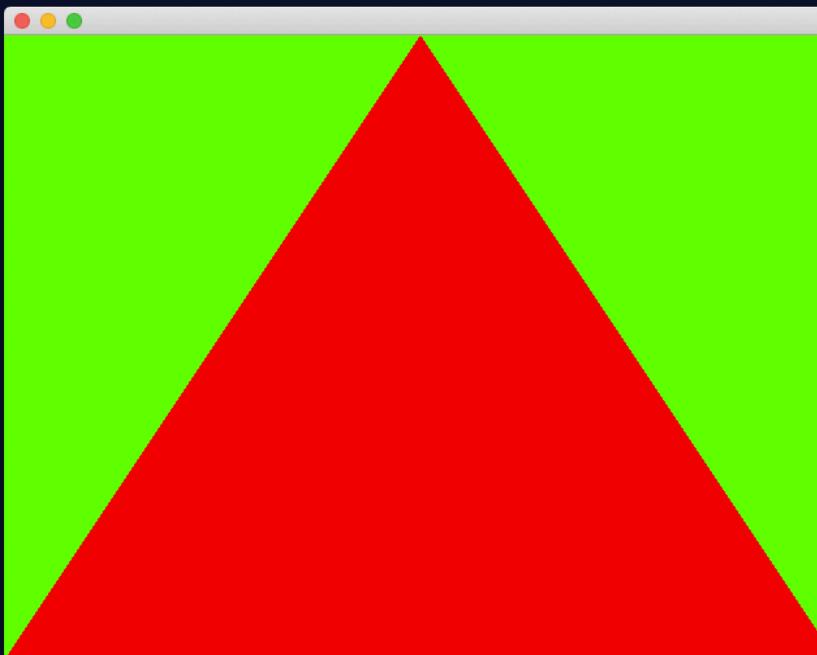
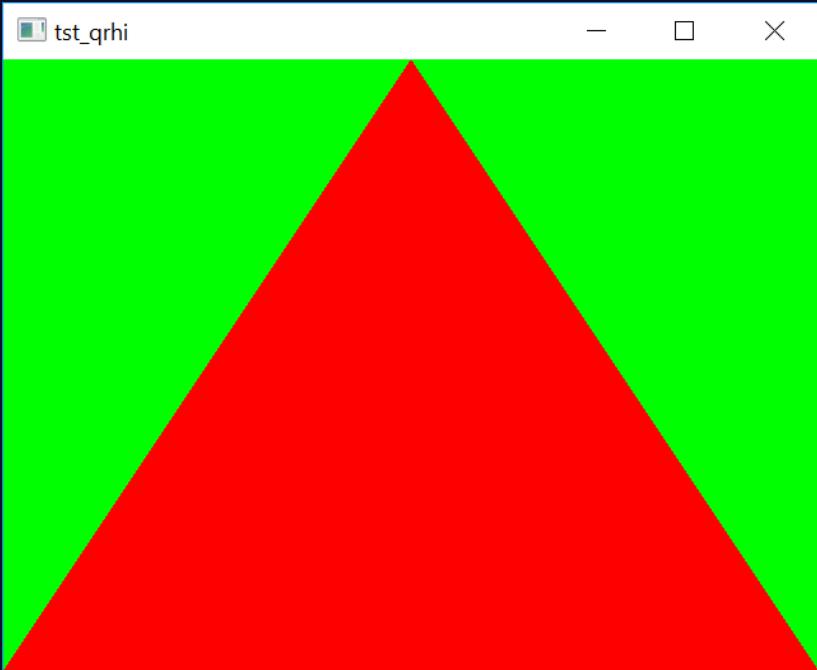
Qt

...this is  
what  
we get.  
Yay!

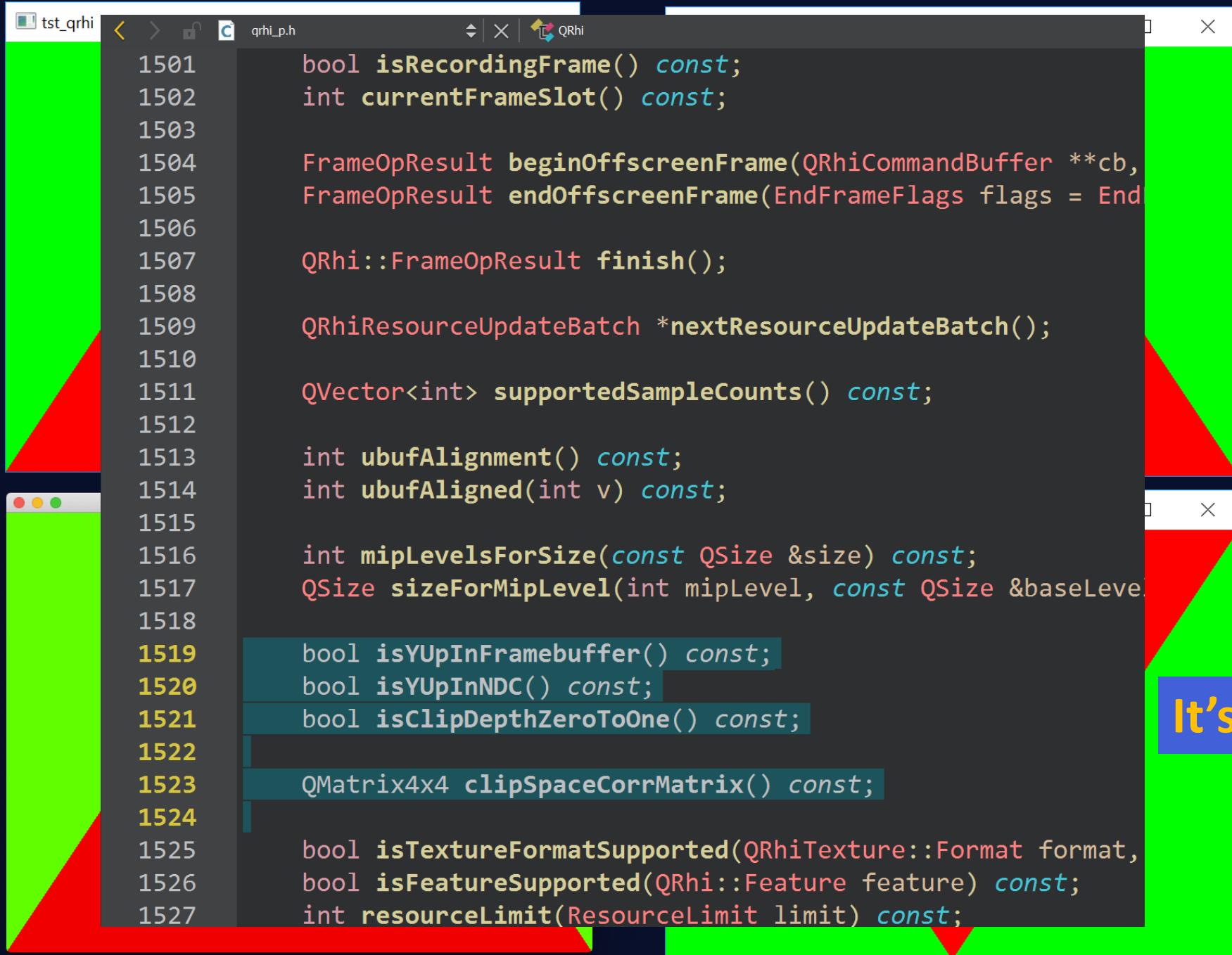


Qt

...this is  
what  
we get.  
Yay!



...this is  
what  
we get.  
Yay!

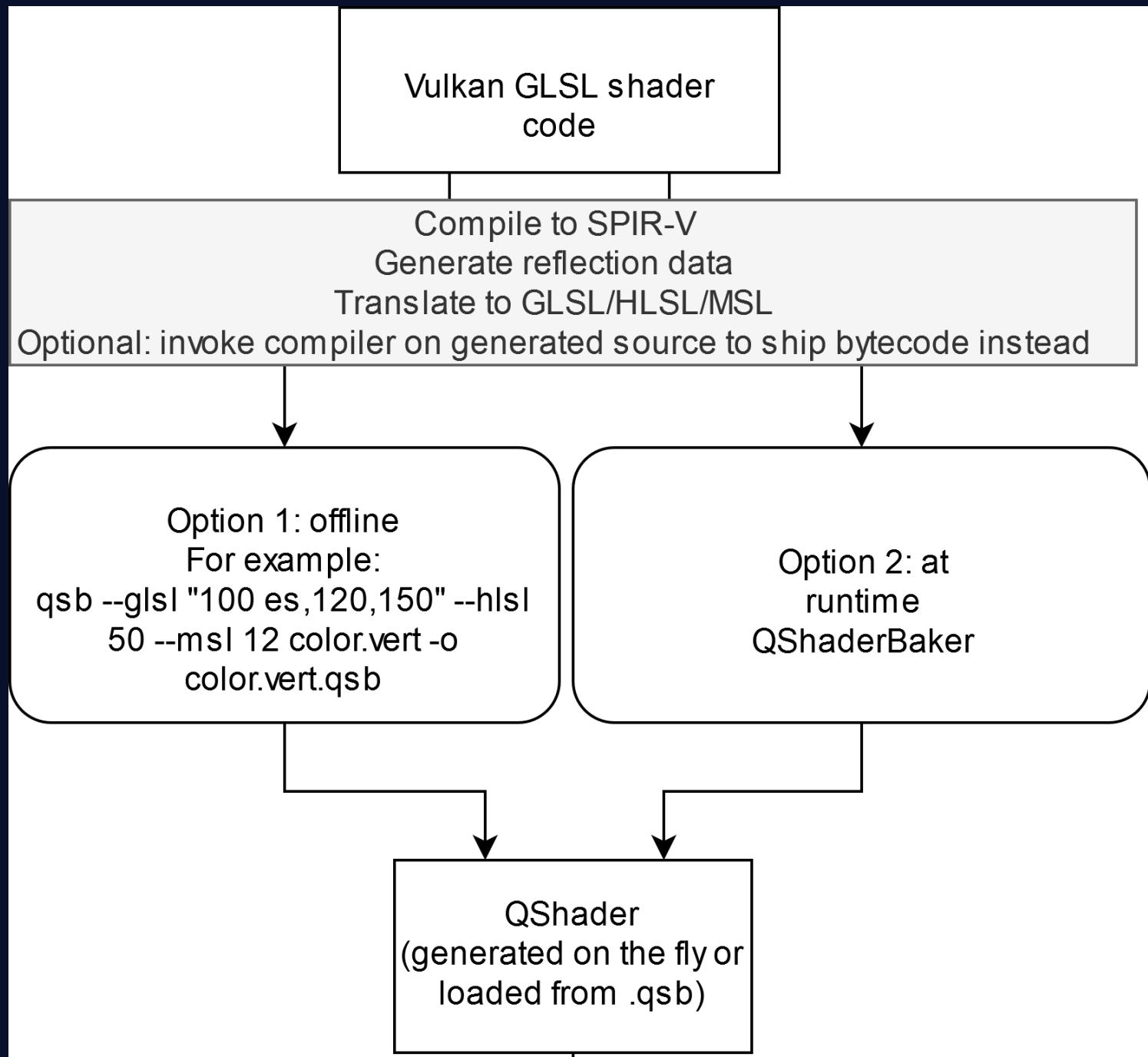


A screenshot of a terminal window titled "tst\_qrhi". The window displays a block of C++ code from "qrhi\_p.h". The code includes several methods from the "QRhi" class, such as "isRecordingFrame()", "beginOffscreenFrame()", "endOffscreenFrame()", "finish()", "nextResourceUpdateBatch()", "supportedSampleCounts()", "ubufAlignment()", "ubufAligned()", "mipLevelsForSize()", "sizeForMipLevel()", "isYUpInFramebuffer()", "isYUpInNDC()", "isClipDepthZeroToOne()", "clipSpaceCorrMatrix()", "isTextureFormatSupported()", "isFeatureSupported()", and "resourceLimit()". The code is color-coded with syntax highlighting. The background of the terminal window is dark, and the code is displayed in white and light blue text. The window has a standard OS X-style title bar with icons for minimizing, maximizing, and closing.

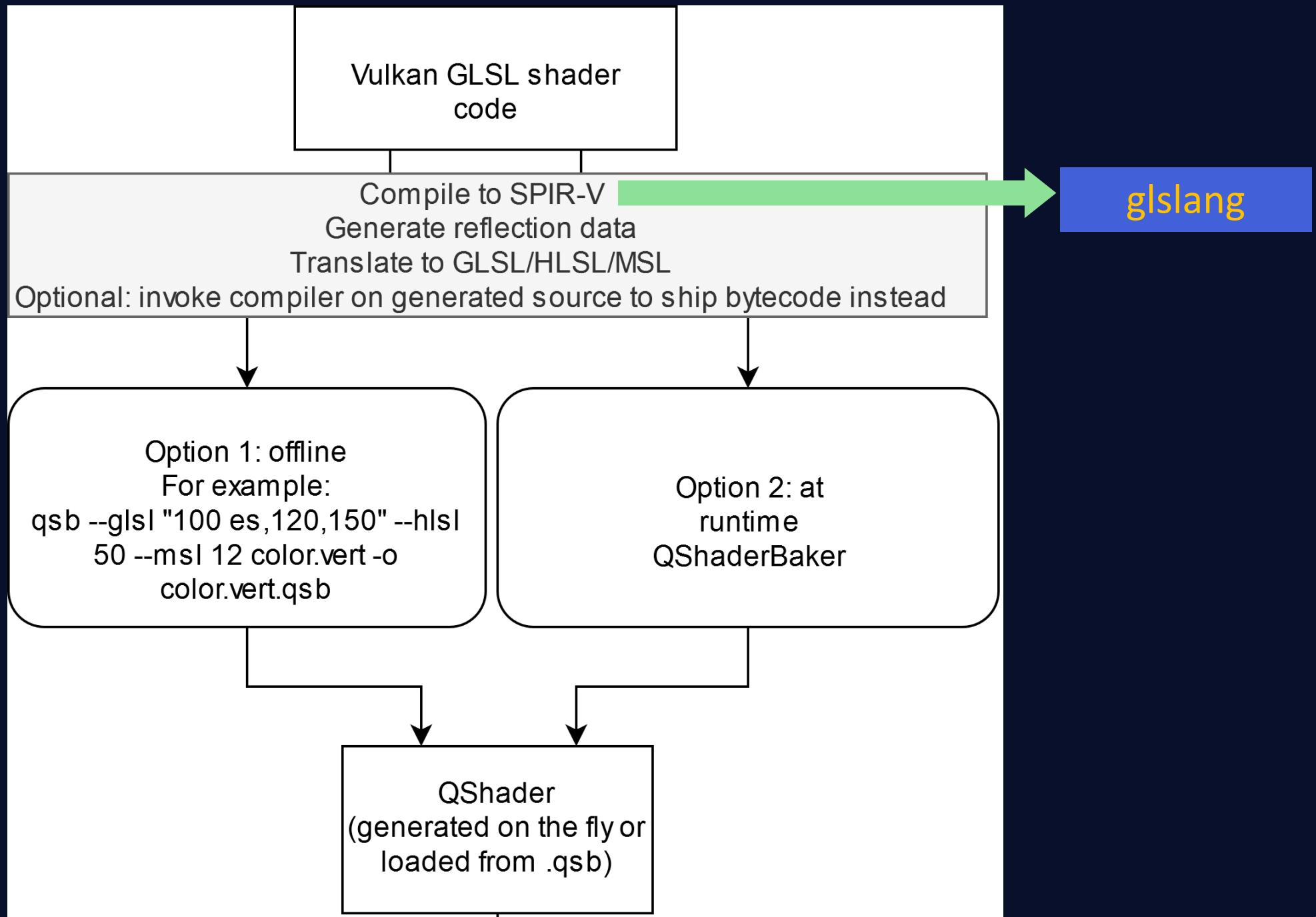
```
1501     bool isRecordingFrame() const;
1502     int currentFrameSlot() const;
1503
1504     FrameOpResult beginOffscreenFrame(QRhiCommandBuffer **cb,
1505                                         FrameOpResult endOffscreenFrame(EndFrameFlags flags = EndFrameFlagNone);
1506
1507     QRhi::FrameOpResult finish();
1508
1509     QRhiResourceUpdateBatch *nextResourceUpdateBatch();
1510
1511     QVector<int> supportedSampleCounts() const;
1512
1513     int ubufAlignment() const;
1514     int ubufAligned(int v) const;
1515
1516     int mipLevelsForSize(const QSize &size) const;
1517     QSize sizeForMipLevel(int mipLevel, const QSize &baseLevel) const;
1518
1519     bool isYUpInFramebuffer() const;
1520     bool isYUpInNDC() const;
1521     bool isClipDepthZeroToOne() const;
1522
1523     QMatrix4x4 clipSpaceCorrMatrix() const;
1524
1525     bool isTextureFormatSupported(QRhiTexture::Format format,
1526                                 bool isFeatureSupported(QRhi::Feature feature) const;
1527     int resourceLimit(ResourceLimit limit) const;
```

It's fine.

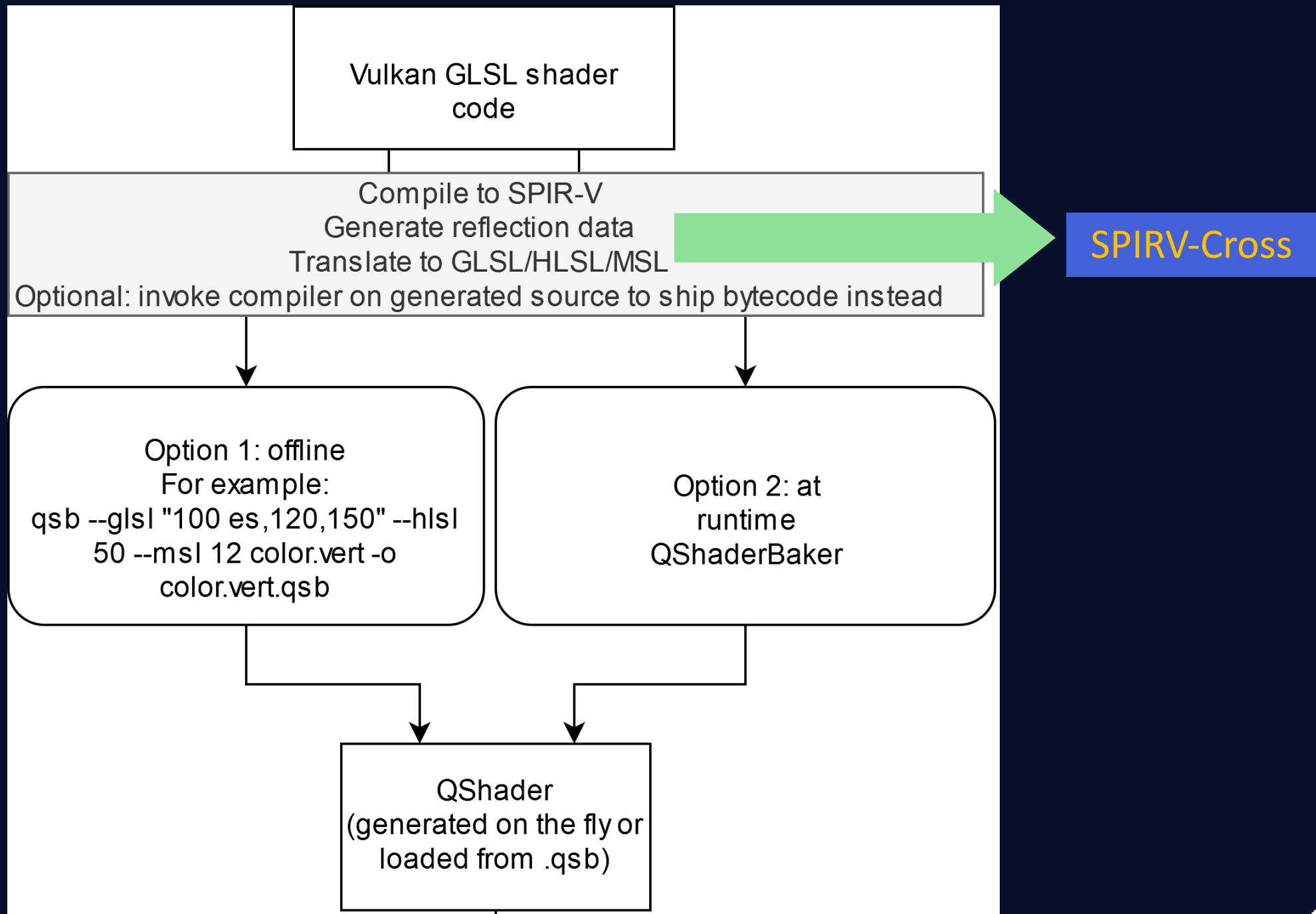
Qt



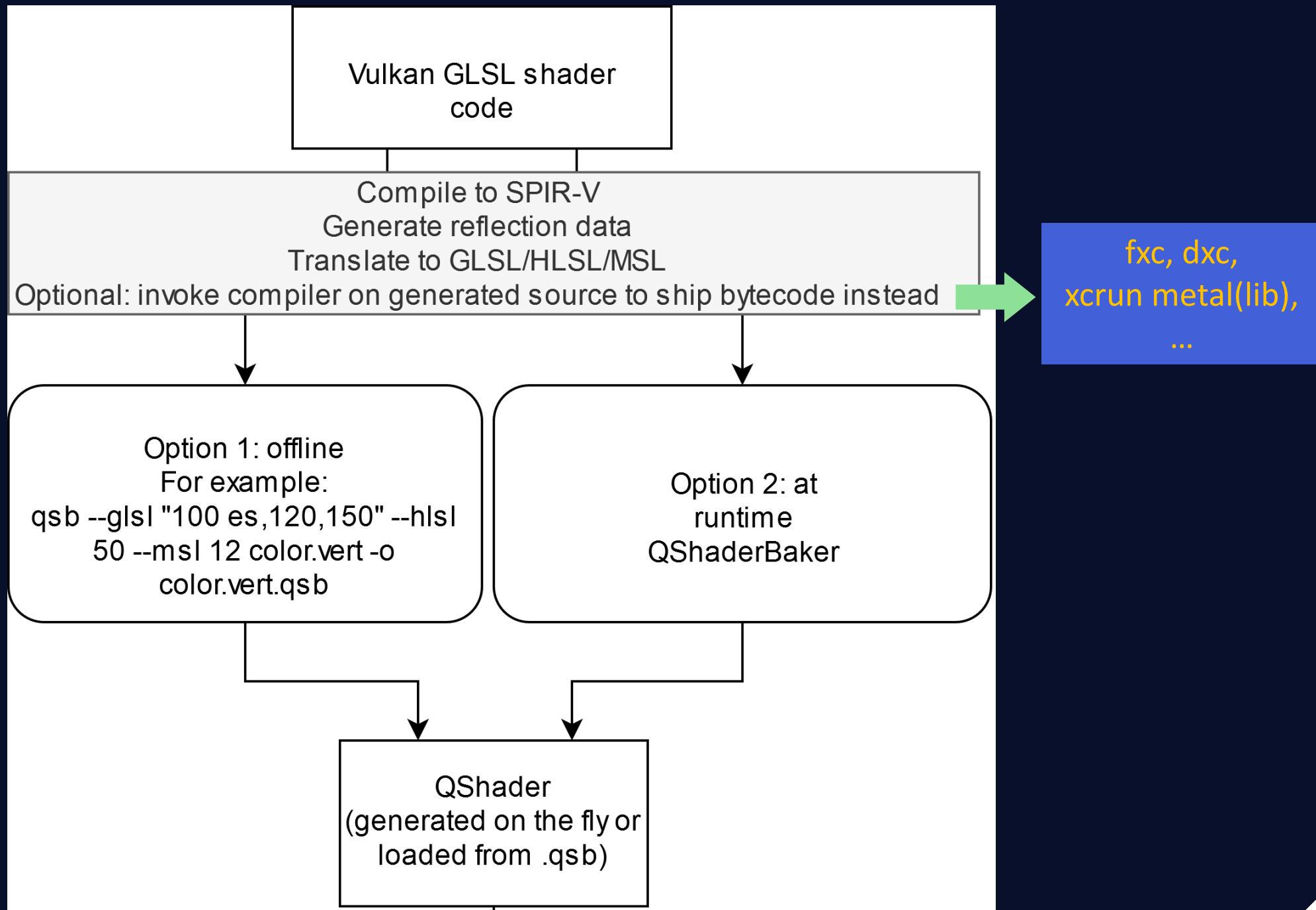
Qt



Qt

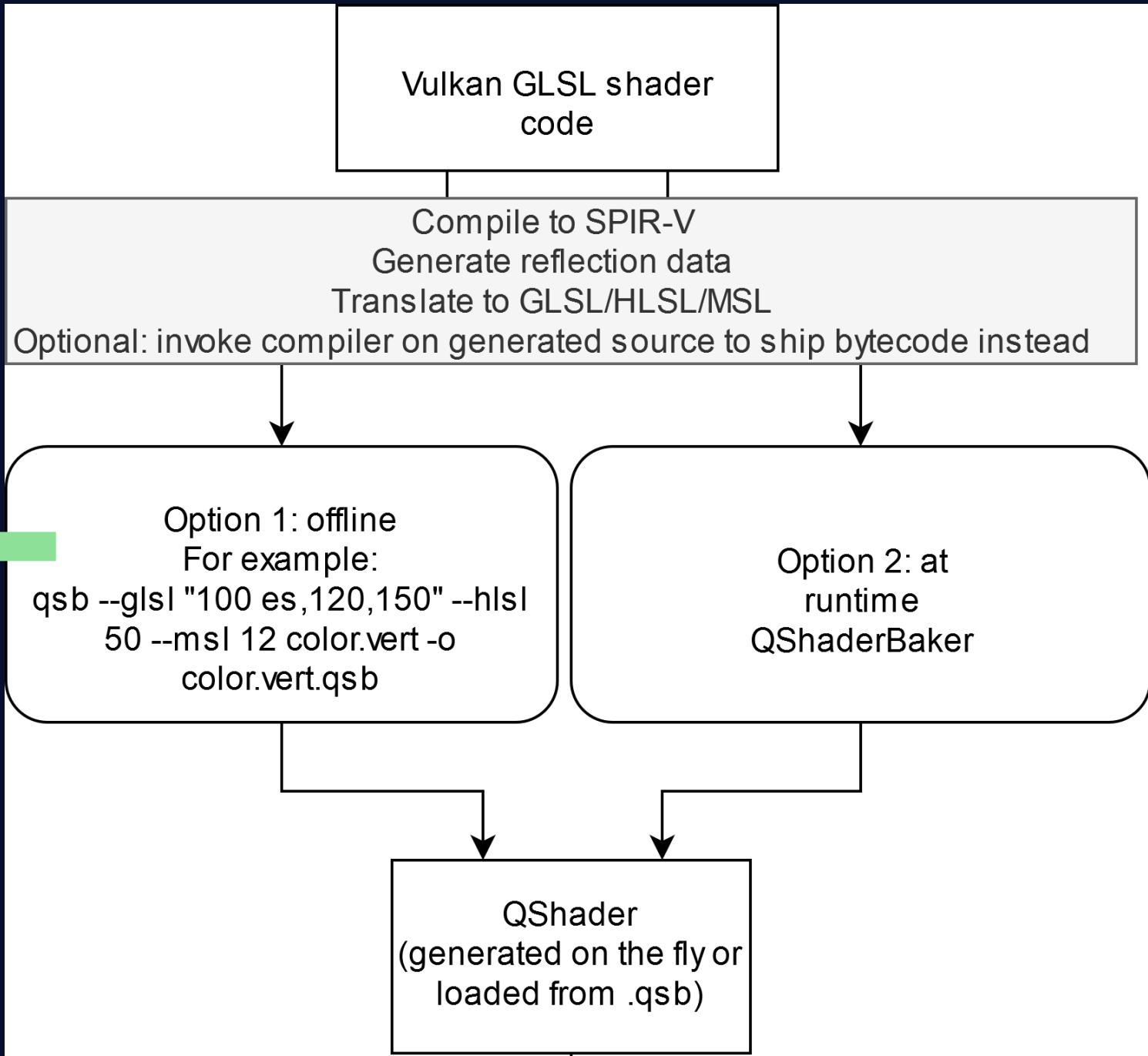


Qt



Qt

To be integrated  
with the build  
system in 6.x



```

#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": [

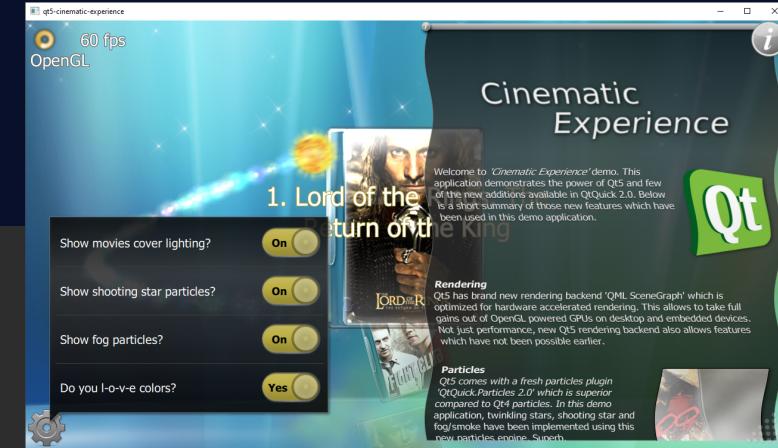
{  
"binding": 0,  
"blockName": "buf",  
"members": [  
 {  
 "matrixStride": 16,  
 "name": "matrix",  
 "offset": 0,  
 "size": 64,  
 "type": "mat4"  
 },  
 {  
 "name": "color",  
 "offset": 64,  
 "size": 16,  
 "type": "vec4"  
 },  
 {  
 "name": "textureScale",  
 "offset": 80,  
 "size": 8,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

# Qt Quick

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



```
[ ] 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 xl= 0 yl= 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 xl= 0 yl= 0 x2= 256 y2= 256 materialtype= 0x7fe4c166ad32 ) "internalimage" 1000001 order 0
[ ] TransformNode( 0x7fe4881adaef0 det= 0.444444 "QQuickItem(DelegateItem_QMLTYPE_0:)" ) 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
        }
    }
}
```



File Window Tools Help

Timeline - Frame #434

EID:	20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400	420	440	460	480	500	520	540	560	580	606	620	640	660	680	700	720	740	760	780					
	+ 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 (▲)

&lt;

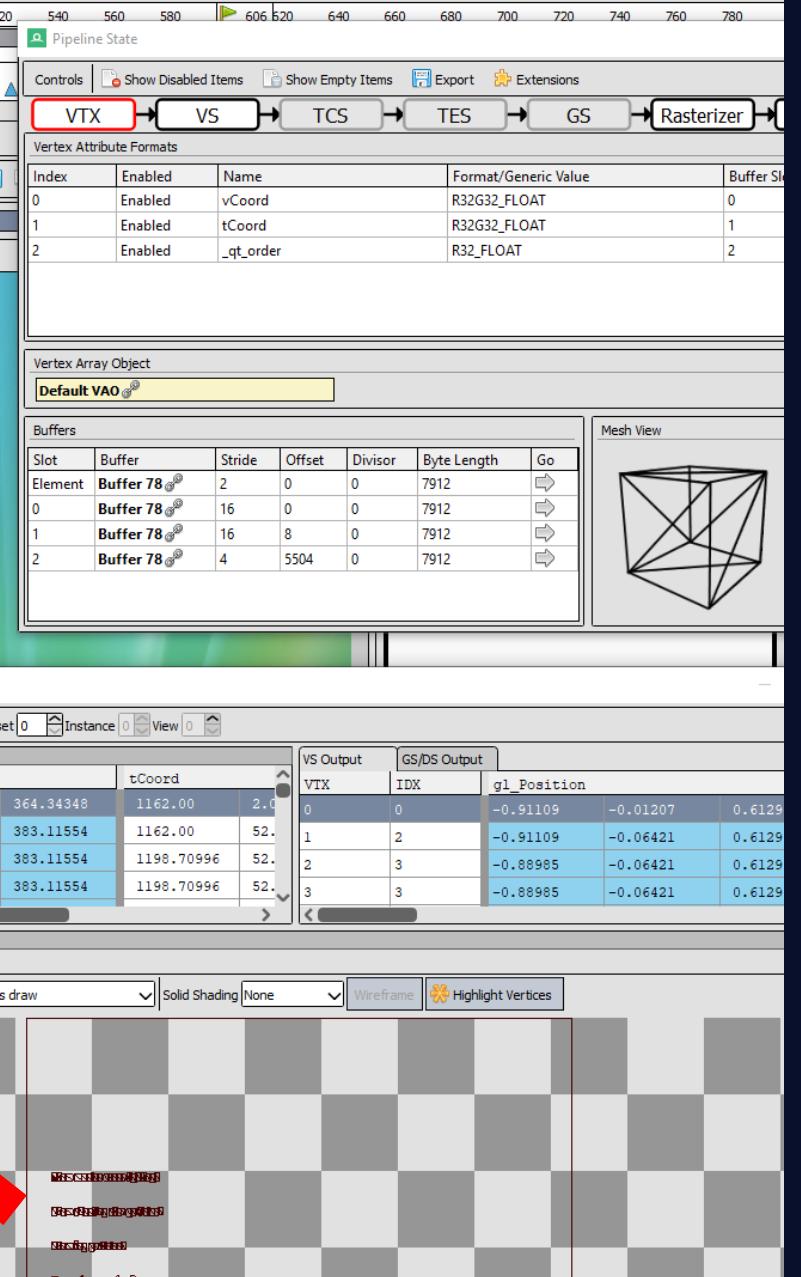
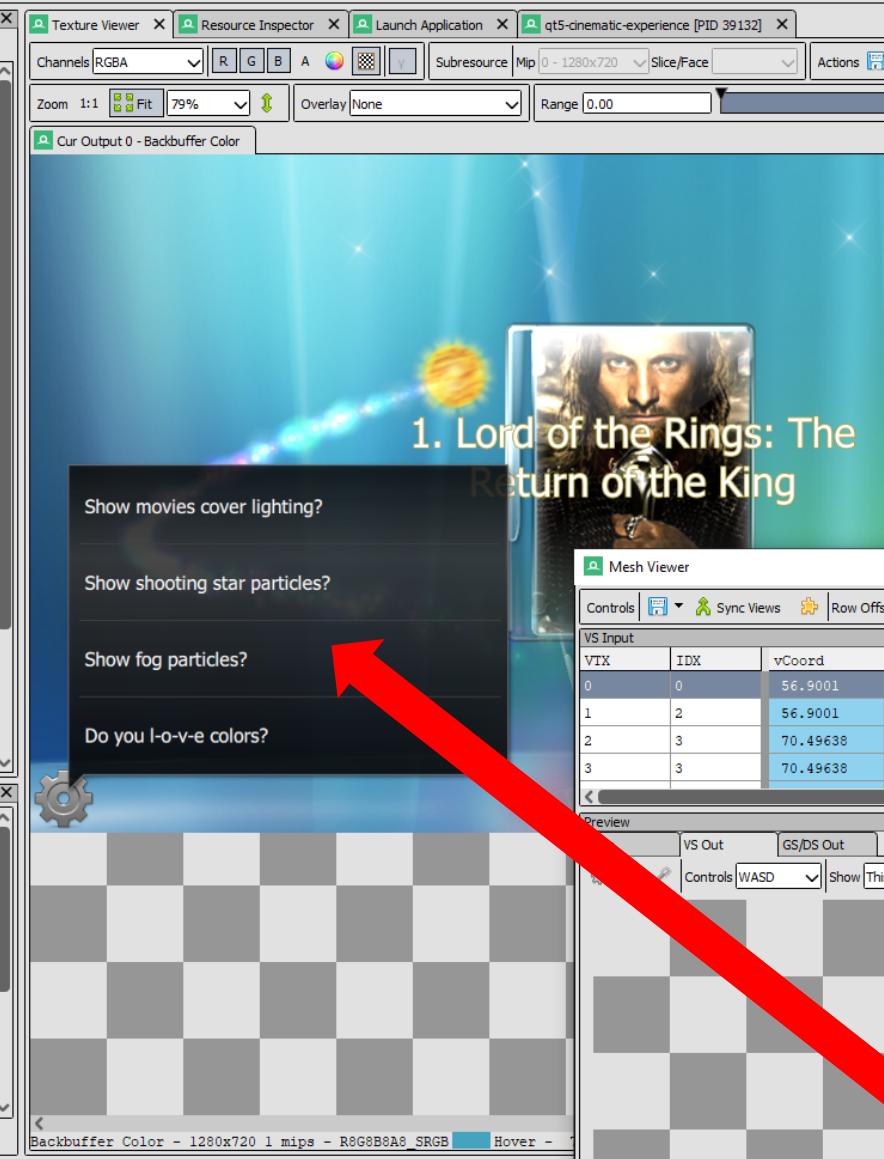
Event Browser

Controls	EID	Name
	0	Frame Start
	18-103	Colour Pass #1 (1 Targets + Depth)
	132-213	Colour Pass #2 (1 Targets + Depth)
	242-323	Colour Pass #3 (1 Targets + Depth)
	242	glClear(Color = <0.000000, 0.000000, 0.000000, 0.000000>, Depth = <1.00...
	259	glDrawElements(4)
	278	glDrawElements(192)
	297	glDrawElements(12)
	309	glDrawElements(18)
	323	glDrawElements(4)
	390-782	Colour Pass #4 (1 Targets + Depth)
	390	glClear(Color = <1.000000, 1.000000, 1.000000>, Depth = <1.00...
	401	glDrawElements(16)
	414	glDrawElements(4)
	436	glDrawElements(120)
	458	glDrawElements(6)
	475	glDrawElements(6)
	492	glDrawElements(6)
	509	glDrawElements(210)
	523	glDrawElements(6)
	542	glDrawElements(1200)
	558	glDrawElements(264)
	574	glDrawElements(4)
	588	glDrawElements(54)
	606	glDrawElements(516)
	624	glDrawElements(6)
	638	glDrawElements(4)
	655	glDrawElements(6)
	669	glDrawElements(4)
	686	glDrawElements(6)
	700	glDrawElements(4)

API Inspector

EID	Event
> 589	glBindBuffer
> 590	glBindBuffer
> 591	glBindBuffer
> 592	glBindBuffer
> 593	glUseProgram
> 594	glBlendFunc
> 595	glBlendColor
> 596	glUniform1fv
> 597	glUniform4fv
> 598	glUniformMatrix4fv
> 599	glUniform1fv

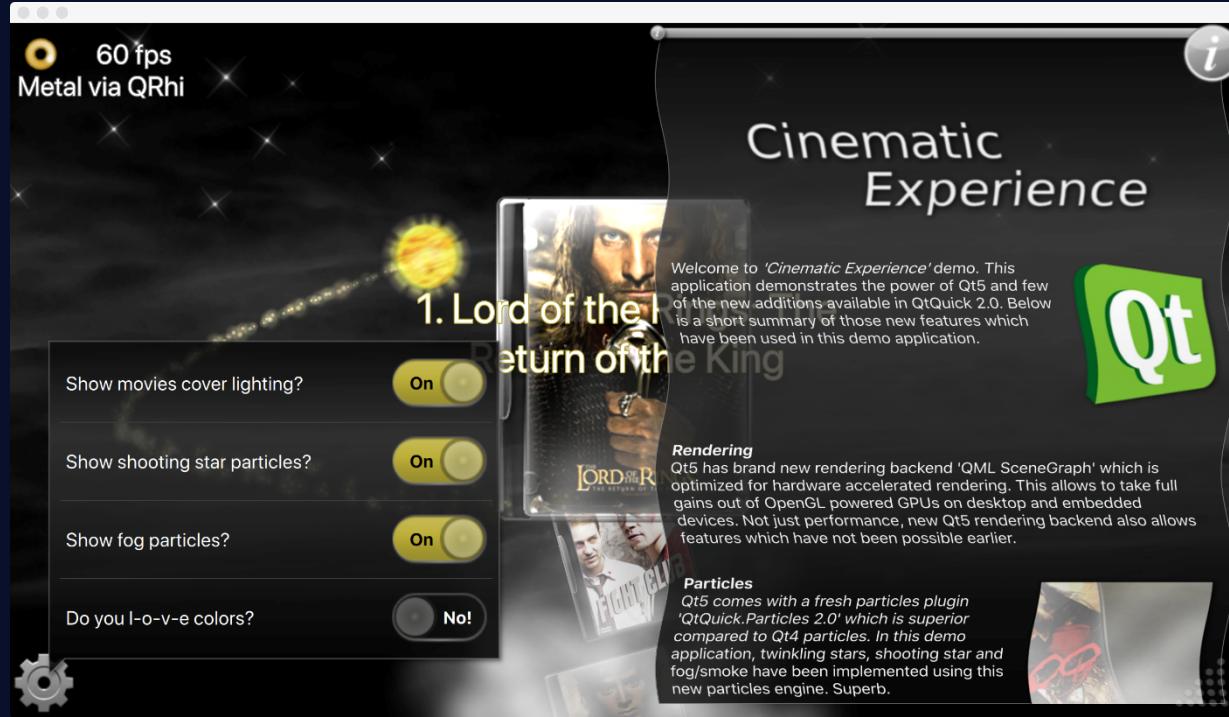
Callstack



Qt

# Qt Quick

- › QQuickItem tree
  - › described in QML
  - › main thread
- › QSGNode tree + material system
  - › the “scene graph”
  - › render thread
- › Vulkan, Metal, Direct 3D, OpenGL



Qt

# Materials

```
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();
```

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;

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

    if (state.matrixDirty())
        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();
```

Q

```
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);
```

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
};
```

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 OSCMaterialDbiShader
{
public:
    class Q_QUICK_EXPORT OSCMaterialDbiShaderPrivate;
public:
    inline bool update();
    ...
    QMatrix4x4 transform();
    QByteArray vertexCode();
    ...
};

enum Flag {
    UpdatesGraph,
    ...
};

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;
};

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

## A material (be

```
virtual bool up
```

```
virtual void up
```

```
virtual bool up
```

```
void setFlag(Fl
```

```
// filename is
```

, and only data.

```
oldMaterial);
```

```
ture **texture,
oldMaterial);
```

```
ipelineState *ps,
aterial *oldMaterial);
```

```
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);
}
```

Qt

# ShaderEffect

Qt



```
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;
                }"
        }
    }
}
```

Qt



```
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 {
    id: shaderItem

    fragmentShader: "qrc:/qt-project.org/imports/QtQuick/Controls.2/Material/shaders/RectangularGlow.frag"

    x: (parent.width - width) / 2.0
    y: (parent.height - height) / 2.0
    width: parent.width + rootItem.glowRadius * 2 + cornerRadius * 2
    height: parent.height + rootItem.glowRadius * 2 + cornerRadius * 2

    function clampedCornerRadius() {
        var maxCornerRadius = Math.min(rootItem.width, rootItem.height) / 2 + rootItem.glowRadius;
        return Math.max(0, Math.min(rootItem.cornerRadius, maxCornerRadius))
    }

    property color color: rootItem.color
    property real inverseSpread: 1.0 - rootItem.spread
    property real relativeSizeX: ((inverseSpread * inverseSpread) * rootItem.glowRadius + cornerRadius * 2.
    property real relativeSizeY: relativeSizeX * (width / height)
    property real spread: rootItem.spread / 2.0
    property real cornerRadius: clampedCornerRadius()
}
```

```
ShaderEffect {
    id: shaderItem

    fragmentShader: "qrc:/qt-project.org/imports/QtQuick/Controls.2/Material/shaders/RectangularGlow.frag"
        ~/qtquickcontrols2_dev/src/imports/controls/material/shaders $ ls -lR
    x: (parent) total 8
    y: (parent) drwxr-xr-x  3 agocs  staff   96 Jul 10  2018 +glslcore
    width: parent drwxr-xr-x  3 agocs  staff   96 Aug 17 15:31 +qsb
    height: parent -rw-r--r--  1 agocs  staff  660 Jul 10  2018 RectangularGlow.frag

    function createShader() {
        var material = new THREE.ShaderMaterial({
            vertexShader: require("./+glslcore"),
            fragmentShader: require("./+qsb")
        });
        return material;
    }
}

property real cornerRadius: cornerRadius * 2.0
property real relativeSizeY: relativeSizeX * (width / height)
property real spread: rootItem.spread / 2.0
property real cornerRadius: clampedCornerRadius()
}
```

Qt

# Integrating custom Vulkan/Metal/D3D11/OpenGL rendering

Qt

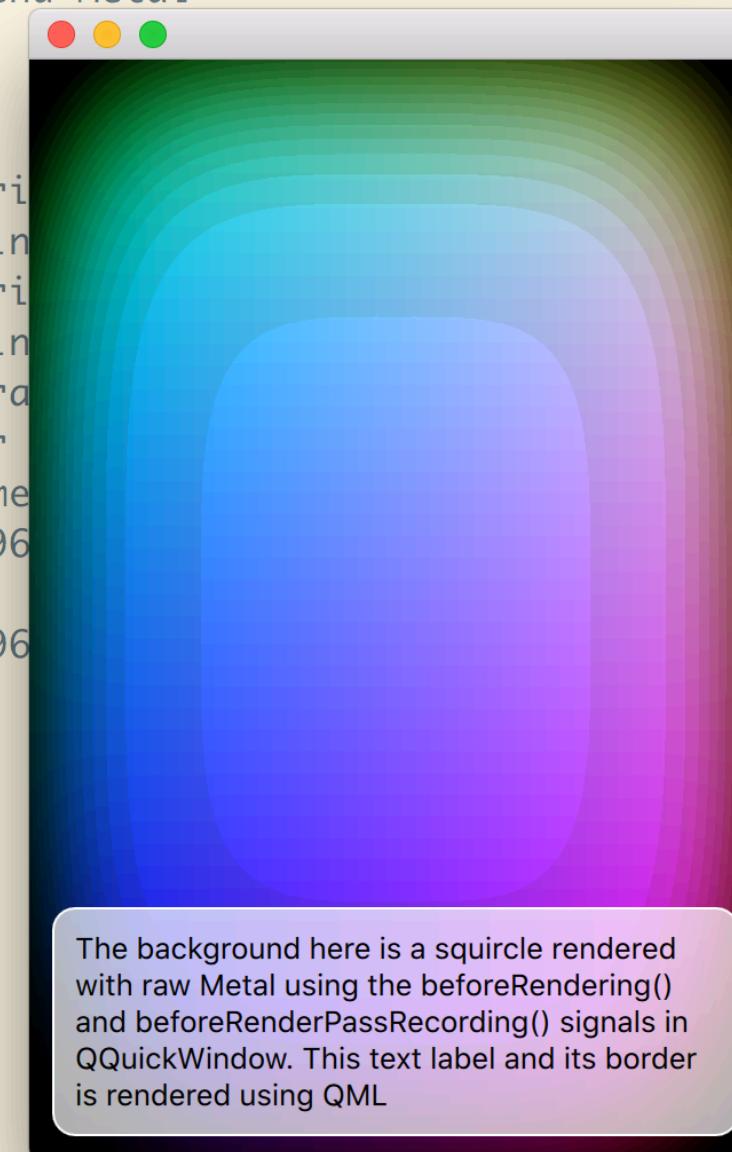
Underlay/overlay

Integrating custom  
Vulkan, Metal, D3D11, OpenGL

rendering QSGRenderNode

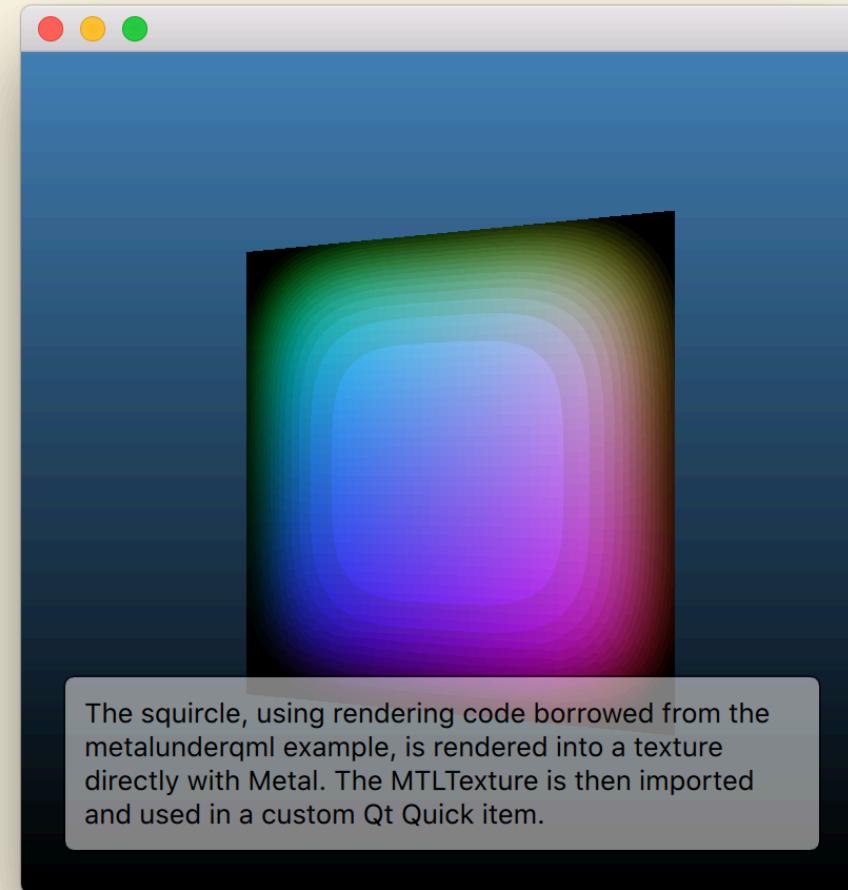
1. metalunderqml

```
~/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
qt.scenegraph.general: rhi texture atlas dime
qt.rhi.general: got CAMetalLayer, size 640x96
init
qt.rhi.general: got CAMetalLayer, size 640x96
□
```



1. metatextureimpo

```
~/qtdeclarative_dev/examples/quick/scenegraph/metatextureimport $ ./metatextureimport.app/Contents/MacOS/metatextureimpo  
rt  
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;
```

Key takeaway from porting the  
Qt Quick Renderer onto QRhi

# Qt

```
< > qsgbatchrenderer.cpp | X | Renderer::renderBatches() -> void  
4003  
4004     if (Q_LIKELY(renderOpaque)) {  
4005         for (int i=0; i<m_opaqueBatches.size(); ++i) {  
4006             Batch *b = m_opaqueBatches.at(i);  
4007             if (b->merged)  
4008                 renderMergedBatch(b);  
4009             else  
4010                 renderUnmergedBatch(b);  
4011         }  
4012     }  
4013  
4014     glEnable(GL_BLEND);  
4015     if (m_useDepthBuffer)  
4016         glDepthMask(false);  
4017     glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);  
4018  
4019     if (Q_LIKELY(renderAlpha)) {  
4020         for (int i=0; i<m_alphaBatches.size(); ++i) {  
4021             Batch *b = m_alphaBatches.at(i);  
4022             if (b->merged)  
4023                 renderMergedBatch(b);  
4024             else if (b->isRenderNode)  
4025                 renderRenderNode(b);  
4026             else  
4027                 renderUnmergedBatch(b);  
4028         }  
4029     }
```



## QHi-based code path

```
// depth test stays enabled but no need to write out depth from the
// transparent (back-to-front) pass
m_gstate.depthWrite = false;

QVarLengthArray<PreparedRenderBatch, 64> alphaRenderBatches;
if (Q_LIKELY(renderAlpha)) {
    for (int i = 0, ie = m_alphaBatches.size(); i != ie; ++i) {
        Batch *b = m_alphaBatches.at(i);
        PreparedRenderBatch renderBatch;
        bool ok;
        if (b->merged)
            ok = prepareRenderMergedBatch(b, &renderBatch);
        else if (b->isRenderNode)
            ok = prepareRhiRenderNode(b, &renderBatch);
        else
            ok = prepareRenderUnmergedBatch(b, &renderBatch);
        if (ok)
            alphaRenderBatches.append(renderBatch);
    }
}

if (m_visualizer->mode() != Visualizer::VisualizeNothing)
    m_visualizer->prepareVisualize();

QRhiCommandBuffer *cb = commandBuffer();
cb->beginPass(renderTarget(), m_pstate.clearColor, m_pstate.dsClear, m_resourceUpdates);
m_resourceUpdates = nullptr;

for (int i = 0, ie = opaqueRenderBatches.count(); i != ie; ++i) {
    PreparedRenderBatch *renderBatch = &opaqueRenderBatches[i];
    if (renderBatch->batch->merged)
        renderMergedBatch(renderBatch);
    else
        renderUnmergedBatch(renderBatch);
}
```

# Qt

```
// depth test stays enabled but no need to write out depth from the
// transparent (back-to-front) pass
m_gstate.depthWrite = false;

QVarLengthArray<PreparedRenderBatch, 64> alphaRenderBatches;
if (Q_LIKELY(renderAlpha)) {
    for (int i = 0, ie = m_alphaBatches.size(); i != ie; ++i) {
        Batch *b = m_alphaBatches.at(i);
        PreparedRenderBatch renderBatch;
        bool ok;
        if (b->merged)
            ok = prepareRenderMergedBatch(b, &renderBatch);
        else if (b->isRenderNode)
            ok = prepareRhiRenderNode(b, &renderBatch);
        else
            ok = prepareRenderUnmergedBatch(b, &renderBatch);
        if (ok)
            alphaRenderBatches.append(renderBatch);
    }
}

if (m_visualizer->mode() != Visualizer::VisualizeNothing)
    m_visualizer->prepareVisualize();

QRhiCommandBuffer *cb = commandBuffer();
cb->beginPass(renderTarget(), m_pstate.clearColor, m_pstate.dsClear, m_resourceUpdates);
m_resourceUpdates = nullptr;

for (int i = 0, ie = opaqueRenderBatches.count(); i != ie; ++i) {
    PreparedRenderBatch *renderBatch = &opaqueRenderBatches[i];
    if (renderBatch->batch->merged)
        renderMergedBatch(renderBatch);
    else
        renderUnmergedBatch(renderBatch);
}
```

Prepare vertex, index,  
uniform buffers, shader  
res.binding and pipeline  
state objects.

Materials provide **data**, and  
only data. (no messing with  
QRhi or graphics API)

# Qt

```
// depth test stays enabled but no need to write out depth from the
// transparent (back-to-front) pass
m_gstate.depthWrite = false;

QVarLengthArray<PreparedRenderBatch, 64> alphaRenderBatches;
if (Q_LIKELY(renderAlpha)) {
    for (int i = 0, ie = m_alphaBatches.size(); i != ie; ++i) {
        Batch *b = m_alphaBatches.at(i);
        PreparedRenderBatch renderBatch;
        bool ok;
        if (b->merged)
            ok = prepareRenderMergedBatch(b, &renderBatch);
        else if (b->isRenderNode)
            ok = prepareRhiRenderNode(b, &renderBatch);
        else
            ok = prepareRenderUnmergedBatch(b, &renderBatch);
        if (ok)
            alphaRenderBatches.append(renderBatch);
    }
}

if (m_visualizer->mode() != Visualizer::VisualizeNothing)
    m_visualizer->prepareVisualize();

QRhiCommandBuffer *cb = commandBuffer();
cb->beginPass(renderTarget(), m_pstate.clearColor, m_pstate.dsClear, m_resourceUpdates);
m_resourceUpdates = nullptr;

for (int i = 0, ie = opaqueRenderBatches.count(); i != ie; ++i) {
    PreparedRenderBatch *renderBatch = &opaqueRenderBatches[i];
    if (renderBatch->batch->merged)
        renderMergedBatch(renderBatch);
    else
        renderUnmergedBatch(renderBatch);
}
```

Start the renderpass,  
clear color/depth/stencil.

# Qt

```
// depth test stays enabled but no need to write out depth from the
// transparent (back-to-front) pass
m_gstate.depthWrite = false;

QVarLengthArray<PreparedRenderBatch, 64> alphaRenderBatches;
if (Q_LIKELY(renderAlpha)) {
    for (int i = 0, ie = m_alphaBatches.size(); i != ie; ++i) {
        Batch *b = m_alphaBatches.at(i);
        PreparedRenderBatch renderBatch;
        bool ok;
        if (b->merged)
            ok = prepareRenderMergedBatch(b, &renderBatch);
        else if (b->isRenderNode)
            ok = prepareRhiRenderNode(b, &renderBatch);
        else
            ok = prepareRenderUnmergedBatch(b, &renderBatch);
        if (ok)
            alphaRenderBatches.append(renderBatch);
    }
}

if (m_visualizer->mode() != Visualizer::VisualizeNothing)
    m_visualizer->prepareVisualize();

QRhiCommandBuffer *cb = commandBuffer();
cb->beginPass(renderTarget(), m_pstate.clearColor, m_pstate.dsClear, m_resourceUpdates);
m_resourceUpdates = nullptr;

for (int i = 0, ie = opaqueRenderBatches.count(); i != ie; ++i) {
    PreparedRenderBatch *renderBatch = &opaqueRenderBatches[i];
    if (renderBatch->batch->merged)
        renderMergedBatch(renderBatch);
    else
        renderUnmergedBatch(renderBatch);
}
```

Record draw calls



```
// depth test stays enabled but no need to write out depth from the
// transparent (back-to-front) pass
m_gstate.depthWrite = false;

QVarLengthArray<PreparedRenderBatch, 64> alphaRenderBatches;
if (Q_LIKELY(renderAlpha)) {
    for (int i = 0, ie = m_alphaBatches.size(); i != ie; ++i) {
        Batch *b = m_alphaBatches.at(i);
```

- › **Prepare:** Gather all data (geometry, pipeline states, shader res.) needed for the current frame, enqueue buffer (vertex, index, uniform) and texture resource updates.
- › **Render:** start the pass, record binding ia/shader/pipeline stuff, record draw call, change bindings if needed, record draw call, ..., end pass.
- › **Submit and present.**

```
for (int i = 0, ie = opaqueRenderBatches.count(); i != ie; ++i) {
    PreparedRenderBatch *renderBatch = &opaqueRenderBatches[i];
    if (renderBatch->batch->merged)
        renderMergedBatch(renderBatch);
    else
        renderUnmergedBatch(renderBatch);
}
```

# 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>