Porting to QRhi 1

No intermixed copy operations, state changes, and draw calls.

Porting to QRhi 2

Must collect and submit resource updates before starting to record a render or compute pass.

Qt

```
void nextFrame()
{
    QRhiResourceUpdateBatch *resourceUpdates = m_rhi->nextResourceUpdateBatch();
    if (m_initialUpdates) {
        resourceUpdates->merge(m_initialUpdates);
        m_initialUpdates->release();
        m_initialUpdates = nullptr;
    }

    QMatrix4x4 triMvp = m_triBaseMvp;
    triMvp.rotate(m_triRot, 0, 1, 0);
    resourceUpdates->updateDynamicBuffer(m_triUbuf, 0, 64, triMvp.constData());

    cb->beginPass(rt, Qt::black, { 1.0f, 0 }, resourceUpdates);
```

Porting to QRhi 3

Must create up front and then reuse resources like:

- pipeline state objects
- objects describing the shader resource bindings (which uniform buffers, textures, etc. are visible to which shader stages)

Qt

Porti

Must cr

- pipe
- obje
  buff

```cpp
3236 pool Renderer::ensurePipelineState(Element *e, const ShaderManager::Shader *sms) // RHI
3237 {
3238     // In unmerged batches the srbs in the elements are all compatible layout-wise.
3239     const GraphicsPipelineStateKey k { m_gstate, sms, renderPassDescriptor(), e->srb };
3240
3241     // See if there is an existing, matching pipeline state object.
3242     auto it = m_pipelines.constFind(k);
3243     if (it != m_pipelines.constEnd()) {
3244         e->ps = *it;
3245         return true;
3246     }
3247
3248     // Build a new one. This is potentially expensive.
3249     QRhiGraphicsPipeline *ps = m_rhi->newGraphicsPipeline();
3250     ps->setShaderStages(sms->programRhi.shaderStages);
3251     ps->setVertexInputLayout(sms->programRhi.inputLayout);
3252     ps->setShaderResourceBindings(e->srb);
3253     ps->setRenderPassDescriptor(renderPassDescriptor());
3254
3255     QRhiGraphicsPipeline::Flags flags = 0;
3256     if (needsBlendConstant(m_gstate.srcColor) || needsBlendConstant(m_gstate.dstColor))
3257         flags |= QRhiGraphicsPipeline::UsesBlendConstants;
3258     if (m_gstate.usesScissor)
3259         flags |= QRhiGraphicsPipeline::UsesScissor;
3260     if (m_gstate.stencilTest)
3261         flags |= QRhiGraphicsPipeline::UsesStencilRef;
3262
3263     ps->setFlags(flags);
3264     ps->setTopology(qsg_topology(m_gstate.drawMode));
3265     ps->setCullMode(m_gstate.cullMode);
3266
3267     QRhiGraphicsPipeline::TargetBlend blend;
3268     blend.colorWrite = m_gstate.colorWrite;
3269     blend.enable = m_gstate.blending;
3270     blend.srcColor = m_gstate.srcColor;
```

rm

Porting to QRhi 4

Must declare up front certain usages of resources like textures.

Qt

Pc

Mu

qrhi_p.h QRhiTexture

```
class Q_GUI_EXPORT QRhiTexture : public QRhiResource
{
public:
    enum Flag {
        RenderTarget = 1 << 0,
        CubeMap = 1 << 2,
        MipMapped = 1 << 3,
        sRGB = 1 << 4,
        UsedAsTransferSource = 1 << 5,
        UsedWithGenerateMips = 1 << 6,
        UsedWithLoadStore = 1 << 7
    };
    Q_DECLARE_FLAGS(Flags, Flag)
```

Porting to QRhi 5

Uniform buffers. No individual uniforms.

Porting to QRhi 5

Uniform buffers. No individual uniforms.

From QRhi API perspective. Backends may not actually operate with native uniform buffers.

Porting to QRhi 5

Uniform buffers. No individual uniforms.

From QRhi API perspective. Backends may not actually operate with native uniform buffers. (hello OpenGL ES 2.0 and broken ES 3.x implementations!)

Porting to QRhi 6

No target buffer (color/depth/stencil) clears at arbitrary times.

Qt

Port

No tar

```
if (!(clipType & ClipState::StencilClip)) {
    if (!m_clipProgram.isLinked()) {
        QSGShaderSourceBuilder::initializeProgramFromFiles(
            &m_clipProgram,
            QStringLiteral(":/qt-project.org/scenegraph/shaders/stencilclip.vert"),
            QStringLiteral(":/qt-project.org/scenegraph/shaders/stencilclip.frag"));
        m_clipProgram.bindAttributeLocation("vCoord", 0);
        m_clipProgram.link();
        m_clipMatrixId = m_clipProgram.uniformLocation("matrix");
    }

    glClearStencil(0);
    glClear(GL_STENCIL_BUFFER_BIT);
    glEnable(GL_STENCIL_TEST);
    glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
    glDepthMask(GL_FALSE);

    m_clipProgram.bind();
    m_clipProgram.enableAttributeArray(0);

    clipType |= ClipState::StencilClip;
}

glStencilFunc(GL_EQUAL, m_currentStencilValue, 0xff); // stencil test, ref, test mask
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR); // stencil fail, z fail, z pass

const QSGGeometry *g = clip->geometry();
Q_ASSERT(g->attributeCount() > 0);
const QSGGeometry::Attribute *a = g->attributes();
```

Qt

Port

No tar

```
if (!(clipType & ClipState::StencilClip)) {
    if (!m_clipProgram.isLinked()) {
        QSGShaderSourceBuilder::initializeProgramFromFiles(
            &m_clipProgram,
            QStringLiteral(":/qt-project.org/scenegraph/shaders/stencilclip.vert"),
            QStringLiteral(":/qt-project.org/scenegraph/shaders/stencilclip.frag"));
        m_clipProgram.bindAttributeLocation("vCoord", 0);
        m_clipProgram.link();
        m_clipMatrixId = m_clipProgram.uniformLocation("matrix");
    }

    glClearStencil(0);
    glClear(GL_STENCIL_BUFFER_BIT);
    glEnable(GL_STENCIL_TEST);
    glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
    glDepthMask(GL_FALSE);

    m_clipProgram.bind();
    m_clipProgram.enableAttributeArray(0);

    clipType |= ClipState::StencilClip;
}

glStencilFunc(GL_EQUAL, m_currentStencilValue, 0xff); // stencil test, ref, test mask
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR); // stencil fail, z fail, z pass

const QSGGeometry *g = clip->geometry();
Q_ASSERT(g->attributeCount() > 0);
const QSGGeometry::Attribute *a = g->attributes();
```

!#$&^#@(*

Porting to QRhi 7

Two buffers are good, one is better.

```
drawCall.vbufOffset = aligned(vOffset, 4);
const int vertexByteSize = g->sizeOfVertex() * g->vertexCount();
vOffset = drawCall.vbufOffset + vertexByteSize;

int indexByteSize = 0;
if (g->indexCount()) {
    drawCall.ibufOffset = aligned(iOffset, 4);
    indexByteSize = g->sizeOfIndex() * g->indexCount();
    iOffset = drawCall.ibufOffset + indexByteSize;
}

drawCall.ubufOffset = aligned(uOffset, m_ubufAlignment);
uOffset = drawCall.ubufOffset + StencilClipUbufSize;

QMatrix4x4 matrixYUpNDC = m_current_projection_matrix;
if (clip->matrix())
    matrixYUpNDC *= *clip->matrix();

m_resourceUpdates->updateDynamicBuffer(batch->stencilClipState.ubuf,
                                       drawCall.ubufOffset, 64, matrixYUpNDC.constData());
m_resourceUpdates->updateDynamicBuffer(batch->stencilClipState.vbuf,
                                       drawCall.vbufOffset, vertexByteSize, g->vertexData());
if (indexByteSize)
    m_resourceUpdates->updateDynamicBuffer(batch->stencilClipState.ibuf,
                                           drawCall.ibufOffset, indexByteSize, g->indexData());
```

Consequence

Typically, a **render** step becomes **prepare** + **render**.

Consequence

- 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.

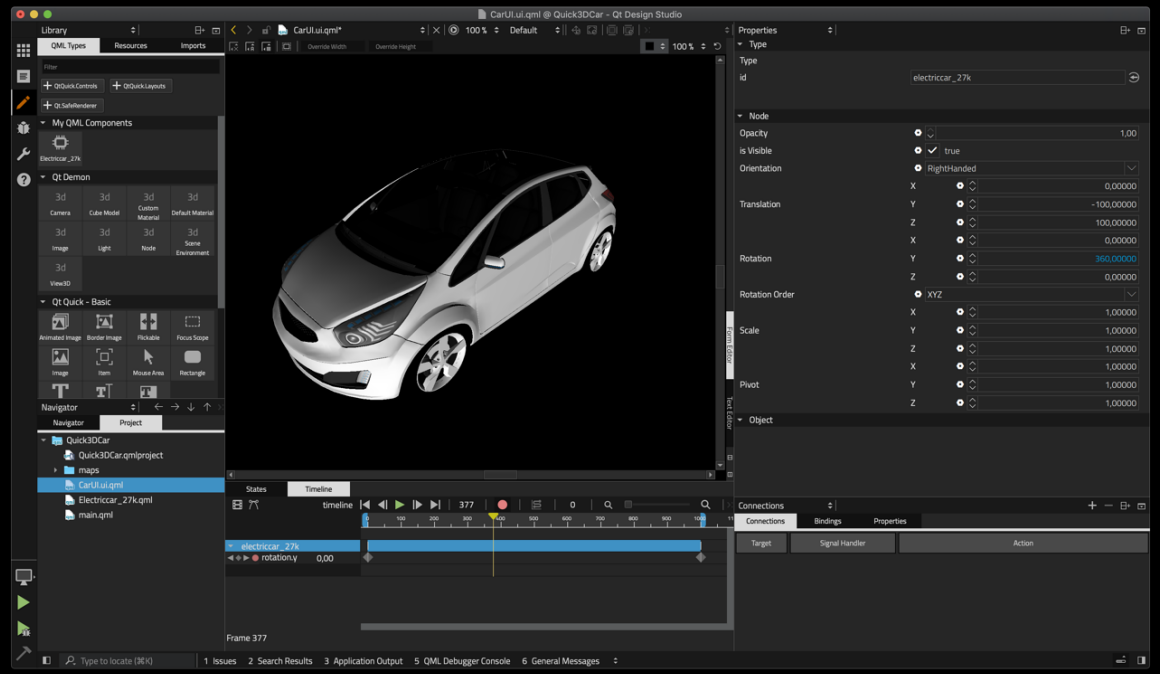
What does it mean?

Qt 5.14 is out, now what?
What to expect in Qt 6.0?

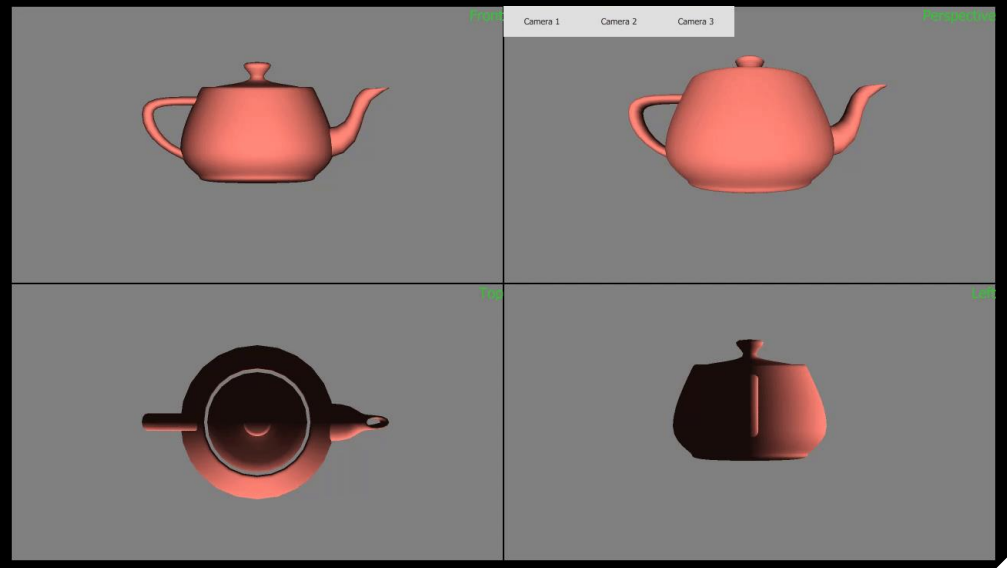
Qt Quick

- The opt-in rendering path in Qt Quick proves the concept.
- Now can move towards making it the default in Qt 6.0.
- And then remove the other code path.

Qt Quick 3D



- The other big fish.
- To be ported completely to QRhi, similarly to Qt Quick.



Disclaimer:
Pay attention to the question marks
from this point on.

QPainter

- QRhi-based paint engine?
  - To replace the aging OpenGL backend of QPainter.
- Enable rendering QWidgets via this?
  - Remember *–graphicssystem opengl* in Qt 4?

Platform specifics

- Windows: Could ANGLE and associated plumbing be removed in Qt 6?
- macOS/iOS: Could OpenGL support be purged from the platform plugins?
- macOS: Use Metal for more efficient QWidget backingstore handling?

Platform specifics

- Android: Is OpenGL relevant much longer? If not, purge?
- UWP (WinRT): Rely on D3D11 via QRhi, instead of ANGLE?
- Vulkan on embedded without a windowing system?

Various Qt modules

- Direct OpenGL and GLSL usage is problematic.
  - **But not necessarily a showstopper!**
- Investigations on-going.
  - graphicaleffects, charts, datavisualization, multimedia, wayland (compositor), location, webengine