QtWS23

Low-Level 2D/3D Graphics in Qt 6.6 and Beyond

Laszlo Agocs

The Qt Company, Norway

Qt World Summit 2023

Berlin

```
                                                    ->newBuffer(QRhiBuffer::Dynamic, QRhiBuffer::UniformBuffer, 64));
                                                    );

                                                    et(m_rhi->newShaderResourceBindings());
                                                    ->setBindings({
                                                      QRhiShaderResourceBinding::uniformBuffer(0, QRhiShaderResourceBinding::VertexStage, m_ubuf.g
});
m_srb->create();

m_pipeline.reset(m_rhi->newGraphicsPipeline());
m_pipeline->setShaderStages({
    { QRhiShaderStage::Vertex, getShader(QLatin1String(":/shaders/color.vert.qsb")) },
    { QRhiShaderStage::Fragment, getShader(QLatin1String(":/shaders/color.frag.qsb")) }
});
QRhiVertexInputLayout inputLayout;
inputLayout.setBindings({
    { 5 * sizeof(float) }
});
inputLayout.setAttributes({
    { 0, 0, QRhiVertexInputAttribute::Float2, 0 },
    { 0, 1, QRhiVertexInputAttribute::Float3, 2 * sizeof(float) }
});
m_pipeline->setVertexInputLayout(inputLayout);
m_pipeline->setShaderResourceBindings(m_srb.get());
m_pipeline->setRenderPassDescriptor(renderTarget()->renderPassDescriptor());
m_pipeline->create();

QRhiResourceUpdateBatch *resourceUpdates = m_rhi->nextResourceUpdateBatch();
resourceUpdates->uploadStaticBuffer(m_vbuf.get(), vertexData);
cb->resourceUpdate(resourceUpdates);


t QSize outputSize = colorTexture()->pixelSize();
ewProjection = m_rhi->clipSpaceCorrMatrix();
ewProjection.perspective(45.0f, outputSize.width() / (float) outputSize.height(), 0.01f, 1000.0f
ewProjection.translate(0, 0, -4);


mpleRhiWidget::render(QRhiCommandBuffer *cb)
```

Topics

- QRhi is now "semi-public"
  - Rendering off-screen or to a QWindow (Qt 6.6+)
- QRhi-based rendering with Qt Widgets
  - QRhiWidget (Qt 6.7+)
- QRhi-based rendering in Qt Quick
  - QQuickRhiItem (Qt 6.7+)
  - Other approaches (Qt 6.6+)
- Redirecting Qt Quick rendering
  - QQuickRenderControl with QRhi (Qt 6.6+)
- New and upcoming
  - D3D12, GPU timings, etc.

Qt 5


Qt Core
OS, windowing system
Platform plugins
QPA
Qt Gui
OpenGL, WSI
Qt Widgets
Qt Quick

Qt 6


Qt Core
OS, windowing system
Platform plugins
QPA
Qt Gui
QRhi
Qt Shader Tools
3D API, WSI
QRhi backends (OpenGL, Vulkan, Metal, D3D11, D3D12)
Qt Widgets
Qt Quick
Qt Quick 3D

Accelerated graphics in Qt 6

- Qt prefers using the platform's best supported accelerated 3D API.
  - Offering the application/users the choice, if there are multiple options.
- QRhi wraps OpenGL (ES), D3D11, D3D12, Vulkan, or Metal.
  - plus a dummy Null backend
- Used within Qt to implement
  - Qt Quick
  - Qt Quick 3D
  - Qt Multimedia (Qt 6.2+)
  - Qt Widgets' mini-compositor when texture-based widgets are involved (Qt 6.4+)
  - …

History

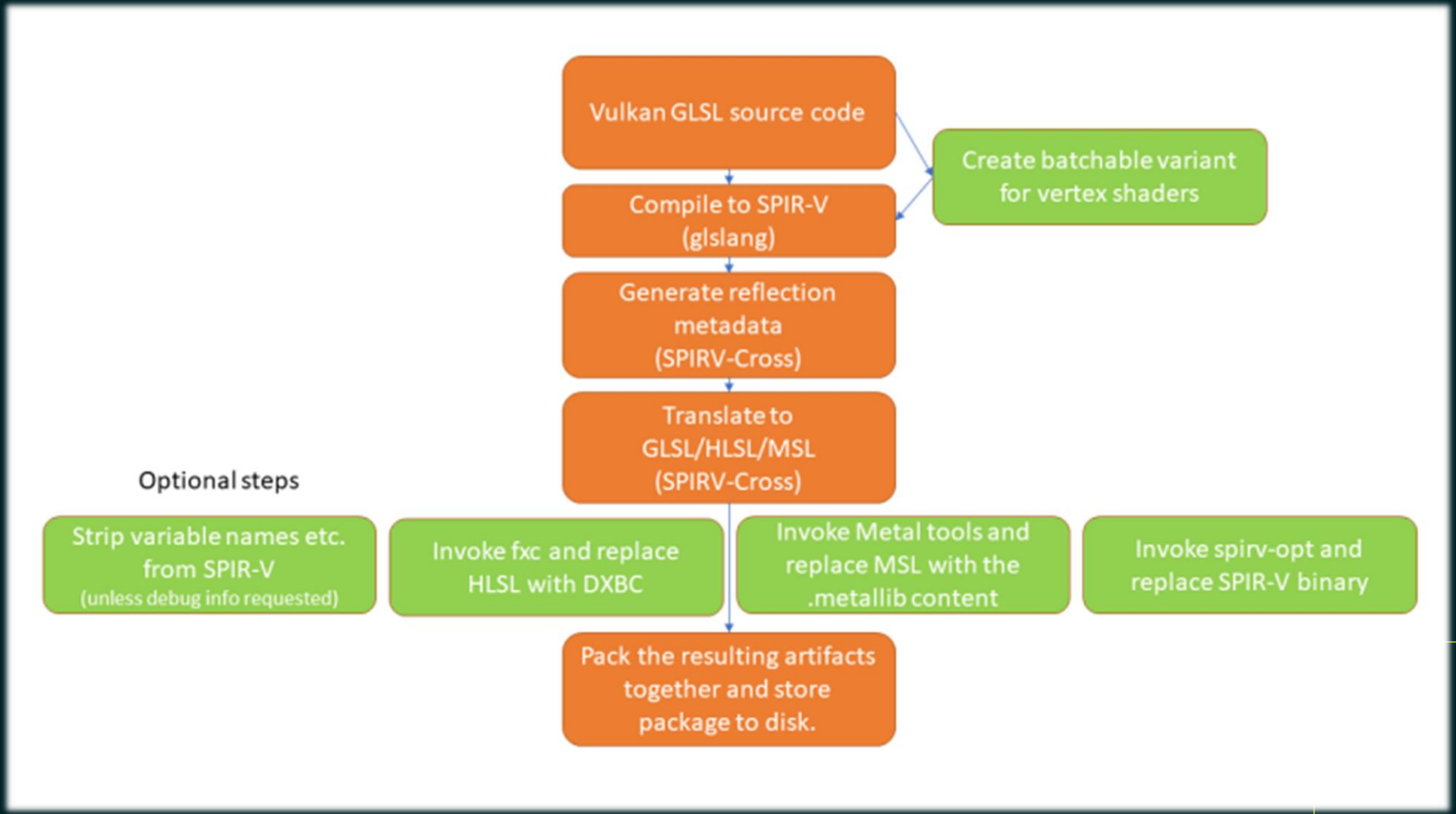
- World
  - 2014 – Metal
  - 2015 – Direct 3D 12
  - 2016 – Vulkan
- Qt
  - 2016 - Experimental Direct 3D 12 scenegraph plugin for Qt Quick -> removed in Qt 6.0
  - 2017 - Shader handling experiments (glslang, SPIRV-Cross) -> foundations of Qt Shader Tools
  - 2018 - Experiment with a Vulkan rendering helper for Qt Quick -> eventually morphs into QRhi
  - 2019 - Merged QRhi and shader infra to Qt 5.14, with an opt-in, experimental port of Qt Quick
  - 2020 - Qt 6.0 comes with Qt Quick and Qt Quick 3D fully ported
  - 2023 - Qt 6.6 opens the QRhi APIs and docs to application developers
  - 2024 - ...

QRhi in Qt 6

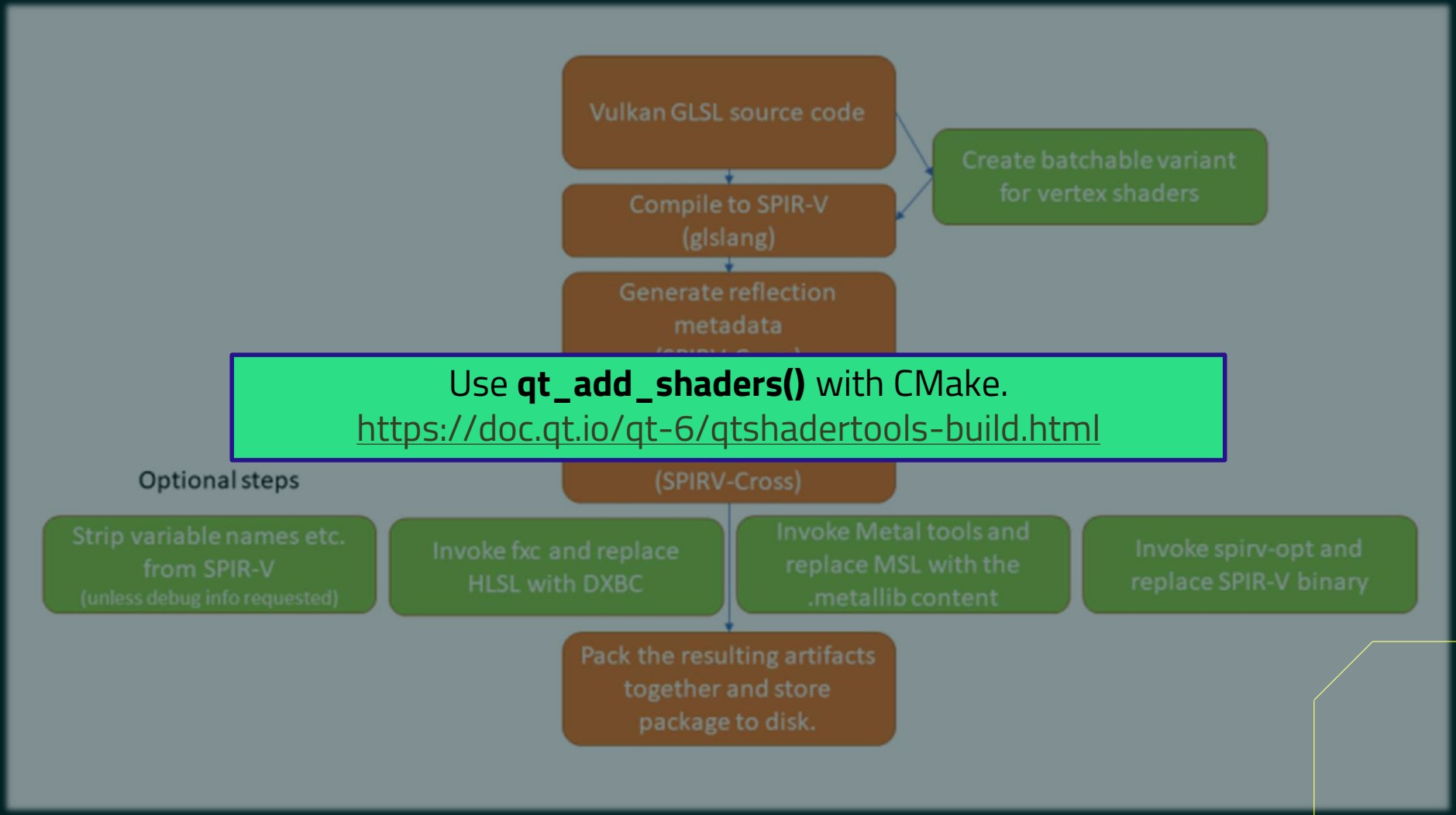
- Private API in Qt 6.0 – 6.5.
- QPA-style public-ish API with limited compatibility guarantees from 6.6 on.
- Applications need to pull in Qt::GuiPrivate (CMake) still.
- Can then do #include <rhi/qrhi.h>
- Same story as with #include <qpa/qplatform*.h>
- Documentation! https://doc.qt.io/qt-6/qrhi.html
- Examples

Walkthrough
Offscreen QRhi example
(render to texture, read back to QImage, save to file)
https://git.qt.io/laagocs/qtws23_graphics_examples
01_offscreen

Shader pipeline

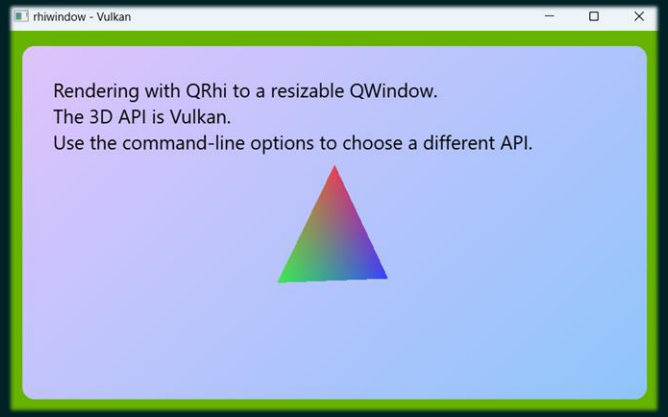


Shader pipeline



Use **qt_add_shaders()** with CMake.
https://doc.qt.io/qt-6/qtshadertools-build.html

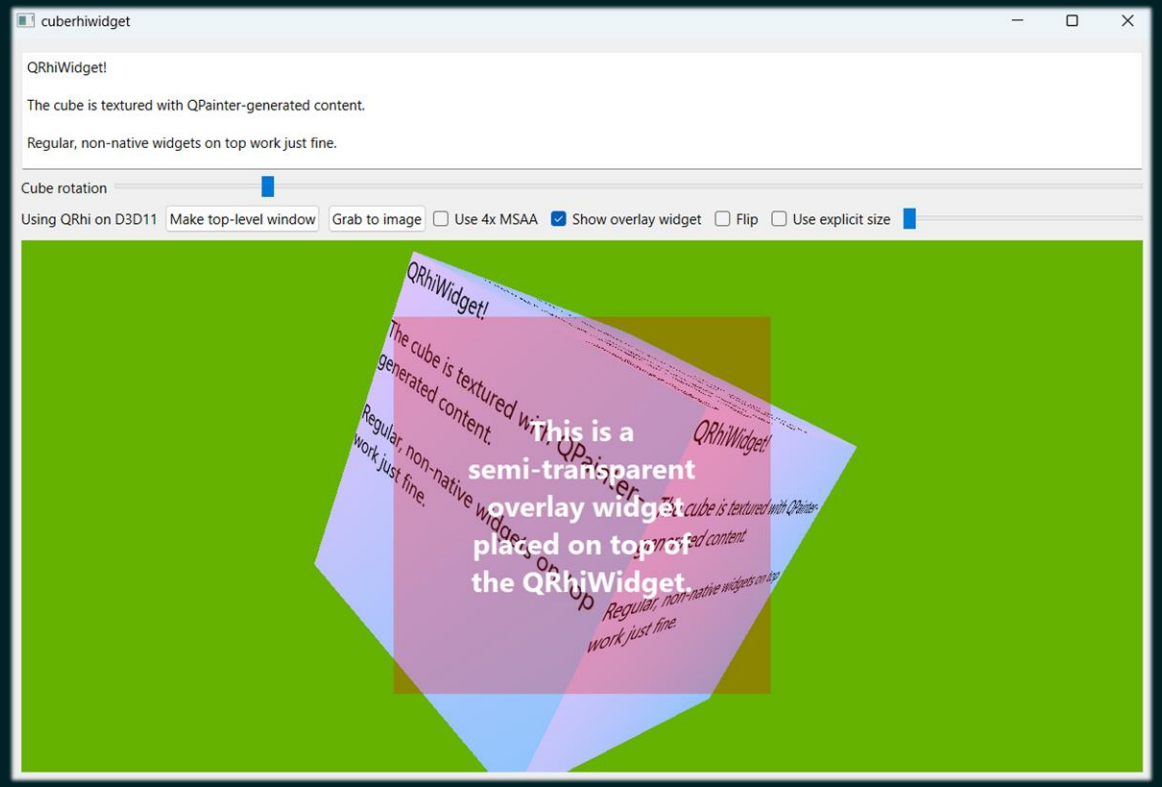
Now in a QWindow



- https://doc.qt.io/qt-6/qtgui-rhiwindow-example.html
- No widgets, no QML, just using QtGui
  - Uses QPainter to draw into a QImage, gets a QRhiTexture from that, and textures a fullscreen quad with it.
  - Then draws a triangle as well.
- Needs to deal with swapchains, expose events, preparing for resizes, etc.
- Relevant if:
  - You make your own UI and rendering engine and only really need a window.
  - You are a Qt developer.
    - e.g., this is the foundation of QQuickWindow and its render loops

QRhiWidget

- Coming in Qt 6.7.
  - If all goes as planned.
- https://doc-snapshots.qt.io/qt6-dev/qrhiwidget.html
- In principle similar to QOpenGLWidget.
  - API is slightly modernized.



QRhiWidget / QQuickWidget

- Builds on the same new Qt 6.4 architecture that enables QQuickWidget to function with Vulkan, Metal, D3D, and whatever QRhi supports.

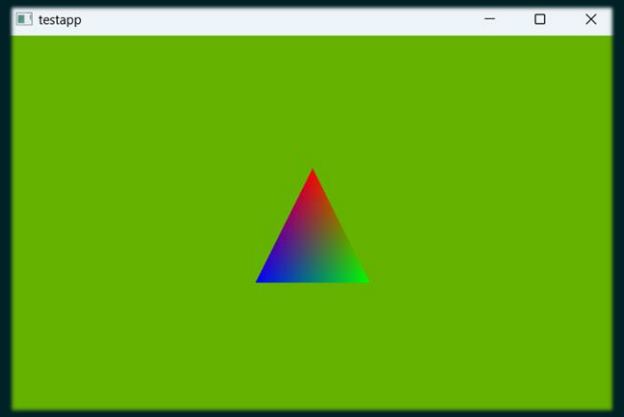
https://doc.qt.io/qt-6/qquickwidget.html#graphics-api-support

Graphics API Support

QQuickWidget is functional with all the 3D graphics APIs supported by Qt Quick, as well as the `software` backend. Other backends, for example OpenVG, are not compatible however and attempting to construct a QQuickWidget will lead to problems.

Overriding the platform's default graphics API is done the same way as with QQuickWindow and QQuickView: either by calling QQuickWindow::setGraphicsApi() early on before constructing the first QQuickWidget, or by setting the QSG_RHI_BACKEND environment variable.

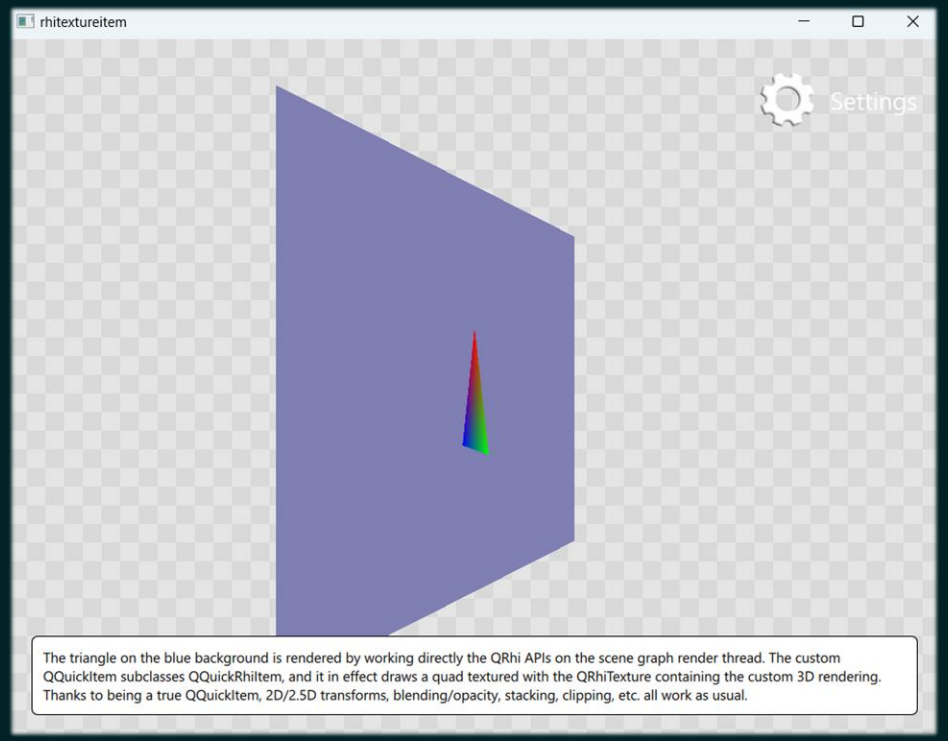
> **Note:** One top-level window can only use one single graphics API for rendering. For example, attempting to place a QQuickWidget using Vulkan and a QOpenGLWidget in the widget hierarchy of the same top-level window, problems will occur and one of the widgets will not be rendering as expected.

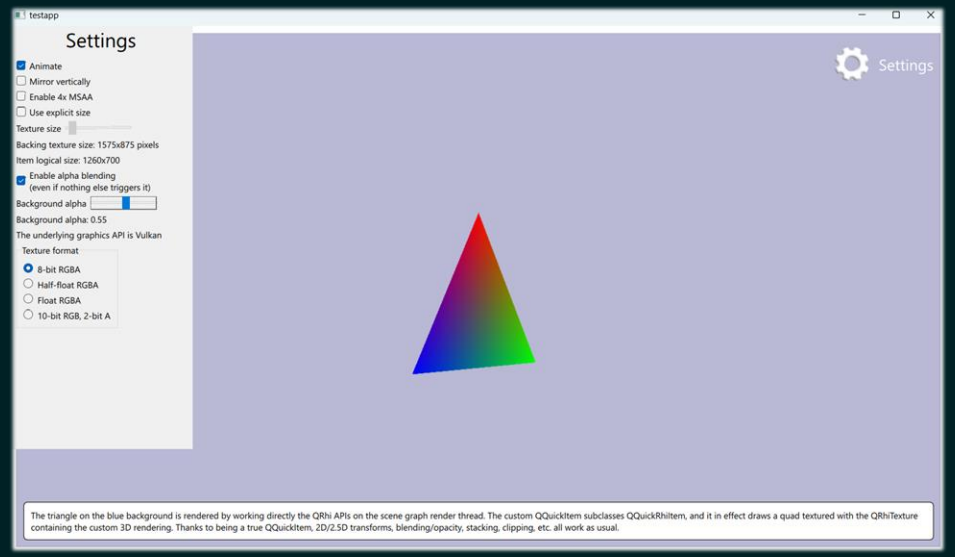


Walkthrough
QRhiWidget-based minimal example
https://git.qt.io/laagocs/qtws23_graphics_examples
02_qrhiwidget

QQuickRhiItem

- Coming in Qt 6.7.
  - If all goes as planned.
- https://doc-snapshots.qt.io/qt6-dev/qquickrhiitem.html
- In principle similar to QQuickFramebufferObject.
  - API is slightly modernized.
  - Matches QRhiWidget as much as possible.

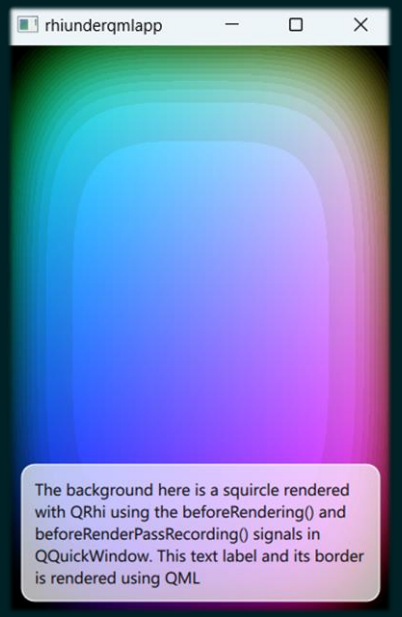




Walkthrough
QQuickRhiItem-based example
https://git.qt.io/laagocs/qtws23_graphics_examples
03_qquickrhiitem

The other two

- Underlay/overlay
  - QQuickWindow::beforeRendering/afterRendering (+beforeRenderPassRecording/afterRenderPassRecording) signals
- QSGRenderNode
  - Inline. Powerful but low-level, with some pitfalls.



https://doc.qt.io/qt-6/qtquick-scenegraph-rhiunderqml-example.html

Redirecting a Qt Quick scene

- Nothing says a QQuickWindow has to be *visible*.
- QQuickRenderControl, QQuickRenderTarget, and friends allow redirecting into a QRhiTexture.
  - Before 6.6: could only (publicly) redirect to OpenGL texture, Vulkan image, Metal texture, etc.

	QQuickRenderControl(QObject **parent* = nullptr)
virtual	**~QQuickRenderControl**() override
void	**beginFrame**()
QRhiCommandBuffer *	**commandBuffer**() const
void	**endFrame**()
bool	**initialize**()
void	**invalidate**()
void	**polishItems**()
void	**prepareThread**(QThread **targetThread*)
void	**render**()
virtual QWindow *	**renderWindow**(QPoint **offset*)
QRhi *	**rhi**() const
int	**samples**() const
void	**setSamples**(int *sampleCount*)
bool	**sync**()
QQuickWindow *	**window**() const

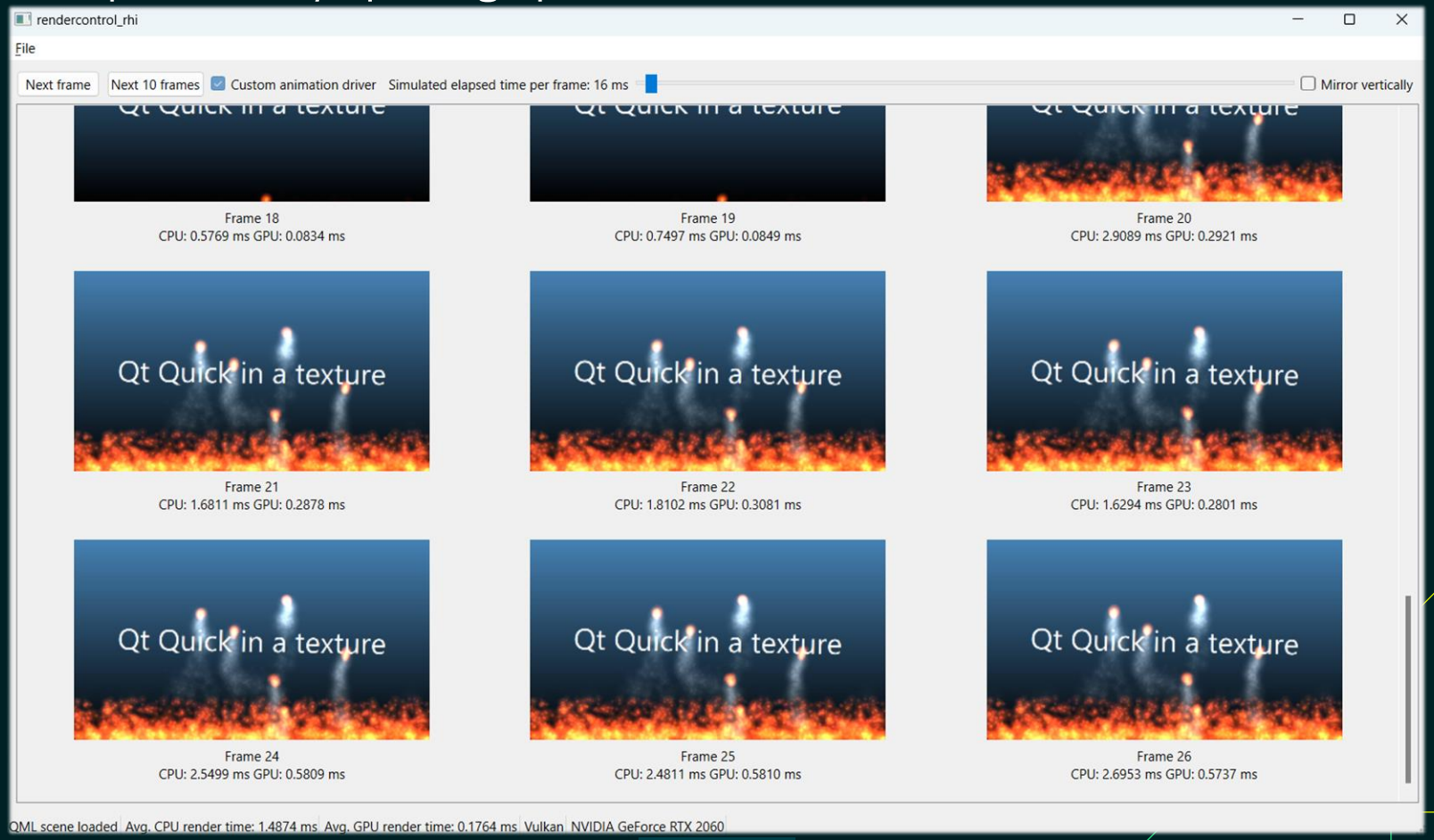
QQuickRenderTarget	**fromD3D11Texture**(void **texture*, uint *format*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromD3D11Texture**(void **texture*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromD3D12Texture**(void **texture*, int *resourceState*, uint *format*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromMetalTexture**(MTLTexture **texture*, uint *format*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromMetalTexture**(MTLTexture **texture*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromOpenGLRenderBuffer**(uint *renderbufferId*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromOpenGLTexture**(uint *textureId*, uint *format*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromOpenGLTexture**(uint *textureId*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromPaintDevice**(QPaintDevice **device*)
QQuickRenderTarget	**fromRhiRenderTarget**(QRhiRenderTarget **renderTarget*)
QQuickRenderTarget	**fromVulkanImage**(VkImage *image*, VkImageLayout *layout*, VkFormat *format*, const QSize &*pixelSize*, int *sampleCount* = 1)
QQuickRenderTarget	**fromVulkanImage**(VkImage *image*, VkImageLayout *layout*, const QSize &*pixelSize*, int *sampleCount* = 1)

QQuickGraphicsDevice	**fromAdapter**(quint32 *adapterLuidLow*, qint32 *adapterLuidHigh*, int *featureLevel* = 0)
QQuickGraphicsDevice	**fromDeviceAndCommandQueue**(MTLDevice **device*, MTLCommandQueue **commandQueue*)
QQuickGraphicsDevice	**fromDeviceAndContext**(void **device*, void **context*)
QQuickGraphicsDevice	**fromDeviceObjects**(VkPhysicalDevice *physicalDevice*, VkDevice *device*, int *queueFamilyIndex*, int *queueIndex* = 0)
QQuickGraphicsDevice	**fromOpenGLContext**(QOpenGLContext **context*)
QQuickGraphicsDevice	**fromPhysicalDevice**(VkPhysicalDevice *physicalDevice*)
QQuickGraphicsDevice	**fromRhi**(QRhi **rhi*)

Qt 6.6

Redirecting a Qt Quick scene

- New example in Qt 6.7: examples/quick/rendercontrol/rendercontrol_rhi
- Fully portable, made possible by opening up QRhiTexture & co.



```qml
void MainWindow::load(const QString &filename)
{
    m_renderControl.reset(new QQuickRenderControl);
    m_scene.reset(new QQuickWindow(m_renderControl.get()));
#if QT_CONFIG(vulkan)
    if (m_scene->graphicsApi() == QSGRendererInterface::Vulkan)
        m_scene->setVulkanInstance(m_vulkanInstance);
#endif

    m_qmlEngine.reset(new QQmlEngine);
    m_qmlComponent.reset(new QQmlComponent(m_qmlEngine.get(), QUrl::fromLocalFile(filename)));
    if (m_qmlComponent->isError()) { ... }
    QObject *rootObject = m_qmlComponent->create();
    if (m_qmlComponent->isError()) { ... }
    QQuickItem *rootItem = qobject_cast<QQuickItem *>(rootObject);
    if (!rootItem) { ... }

    m_scene->contentItem()->setSize(rootItem->size());
    m_scene->setGeometry(0, 0, rootItem->width(), rootItem->height());
    rootItem->setParentItem(m_scene->contentItem());

    if (!m_renderControl->initialize()) { ... }

    QRhi *rhi = m_renderControl->rhi();
    const QSize pixelSize = rootItem->size().toSize();
    m_texture.reset(rhi->newTexture(QRhiTexture::RGBA8, pixelSize, 1,
                                    QRhiTexture::RenderTarget | QRhiTexture::UsedAsTransferSource));
    if (!m_texture->create()) { ... }

    m_ds.reset(rhi->newRenderBuffer(QRhiRenderBuffer::DepthStencil, pixelSize, 1));
    if (!m_ds->create()) { ... }

    QRhiTextureRenderTargetDescription rtDesc(QRhiColorAttachment(m_texture.get()));
    rtDesc.setDepthStencilBuffer(m_ds.get());
    m_rt.reset(rhi->newTextureRenderTarget(rtDesc));
    m_rpDesc.reset(m_rt->newCompatibleRenderPassDescriptor());
    m_rt->setRenderPassDescriptor(m_rpDesc.get());
    if (!m_rt->create()) { ... }

    m_scene->setRenderTarget(QQuickRenderTarget::fromRhiRenderTarget(m_rt.get()));

    render();
}
```

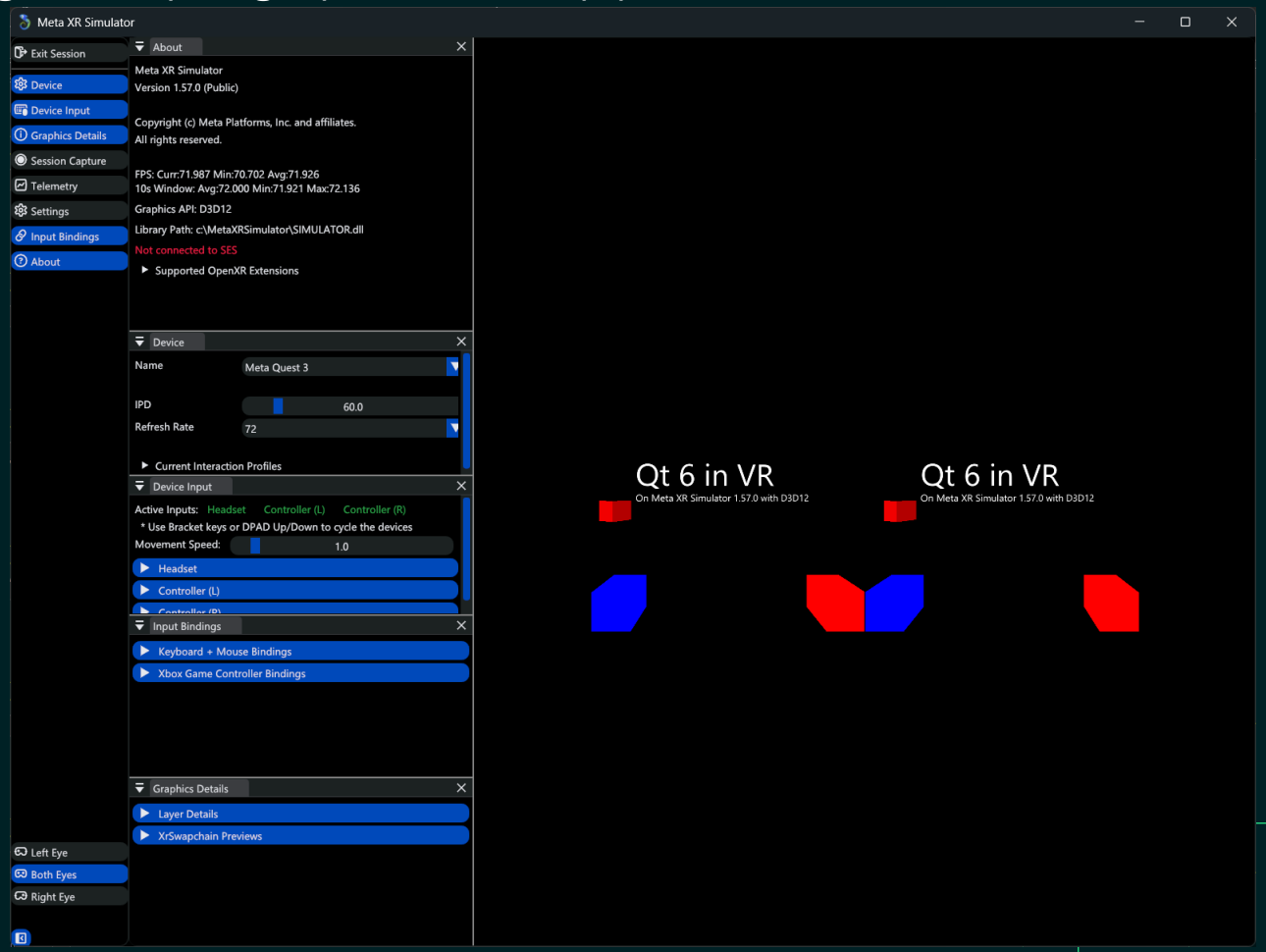
Before Qt 6.6:
Replace with direct OpenGL, Vulkan, Metal, D3D code. Not a great story.

Same here

```
void MainWindow::render()
{
    if (m_frameCount > 0)
        m_animationDriver->advance();

    m_renderControl->polishItems();

    m_renderControl->beginFrame();

    m_renderControl->sync();
    m_renderControl->render();

    QRhi *rhi = m_renderControl->rhi();
    QRhiReadbackResult readResult;
    QRhiResourceUpdateBatch *readbackBatch = rhi->nextResourceUpdateBatch();
    readbackBatch->readBackTexture(m_texture.get(), &readResult);
    m_renderControl->commandBuffer()->resourceUpdate(readbackBatch);

    m_renderControl->endFrame();

    QImage wrapperImage(reinterpret_cast<const uchar *>(readResult.data.constData()),
                        readResult.pixelSize.width(), readResult.pixelSize.height(),
                        QImage::Format_RGBA8888_Premultiplied);
    QImage result;
    if (rhi->isYUpInFramebuffer())
        result = wrapperImage.mirrored();
    else
        result = wrapperImage.copy();

    ...
}
```

Redirecting a Qt Quick scene

- Plenty of uses, from making movies to VR/AR.
  - The foundation for things like https://git.qt.io/annichol/qtquick3dxr (experimental; expect more news H1/2024; if interested now, check its 6.7 branch)



Direct 3D 12

- Qt 6.6 has a new QRhi backend for Direct 3D 12.
  - First class support. Qt is no longer limited to D3D11On12.
- For Qt Quick and QRhiWidget the defaults continue to be D3D11. (on Windows)
  - QSG_RHI_BACKEND=d3d12 / QQuickWindow::setGraphicsApi(QSGRendererInterface::Direct3D12)
  - widget.setApi(QRhiWidget::Api::D3D12)
- Great if you need interop with something D3D12-based.
- Allows using things not in D3D11.
  - For example, view instancing (multiview for VR/AR)

Graphics profiling and debugging

- Qt applications rendering via QRhi are no different from any non-Qt applications rendering with Vulkan, Metal, OpenGL, or Direct 3D.
  - All the tools one would use when developing, for example, a game, are just as applicable.
- Validation and debug layers.
  - Vulkan, Direct 3D, Metal
- Frame capture and profiling tools.
  - RenderDoc, NVIDIA Nsight Graphics/Systems, AMD Radeon GPU tools, PIX, …
- On-screen monitors.
  - MSI Afterburner / Rivatuner, Intel PresentMon, …

GPU timings

- Finally! GPU-side timings, in addition to QElapsedTimer for the CPU side.
- Partially in Qt 6.6. Only really completed in Qt 6.7.
  - 6.7: added timestamp query support for OpenGL 3.3+ and D3D12, fixed up D3D11, etc.
- Simple get-time-for-last-completed-frame, modelled after Metal.
  - Implemented either by timestamp queries or by other means under the hood.

GPU timings

1. QRhiCommandBuffer::lastCompletedGpuTime()

   QRhi::EnableTimestamps / QSG_RHI_PROFILE=1 / QQuickGraphicsConfiguration::setTimestamps()

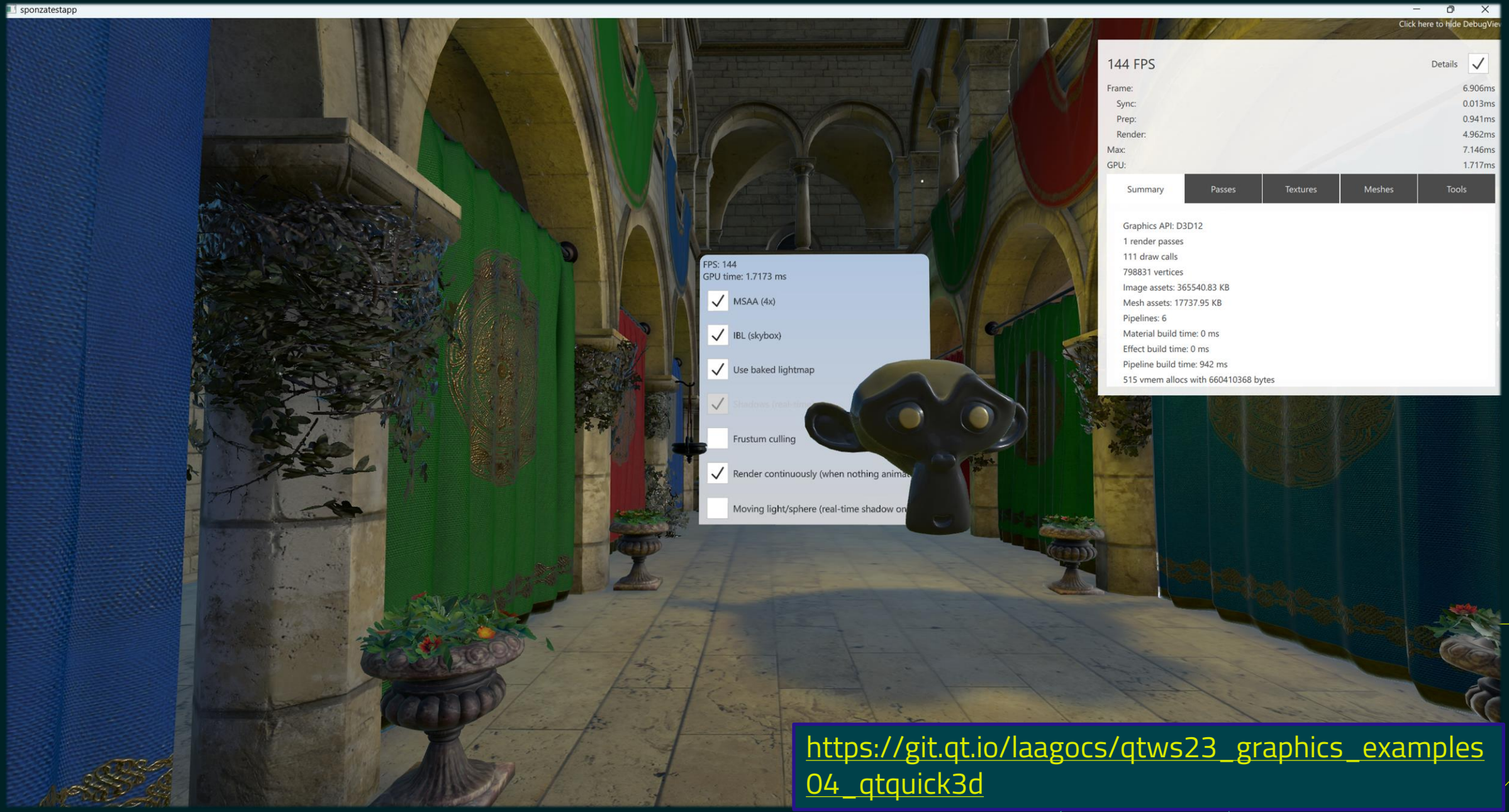
2. When enabled, also shows up in the logs from *qt.scenegraph.time.renderloop*

   (aka QSG_RENDER_TIMING=1)

3. and in Qt Quick 3D's **DebugView** item

   and in the properties of View3D.renderStats.

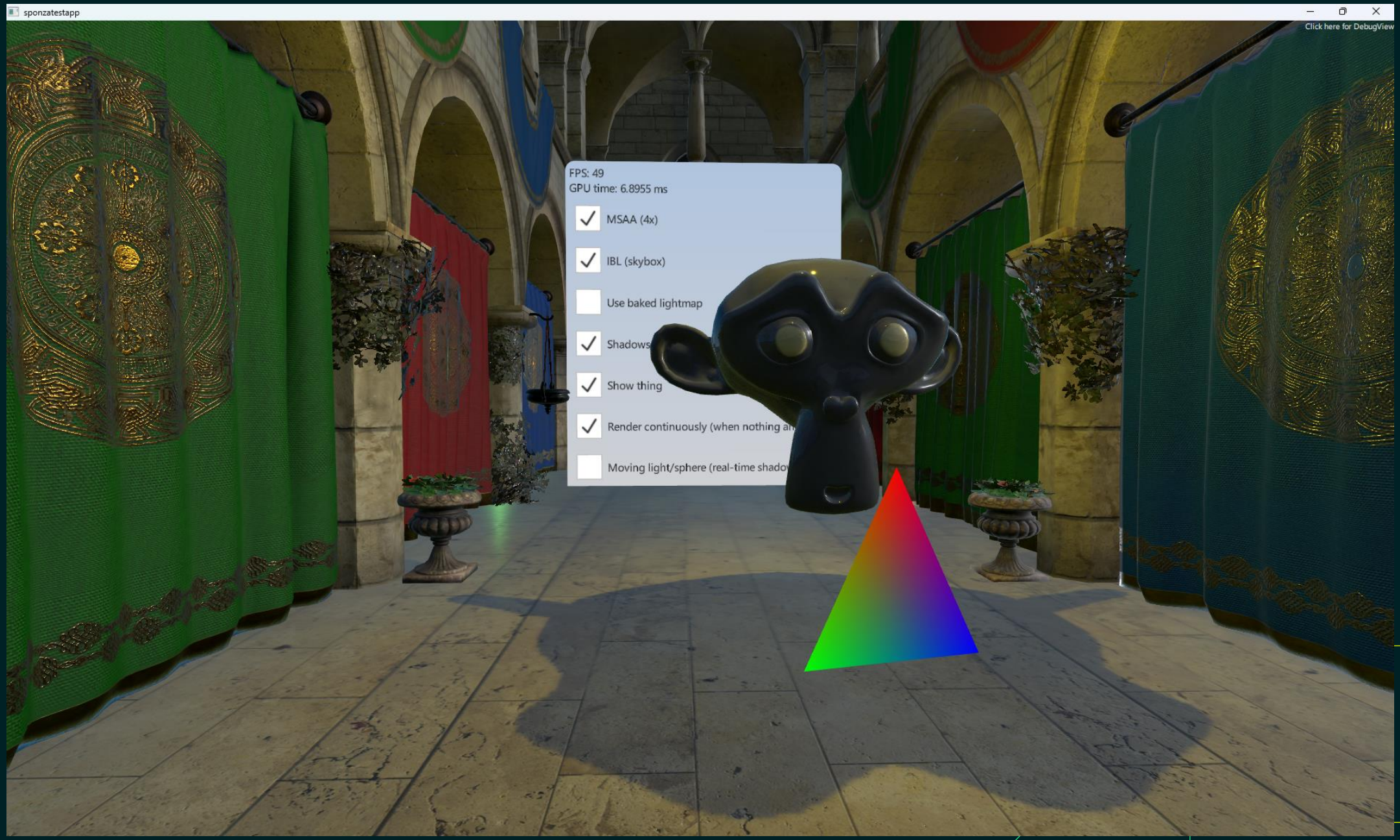
Beware of dynamic frequency scaling. See docs.



https://git.qt.io/laagocs/qtws23_graphics_examples04_qtquick3d

Qt Quick 3D

- Coming soon: Extension objects on the View3D!
- Work in progress.
- Customize the graphics pipeline (render passes, renderable lists) from C++.
- Also allows injecting your custom QRhi-based rendering into the 3D scene.
  - Or into a render pass targeting a texture that then is used in a Principled/CustomMaterial or a post-processing effect.
  - While accessing things like the scene's camera, to get the view-projection matrix for example.


sponzatestapp
Click here for DebugView
FPS: 49
GPU time: 6.8955 ms
MSAA (4x)
IBL (skybox)
Use baked lightmap
Shadows
Show thing
Render continuously (when nothing a
Moving light/sphere (real-time shado

QtWS23

QtWS23

QtWS23

QtWS23

QtWS23