Interactive UIs in Qt Quick 3D

Shawn Rutledge
shawn.rutledge@qt.io
ecloud on #qt-labs, #qt-quick etc.

https://github.com/ec1oud/qtquick3d-input-demo
https://github.com/ec1oud/qt-presentations : qtquick3d-interactive-ui branch

About me

- Qt user since ~2004
- The Qt Company - Oslo since 2011
- Pointing devices: touch, Wacom tablets
- Event delivery and handlers in Qt Quick
- Linux/X11, Wayland, and macOS
- QtPDF
- Qt Quick, Controls and Dialogs

Agenda

- Intro
- Fixed content in 3D apps
- 3D content in 2D apps
- 2D content in 3D apps
- Interactive 3D apps
- Event delivery details
- Remaining work
- Q&A

Disclaimers

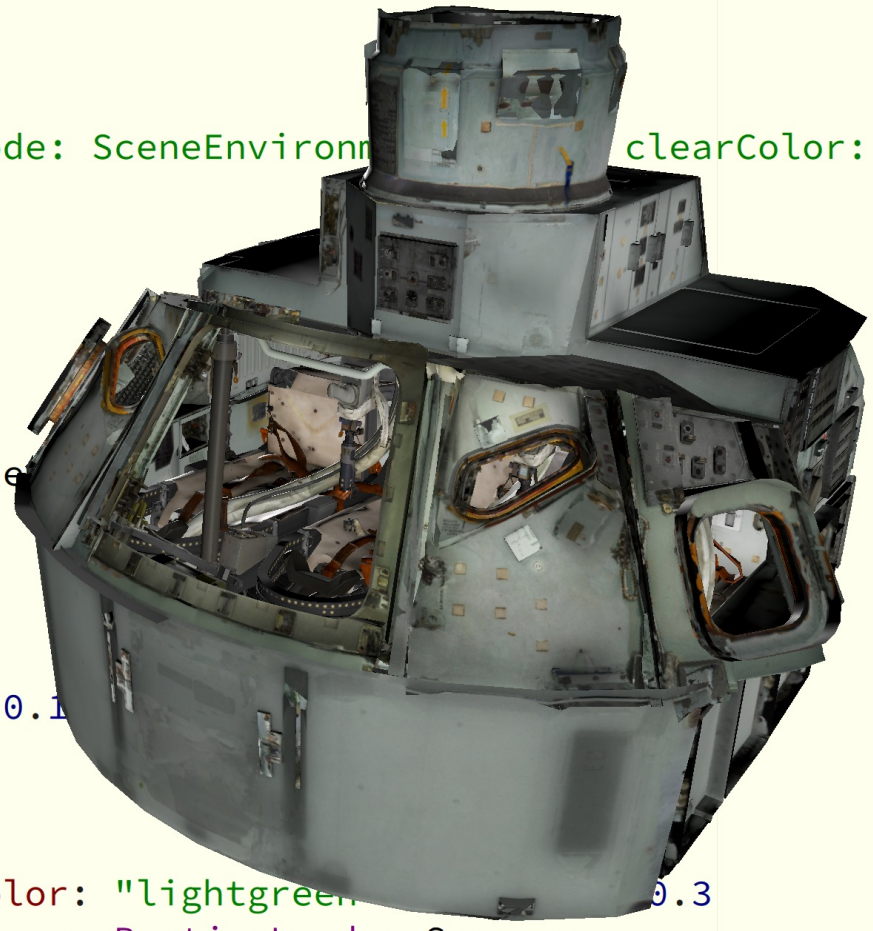
- I didn't implement Qt Quick 3D, only event delivery
- This presentation contains features we haven't shipped

Intro

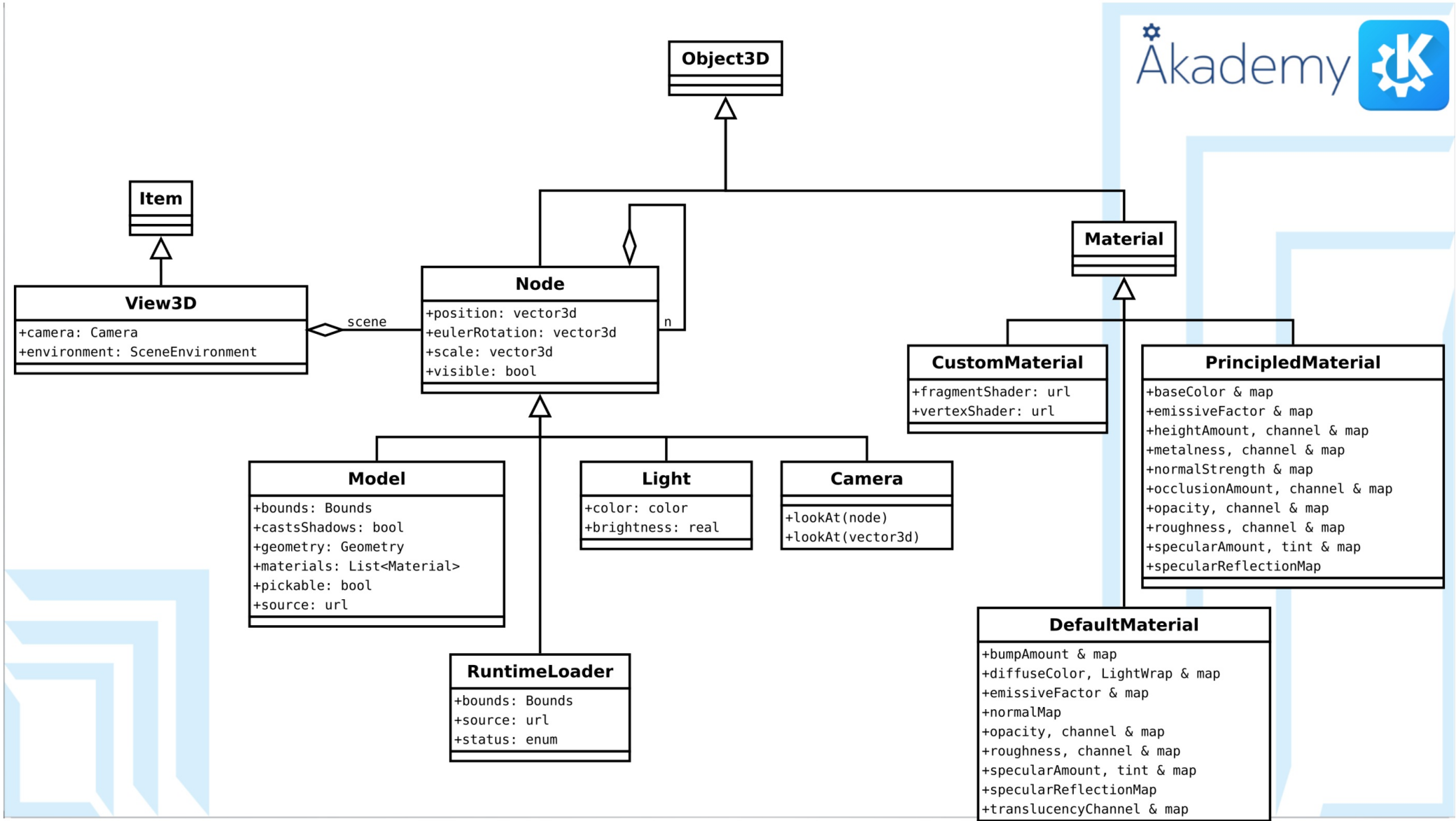
- What is a 3D interactive application?
- What sort of applications need 3D?

Minimal Model Viewer

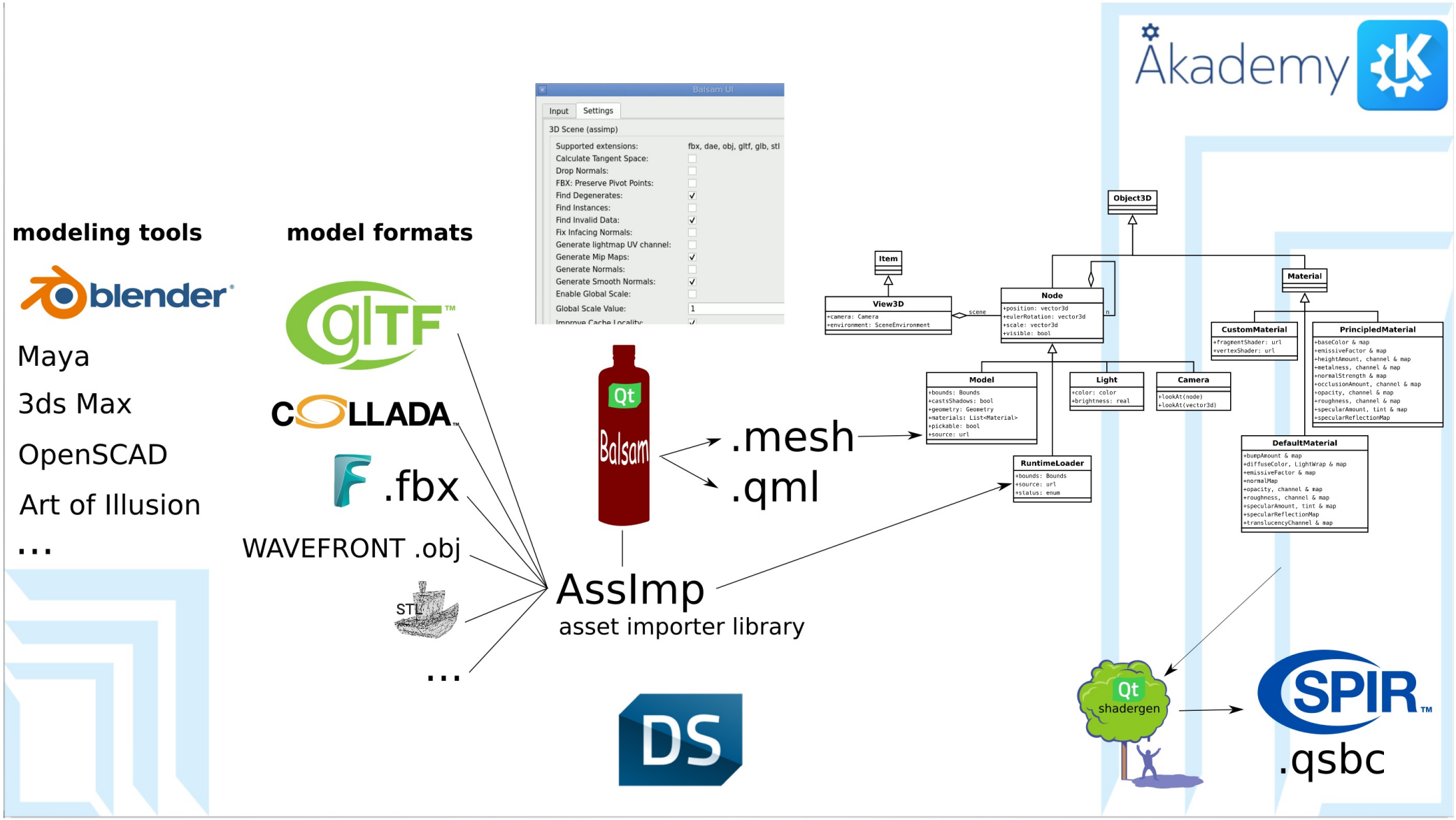
```
import QtQuick
import QtQuick.Dialogs

import Qt.labs.settings

import QtQuick3D
import QtQuick3D.Helpers
import QtQuick3D.AssetUtils

View3D {
    id: view3D; width: 2048; height: 2048; y: 100

//    environment: SceneEnvironment { backgroundMode: SceneEnvironm                clearColor: "#3
    DirectionalLight { }
    DirectionalLight { eulerRotation.x: 180 }
    PerspectiveCamera { id: camera }

    RuntimeLoader {
        id: loader
        property real sf: Math.pow(1.1, wheelHandle
        scale: Qt.vector3d(sf, sf, sf)
        x: dragHandler.persistentTranslation.x
        y: dragHandler.persistentTranslation.y

        AxisHelper { id: axes; scale: Qt.vector3d(0.1

        Model {
            id: boundRP; source: "#Cube"
            materials: PrincipledMaterial { baseColor: "lightgreen                0.3
            visible: axes.visible && loader.status === RuntimeLoader.Success
            position: Qt.vector3d((loader.bounds.maximum.x + loader.bounds.minimum.x) / 2,
```



Most-common QML Types

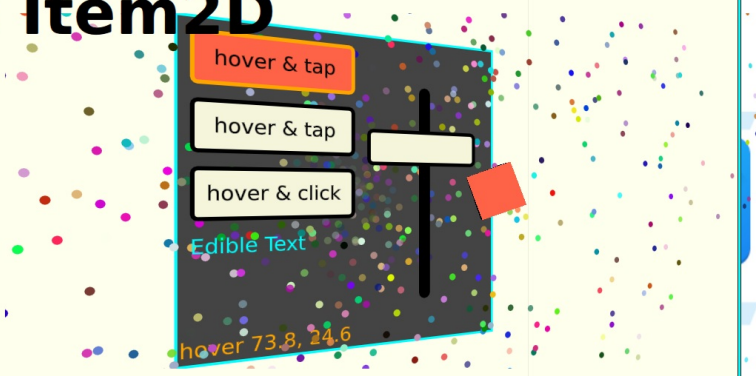


Design Tools Workflow

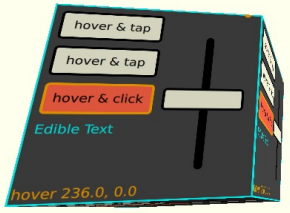


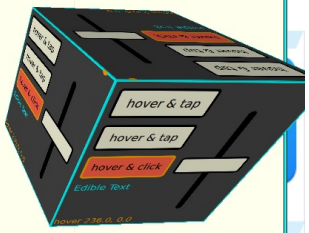
2D content in a 3D app: Item2D

```
import QtQuick
import QtQuick.Particles
import QtQuick3D

View3D {
    width: 1024; height: 480

    PerspectiveCamera { z: 600 }

    DirectionalLight { }

    Node {
        x: -150; y: 128; z: 380
        eulerRotation.y: (bb.sliderValue - 0.4) * 40
        BusyBox { // Root of a 2D subscene!
            id: bb
            ParticleSystem {
                anchors.fill: parent

                ItemParticle {
                    delegate: Rectangle {
                        color: Qt.rgba(Math.random(), Math.random(), Math.random(), 1)
                        radius: 3; width: radius * 2; height: width
                    }
                }
                Emitter {
                    anchors.centerIn: parent
                    enabled: bb.topButtonPressed
                    emitRate: 500; lifeSpan: 2000
                    velocity: PointDirection{ yVariation: 360; xVariation: 360}
                }
```



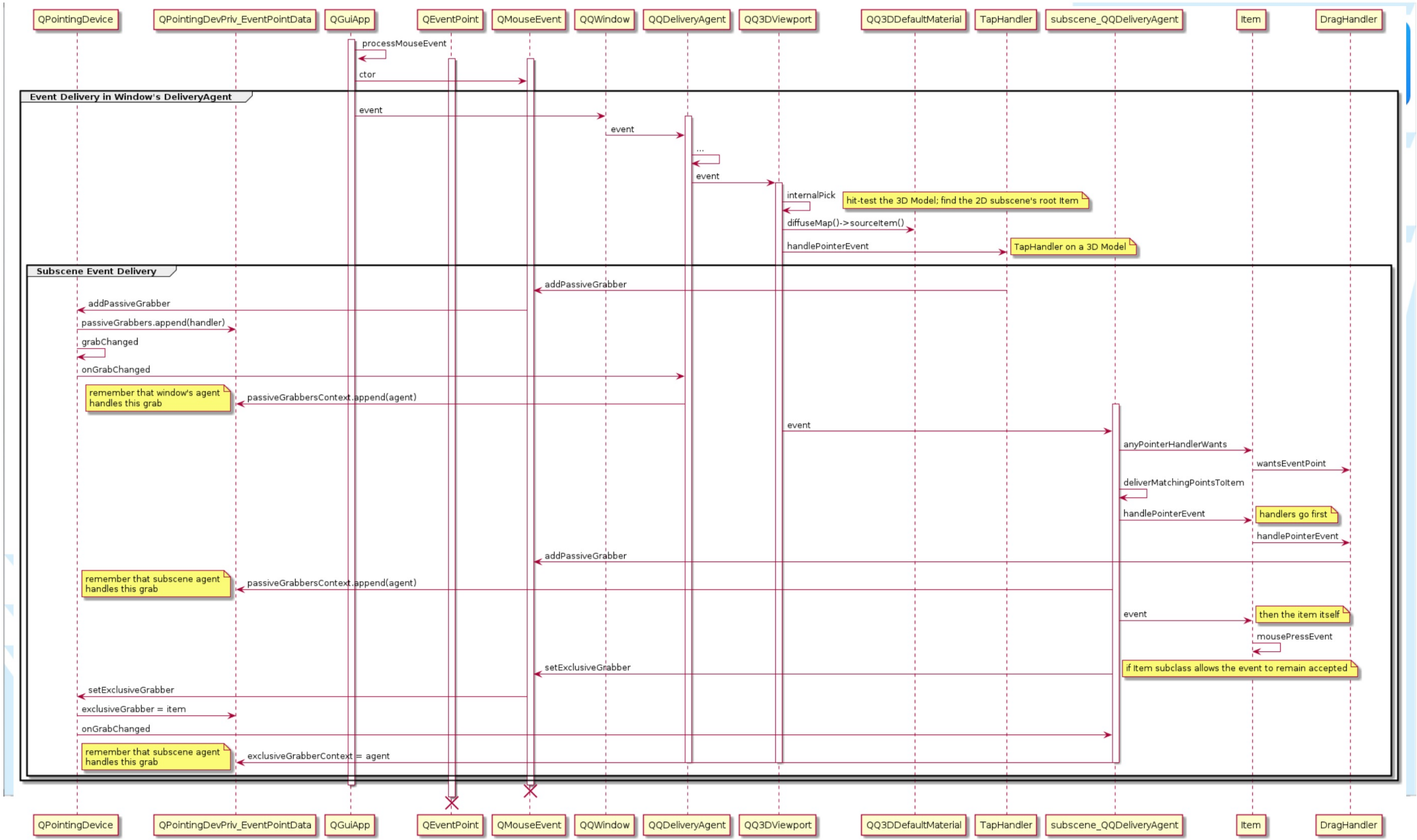
Materials with shared Textures





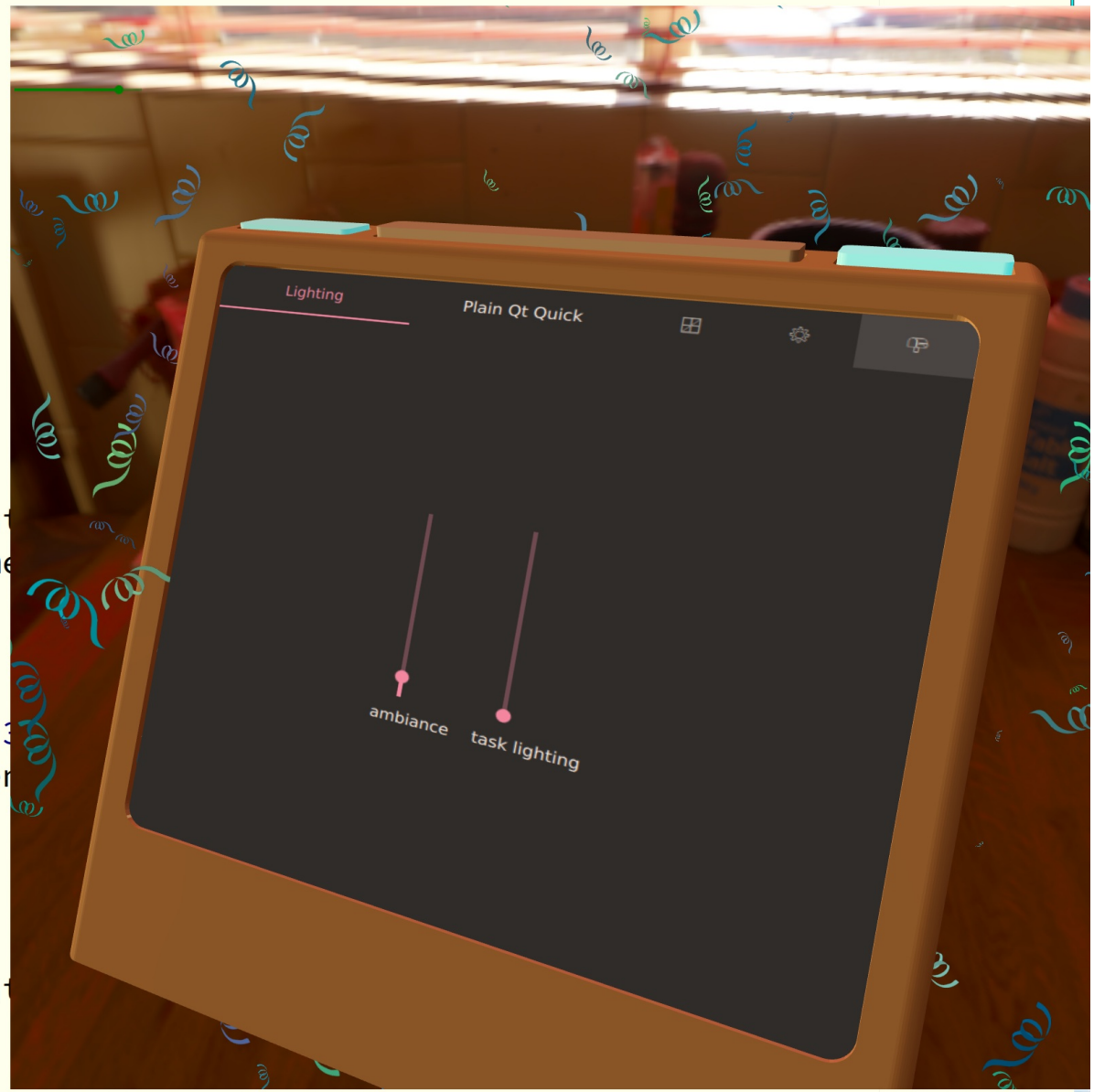

```
import QtQuick
import QtQuick3D

View3D {
    width: 1024; height: 480

    DirectionalLight { eulerRotation.y: 90 }
    DirectionalLight { eulerRotation.y: -45 }
    PerspectiveCamera { z: 600 }

    Node {
        objectName: "left object"
        x: -120; z: 400
        eulerRotation.x: -15
        eulerRotation.y: 10
        Model {
            source: "#Cube"
            pickable: true // <-- important!
            materials: DefaultMaterial {
                diffuseMap: Texture {
                    sourceItem: BusyBox {
                        id: leftBusybox
                        x: 20 - width
                        layer.enabled: true // to make a Texture available
                        layer.textureSize: Qt.size(512, 512) // <-- suitable resolution
                    }
                }
            }
        }
    }
```

Event Delivery to Subscene (simplified)



Virtual Screen on a Model

```
import QtQuick
import QtQuick3D
import QtQuick3D.Particles3D
import QtQuick.Controls
import QtQuick.Controls.Material

View3D {
    id: view
    width: 1600
    height: 1600
    y: 100

    environment: SceneEnvironment {
        clearColor: "black"
        backgroundMode: SceneEnvironment
        antialiasingMode: SceneEnvironme
        lightProbe: Texture {
            source: "blinds_2k.hdr"
        }
        probeOrientation: Qt.vector3d(-3
        probeExposure: mainScreen.ambier
    }

    SpotLight {
        z: 300
        brightness: mainScreen.taskLight
        ambientColor: Qt.rgba(0.2, 0.2,
        eulerRotation.y: -20
        coneAngle: 50
        innerConeAngle: 10
    }
```



Interactive 3D app

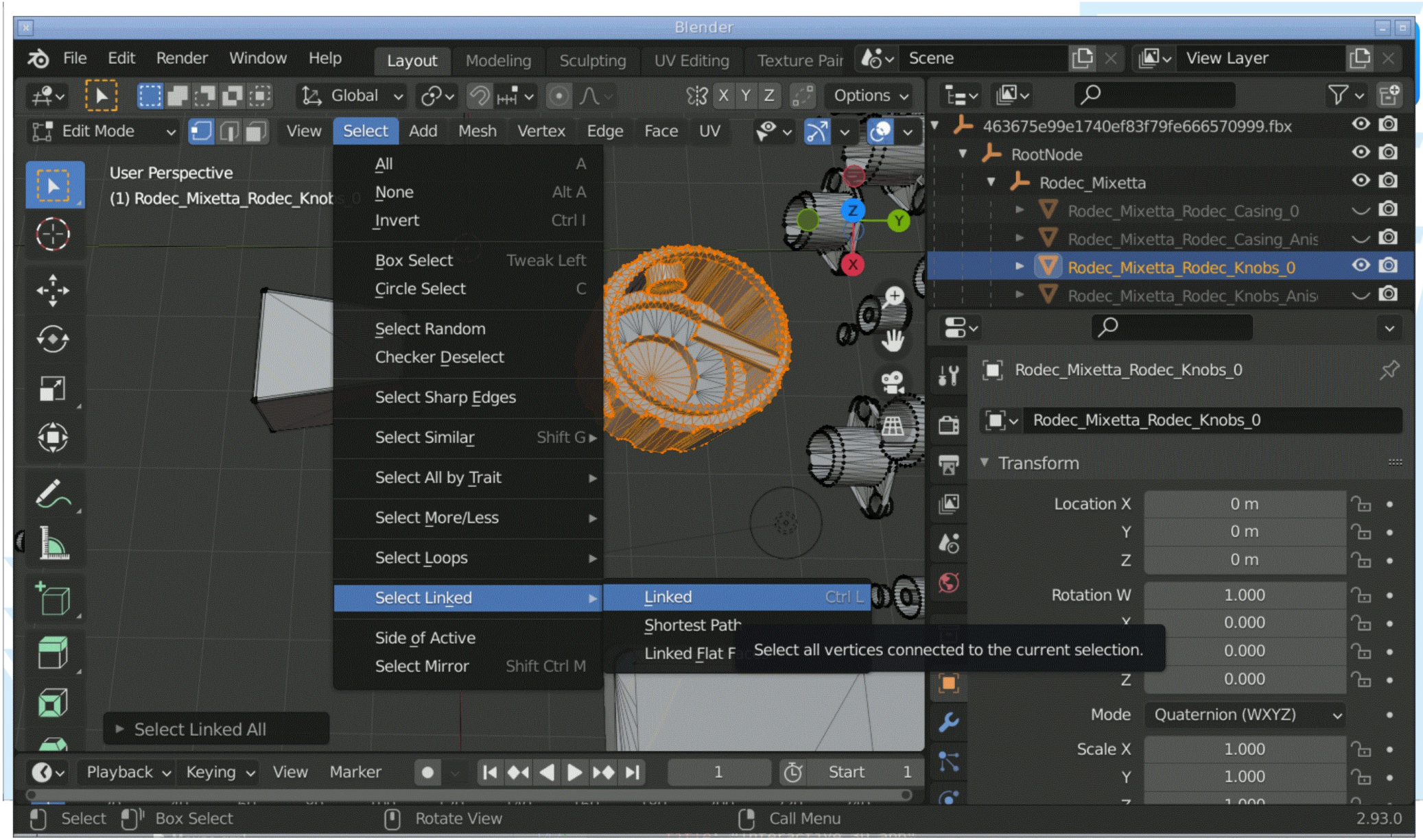
```
import QtQuick
import QtQuick.Controls
import QtQuick.Dialogs

import Qt.labs.settings
```


```
        tape1rot: slider1.value
        tape2rot: slider1.value

        microbassRot: mbwh.rotation
        levelRot: slider1.value
```

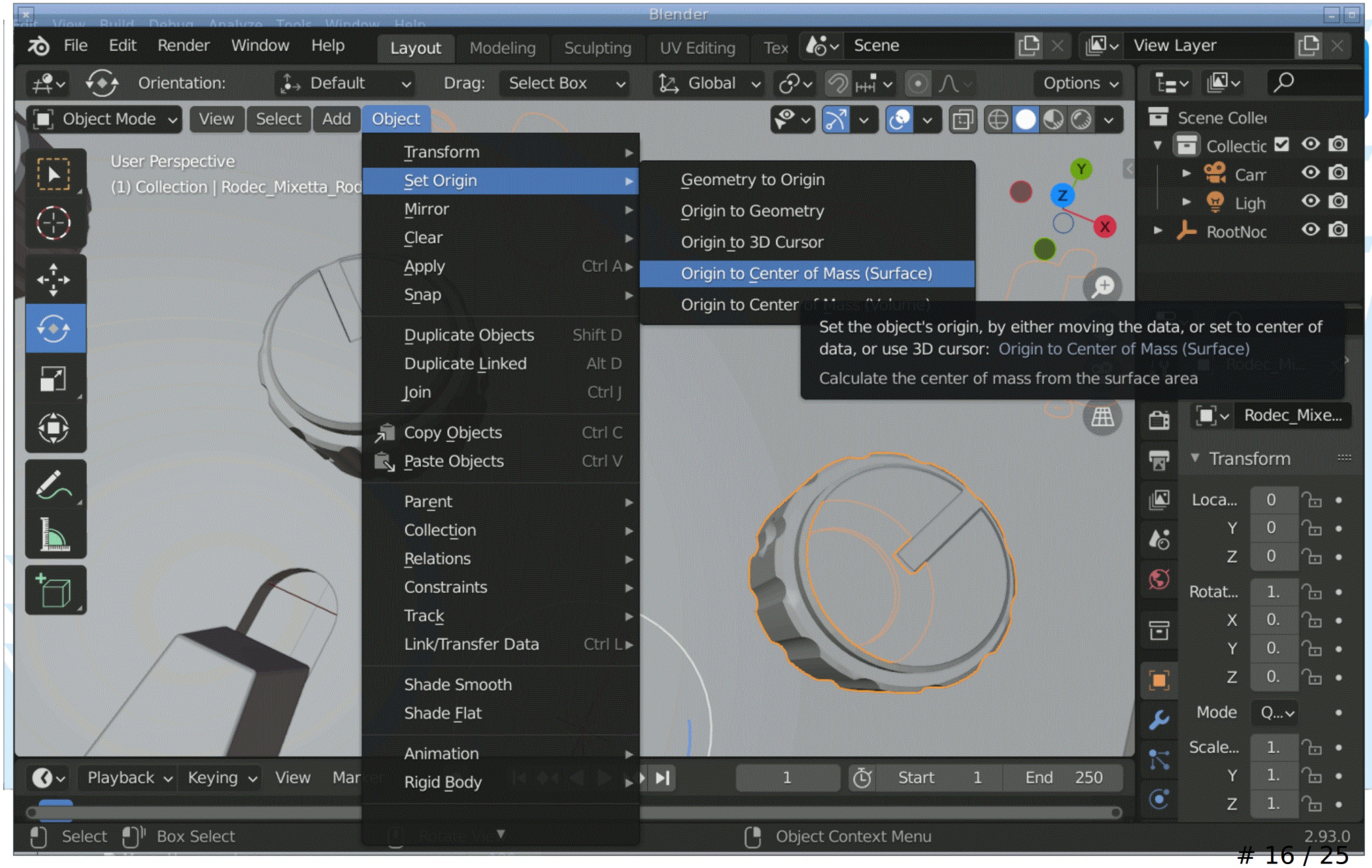
Preparing third-party GLTF for QQ3D

- Download something interesting (http://skfb.ly/ etc.)
- Import into Blender
- Split meshes of interactive elements into separate objects
- Set origins of rotable elements to centers
- Set 3D cursor to origin
- Move/rotate interactive controls to 'zero'
- Re-export to gltf / glb
- $ balsam myfile.glb
- write a QML viewer and look at Myfile.qml in it

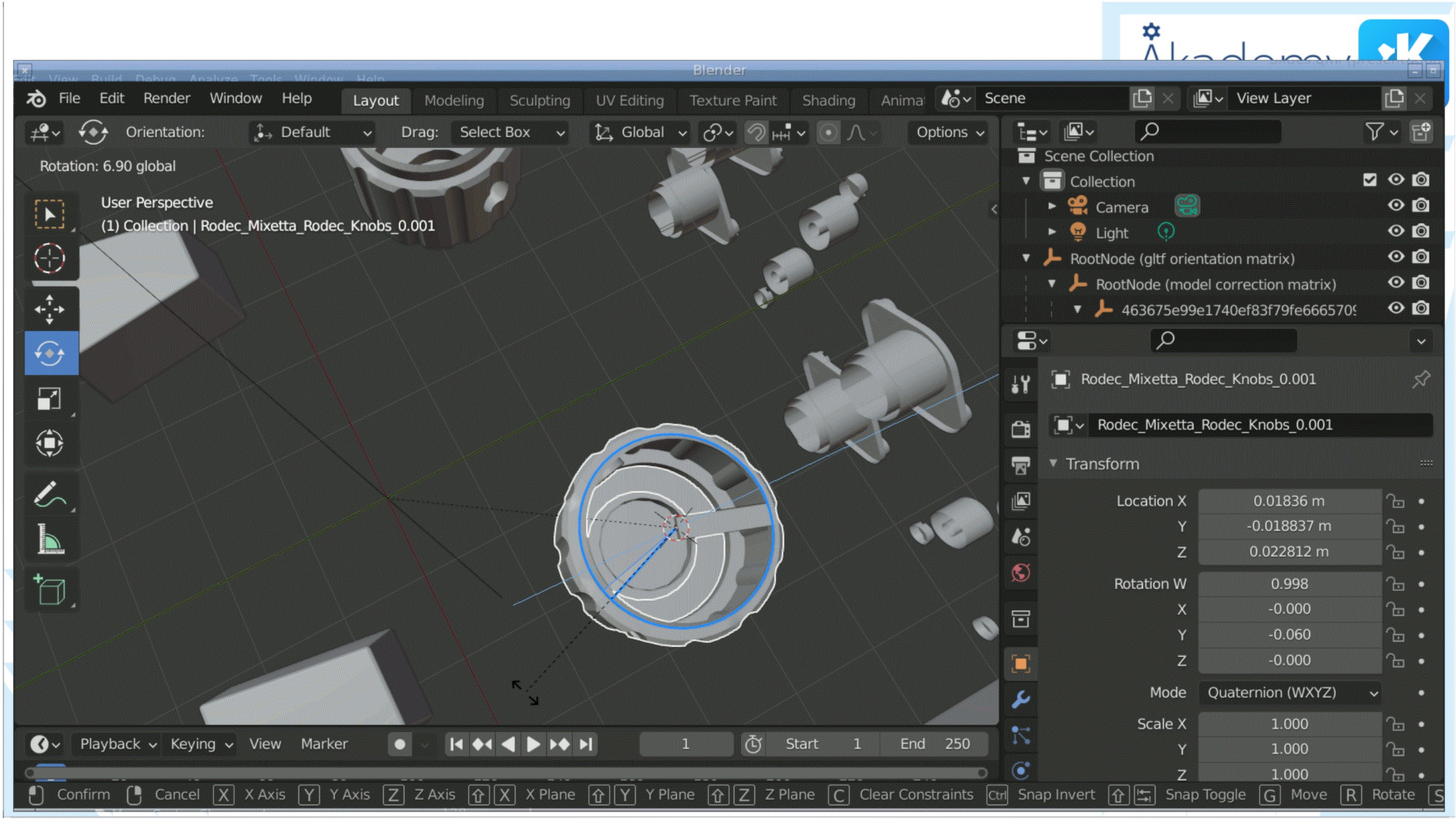
Blender: lasso vertices, select linked vertices



Blender: set origin



Blender: rotate an object

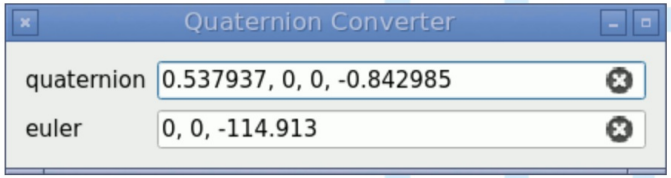


Balsam output

```
Node {
    id: rootNode
    Node {
        id: rodec_Mixetta
        rotation: Qt.quaternion(0.707107, -0.707107, 0, 0)
        scale.x: 100
        scale.y: 100
        scale.z: 100
        Model {
            id: rodec_Mixetta_Rodec_Casing_0
            source: "meshes/rodec_Mixetta_Rodec_Casing_0.mesh"

            PrincipledMaterial {
                id: rodec_Casing_material
                baseColorMap: Texture {
                    source: "maps/Rodec_Casing_Aniso_baseColor.png"
                    generateMipmaps: true
                    mipFilter: Texture.Linear
                }
                opacityChannel: Material.A
                metalnessMap: Texture {
                    source: "maps/Rodec_Casing_Aniso_metallicRoughness.png"
                    generateMipmaps: true
                    mipFilter: Texture.Linear
                }
                metalnessChannel: Material.B
                roughnessMap: Texture {
                    source: "maps/Rodec_Casing_Aniso_metallicRoughness.png"
                    generateMipmaps: true
                    mipFilter: Texture.Linear
                }
```

Make Balsam output interactive

- Simplify the QML (reduce nesting etc.)
- Rotatable element? convert quaternion to Euler angles
  - QQuaternion::toEulerAngles()
- Make Models pickable
- Add handlers to them
- Make bindings to rotate, drag etc.
  - DragHandler.persistentTranslation
  - WheelHandler.rotation
  - TapHandler.pressed
  - PinchHandler.scale
  - PinchHandler.persistentTranslation is missing so far



Rigging Balsam output

```
+    property real levelRot: 0
+
+    property real leftMeterRot: 0
+    property real rightMeterRot: 0
+
+    property alias phono1sliderValue: rodec_Mixetta_Rodec_Slider_Phono1.value
+
+    property alias tweak1: powerCube.position
+    property real tweak2
+
+//    property alias rot1: rodec_Mixetta_Rodec_Knobs_Aniso_Phono1_wh.rotation
+//    property alias rot2: rodec_Mixetta_Rodec_Knobs_Phono1_wh.rotation
+
     Node {
         id: rootNode__gltf_orientation_matrix_
         rotation: Qt.quaternion(0.707107, -0.707107, 0, 0)
@@ -115,7 +138,7 @@ Node {
                             x: 0.0855515
                             y: -0.0758887
                             z: 0.019
-                            rotation: Qt.quaternion(0.537937, 0, 0, -0.842985)
+                            eulerRotation: Qt.vector3d(0, 0, -115 - rOOT.levelRot)
                             scale.x: 1
                             scale.y: 1
                             source: "meshes/rodec_Mixetta_Rodec_Knobs_Aniso_Level.mesh"
@@ -165,7 +188,7 @@ Node {
                             x: 0.160732
                             y: -0.0208635
                             z: 0.019
-                            rotation: Qt.quaternion(0.969134, 0, 0, 0.246536)
+                            eulerRotation: Qt.vector3d(0, 0, 28.5 - rOOT.microbassRot)
                             source: "meshes/rodec_Mixetta_Rodec_Knobs_Aniso_MicroBass.mesh"
```

Ray Picking

```
import QtQuick
import QtQuick3D
import QtQuick3D.Helpers
import QtQuick3D.Particles3D

View3D {
    id: view
    width: 1280; height: 720; y: 100
    camera: camera

    environment: SceneEnvironment {
        backgroundMode: SceneEnvironment.SkyBox
        lightProbe: Texture { source: "maps/OpenfootageNET_lowerAustria01-1024.hdr" }
        probeExposure: 2
    }

    DirectionalLight { eulerRotation.x: -90; shadowFactor: 100 }

    Node {
        id: character
        position: "200, 300, -50"
        eulerRotation.y: -60
        SimpleSpaceship {}
        Node {
            id: ray
            property int length: 0
            property real scaleWidth: length ? 0.07 : 0.04
            scale: Qt.vector3d(scaleWidth, scaleWidth, length ? length/100 : 50.0)
            Model {
                z: -50; eulerRotation.x: -90
```

Stuff left to work on

- Ensure that we can really use handlers in 3D
- Hover bugs
- WasdController stealing events
- Controls stealing events

Suggestions for the community

- Play with it!
- Imagine new use cases
- Design standard toolbar icons for 3D applications

Other demo videos

- Hackathon spaceship: https://d.tube/#!/v/ecloud/QmWJVDywsnaK8WCFTb8bMLUwEKmxvfBQVw8G3ATepbGhqB
- Digital assistant: https://d.tube/#!/v/ecloud/QmVATRbK6pDpwXwS1sTfUagpVCtdGye1v66mgonfvYTDGF

Interactive UIs in Qt Quick 3D

Shawn Rutledge
shawn.rutledge@qt.io
ecloud on #qt-labs, #qt-quick etc.

https://github.com/ec1oud/qtquick3d-input-demo
https://github.com/ec1oud/qt-presentations : qtquick3d-interactive-ui branch