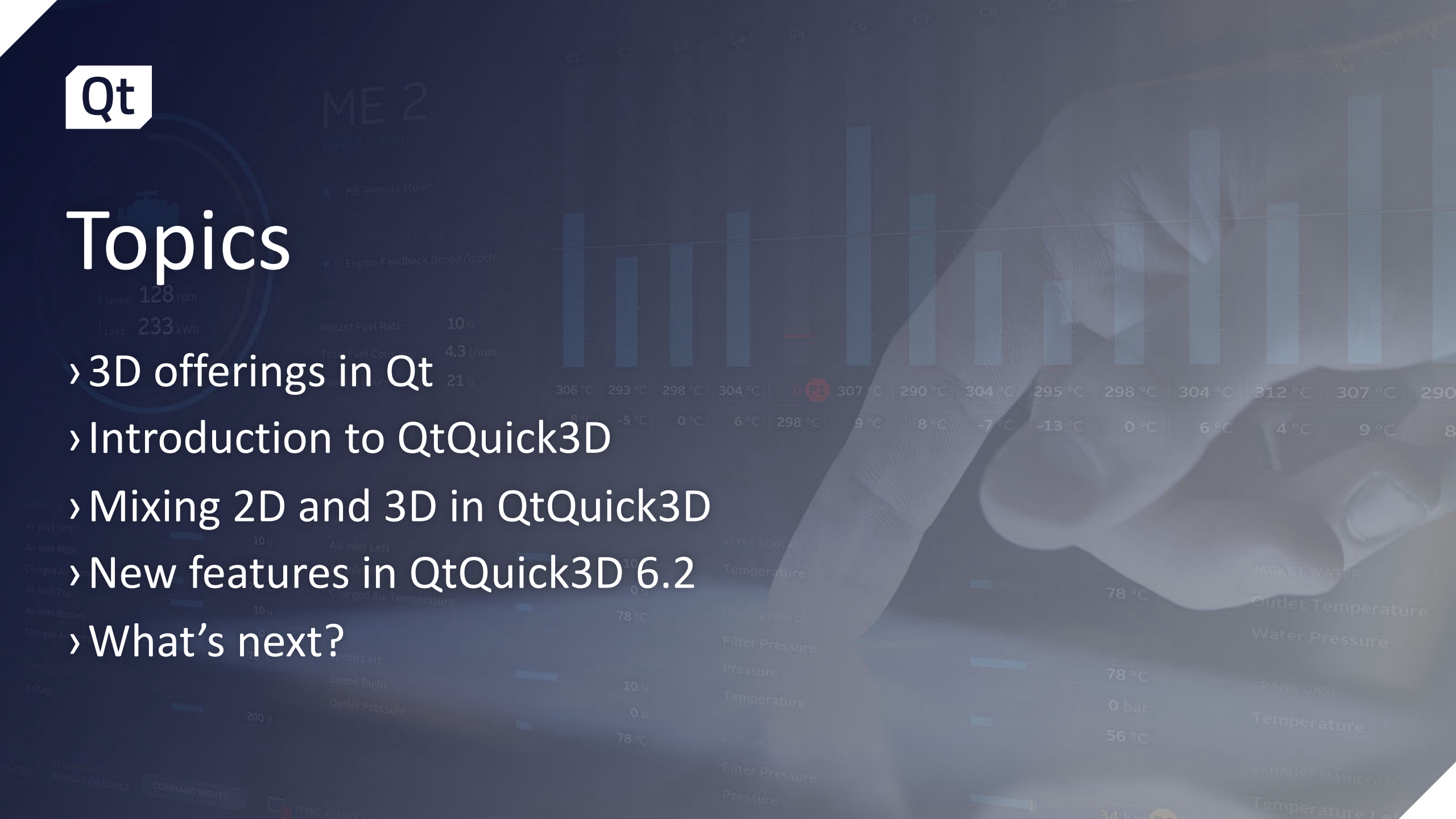
QtQuick3D 6.2 introduction

Christian Strømme

2021



Topics

- 3D offerings in Qt
- Introduction to QtQuick3D
- Mixing 2D and 3D in QtQuick3D
- New features in QtQuick3D 6.2
- What’s next?

3D Offerings in Qt

- Qt3D, Qt 3D Studio and QtQuick3D
  - Qt3D
    - High-level abstraction with low-level APIs.
    - Extremely flexible.
    - Flexibility and abstraction comes at a cost.
  - Qt 3D Studio
    - A complete suite of formats and concepts foreign to Qt.
    - Heavily tied to OpenGL.
    - Closer integration with Qt proven difficult.
  - QtQuick3D
    - High-level APIs with high-level concepts.
    - Good integration with QtQuick.
    - Replaces Qt 3D Studio.

3D Offerings in Qt - Which one should I choose?

› If you must ask, the right choice is most likely QtQuick3D.

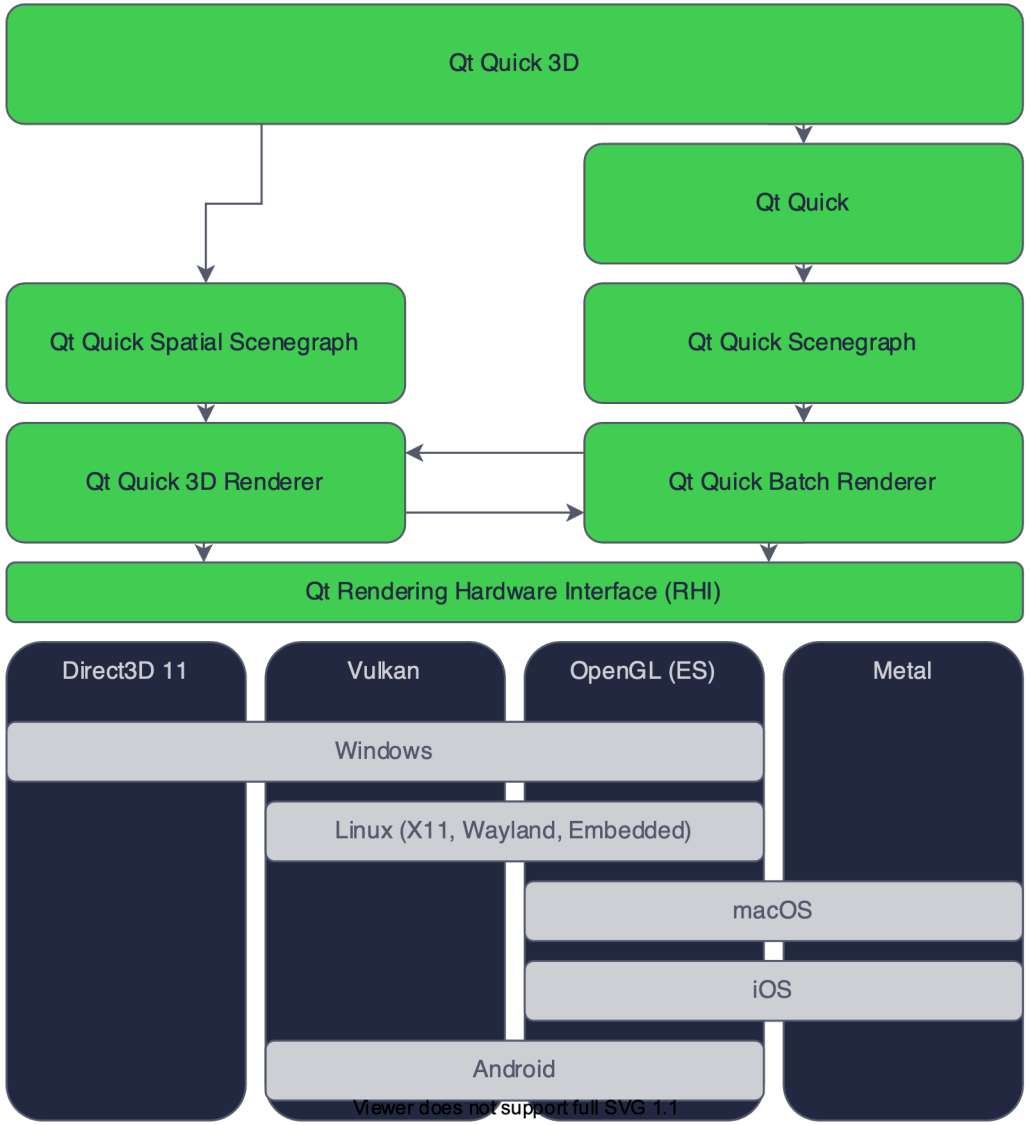
› Qt3D offers a completely different abstraction that will make sense for those looking for it.

Introduction to QtQuick3D

- Primary goals:
  - Simple and easy to use.
    - High level concepts with sane default values.
    - No prior knowledge about 3D or 3D APIs needed.
    - Excellent documentation.
  - Light weight.
    - Embedded focused.
    - Understanding, reasoning and debugging the engine should be “easy.”
  - Integrate well with QtQuick.
    - Mixing 2D and 3D should be easy.
    - “Unified rendering.”
    - Good looking text.
  - Tooling.

Architectural overview

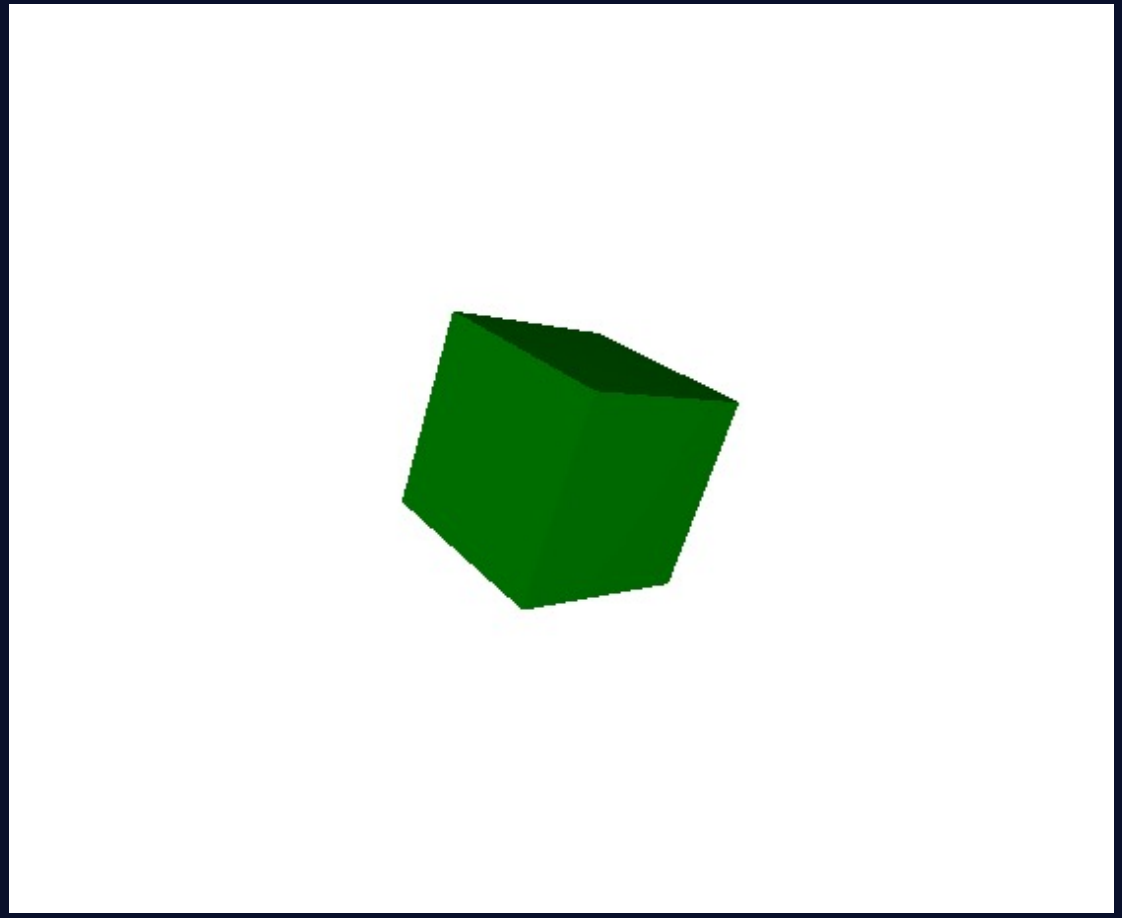
- Sits on top of QtRhi
- Has its own spatial scene graph
- Interacts with the QtQuick Renderer



Assets and Asset conditioning

- Balsam
- Shadergen
- RuntimeLoader (6.2)

Minimal scene



```qml
import QtQuick.Window
import QtQuick3D
import QtQuick3D.Helpers

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello QtQuick3D")

    View3D { // View into the 3D scene
        id: view3d
        anchors.fill: parent

        PerspectiveCamera { // Camera
            id: camera
            position: Qt.vector3d(0, 0, 400)
        }

        DirectionalLight { // Light
            id: light
        }

        Model { // Model
            source: "#Cube"
            rotation: Quaternion.fromEulerAngles(25, 45, 0)
            materials: PrincipledMaterial {
                baseColor: "green"
            }
        }
    }
}
```

```qml
import QtQuick.Window
import QtQuick3D
import QtQuick3D.AssetUtils

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello QtQuick3D")

    View3D { // View into the 3D scene
        id: view3d
        anchors.fill: parent
        environment: SceneEnvironment {
            lightProbe: Texture {
                source: "field.hdr"
            }
            backgroundMode: SceneEnvironment.SkyBox
        }

        PerspectiveCamera { // Camera
            id: camera
            position: Qt.vector3d(0, 0, 200)
        }

        DirectionalLight { // Light
            id: light
        }

        RuntimeLoader {
            scale: Qt.vector3d(80, 80, 80)
            rotation: Quaternion.fromEulerAngles(15, 35, 0)
            source: "helmet.glb"
        }
    }
}
```

Mixing 2D and 3D

- Both 2D and 3D items defined in the same scene
- Inline rendering for both 2D and 3D
- Scenes can still be rendered into a texture

Direct path

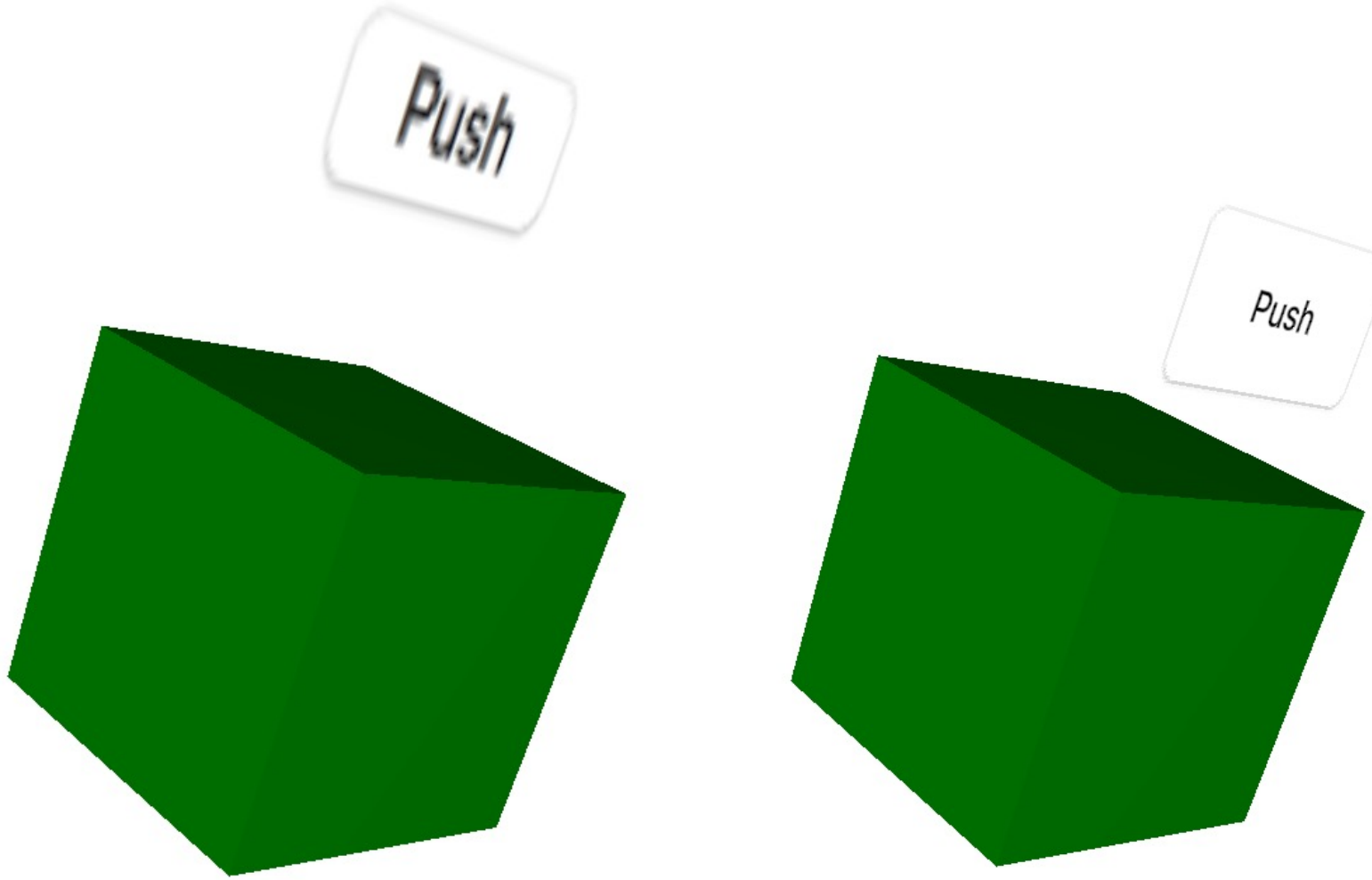
- Items are rendered without the need and cost of rendering into a texture.
- Transforms are applied to the 2D items resulting in a more pleasing result.
- However, there are some caveats.
  - Different scene coordinates (y-axis)
    - Can be solved by wrapping items in a node.
  - Anchoring.

```
import QtQuick.Window
import QtQuick3D
import QtQuick3D.AssetUtils
import QtQuick.Controls

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello QtQuick3D")

    View3D { // View into the 3D scene
        id: view3d
        anchors.fill: parent

        PerspectiveCamera { // Camera
            id: camera
            position: Qt.vector3d(0, 0, 400)
        }

        DirectionalLight { // Light
            id: light
        }

        Node {
            rotation: Quaternion.fromEulerAngles(25, 45, 0)
            Button {
                text: "Push"
                width: 100
                height: 60
                y: -120 // Y is up, in the 3D scene.
                onClicked: cubeMaterial.baseColor = "red"
            }

            Model { // Model
                source: "#Cube"
                materials: PrincipledMaterial {
                    id: cubeMaterial
                    baseColor: "green"
                }
            }
        }
    }
}
```

Texture path

- QtQuick items are rendered into texture
- Flexible
- Again, there are some caveats

```
import QtQuick.Window
import QtQuick3D
import QtQuick3D.AssetUtils
import QtQuick.Controls

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello QtQuick3D")

    View3D { // View into the 3D scene
        id: view3d
        anchors.fill: parent

        PerspectiveCamera { // Camera
            id: camera
            position: Qt.vector3d(0, 0, 400)
        }

        DirectionalLight { // Light
            id: light
        }

        Model {
            rotation: Quaternion.fromEulerAngles(25, 45, 0)
            Model {
                source: "#Rectangle"
                position: Qt.vector3d(0, 120, 0)
                scale: Qt.vector3d(1, 0.6, 0)
                pickable: true
                materials: PrincipledMaterial {
                    baseColorMap: Texture {
                        sourceItem: Button {
                            text: "Push"
                            width: 100
                            height: 60
                            onClicked: cubeMaterial.baseColor = "red"
                        }
                    }
                }
            }

            Model { // Model
                source: "#Cube"
                materials: PrincipledMaterial {
                    id: cubeMaterial
                    baseColor: "green"
                }
            }
        }
    }
}
```



New features in 6.2

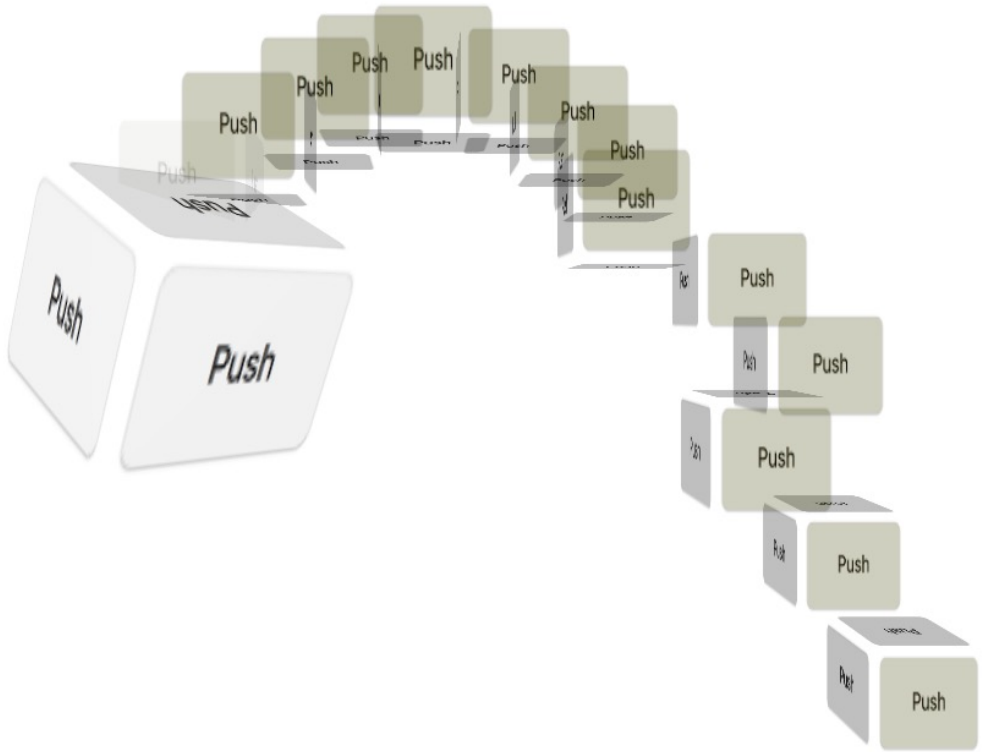
- Instancing
- Particles
- Run-time asset loading
- Parallax occlusion mapping

New features in 6.2

- Instancing
- Particles
- Run-time asset loading
- Parallax occlusion mapping

New features in 6.2

- Instancing
- Particles
- Run-time asset loading
- Parallax occlusion mapping

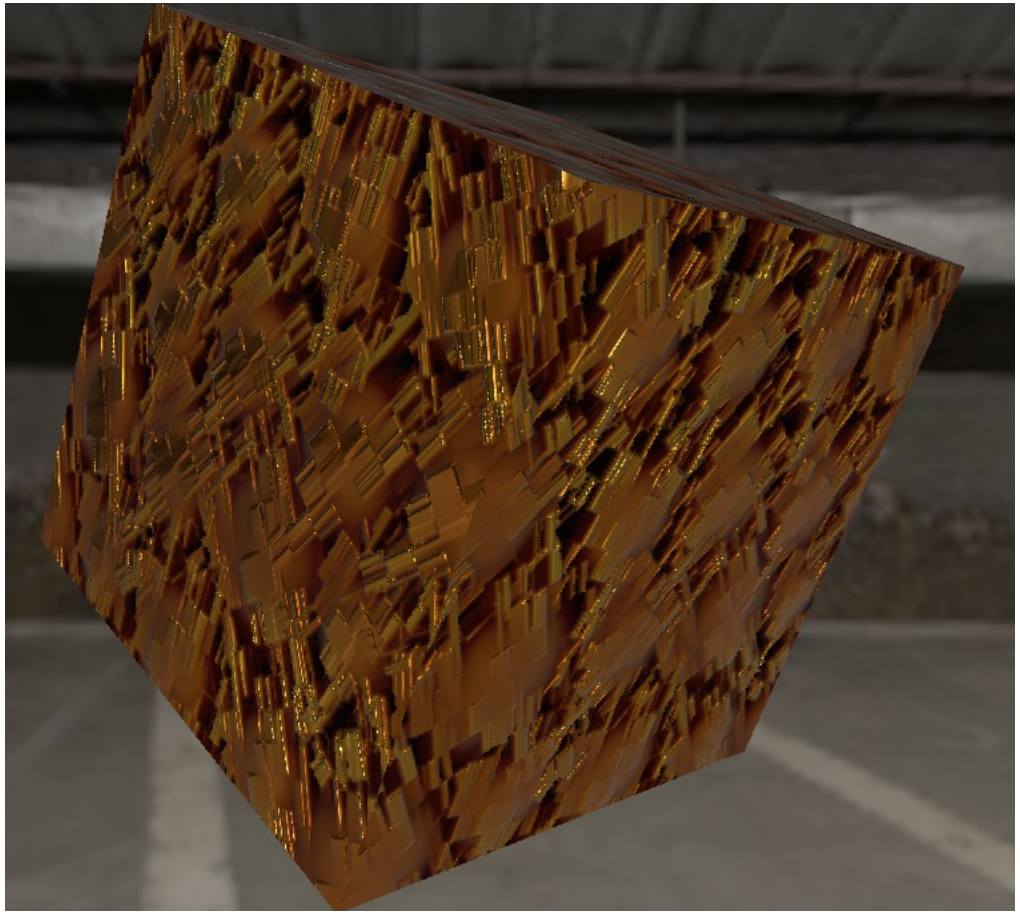


New features in 6.2

- Instancing
- Particles
- Run-time asset loading
- Parallax occlusion mapping

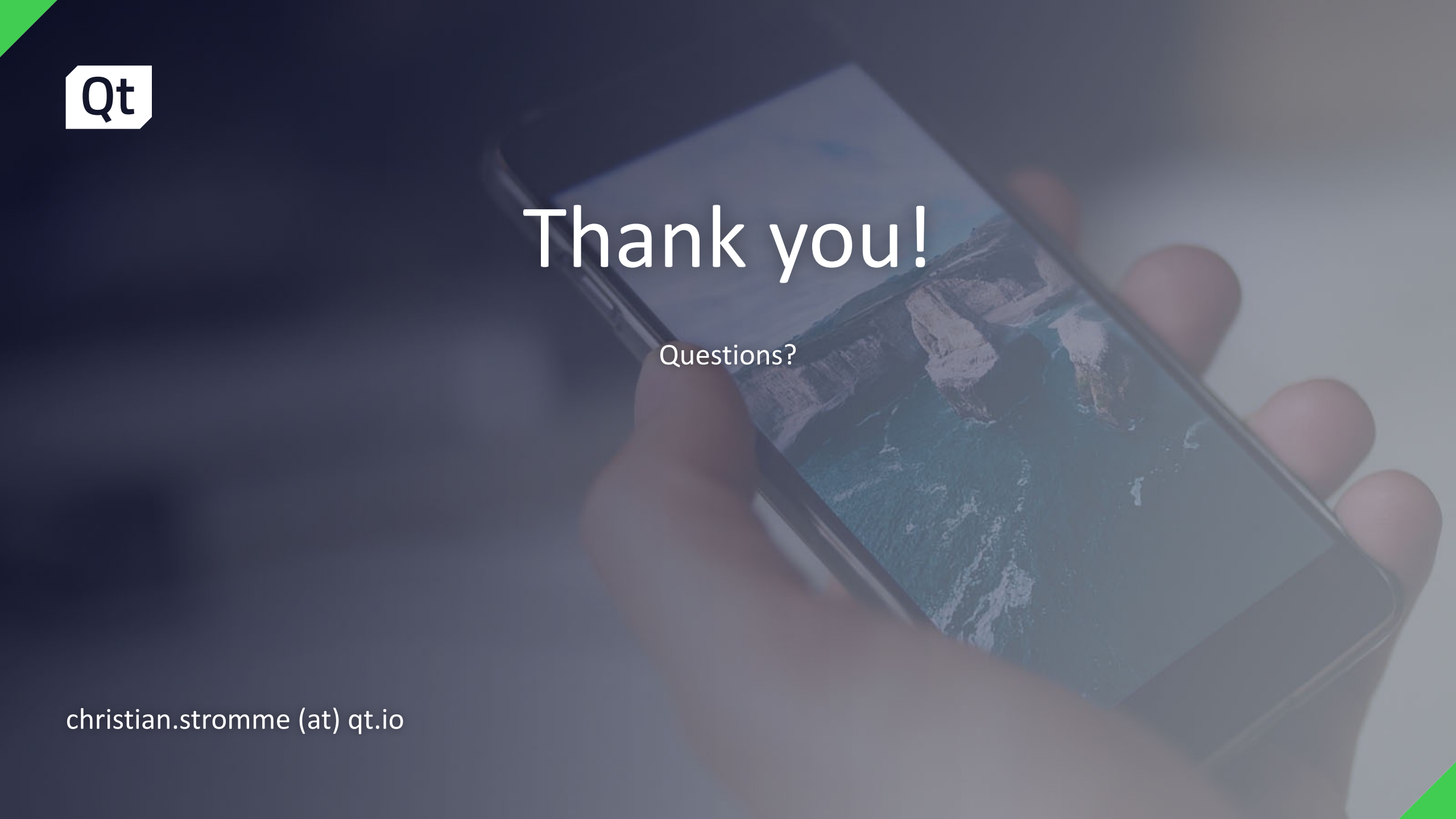
New features in 6.2

- Instancing
- Particles
- Run-time asset loading
- Parallax occlusion mapping



What’s next?

- Tooling
  - Design Studio
  - Asset conditioning pipeline
- Getting more feedback



Thank you!

Questions?

christian.stromme (at) qt.io