Using the Qt Scene Graph from C++ with QSkinny

https://github.com/uwerat/qskinny

What is QSkinny? What is the scene graph? I hope we can answer that during this presentation.

Who am I?

» working on Qt since 2008

» former QtNetwork maintainer

» @peha23 on Twitter

https://github.com/uwerat/qskinny

on Twitter or of course after the talk etc.

What is this talk about?

Using the Qt graphic stack from C++

https://github.com/uwerat/qskinny

How many of you have used QtWidgets? How many have used QML?
-> Desktop is still using QtWidgets (e.g. QtCreator)
Why?
-> It's C++ (type safety, static code checking, not having to learn C+
+)
-> QtWidgets familiarity
-> Qt for Python
What to take away from this presentation?
-> What is missing in Qt right now
-> How QSkinny works
-> How a combined effort could look like
What is this talk not about?
-> Bashing QML, rather offering options to mix and match C++ more freely (including the option to use only C++)

Agenda

1. QML under the hood
2. The QML / C++ boundary
3. QSkinny
4. Outlook

https://github.com/uwerat/qskinny

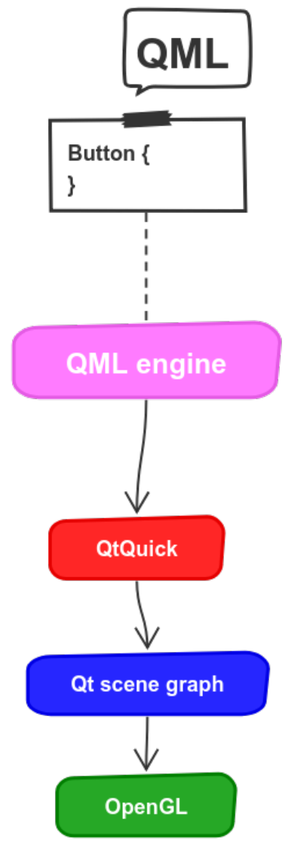
under the hood: How is QML turned to objects that are rendered on the screen? boundary: How much QML do I want in my

Agenda

1. QML under the hood
2. The QML / C++ boundary
3. QSkinny
4. Outlook

https://github.com/uwerat/qskinny

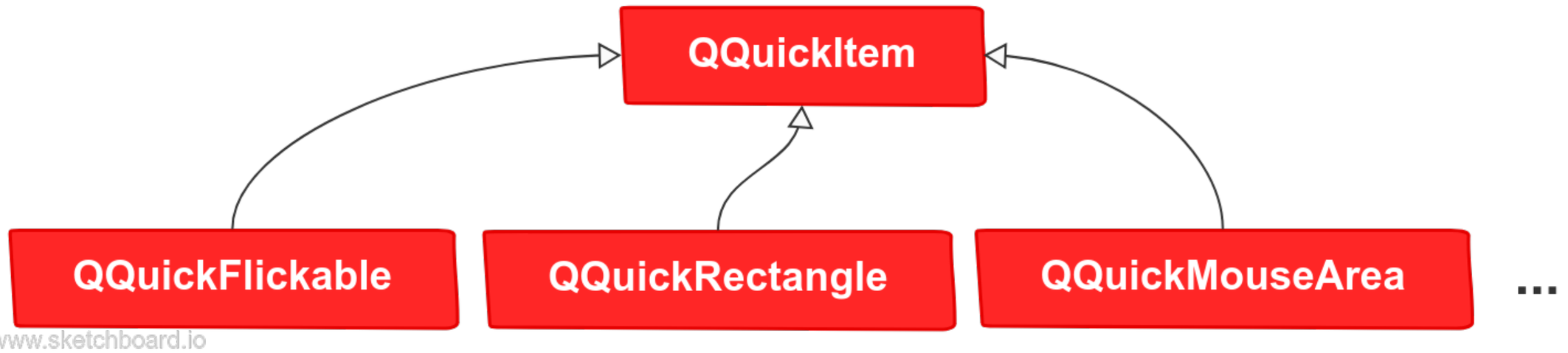
QML under the hood



https://github.com/uwerat/qskinny

- The QML engine "translates" the QML code that the user writes into QtQuickItems (e.g. QQuickRectangle, QQuickText); the API is not public
QtQuick creates scene graph nodes, i.e. graphical representation of objects (e.g. rectangle node etc.)
- scene graph nodes are rendered with OpenGL (and others in the future) and uploaded onto the GPU
- What is the scene graph? A representation of graphical elements that make it performant to be rendered on the graphics card
- interestingly, the scene graph API is "more public" than QtQuick

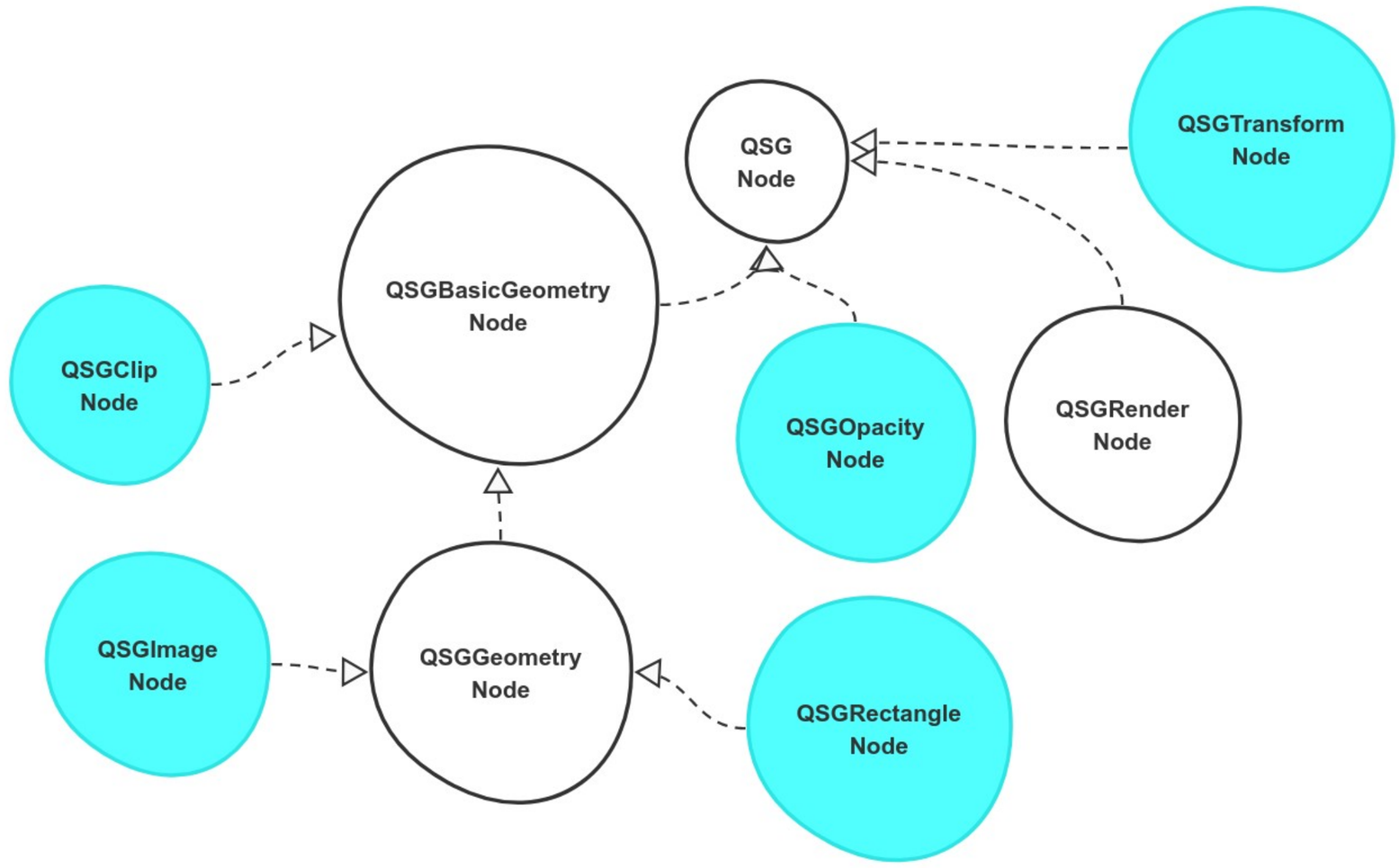
Types of QtQuick items



https://github.com/uwerat/qskinny

QQuickItem is public, the others are not

Types of scene graph nodes



https://github.com/uwerat/qskinny

- Don't remember all the details, just this: All QML objects are ultimately broken down into a set of scene graph nodes
- The blue nodes are important ones

QML example

```
Rectangle {
    id: outterRectangle
    width: 200
    height: 200
    color: "red"
    opacity: 0.5

    Rectangle {
        id: innerRectangle
        width: 50
        height: 50
        clip: true
        anchors.bottom: parent.bottom
        anchors.right: parent.right
        color: "green"
    }
}
```

https://github.com/uwerat/qskinny

clip: Will clip the contents to the bounding rectangle

QQuickRootItem — is a QQuickItem

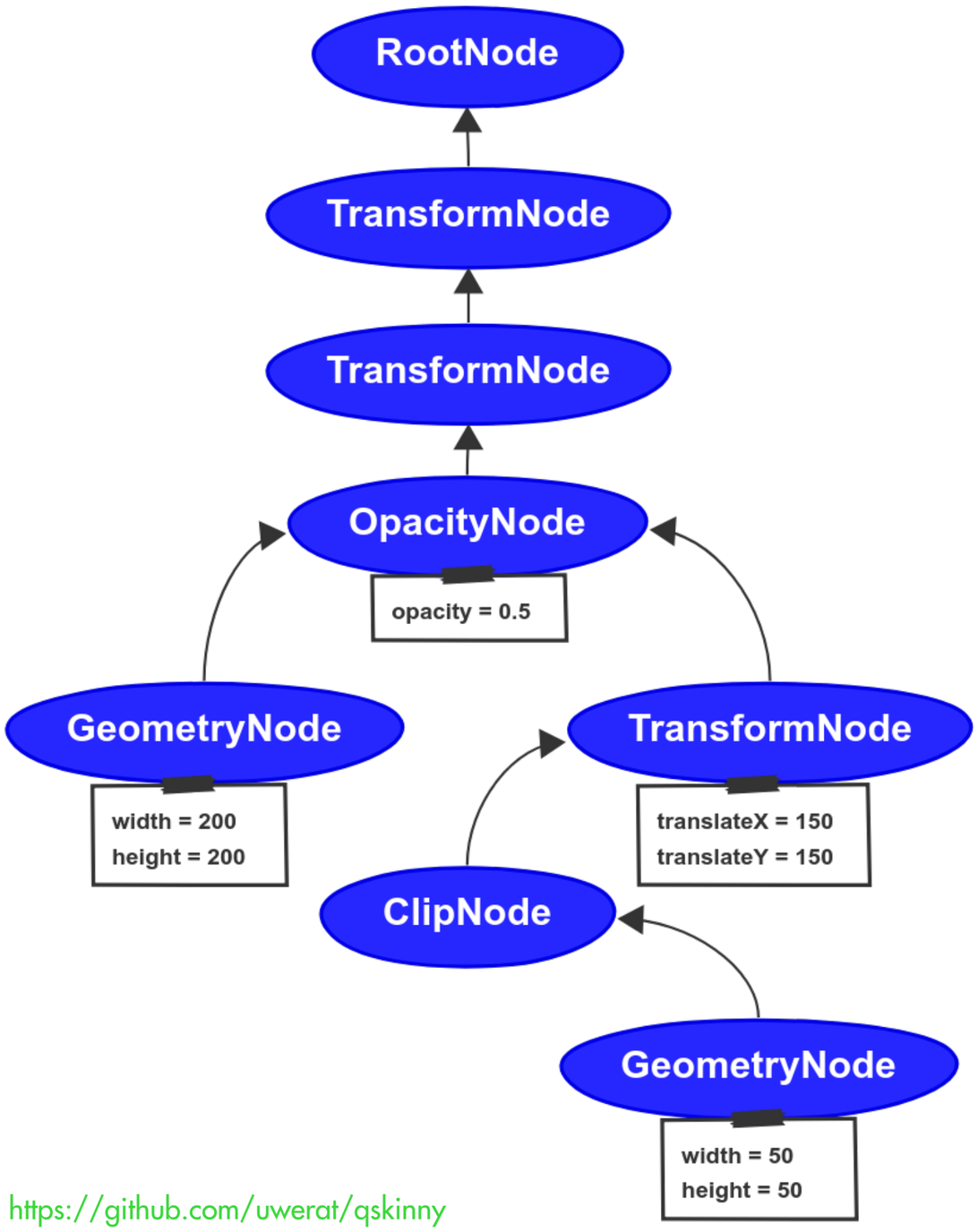
QQuickRectangle — is a QQuickItem

- qreal x = 0
- qreal y = 0
- qreal width = 200
- qreal height = 200
- qreal opacity = 0.5

QQuickRectangle — is a QQuickItem

- qreal x = 150
- qreal y = 150
- qreal width = 50
- qreal height = 50

https://github.com/uwerat/qskinny

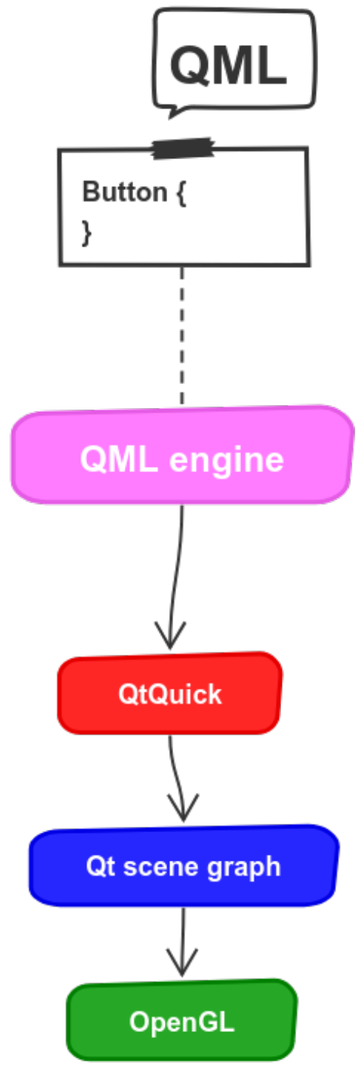


https://github.com/uwerat/qskinny

Agenda

1. QML under the hood
2. The QML / C++ boundary
3. QSkinny
4. Outlook

https://github.com/uwerat/qskinny



https://github.com/uwerat/qskinny

- each QML object is a QObject (or QGadget); this might be not fine-grained enough for gradient stops / speedometer ticks etc.
- Which / how much code do you want to write in QML and how much in C++?

QtQuickControls 1

Write everything in QML

```
Control {
    id: slider
    (...)
    style: Settings.styleComponent(Settings.style, "SliderStyle.qml", slider)
    property Component tickmarks: Repeater {
    Rectangle {
        color: "#777"
        width: 1
        height: 3
        y: (...)
        x: (...)
        }
    }
}
```

https://github.com/uwerat/qskinny

problem: too slow
has been abandoned by now
too many QObjects created
(see e.g. tickmarks above)

QtQuickControls 2

some parts QML, some C++

```
--- qquickslider_p.h:

class Q_QUICKTEMPLATES2_PRIVATE_EXPORT QQuickSlider : public QQuickControl
{
    Q_OBJECT
    Q_PROPERTY(qreal from READ from WRITE setFrom NOTIFY fromChanged FINAL)
    Q_PROPERTY(qreal to READ to WRITE setTo NOTIFY toChanged FINAL)
    (...)
};
```

```
--- Slider.qml:

T.Slider {
    id: control
}
```

https://github.com/uwerat/qskinny

problem: 1. private API
2. font / palette / locale cannot inherited for user types
-> creates boundary: Qt code C++, user code QML. Was not designed with extensibility in mind
user defined controls cannot be written in C++
also: QQC2 cannot be used without a QML engine right now
historically, the portion of QML is getting less and less

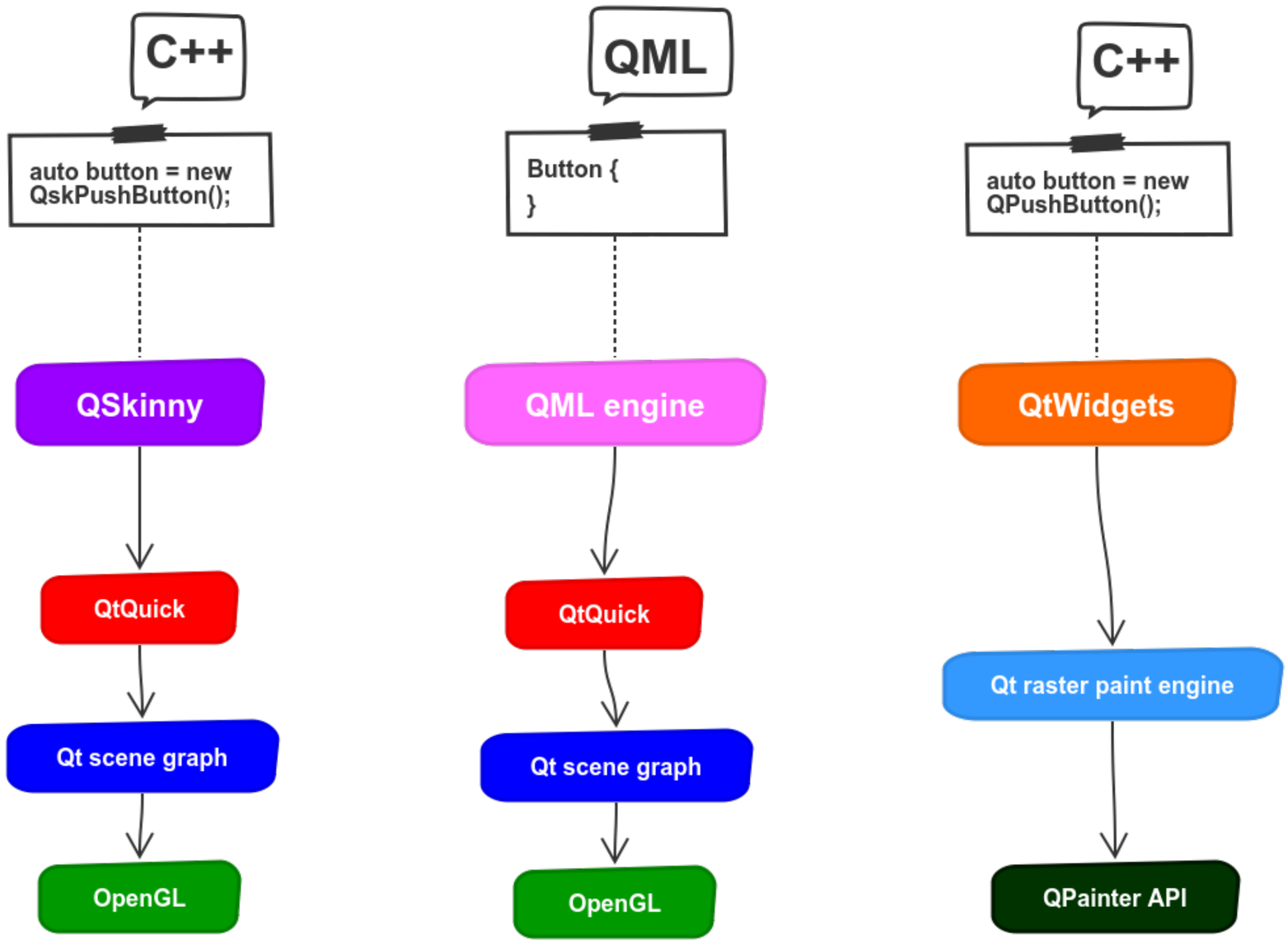
Agenda

1. QML under the hood
2. The QML / C++ boundary
3. QSkinny
4. Outlook

https://github.com/uwerat/qskinny

big question: Why not make QQC2 public and let people inherit?

-> there are some more differences which we will see soon



https://github.com/uwerat/qskinny

QSkinny and QML both use QtQuick -> they could be mixed and matched, and there are examples for that

QSkinny makes QML optional though (and in one project there is no QML) code-wise it looks like Widgets code (we will see an example later), but is hardware accelerated just like QML

QSkinny design goals

» lightweight

» flexible theming

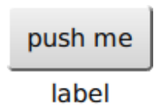
» dynamic sizing

https://github.com/uwerat/qskinny

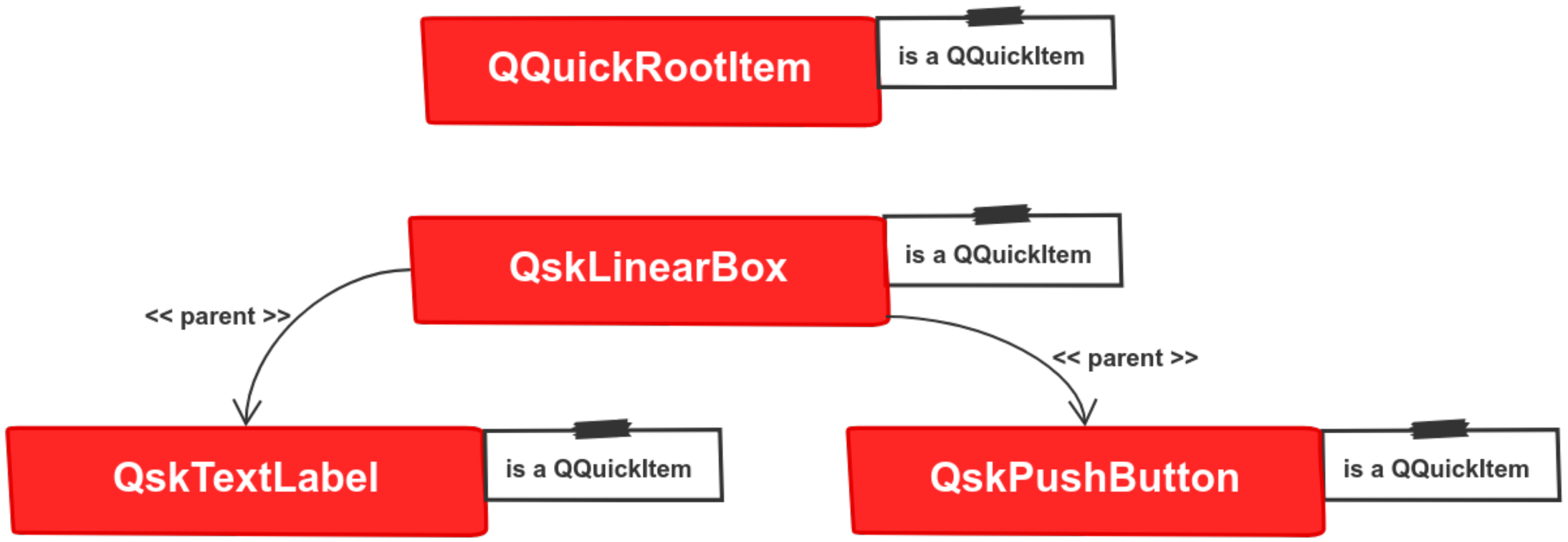
lightweight: don't create unnecessary QObjects or QQuickItems / cache items / don't create scene graph nodes until they are necessary (e.g. QML creates all nodes even if they are outside of the bounding rectangle)
theming: QStyle not enough / separate content from theming
dynamic sizing: vector graphics / layouts. Why dynamic sizing? screens on embedded are fixed, but layouts need to be resized e.g. when changing theme / language, or switching to another physical screen size. Vector graphics allow for size adaptations (widthForHeight() etc.)

QSkinny API

```
QskWindow window;
auto box = new QskLinearBox(Qt::Vertical);
auto button = new QskPushButton("push me", box);
auto label = new QskTextLabel("label", box);
window.addItem(box);
window.show();
```

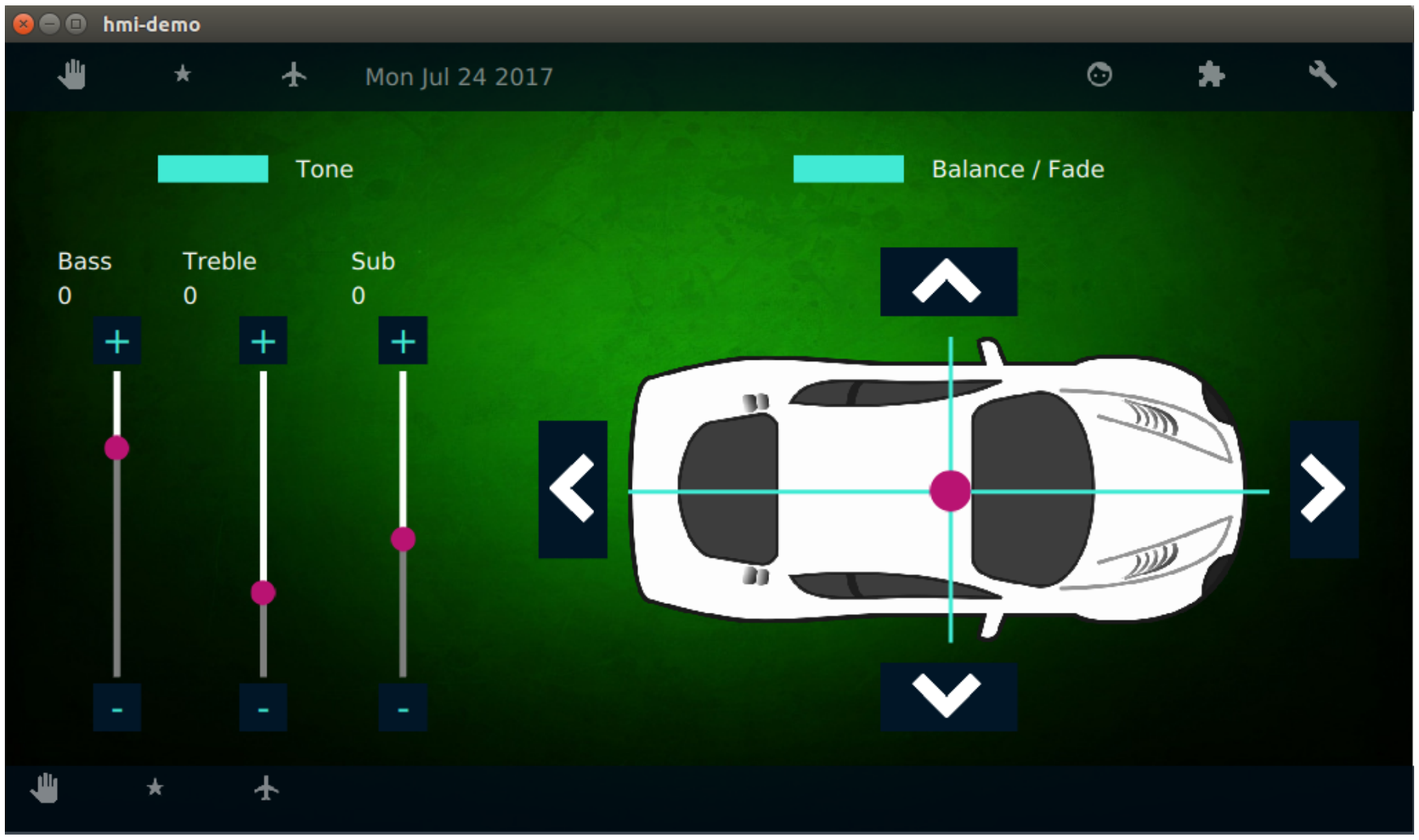


https://github.com/uwerat/qskinny



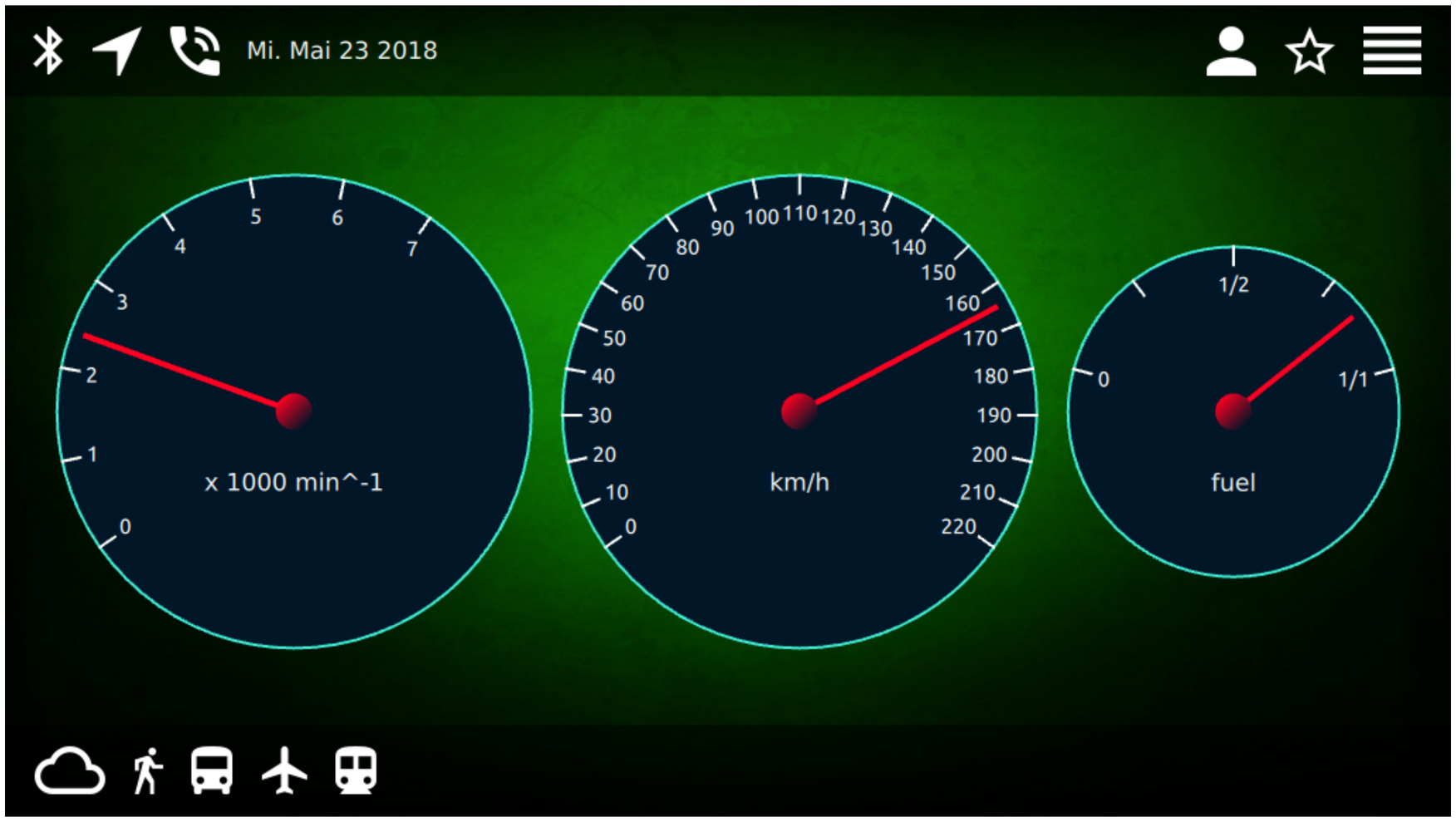
https://github.com/uwerat/qskinny

example



https://github.com/uwerat/qskinny

example



https://github.com/uwerat/qskinny

Agenda

1. QML under the hood
2. The QML / C++ boundary
3. QSkinny
4. Outlook

https://github.com/uwerat/qskinny

QSkinny

» polishing

» documentation

Qt 6

» new styling?

» opening up QtQuickControls 2?

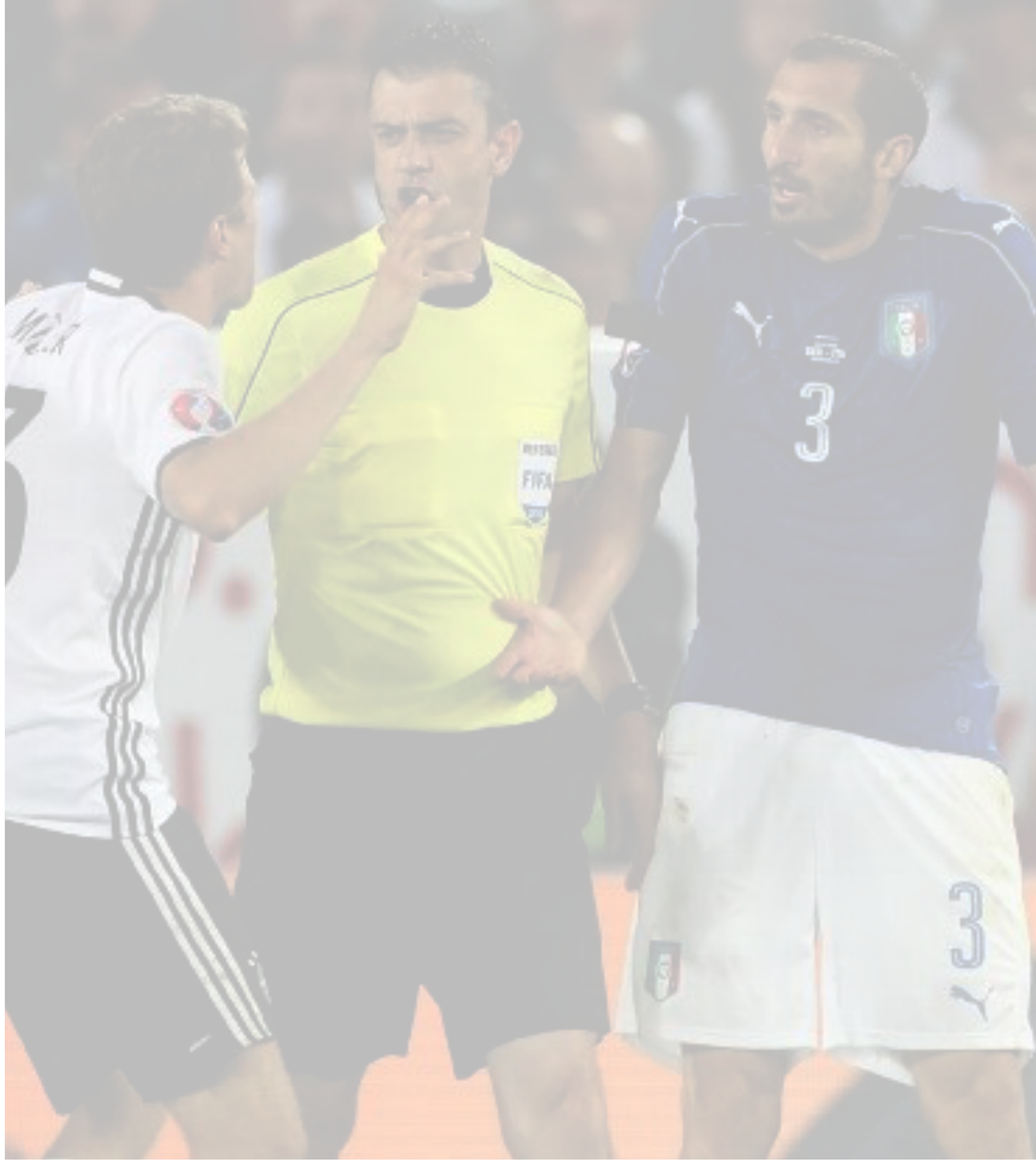
https://github.com/uwerat/qskinny

still: Can people use it without QML engine?

Discussion

feedback to @peha23 on Twitter

https://github.com/uwerat/qskinny



also feel free to contact via QSkinny email address