Åkademy

Almería, 22nd - 27th July 2017

Advances in Qt 3D

Kévin Ottens, Software Craftsman at KDAB

Åkademy

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Advances in Qt 3D

- **Feature Set**
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

What is Qt 3D?

- It is not about 3D!
- Multi-purpose, not just a game engine
- Soft real-time simulation engine
- Designed to be scalable
- Extensible and flexible

Simulation Engine

- The core is not inherently about 3D
- It can deal with several functional domains at once
  - AI, logic, audio, etc.
  - And of course it contains a 3D renderer too!
- All you need for a complex system simulation
  - Mechanical systems
  - Physics
  - ... and also games

Scalability

- Frontend / backend split
  - Frontend is lightweight and on the main thread
  - Backend executed in a secondary thread
    - Where the actual simulation runs
- Non-blocking frontend / backend communication
- Backend maximizes throughput via a thread pool

Extensibility and Flexibility

KDAB

- Functional domains can be added by extending the runtime
  - ... only if there's not something fitting your needs already
- Provide both C++ and QML APIs
- Integrates well with the rest of Qt
  - Pulling your simulation data from a database anyone?
- Entity Component System is used to combine behavior in your own objects
  - No deep inheritance hierarchy

KDAB

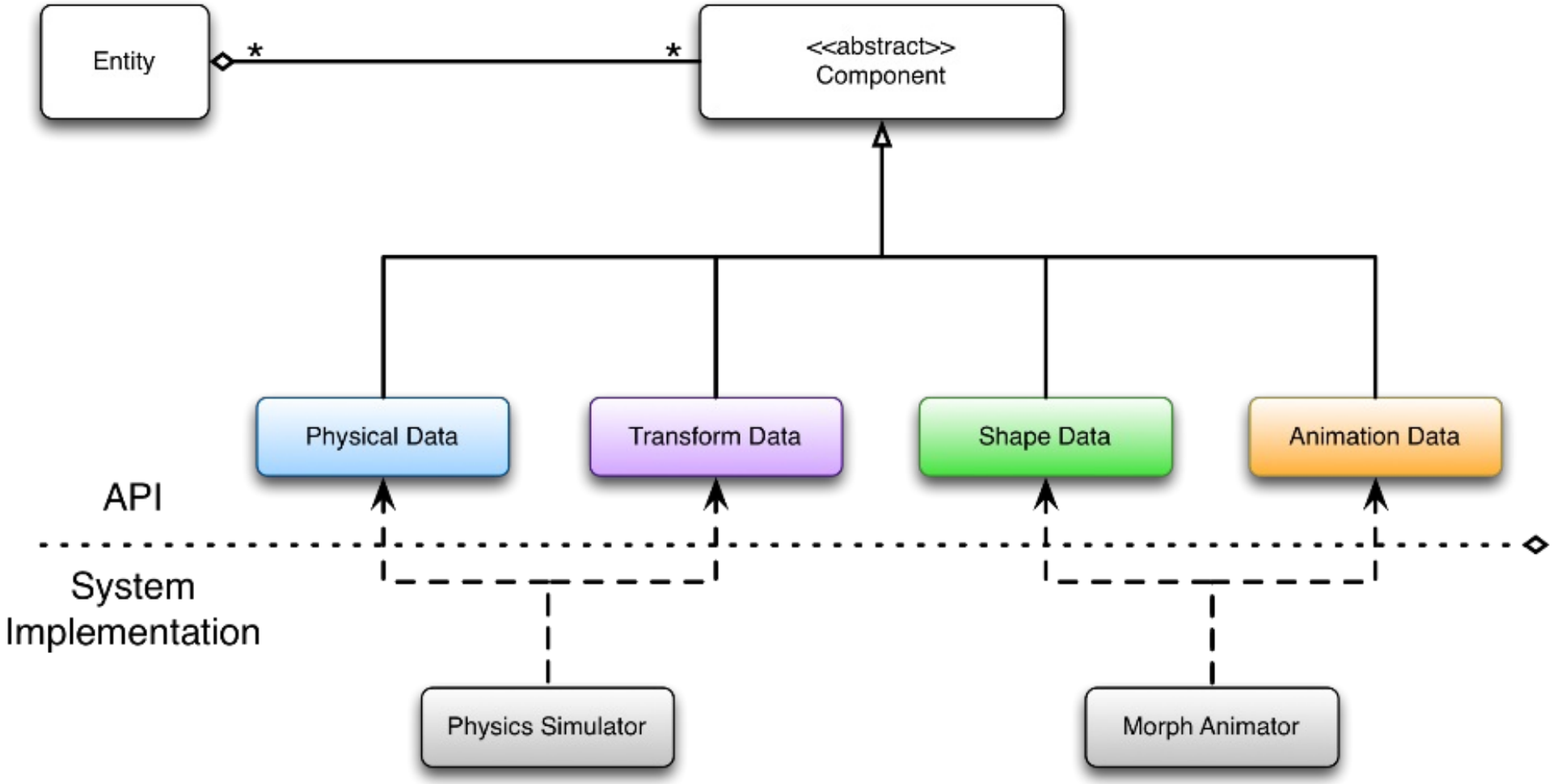
- Feature Set
- **Entity Component System? What's that?**
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

ECS: Definitions

- ECS is an architectural pattern
  - Popular in game engines
  - Favors composition over inheritance
- An entity is a general purpose object
- An entity gets its behavior by combining data
- Data comes from typed components

Entity Component System

- The Entity/Component data split gives flexibility to manage the API
- The System separation moves the behavior away from data avoiding dependencies between Components



Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- **Hello Donut**
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Hello Donut (QML)

- Good practice having root `Entity` to represent the scene
- One `Entity` per "object" in the scene
- Objects given behavior by attaching component subclasses
- For an `Entity` to be drawn it needs:
  - A mesh geometry describing its shape
  - A material describing its surface appearance

Demo qt3d/ex-hellodonut-qml

C++ API vs QML API

- QML API is a mirror of the C++ API
- C++ class names like the rest of Qt
- QML element names just don't have the Q in front
  - `Qt3DCore::QNode` vs `Node`
  - `Qt3DCore::QEntity` vs `Entity`
  - ...

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- **Input Handling**
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Previously in Input Handling

- Physical devices such as `KeyboardDevice` and `MouseDevice` produce events
- Handlers such as `KeyboardHandler` and `MouseHandler`:
  - Process events by converting events to signals for user code to react to
  - Are components that should be added to `Entity`s to provide behavior related to input
- `ObjectPicker` provides high-level picking functionality
- `LogicalDevice`s:
  - Allow analog axis values to be produced
  - Allow mapping multiple physical devices onto `Axis` and `Action` elements

How to Control a Value over Time?

- Obviously using an `Axis`
- But we got only the axis position...
- Force us to use imperative code executed in the main thread
  - Typically increment a value based on the axis position
  - Needs to sample over time (and eventually integrate!)
- Or use `AxisAccumulator` which does it for you
  - Manage the value over time based on an input axis
  - Can treat the axis position as a velocity or an acceleration
  - All the work is done in secondary threads

Axis Accumulator (since 5.8)

```
1  import Qt3D.Input 2.9
2  ...
3
4  LogicalDevice {
5      axes: Axis {
6          id: mouseYAxis
7          AnalogAxisInput {
8              sourceDevice: mouseDevice
9              axis: MouseDevice.Y
10         }
11     }
12 }
13
14 AxisAccumulator {
15     sourceAxis: mouseYAxis
16     sourceAxisType: AxisAccumulator.Velocity
17     scale: 50
18     // Can bind on value
19 }
```

Demo qt3d/sol-moving-boxes-qml-step3

Demo qt3d/sol-moving-boxes-qml-step4

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- **Animation with Qt 3D**
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Animation Support in Qt 3D

- You could use QtQuick animations but...
  - They are executed on the main thread
  - They are not synchronized with the Qt 3D engine frame rate
- Instead, you can have animations in the Qt 3D engine by registering the `Qt3DAnimation::QAnimationAspect`
- Like any other aspect it then provides API, mainly types inheriting from:
  - `AbstractAnimationClip` which contain the data representing a given animation
  - `AbstractClipAnimator`, `Component`s which run clips and map them to other components properties

AnimationClip, a Key Frame Based Clip

- `AnimationClip` represents a key frame based clip
- It holds the `AnimationClipData` in its clipData property
- Currently `AnimationClipData` instances can only be created from C++
- Clip data has a set of `QChannel` describing the properties know to the clip
- Each `QChannel` has one or more `QChannelComponent` allowing to represent complex types
  - Typically a color channel has three channel components
- A `QChannelComponent` is a list of key frames for the given channel component

AnimationClipLoader

- Creating a `AnimationClip` and its data can be tedious and hard to maintain
- Also it is not accessible to artists
- `AnimationClipLoader` can load a clip from a JSON file
  - The format is easy to export from a design tool
  - Currently a plugin for Blender is available

```
1 import Qt3D.Animation 2.9
2 ...
3
4 AnimationClipLoader { source: "qrc:/animation.json" }
5 ...
```

Demo qt3d/ex-animationclip-loader

How to Run an Animation Clip?

```
1  import Qt3D.Animation 2.9
2  ...
3  ClipAnimator {
4      clip: AnimationClipLoader { source: "qrc:/animation.json" }
5
6      channelMapper: ChannelMapper {
7          ChannelMapping {
8              channelName: "Location"
9              target: transform
10             property: "translation"
11         }
12         ChannelMapping {
13             channelName: "Rotation"
14             target: transform
15             property: "rotation"
16         }
17         ChannelMapping {
18             channelName: "Color"
19             target: material
20             property: "ambient"
21         }
22     }
23 }
24 ...
```

Demo qt3d/ex-animationclip-loader

Animation Blending

- It is often useful to combine several animations into one
- Makes it easier to tune simpler animations separately, then let the engine combine them
- This is done via blending operators
- Makes it possible to create new variations from a basic set of animations
- Typical examples in games are:
  - A character walking then starting to run
  - A character jumping while walking or during the transition between walking and running

BlendedClipAnimator

```
1  import Qt3D.Animation 2.9
2  ...
3  BlendedClipAnimator {
4      blendTree: AdditiveClipBlend {
5      additiveFactor: 0.4
6      baseClip: LerpClipBlend {
7          blendFactor: 0.2
8          startClip: ClipBlendValue {
9              clip: AnimationClipLoader { source: "qrc:/walk.json" }
10         }
11         endClip: ClipBlendValue {
12             clip: AnimationClipLoader { source: "qrc:/run.json" }
13         }
14     }
15     additiveClip: ClipBlendValue {
16         clip: AnimationClipLoader { source: "qrc:/jump.json" }
17     }
18 }
19 ...
```

Demo qt3d/sol-toyplane-pilot

Advances in Qt 3D

KDAB

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- **New PBR Materials**
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Metal/Rough Materials

- Qt 3D 5.9 introduces two new materials with much more realistic rendering
  - `Qt3DExtras::QMetalRoughMaterial`
  - `Qt3DExtras::QTexturedMetalRoughMaterial`
- This is based on proper physics to model the lighting
- It also introduces new richer lights

Qt Demo qt3d-examples/pbr-textured-cube

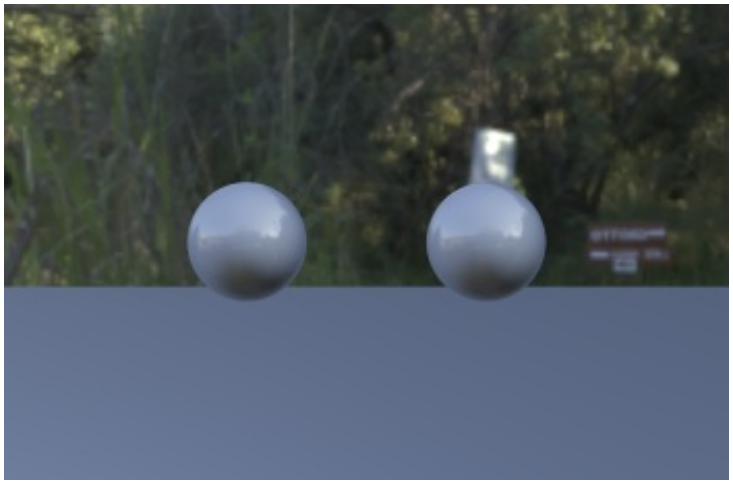
Qt Demo qt3d-examples/pbr-sphere

Qt Demo qt3d-examples/pbr-spheres

Environment Light (since 5.9)

```
1  import Qt3D.Core 2.0
2  import Qt3D.Render 2.9
3  ...
4
5  components: [
6      EnvironmentLight {
7          irradiance: TextureLoader { ... }
8          specular: TextureLoader { ... }
9      }
10 ]
```

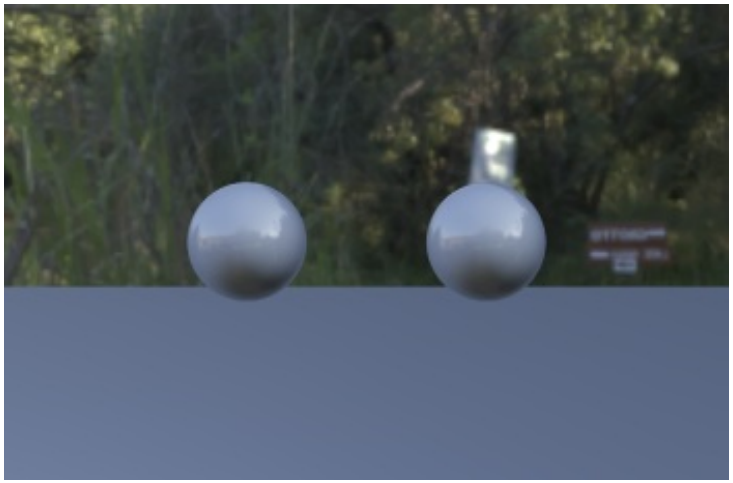
Demo qt3d/ex-lights-qml



Sky Box (since 5.9)

```
1 import Qt3D.Extras 2.9
2 ...
3
4 SkyboxEntity {
5     baseName: "radianceTexture"
6     extension: ".dds"
7     gammaCorrect: true // Since 5.9
8 }
```

Demo qt3d/ex-lights-qml



Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- **Painted Textures**
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

Integrating `QPainter` Code (since 5.8)

- Often, we have legacy `QPainter` code
- Needs integration to be usable with `Texture`s
- Provided by `Qt3DRender::QPaintedTextureImage`
  - Inherit from it
  - Override the `paint()` function
  - Use like any other `TextureImage`

Demo qt3d/ex-painted-cube

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- **Integrating Qt Quick with Qt 3D again**
- Capturing the Rendering
- Level of Detail
- Displaying Text
- The Future of Qt 3D

The Scene2D Element (since 5.9)

- Provided by the `QtQuick.Scene2D` module
- Takes an `Item` as child which will be your whole 2D scene
- It renders the 2D scene into a `RenderTargetOutput` controlled by the output property
  - Its texture can be used by any material
- The entities property allows to declare on which entities the texture will be used
  - Necessary for mouse event handling
  - Requires PickingSettings.TrianglePicking to be set to have the triangle information
- Mouse events are only accepted if the mouseEnabled property is true

Demo qt3d/ex-samegame

Advances in Qt 3D

KDAB

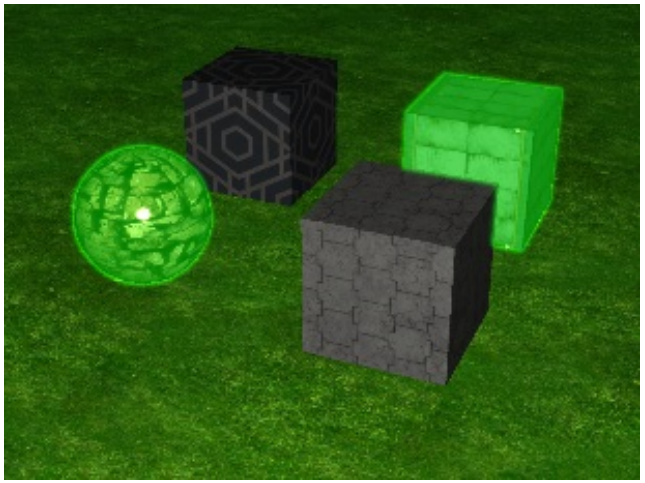
- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- **Capturing the Rendering**
- Level of Detail
- Displaying Text
- The Future of Qt 3D

The RenderCapture Element (since 5.9)

- Allows to create screenshots of the scene rendering
- Also allows to debug complex multi-pass rendering
  - One can save as an image one of the intermediate steps
- `RenderCapture` is a `FrameGraphNode`
- Each time a capture is needed, a call to `requestCapture()` is necessary
  - Such requests are processed asynchronously

Debugging Multi-Pass Rendering

- The scene allows to select objects by clicking on them
- A selected object glows
- The effect is implemented using a multi-pass render
- With `RenderCapture` it is easier to see what each stage is doing



Demo qt3d/sol-screenshot

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- **Level of Detail**
- Displaying Text
- The Future of Qt 3D

Complex Objects vs Distance

- Scenes often contain complex objects
- Such objects are expensive to display
- Does it still make sense if they are far from the camera?
- With level of detail management, simpler objects can be displayed instead
- This feature is provided with `LevelOfDetail` and `LevelOfDetailLoader`

The LevelOfDetail Element (since 5.9)

```
1  import Qt3D.Render 2.9
2  ...
3
4  SphereMesh {
5      slices: rings
6      rings: [30, 6, 4][lod.currentIndex]
7  },
8  LevelOfDetail {
9      id: lod
10     camera: mainCamera
11     thresholds: [100, 500, 1000]
12     thresholdType: LevelOfDetail.DistanceToCameraThreshold
13 }
14 ...
```

Demo qt3d/ex-lod

Demo qt3d/sol-ogrehead

Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- **Displaying Text**
- The Future of Qt 3D

Extruded Text Meshes (since 5.9)

- Generating geometry out of text is done with `ExtrudedTextGeometry` or `ExtrudedTextMesh`
- They can be used like any other `Geometry` or `GeometryRenderer`
- font and text are controlled using properties
- The length of the extrusion is controlled with the depth property

Demo qt3d/ex-text-3d

Distance Field Text (since 5.9)

- Distance field text is provided by `Text2DEntity`
- This is a full fledged `Entity` to put in the object tree
- font, color and text are controlled using properties
- The size of the surface on which the text is rendered can be controlled via width and height

Demo qt3d/ex-text-2d

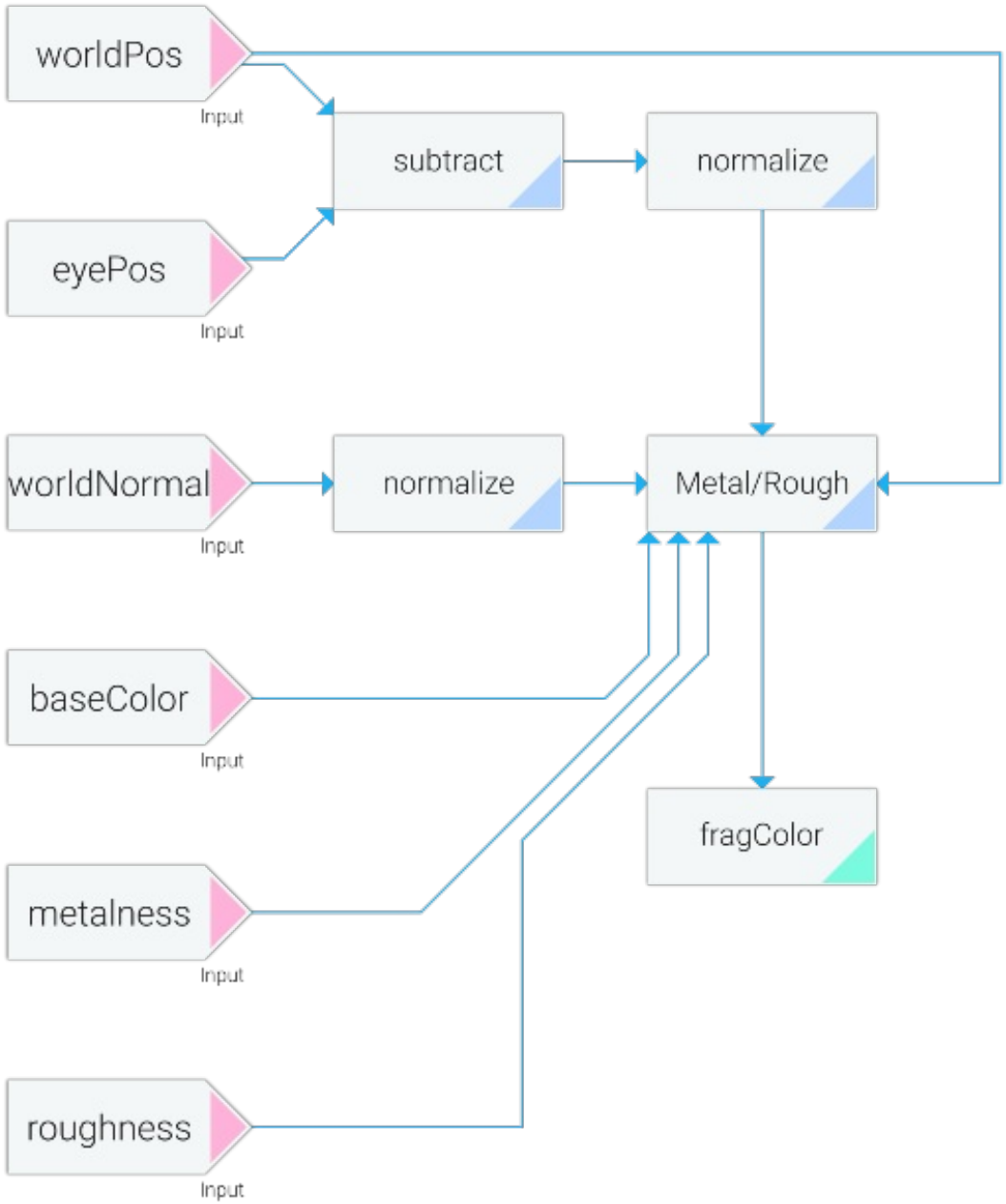
Advances in Qt 3D

- Feature Set
- Entity Component System? What's that?
- Hello Donut
- Input Handling
- Animation with Qt 3D
- New PBR Materials
- Painted Textures
- Integrating Qt Quick with Qt 3D again
- Capturing the Rendering
- Level of Detail
- Displaying Text
- **The Future of Qt 3D**

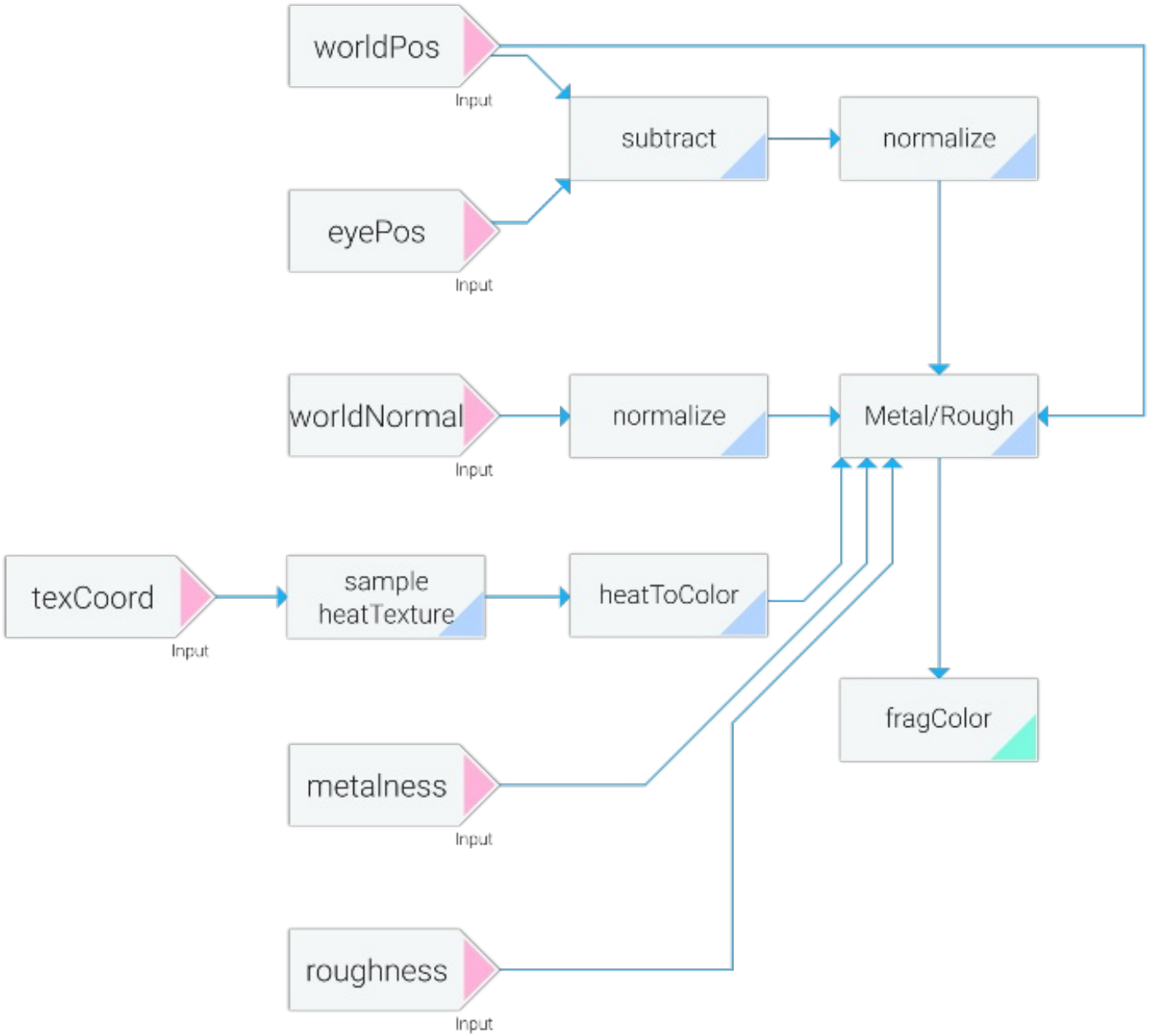
Shader Graphs (coming in 5.10)

- Currently difficult to reuse and tune materials
- Forced to fork the shader implementations...
- Introducing `ShaderProgramBuilder` which allows to load shaders from a graph
- JSON format
- Finer grained building blocks which can be reused and reorganized

Shader Graphs: QMetalRoughMaterial



Shader Graphs cont'd



What does the future hold for Qt 3D?

- Qt 3D Core
  - Efficiency improvemments
  - Backend threadpool and job handling improvements - jobs spawning jobs
- Qt 3D Render
  - Billboards - camera facing entities
  - Particle systems
- Qt 3D Input
  - Additional input device support
    - 3D mouse controllers, game controllers
  - Enumerated inputs such as 8-way buttons, hat switches or dials

What does the future hold for Qt 3D?

KDAB

- New aspects:
  - Collision Detection Aspect
    - Allows to detect when entities collide or enter/exit volumes in space
  - Animation Aspect
    - Skeletal animation
    - Morph target animation
    - Removes animation workload from main thread
  - Physics Aspect
    - Rigid body and soft body physics simulation
  - AI Aspect, 3D Positional Audio Aspect ...
- Tooling:
  - Design time tooling - scene editor
  - Build time tooling - asset conditioners for meshes, textures etc.

Thank you!

www.kdab.com

kevin.ottens@kdab.com