Qt 3D : Point d'Étape

Kévin Ottens, Artisan Logiciel à KDAB

Capitole du Libre 2017

KDAB

Qt 3D : Point d'étape

Fonctionnalités

Qu'est-ce que Qt 3D?

- Ce n'est pas qu'une moteur 3D!
- Généraliste, pas uniquement pour les jeux
- Moteur de simulation temps-réel souple
- Conçu pour le passage à l'échelle
- Extensible et flexible

Moteur de simulation

- Le cœur n'est pas spécifique à la 3D
- Il peut gérer plusieurs domaines simultanément
  - IA, logique, audio, etc.
  - Et bien sûr il fait aussi le rendu 3D !
- Tout le nécessaire pour des systèmes de simulation complexes
  - Systèmes mécaniques
  - Physique
  - ... et aussi les jeux

Passage à l'échelle

- Séparation frontend / backend
  - Frontend léger sur la thread principale
  - Backend exécuté dans une thread secondaire
    - Là où réside la simulation
- Communication frontend / backend non-bloquante
- Le backend maximise la charge via une pool de threads

Extensible et flexible

- Les domaines sont ajoutés via des aspects indépendants
  - ... uniquement si rien ne correspond à vos besoins
- Fourni à la fois des APIs C++ et QML
- S'intègre bien avec le reste de Qt
  - Qui veut obtenir ses données de simulation d'une base de données?
- Système Entité Composant (ECS) utilisé pour combiner les comportements dans vos objets
  - Pas de hiérarchie d'héritage profonde

KDAB

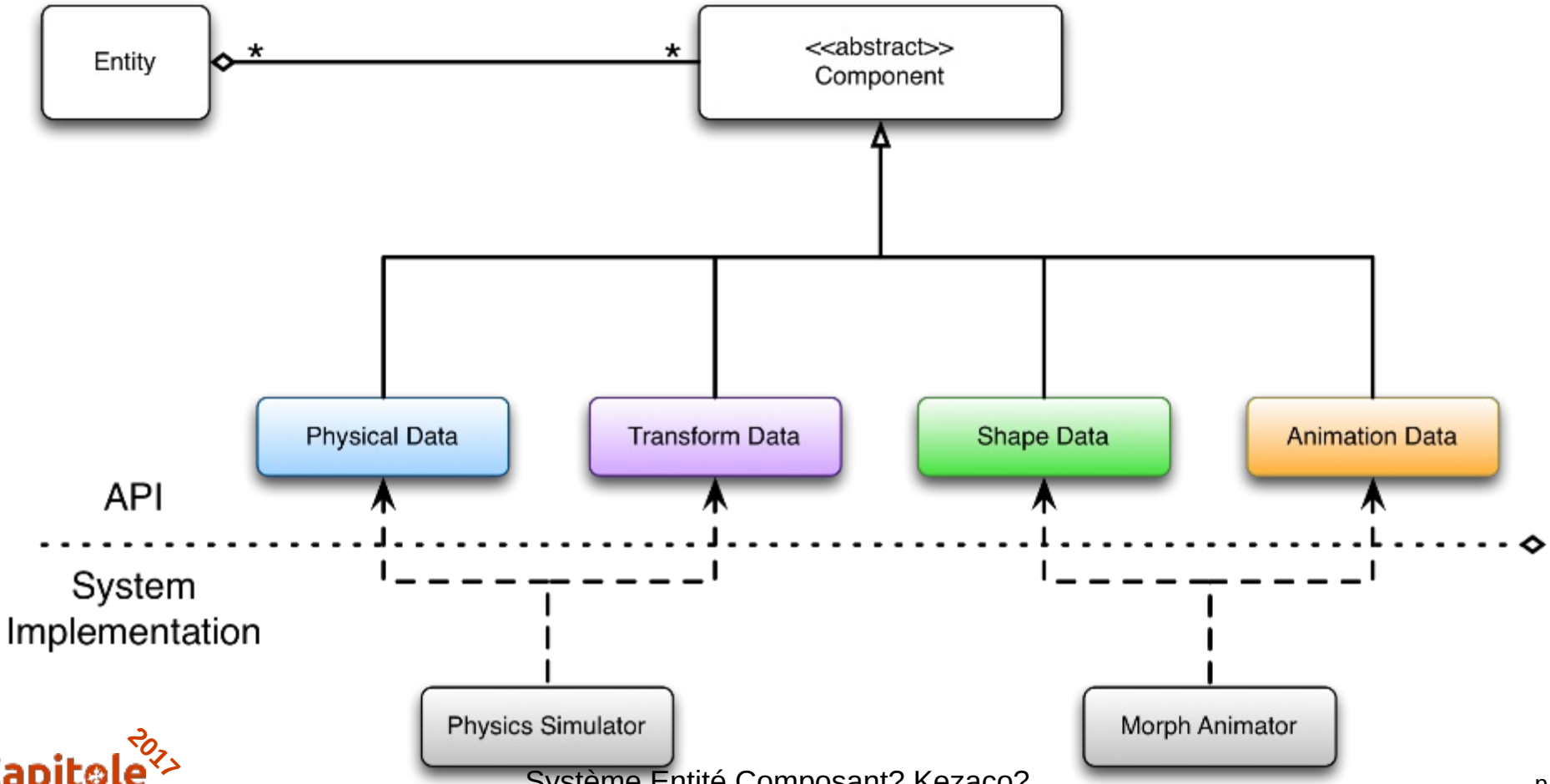
Système Entité Composant? Kezaco?

ECS: Définitions

- ECS est un patron d'architecture
  - Populaire dans les moteurs de jeux
  - Favorise la composition plutôt que l'héritage
- Une entité est un objet généraliste
- Une entité obtient son comportement en combinant de la donnée
- La donnée provient de composants typés

Système Entité Composant

- La séparation des données Entité/Composant donne de la flexibilité pour gérer l'API
- La séparation du système éloigne le comportement de la donnée, évitant les dépendances entre composants



Hello Donut

Hello Donut (QML)

- Avoir une `Entity` racine pour représenter la scène est une bonne idée
- Une `Entity` par "objet" dans la scène
- Le comportement des objets est obtenu en attachant des composants
- Pour afficher une `Entity` il faut :
  - Une géométrie pour décrire sa forme
  - Un matériau pour décrire l'apparence de la surface

Demo qt3d/ex-hellodonut-qml

API C++ vs API QML

- L'API QML est un miroir de l'API C++
- Les classes C++ sont nommées comme le reste de Qt
- Le nom des éléments QML perdent le préfixe Q
  - `Qt3DCore::QNode` vs `Node`
  - `Qt3DCore::QEntity` vs `Entity`
  - ...

Gestion des entrées

Précédemment dans la gestion des entrées

- Les périphériques physiques tels que `KeyboardDevice` et `MouseDevice` produisent des événements
- Les gestionnaires d'entrées tels que `KeyboardHandler` et `MouseHandler`:
  - Traitent les événements et les convertissent en signaux exploitables par le code utilisateur
  - Sont des composants à ajouter aux `Entity`s pour fournir un comportement lié aux entrées
- `ObjectPicker` fourni la fonctionnalité de pointage de haut niveau
- `LogicalDevice`s:
  - Permettent de produire des axes analogiques
  - Permettent de liés plusieurs périphériques physiques via les éléments `Axis` et `Action`

Comment contrôler une valeur dans le temps?

- Clairement en utilisant un `Axis`
- Mais on ne dispose que de la position de l'axe...
- Cela nous force à utiliser du code impératif exécuté dans la thread principale
  - Typiquement on incrémente une valeur basée sur la position de l'axe
  - Nécessite d'échantilloner dans le temps (et éventuellement d'intégrer!)
- Ou on utilise `AxisAccumulator` qui fait le travail pour vous
  - Contrôle la valeur dans le temps basé sur un axe d'entrée
  - Peut traiter la position de l'axe comme vélocité ou accélération
  - Tout le travail est effectué dans des threads secondaires

Axis Accumulator (since 5.8)

```
 1 import Qt3D.Input 2.1
 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

KDAB

Animations avec Qt 3D

Support des animations dans Qt 3D

- Vous pourriez utiliser les animations QtQuick mai...
  - Elles sont exécutées sur la thread principale
  - Elles ne sont pas synchronisées avec le frame rate de Qt 3D
- A la place, vous pouvez activer le support des animations dans le moteur en enregistrant `Qt3DAnimation::QAnimationAspect`
- Comme n'importe quel autre aspect une API est fournie, principalement des types héritant de:
  - `AbstractAnimationClip` contenant les données représentant une animation
  - `AbstractClipAnimator`, des `Component`s qui jouent des clips et les associent aux propriétés d'autres composants

AnimationClip, un clip basé sur des key frames

- `AnimationClip` représente un clip de key frames
- Il contient les données sous la forme d'un `AnimationClipData` lié à sa propriété clipData
- Pour le moment les instances de `AnimationClipData` ne peuvent être créées que depuis le code C++
- La donnée d'un clip est un ensemble de `QChannel` décrivant les propriétés connues par le clip
- Chaque `QChannel` a un ou plus `QChannelComponent` permettant de représenter des types complexes
  - Par exemple un canal couleur a trois composants
- Un `QChannelComponent` est une liste de key fames pour le composant donné

AnimationClipLoader

- Créer un `AnimationClip` et ses données peut être fastidieux et difficile à maintenir
- De plus ce n'est pas faisable par un artiste
- `AnimationClipLoader` peut charger un clip depuis un fichier JSON
  - Le format est facile à exporter depuis un outil de conception
  - Un plugin pour Blender est disponible

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

Demo qt3d/ex-animationclip-loader

Comment exécuter un 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

Mélanger des animations

- Il est souvent utile de pouvoir combiner plusieurs animations en une seule
- Il est plus facile de mettre au point des animations simples séparément puis de laisser le moteur les combiner
- Implémenté par des opérateurs de mélange
- Rend possible de créer de nouvelles variations depuis un ensemble d'animations simples
- Exemples typiques dans les jeux:
  - Un personnage marche puis commence à courir
  - Un personnage saute pendant qu'il marche ou pendant la transition entre la marche et la course

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

Nouveaux matériaux PBR

Matériaux metal/rough

- Qt 3D 5.9 introduit deux nouveaux matériaux avec un rendu plus réaliste
  - `Qt3DExtras::QMetalRoughMaterial`
  - `Qt3DExtras::QTexturedMetalRoughMaterial`
- Ils sont basés sur un vrai modèle physique de la lumière
- Ils permettent aussi d'introduire de nouvelles sources lumineuses plus riches

Qt Demo qt3d-examples/pbr-textured-cube

Qt Demo qt3d-examples/pbr-sphere

Qt Demo qt3d-examples/pbr-spheres

Environment Light (depuis 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 (depuis 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

KDAB

Peindre des textures

Intégrer le code utilisant `QPainter` (depuis 5.8)

- Souvent, nous avons de l'ancien code utilisant `QPainter`
- Il est nécessaire de pouvoir l'utiliser avec des `Texture`s
- C'est possible avec `Qt3DRender::QPaintedTextureImage`
  - Hériter de la classe
  - Réimplémenter la fonction `paint()`
  - L'utiliser comme n'importe quel autre `TextureImage`

Demo qt3d/ex-painted-cube

KDAB

Intégrer encore mieux Qt Quick et Qt 3D

L'élément Scene2D (depuis 5.9)

- Fourni par le module `QtQuick.Scene2D`
- Prend un `Item` en tant qu'enfant qui sera toute la scène 2D
- Il effectue le rendu de la scène 2D dans un `RenderTargetOutput` contrôlé par la propriété output
  - Cette texture peut-être utilisée par n'importe quel matériau
- La propriété entities permet de déclarer sur quelles entités la texture sera utilisée
  - Nécessaire pour la prise en charge de la souris
  - Requiert de positionner PickingSettings.TrianglePicking pour avoir les informations de triangulation
- Les événements souris sont acceptés uniquement si la propriété mouseEnabled est vraie

Demo qt3d/ex-samegame

KDAB

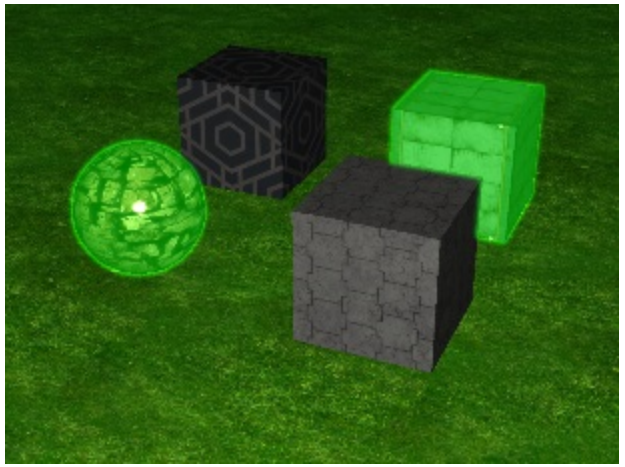
Capturer le rendu

L'élément RenderCapture (depuis 5.9)

- Permet d'obtenir des captures du rendu de la scène
- Permet aussi de déverminer les rendus à passes multiples
  - On peut sauver dans une image les étapes intermédiaires
- `RenderCapture` est un `FrameGraphNode`
- Chaque fois qu'une capture est nécessaire, il suffit d'appeler `requestCapture()`
  - Ces requêtes sont traitées de manière asynchrone

Déverminer le rendu à passes multiples

- La scène permet de sélectionner des objets en cliquant dessus
- Un objet sélectionné devient luisant
- L'effet est implémenté en utilisant un rendu à passes multiples
- Avec `RenderCapture` il est plus facile de voir ce que chaque étape réalise



Demo qt3d/sol-screenshot

Niveau de détails

Objets complexes vs distance

- Les scènes contiennent souvent des objets complexes
- De tels objets sont coûteux à afficher
- Est-ce que cela a toujours du sens s'ils sont loin de la caméra ?
- Avec la gestion du niveau de détails, des objets plus simples peuvent être affichés à la place
- Cette fonctionnalité est fournie par `LevelOfDetail` et `LevelOfDetailLoader`

L'élément LevelOfDetail (depuis 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-levelofdetail

KDAB

Afficher du texte

Géométrie de texte extrudé (depuis 5.9)

- Générer une géométrie à partir d'un texte est fait avec `ExtrudedTextGeometry` ou `ExtrudedTextMesh`
- Ils peuvent être utilisés comme n'importe quel autre `Geometry` ou `GeometryRenderer`
- font et text sont contrôlé par des propriétés
- La longueur de l'extrusion est contrôlée avec la propriété depth

Demo qt3d/ex-text-3d

Texte par carte de distance (depuis 5.9)

- Le texte par carte de distance est fourni par `Text2DEntity`
- Il s'agit d'une `Entity` complète à insérer dans l'arbre d'objets
- font, color et text sont contrôlés par des propriétés
- La taille de la surface de rendu du texte peut être contrôlé via width et height

Demo qt3d/ex-text-2d

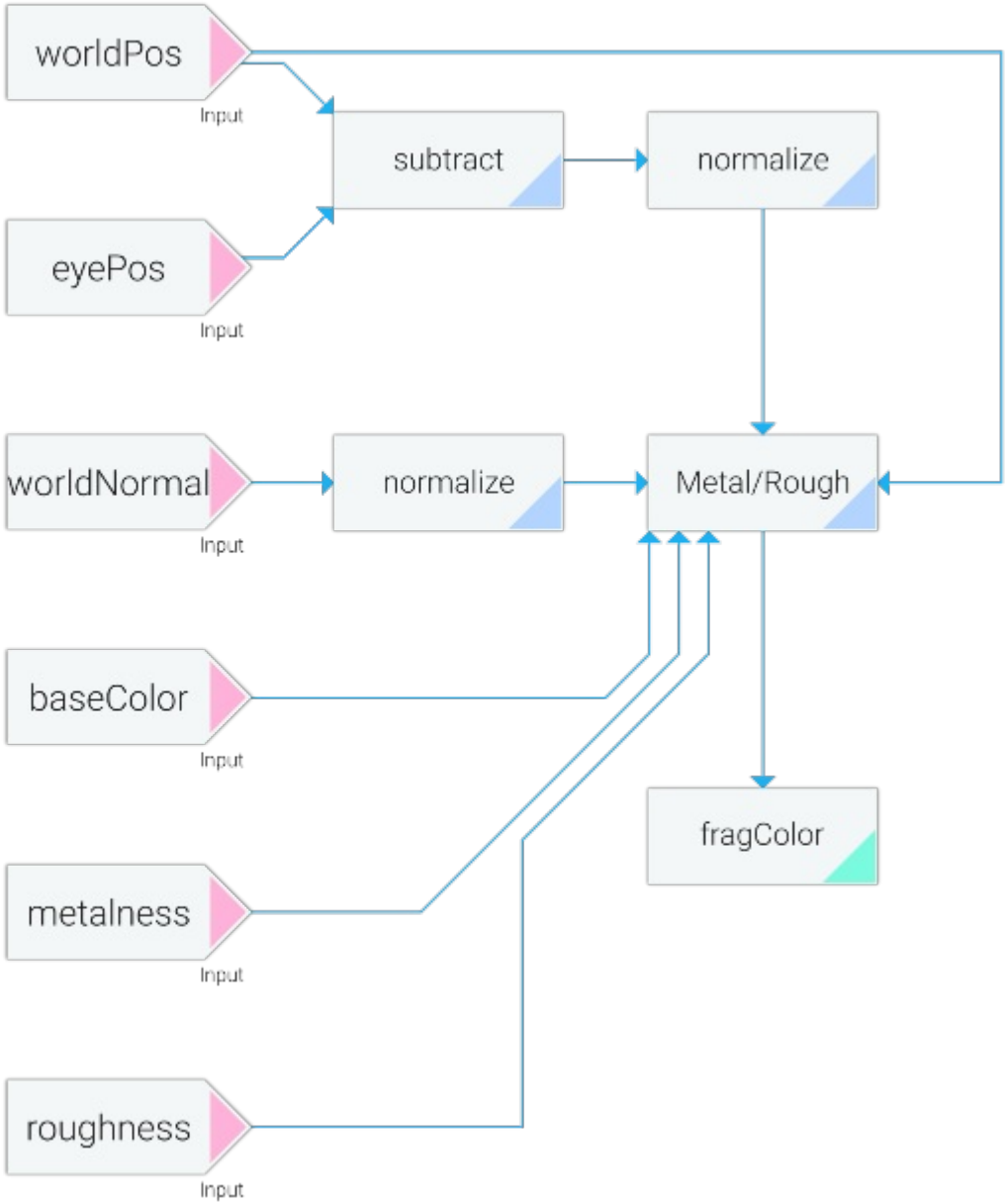
KDAB

Le futur de Qt 3D

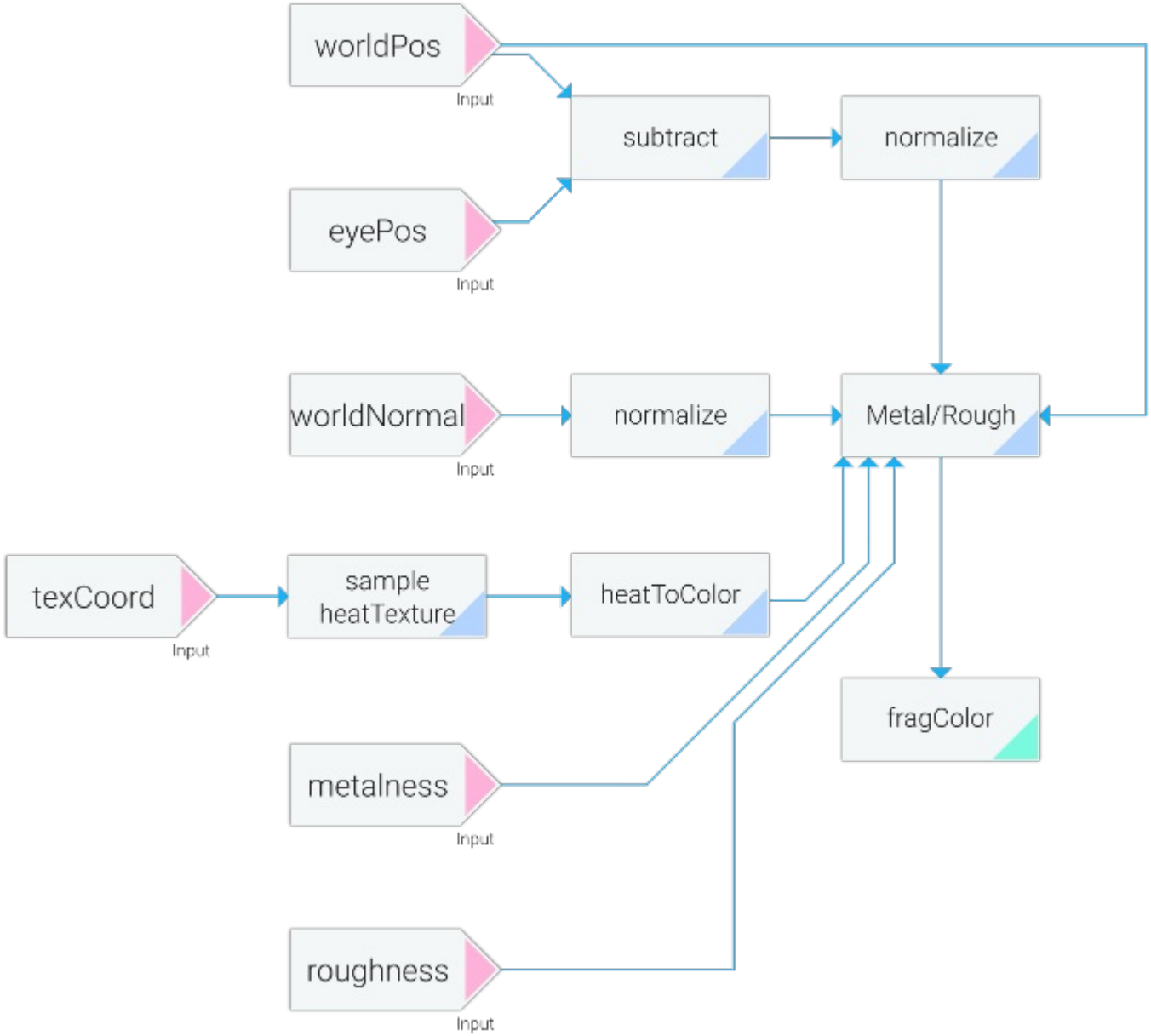
Graphes de Shader (introduits dans 5.10)

- Pour l'instant il est difficile de réutiliser ou de modifier des matériaux existants
- Forcé de dupliquer les implémentations de shaders
- Le nouvel élément `ShaderProgramBuilder` permet de charger des shaders depuis un graphe
- Format JSON
- Blocs plus petits permettant de réutiliser et de réorganiser le code

Graphes de Shader: QMetalRoughMaterial



Graphes de Shader



Qu'est-ce qui arrivera dans le futur de Qt 3D ?

KDAB

- Qt 3D Core
  - Amélioration des performances
  - Amélioration de la gestion du pool de threads sur le backends - jobs qui créent des jobs
- Qt 3D Render
  - Billboards - objets faisant toujours face à la caméra
  - Systèmes de particules
- Qt 3D Input
  - Support de périphériques d'entrée additionnels
    - Souris 3D, manettes de jeux
  - Entrées énumérées comme les boutons 8 positions, etc.

Qu'est-ce qui arrivera dans le futur de Qt 3D ? (cont'd)

- Nouveaux aspects:
  - Détection de collisions
    - Permet de détecter quand des entités sont en collision ou entrent/ sortent de volumes dans l'espace
  - Animation
    - Animation de squelettes
    - Morphing
  - Physique
    - Simulation des corps rigides et souples
  - IA, Audio spatial...
- Outillage:
  - Editeur de scène
  - Optimisation d'assets pour les géométries, textures, etc.

Merci!

www.kdab.com

kevin.ottens@kdab.com