Les bases de Qt 3D

Kévin Ottens, Artisan Logiciel à KDAB

Capitole du Libre 2016

Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

Les bases de Qt 3D

- **Fonctionnalités**
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

Qu'est-ce que Qt 3D?

- Ce n'est pas qu'un 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

Les bases de Qt 3D

- Fonctionnalités
- **Système Entité Composant? Kezaco?**
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

ECS: Définitions

KDAB

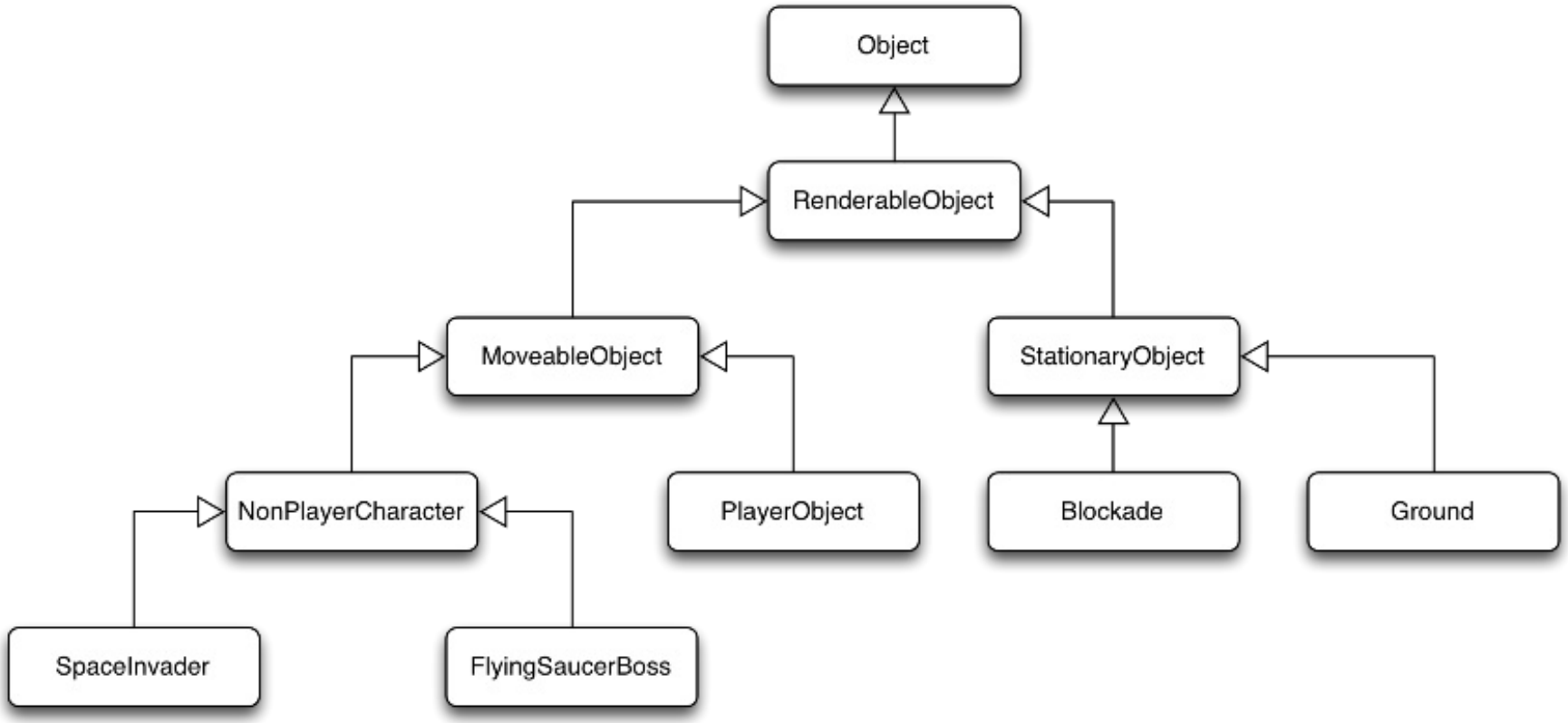
- ECS est un patron d'architecture
  - Populaire dans les moteurs de jeux
  - Favorise la composition plutôt que l'héritage
- Une entité est un objet généraliste
- Une entité obtient son comportement en combinant de la donnée
- La donnée provient de composants typés

Composition vs Héritage

- Analysons un exemple familier : Space Invaders

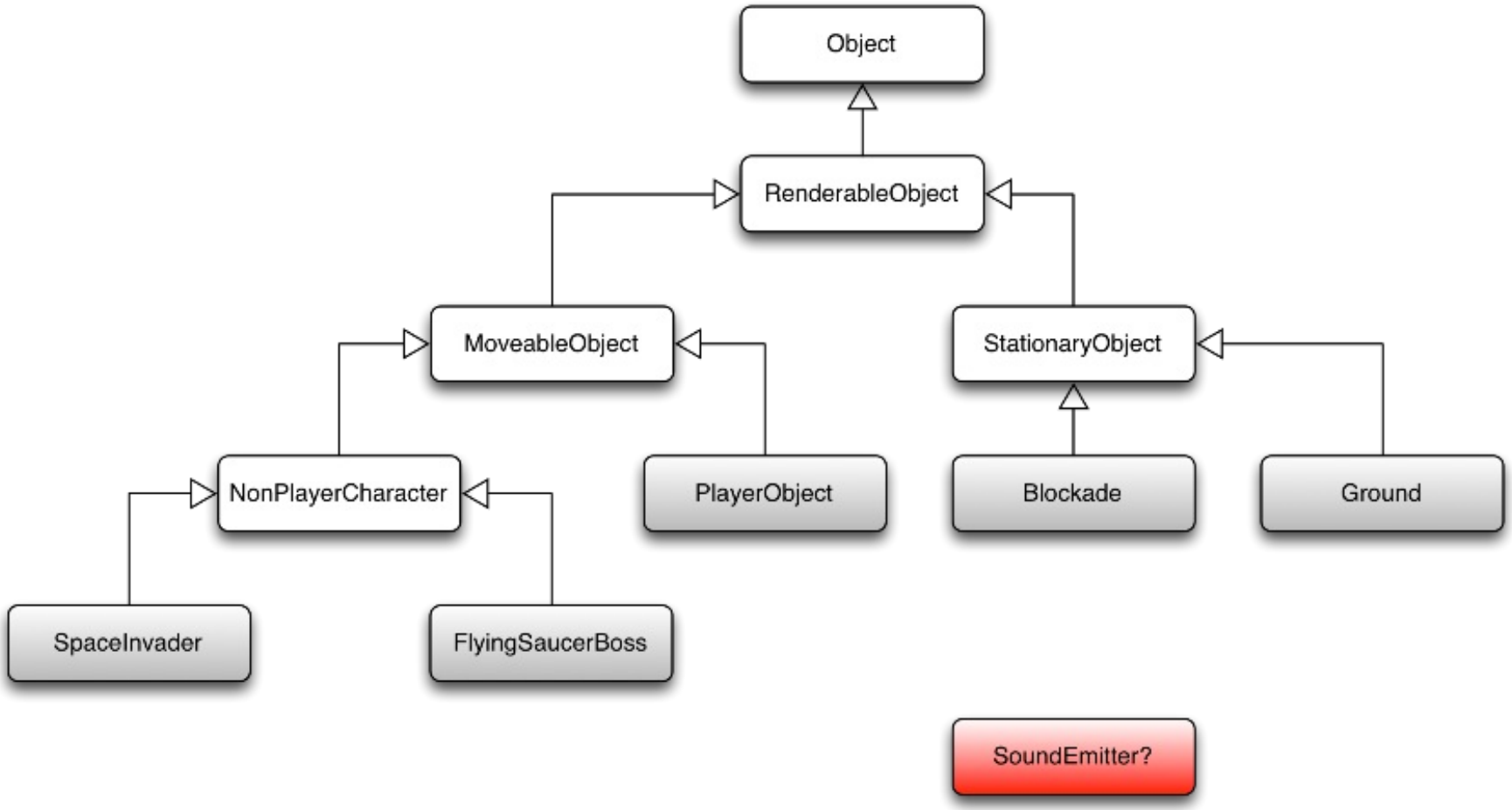
Composition vs Héritage (suite)

- Hiérarchie d'héritage typique



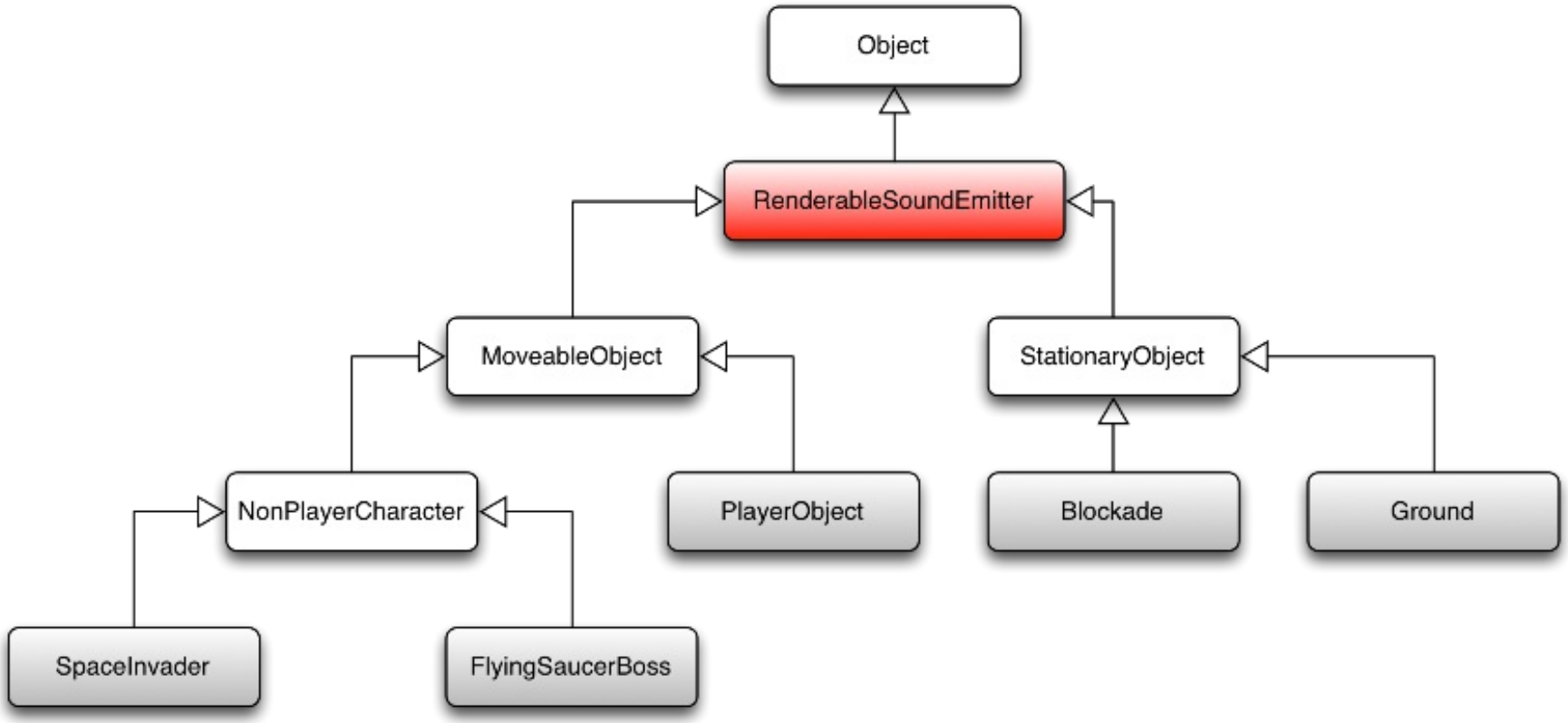
Composition vs Héritage (suite)

- Tout va bien jusqu'à ce que le client demande une nouvelle fonctionnalité :



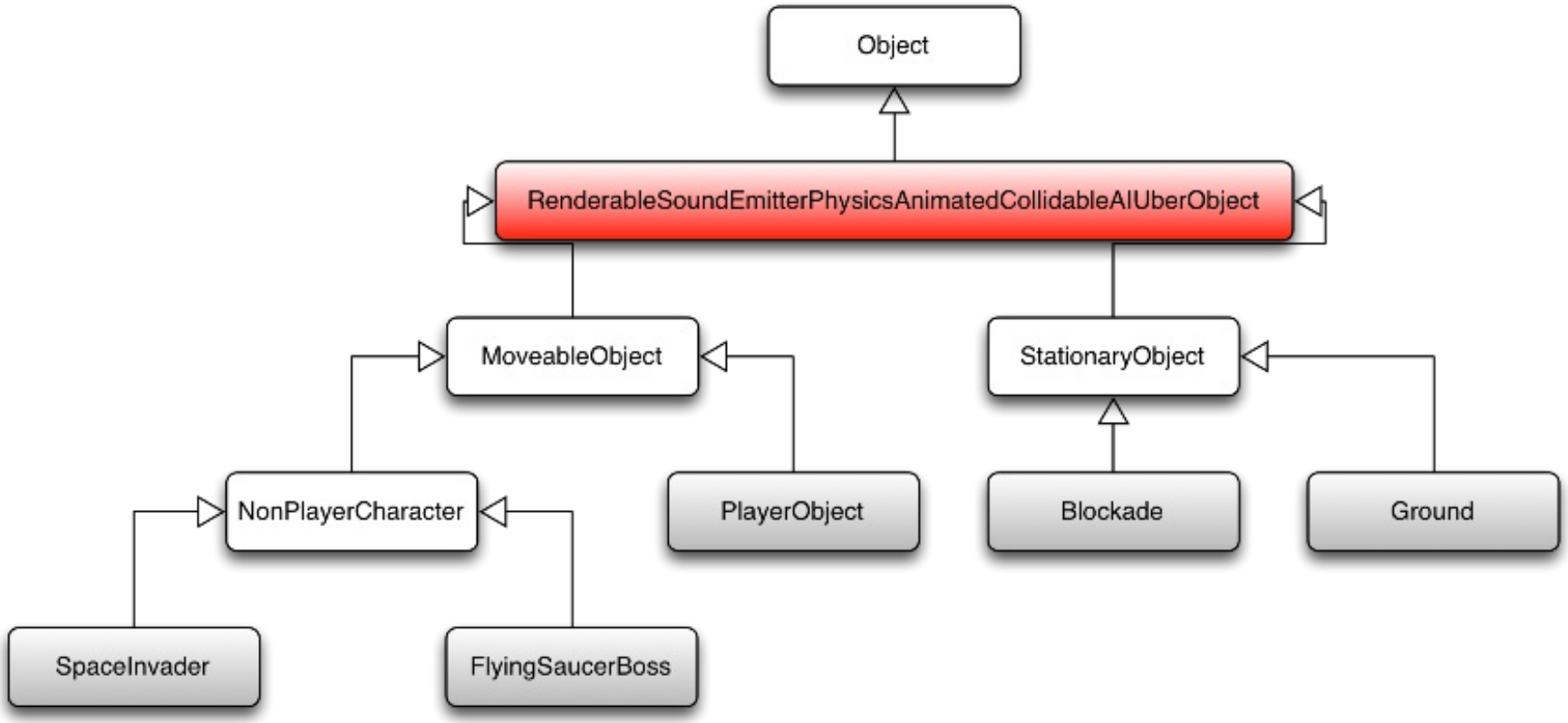
Composition vs Héritage (suite)

- Solution typique : ajouter à la classe de base



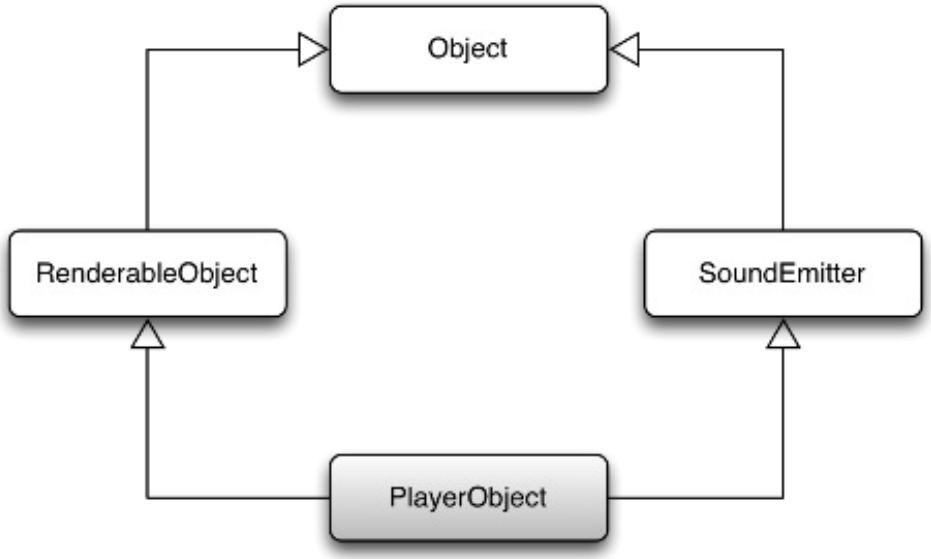
Composition vs Héritage (suite)

- Ne passe pas à l'échelle :



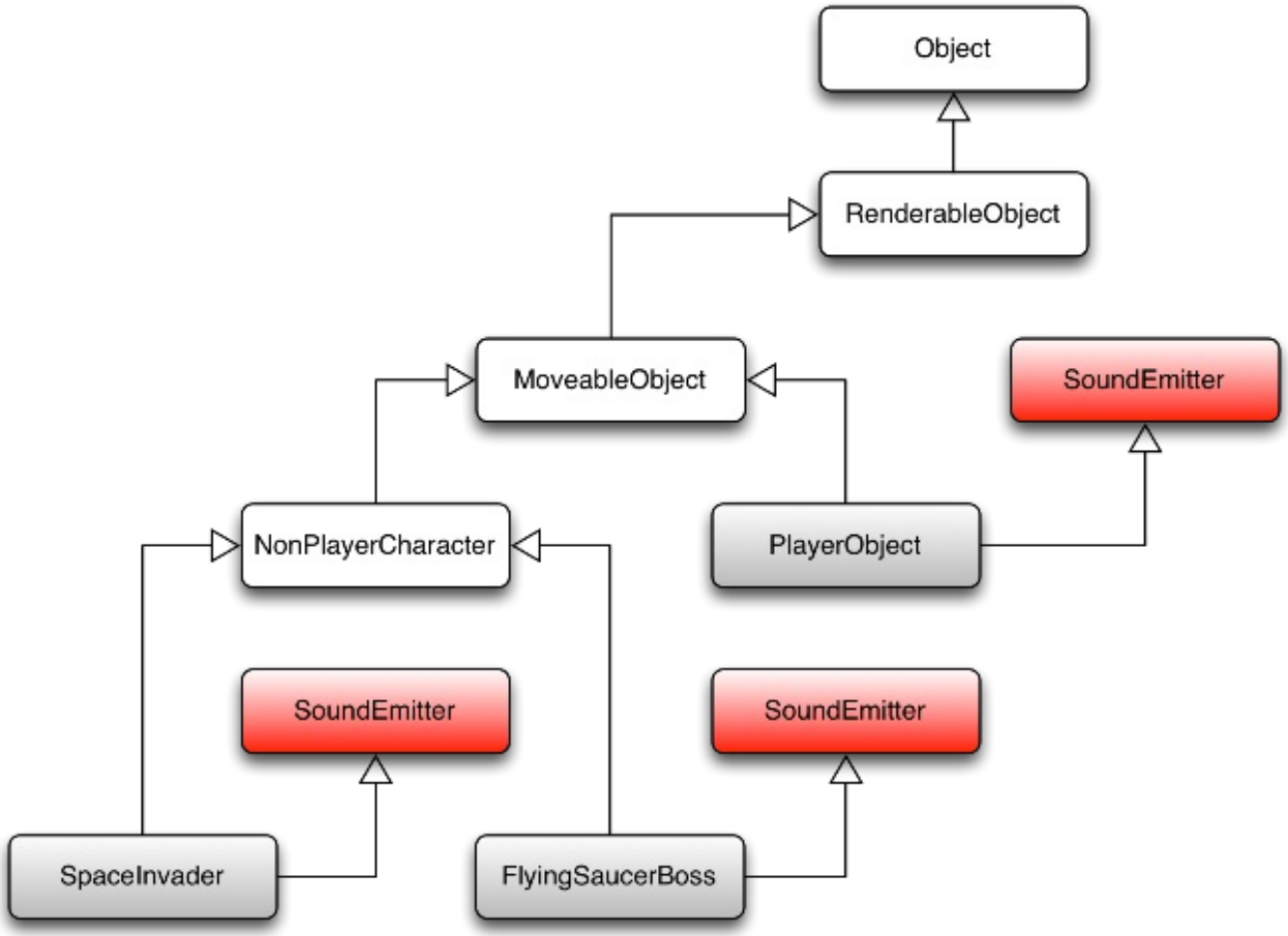
Composition vs Héritage (suite)

- Et l'héritage mutiple ?



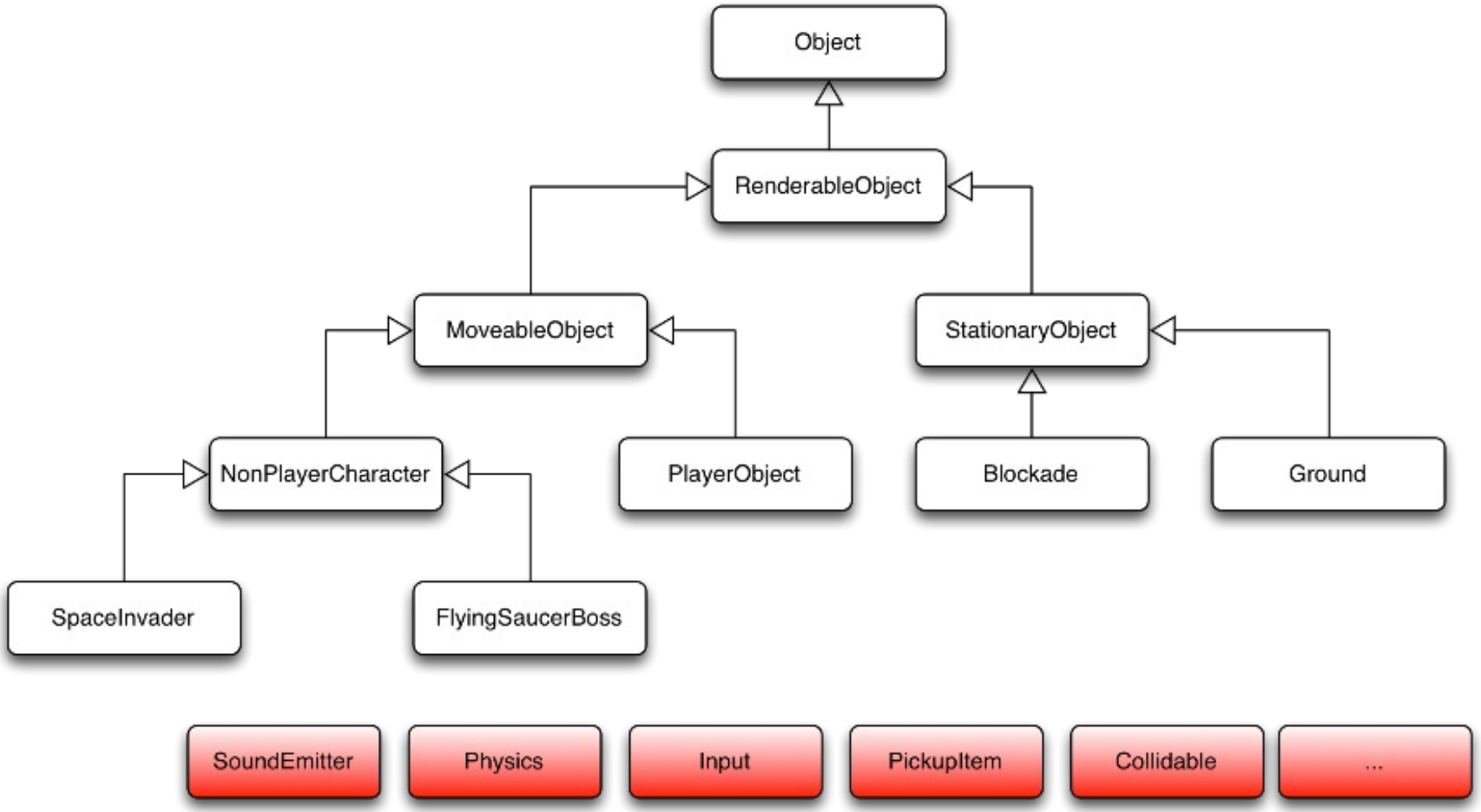
Composition vs Héritage (suite)

- Et l'utilisation d'interfaces ?



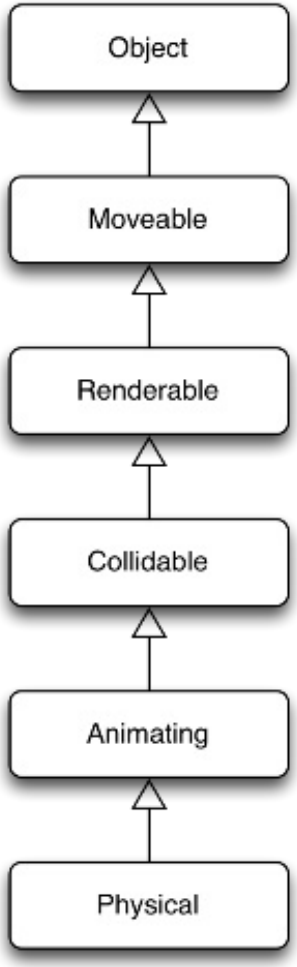
Composition vs Héritage (suite)

- Est-ce que cela passe à l'échelle ?



Composition vs Héritage (suite)

- L'héritage est-il suffisamment flexible ?



Composition vs Héritage (suite)

KDAB

- Héritage :
  - Relation figées à la conception
  - Hiérarchies d'héritage complexes : profondes, larges, héritages multiples
  - Les fonctionnalités tendent à migrer vers la classe de base
- Système Entité Composant
  - Autorise les modifications à l'exécution
  - Évite les limitations de l'héritage
  - A un coût supplémentaire :
    - Plus de QObjects
    - Différent des habitudes de la plupart des développeurs objet
  - Pas besoin de figer des suppositions sur Qt 3D qui ne pourront pas être changées plus tard lors de l'ajout de nouvelles fonctionnalités

Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- **Hello Donut**
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

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

KDAB

- 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`
  - ...

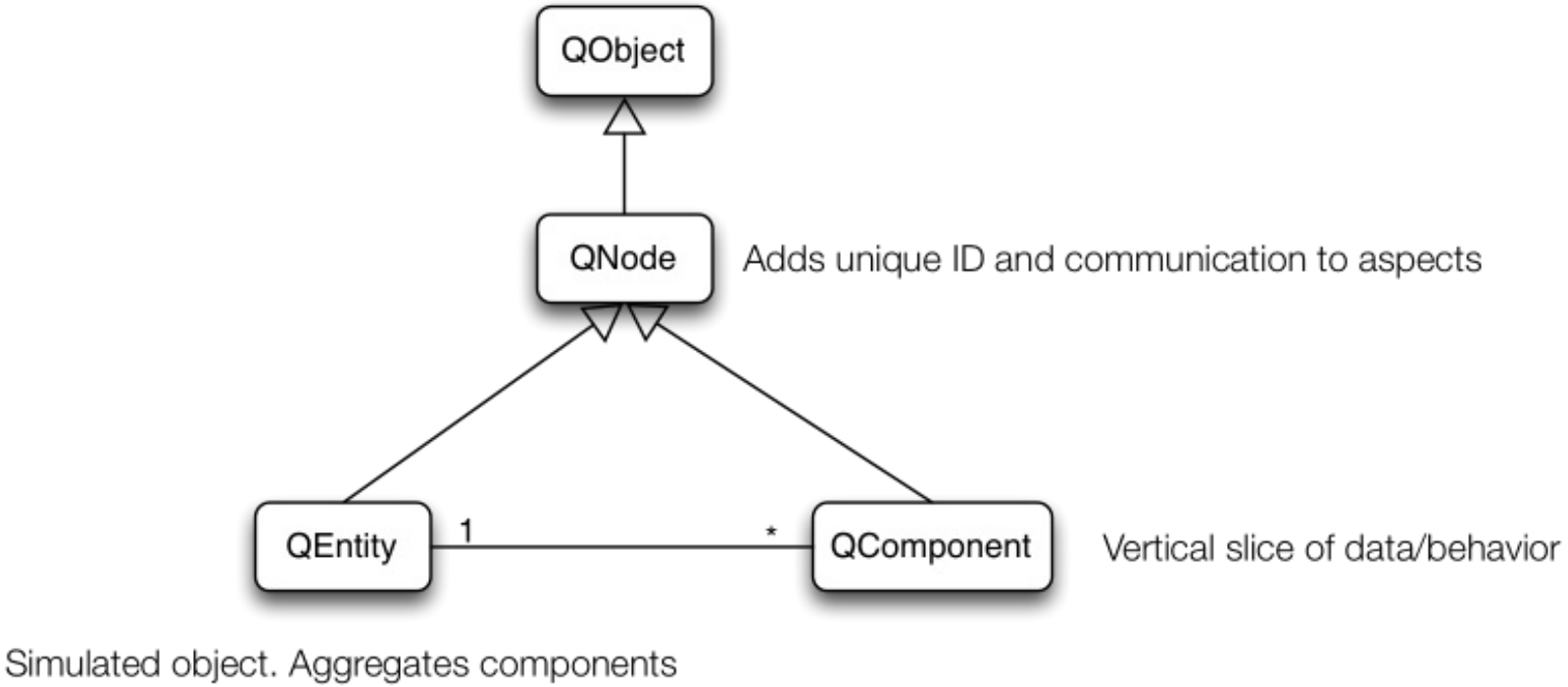
Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- **L'ECS de Qt 3D expliqué**
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

Tout est QNode

- `Qt3DCore::QNode` le type de base utilisé partout
  - Il hérite de `QObject` et de toutes ses fonctionnalités
  - Ajoute la communication frontent/backend en interne
- `Qt3DCore::QEntity`
  - Il hérite de `Qt3DCore::QNode`
  - Il aggrège simplement des `Qt3DCore::QComponents`
- `Qt3DCore::QComponent`
  - Il hérite de `Qt3DCore::QNode`
  - Les données sont fournies par ses classes filles
    - `Qt3DCore::QTransform`
    - `Qt3DRender::QMesh`
    - `Qt3DRender::QMaterial`
    - ...

Tout est QNode (suite)



Il vous faut encore un système

- La simulation est exécutée par `Qt3DCore::QAspectEngine`
- Les instances de sous-classes de `Qt3DCore::QAbstractAspect` sont enregistrées sur le moteur
  - Le comportement provient des aspects traitant les données des composants
  - Les aspects contrôlent les domaine manipulés par votre simulation
- Qt 3D fourni
  - `Qt3DRender::QRenderAspect`
  - `Qt3DInput::QInputAspect`
  - `Qt3DLogic::QLogicAspect`
- Notez que les aspects n'ont pas d'API
  - Tout est fourni par des sous-classes de `Qt3DCore::QComponent`

Les bases de Qt 3D

KDAB

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- **Gestion des entrées**
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- Le futur de Qt 3D

Périphériques physiques

- Pour gérer les entrées il nous faut d'abord générer les événements d'entrée
- Les sous-classes de `Qt3DInput::QAbstractPhysicalDevice` représentent les périphériques d'entrée
  - `Qt3DInput::QKeyboardDevice`
  - `Qt3DInput::QMouseDevice`
  - D'autres peuvent être ajoutés plus tard
- Par lui même un périphérique ne fait pas grand chose
  - Les gestionnaires d'entrée exposent des signaux émis en réponse aux événements

Pointage

- Le pointage de haut-niveau est fourni par le composant `Qt3DRender::QObjectPicker`
  - Implicitement associé à la souris
  - Utilise un pointage basé sur du lancer de rayons
- `Qt3DRender::QObjectPicker` émets des signaux :
  - `pressed(), released(), clicked()`
  - `moved()` - uniquement quand dragEnabled est vrai
  - `entered(), exited()` - uniquement quand hoverEnabled est vrai
- La propriété containsMouse fourni une alternative déclarative pour `entered(), exited()`

Périphériques physiques vs périphériques logiques

- Les périphériques physiques fournissent uniquement des événements discrétisés
- Il est difficile de les utiliser pour contrôler une valeur dans le temps
- Les périphériques logiques permettent:
  - D'obtenir une vue analogique sur un périphérique physique
  - D'aggréger plusieurs périphériques physiques comme un périphérique unifié

Action d'entrée logique

- `Qt3DInput::QAction` expose une valeur binaire
- Elle est activée par une entrée qui peut être :
  - Un simple bouton avec `Qt3DInput::QActionInput`
  - Une combinaison simultanée de boutons avec `Qt3DInput::QInputChord`
  - Une séquence de boutons avec `Qt3DInput::QInputSequence`
- Lorsque l'état de l'action change, la propriété active change d'état

Demo qt3d/ex-logical-input-qml

Axe d'entrée logique

- `Qt3DInput::QAxis` expose une valeur analogique entre -1 et 1
- Elle varie dans le temps quand une entrée est générée :
  - Lorsque un axe physique change avec `Qt3DInput::QAnalogAxisInput`
  - Pendant qu'un bouton est pressé avec `Qt3DInput::QButtonAxisInput`
- Lorsque l'état de l'axe change, la propriété value change

Demo qt3d/ex-logical-axes-qml

Tout assembler pour déplacer des boîtes

- Le focus est géré avec la touche tabulation
- La boîte ayant le focus est plus grande
- Les flèches déplacent la boîte sur le sol
- Page précédente/suivante fait tourner la boîte sur son axe Y
- Les boîtes s'éclairent lorsque la souris les survole
- Cliquer sur une boîte lui donne le focus
- Les boîtes peuvent être déplacées avec la souris

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

Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- **Les bases du tracé 3D**
- La partie immergée de l'iceberg
- Le futur de Qt 3D

Le graphe de scène

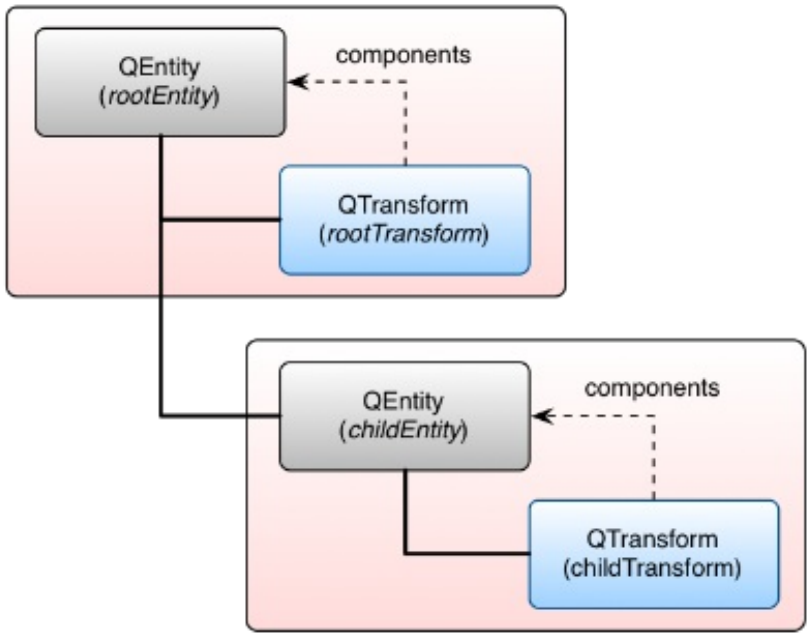
- Le graphe de scène fourni la représentation spatiale de la simulation
  - `Qt3DCore::QEntity` : ce qui participe à la simulation
  - `Qt3DCore::QTransform` : où cela se trouve, quelle taille cela a, quelle orientation
- Les transformations hiérarchiques sont contrôlées par la relation parent/enfant
  - Similaire à `QWidget`, `QQuickItem`, etc.
- Si la scène est tracée, nous avons besoin d'un point de vue dessus
  - Il est décrit avec `Qt3DRender::QCamera`

Qt3DCore::QTransform

- Inherits from `Qt3DCore::QComponent`
- Represents an affine transformation
- Three ways of using it:
  - Through properties: scale3D, rotation, translation
  - Through helper functions: `rotateAround()`
  - Through the matrix property
- Transformations are applied:
  - to objects in Scale/Rotation/Translation order
  - to coordinate systems in Translation/Rotation/Scale order
- Transformations are multiplied along the parent/child relationship

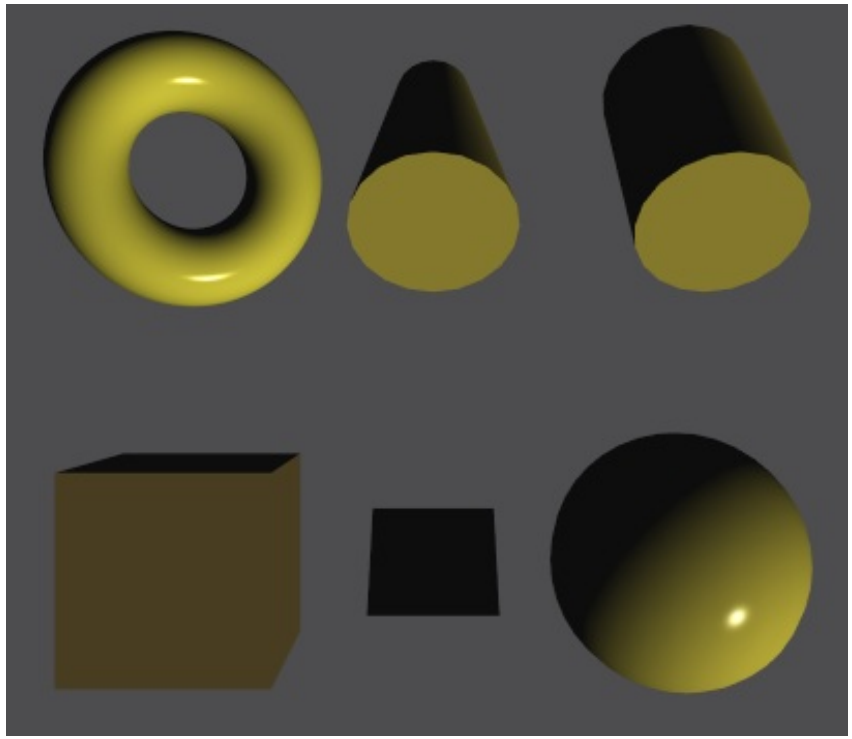
Transformations

```
1  import Qt3D.Core 2.0
2
3  Entity {
4      components: [
5          Transform {
6              scale3D: Qt.vector3d(1, 2, 1.5)
7              translation: Qt.vector3d(0, 0, -1)
8          }
9      ]
10
11     Entity {
12         components: [
13             Transform { translation: Qt.vector3d(0, 1, 0) }
14         ]
15     }
16 }
```



Géométries

- `Qt3DRender::QRenderAspect` trace les `Qt3DCore::QEntitys` avec une forme
- La propriété geometry de `Qt3DRender::QGeometryRenderer` spéfifie la forme
- Qt 3D fourni des sous-classes de `Qt3DRender::QGeometryRenderer` :
  - `Qt3DExtras::QSphereMesh`
  - `Qt3DExtras::QCuboidMesh`
  - `Qt3DExtras::QPlaneMesh`
  - `Qt3DExtras::QTorusMesh`
  - `Qt3DExtras::QConeMesh`
  - `Qt3DExtras::QCylinderMesh`



Qt Demo examples/qt3d/basicshapes-cpp

Matériaux

- Si une `Qt3DCore::QEntity` a uniquement une forme elle apparaîtra noire
- Le composant `Qt3DRender::QMaterial` fourni l'apparence d'une surface
- Qt 3D fourni des sous-classe de `Qt3DRender::QMaterial` :
  - `Qt3DExtras::QPhongMaterial`
  - `Qt3DExtras::QPhongAlphaMaterial`
  - `Qt3DExtras::QDiffuseMapMaterial`
  - `Qt3DExtras::QDiffuseSpecularMapMaterial`
  - `Qt3DExtras::QGoochMaterial`
  - ...

Demo qt3d/sol-textured-scene

Lumières

- Même avec les géométries et matériaux on ne verrait rien
- Nous avons besoin de lumières
  - ... heureusement Qt 3D positionne une lumière par défaut si aucune n'est fournie
- En général on veut contrôler l'éclairage d'une scène
- Qt 3D fourni les types de lumières suivants :
  - `DirectionalLight`
  - `PointLight`
  - `SpotLight`

Lab qt3d/ex-lights-qml

Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- **La partie immergée de l'iceberg**
- Le futur de Qt 3D

Faire ses propres géométries

- Grâce à `Qt3DRender::QBuffer` nous pouvons créer nos propres vertices
- `GeometryRenderer` contrôle comment les buffers sont combinés et interprêtés
- Utile pour programmer ses propres géométries :
  - Depuis une fonction
  - Depuis un ensemble de données
  - Depuis une interaction utilisateur

Demo qt3d/ex-surface-function

Composition de textures et filtrage

- Il est possible d'utiliser plusieurs textures depuis un seul matériau
- Il est aussi facile de réutiliser le modèle d'éclairage par défaut
- Ensuite vous pouvez combiner comme vous le souhaitez depuis un shader

Demo qt3d/sol-earth

Textures procédurales

- Beaucoup d'exemples sont disponibles sur le Web
  - `https://www.shadertoy.com/`
  - En général écrits pour WebGL ou OpenGL ES 2
  - Peut nécessiter un peu d'adaptation
  - La plupart sont loin d'être triviaux !
- Mais ils sont faciles à intégrer dans `Material` et à paramétrer

Demo qt3d/ex-plasma

Intégrer avec QtQuick grâce à Scene3D

- Fourni par le module `QtQuick.Scene3D`
- Prend une `Entity` en enfant qui sera votre scène complète
- Les aspects chargés sont contrôlés par la propriété aspects
- Les événements de survol de la souris sont acceptés uniquement si la propriété hoverEnabled est vraie

Demo qt3d/ex-controls-overlay

Et plus...

- Gestion de couches
- Matériaux et modèles d'éclairage spécifiques
- Texture mipmaps
- Cube Maps
- Portabilité de votre code sur plusieurs versions d'OpenGL
- Contrôle complet sur l'algorithme de rendu
- Chargement d'objets ou de scènes complètes depuis des fichiers (3ds, collada, qml...)
- Effets de post-processing (simple ou multi-passes)
- Instanced rendering
- etc.

Demo qt3d/ex-multiple-effects

Demo qt3d/sol-asteroids

Les bases de Qt 3D

- Fonctionnalités
- Système Entité Composant? Kezaco?
- Hello Donut
- L'ECS de Qt 3D expliqué
- Gestion des entrées
- Les bases du tracé 3D
- La partie immergée de l'iceberg
- **Le futur de Qt 3D**

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 thread sur le backend - jobs qui créent des jobs
- Qt 3D Render
  - Utiliser Qt Quick ou QPainter pour tracer dans une texture
  - Embarquer Qt Quick dans Qt 3D avec gestion des entrées
  - Support du niveau de détail pour les géométries
  - Billboards - objets faisant toujours face à la caméra
  - Support du texte - 2D et 3D
  - Matériaux additionnels tel que le support du Physics Based Rendering
  - Systèmes de particules
- Entrées Qt 3D
  - Axes d'entrée qui applique une valeur cumulée telle que la position, la vélocité ou l'accélération
  - 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 ?

KDAB

- Nouveaux aspects:
  - Détection de collisions
    - Permet de détecter dans des entités sont en collision ou entrent/ sortent de volumes dans l'espace
  - Animation
    - Animation par keyframe
    - Animation de squelettes
    - Morphing
    - Retirer les animations de la thread principale
  - 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