Qt 3D Basics

Kévin Ottens, Software Craftsman at KDAB
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut
- Qt 3D ECS Explained
- Input Handling
- Drawing Basics
- Beyond the Tip of the Iceberg
- The Future of Qt 3D
Qt 3D Basics

- Feature Set
  - Entity Component System? Kezaco?
  - Hello Donut
  - Qt 3D ECS Explained
  - Input Handling
  - Drawing Basics
  - Beyond the Tip of the Iceberg
  - 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 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

- Domains can be added via independent aspects
  - ... 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
Qt 3D Basics

- Feature Set
- **Entity Component System? Kezaco?**
- Hello Donut
- Qt 3D ECS Explained
- Input Handling
- Drawing Basics
- Beyond the Tip of the Iceberg
- 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
Composition vs Inheritance

- Let's analyse a familiar example: Space Invaders
Composition vs Inheritance cont'd

- Typical inheritance hierarchy

![Inheritance Hierarchy Diagram]

Entity Component System? Kezaco?
Composition vs Inheritance cont’d

- All fine until customer requires new feature:
Composition vs Inheritance cont'd

- Typical solution: Add feature to base class
Composition vs Inheritance cont'd

- Doesn't scale:

Entity Component System? Kezaco?
Composition vs Inheritance cont'd

- What about multiple inheritance?
Composition vs Inheritance cont'd

- What about mix-in multiple inheritance?
Composition vs Inheritance cont'd

- Does it scale?

Entity Component System? Kezaco?
Composition vs Inheritance cont'd

- Is inheritance flexible enough?
Composition vs Inheritance cont'd

- Inheritance:
  - Relationships baked in at design time.
  - Complex inheritance hierarchies: deep, wide, multiple inheritance
  - Features tend to migrate to base class

- Entity Component System
  - Allows changes at runtime
  - Avoids inheritance limitations
  - Has additional costs:
    - More QObjects
    - Different to most OOP developer's experience
  - We don't have to bake in assumptions to Qt 3D that we can't later change when adding features.
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- **Hello Donut**
  - Qt 3D ECS Explained
  - Input Handling
  - Drawing Basics
  - Beyond the Tip of the Iceberg
  - 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
  - ...
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut

**Qt 3D ECS Explained**

- Input Handling
- Drawing Basics
- Beyond the Tip of the Iceberg
- The Future of Qt 3D
Everything is a QNode

- `Qt3DCore::QNode` is the base type for everything
  - It inherits from `QObject` and all its features
  - Internally implements the frontend/backend communication

- `Qt3DCore::QEntity`
  - It inherits from `Qt3DCore::QNode`
  - It just aggregates `Qt3DCore::QComponents`

- `Qt3DCore::QComponent`
  - It inherits from `Qt3DCore::QNode`
  - Actual data is provided by its subclasses
    - `Qt3DCore::QTransform`
    - `Qt3DRender::QMesh`
    - `Qt3DRender::QMaterial`
    - ...

---

Qt 3D ECS Explained
Everything is a QNode cont'd

QObject

QNode

QEntity 1

QComponent

Adds unique ID and communication to aspects

Vertical slice of data/behavior

Simulated object. Aggregates components
You Still Need a System

- The simulation is executed by Qt3DCore::QAspectEngine
- Qt3DCore::QAbstractAspect subclass instances are registered on the engine
  - Behavior comes from the aspects processing component data
  - Aspects control the domains manipulated by your simulation
- Qt 3D provides
  - Qt3DRender::QRenderAspect
  - Qt3DInput::QInputAspect
  - Qt3DLogic::QLogicAspect
- Note that aspects have no API of their own
  - It is all provided by Qt3DCore::QComponent subclasses
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut
- Qt 3D ECS Explained
- **Input Handling**
  - Drawing Basics
  - Beyond the Tip of the Iceberg
  - The Future of Qt 3D
Physical Devices

- To handle input we first need to generate input events
- Subclasses of Qt3DInput::QAbstractPhysicalDevice represent input devices
  - Qt3DInput::QKeyboardDevice
  - Qt3DInput::QMouseDevice
  - Others can be added later
- On its own a device doesn't do much
  - Input handlers expose signals emitted in response to events
Picking

- High level picking provided by Qt3DRender::QObjectPicker component
  - Implicitly associated with mouse device
  - Uses ray-cast based picking

- Qt3DRender::QObjectPicker emits signals for you to handle:
  - pressed(), released(), clicked()
  - moved() - only when dragEnabled is true
  - entered(), exited() - only when hoverEnabled is true

- The containsMouse property provides a more declarative alternative to entered(), exited()
Physical Devices vs Logical Devices

- Physical devices provide only discrete events
- Hard to use them to control a value over time
- Logical device provides a way to:
  - Have an analog view on a physical device
  - Aggregate several physical devices in a unified device
Logical Input Action

- Qt3DInput::QAction provides a binary value
- It is activated by some input, can be:
  - A single button input with Qt3DInput::QActionInput
  - A simultaneous combination of button inputs with Qt3DInput::QInputChord
  - A sequence of button inputs with Qt3DInput::QInputSequence
- When the action state changes the active property is toggled

Demo qt3d/ex-logical-input-qml
Logical Input Axis

- `Qt3DInput::QAxis` provides an analog value between -1 and 1.
- It varies over time when some input is generated, can be:
  - When a physical axis varies with `Qt3DInput::QAnalogAxisInput`
  - While a button is pressed with `Qt3DInput::QButtonAxisInput`
- When the axis state changes the `value` property changes

Demo `qt3d/ex-logical-axes-qml`
Putting it All Together: Moving Boxes

- Focus managed using tab
- Focused box appears bigger
- The arrows move the box on the plane
- Page up/down rotate the box on its Y axis
- Boxes light up when on mouse hover
- Clicking on a box gives it the focus
- Boxes can be moved around with the mouse

Demo qt3d/sol-moving-boxes-qml-step3
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut
- Qt 3D ECS Explained
- Input Handling

**Drawing Basics**
- Beyond the Tip of the Iceberg
- The Future of Qt 3D
The Scene Graph

- The scene graph provides the spatial representation of the simulation
  - Qt3DCore::QEntity: what takes part in the simulation
  - Qt3DCore::QTransform: where it is, what scale it is, what orientation it has

- Hierarchical transforms are controlled by the parent/child relationship
  - Similar to QWidget, QQuickItem, etc.

- If the scene is rendered, we need a point of view on it
  - This is provided by 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
import Qt3D.Core 2.0

Entity {
    components: [
        Transform {
            scale3D: Qt.vector3d(1, 2, 1.5)
            translation: Qt.vector3d(0, 0, -1)
        }
    ]
}

Entity {
    components: [
        Transform {
            translation: Qt.vector3d(0, 1, 0)
        }
    ]
}

Diagram of entities and transformations
Geometries

- Qt3DRender::QRenderAspect draws Qt3DCore::QEntitys with a shape
- Qt3DRender::QGeometryRenderer's geometry property specifies the shape
- Qt 3D provides convenience subclasses of Qt3DRender::QGeometryRenderer:
  - Qt3DExtras::QSphereMesh
  - Qt3DExtras::QCuboidMesh
  - Qt3DExtras::QPlaneMesh
  - Qt3DExtras::QTorusMesh
  - Qt3DExtras::QConeMesh
  - Qt3DExtras::QCylinderMesh

Qt Demo examples/qt3d/basicshapes-cpp
Materials

- If a Qt3DCore::QEntity only has a shape it will appear black

- The Qt3DRender::QMaterial component provides a surface appearance

- Qt 3D provides convenience subclasses of Qt3DRender::QMaterial:
  - Qt3DExtras::QPhongMaterial
  - Qt3DExtras::QPhongAlphaMaterial
  - Qt3DExtras::QDiffuseMapMaterial
  - Qt3DExtras::QDiffuseSpecularMapMaterial
  - Qt3DExtras::QGoochMaterial
  - ...

Demo qt3d/sol-textured-scene
Lights

- Even with shapes and materials we would see nothing
- We need some lights
  - ... luckily Qt 3D sets a default one for us if none is provided
- In general we want some control of the scene lighting
- Qt 3D provides the following light types:
  - DirectionalLight
  - PointLight
  - SpotLight

Lab qt3d/ex-lights-qml
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut
- Qt 3D ECS Explained
- Input Handling
- Drawing Basics
- Beyond the Tip of the Iceberg
- The Future of Qt 3D
Making your Own Geometries

- Using `Qt3DRender::QBuffer` we can create our own vertices
- `GeometryRenderer` controls how buffers are combined and parsed
- Useful to make your own geometries programmatically:
  - From a function
  - From data sets
  - From user interaction

Demo `qt3d/ex-surface-function`
Texture Composition and Filtering

- Possible to sample several textures in a single material
- Also easy to reuse stock lighting model
- Then you can blend as you see fit in the shader

Demo qt3d/sol-earth
Procedural Textures

- Lots of examples available on the Internet
  - https://www.shadertoy.com/
  - Usually written for WebGL or OpenGL ES 2
  - May require some adaptation
  - Many are far from simple!

- But they are easy to plug in the Material system and to parameterize

Demo qt3d/ex-plasma
Integrating with QtQuick using Scene3D

- Provided by the QtQuick.Scene3D module
- Takes an Entity as child which will be your whole scene
- Loaded aspects are controlled with the aspects property
- Hover events are only accepted if the hoverEnabled property is true

Demo qt3d/ex-controls-overlay
And more...

- Layer management
- Own materials and lighting models
- Texture mipmaps
- Cube Maps
- Portability of your code across several OpenGL versions
- Complete control over the rendering algorithm
- Loading complete objects or scenes from files (3ds, collada, qml...)
- Post-processing effects (single or multi-pass)
- Instanced rendering
- etc.

Demo qt3d/ex-multiple-effects
Demo qt3d/sol-asteroids
Qt 3D Basics

- Feature Set
- Entity Component System? Kezaco?
- Hello Donut
- Qt 3D ECS Explained
- Input Handling
- Drawing Basics
- Beyond the Tip of the Iceberg
- **The Future of Qt 3D**
What does the future hold for Qt 3D?

- **Qt 3D Core**
  - Efficiency improvements
  - Backend threadpool and job handling improvements - jobs spawning jobs

- **Qt 3D Render**
  - Use Qt Quick or QPainter to render into a texture
  - Embed Qt Quick into Qt 3D including input handling
  - Level of Detail (LOD) support for meshes
  - Billboards - camera facing entities
  - Text support - 2D and 3D
  - Additional materials such as Physics Based Rendering (PBR) materials
  - Particle systems

- **Qt 3D Input**
  - Axis inputs that apply cumulative axis values as position, velocity or acceleration
  - 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?

- New aspects:
  - Collision Detection Aspect
    - Allows to detect when entities collide or enter/exit volumes in space
  - Animation Aspect
    - Keyframe animation
    - 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.