Qt 3D Basics

CERN HSF Visualization workshop, 28/03/2017

Presented by Mike Krus - mike.krus@kdab.com

Material based on Qt 5.5, created on March 28, 2017
Introducing Qt Quick

What is Qt Quick?

A set of technologies including:
- Declarative markup language: QML
- Language runtime integrated with Qt
- Qt Creator IDE support for the QML language
- Graphical design tool
- C++ API for integration with Qt applications

Philosophy of Qt Quick

- Intuitive user interfaces
- Design-oriented
- Rapid prototyping and production
- Easy deployment

Qt 3D Basics
Feature Set

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

Feature Set

- 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

ECS: Definitions

Entity Component System?

Qt 3D Basics

Entity Component System?

Composition vs Inheritance

- Let's analyse a familiar example: Space Invaders

Entity Component System?

Entity Component System?
Composition vs Inheritance cont'd

- Typical inheritance hierarchy

```
    Object
     |     |     |     |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+

Entity Component System?
```

Composition vs Inheritance cont'd

- All fine until customer requires new feature:

```
    Object
     |     |     |     |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+

Entity Component System?
```

Composition vs Inheritance cont'd

- Typical solution: Add feature to base class

```
    Object
     |     |     |     |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+

Entity Component System?
```

Composition vs Inheritance cont'd

- Doesn't scale:

```
    Object
     |     |     |     |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+
     |   |   |   |   |   |
    +---+---+---+---+---+

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

- What about multiple inheritance?

Entity Component System?

- What about mix-in multiple inheritance?

Entity Component System?

Composition vs Inheritance cont'd

- Does it scale?

Entity Component System?

- Is inheritance flexible enough?

Entity Component System?
Composition vs Inheritance cont'd

- Is traditional fixed composition the panacea?

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

Entity Component System Wrap-up

- Inheritance:
  - Relationships baked in at design time
  - Complex inheritance hierarchies: deep, wide, multiple inheritance
  - Features tend to migrate to base class
- Fixed Composition
  - Relationships still baked in at design time
  - Fixed maximum feature scope
  - Lots of functional domain details in the scene object
  - If functional domain objects contain both data and behavior they will have lots of inter-dependencies
- 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

Hello Donut
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

---

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**

**Qt 3D ECS Explained**

- `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`
    - ...

---

Demo [qt3d/ex-helldonut-qml](http://example.com)

---

Everything is a QNode


**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**

- 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`

---

**The Scene Graph**

```plaintext
Simulated object. Aggregates components
```

Vertical slice of data/behavior

- Adds unique ID and communication to aspects

---

**Everything is a QNode cont’d**

```plaintext
QObject

QNode

QEntity

QComponent
```

Simulated object. Aggregates components
The Scene Graph

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

Transforms cont’d

```plaintext
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) }
  ]
}
```}

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

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
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
  - ...

Lab qt3d/ex-lights-qml

Demonstration:

```cpp
import Qt3D.Render 2.0
...
Material {
  parameters: Parameter ( name: "colorTint"; value: "yellow" )
  effect: Effect {
    techniques:
      Technique {
        filterKeys: FilterKey ( name: "renderingStyle"; value: "forward" )
        graphicsApiFilter {
          api: GraphicsApiFilter.OpenGL
          majorVersion: 3
          minorVersion: 2
          profile: GraphicsApiFilter.CoreProfile
        } renderPasses: RenderPass {
          shaderProgram: ShaderProgram {
            vertexShaderCode: loadSource("qrc:/customshader.vert")
            fragmentShaderCode: loadSource("qrc:/customshader.frag")
          }
        }
      }
  }
}
```

Lab qt3d/ex-glsl

Textures 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/ex-glsl

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

Demonstration:

```cpp
import Qt3D.Render 2.0
...
```
Interacting with the scene

Qt 3D Basics

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(pick), released(pick), clicked(pick)
  - moved(pick) - only when dragEnabled is true
  - entered(), exited() - only when hoverEnabled is true
- The containsMouse property provides a more declarative alternative to entered(), exited()
- The pick parameter of the signals is a Qt3DRender::QPickEvent
  - position in screen space
  - localIntersection in model space
  - worldIntersection in world space

Demo qt3d/ex-object-picker

Demo qt3d/ex-object-picker-qml

Interacting with the scene

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 it's own a device doesn't do much
  - Input handlers expose signals emitted in response to events

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
- Combined with input handler

Demo qt3d/ex-mouse-handler-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

[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/sol-moving-boxes-qml-step3

In 5.9, Scene2D, fully interactive Qt Quick UI mapped onto 3D geometry

Qt 3D Basics

The Qt 3D Frame Graph

- With what we have seen so far, we can:
  - Draw geometry loaded from disk or generated dynamically
  - Use custom materials with shaders to change surface appearance
  - Make use of textures to increase surface details
- What about shadows?
- What about transparency?
- What about post processing effects?
- All these and others require control over how we render the Scene Graph
- The Frame Graph describes the rendering algorithm

[Scene2D]

Demo qt3d/ex-controls-overlay
**Structure and Behavior**

- The nodes of the Frame Graph form a tree
- The entities of the Scene Graph form a tree
- The Frame Graph and Scene Graph are linearized into render commands

**The Simplest Frame Graph**

- It is important to structure your Frame Graph properly for performance reasons
- Might lead to deep and narrow trees
- Simplest case being a one pass forward renderer

**Several Points of View on a Scene**

- **Camera** describes a point of view on a scene
- **Viewport** allows to split the render surface in several areas
  - They can be nested for further splitting
- **CameraSelector** allows to select a camera to render in a Viewport
- **ClearBuffers** describes which buffers are cleared during the rendering
  - Generally necessary to get anything on screen
  - Also an easy way to control background color
- To avoid a branch to trigger a rendering give it a **NoDraw** element as leaf

**Showing Different Scenes in Viewports**

- Our **Viewports** all display the same scene...
- But they can display different subsets of the scene using layers
- Attach each entity to a **Layer**
- Have each **Viewport** display a subset of the entities using **LayerFilter**

*Demo qt3d/ex-viewports-and-layers*
Image-Base Techniques

- Rendering to a texture, support for framebuffers
- Multiple passes
- Post-Processing Effects
  Demo qt3d/ex-gaussian-blur
  Demo qt3d/ex-multiple-effects

Beyond the Tip of the Iceberg

And more...

- 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/sol-asteroids
Demo qt3d/ex-instanced-geometry

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 (5.9)
- Level of Detail (LOD) support for meshes (5.9)
- Billboards - camera facing entities
- Text support - 2D and 3D (5.9)
- 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

New aspects:
- Collision Detection Aspect
  - Allows to detect when entities collide or enter/exit volumes in space
- Animation Aspect
  - Keyframe animation (5.9 TP)
  - 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
  - Qt 3D Studio
  - Build time tooling - asset conditioners for meshes, textures etc.

Qt 3D and the rest of Qt:
- DataVis, Mapping are likely to be based on Qt 3D
- Work on unifying rendering toolset
  - Single renderer, Vulkan, D12, Metal backends