Qt World Summit 2017

October 10-12 | Berlin, Germany

Interacting with 3D Content

Mike Krus, Senior Software Engineer at KDAB

Interacting with 3D Content

- Why this talk?
- What is Qt3D?
- Handling input devices
- Adding controls
- Mixing with traditional UIs

Why this talk?

- A third input handling stack : (
- A third dimension
- No predefined controls
- No user experience

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

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.
- Create objects to be rendered
  - `Qt3DRender::QGeometryRenderer`'s geometry property specifies the shape
  - The `Qt3DRender::QMaterial` component provides a surface appearance
  - Subclasses of `Qt3DRender::QAbstractTexture` provide different types of texture
- If the scene is rendered, we need a point of view on it
  - This is provided by `Qt3DRender::QCamera`

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`
  - ...

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-qml

Pick Settings

- `RenderSettings` is a `Component` allowing to control the render aspect
- Only one instance is allowed
- It is generally set on the root `Entity` of the scene
- It allows to control picking via the pickingSettings grouped property
  - By default it uses bounding sphere volume picking (PickingSettings.BoundingVolumePicking)
  - Some scenes require the more expensive triangle picking (PickingSettings.TrianglePicking)
  - As of 5.10, also pick lines and points
  - This changes the type of event received in `ObjectPicker` handlers

Demo: Moving Boxes - Part 1

- Light up each box when the mouse hovers over it
- Give focus by clicking on a box
- Focused box should appear bigger
- Optional:
  - Move focused box around using the object picker

Demo qt3d/sol-moving-boxes-qml-step1

Physical Devices

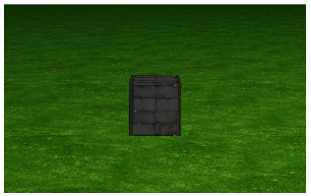
- To handle input we first need to generate input events
- Subclasses of `Qt3DInput::QAbstractPhysicalDevice` represent input devices
  - `Qt3DInput::QKeyboardDevice`
  - `Qt3DInput::QMouseDevice`
  - Others can be added later
- On it's own a device doesn't do much

Input Handlers

- Physical devices need to be partnered with an input handler
- `Qt3DInput::QKeyboardHandler` and `Qt3DInput::QMouseHandler` are both components
  - Attach them to an entity
  - Associate a physical device with its handler by the handler's sourceDevice property
  - The handler then receives events from the physical device
  - The `Qt3DInput::QKeyboardHandler` only receives events if its focus property is true
- Both handlers expose signals that are emitted in response to events

Mouse Handler (QML)

```
1  import Qt3D.Input 2.0
2  ...
3
4  MouseDevice {
5      id: mouseDevice
6  }
7
8  MouseHandler {
9      sourceDevice: mouseDevice
10
11     onReleased: {
12         switch (mouse.button) {
13         case Qt.LeftButton:
14             box.textureBaseName = "pattern_10";
15             break;
16         case Qt.RightButton:
17             box.textureBaseName = "pattern_09";
18             break;
19         }
20     }
21 }
```



Demo qt3d/ex-mouse-handler-qml

Keyboard Handler (QML)

```
1  import Qt3D.Input 2.0
2  ...
3
4  KeyboardDevice {
5      id: keyboardDevice
6  }
7
8  KeyboardHandler {
9      sourceDevice: keyboardDevice
10     focus: true
11     onUpPressed: box.position.z -= 0.5
12     onDownPressed: box.position.z += 0.5
13     onLeftPressed: box.position.x -= 0.5
14     onRightPressed: box.position.x += 0.5
15 }
```

Demo qt3d/ex-keyboard-handler-qml

Demo: Moving Boxes - Part 2

- Give focus to a box using tab
- Move the box on the plane using the arrows
- Optional:
  - Allow to rotate boxes on their Y axis with page up/down

Demo qt3d/sol-moving-boxes-qml-step2

Physical vs Logical

- 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

Demo: Moving Boxes - Part 3

- The keyboard control of the boxes is still step by step
- Improve the code so that the boxes move and rotate smoothly when controlled with the keyboard

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

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 (QML, 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: Moving Boxes - Part 4

- When the axis value reaches it's maximum, nothing happens anymore (very visible on rotation)
- One would expect the movement to carry on indefinitely
- Improve the code so that the boxes move and rotate indefinitely when the corresponding key is pressed

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

The Scene3D Element

- 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
- Works with the usual `QQuickView` or `QQmlApplication` in your `main()`

The Scene3D Element

Demo qt3d/ex-controls-overlay

Qt Demo examples/qt3d/scene3d

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

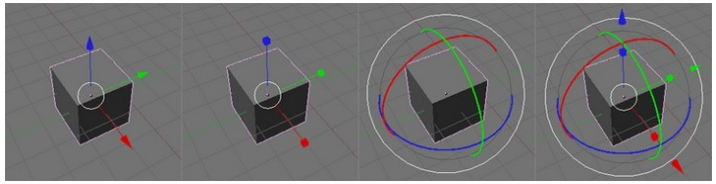
Demo qt3d/ex-samegame

Future in Qt3D interaction

- Extension to picking
  - Get all picked objects (as a list)
  - Nothing picked event
  - Event *bubbling*
  - Generalised picking (non-event based, not in screen space only)
- More controllers, especially related to VR
- Haptic feedback?
- Combine with physics and collision detection

Manipulators

- Moving and deforming objects in 3D is hard
- Needs contraining to separate dimensions and operations
- Combine picking and generale device handling
- Manipulators are the controls of the 3D world



Examples from Blender

Demo qt3d/dragging

Qt World Summit 2017

October 10-12 | Berlin, Germany

Thank you!

www.kdab.com

mike.krus@kdab.com