Breathing life into your applications: Animation with Qt 3D

Dr Sean Harmer
Managing Director, KDAB (UK)
sean.harmer@kdab.com
Contents

• Overview of Animations in Qt 3D
• Simple Animations
• Skeletal Animations
• Blended Animations
• For the future

Credit to Johann Woelpner and Timo Buske for producing the assets
Overview of Animations in Qt 3D
Why are we doing this?

- Qt is not only for developers
- Content creators too
- Developers not great at complex content
- Artists not great at software development
- Let each do what they are good at!
Why are we doing this?

- Scalability
- Complex animations consume lots of data
- Qt 3D scales well to many cores
- Non-blocking on main thread
- Frontend objects can opt-in to property updates
What is Animation?

• Sequence of still frames
• Rapid display fools our primitive monkey brains
• Traditional animators draw every frame
Key Framed Animations

- Computers are good at maths
- Animators set positions at key points in time (frames)
- Get the computer to interpolate
Offline vs Real-time

• Offline rendering (TV & Movies)
  • Time of each frame known exactly
  • Exactly scripted

• Real-time rendering (Applications/Games)
  • Variations in timing
  • Interactive
Offline vs Real-time

• Offline rendering
  • Artists can set exact positions at every frame
  • Animations for one specific use
Offline vs Real-time

• Real-time rendering
  • Need to calculate position at any time
  • Animations need to be reusable
  • Ideally animations should be able to be composed
Interpolation

- Linear Interpolation (LERP)
  - Simpler mathematics
  - More data

- Higher Order
  - More complicated mathematics
  - Less data

\[ y = y_1 + (x - x_1) \frac{y_2 - y_1}{x_2 - x_1} \]
Workflow

- Artists author using higher order
- Asset conditioning
  - Process to run-time format
  - Export curves or re-sample curves for lerp
- Consumption
Authoring Example

- Simple bouncing ball
Animation Tips

- Artists can use familiar tools
- Using data rather than forcing programmer art:
  - Squash and stretch
  - Anticipation
  - Variation
  - Fine control (rebound, inertia etc.)
Simple Animations
Animations at Runtime

• 3 concepts:
  • Animation data
  • Animation playback
  • Target for the animation

• Qt Quick animations conflate all 3
• Qt 3D separates them for reuse, flexibility and efficiency
Animation Data

- Data from artist:
  - AnimationClipLoader
- Data from application:
  - AnimationClip

AnimationClipLoader {
  id: animationClip
  source: "qrc:/assets/gltf/2.0/Robot/robot.gltf"
}

AnimationClip {
  clipData: _animation.createData()
}
Animation Playback

- Simple playback achieved with ClipAnimator
- More advanced options available (see later)

```javascript
clipAnimator {
    id: animator
    clip: AnimationClip {
        clipData: _animation.createData()
    }
}
```
Animation Targets

- Animations are reusable
- Map animation data to multiple properties of multiple objects
- ChannelMapper and ChannelMapping

```javascript
clipAnimator {
    id: animator

    channelMapper: ChannelMapper {
        ChannelMapping { channelName: "Location"; target: transform; property: "translation" }
        ChannelMapping { channelName: "Rotation"; target: transform; property: "rotation" }
        ChannelMapping { channelName: "Color"; target: material; property: "ambient" }
    }
}
```
Playback Example

- Simple bouncing ball
Skeletal Animations
What is Skeletal Animation?

• So far we’ve dealt with rigid body transforms
  • Acts on whole mesh
  • Limitations on how “alive” we can make objects feel
• Skeletons (Armatures) allow to deform a mesh
  • Living creatures deform
  • Handy to be able to animate parts of a mesh
  • Does not need to be “squishy” like us meatbags
• Before we animate…
• We need to know how to render skinned meshes
Creating Skinned Meshes

• Artist:
  • Creates mesh
  • Creates a skeleton
  • Binds vertices of mesh to one or more bones
  • Bone indices and weights stored as per-vertex attributes of the mesh
  • Usually limited to 4 bones influencing each vertex
  • Creates animations for bones (key framed poses)
  • Export
Joint vs Bone

- Literature refers to **Joints** and **Bones**
- The terms are interchangeable
- DCC tools often show bones graphically
- Joint is technically more correct
- Not physical bones
- Skeleton is just a hierarchy of nested coordinate systems
  - Usual parent child inheritance of transforms
  - Just applied within a single Entity
Drawing a Skinned Mesh
Single vertex single joint

\[ \mathbf{v}_j = (4, 3) \]

\[ \mathbf{v}_{global} = (10, 16) \]

\[ \mathbf{v}_{j,C} = (4, 3) \]

\[ \mathbf{v}_{j,C} = C_j B_j^{-1} \mathbf{v}_j \]
Drawing a Skinned Mesh

- Realistic Case: Many vertices, weighted joints
- Exactly the same maths!
- We store the joint poses as local transforms
- To calculate joint global pose we need to compose it with parent, grandparent…
- Inverse bind matrix never changes
- Best performance by linearising the joint hierarchy
- Calculate skinning matrix for each joint in the array
- Pass to shader program as uniform array or UBO
- Vertex shader applies weighted skinning matrices to vertices
Skinned Meshes in Qt 3D

- Entity should aggregate:
  - GeometryRenderer component referencing…
    - Geometry containing joint indices and joint weights
    - Use Mesh for loading from file
  - Armature component referencing…
    - Skeleton of joints
    - Use SkeletonLoader for loading from file
    - Can optionally create frontend hierarchy of Joints
Skinned Mesh Example

```javascript
Entity {
    id: root

    components: [
        Transform {
            id: transform
        },
        Mesh {
            source: "qrc:/assets/gltf/2.0/Robot/robot.gltf"
        },
        Armature {
            skeleton: SkeletonLoader {
                id: skeleton
                source: "qrc:/assets/gltf/2.0/Robot/robot.gltf"
                onStatusChanged: console.log("skeleton loader status: " + status)
                onJointCountChanged: console.log("skeleton has " + jointCount + " joints")
            },
            Material { id: material... }  
        }
    ]
}
```
Animating a Skinned Mesh

• We already have everything we need!
• Animation data contains local transforms of joints
• Map animation data to skeleton with SkeletonMapping
• Animator updates skeleton pose and sends to render aspect
  • Simple or with a blend tree
• Renderer calculates new skinning matrix palette and…
• Draws skinned mesh as usual
Skeletal Animation Example

• Humanoid
Playback Speed

• Nice to be able to control animation playback speed
• Default uses global simulation time (wall time)
• Can set a Clock on one or more ClipAnimators
• Control speed with Clock’s playbackRate property
• All associated animators affected
• Useful for some effects
  • E.g. Allows to slow down animation of 3D objects whilst keeping 2D UI fluid
Blended Animations
Why Blend?

• Avoid combinatorial explosion
• Smooth transitions
• Run-time control
**Uses of Animation Blending**

- Smoothly mix walk cycle and run cycle
- Animate head in different ways whilst walking
- Blend from healthy to injured
- Transition from idle to walking
- Blend from backward/forward motion to strafing
How to BlendAnimations

• Simple case: Walking vs Running
• Walk animation cycle: 3 seconds
• Run animation cycle: 2 seconds
• Blend factor controlled by user input (Axis)
• Or any other program data
How to Blend Animations

- Work in normalised time (phase on range \([0,1]\)) for each contributing clip
- Requires feet to hit ground at same phase in both clips
How to Blend Animations

• From global time, calculate phase, $\phi$
• Evaluate channels from walk clip at $\phi, A(\phi)$
• Evaluate channels from run clip at $\phi, B(\phi)$
• LERP to get resulting channel values $C = (1 - \beta)A + \beta B$
• Complicated to deal with missing data
Other Types of Blend

• LERP is common

• Other possible types:
  • Additive
  • Generalised 1D LERP
  • Bilinear 2D LERP
  • Barycentric LERP
  • Generalised Barycentric LERP
Combining Blends

• No reason to limit to a single blend?
• Combine in an arbitrary **blend tree**
• An Abstract Syntax Tree (AST) for animations
• “Constants” are animation clips
• Arguments are blend parameters
  • Bound to user inputs or
  • Application data
Animation Blending Example

• Simple LERP Blend Tree
For the Future
Future Animation Work

• Morph target animation: Absolute and relative targets
• Orchestrated clip animator: State machine controlled set of blend trees with transitions
• Channel masking and blending operations
• Root motion extraction
• More optimisations
• Tooling: Graphical blend tree designer
Summary

- Qt 3D offers high performance animations
- Opt-in to property changes
- Artists create data
- Developers integrate data into application
- First class support for skeletal animations
- Playback rate support
- Animation blending
Thank you for listening!
Any questions?

https://www.kdab.com
Sean Harmer <sean.harmer@kdab.com>