photon

Search

FUSION

QUANTUM

API Reference

Getting Started

Quantum 100

Game Samples

Technical Samples

AddOns

Manual

- Quantum Ecs
- Animation
- Assets
- Cheat Protection
- Commands
- Configuration Files
- Custom Server Plugin
- Entity Prototypes
- Entity View
- Frames
- Game Session
- Git Ignore Files
- Input
- Materialization
- Map Baking

Physics
Player
Prediction Culling
Profiling
Quantum Project
Replays
RNG Session
WebGL

Concepts And Patterns

Consoles

Gaming Circle

Reference

REALTIME

CHAT

VOICE

SERVER

PUN

BOLT

VR | AR | MR

photon

Animation

- Overview
- Polling Based Animation
  - Trigger Events
  - Tips
- Deterministic Animation
  - Tips

Overview

In Quantum there are two distinct ways to handle animation:

- Poll the game state from Unity; and,
- Deterministic animation using the Custom Animator.

Back To Top

Polling Based Animation

Most games use animation to communicate the state of an object to the player. For instance, when the playable character is walking or jumping, the animations are actually ***In Place*** animations and the perceived movement is driven by code.

In other words, the scripts managing the characters' respective (Unity) animator are stateless and simply derive values to pass on as animation parameters based on data polled from the game simulation (in Quantum).

N.B.: If the gameplay systems rely on ***Root Motion*** or have to be aware of the animation state, then skip to the next section.

Here is a code snippet showcasing a basics of caching relevant Unity components and getting a frame from a QuantumGame retrieved from the update view callback, then reading relevant data from the frame and applying it to the animator. On Unity, make sure to only use the Frame API for **ready only** operations, as writing to it from Unity would be non-deterministic.

```
using Quantum;
using UnityEngine;

public class CharacterAnimations : QuantumCallbacks
{
  private EntityView _entityView;
  private Animator _animator;

  private void Awake()
  {
    _entityView = GetComponent<EntityView>();
    _animator = GetComponentInChildren<Animator>();
  }

  public override void OnUpdateView(QuantumGame game)
```

```
      _animator.SetFloat("Speed", kcc.Velocity.Magnitude.AsFloat);
    }
  }
```

Back To Top

Trigger Events

Some animations are based on a particular events taking place in the game; e.g. a player pressing jump or getting hit by an enemy. In these cases, it is usually preferable to raise an event from the simulation and have the view listen to it. This ensures decoupling and work well in conjunction with the polling based animation approach.

For a comprehensive explanation on events and callbacks, refer to the `Quantum ECS > Events & Callbacks` page in the Manual.

From Quantum, a system can trigger a Quantum Event when a character jumps:

```
using Photon.Deterministic;
namespace Quantum
{
    public unsafe struct PlayerMovementFilter
    {
        public EntityRef EntityRef;
        public PlayerID* PlayerID;
        public Transform3D* Transform;
        public CharacterController3D* Kcc;
    }

    unsafe class MovementSystem : SystemMainThreadFilter<PlayerMovementFilter>
    {
        public override void Update(Frame f, ref PlayerMovementFilter filter)
        {
            var input = f.GetPlayerInput(filter.PlayerID->PlayerRef);

            if (input->Jump.WasPressed)
            {
                f.Events.PlayerJump(filter.EntityRef);
                filter.Kcc->Jump(f);
            }
        }
    }
}
```

On the Unity side, a Unity component can listen to the `PlayerJump` event and reacts to it. These steps are necessary to achieve this:

1. Define a method that can receive the event - `void Jump(EventPlayerJump e)` .
2. Subscribe to the event in question.
3. When the event is received, check if it is meant for the GameObject the script is located on by comparing the `EntityRef` contained in the event against the one cached earlier.

```
using Quantum;
using UnityEngine;

public class CharacterAnimations : QuantumCallbacks
{
  private EntityView _entityView;
  private Animator _animator;

  private void Awake()
  {
    _entityView = GetComponent<EntityView>();
    _animator = GetComponentInChildren<Animator>();

    QuantumEvent.Subscribe<EventPlayerJump>(this, OnPlayerJump);
  }

  private void OnPlayerJump(EventPlayerJump e)
  {
    if (e.EntityRef == _entityView.EntityRef)
    {
      _animator.SetTrigger("Jump");
    }
  }
}
```

Back To Top

Tips

- Place the model and its animator component on a child object.
- Events are not part of the game state and thus are not available to late/re-joining players. It is therefore advisable to first initialize the animation state by polling the latest game state if the game has already started.
- Use synchronised events for animations that need to be triggered with 100% accuracy, e.g. a victory celebration.
- Use regular non-synchronised events for animations that need to be snappy, e.g. getting hit.
- Use the `EventCanceled` callback to graciously exit from animations triggered by a cancelled non-synchronised events. This can happen when the event was raised as part of a prediction but was rolled back during a verified frame.

Back To Top

Deterministic Animation

The main advantage of using a deterministic animation system is tick precise animations which are 100% synchronised across all clients and will snap to the correct state in case of a rollback. While this may sounds ideal, it comes with a performance impact since animations and their state are now part of the simulated game state. In reality only few games require and benefit from a deterministic animation system; among those are Fighting games and ***some*** Sports games,.

photon

so on.

Development of the Custom Animator has been halted due to the dependencies it created with Unity's Mecanim. However, the code has been open sourced and is available for download on the `Addons > Custom Animator` page. This page also provides an overview and a quick-guide on how to import and use the Custom Animator.

Keep in mind its features are limited and it will likely have to be adapted to your needs.

Back To Top

Tips

- Before using the Custom Animator, consider whether the animations are tied to the gameplay or merely a visual representation thereof. In the former case the Custom Animator is a suitable solution, otherwise polling based animation is the way to go.

To Document Top

photon

Connect

Public Discord
YouTube
Facebook
Twitter
Blog
Contact Us

Languages

English
日本語
한국어
简体中文
繁体中文