This document is about: **QUANTUM 2**

Frames

Introduction

Quantum's predict-rollback architecture allows to mitigate latency. Quantum always rolls-back and re-simulates frames. It is a necessary for determinism and involves the validation of player input by the server. Once the server has either confirmed the player input or overwritten/replaced it (only in cases were the input did not reach the server in time), the validated input of all players for a given frame is sent to the clients. Once the validated input is received, the last verified frame advances using the confirmed input.

N.B.: A player's own input will be rolled back if it has not reached the server in time or could not be validated.

Types of Frame

Quantum differenciates between two types of frame:

- verified; and,
- predicted.

Verified

A *Verified* frame is a *trusted* simulation frame. A verified frame is guaranteed to be deterministic and identical on all client simulations. The verified simulation only simulates the next verified frame once it has received the server-confirmed inputs; as such, it moves forward proportional to RTT/2 from the server.

- the input from **ALL** players is confirmed by the server for this tick; and,
- all previous ticks it follows are verified.

A partial tick confirmation where the input from ***only a subset*** of player has been validated by the server will not result in a verified tick/frame.

Predicted

Contrary to ***verified*** frames, ***predicted*** frames do not require server-confirmed input. This means the predicted frame advances with prediction as soon as the simulation has accumulated enough delta time in the local session.

The Unity-side API offers access to various versions of the predicted frame, see the API explanation below.

- `Predicted` : the simulation "head", based on the synchronised clock.
- `PredictedPrevious` (predicted - 1): used for main clock-aliasing interpolation (most views will use this to stay smooth, as Unity's local clock may slightly drift from the main server clock. Quantum runs from a separate clock, in sync with the server clock - smoothly corrected).
- `PreviousUpdatePredicted` : this is the exact frame that was the "Predicted/Head" the last time `Session.Update` was called (with the "corrected" data in it). Used for error correction interpolation (most of the time there will be no error).

API

The concept of ***Verified*** and ***Predicted*** frames exists in both the simulation and the view, albeit with a slightly different API.

Simulation

In the simulation, one can access the state of the currently simulated frame via the `Frame` class.

IsVerified	bool	Returns true if the frame is deterministic across all clients and uses server-confirmed input.
IsPredicted	bool	Returns true if the frame is a locally predicted one.

View

In the view, the ***verified*** and ***predicted*** frames are made available via `QuantumRunner.Default.Game.Frames` .

Method	Description
Validated	Trusted simulation frame, identical across all clients.
Predicted	The local simulation "head" based on the synced Quantum clock. Can differ between clients.
PredictedPrevious	Predicted - 1 Used for main clock-aliasing interpolation, most views will use this to stay smooth. As Unity's local clock may slightly drift from the main server clock, Quantum runs from a separate clock which is in sync with the server clock - smoothly corrected
PreviousUpdatePredicted	The re-simulated version of the frame that had been the "Predicted/Head" frame when the last time Session.Update was called. This is necessary in case of rollbacks in order to "correct" data held by it. It is used by the View for error-correction in the interpolation - this is a safety measure and rarely ever necessary.

Using Frame.User

You can extend the Frame by adding data to `Frame.User.cs` . However, in doing so you will also have to implement the corresponding initialization, allocation and serialization methods used by the frame.

C#

```
partial void SerializeUser(FrameSerializer serializer) // De/Seri
partial void CopyFromUser(Frame frame) // Copy to next Frame

partial void AllocUser() // Allocate space
partial void FreeUser() // Free allocated space
```

NOTE: Adding an excessive amount of data to the frame will impact performance (de/serialization), as well as affect late joins.

Example

This is a very simple example which does not require manual memory allocation.

C#

```
using System;

namespace Quantum {
    unsafe partial class Frame    {
        public byte[] Grid => _grid;
        private byte[] _grid;

        partial void InitUser() {
            _grid = new byte[RuntimeConfig.GridSize];
        }

        partial void SerializeUser(FrameSerializer serializer)
        {
            serializer.Stream.SerializeArrayLength<Byte>(ref _gri
            for (int i = 0; i < Grid.Length; i++)
            {
                serializer.Stream.Serialize(ref Grid[i]);
            }
        }

        partial void CopyFromUser(Frame frame)
        {
    }
}
```

Back to top

We Make Multiplayer Simple

Products

Fusion
Quantum
Realtime
Chat
Voice
PUN

Memberships

Gaming Circle
Industries Circle

Documentation

Fusion
Quantum
Realtime
Chat
Voice
PUN
Bolt
Server
VR | AR | MR

Support

Gaming Circle
Industries Circle
Circle Discord
Circle Stack Overflow

Resources

Dashboard
Samples
SDK Downloads
Cloud Status

Connect

Public Discord
YouTube

Languages

English
日本語

photon

Blog

繁体中文

Contact Us

Terms Regulatory Privacy Policy Privacy Code of Conduct Cookie Settings