Fast-Paced Multiplayer (Part IV): Lag Compensation

Client-Server Game Architecture | Client-Side Prediction and Server Reconciliation | Entity Interpolation | Lag Compensation | Live Demo
Translations: Korean | Russian

Introduction

The previous three articles explained a client-server game architecture which can be summarized as follows:

- Server gets inputs from all the clients, with timestamps
- Server processes inputs and updates world status
- Server sends regular world snapshots to all clients
- Client sends input and simulates their effects locally
- Client get world updates and
  - Syncs predicted state to authoritative state
  - Interpolates known past states for other entities

From a player's point of view, this has two important consequences:

- Player sees **himself** in the **present**
- Player sees **other entities** in the **past**

This situation is generally fine, but it's quite problematic for very time- and space-sensitive events; for example, shooting your enemy in the head!

Lag Compensation

So you're aiming perfectly at the target's head with your sniper rifle. You shoot - it's a shot you can't miss.

But you miss.

Why does this happen?

Because of the client-server architecture explained before, you were aiming at where the enemy's head was 100ms *before* you shot - *not* when you shot!

In a way, it's like playing in an universe where the speed of light is really, really slow; you're aiming at the past position of your enemy, but he's long gone by the time you squeeze the trigger.

Fortunately, there's a relatively simple solution for this, which is also pleasant for *most* players *most* of the time (with the one exception discussed below).

Here's how it works:

- When you shoot, client sends this event to the server with full information: the exact timestamp of your shot, and the exact aim of the weapon.
- **Here's the crucial step**. Since the server gets all the input with timestamps, it can authoritatively reconstruct the world at any instant in the past. In particular, it can reconstruct the world exactly as it looked like to any client at any point in time.
- This means the server can know exactly what was on your weapon's sights the instant you shot. It was the *past* position of your enemy's head, but the server knows it was the position of his head in *your* present.
- The server processes the shot *at that point in time*, and updates the clients.

And everyone is happy!

The server is happy because he's the server. He's always happy.

You're happy because you were aiming at your enemy's head, shot, and got a rewarding headshot!

The enemy may be the only one not entirely happy. If he was standing still when he got shot, it's his fault, right? If he was moving… wow, you're a really awesome sniper.

But what if he was in an open position, got behind a wall, and *then* got shot, a fraction of a second later, when he thought he was safe?

Well, that can happen. That's the tradeoff you make. Because you shoot at him in the past, he may still be shot up to a few milliseconds after he took cover.

It is somewhat unfair, but it's the most agreeable solution for everyone involved. It would be much worse to miss an unmissable shot!

Conclusion

This ends my series on Fast-paced Multiplayer. This kind of thing is clearly tricky to get right, but with a clear conceptual understanding about what's going on, it's not exceedingly difficult.

Although the audience of these articles were game developers, it found another group of interested readers: gamers! From a gamer point of view it's also interesting to understand why some things happen the way they happen.

Further Reading

As clever as these techniques are, I can't claim any credit for them; these articles are just an easy to understand guide to some concepts I've learned from other sources, including articles and source code, and some experimentation.

The most relevant articles about this topic are What Every Programmer Needs to Know About Game Networking and Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization.

<< Part III: Entity Interpolation · **Live Demo >>**

Stay in touch! Your email: Keep me posted