Multiplayer Game Programming

Lecture 10: Consistency Despite Latency

ITP 484

Review

- When talking about consistency, what does it mean that "Causality" should be preserved?
- What is the commonly used, though not officially defined, definition of "network latency"?
- Give three reasons you'd experience end-to-end graphical latency (i.e. with no networking involved)
- List and explain the four delays a packet encounters travelling over the network.

Maintaining Consistency Despite Latency

- Conservative
  - Peer to Peer Lock Step
    - Buffered Turns
    - Packet Server
  - Dumb Client
    - With Interpolation
- Optimistic
  - Predictor / Corrector
    - Prediction Replay
    - Dead Reckoning / Extrapolation
    - Correction

Peer-To-Peer Lock Step

- DOOM / Age of Empires / Star Craft
- Device I/O Sharing!
- Requires deterministic simulation
- Why?
  - Useful when changed game state is just too big to replicate in a packet (RTS)

Peer-To-Peer Lock Step

- Game time is broken into Turns (say 35 ms)
- During a Turn, peer collects user input
- Turn input is sent to all other peers
- At start of next turn, wait until you have input from all peers for previous Turn
- Advance simulation and start collecting new input

Peer to Peer Lock Step

- Game must be deterministic
- Turn duration not always Frame duration
  - Why Useful?
  - Rendering can interpolate between simulation states for smoother experience
  - Different peers can render at different fps

Peer To Peer Lock Step

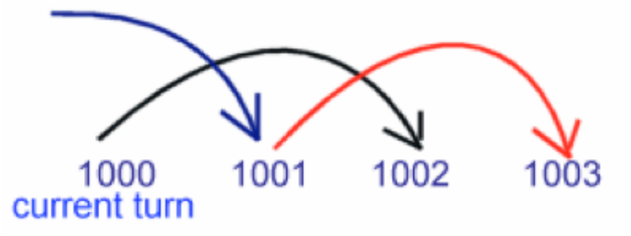
- What happens if a Peer doesn't have Turn input for another Peer when it needs it?
  - Nothing! Game has to wait…
  - So, simulation is limited by "slowest" peer
    - Limited by peer's CPU, running simulation and collecting input
    - Limited by peer's upstream latency, sending input

Lock Step with Buffered Turns

- When Input is collected during a given Turn, it is only used two (or more) Turns later.
- What does this mean?
  - One or more Turn packets can be in flight while input is being gathered and another Turn is being simulated.

Age Of Empires Example

- Turn is approximately 200 ms
- Turn Inputs is buffered to be used two turns later
  - Input collected Turn 1000 used on Turn 1002



  - 400 ms between Input and Simulation
  - Why good?
    - Laggy clients have 200 ms to get their turn data delivered without causing a stutter
    - In RTS, constant input latency of up to 500 ms not very noticeable, and much preferred to wildly varying input latency.

Age Of Empires Example

- Turn time can be adjusted
  - One dedicated peer can announce a turn on which turn time should change
    - If Turn Input arriving late, increase turn time!
      - Decreases chance of simulation freezing due to missing input
      - Increase latency
    - If Turn Input arriving early, decrease turn time!
      - Decreases latency

Lock Step with Packet Server

- Instead of Peer-to-Peer, Lock Step can work with Client Server
- Clients send own Turn to Server
- Server relays other Turns to Client
- If Server has not received a Client's Turn, it can resend previous Turn
- Advantages:
  - Prevents slow Client from freezing sim
  - Scales better (less upstream bandwidth per Client)
- Disadvantages
  - Increased latency, but hopefully hidden by Turn buffer
  - Less accurate! Less conservative. Usually not good enough for an RTS.

Dumb Client

- Client sends Input to Server
- Server sends snapshots of game state to Clients
- Lab 3!
- What is minimum delay between Client Input and Viewing effects of that input?
  - RTT!
  - Why might it be longer than that?
    - Snapshot sampling might not line up with time input is processed

Dumb Client With Interpolation

- Each snapshot has timestamp from server
- You know how long it is between snapshots, so take that long to interpolate between them

Z = 0

Z = 0.5

Z = 1

Packet:
Server Time = 0
Master Chief.Z = 0

Packet:
Server Time = 1
Master Chief.Z = 1

Client Time = 0

Client Time = 1

Client Time = 1.5

Client Time = 2

Dumb Client With Interpolation

- Delay enough so that you always have snapshots ready
- Increases smoothness
- Increases latency

Optimistic Algorithm

- Conservative Algorithms also known as Pessimistic
  - They assume the world is going to beat them down and thus there's nothing they can do about latency.
- Optimistic Algorithms
  - Have a better outlook!
  - Think that the world is a fair place and if they work hard enough, they can hide latency and everybody will like them

Client Side Prediction: For Player

- How to hide input-to-response latency?
  - Input takes time to get to server no matter what
  - To reduce latency, must process input on client
- For every simulation step on client
  - Sample Input
  - Apply Input to player controlled object on client
  - Send Input to server
- When server receives input for client
  - apply input to client's object on server

Client Side Prediction Example

Latency from client to server is 30 ms

Client ms	Client	Server ms	Server
10	Input.X = +1 Position.X = 1	10	
20	Input.X = +1 Position.X = 2	20	
30	Input.X = +1 Position.X = 3	30	
40	Input.X = +1 Position.X = 4	40	Input.X = 1 Position.X = 1
50	Input.X = +1 Position.X = 5	50	Input.X = 1 Position.X = 2
60	Input.X = +1 Position.X = 6	60	Input.X = 1 Position.X = 3
70	Input.X = +1 Position.X = 7	70	Input.X = 1 Position.X = 4
80	Input.X = +1 Position.X = 8	80	Input.X = 1 Position.X = 5
90	Input.X = +1 Position.X = 9	90	

Player Correction from Server

- When the server sends a player an update of her own state, what's wrong?
  - It's stale!

ms	Client		ms	Server
10	Input.X = +1 Position.X = 1	Input.X= 1	10	
20	Input.X = +1 Position.X = 2		20	
30	Input.X = +1 Position.X = 3		30	
40	Input.X = +1 Position.X = 4		40	Input.X = 1 Position.X = 1
50	Input.X = +1 Position.X = 5		50	Input.X = 1 Position.X = 2
60	Input.X = +1 Position.X = 6		60	Input.X = 1 Position.X = 3
70	Input.X = +1 Position.X = 7	Position.X = 1	70	Input.X = 1 Position.X = 4
80	Input.X = +1 Position.X = 8		80	Input.X = 1 Position.X = 5
90	Input.X = +1 Position.X = 9		90	

Player Correction from Server

- Stale Player updates from server
  - It left the server half an RTT earlier
  - But it's in response to player input a full RTT ago
- Teleporting to new state would be like moving backwards
- Solution?
  - Replay moves that server hasn't received yet

Move Replay

- Client samples Input and copies into a “Move”
- Client adds local Client Timestamp to Move
- Client saves Move in Move List
- Client sends Move with Client Timestamp to server
- When server responds with updated player state, it includes Client Timestamp of most recently processed Input
- When Client receives Player state it applies it to Player but then…
- Client examines Client Timestamp on state
  - Removes all Moves from Move List that are <= Timestamp
    - They have been processed by server and are incorporated in update
  - Reapplies all Moves from Move List that are later
    - This means rerunning multiple simulation steps for just the player

Move Replay Example

Client ms	Client	Server ms	Server
~~10~~	~~Input.X = +1 Position.X = 1~~	10	
20	Input.X = +1 Position.X = 2	20	
30	Input.X = +1 Position.X = 3	30	
40	Input.X = +1 Position.X = 4	40	Input.X = 1 Position.X = 1
50	Input.X = +1 Position.X = 5	50	Input.X = 1 Position.X = 2
60	Input.X = +1 Position.X = 6	60	Input.X = 1 Position.X = 3
70	Input.X = +1 Position.X = 7	70	Input.X = 1 Position.X = 4
80	Input.X = +1 Position.X = 8	80	Input.X = 1 Position.X = 5
90	Input.X = +1 Position.X = 9	90	

TS = 10ms
Input.X= 1

TS = 10 ms
Position.X = 1

+ 6 moves

Position.X = 7

Move Replay Optimization

- When Client sends Move, it can also send expected result state of Move
- Server can examine state and determine if client is correct, or if it needs an update
- If it's correct, it doesn't need to send an update of state
  - Needs to occasionally send OK to clear Moves out of Move List
- Trades use of client upstream for server upstream

Rest of the World out of sync

- Through Move Replay, Local Player on Client is one full RTT ahead of state from Server
- Any other state received from the Server (other players, bullets, moving platforms) is one RTT behind Local Player!
- Two options:
  - 1) Compensate when player tries to shoot anything
  - 2) Predict the rest of the world too

Dead Reckoning

- Extrapolation! "Keep doing what you're doing"
- Great for simulation of deterministic things like bullets
  - Okay for nondeterministic things like players
- Extrapolate each update from server to be in sync with Local Player
  - Since Local Player is ahead by RTT, extrapolate by RTT
  - Remember how to measure RTT?

Dead Reckoning

- Server and client must have same physics model for object
  - First order:

    $$P_1 = P_0 + Vt$$

  - Second order:

    $$P_1 = P_0 + Vt + \frac{1}{2}At^2$$

    - follows curves better
    - requires replicating Acceleration
  - Should run through normal physics sim
    - Collision with static geometry, etc.

Dead Reckoning Corrections

- When predicting nondeterministic entity, prediction can be incorrect
  - First order, whenever velocity changes
  - Second order, whenever acceleration changes
- How to correct?
  - Snap
    - Accurate and Ugly
  - Interpolation
    - For a set amount of time, less than time between state updates
      - Perform reckoning based on incorrect position
      - Perform reckoning based on correct, recently replicated position
      - Interpolate from incorrect based reckoning to correct based reckoning
    - Less accurate, less ugly
    - Entities might drift in odd directions
  - Planning
    - Adjust client side velocity, rotation, etc, to get back on course
    - Least accurate, least ugly

Dead Reckoning Problems

- Relies on accurate RTT estimate
  - Pwned by jitter
- Can't accurately predict nondeterministic things like other players
  - Can't Dead Reckon a bunny hopper!

Source Engine “Lag Compensation” (for Instant Hit Weapons)

- Valve wanted more accuracy than dead reckoning could provide
- Solution: Server Side Time Rewind!
  - Client side prediction of just local player
    - like normal client side prediction, with corrections form server
  - Interpolate replicated state of everything else nondeterministic (including other players)
    - Like Dumb Client Interpolation
    - Adds extra latency to state updates from server
  - Means the local player is out of sync with the rest of the players!
  - When server receives user command (“Shoot!”)
    - Server finds world states between which the client was interpolating
    - Server reads interpolation ratio which client encoded in command
    - Server uses interpolation ratio to interpolate between those states
    - Server rewinds other players to state they had exactly when command issued
    - Server executes command- does ray cast for hit
    - Boom Headshot

Lag Compensation Weaknesses

- Only rewinds players, not all physically simulated objects
  - Too CPU intensive
- Only works for Instant Hit weapons
  - Non instant hit weapons are acceptably lag sensitive in Source
- Bullets around corner effect
  - Design decision: Better than bulletproof players