An introduction on First-Person Shooter game architectures

Giuseppe Reina
Motivation

• Huge business
  – Rapidly evolving from small 2-20 players rooms to Massively Multiplayer Online (MMO) worlds
  – World of Warcraft
    • 11 million subscribers on Nov. 2011
    • 15 euro monthly subscription
    • 750 millions euro/year

• Massive Multiplayer Online Games are expensive to deploy
  – Need for expensive hardware resources
  – Demand for distributed game techniques to offload the dedicated machines

• First-Person Shooter games (FPS)
  – … or more in general Fast-paced games
  – Best selling games for console (Halo, Call of duty, Battlefield, etc…)
  – Most challenging to distribute
First-Person Shooter gameplay
How is this talk organized

• 1\textsuperscript{st} part (today) – Tech talk
  – Technical background on game engines
  – Multiplayer architectures
  – Latency compensation techniques

• 2\textsuperscript{nd} talk (next week – Dec 6\textsuperscript{th})
  – High scale distributed gaming
  – Peer-to-peer gaming challenges
  – Our contribution
What happens when you play a game?
The key concepts of a game engine

• The game state
  – Group of virtual objects (called entities)
  – They are monsters, avatars, weapons, NPCs
  – …but also bullets, explosions and other special effects!

• Command
  – Represents the user intention in controlling one or more entities
  – A command, when executed, modifies the game state

• The simulation
  – Natural evolution of the game entities according to the physics engine, the collision detection system and the AI

• Game loop
  – Keeps everything together!
  – Cycles of the loop are called ticks
  – The tick rate determines the pace of the game
    • ~60 Hz in FPS (1 tick every 15 ms)
Game loop (local game)
CASE SCENARIO: SUPER MARIO BROS
Player command fetching

Input state

Command

"Jump & Shoot"
Player command execution

Mario.velocity.up = JUMP_SPEED;
<<new Fireball>>;
Firewall.position = Mario.position;
Fireball.velocity = FIREBALL_CONST_VEL;
Simulation execution

```java
// AI computation
KoopaTurtle.artificialIntelligence.step();

// Physics simulation
Mario.position += Mario.velocity;
Mario.velocity -= GRAVITY;
Fireball.position += Fireball.velocity;
KoopaTurtle.position += KoopaTurtle.velocity;

// Collision detection
detectCollisions(Mario);
detectCollisions(Fireball);
detectCollisions(KoopaTurtle);
```
Graphics and Audio Rendering

Player command fetching

Graphics and Audio Rendering

Player Command Execution

Simulation (Physics, collision and AI)
Toward multiplayer games

PEER-TO-PEER ARCHITECTURE
Peer-to-peer: Lockstep algorithm

Peer 1
Waiting CMD4

Peer 2
Waiting CMD1 & CMD4

Peer 3
Executing Commands

Peer 4
Waiting ACKs
Lockstep algorithm (contd.)

• Pro
  – Easy to implement
  – Works with big number of entities
  – Robust to cheating
  – Peer-to-peer

• Cons
  – Consistency
    • Hard to ensure that a game is completely deterministic
  – Latency
    • Each player in the game has latency equal to the most lagged player
  – Hard to support late join
  – 1-to-N communication

• The Doom case
  – Very poor performances!
CLIENT/SERVER ARCHITECTURE
Client/Server: The dumb client

Client Side

Server

Server Side

Simulation (Physics, collision and AI)

Player Command Execution

Frame

Obj1
Obj2
Obj3...

CMD1

CMD2

CMD3

CMD4

Graphics and Audio Rendering

Player Command Fetching

Client

Client

Client

Client
The dumb client (contd.)

• **Pro**
  – Single authoritative machine
  – Tolerant to cheating
  – 1-on-1 communication

• **Cons**
  – Scalability
  – Robustness
  – Does not work with big number of entities
Latency (or Lag)

- **Effects of lag**
  - Introduce inconsistencies
  - Consistency vs. Reactiveness
  - Affects fairness
  - Reduce game experience
  - One of the biggest causes of game failures

- **User tolerance in FPS**
  - 50 to 100ms for *direct manipulation tasks*
  - 100 to 150ms for *indirect control tasks*
  - Similar results have been shown for other non-FPGs (RTS games, Sport games and so on.)
Common issue: Fireproof player

Shooter (Player\textsubscript{A})

Target (Player\textsubscript{B})

Client\textsubscript{A}

Server

Client\textsubscript{B}
Common issue: Shoot-round corners

Shooter (Player_A)  Target (Player_B)

Client_A            Server                Client_B
Optimization and compensation techniques

• Real time illusion
  – Entity interpolation
  – Client prediction

• Fairness
  – Lag compensation

• Bandwidth optimization
  – Delta encoding
  – Area of Interest
Client prediction

• The problem
  – Unacceptable to wait for message exchange between the client and the server
  – Need for immediate feedback for direct manipulation task

• The solution
  – Decoupling the simulation!
  – The client can simulate some state locally because the server will necessarily come up with the same result
  – Independent from the server simulation
  – Gives the feeling of immediate control
Client prediction (contd.)

P₀ to P₁

P₁ to P₂

P₂ to P₃

P₃ to P₄

Move?
P₀ to P₁

Move?
P₁ to P₂

Move?
P₂ to P₃

Move?
P₃ to P₄

Move
P₀ to P₁

Move
P₁ to P₂

Move
P₂ to P₃

Move
P₃ to P₄

Clientₐ

Server
Entity Interpolation

• The problem
  – Cope with choppy and jittery entity animation of a moving object

• The solution
  – Go back in time for rendering to interpolate between two different state of the game entity
  – Drawback: it increase latency
Lag compensation

• The problem
  – How to make sure that aiming at a player will actually give you the possibility to kill him?

• The solution
  – Keep a history of all recent player positions for a fixed amount of time
  – Move back in time the player’s avatar to check eventual collisions
Bandwidth Optimization

• The problem
  – There are on average 200 active entities on a 8 player FPS match
  – Pack them into a server frame is extremely expensive.

• The solution
  – *Area of Interest* and local perception filters
    • Define an area of interest of an avatar (everything the avatar can see or hear)
    • Send object updates *only* for the objects inside the area of interest
    • Use perception filters to set the update rate
  – *Delta Encoding*
    • Ship only the fields that changed from the last acknowledged update
Conclusion

• Overview of how a game engine works
• Multiplayer architectures techniques
• Latency optimizations for games

• Next talk
  – Scalability issues and distributed architectures
  – Peer-to-peer gaming challenges
  – Our contribution
Questions
THANK YOU FOR YOUR ATTENTION