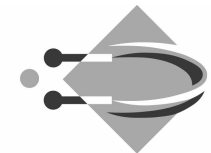
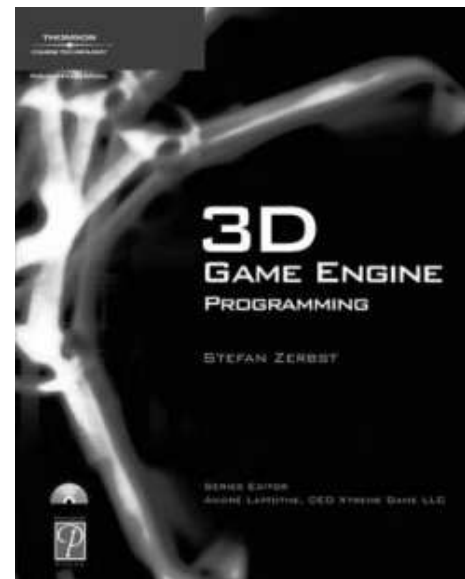
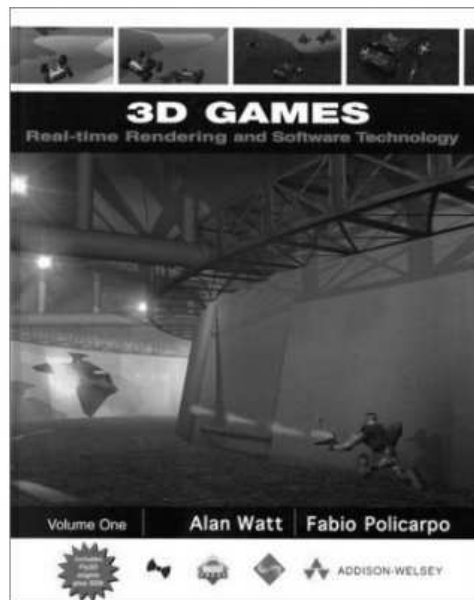


Network games

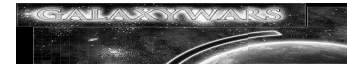
Multi-player game technology



Different requirements for different types

Session-based Games

- No persistent data is stored (except simple high scores)
- Each player makes a move or turn (Chess, Cards).
- All turns can be made in the same time (first-person shooter)



Death matches (Quake, Unreal)

Cooperative games (Battlefield 1942)



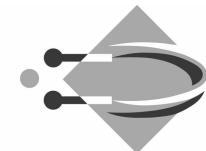
Persistent virtual worlds

- Persistent data (attributes, health, equipment)

Leveling

Massive Multiplayer Online Role-playing Games

(multiple instances or multiple continents)



The **IT** University
of Copenhagen

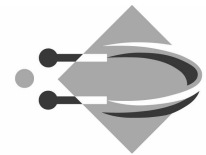
Common Factors in Multi-player game technology

How much information is sent to each player and what is the bandwidth

How many players
(System often don't scale linearly)

Required reliability

Network architecture



The **IT** University
of Copenhagen

Terminology

- Packet : The indivisible unit of information send across the internet which is obviously not a dedicated channel like a fax. Packets can be lost!

- Protocol : Data format standard, Realiability and delays are inversely proportional
Transmission Control Protocol, guaranties message
User Datagram Protocol, no resending of lost message

-Lag or latency:

Time delay between the message is sent by the source and received at a destination.

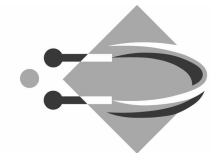
The mother of most multi-player game problems.

LAN(~1 ms), WAN(~100ms), Light US & EU (~10ms)

Variations in latency is a killer of immersiveness.

- Bandwidth: Maximum rate of transport of data through a channel.

- Network architecture: Peer-to-Peer or Client/server

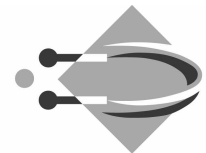


The IT University
of Copenhagen

Simple framework.

1. Receive message from all other players, updating their status
2. Process these messages with game logic
3. Send messages updating my status to all others
4. Render the frame and loop.

One and three can involve unpredictable delays.



Network architecture

Client/server

Client: Join/start, receive app. or level, play, exit

Server: Seperate app.

Server collects messages, issues acknow.

Server applies cmd, issues acknow.

On receipt acknow. , client applies cmd

Peer-to-Peer

No server app. , each player has entire game.

Simpler : send, receive and process.

Delays grow quadratically (unless broadcast is used)



Delays

Game cycle: Read input, Move objects, Render.

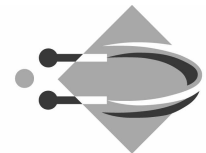
Input arrives on average halfway through rendering. On a single machine the average delay time is then $1.5c$.

On a client/server:

1. Input $0.5 c$ Influence lag
2. Send message $0.5 s$ Influence lag
3. Server cycle $1 s$
4. Server sends message $0.5 c$
5. Client see effect $1.0c$

Total = $2 c + 1.5 s$ ($c=s=50ms$, total = $175ms$ or $6-7fps$)

On top of this comes network delays.



Semantics

Delays, Message loss and Reordering.

Positional update delays

You shoot someone on the local client. On opponent client your firing is delayed and his position was not the position you saw him at.

Peer-to-peer : "I am hit" instead of "you are dead".

Client/server : Server resolves the problem

Synchronous views are not necessarily a critical requirement.

Acquisition of objects: Two players go for the same object.

Client/server : Server resolves

Peer-to-peer : request is send on availability of the object.

Smoothness of play

Client/Server: Fast connection to server.

Peer-to-Peer: Play is smooth but other players can jump in case of delays.

Packet loss (UDP or TCP)

Some messages are mandatory, some are not.

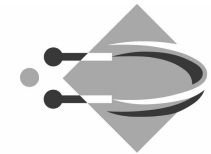
Position is not mandatory.

"I am dead" is mandatory and needs confirmation.

Message reordering

Visual effects for instance in case of position reordering.

Other cases might cause application crash.



Reducing the amount of information

Data compression

Send the difference between values. Errors accumulate

Send commands only: Collect commands from all players and execute locally. Such a "player delay" is often not possible but resolves a lot of problems.

Relevance filtering/interest | data distribution management

Participants receive only relevant information.

Range of interest can be dynamic depending on context.

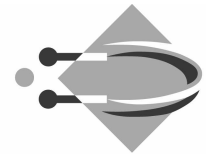
Analog to collision detection.

Grid based: messages are send to participants nearby.

Object based: Relevance test is made on behalf of the object.

Players have an aura. Communication is established if aura intersect.

Data is divided into static, possible outdated dynamic data and always up to date data.



Predictive methods

Dead reckoning is the process of estimating a global position of a vehicle by advancing a known position using course, speed, time and distance to be traveled. In other words figuring out where you momentarily are or where you will be at a certain time if you hold the speed, time and course you plan to travel.

Player controls own objects. Other objects (controlled by other players) called ghosts are moved using dead reckoning.

Player A makes dead reckoning from last send message. If predicted position/velocity diverges too much from the actually then a new message is send.

Gives trade-off between number of messages and accuracy.

Only players are treated by the dead reckoning method.

Changes to the enviroment must be send.

