Improving Performance of Networked Games

Ali Anvari

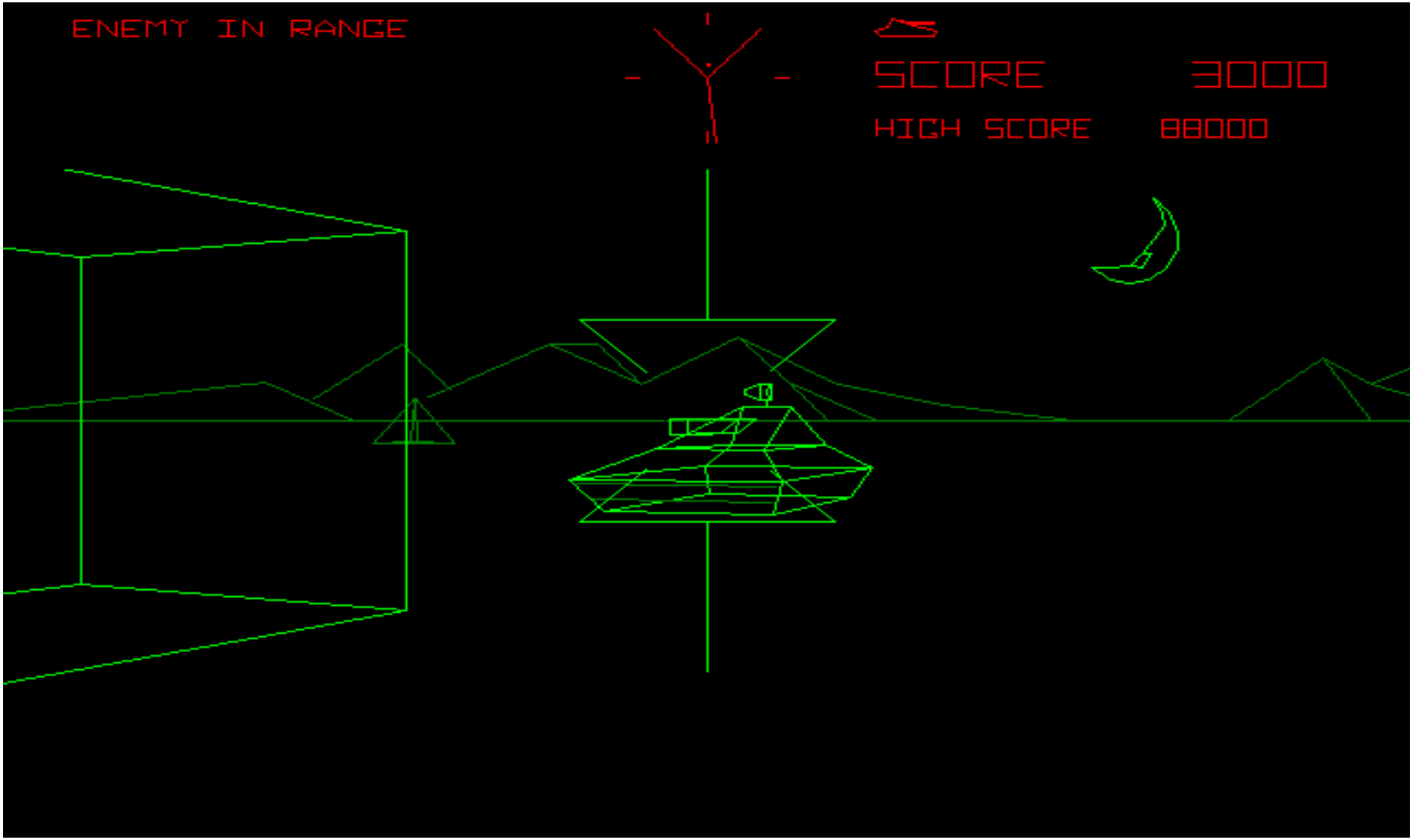
Electronic Engineering Department

University College London

9 July 2010

ali.anvari@gmail.com

Example of a game: Tank Deathmatch



"Battlezone" (Atari), fetched from Photobucket.com (benchilada)

Overview of Games

Game state (tank positions, armour rating, ammunition)

Feedback of game state to player (camera position)

Updates of state by players (push joysticks, drive tanks)

Processing of interactions (collisions, shooting)

Transient state (bullets)

Long-lived state (walls, mountains, moon)

Bandwidth needed grows with # of players

Toeach of n game clients *player*

    Send data (graphics, sound) about *target*

        Foreach of N game objects *target*

So the game design is not strictly networking,

yet still impacts the network

Too many objects could overwhelm client bandwidth

LAN games exception handling for load: State of the art: The server falls over

Delays could cause clients to freeze up

So every client with consistently slow latency is ejected

Clients lacking bandwidth get backlogged packets

Backlog -> Lag -> Ejection by server

LAN servers dimensioned for 128 players +/- factor of 2 (Donnybrook claims 900)

Try to solve with Interest Management

Don’t need to process game entities too far away to interact with (tanks out of range)

Don’t need to process game entities too far away to observe (tanks disappear into dots on horizon)

Don’t need to track changes of game entities that don’t change (mountains, wall, moon constant, indestructible)

Geographic Interest Management.

Problems moving from LAN to Internet

Latency goes up

<10 ms to next room, >200 ms to China

Bandwidth goes down

Gbps ethernet vs 256 kbps broadband

Contention goes up

8 port switch vs 50:1 broadband connection

<40 kbps per user

(“Developing Online Games” Patrovsky & Mulligan)

Player numbers go up

<100 in living room, >1 million (World of Warcraft)

Review of Bandwidth Optimization: Geographic Interest Management

Send data to client

~~Foreach game object *target*~~

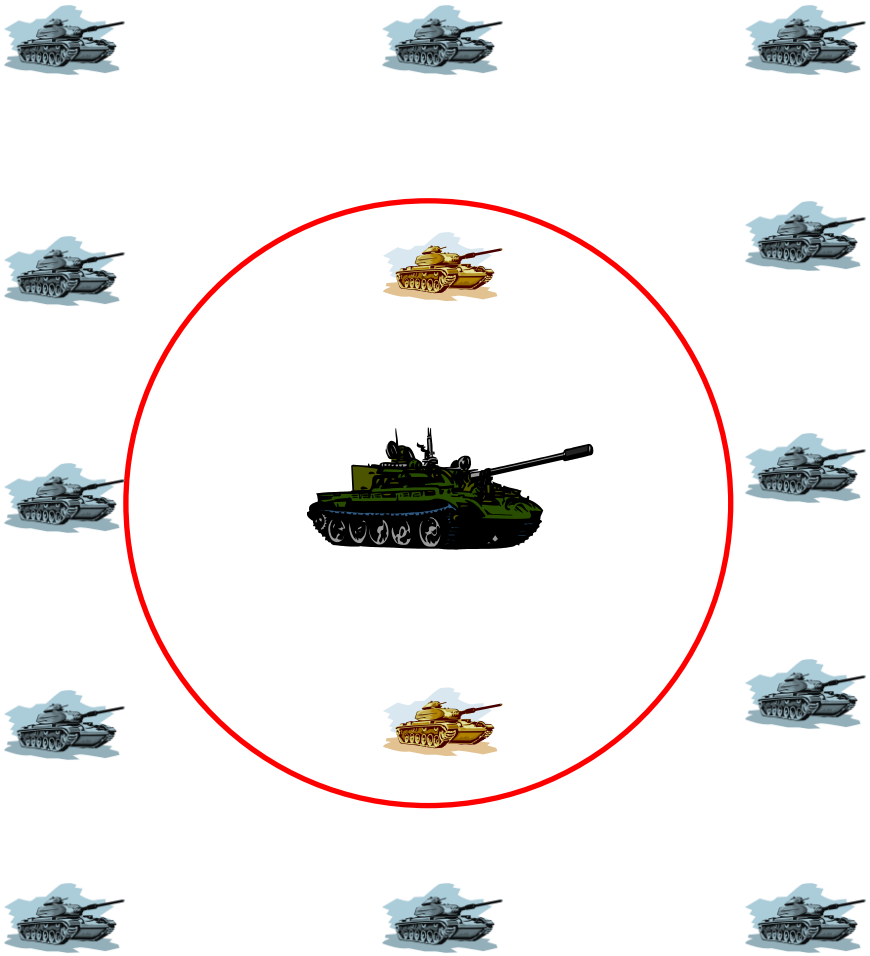
~~Foreach game object that moves *otherplayer*~~

~~Foreach game object in same room *roommate*~~

Foreach game object in area of interest *neighbour*

Area Of Interest:

Theory



Practice

Sharding:
(In case Area Of Interest fails)

Slice game up or make “parallel universe” replicas

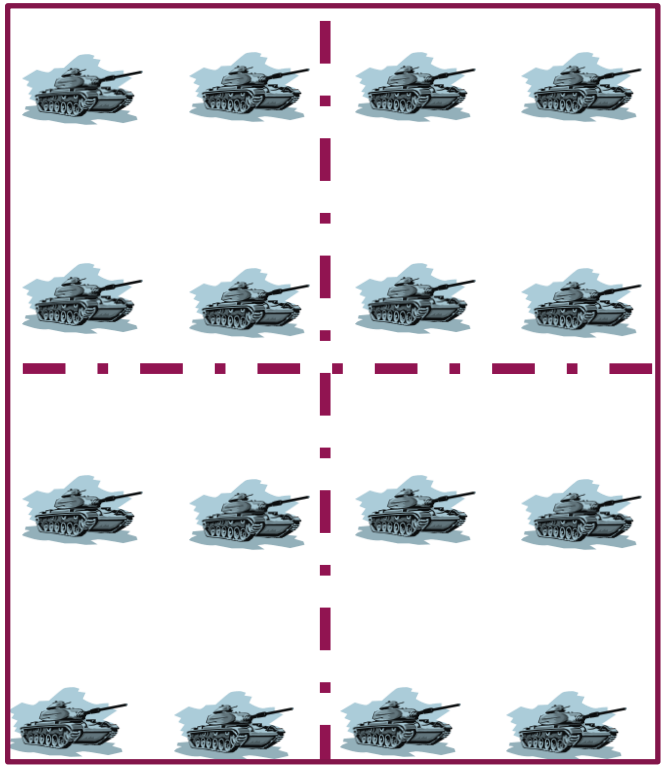
Put each slice on a different server

Fix map data as constant for each slice

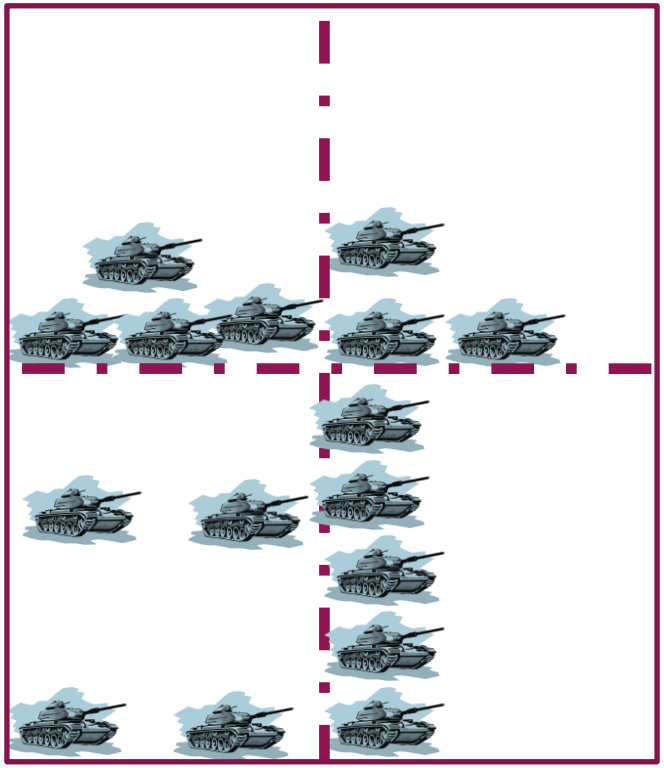
Allocate a certain number of players to each slice

Sharding

Theory



Practice



Players have reason to gather in same area (So reduce Sharding, it ruins experience)

"Return of the Jedi" (20th Century Fox), screengrab

Density Based Interest Management: VELVET proposal (de Oliveira)

Send data to client

Foreach of k nearest neighbours *NN*

Implemented by changing size of area of interest

Increase area of interest for clients with more bandwidth,
Decrease area of interest for clients with less bandwidth

Increase area of interest for zones with few players,
Decrease area of interest for zones with many players

Peer to peer to avoid server bottlenecks

Unanswered Questions

If the bottleneck is at the clients (40 kbps), why dump server workload on them by using P2P?

- Will not consider further

How to calibrate area of interest for client bandwidth?

How to calibrate area of interest for player density?

N^2 problem to calculate distances, densities

Standard solution is Spatial Indexing (NLogN),

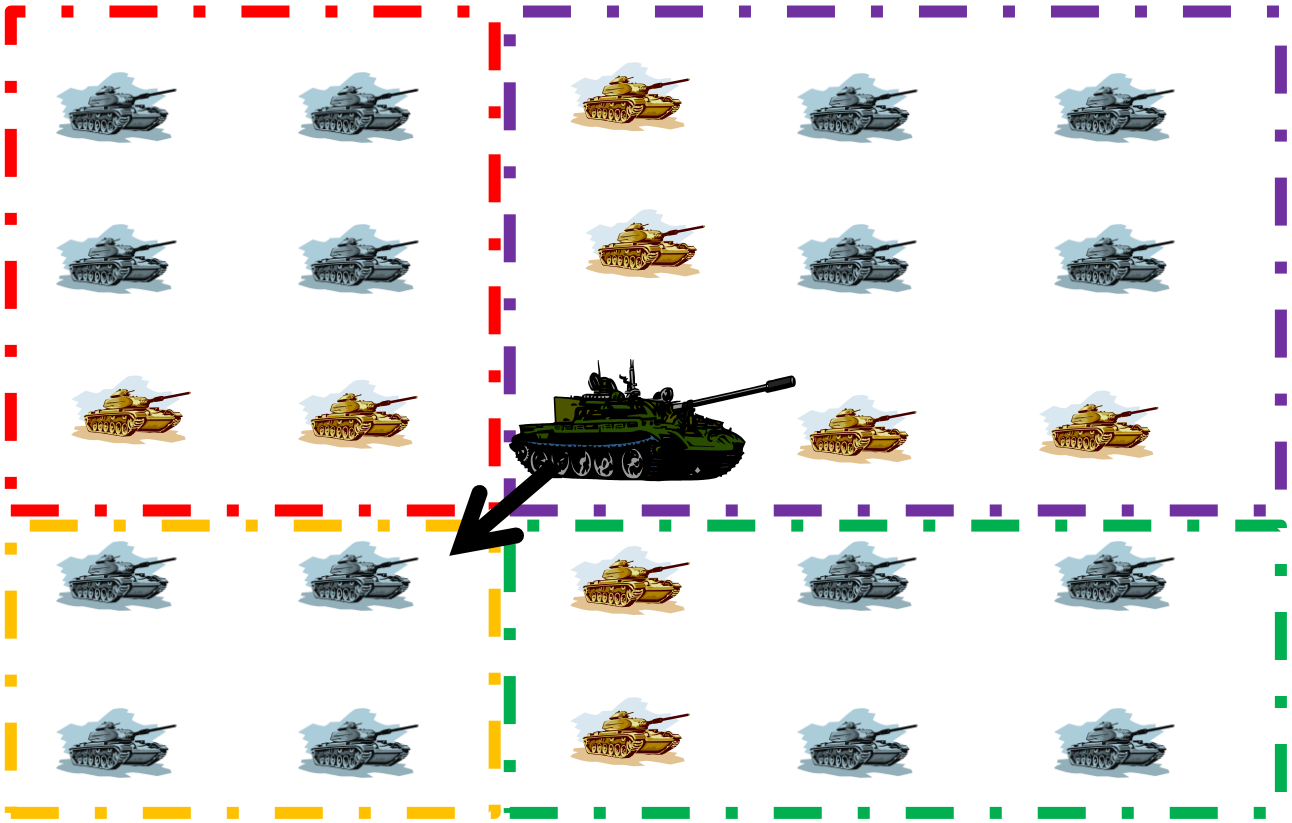
But maintaining the Spatial Index is expensive

As player positions change, Index must be rebalanced

Rebalancing cost can be high (millions of users)

Spatial Index is a global bottleneck for central server

Spatial Indexes based on Space Partitioning



Rebalance when objects cross partition boundaries

For more details, see literature on BSP Trees, RTrees

Spatial Indexing State Of The Art

EVE online

Only 2 shards

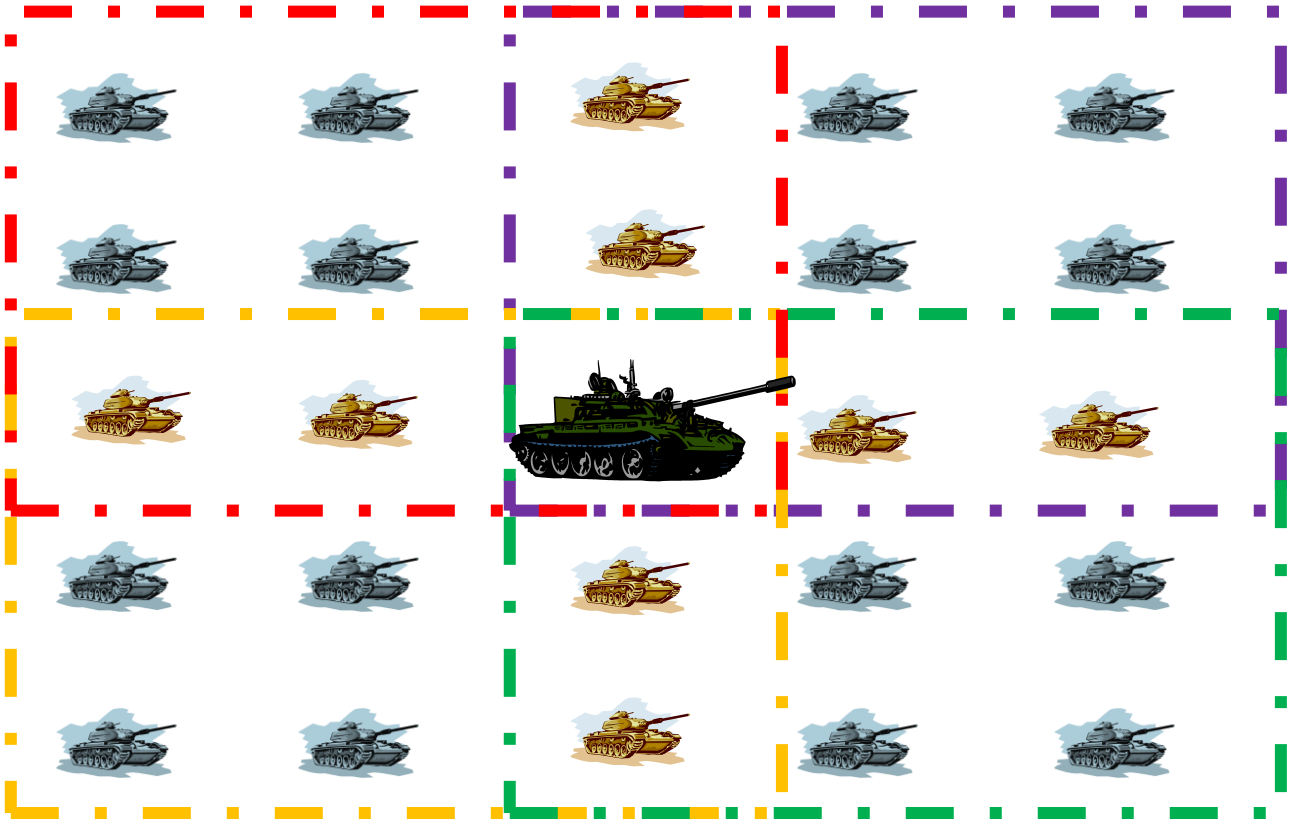
Up to 700 players per battle

60,000 peak users in a shard

Far smaller than World of Warcraft

200 node supercomputer with high speed interconnect, solid state drives

Seamless Servers (Beardsley)



Need to keep entries for objects in multiple partitions

But: Less rebalancing needed?

Problems with bandwidth adaptation

Standard method is congestion control as in TCP
- Additive Increase, Multiplicative Decrease
- Feedback (ACKs) based on Round Trip Time

Games run at high frequency
- Upward of 20 frames per second
- Less than100 ms for each transmission or game fails
- Congestion control must finish in this time

Internet servers have high latency
- Server processing, ISP peering, Speed of light
- More than100 ms before Feedback arrives
- Not enough time for standard congestion control

How to make bandwidth adaptation work in real time? (Suggestions Welcome!)

Network measurements of access network from ISP?

Game server bandwidth control to be run in ISP network?

In local exchange?

DiffServ?

LEDBAT?

Congestion control if low latency, else assume worst case?

Summary

Area Of Interest used to reduce bandwidth, CPU load in LAN games. Overwhelmed at Internet Scale (especially by Flash Mobs)

It has been proposed to scale AOI based on player Density, client Internet Bandwidth.

Open Questions:

1. How to calculate player density in real time?
2. How to adapt bandwidth usage in real time?

Questions? Suggestions?

“Return of the Jedi” (20th Century Fox), screengrab