Client side prediction methods in networked multiplayer first person shooter games

Richard Steele

***Abstract*—This paper critically examines the effectiveness of the current methods of client side prediction in networked multiplayer digital games, specifically first person shooter games where the accuracy of each game agent's behaviour is critical to the player's experience. The de facto use of *dead reckoning* is accepted as a standard tool to help minimise the number of required network updates, but its results can be error prone which are a detriment to the player's experience. This paper looks at the currently used dead reckoning algorithms and suggests using further data about the game state to provide a more accurate estimation of a game agent's location and behaviour at a future time. Knowledge about the layout of a level is used to dynamically alter the threshold of the dead reckoning algorithm's allowance to better reflect an agent's behaviour at critical points of the game by sending an increased amount of player updates.**

I. Introduction

With modern networked multiplayer digital games where the game state must be updated often, it is impractical to update each client's representation of the game state every frame. Instead, *dead reckoning* algorithms are used to predict future states of the game, and network updates are needed only when the actual behaviour differs from the predicted behaviour by a preset threshold. While effective at greatly reducing the required network updates and reducing the impact of network transmission delay [**?**], impossible behaviours such as tunnelling, or improbable behaviours counter intuitive to the game are possible. This paper proposes that by using further information about the game state, such as level layout, an agent's behaviour can be predicted with greater accuracy to help minimise unwanted behaviour.

Previous papers in the context of games describe dead reckoning as a measure to account for network latency compensation. Few expand beyond this and are generally concerned with how best to marry the predicted state with the actual state by interpolation or rolling back. This paper explores improving the accuracy of the simulation by dynamically adjusting the threshold allowed by traditional dead reckoning methods to prompt more frequent state updates at critical times where the greatest accuracy is required.

The research question that this paper addresses is: can client side prediction in networked multiplayer first-person shooter games that use dead reckoning be improved in accuracy by using player modelling and information about the game's state to more reliably predict a player's actions?

II. Background

This literature review first confirms the definition of client side prediction, the problems that it solves, and the trade off against absolute accuracy. Dead reckoning is explained and explored next with a focus on the real world applications and techniques of its use within games. Finally, alternative methods of client side prediction are explained and acknowledged.

A. Client side prediction

Networked multiplayer games are defined as sharing a game space between multiple clients [**?**]. Each client holds a representation of that game space which it uses to display information to the user, and allows the user to change within the rules of the game. When elements or details of the agents of that game space are changed by a client's actions, the client assumes that the action is successful, performs that action on its own game state representation, then sends the action information to the network so as all other clients can update their own game space representation to reflect the performed action [**?**].

Problems can occur when:

- the updated information is received too late (network latency)
- the updated information is not received (packet loss)

Network latency forms the primary bottleneck for online game performance with an acceptable value being under 100 milliseconds [**?**] and worst case overall latencies claimed to be up over 1 second [**?**]. Performance often also degrades significantly under variance in latency (or *Jitter*) [**?**] [**?**]. To counter this, each client uses dead reckoning to predict the behaviour of each agent, and to project forward the game state by this time amount to maintain a consistency of all entities states throughout. The jitter must be kept account of as this time amount can be subject to change.

Given the vast quantity and unreliable nature [**?**] of the messages being sent, packet loss must be planned for. The same dead reckoning methods used to account for network latency and outlined in *B.* can be used to replace missing packets of information, or application data units (ADUs) [**?**], to supply a prediction of where the ADU's agent will "most probably" be.

B. Dead reckoning

Dead reckoning methods are based on a navigational technique of estimating one's position based on a known starting point and velocity [**?**]. In the simplest implementation, we take the last position we received on the network, and project it forward in time with at its last known velocity [**?**].

The player tracks both its actual position and the predicted position calculated with dead reckoning. An updated Entity State PDU is sent out on the network when the two postures

differ by a predertimined error threshold, or when a fixed amount of time has passed since the last update (nominally 5 seconds) [**?**]

The controller of an entity regularly checks whether the difference between the prediction and the actual state exceeds a certain threshold. If this is the case the controller of the entity transmits the state so that other applications may learn about the correct new state of the entity [**?**]

C. What else can we do?

area of interest filtering only sends a client information for entities that are within that player's potential field of sight [**?**]

In Quake the last three commands are also sent in each packet to compensate for lost packets [**?**]

Approaches which introduce a local presentation delay, e.g., as that used within MiMaze [**?**] have been developed to provide for a consistent, distributed state. However, these increase the application-level delay even more.

Buckets or late updates may lead to repairing inconsistencies requiring a rollback which will cause more drastic position jumps [**?**]

III. Methodology

Even if so, network latency would cause the received data to be slightly behind,

In the case of lost packets resulting in no update data for that agent. Instead, dead reckoning methods are used to simulate an agent's behaviour, allowing client-side prediction to reduce the impact of a network transmission delay [**?**]. When used effectively this can help to synchronise the game state across all clients.

multiplayer games where the game state is updated often, the effects of late or lost packets are minimised with the use of a simple dead reckoning algorithm. This can smooth trajectories between state updates, and also decrease the frequency of state transmission.

A. citations

allows to prolong the interval of message transmissions and abolish the network latency at the cost of data consistency [**?**]

Globally synchronised clocks [**?**]

Leading a shooting mechanic to account for lag [**?**] Having an authoritative server means that even if the client simulates different results than the server, the server's results will eventually correct the client's incorrect simulation [**?**] player's can turn instantaneously and apply unrealistic forces to create huge accelerations at arbitrary angles and you'll see that the extrapolation is quite often incorrect [**?**]

DR is a distributed approach that does not require a centralised server. Using a distributed approach is mandatory for many distributed virtual environments (DVEs) in order to avoid the well known problems of centralised systems such as increased latency, single-point-of-failure and lack of scalability [**?**]

Previous synchronisation mechanisms such as bucket synchronisation and Time Warp are not well suited to the demands of a real-time multiplayer game. Trailing state synchronisation is designed specifically for real-time multiplayer games, and achieves better responsiveness and scalability while maintaining near-perfect consistency [**?**]

B. Next

Describe state of field in relation to literature Define the problem, grey lit for dead man can shoot gamasutra etc find papers and authors that support the position I'm taken. Commercial context - why important to have good solutions of the problem - see reviews? Metacritic Context for problem space A lot of fps have problems because of network problems, find reviews, quote popularity. See criticism.spread the assumptions across white and grey. Pull in grey lit and peer review. compare the milit/games.

practical motivation

Academic motivation

Gaps in literature Dead reckoning algorithms only use information on the player's current and simulated location and velocity.

tunnelling problem co cognition bots but intent needs to be carried across

what is dr

why have dr

how when where

What is good about it What problems does it solve strengths

What is bad about it weaknesses is it a problem

propose solution

C. Lit review

D. Methodology

IV. Conclusion

The conclusion goes here.

Appendix A
First appendix

Appendices are optional. Delete or comment out this part if you do not need them.