Extending Client Side Prediction Methods in Networked Multiplayer First Person Shooter Games to Include Level Knowledge

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 send and receive network updates on 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 threshold amount. This idea of an acceptance threshold accepts that the perception of an entity for all other clients on a network may differ to the actual state of that entity. While effective at greatly reducing the required network updates and reducing the impact of network transmission delay [1], impossible behaviours such as tunnelling (projecting a position past a barrier), 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 concerned generally 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 when the greatest accuracy is required.

It is important to note that not all networked multiplayer games suffer from client side prediction problems. For instance turn based games consider each client sequentially. Where there is no scope or benefit for a player to react immediately, no measures must be taken to account for problems incurred by some milliseconds lost in transmission. These issues are prevalent in fast paced competitive games such as racing games, or the first person shooter genre. This paper then bases its study upon the first person shooter game genre, sometimes referred to as *twitch* shooters. The need to respond quickly to other clients' perceived actions is critical to the player experience, therefore it is crucial that those actions are shown as accurately as possible.

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 solving the the problems associated with network latency are explained and acknowledged.

A. Client side prediction

Networked multiplayer games are defined as sharing a game space between multiple clients [2]. 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 [3].

Problems can occur when:

- the updated information is received too late (network latency)
- the updated information is not received (packet loss)
- the action is not successful (simultaneous conflicting actions are performed by many clients).

Network latency forms the primary bottleneck for online game performance with an acceptable value being under 100 milliseconds [4] and worst case overall latencies claimed to

be up over 1 second [5]. To counter this, each client takes the slightly old network state update that it receives and brings it approximately up to date through extrapolation before displaying it to the user. Performance degrades significantly under variance in latency (or *Jitter*) [6] [7]. The jitter effect must be continually accounted for as the delta time value between updates can be subject to change. As such, the current time in which an action is performed is included with the update packet of data sent and received by clients, and forms an integral part of the prediction algorithm. The latency time difference must be continuously evaluated by sending and receiving time stamped packets to maintain a correct delta time value [8].

Given the vast quantity and unreliable nature of the messages being sent [9], 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) [2] sometimes referred to as protocol data units (PDUs) [10], to supply a prediction of where the ADU's agent will "most probably" be at a given time.

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 [4]. 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 [11].

Fig. 1. Image sourced from [11]

This simple representation of projecting forward is correct if an agent has linear movement and constant velocity. However, agents in a first person shooter game often have complex non linear movements which will need to be checked for and accommodated. Real world applications of linear dead reckoning also suffer with having to account for unpredictable behaviour e.g. drift due to wind or wheel slippage [12] [13]. The techniques used for managing these behaviours are transferable to the problems of dead reckoning networked multiplayer game agents with complex behaviour.

In 1983, with substantial support from the U.S. army, the Defense Advanced Research Projects Agency (DARPA) initiated the Simulator Networking (SIMNET) program which developed dead reckoning techniques to support a virtual world to train soldiers [14]. Their approach to networking solutions has been incorporated into the Distributed Interactive Simulation (DIS) standard which is now an IEEE standard [10]. It outlines the use of a PDU to help dead reckoning handle complex behaviour [15].

Each client maintains a dead reckoned model of itself that corresponds to the model used by other clients to "see" its represented agent. The client must regularly check whether the difference between the predicted state calculated with dead reckoning that all clients share and the actual state exceeds a certain threshold. If this is the case, the client is then responsible to generate an updated entity state for the dead reckoning algorithm to apply to the model. Then send that updated state to all clients on the network to learn about the correct new state of the entity to then use to update their own model of that client's agent [14] [16]. This also happens when a fixed amount of time has passed since the last update (normally 5 seconds) [17].

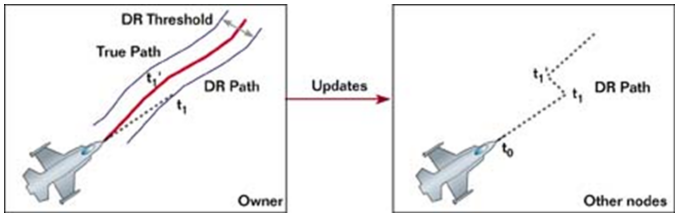

Fig. 2. Image sourced from [18]

Were this updated information used immediately, the modelled entity would appear to jump (fig. 2). Instead, the dead reckoning algorithm uses interpolation and projective velocity blending [11] to tend towards and eventually reconcile it's own dead reckoned model with the newly updated state information (fig. 3, fig. 4).

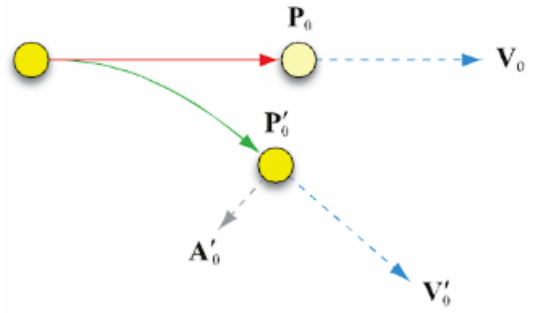

Fig. 3. Image sourced from [11]

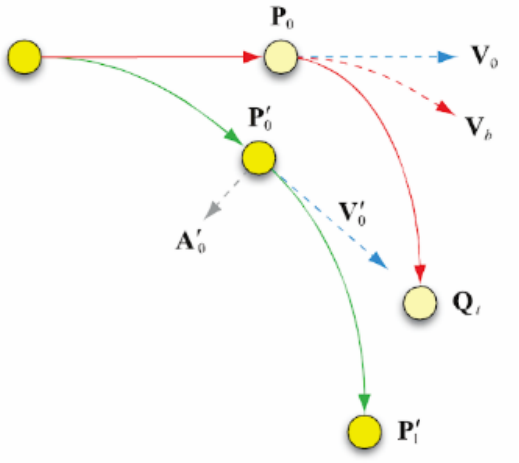

Fig. 4. Image sourced from [11]

$$Q_t = P_0^{'} + V_0^{'}T + \frac{1}{2}A_0^{'}T^2 \tag{1}$$

$$\begin{aligned} V_b &= V_0 + (V_0^{'} - V_0)\hat{T} \\ P_t &= P_0 + V_b T_b + \frac{1}{2}A_0^{'}T_t^2 \\ P_t^{'} &= P_0^{'} + V_0^{'}T_t + \frac{1}{2}A_0^{'}T_t^2 \\ Q_t &= P_t + (P_t^{'} - P_t)\hat{T} \end{aligned} \tag{2}$$

[11]

1 Linear dead reckoning as shown in fig 1. $P_0^{'}$ is the initial position of the entity. $V_0^{'}$ is its velocity. $A_0^{'}$ is its acceleration. Q_t is its dead reckoned position at time T.

2 Projective velocity blending as shown in fig 3 and fig 4

It is apparent that the blending may result in compromising behaviour. Demonstrated in first person shooter games as appearing to other players in a vulnerable position outside of cover, or moving through an impenatrable area as defined by the game's movement rules.

Dead reckoning is an *optimistic algorithm*. It assumes it is correct and when it is not, it must adjust itself.

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 [9]

In Quake the last three commands are also sent in each packet to compensate for lost packets [9]

Approaches which introduce a local presentation delay, e.g., as that used within MiMaze [19] 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 [20]

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 [1]. 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 [4]

Globally synchronised clocks [21]

Leading a shooting mechanic to account for lag [3] 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 [3] 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 [3]

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 [16]

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 [9]

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 taking.

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. [22] [23]

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

References

[1] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Proceedings of the 1st workshop on Network and system support for games*. ACM, 2002, pp. 79–84.

[2] C. Diot and L. Gautier, "A distributed architecture for multiplayer interactive applications on the internet," *IEEE network*, vol. 13, no. 4, pp. 6–15, 1999.

[3] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Game Developers Conference*, vol. 98033, no. 425, 2001.

[4] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," *The Electronic Library*, vol. 20, no. 2, pp. 87–97, 2002.

[5] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[6] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003®," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 144–151.

[7] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–7.

[8] J. Glazer and S. Madhav, *Multiplayer game programming: Architecting networked games*. Addison-Wesley Professional, 2015.

[9] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," in *University of Michigan*. Citeseer, 2001, pp. 3–6.

[10] D. S. Committee *et al.*, "Ieee standard for distributed interactive simulation-application protocols," *IEEE Standard*, vol. 1278, pp. 1–52, 1998.

[11] C. Murphy, "Believable dead reckoning for networked games," *Game engine gems*, vol. 2, pp. 307–313, 2011.

[12] H. Chung, L. Ojeda, and J. Borenstein, "Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 1, pp. 80–84, 2001.

[13] L. Ojeda, G. Reina, and J. Borenstein, "Experimental results from flexnav: An expert rule-based dead-reckoning system for mars rovers," in *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, vol. 2. IEEE, 2004, pp. 816–825.

[14] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The simnet virtual world architecture," in *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*. IEEE, 1993, pp. 450–455.

[15] W. D. McCarty, S. Sheasby, P. Amburn, M. R. Stytz, and C. Switzer, "A virtual cockpit for a distributed interactive simulation," *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 49–54, 1994.

[16] M. Mauve, "How to keep a dead man from shooting," in *International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Springer, 2000, pp. 199–204.

[17] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," University of Delaware, Tech. Rep., 03 1992.

[18] J. Arronson. (1997) Dead reckoning: Latency hiding for networked games. [Online]. Available: http://www.gamasutra.com/view/feature/131638/dead_reckoning_latency_hiding_for_.php

[19] L. Gautier and C. Diot, "Design and evaluation of mimaze a multiplayer game on the internet," in *Multimedia Computing and Systems, 1998. Proceedings. IEEE International Conference on*. IEEE, 1998, pp. 233–236.

[20] E. Cronin, B. Filstrup, A. R. Kurc, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," in *Proceedings of the 1st workshop on Network and system support for games*. ACM, 2002, pp. 67–73.

[21] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 161–165.

[22] u/Solaire Lives. (2016) Dying round corners. [Online]. Available: https://www.reddit.com/r/Infinitewarfare/comments/5bci72/dying_round_corners_losing_every_gunfight_wtf_is/

[23] u/Her Protector. (2015) Getting killed through a wall or just after rounding a corner? [Online]. Available: https://www.bungie.net/en/Forums/Post/109909127