NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Development of a Physics-Aware Dead Reckoning Mechanism for Distributed Interactive Applications

A thesis presented to the National University of Ireland, Maynooth

by

Patrick Joseph Walsh, B. E.

for the degree of Masters of Engineering Science by Research

October 2011

Department of Electronic Engineering
Faculty of Science and Engineering

Supervisor: Dr. Séamus McLoone
Co-supervisor: Dr. Tomás Ward
Head of Department: Dr. Seán McLoone

ABSTRACT

Distributed Interactive Applications (DIAs) are a class of software that allow geographically remote users to interact within a shared virtual environment. Many DIAs seek to present a rich and realistic virtual world to users, both on a visual and behavioural level. A relatively recent addition to virtual environments (both distributed and single user) to achieve the latter has been the simulation of realistic physical phenomena between objects in the environment.

However, the application of physics simulation to virtual environments in DIAs currently lags that of single user environments. This is primarily due to the unavailability of entity state update mechanisms which can maintain consistency in such physics-rich environments. The difference is particularly evident in applications built on a peer-to-peer architecture, as a lack of a single authority presents additional challenges in synchronising the state of shared objects while also presenting a responsive simulation.

This thesis proposes a novel state maintenance mechanism for physics-rich environments in peer-to-peer DIAs composed of two parts: a dynamic authority scheme for shared objects, and a physics-aware dead reckoning model with an adaptive error threshold. The first part is intended to place a bound on the overall inconsistency present in shared objects, while the second is implemented to minimise the instantaneous inconsistency during users' interactions with shared objects. A testbed application is also described, which is used to validate the performance of the proposed mechanism.

The state maintenance mechanism is implemented for a single type of physics-aware application, and demonstrates a marked improvement in consistency for that application. However, several flexible terms are described in its implementation, as well as their potential relevance to alternative applications. Finally, it should be noted that the physics-aware dead reckoning model does not depend on the authority scheme, and can therefore be employed with alternative authority schemes.

ACKNOWLEDGEMENTS

Firstly I would like to thank my supervisors, Dr. Séamus McLoone and Dr. Tomás Ward, for their advice, guidance and support in writing this thesis. Thanks also to Dr. Aaron McCoy, Dr. Damien Marshall, and all the other members of the DIA group for the assistance provided over the years. If not for all of you, this would not have been written.

Thanks also to the staff in the Electronic Engineering department, as friendly and helpful a group of people as I have ever met.

Thank you also to the Irish Research Council for Science, Engineering & Technology, who provided the funding to make this possible.

Thanks to all my friends, especially Andrew and Tadhg, who were always there to talk to, and were always willing to help if I asked, and even when I didn't.

I would like to thank my mother for all her support and assistance, I'm not sure she will ever realise how much I appreciate it. I would also like to thank my brother, Donncha, for shouldering so much work at home; it gave me the opportunity focus on this.

Finally, thank you, Lorraine. You have always been supportive and patient, despite having your own thesis and just as much (probably more!) of a workload to juggle. Your encouragement has helped me at least get this far.

This thesis is dedicated to my parents, Freda Creavin and John Walsh (deceased).

DECLARATION

I hereby declare that this thesis is my own work and has not been submitted in any form for another award at any university or institute of tertiary education. Information derived from published and unpublished work of others has been acknowledged in the text, and a list of references has been provided.

Signed: _____

Date: _____

CONTENTS

Chapter 1
Introduction

1.1 Background

Distributed Interactive Applications (DIAs) are an expanding subset of computer applications. These applications operate over computer networks, and allow users to interact in real time via graphically rich environments. Many of these applications take the form of online multiplayer games, in which participant numbers can reach several thousand. These applications have evolved in complexity over time, from relatively simple early games like Quakenet, to modern games like World of Warcraft, Halo, and Call of Duty. Development in these applications has occurred on various fronts, ranging from scaling the number of users supported, to increasing the spatial size and richness of the simulated environment.

While some developments in DIAs relate specifically to their networked multi-user nature, others (such as improvements in graphical detail and presentation) have originated in single player applications and games and have then been applied to DIAs via their multi-user equivalents. Physics simulation, for an added sense of realism and immersion, has been a relatively recent development in single player applications, and attempts are ongoing to incorporate this into equivalent distributed applications.

DIAs, and virtual environments in general, aim to present to users an environment whose behaviour is consistent with the users' experience and perception of the real world. Traditionally the level of realism achievable has been limited by technological factors such as the availability of processing power, and simulations have restricted the level of detail presented in the environment as a result. Over time, however, hardware and software performance have advanced steadily, which has allowed developers of virtual environments to increase the level of detail presented. This increase in detail has varied from improvements in graphical rendering and visual presentation, to the

simulation of more dynamic environments with more true to life behaviour. Recently this has led to increased incorporation of simulated physics into such applications. While physics simulation in single user applications is limited only by the capability of the hardware and software present within a single machine, such simulation in a distributed environment presents challenges of its own. These challenges are related to a number of restrictions that the hardware and underlying network impose on DIAs in general.

1.2 Restrictions in Physics-Aware DIA Development

The distributed nature of DIAs presents several unique restrictions to the field that must be accounted for in designing and implementing a DIA. Specifically, the geographical separation of individual users or hosts in DIAs means that designers have to overcome challenges presented by real world communication networks, most commonly the Internet. Data traffic transmitted via the Internet (and indeed other networks) is subject to a number of influences which must be accounted for or overcome in the implementation of a DIA. These influences, and their relevance to distributed simulation of physics, are outlined as follows:

Bandwidth – Bandwidth is a metric of the amount of data that may be transmitted over a communications link per unit of time. The number of users, and indeed simulated entities or objects that an application can support, may be limited by bandwidth, as a finite amount of this resource is required for the synchronisation of each change of state. The introduction of additional entities for simulation of physics adds state variables that may potentially require updating in a given simulation tick and therefore may exhaust the available bandwidth. Such a situation leads to excessive message queuing and a subsequent increase in latency which causes divergence of the simulation state across peers. Many traffic reduction techniques permit approximation errors in the state of entity state variables in order to reduce the frequency of updates.

Latency – Latency is the time taken for a packet of data to be transmitted across a network, from the application layer of one host, to the application layer on another (Blow, 1998). Latency is a dynamic, time-

variant quantity, with variation in latency over time being called jitter. Latency presents an obstacle in distributed physics simulation as its presence means that a finite time is required for an update, or change in state, generated at one host to be apparent at another host. Jitter is a further complication as it precludes the incorporation or masking of latency as a constant value in simulations.

Network topology/architecture – The topology of the network employed influences how state information in DIAs is synchronised (Smed et al., 2002). For example, in a client-server application, the server maintains an authoritative copy of world state and all clients communicate with it, and verify their actions. The server in turn disseminates appropriate information about all entities in the world to clients. In some applications this even makes it possible for the server to send a complete world state update if the local copy is sufficiently in error (Valve, 2005c). By contrast, this is not an option in peer-to-peer applications, as traditionally each peer is only authoritative about a subset of the entities making up the simulation, not the complete set.

Network reliability and ordering – In simulating environmental physics (as opposed to simple physical laws like gravity acting on a controlled entity) within a DIA, a reliable network is very important, as packet loss resulting in retransmission can lead to increased latency (Tanenbaum, 1996). This leads to event synchronisation difficulties in the simultaneous presentation of an event to all peers in the simulation. Additionally, where physical simulation takes place, there is potential for one event to cause another to happen, and lost or delayed transmission of the first event could lead to an effect preceding its cause.

Each of the limitations listed above presents challenges to the design and implementation of any DIA when compared to a single user equivalent. However, in the case of physics-aware DIAs, these influences can present more specific challenges, due to an increased number of simulated entities, many of which are not controlled by users.

1.3 Simulation of Distributed Peer-to-Peer Physics

As noted in the previous section, a potential issue of the peer-to-peer networking architecture in DIAs is the absence of a single authoritative copy of the world state. While this must be overcome in the design of any DIA, those DIAs that support the presence of non-user-controlled entities governed by the laws of physics present specific challenges. In the course of this thesis, such DIAs will be referred to as "physics-aware" DIAs, and the non-user-controlled entities as physics-aware entities. The presence of such physics-aware entities will be the distinction between truly physics-aware DIAs and those where simple constraints like gravity are applied to user-controlled entities. These physics-aware entities form an interactive part of the environment from the users' perspective, and as such they are shared objects. Examples of physics-aware entities could include balls, boxes, furniture, etc.

Challenges facing the implementation of such a physics-aware, peer-to-peer DIA are introduced below.

1.3.1 Authority

Traditional peer-to-peer architectures grant authority to each peer over specific entities, usually entities controlled by users at that peer. In a physics-aware DIA, however, there are non-user-controlled entities for which the granting of authority is more complicated. Indeed to grant permanent authority to a single peer over some or all of these entities may be inappropriate, and would render the application more akin to client-server than true peer-to-peer. As an application architecture, peer-to-peer is generally more fault-tolerant than client-server, as it lacks the single point of failure at the server. Granting authority over all physics-aware entities to a single peer would actually create a single point of failure, which is undesirable.

Alternatively, authority over some entities could be granted to each peer, thus distributing the simulation load. However in the event of two entities with different authoritative peers coming into contact, the behaviour of each would be

governed by a different physics simulation. Thus it seems that in physics-aware DIAs, the authority scheme must also be physics-aware.

1.3.2 Responsiveness

Latency and the challenges it presents to a DIA have already been introduced, but once again the addition of non-user-controlled entities introduces additional considerations. These considerations are in part related to the aforementioned authority concerns. For example, if a user interacts with a physics-aware entity, they expect to see the result of their interaction promptly, and if they have an expectation of the result, accurately. However, if the user interacts with an entity for which its peer is not authoritative, then a delay may be experienced while their local peer validates the interaction with the remote authoritative peer. Further, a remote authoritative peer may have a different view of the interaction, ultimately resulting in the behaviour differing from a user's expectation.

1.3.3 Consistency

Consistency in a DIA refers to the ability of the DIA to ensure that each user's view of the world is identical, or as close to identical as can be achieved for given conditions. Traditional consistency metrics examine entities on an individual basis, often giving a measure of the application's ability to represent a host's controlled entities' states at remote hosts. A basic aspiration for a distributed simulation is to present entities as being 'in the right place at the right time' and a simple metric to capture this notion are spatially-derived measures of consistency such as can be calculated using distance measures between the entity positional state at its local peer, and the representations at remote peers.

In real-time applications, the *Consistency-Throughput Tradeoff* (Singhal and Zyda, 1999) acknowledges that true consistency is unachievable. Consequently most real-time DIAs accept a controlled level of inconsistency, and utilise approximated models of controlled entity motion as part of traffic reduction mechanisms, such as the dead reckoning algorithms employed within the Distributed Interactive Simulation (DIS) standard (Durbach and Fourneau, 1998,

IEEE, 1998). Dead reckoning is a linear extrapolation algorithm which uses historical entity state to generate a prediction of future entity state. All peers model all entities, with each peer thus knowing both the modelled and actual behaviour of local entities. These behaviours are continuously compared for local entities, with updates being transmitted to remote users when the difference between the models exceeds a set error threshold.

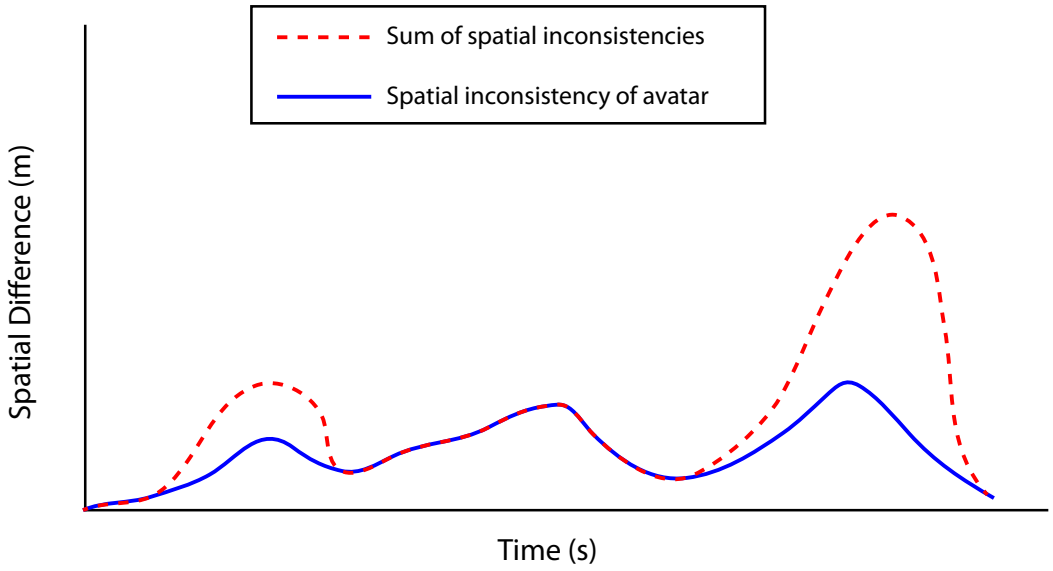


Figure 1.1: Controlled entity (avatar) inconsistency compared to all inconsistencies present in a physics-aware DIA.

By only considering single entities, and those controlled by the host (i.e. first order generation of inconsistencies by local entities), these metrics are less than ideal for use in physics-aware DIAs, as they may not present a clear picture of all the inconsistency in the environment. A qualitative hypothetical example of this is given in Figure 1.1. The solid line represents the spatial inconsistency (difference in position between local and remote representations of state) present in the state of a controlled entity, or avatar, while the dashed line represents the sum of the spatial inconsistencies present in the state of both the avatar, and physics-aware environmental entities with which the controlled entity collided.

This added error in the simulation of collisions arises in part from the use of approximated models of controlled entity motion in traffic reduction mechanisms, such as the dead reckoning algorithms employed within the

Distributed Interactive Simulation (DIS) standard (Durbach and Fourneau, 1998, IEEE, 1998). Dead reckoning is a linear extrapolation algorithm which uses historical entity state to generate a prediction of future entity state. All peers model all entities, with each peer thus knowing both the modelled and actual behaviour of local entities. These behaviours are continuously compared for local entities, with updates being generated to remote users when the difference between the models exceeds a set error threshold.

In static environments, where controlled entities may only interact with other controlled entities, this error is limited to only being present in the dead reckoned models. However, in environments with physics-aware entities, this error can potentially be communicated, or passed on, to such entities. Consider a scenario in an application where a physics-aware ball atop a hill is simulated, as illustrated in Figure 1.2. If an entity as represented by ball A (at its local peer) were to approach the ball (ball B), but stop with a distance between it and ball B that is less than or equal to the error threshold of the dead reckoning model, a remote peer could observe ball A disturbing ball B, and ball B subsequently rolling away down the hill, resulting in a spatial inconsistency of δ in the state of ball B. This would manifest as a difference between the solid and dashed lines in Figure 1.1. This potential for state divergence, and the resulting visual disturbance to the application required to correct it, can be considered a "physics-consistency-cost" associated with the entity's path. The exact means of correcting inconsistency in physics-aware entities might vary across applications. For example, an entity with an authoritative peer could be updated by means of a periodic heartbeat packet similar to controlled entities in the Distributed Interactive Simulation standard (IEEE, 1998).

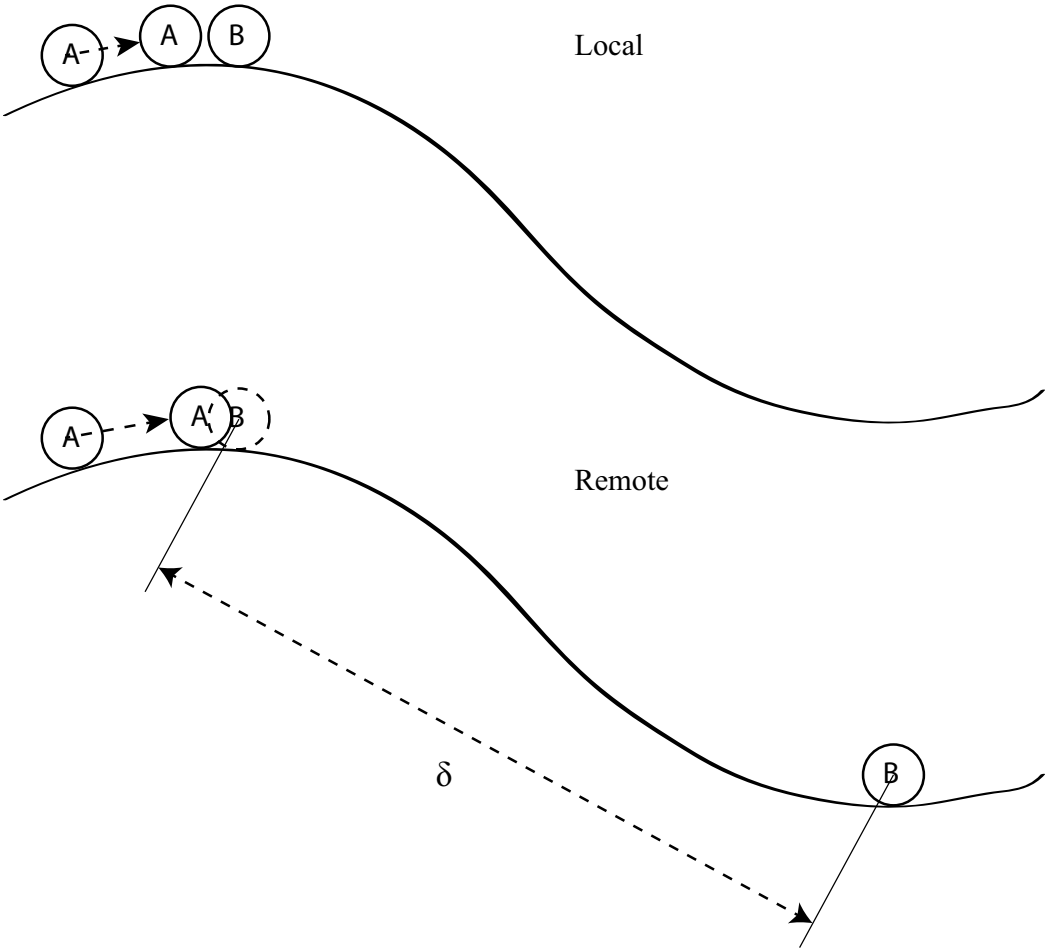


Figure 1.2: Local and remote representation of Ball A moving close to, but stopping short of, Ball B. At the remote peer ball A is incorrectly observed to collide with ball B as the error threshold of the dead reckoning model is not exceeded until it is close enough to collide with ball B. δ represents the spatial inconsistency present in the state of ball B.

A similar scenario, but with a potentially much higher cost, can be framed by considering a row of dominos. As per the previous example, an entity could be observed at a remote peer to collide with a domino, while its true behaviours is to closely miss the domino. A specific example of this is presented in Figure 1.3, where a ball approaches a row of dominos, but stops close to them. However the dead reckoning model is observed to collide with the first domino, which in turn topples the row. In this instance, the magnitude of the spatial inconsistency communicated to the first domino (δ`) may be less than that communicated to the ball previously, but due to the nature of dominos, a single incorrectly disturbed domino can in turn disturb its neighbours, resulting in a spatial inconsistency of approximately $N$ x δ`, for N dominos. The magnitude of the inconsistency imparted to a single domino, and even the sum imparted to them all, may be

significantly smaller than that of the ball in the previous example, but the number of entities involved is significantly more. Thus even small spatial inconsistency in physics-aware entities can result in an appreciable physics-consistency-cost if complexity (in which *N*, the number of dominos is a factor) is considered.

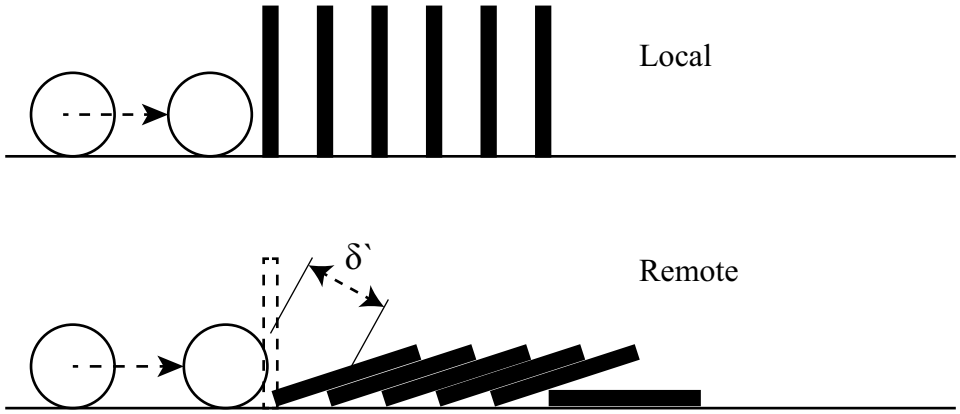


Figure 1.3: Local and remote representation of a ball moving close to, but stopping short of, a row of dominos. Due to the error threshold of the dead reckoning model not being exceeded until after it shows the ball colliding with the first domino, the remote peer incorrectly observes all the dominos being toppled as a result. δ` represents the inconsistency present in the state of a single domino.

Both of the previous examples present scenarios with a high susceptibility to incurring a physics-consistency-cost. Of course not all physics-enabled interactions lead to such cascades in consistency loss. As a counter example, consider a scenario where the dead reckoning model of a remote entity is incorrectly observed to collide with a much larger physics-aware entity, with large inertia, e.g. a human avatar colliding with a car or a van. Under realistic physics simulation the modelled avatar may incorrectly impart a spatial disturbance to the van, but this would be significantly smaller than the disturbance experienced by the ball in the first example (especially in terms of relative size) – it may even be imperceptible to the user. Consequently such an error in state is easier to correct than the dominos in the second example. The incorrect disturbance is still a physics-consistency-cost, but is significantly smaller than the previous two examples.

Currently entity state update protocols for peer-to-peer distributed simulations such as dead reckoning do not explicitly take into account these higher-order

inconsistency effects for physics-enabled environments. Therefore the ability of these techniques to regulate consistency in such situations is not optimal and improvements may be possible by incorporating information about the local physics environment. Each of the above examples will be referred to as the "ball example", the "domino example" and the "large inertia example" later in this thesis.

1.4 Aim of Thesis

This thesis aims to implement a physics-aware mechanism for synchronisation of entity state within a peer-to-peer, physics-aware DIA employing dead reckoning algorithms for the purposes of traffic reduction. It builds on previous work relating to DIAs and simulation of physics in virtual environments.

1.5 Contributions

The main contribution of this thesis is the development of an adaptive-threshold, physics-aware algorithm for consistency maintenance in peer-to-peer, physics-aware DIAs that utilise dead-reckoning algorithms. Additionally, a proposed means of identifying physics-consistency-costs is described. Finally, a testbed application supporting two-dimensional physics simulation in environments synchronised via a simulated network connection is described. Using this testbed, results are presented which illustrate the efficacy of the proposed identification mechanism for physics-consistency-costs.

1.6 Layout of Thesis

The remainder of this thesis is laid out as follows:

Chapter 2 – In this chapter, a brief background to the subject area is provided. Simulation of physics in virtual environments, both distributed and local is introduced, and the challenges of supporting physics simulation in distributed interactive applications are identified.

Chapter 3 – At this point, the development of the state management algorithm for physics-aware entities is outlined, as well as some

alternatives considered during the course of this development. The structure of the testbed application is also explained.

Chapter 4 – Here the performance of the state management algorithm as proposed is analysed via implementation in the testbed.

Chapter 6 – To conclude, the potential for future work based on this algorithm and testbed is discussed, along with conclusions drawn from the results collected from testing of the algorithm.

Chapter 2
Background

In this chapter the background to the research area is outlined. Firstly an introduction to the history of Distributed Interactive Applications (DIAs) is given, as well a means of categorising different types of DIAs. Following this, the advent of physics-simulation in virtual worlds, both single user and distributed, is explored and the developments in the field are outlined. Immersion in simulated worlds is then introduced, and the role that physics simulation plays in achieving immersion is explained. The concept of consistency in DIAs is then explained, and the implications of physics-awareness in this area examined. Some commonly used traffic reduction mechanisms in DIAs are then outlined, and their impact on a physics-aware DIA is examined.

2.1 DIA History

DIAs have been undergoing active development for approximately 25 to 30 years and have seen many refinements and improvements in that time. These improvements have been driven by a number of factors, including (but not limited to):

- Developments and advances in communications networks,
- Improvements in processing power,
- Competition between developers of commercial DIAs and games,
- Requirements and demands of end users, and
- Improvements in similar single-user, local applications.

Development of DIAs has taken place primarily in three domains, specifically military, academia, and industry (Delaney et al., 2003, Marshall et al., 2004). The motivations, and thus ultimate aims, in each of these domains have traditionally differed, resulting in a variety of both experimental and production platforms and architectures in each domain. However, developments in one field have been applied to, and proven beneficial, in other fields. An example of this would be the dead reckoning algorithm, a traffic reduction mechanism used in a

number of modern DIAs (Hardt and White, 1998, IEEE, 1998) which was developed as part of military simulations, but is now used in multiplayer games.

2.1.1 Military

Historically the Department of Defence (DoD) has been a significant force in the development of DIAs, having constructed a series of platforms, each refining and improving upon the previous. Their interest in the area of DIAs stemmed from a need to be able to simulate wartime scenarios in a safe and cost-effective manner, thus minimising both expense and loss of life in real deployments. The DIA platforms developed by the military are outlined below.

SIMNET

The Simulation Network (SIMNET) (Calvin et al., 1993) was the first major contribution from the US DoD to the field of DIAs. Development of SIMNET commenced in 1983, with the United States Army receiving a production package in 1990. The primary aims of SIMNET were to provide high quality simulators at a low cost, and to network multiple simulators together, in order that multiple simulators could make up one single DIA.

The networking architecture underlying SIMNET is a peer-to-peer one, whereby each player is responsible for their own state, and for notifying other participants of changes in their state. By multiple players taking this approach, each of them is able to use the updates they receive to maintain a local model, or view, of the world including the state of remote players by making appropriate changes in response to each update. As a mechanism to reduce the network traffic generated and received by each peer, SIMNET utilises dead reckoning mechanisms, which allow peers to model the positions of remote entities by means of extrapolation from instantaneous state. Each peer maintains both the true and modelled state of its own entities, and transmits updates of state to other peers when the modelled state differs from the true state by more than a preset threshold value. This is explained in more detail in Section 2.6.3.

DIS

Ultimately the protocol used by SIMNET proved too simulation specific, and an improved system was needed. Hence the Distributed Interactive Simulation Network (DIS) platform was developed, with a view to providing a more general, and therefore flexible and extensible, protocol (Hardt and White, 1998).

In order to meet its requirements, the DIS architecture implements the protocol data unit (PDU). PDUs are messages that are generated to inform of an event occurring, e.g. a missile being launched. There is a finite set of PDUs, but each one is designed to be generic enough that updates from all types of participants can be handled. The DIS architecture also extends SIMNET's dead-reckoning mechanisms by defining nine additional algorithms. These algorithms include both higher order models (incorporating acceleration as well as velocity), and models for orientation of entities, i.e. angular state.

High Level Architecture

Research within the military at present is focussed on the High Level Architecture (HLA) (Dahmann et al., 1997). Interoperability and extensibility are at the heart of the HLA, by contrast to DIS which, while versatile, still requires the development of application specific components. During the design of the HLA, it was recognised that no single simulation is capable of meeting every potential user requirement, due to the vast range of potential requirements. Consequently it is designed to allow for interoperability between current and future simulations within the DoD, as well as the reuse of simulation components, and complete simulations. By ensuring such flexibility and compatibility, it is hoped that time and money can be saved in the long term.

2.1.2 Academic

Concurrently to the development of DIAs by the US DoD within the military, there was also significant interest among the academic community. Several of the key DIAs developed in academia are now discussed.

NPSNET

One of the earliest academic research teams in the area of DIAs also had a connection to the military, in the form of the Naval Postgraduate School Research Group (NRG), who developed the NPSNET project (Capps et al., 2000). There have been five iterative versions of NPSNET to date, with the current being NPSNET-V. The goal of the NPSNET-V design reflects the broad range of research already conducted by the NRG (including consistency, extensibility, scalability and interoperability) in aiming to provide an architecture that allows for research to be conducted into all aspects of DIAs. The previous incarnation, NPSNET-IV, utilised an IP multicast protocol alongside the DIS application protocols previously mentioned (Macedonia et al., 1994).

Spline

Scalable Platform for Large Interactive Network Environments (Spline) is a peer-to-peer DIA that was developed at the Mitsubishi Research Laboratories (Waters et al., 1996). Spline was developed with scalability as a key concern, and as such is designed to accommodate large numbers of users. An important means in achieving this scalability is the division of the environment into smaller areas called locales (Barrus et al., 1996). Information from each locale is only disseminated to users that are interested in it, rather than all users, for whom it may be irrelevant.

CAVERNsoft

The CAVE Research Network (CAVERN) is a group of institutions with virtual reality technology (e.g. CAVEs (Jones, 1998) and Immersadesks), and CAVERNsoft (Leigh et al., 1997) is the architecture they have developed to facilitate collaborative virtual reality (VR) between geographically separate locations. It facilitates the use of a variety of network architectures, including peer-to-peer and client-server, depending on the nature and requirements of the specific application.

DIVE

DIVE is a DIA utilising a peer-to-peer architecture that was developed by the Swedish Institute of Computer Science (Frécon and Stenius, 1998). DIVE shares some features with the previously described Spline, such as replicating subsections of the environment only to those with an interest in that area, and scalability for large numbers of users. However, a significant difference between the two is the use of a shared world database in DIVE, whereby each peer maintains a copy of the world, and changes are actively replicated to other peers. This prioritises interactivity, while tolerating slight differences between world copies.

MASSIVE

MASSIVE is another project that, like DIVE and Spline, employs the concept of locales to improve scalability (Greenhalgh et al., 2000). Locales also have boundaries between them, which serve as inter-locale links, and can be used to create relationships between locales. Another feature of MASSIVE is that it allows "aspects" of varying fidelity for each locale, so that peers with fewer resources may join the simulation.

2.1.3 Industry (Multiplayer games)

DIAs have always been a popular class of application in industry, but in contrast to the military and academia as previously outlined, most industrial DIAs are developed as products to be sold to end users, rather than for the organisations own use. In many instances, studios traditionally developing single player games expanded into multiplayer games to meet user demand, and to access additional markets. Multiplayer games can be very varied in the forms that they take, and the nature of the simulation (e.g. first-person shooter (FPS), real time strategy (RTS), and flight simulations all present a different user experience) may mean that different demands are made of the underlying architecture. While multiplayer games have existed since the 1970s, e.g. Spasim and Maze War, they have seen much more intensive development since the 1990's.

id Software

id Software (www.idsoftware.com) have been a driving force in the development of both multiplayer and single player games for many years, having created both the Doom and Quake series, and more importantly the underlying engines, including idTech 1-5. These game engines would be in turn used in other applications, developed both by id Software, and other companies.

The first id Software title to feature multiplayer gameplay was the original Quake, though the popularity of this element was limited until the release of QuakeWorld. This was a free update that implemented a Client Side Prediction algorithm to mask latency, and thus allowed users on slower connections to enjoy a responsive gaming experience.

Tribes Engine

In 1998, a game studio named Dynamix released the first game in the Tribes series. The first game was unusual for its time in being solely a multiplayer game, with no singleplayer campaign or experience. The Tribes engine employed novel networking code and algorithms (Frohnmayer and Gift, 2000), which would in turn be used in the Torque game engine and OpenTNL network engine, both released by former employees of Dynamix under the Garage Games moniker. The significance of the networking model employed by the Tribes Engine is evidenced by its use as the basis for multiplayer games released over 10 years later, such as Halo: Reach (Aldridge, 2011).

Valve/Source Engine

Valve Software (www.valvesoftware.com) were one such company who licensed the Quake engine as above in order to develop a game of their own, namely Half-Life. In and of itself, this is unremarkable, but the success of this game prompted Valve to develop a sequel, and as part of this, they developed the Source game engine (Valve, 2005b). The Source engine was designed to be modular and easily upgradeable, and as such it has been available for seven years without a conventional change in version number.

In addition to this, and continuing to provide multiplayer gameplay over networks, the Source engine also provided detailed physics simulation by integrating a fully featured physics engine, an important development in the context of this thesis. This was utilised in Half-Life 2 upon release and later in Portal, another Valve Software title in which players were required to solve physics-based puzzles.

Frostbite Engine

The Frostbite engine was developed by an Electronic Arts Studio named DICE (www.dice.se) in 2008, with an update released in 2009. While not implementing significant advances in terms of networking, a significant feature of this engine in the context of this thesis was the addition of dynamically destructible environments. Initially this was limited to individual walls, but the 2009 update allowed for destruction of entire buildings. This was significant in so far as the game engine utilises an underlying physics engine to simulate this destruction dynamically, in contrast to previous games, which accomplish this through pre-scripted and rendered animations.

In the next section, criteria for classifying DIAs according to their mechanisms of operation is introduced and explained.

2.2 Categorising DIAs

2.2.1 Network Architecture

There is a wide variety of network architectures available to designers of DIAs at present. The architecture of a DIA in this context refers to both the path(s) that network traffic used for synchronisation follows between users, and the manner in which the world state is stored.

Traditionally there have been two main architecture types: client-server, and peer-to-peer. Figure 2.1 below illustrates the difference in structure between the two. In the client-server architecture, there is one central server that maintains an

authoritative copy of the world, and each client communicates with that. There is no direct communication between clients. Updates are sent by the clients to the server, and the server processes these, and disseminates any resulting changes in state to clients as necessary. In a peer-to-peer architecture, by contrast, there is no central authority, and all peers communicate with each other.

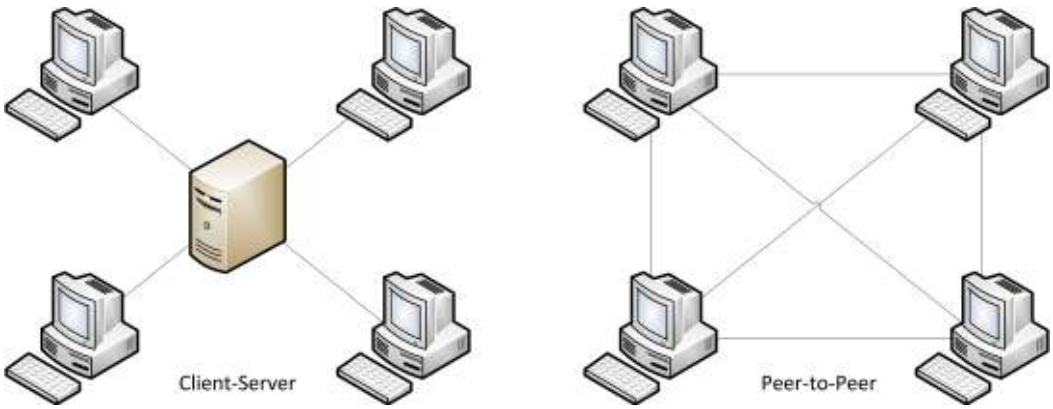


Figure 2.1: Comparison of the structure of Client-Server architectures and Peer-to-Peer architectures

Each of these architectures traditionally has its own advantages over the other, with selection of one to use often being related to the requirements of the application. For example, latency with a peer-to-peer architecture can be less than client-server, as data does not have to be relayed through an intermediate computer, i.e. the server. Additionally, the scalability of a client-server application depends heavily on the processing power and bandwidth available at the server, as it has to have both sufficient bandwidth to receive from and send to all clients, as well as simulating the entire world, where clients may only simulate an area of interest for performance reasons. Conversely, the scalability of peer-to-peer applications may be limited by the networking capability of a single peer, as each peer has to handle updates from all other peers in a similar way to the server in a client-server deployment (Fiedler, 2010b).

In recent years, hybrid architectures have been developed for applications, an example of which is shown in Figure 2.2. In this example, each user's machine behaves like a client in a client-server scenario, but there are multiple servers arranged in a peer-to-peer architecture. In this manner each server could, for

instance, be responsible for simulating a particular area of the world, and a client communicates with the appropriate server for the area of the world in which they reside (Assiotis and Tzanov, 2006).

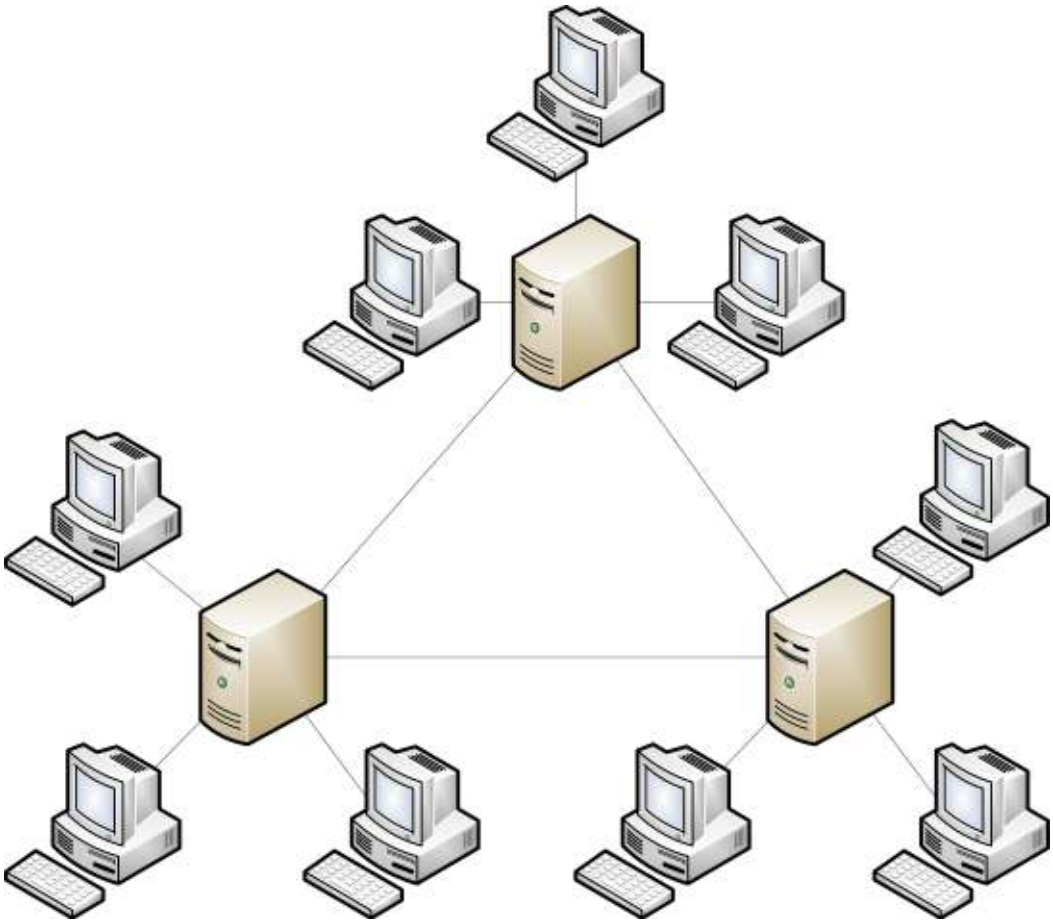


Figure 2.2: Example of an architecture implementing a hybridisation of the client-server and peer-to-peer architectures

There is no consensus among commercial game developers at present as to a single ideal architecture to employ for every application, with that decision being made on a per-application basis (Davis, 2008). For example, the recently released Gears of War 3 shooter utilises client-server networking with dedicated servers for its multiplayer features (Watts, 2010), whereas real time strategy games (RTS) may often employ a peer-to-peer architecture (Bettner and Terrano, 2001). Yet another alternative, in the form of a hybrid architecture, has been suggested for use in Massively Multiplayer Online Role Playing Games (MMORPGs) (Assiotis and Tzanov, 2006).

Entity Replication in Peer-to-Peer DIAs

Figure 2.3 illustrates how peers maintain ownership and authority over entities local to them, while modelling entities owned by other peers. These models of remote entities at each host or peer are based on the updates received by that peer, and are, at best, only an approximation to those entities' true states.

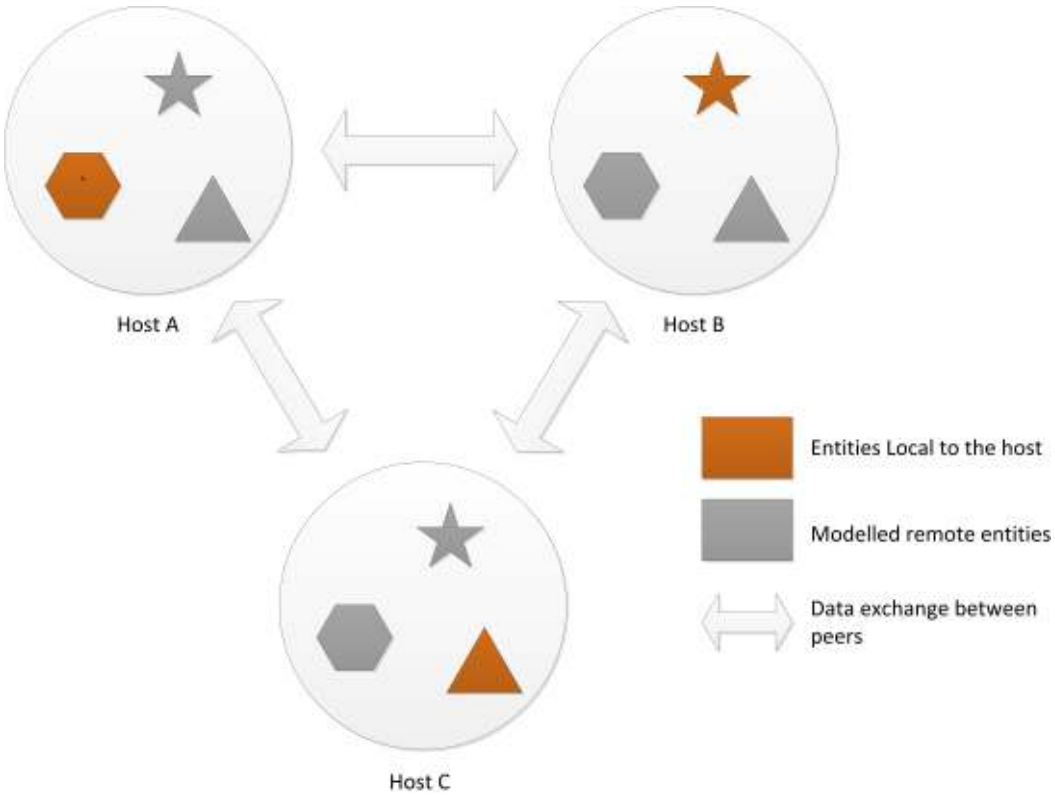


Figure 2.3: Ownership and modelling of local and remote entities respectively, in peer-to-peer architecture

A flaw with peer-to-peer for competitive applications that has been identified in the literature is a lack of a single, central authority, which can leave the application more susceptible to cheating by players (Webb et al., 2007, Yan and Randell, 2005). The reason for this susceptibility is that without an authority to validate the information sent by each peer, a peer could send false information in its updates to gain an advantage over other peers. For example, a user could modify their local program to allow them to move faster, or to have greater manoeuvrability than others. However, modifications to the basic peer-to-peer application structure have been proposed to counter such cheating, one of these being the Referee Anti-Cheat Scheme (RACS) (Webb et al., 2007). Under this

scheme, peers send updates to other peers as normal, as well as to a referee process. This referee process runs on a trusted host, and is able to validate updates (in order to detect cheating), and resolve any conflicts arising between peers. It does not send updates to peers unless conflict resolution is necessary, or cheating is detected, and correction deemed necessary. A further proposed extension of this scheme, for scalability purposes, is the implementation of distributed referees.

2.2.2 Time Management

A further quality by which DIAs may be categorised is the mechanisms employed to manage the simulation of time within the environment. This is an important categorisation as it restricts the types of behaviour that are possible within the DIA. Time management mechanisms in DIAs can be divided into two key categories:

1. **Logical Time** – Passage of time occurs only in response to the generation of events within the simulation (Lamport, 1978). If no events occur, then no time is considered to have elapsed, and time does not increment. This has also been referred to as causal time.
2. **Real Time** – Passage of time occurs regardless of simulation events, similar to the passage of real world time. Consequently it is also referred to as physical time, or wall-clock time.

Applications whose execution is driven by logical time can be described as logical time applications (Page and Smith, 1998). Turn-based games, such as chess, or collaborative document editing environments would be examples of such logical time applications. In contrast to this, games such as Doom and Quake, as mentioned earlier, are examples of DIAs utilising real time management. In this thesis, we will focus on real time applications, as simulation of physical systems is ill-suited to the logical, or causal, time.

The next section will provide a background to physics simulation in both local and distributed virtual environments, as well as examining the motivations behind such simulation.

2.3 Physics Simulation in virtual environments

As already stated, DIAs have become more complex as computational hardware has become more powerful, and more efficient. The same is true of video games in general, both in the multiplayer games that fall within the realm of DIAs, and single player games which do not. These advances have taken place in the realms of artificial intelligence (AI), graphical and visual rendering, and environmental detail and complexity, to name a few. Improvements in physics simulation have been driven in part by the latter two areas, as well as a desire for increased realism in games.

Games offering physics simulation in one form or another have been available for over 25 years, but the level of detail and complexity has increased significantly over time (Hecker, 2000). Early games, such as Tetris, Super Mario, or even Pong, employed simple simulation of physical laws to simulate movement in two dimensions, and to ensure that objects could collide with each other, as opposed to passing through each other. With limited processing time available in many computers at the time, this was all that could be achieved while still being able to execute all other aspects of the game. An added issue for developers was that these physics simulations often had to be constructed on a per game basis, due to each game having differing requirements, and optimisation and minimisation of program code being required due to hardware constraints, as already alluded to.

As computational platforms became more powerful, however, physics simulation within games advanced. Initially this took the form of more detailed simulation within two-dimensional games (e.g. Sonic the Hedgehog, Micro Machines), but with the advent of three-dimensional game environments, with six degrees of freedom (e.g. Quake, Half-Life) scope for further advances was soon available. In the 1990's, a shift took place in the way in which physics simulations were implemented in games. Where previously physics simulations were tailor made

for each game, a new type of middleware called physics engines emerged (Millington, 2007). A physics engine is a piece of reusable software that can be integrated into a larger application to simulate the physics of the virtual environment.

In general, physics engines provide collision detection functionality, and implement Newtonian mechanics, with early engines being limited to rigid body dynamics. As before, however, availability of increased processing power, and refinement of simulation techniques led to more complex and flexible engines, with fuller feature sets. The advent of techniques such as ragdoll physics (Glimberg and Engel, 2007, Mulley and Bittarelli, 2007) allowed developers to replace scripted animations with real time simulations that responded to unique instantaneous inputs. From a game perspective, this allowed for the presentation of a more realistic, and hence believable, world.

As developers began to realise the potential of accurate physics simulation, and by extension physics engines, efforts emerged attempting to make more processing power available to simulations. This included the development of Physics Processing Units (PPUs), devices (Yardi et al., 2006, Yeh et al., 2007, Bishop et al., 2000) which were optimised for the typical calculations arising in physics simulations. In 2005, AGEIA Technologies released the first commercial PPU with support for their PhysX SDK (Ageia, 2005a, Ageia, 2005b). Ultimately this approach was superseded by the development of technologies for General Purpose Computing on Graphics Processing Units (GPGPU) (Enhua and Youquan, 2008), whereby GPUs already present in computers could be used as coprocessors for physics calculations. Nvidia's CUDA (www.nvidia.com/object/cuda_home_new.html) and AMD's Stream (www.amd.com/stream) are examples of language libraries facilitating this, with Nvidia currently owning and developing the PhysX brand and technologies.

Currently, there are many different physics engines available to game developers, ranging from open source projects providing rigid body simulations in two dimensions, such as Chipmunk Physics (code.google.com/p/chipmunk-physics), to the industry standard Havok Physics (www.havok.com) that has been used in

over 150 published games to date, as well as having been incorporated into a number of game engines.

Physics simulation and physics engines have also come to be of interest to developers of virtual worlds outside the domain of games, with academics observing that many of the physics engines available for games are not general enough for use in every type of application (Nourian et al., 2006). For example, highly detailed flight simulations, and first-person shooters might place different priorities on aerodynamics and collision mechanics respectively. The xPheve project designed an extensible physics engine with this in mind, where each physical law to be simulated is implemented as a component, and appropriate priority and precision can be configured for each law. The versatility of this engine can be seen from its use in both a military simulation, and a separate surgery simulation (Nourian et al., 2005), by means of different configurations of physical laws.

2.3.1 Causal and Non-causal simulation

As already stated, physics simulation has been used for a variety of purposes, including addition of realism to rigid bodies, and animation. One such area of animation has been the simulation and rendering of particles, such as dust clouds. In light of this varied application, it is necessary in the context of this thesis to distinguish between these purposes, and the effects that they have on the simulation. This gives rise to the definition of causal and non-causal events.

Causality refers to the relationship between two events, the first being the cause, and the second being the effect, where the effect is brought about as a consequence of the first. Extrapolating from this definition, in the context of the simulation of virtual environments, it is possible to classify events and interactions as either causal, or non-causal. A causal event is one which brings about a corresponding effect, with a non-causal event having no quantifiable or discernible effect. An example of a causal event would be a collision with another entity, and the state of that entity being altered, while a non-causal event would be typified by either a collision with a simulated wall whereby the wall

was undisturbed, or simply the exact motion of particles in a simulated dust cloud.

This distinction is important, as it suggests a way to prioritise events and associated data for replication, as well as to avoid unnecessary traffic generation. It may also provide a criterion for allowing data compression by means of approximation of state variables in certain instances.

2.3.2 Determinism in physics simulation

Determinism is a quality of an algorithm, system, or by extension, a simulation. In a mathematical or computer science context, determinism of an algorithm or system refers to the ability of that algorithm or system to reliably and consistently produce the same output(s) for a given set of inputs and initial conditions (Bullet, 2008, Fiedler, 2010a, Qvist, 2009). A further clarification, in terms of systems, is that the system must always pass through the same set of states in going from the input(s) to the output(s).

In terms of a physics simulation or engine, determinism can be considered to be the ability of a simulation to produce the same final state in response to a given set of initial conditions and inputs. This is an important quality of a physics engine as a form of middleware, as developers utilising the engine need to be satisfied that their application will behave similarly, and produce reliable and consistent results across varied platforms. This is of extra importance when platforms with varied hardware are used to simulate the same world concurrently, as is often, or even usually, the case with multiplayer games, for example PC gamers with different CPUs and GPUs.

However, one potential exception to a requirement for precise determinism could be those events classified as non-causal in the previous section. Here if any discrepancy arising will have no further (causal) influence on later events, it may make no material difference to users or their simulations, and will consequently resolve itself. Therefore, if possible, deterministic simulation should be prioritised for causal events and elements, in the event of both deterministic

simulation and non-deterministic simulation being available. A caveat to this, however, is that any discrepancies arising from such non-deterministic simulation should still be reasonably small, such that two users do not observe a radically different view of the same environment as a consequence. For example, the exact motion of particle effects in a cloud does not need to be simulated deterministically, as the particles have no causal influence on the simulation. It is sufficient to ensure that the cloud as a whole is accurately replicated in position, size and density.

2.3.3 Networked Physics Simulation

Client-Server

As already mentioned in Section 2.1.3, the Source engine developed by Valve was one of the first engines to incorporate physics simulation into a game engine that also provided facility for multiplayer sessions over a network. The Source engine utilises a client-server architecture for multiplayer games (Valve, 2005c), where the server may either be a dedicated server, or one of the users may host the session, whereby one computer serves as both client and server. The Source engine utilises compression, interpolation, prediction and lag compensation to accommodate the issues introduced by networked games. The engine also takes advantage of the presence of an authoritative server in the client-server architecture, by distributing updates to clients in the form of snapshots and "delta snapshots" (changes in world state since the last snapshot) of the world state, by default at a rate of 20 snapshots per second. In order to make this transparent to people developing with the Source engine and SDK, base classes are provided to create both server-side, and client-side versions of an entity, and specific variables within the entities can be configured for replication (Valve, 2005a).

A notable difference in many multiplayer games based on the Source engine when compared to single player titles, however, is the quantity of environmental objects subject to the laws of physics. While there are such entities, they tend to be fewer in number. For example, Counter-Strike Source has less physics-aware objects than Half-Life 2, to compare two games based on the engine that were

released at similar times, and both developed by Valve Software themselves. There is no established literature, or official information from Valve Software to explain why this might be the case. However, there have been indications from other developers that instead of trying to make the network work more effectively to replicate physics-aware bodies, there is a trend towards altering the gameplay to accommodate the network (Aldridge, 2011). This approach is far from invalid, but fails to solve the issues of physics simulations in multiplayer games, preferring to remove, or at least minimise this element. This may explain the aforementioned discrepancy between singleplayer and multiplayer applications.

Peer-to-Peer

As already explained, client-server is one of two basic architectures available to application developers, and even alternative architectures are typically constructed as a hybrid or combination of these two basic arrangements. Thus, while Valve's solution as implemented in the Source engine may be appropriate for client-server architectures, it is not directly transferable or applicable to a peer-to-peer structure. For example, in a peer-to-peer application, there is no central authority analogous to a server to replicate snapshots of the entire world state.

Thus far there has been limited success with the development of physics-aware entity state management techniques for peer-to-peer applications. The key hurdles for such algorithms are issues of authority, and scalability (Fiedler, 2010b). One such algorithm has proposed a local authority scheme, whereby peers can assume authority over physics-aware entities, in addition to already being responsible for those local entities that they control. This algorithm as proposed, however, does not utilise dead reckoning models. Instead it employs a constant update rate, whereby each peer sends a preset number of update packets to remote peers every second. These updates all share a common structure, outlined below:

Sequence Number – The first section of the packet (16 bits proposed for a 30 packets per second example) consist of a sequence number for

ordering purposes, so that if packets arrive out of order, entities are only updated with the most recent state.

Input – The user's input is then described so that remote peers can simulate the user-controlled entity's behaviour. Note that Fiedler's proposal assumes only one user-controlled entity per peer.

Rigid Bodies' State – Finally, the remaining space in the packet is filled with the compressed state of as many rigid bodies, or physics-aware entities, as possible for which the peer is the authority. The specific bodies to include in a given update are determined by means of a "priority accumulator" which considers the time since an entity was last updated, and a weighting factor which allocates greater importance to the updating of some entities.

In the event of two players' areas overlapping, the lowest player ID becomes the authority. However, in the event of a player interacting with an object, they become the authority for that object, until that entity comes to rest again. In the case of multiple peers observing physics-aware entities that are not controlled by an interaction authority, the peer with the lowest player ID is authoritative in the event of conflicting views. Fiedler (2010b) acknowledges that this approach of granting authority renders games susceptible to cheating, and cautions that it only be used in cooperative games.

A potential disadvantage of this approach in a general sense is that peers providing state updates for entities until they come to rest may generate unnecessary data; if the physics engine in use is deterministic the motion of physics-aware entities is therefore predictable provided that no other player interacts with the entity before it comes to rest. However, this is less of a concern in an application using a constant update rate, as in Fiedler's example, than it might be for a dead reckoning application, where update rates can fluctuate.

In the next section, the concept of immersion in interactive applications is introduced, and its influence as a motivation behind the addition of physics-awareness to virtual environments is discussed.

2.4 Immersion in DIAs and Games

Immersion is a term and concept that has been applied to virtual worlds and users' experiences of them for a number of years now. As a term, there have been varying interpretations and understandings of what it means, but most commonly it is used to describe the degree of involvement with a game (Brown and Cairns, 2004) or virtual world. Varying levels of immersion have been described, with one scale classifying these as being engagement, engrossment, and total immersion. A wide range of factors can influence the level of immersion achievable by a user in a virtual world, including the user's preference towards the application type (e.g. game genre, in the context of games), the quality of the construction of the application, and the scenario presented to the user (Jennett et al., 2008). The disconnection from the real world brought about by immersion has also been shown to take a measurable and quantifiable amount of time to be overcome upon stopping interaction with the environment (Cairns et al., 2006). Given that the objective of such games, as recreational applications, is to engage with the virtual world, and arguably to draw some of the user's focus from the real world, this exhibits just how powerful and successful applications can be in terms of developing immersion.

Immersion is not solely applied to the domain of games however, and can be used to describe more general virtual worlds, and by extension, DIAs beyond the field of multiplayer games (Swing, 2000). While research suggests that poor coherence in games and applications can be overcome if immersion has already been achieved, the same research indicated that a lack of coherence can be a barrier to the initial development of immersion (Cheng and Cairns, 2005). Thus presenting a coherent and realistic environment to the user from the outset can aid the development of immersion, and simulation of realistic physics should serve to facilitate this in appropriate applications. For example, some games, such as Battlefield Heroes, (www.battlefieldheroes.com), deliberately present a less realistic environment, and the simulation of "true-to-life" physics might in fact be inappropriate.

Consistency in DIAs can be thought of as the ability of an application to ensure that an entity is 'in the right place at the right time' for all users. Thus if inconsistency is present in a DIA, each user may have a differing view of the environment. This in itself would be unrealistic behaviour, which could serve as a barrier to immersion, especially if users can speak to each other, or communicate otherwise, as in many DIAs.

In the next section, the concept of consistency in DIAs is introduced, and the information outlined so far in this chapter is built upon in relating consistency to physics-aware DIAs specifically.

2.5 Consistency in Distributed Interactive Applications

An important element of any DIA is the ability of the application to maintain a consistent representation of the state variables stored in each host's version of the shared world database. Each host in a DIA transmits synchronisation messages containing the latest values of the relevant state variables across the network connecting hosts to maintain the consistency of this database. Three distinct elements of consistency in DIAs have been identified in the literature (Bouillot and Gressier-Soudan, 2004, Roberts, 2004):

1. **Synchronisation** – Guarantees that all hosts' versions of the state variables are equal within the constraints of real-time simulation.
2. **Concurrency Control** – Participants on different hosts are allowed to make changes to the shared world databases concurrently, with the results of these changes being the same as if made by a single host.
3. **Causal Ordering** – Ensures the maintenance of the causal relationships (cause and effect) between related changes to the shared world database.

For many modern DIAs, the underlying communications network between the hosts is the Internet, and the Internet in its current form thus poses two major obstacles to DIAs. Specifically, these obstacles are network latency (and jitter), and limited network bandwidth.

Network latency is the time taken from the start of exchange of a synchronisation message at the application layer of one participating node to the end of exchange of the same message at the application layer of a second participating node (Pullen and Wood, 1995), while **jitter** is defined as the variation of latency with time (Blow, 1998).

A network link's **bandwidth** is a measure of the maximum throughput of traffic on that link. In the event that the data being transmitted over a link exceeds the available bandwidth, messages will need to be buffered or dropped until the flow of data decreases to being within the available bandwidth again (Roehle, 1997).

For real-time DIAs, complete and exact consistency of the state variables stored within the shared world database is impossible to achieve, due to the limitations of the network connecting the participating hosts. The *Consistency-Throughput Tradeoff* (Singhal and Zyda, 1999) describes this issue:

> *It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state.* (2.1)

In simple terms, this statement means that a DIA can be either consistent, or real-time, but not both (Bhola et al., 1998). For example, modern DIAs operate at an update rate of 30-60Hz. In the case of an application with an update rate of 50Hz, each host updates their local state every 20ms. Thus if the latency incurred by an update is greater than 20ms, as would typically be the case with latency on the Internet, the local state of the variable(s) being updated may have already changed before the update is even received.

If the application were to operate at a lower frequency such as 5Hz, then the variables could only be changed every 200ms, and consistency could be achieved for latencies less than 200ms. However, a 5Hz simulation is far less interactive in real time than the 50Hz simulation.

In more general terms, Singhal and Zyda (1999) propose that equation (2.2) can be used to evaluate the scalability of, or amount of resources required by, an application.

$$\text{Resources} = M \times H \times B \times T \times P \tag{2.2}$$

where:

M is the number of messages exchanged;

H is the average number of destination hosts for each message;

B is the bandwidth required for a message;

T represents the timeliness with which the packets must be delivered to each destination;

P is the number of processor cycles required to receive and process each message.

In equation (2.2) the *M*, *H*, and *B* terms refer to the communications bandwidth required by the DIA, and thus influence the impact that network bandwidth has on the scalability of an application. Reducing the size of one or more of these terms has the overall effect of reducing the network bandwidth required by an application. Several techniques that have been employed to accomplish this are described in Section 2.6.

The impact of network limitations on consistency can manifest itself in a number of ways, which vary depending on the nature of the simulation. The effects can be categorised into one of three groups (Sun et al., 1998):

1. **Divergence** – Events arrive and are executed at hosts in different orders at different hosts, leading to differing final results.
2. **Causality Violation** – Due to non-deterministic network latency, changed to state variables arrive out of their natural cause-effect order.
3. **Expectation / Intention Violation** – Due to concurrent generation of updates on different hosts, the actual effect of an event at its time of execution may differ from the intended or expected effect of the event at its time of execution.

For the purposes of this thesis, the following definition of consistency, which takes into account the real-time nature of modern DIAs (Delaney, 2004), is adopted:

Consistency is the maintenance of a uniform dynamic shared state across all participants in a DIA. Def. (2.3)

2.5.1 Metrics of Consistency

Many DIAs are state driven applications, and thus a common metric for divergence is spatial inconsistency, or drift distance, defined in the literature as a measure that represents absolute spatial difference between representations of the same entity on different hosts (Diot and Gautier, 1999). When using this metric, the state of the DIA is judged to be inconsistent if the drift distance measured is large when compared to some threshold value.

A contrasting measure of consistency is that of temporal consistency, or phase difference, which measures the impact of network latency by measuring the length of time between generation of an event to a state variable, and the same event's application to the remote version of that variable (Lui, 2001). In using this metric, a threshold is defined in relation to the phase difference, and the goal of the DIA is to keep the absolute value of the phase difference less than or equal to this threshold.

More recently, Zhou et al. (2004) has observed that a limitation with both of these metrics is their failure to consider other concerns and influences. For example, spatial inconsistency does not consider the duration for which an inconsistency persists, while temporal inconsistency does not consider the effect of the inconsistency on the world state. Arising from this, Zhou et al. (2004) have proposed a measure that combines both time and spatial measures into a single metric of inconsistency for a continuous variable

$$\Omega = \begin{cases} 0, & \text{if } |\Delta(t)| < \varepsilon \\ \int_{t_0}^{t_0+\tau} |\Delta(t)|\, dt, & \text{if } |\Delta(t)| \geq \varepsilon \end{cases} \tag{2.4}$$

where:

$\Delta\ (t)$ is the spatial difference between entity representations on different hosts;

ε is the minimum perceivable error (from a human-user perspective);

t_0 is the time at which the difference starts;

τ is the duration for which the difference persists.

From this definition, it can be observed that Ω refers to the area under the graph of spatial consistency with respect to a particular entity trajectory over a specific duration of time. When $\Omega = 0$ absolute consistency has been achieved to within the limits of user perception, and the presentation element of inconsistency is minimised (Vaghi et al., 1999). Due to limitations of any network connecting participants, a true zero value for inconsistency is never actually achievable in a real-time DIA, as per the Consistency-Throughput Tradeoff. To account for this, Zhou includes the ε term as a minimum perceptual value in equation (2.4) above. This allows a system to be considered consistent if the inconsistency is below human perceptual limits.

Physics-aware inconsistency measurement

As the algorithms proposed within this thesis pertain to synchronisation of state variables rather than latency masking or other temporal concerns, for the purposes of this work, a spatial metric of consistency will be employed when presenting and considering results. Due to the simulation of non-peer-controlled entities, and a need to consider their effect on the environment as a whole, drift distance as defined above is insufficient in measuring the total inconsistency in a physics-aware environment. The term physics-aware inconsistency will be used to refer to the sum of the spatial inconsistencies of all entities between one host originating stage changes, and another receiving them. This is represented by equation (2.5).

$$\phi = \sum_{i=1}^{N} e_i \tag{2.5}$$

where:

e_i is the spatial inconsistency of the i^{th} entity in the environment;

N is the number of entities present in the environment;

ϕ is the "physics-aware" inconsistency in the environment.

The next section introduces a number of mechanisms commonly used in DIAs to reduce the amount of network traffic, thus increasing the scalability of applications. The specific impact and relevance of these mechanisms in the area of physics-aware DIAs are also outlined in each case.

2.6 Traffic Reduction techniques and their implications for physics-rich environments

In order to improve the scalability of DIAs, a number of techniques have been developed, and the implementation of some of these techniques can have specific relevance to physics-aware DIAs. A number of these techniques are outlined in the following section, as well as the implications of using these techniques in such applications.

2.6.1 Compression

Compression is used to reduce the volume of traffic generated by reducing the size of network packets. There are various compression schemes available for use in DIAs, all of which can be categorised as either lossless or lossy. Lossless compression techniques, as the name suggests, are those which still result in fully precise information being exchanged. An example of a lossless compression scheme is bit packing, as used in the Torque and Tribes engine (Frohnmayer and Gift, 2000), which involves including only the bits necessary to represent the contents of a variable, as opposed to all the bits in the variable. For example,

many integer values can be represented with far less than 32 bits, despite occupying 32 bits in memory.

Conversely, lossy compression algorithms operate on the principle that fully precise information is not necessary. Truncation or rounding of floating point values would be an example of one such approach, and dead-reckoning, as outlined below, is also analogous to lossy compression. The use of such compression schemes may be undesirable in physics-aware simulations, as it potentially negates the determinism of an engine to send state updates with lossy compression due to initial conditions differing. It may be possible to avoid this discrepancy by applying the same compression to internal variables within the engine at the time of updating, but blindly modifying state in this manner has the potential to introduce unpredictable behaviour, e.g. two bodies being brought into contact/overlap.

2.6.2 Packet Aggregation

Packet aggregation refers to a process by which multiple packets are combined into a single larger packet in order to reduce network bandwidth (Bassiouni et al., 1997). This reduction is accomplished by each smaller packet sharing the single header of the larger packet, thus reducing the overall number of headers transmitted. A drawback of the approach in all simulations, particularly physics-aware applications which may be particularly time sensitive, is the potential for latency to be introduced by delaying transmission of packets, for the sake of aggregation.

2.6.3 Dead Reckoning

An already mentioned but as yet unexplained algorithm for traffic reduction, dead reckoning is a short-term client side prediction mechanism (Durbach and Fourneau, 1998, IEEE, 1998). This is an approach which utilises information relating to the dynamics of an entity's state and motion, such as position and velocity, to model and predict future behaviour. The operation of the mechanism is as follows:

(i) All users model all entities, including those local to the user.

(ii) Thus each user knows both the local and modelled behaviour of local entities, and can compare them at all times.

(iii) Local users send updates to remote users when they determine that the error between modelled and actual behaviour has exceeded a preset threshold. A first order example of this is shown in Figure 2.4.

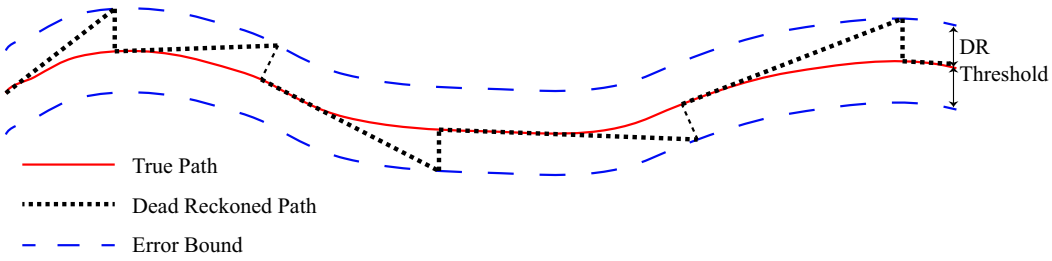


Figure 2.4: Calculation of a first order dead reckoning model from true motion of an entity

Depending on the error threshold employed, it is possible, and indeed likely, that the actual and modelled entity positions will differ somewhat. As a result, when updates are received, modelled remote entities may appear to jump, or "snap" to the updated position. Convergence mechanisms have consequently been employed to smooth this jump (Singhal and Zyda, 1999), with a good convergence algorithm being capable of correcting the modelled behaviour quickly, without presenting too distorted a world view to the user. Equation (2.6) gives an example of how a first-order dead reckoning update can be calculated

$$\mathbf{p}_{t_0+1} = \mathbf{p}_{t_0} + \mathbf{v}_{t_0}\tau \tag{2.6}$$

where:

$p_{t+\tau}$ is the position of an entity at time $t_0+\tau$;

p_t is the position of an entity at time t_0;

v_t is the velocity of an entity at time t_0;

τ is the timestep of the simulation;

The error permitted by dead reckoning for remote entities presents an issue in physics-aware applications as if an error is present when a modelled remote

entity collides with an environmental entity there is automatically an error present within the collision simulation, and potentially in the path followed by the environmental entity before coming to rest.

2.7 Summary

In this chapter, previous work in the areas of DIAs, real-time physics simulation and immersion in games and interactive applications was detailed. The developments in incorporating physics simulation into DIAs were also outlined, as well as the motivations behind this. In the next chapter the development of algorithms for supporting distributed simulation of physics in a peer-to-peer application utilising dead reckoning traffic reduction mechanisms is discussed.

Chapter 3

Physics-aware state management in a P2P DIA

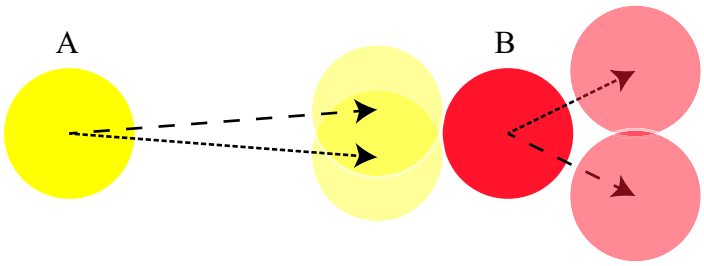
In this chapter a mechanism for synchronisation of entity state in a physics-aware, peer-to-peer DIA is outlined. The development of this algorithm both draws on past work in the field, and incorporates a novel means of minimising physics-consistency-cost. Further, the development of a testbed application for validation of this algorithm is described.

3.1 Physics-consistency-cost

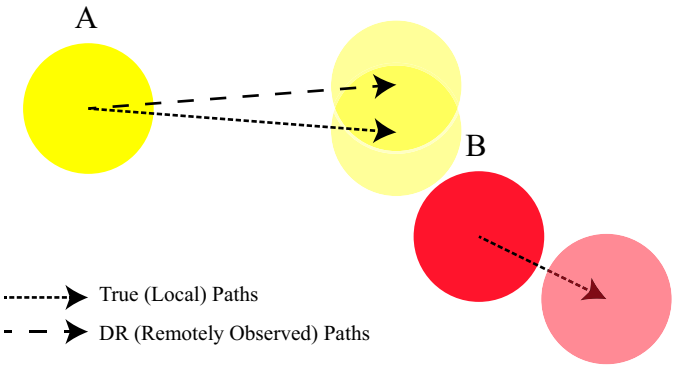
As already mentioned in earlier chapters, physics-aware DIAs introduce a new element to consistency, by the simulation of physics-aware entities. Such entities are capable of having the inconsistency present in peer-controlled entities communicated or transferred to them if the controlled entity's state is inconsistent when they interact, or collide.

Figure 3.1 (a)-(c) illustrates the means by which the inconsistency present in the state of an entity modelled by dead reckoning may be communicated to a physics-aware entity by means of a collision between the two entities. Broadly speaking, such inaccuracies fall into one of the following three categories:

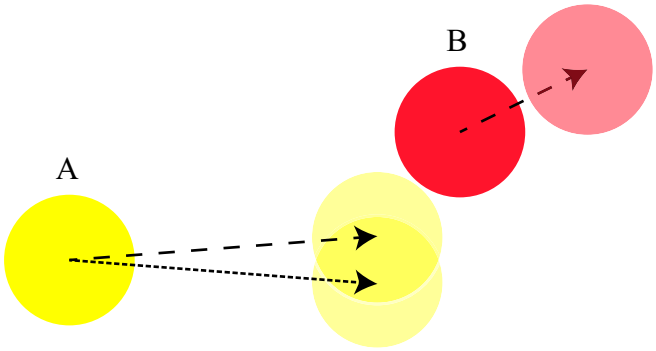
- collisions detected by both local and remote hosts, but simulated inaccurately at the remote host due to incorrect velocities and points of contact as illustrated in Figure 3.1 (a),
- collisions occurring in local world that did not occur remotely (Figure 3.1 (b)), and
- collisions that did not occur locally, but were observed remotely (Figure 3.1 (c)).



(a) Collision simulated inaccurately by remote host due to incorrect point of contact



(b) Collision simulated locally that was undetected at remote host



(c) Collision simulated at remote host that never occurred locally

Figure 3.1 (a)-(c): Examples of collisions between controlled entity (A) and physics-aware entity (B) simulated inaccurately at a remote host due to error in dead reckoning model of controlled entity's motion. Solid shapes represent starting positions, and shaded shapes represent projected positions.

In a similar manner, in a physics-rich environment with densely clustered entities, these inconsistent physics-aware entities can in turn communicate their inconsistency to other such entities. This communicable inconsistency is the source of the difference between the solid and dashed lines in the graphs presented in Figure 3.2, and this difference is what is meant by the term "physics-consistency-cost" in the context of this thesis. At certain times this is

zero (when the two lines are overlaid), but when collisions occur, it typically becomes non-zero (when the two lines diverge).

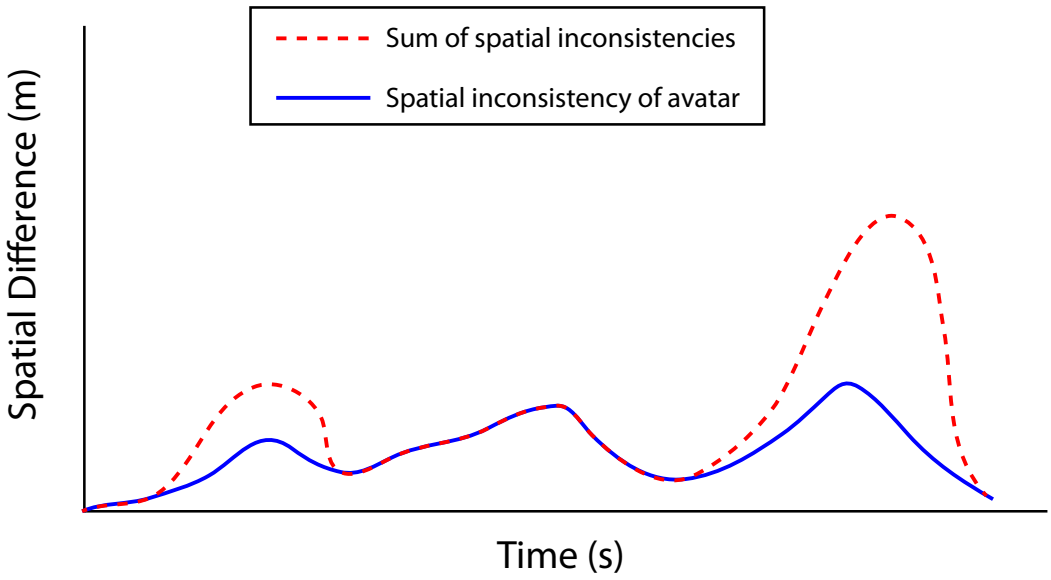


Figure 3.2: Controlled entity (avatar) inconsistency compared to all inconsistencies present in a physics-aware DIA.

While this thesis limits the definition of physics-consistency-cost to this discrepancy, there is scope to consider further factors. To consider the ball and domino examples presented in Chapter 1; in calculating physics-consistency-cost as a sum of spatial inconsistencies as outlined above, the ball example could generate a significantly larger physics-consistency-cost than the domino example, due to the distances involved. However, the domino example requires the correction of more entities if they are simulated inaccurately as falling. To account for this, a complexity factor could be introduced to the calculation of physics-consistency-costs.

Yet a further variation in calculating this could be to consider the relative magnitude of a physics-aware entity's inconsistency compared to its size. Considering the ball example and the large inertia example (also in Chapter 1) previously presented; if the physics-aware entity in each example had a similar inconsistency in their states, the physics-consistency-cost of the latter would be less under these terms, as the difference in positions might be less noticeable for a larger entity.

The above list of potential physics-consistency-costs is not exhaustive, but rather seeks to highlight the generality of the concept, and to point out that the exact means of calculation may need to be chosen according to the requirements or priorities of a particular application.

3.1.1 Anticipation of Physics-consistency-costs

Having identified the source of these physics-consistency-costs, it is now possible to try to identify criteria for predicting, or more accurately anticipating, times at which they may arise. This distinction is made as many, if not most, collisions, are impossible to predict with both 100% certainty (that they will occur, and when) and accuracy (what changes of state will result from the collision).

One means of anticipating collisions between a locally-controlled entity and a physics-aware entity is inspired by the dead reckoning algorithm as outlined earlier. Specifically, this would be accomplished by trying to predict the future states of the environment based on current and/or past states, and then checking for collisions between the locally controlled entity and any physics-aware entities in each future state. This could be accomplished by copying the environment within a certain radius of the controlled entity, and advancing the copied simulation a set number of steps, checking for collisions between steps. This is referred to as a "forecasting" method in the context of this thesis.

An alternative means would be to consider the density of entities either in the immediate vicinity of a controlled entity, or along its projected path. In general the likelihood of physics-consistency-costs arising (i.e. collisions occurring) is dependent on the number of entities in the environment. It should be intuitive that for two environments, identical in every way except for the number of physics-aware entities present, there is a greater likelihood of colliding with a physics-aware entity in the environment with more entities. Similarly, if a user is in a region of an environment that is densely populated with physics-aware entities, they are more likely to collide with such an entity than another user in a region that is sparsely populated with, or empty of, physics-aware entities. In

this thesis, approaches that examine the controlled entity's surroundings for physics-aware entities in this manner are categorised as being based on "entity-density."

3.2 Authority in Physics-aware DIAs

In non-physics-aware, peer-to-peer DIAs, each peer maintains the authoritative copy of state for the entity or entities that are controlled by that peer, i.e. those entities that are local to that peer. These entities could be controlled by users at that peer, by an artificial intelligence process running at that peer, or some similar mechanism.

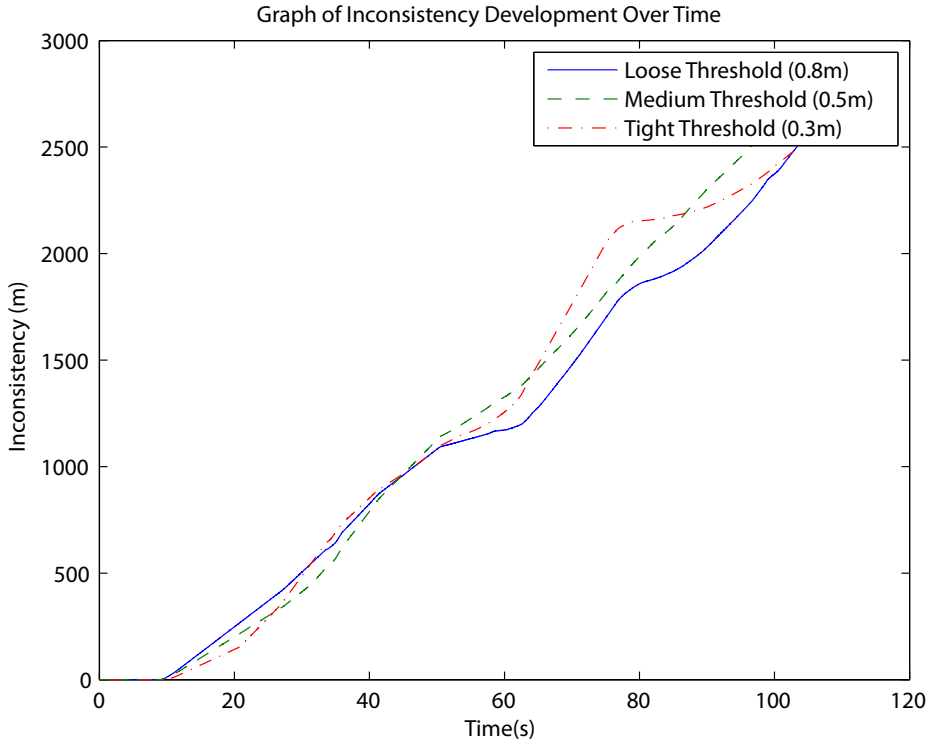


Figure 3.3: Uncontrolled inconsistency developing in a peer-to-peer physics application with no authority for physics-aware entities

In the case of physics-aware DIAs, however, authority is less clear-cut for physics-aware entities. Each peer needs to be able to present a realistic view of physics-governed interactions to its users, and due to the dynamic nature of physical interactions, this imposes a requirement of responsiveness on the

application. A truly responsive simulation could be achieved by giving each peer responsibility and authority over its own copies of physics-aware entities. However, this would lead to quickly developing inconsistency, due to the factors outlined in Figure 3.1. While each peer could accurately simulate the interaction of their local entity or entities with physics-aware entities, the error in the dead reckoning models they provide to remote peers would lead to inaccurate calculation of the resulting state of the physics-aware entities. Such a development of physics-aware inconsistency is demonstrated in Figure 3.3. This graph was generated by running a set of simulations with no explicit synchronisation of the state of physics-aware entities, and recording the development of physics-aware inconsistency. Note also that the size of the error threshold used in the dead reckoning has no bearing on the rate at which the physics-aware inconsistency develops.

Thus it is clear that an explicit means of synchronising the state of these entities is required, and in order to synchronise state, an authority scheme is required. In the context of peer-to-peer DIAs, Fiedler (2010b) describes authority as "like being the server for an object" whereby a peer having authority over an object means that it can inform other players of the object's state. Authority is necessary for two reasons:

1. To determine which peers have an inconsistent state representation, and
2. To correct the state of the inconsistent entities at these peers.

3.2.1 Static Authority

In traditional peer-to-peer DIAs, each peer is authoritative on the state of its local entities. This authority can be considered as being a "static" authority, in that it is assigned when the peer joins the DIA, and is not changed or reassigned after this point. Conversely, if the authoritative peer for a given entity can be changed while a DIA is running, that could be considered dynamic authority.

As noted earlier in this thesis, in DIAs using client-server architectures the server acts as a single, central authority, with clients (the equivalents of peers) receiving

appropriate state updates from it. Hypothetically, authority over physics-aware entities could be granted to a single peer, with all other peers having to validate or seek approval for interactions between their local entities and physics-aware entities. This would, however, place a significant burden on that one peer, effectively turning it into a server for physics simulation. Additionally, with a single peer being authoritative over all physics-aware entities in the environment, the application becomes less fault-tolerant as the entire physics simulation is dependent on that peer.

The flaws of the previously outlined approach could be mitigated somewhat while still maintaining a static authority scheme by allocating authority over particular entities to particular peers. In essence, this would mean the division of physics-aware entities among the peers, with each peer managing the entities allocated to it as local entities, according to its own physics simulation, and dead reckoning models being used by peers for other physics-aware entities. This static distribution of authority presents issues of its own, however. If a user-controlled entity interacts with a physics-aware entity whose authority is held remotely, their interaction or collision must be "approved" by the remote authority peer. In the presence of latency, this can present unrealistic behaviour to a user, whereby they observe a visible delay between their collision (cause) and the physics-aware entity's resulting movement (effect). Note that this would also apply to the previous scenario of a single authoritative peer. Additionally if the physics-aware entity contains modelling error, the result of an interaction may also differ from the user's expectation.

3.2.2 Dynamic Authority

To reiterate the earlier distinction, dynamic authority refers to being able to change which peer provides authoritative state for an entity while the application is running. Motivated by the earlier work of Fiedler (2010b), this thesis proposes a dynamic authority scheme whereby a peer assumes authority over entities with which it interacts. This removes the need for a peer to remotely validate interactions of its entities with physics-aware entities who might otherwise have had remote authoritative peers.

Under such an authority scheme, each peer would be responsible for notifying other remote peers of the resulting state from interactions between their local entities, and physics-aware entities. Under the authority scheme proposed by Fiedler (2010b), each peer takes authority of entities with which their local entities interact, and remain as the authority for that peer until it comes to rest again following the collision. By the author's own admission, a goal of this mechanism is to mitigate the effects of non-deterministic physics simulation, and its utilisation in an application with constant update rate also means that it does not cause an increase in bandwidth, as each entity's state updates are simply transmitted when the peer can accommodate them.

In contrast to this previously detailed approach, this thesis is distinct in utilising dead reckoning models for controlled entities, rather than a constant update rate. A requirement for the use of a deterministic physics engine or simulation at each peer is also imposed. In light of this difference, an alternative approach to providing updates of entity state is employed:

1. Controlled entities are simulated using dead reckoning models as previously outlined.
2. In the event of a controlled entity interacting or colliding with a physics-aware entity, the resulting state of both the controlled entity and the physics-aware entity is supplied by the local peer to all remote peers. Deterministic physics simulation at each peer means that once supplied with the state immediately after the collision (i.e. initial conditions), the simulation of the playout at each peer will result in the same final state.
3. Collisions between modelled remote entities and physics-aware entities are ignored unless a collision notification, or state update, for the physics-aware entity is received. Figure 3.4 shows an example of how the dead reckoned model of an entity, as observed by a remote user, can appear to collide with an entity while the true path of the entity avoids it.

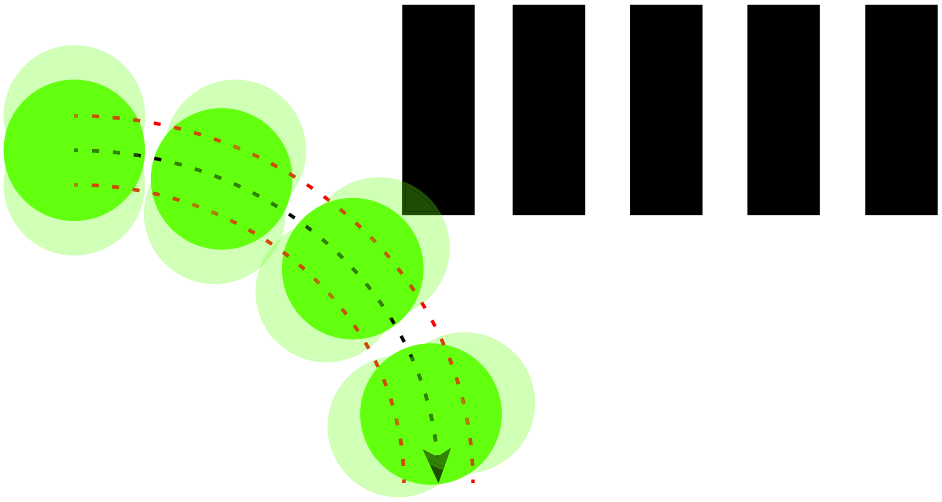


Figure 3.4: Spherical avatar (opaque) passing close to row of dominoes moves along black dashed path. Shaded circles represent the model's maximum divergence at each point.

This approach depends on updates of physics-aware entity state being sent by reliable means, but Fiedler (2010b) acknowledges that packet loss in modern Internet applications is in fact quite rare, and schemes do exist that can be used to add an element of reliability to data sent using the User Datagram Protocol (UDP) (Fiedler, 2008). An alternative approach to point 3 above would be to record collisions that have been simulated with no notification, and restore the state of any environmental body disturbed in the event of a collision update not being received. However this approach risks implementing corrections with significant visual impact in the environment, especially if disturbed entities collide with other entities. Ultimately this is the difference between adopting a pessimistic/conservative approach to consistency (the former), and an optimistic or aggressive approach (the latter) (Bhola et al., 1998, Cronin et al., 2002, Greenberg and Marwood, 1994, Jefferson, 1990, Vaghi et al., 1999), with the conservative approach being proposed in this thesis.

3.3 Updating state for physics-aware entities

Initially the possibility of providing a single update at the instant of contact was investigated, with some success. However, this still proved to be unreliable and insufficient, with inconsistency manifesting after collisions. Upon further investigation, it was determined that many collisions "persisted" beyond a single simulation frame. Consequently, the potential for user input over the course of

the collision meant that the playout was not guaranteed to be deterministic after the initial contact. If a user were to change their speed or direction slightly during the collision, they could potentially remain within the threshold of the dead-reckoned model, and thus not generate an update. Traditionally this would not be an issue in peer-to-peer simulation, but with the sensitivity of physics engines to variation in initial conditions such slight changes were still a source of significant inconsistency.

At this point, three further options were identified to try to resolve this issue:

- Allow and require peers to update their controlled entity and the colliding body for all simulation ticks during the collision.
- Allow and require peers to update the state of their avatar and the colliding body once the collision has ceased, i.e. the bodies are no longer in contact.
- A hybrid of both of the above.

The theoretical merits and flaws of all three systems were identified, and are outlined below.

Updates throughout collision

A system whereby a peer transmits state updates for the physics-aware entity for all simulation ticks that they are in contact would result in a very accurate simulation of the collision, but could potentially result in a large volume of unnecessary traffic. Consider a scenario where a user is required to push an object to an arbitrary location in an application with an update rate of 50Hz, a common update rate in a modern DIA. This could result in up to fifty updates being generated each second, depending on the simulation frequency, with many potentially being sufficiently small as to be imperceptible to a user, and hence unnecessary. This is less of a concern in applications than it has been historically, with increased availability of bandwidth, but still needs to be considered as an influence on the scalability of an application. Additionally, if updates are only transmitted while contact persists, the outcome of the contact

may still be simulated inaccurately, depending on the user input sampled during the final step, when the bodies move apart.

Updates at beginning and end of collision

This method could be considered to have almost the opposite emphasis of the above, with less priority placed on precision in the course of the collision, in favour of ensuring consistency in the aftermath, as well as limiting network traffic. Essentially the collision as an event would be "framed" by two state updates of both the avatar and the environmental body involved. The first is generated at the moment of contact, to ensure a reasonably accurate initial simulation of the collision. Once the collision has ended, any inconsistency introduced in the course of the collision would be corrected by an update generated in the following simulation frame, as the motion of the environmental body will be deterministic once the user's influence has ceased, and this update has been applied. Despite there not being explicit updates made to the state of the environmental body in the course of the collision under this system, there is still a means of controlling the inconsistency which may arise, as standard dead reckoning techniques will still be applied to the avatar.

Hybrid system

As the description suggests, this system would combine elements of both of the previous methods, specifically by utilising the framing updates of the second system, in conjunction with the constant updates of the first. This would provide for accurate simulation of collisions from start to finish, as well as accurate environmental playout when the bodies have moved away from each other. However, this approach also presents the same drawbacks as the first one of generating more traffic than may be necessary, or indeed manageable in larger applications. Due to an inherent assumption here that the simulation will always "play out" the same way for a set of conditions, this approach is highly dependent on the determinism of the physics engine utilised in the application, as described in Section 2.3.2.

Under all three of these systems hosts should not model collisions of modelled remote entities with environmental objects without notification of that collision

having occurred. Were a host to model without such notification, it would risk simulating a collision that had not occurred, and only appeared to do so due to the error present in the dead reckoning model. Many physics engines provide a means of collision filtering which can be used to implement this, whereby detected collisions of modelled remote entities are ignored so that the physics-aware entity remains undisturbed.

An alternate approach (not implemented here) to this would be to record collisions that have been simulated with no notification, and restore or "roll back" the state of any physics-aware entity disturbed in the event of a collision update not being received. However this approach risks implementing corrections with significant visual impact in the environment, especially if disturbed entities collide with other entities.

Preliminary experiments with the testbed as described in Section 3.6 were conducted to record the duration of collisions in the two-dimensional environment supported by the testbed, and it was found that the majority of them lasted only a short time, typically less than 6 simulation ticks, or 120ms. With this being the case, it was observed that with an update being provided at the start of each collision, any given collision would only be capable of introducing a minute inconsistency (related to the velocities, inertias, and potential for acceleration of the bodies involved) before the collision ceased, and another update was generated. Thus it should be expected that for the authority scheme as proposed with no latency, the inconsistency relating to environmental bodies should be limited to a series of small "blips" or peaks that arise while collisions are taking place.

3.3.1 Collisions by proxy

A shortcoming of the peer updating entities with which its controlled entity collides directly has been identified. Specifically, in the case of "collisions by proxy", the collision was not necessarily modelled correctly. Figure 3.5 below illustrates a collision by proxy, where in the course of the user's (circle A) collision with an environmental body (circle B), the body in turn touches a

second environmental body (circle C). In this instance, the ultimate path of body C will be inaccurately modelled, as during the collision, the bodies A and B's paths are approximated. The motion of the latter two bodies will be corrected once they separate, but at this point the remote state of body C will be inaccurate, and will remain inconsistent.

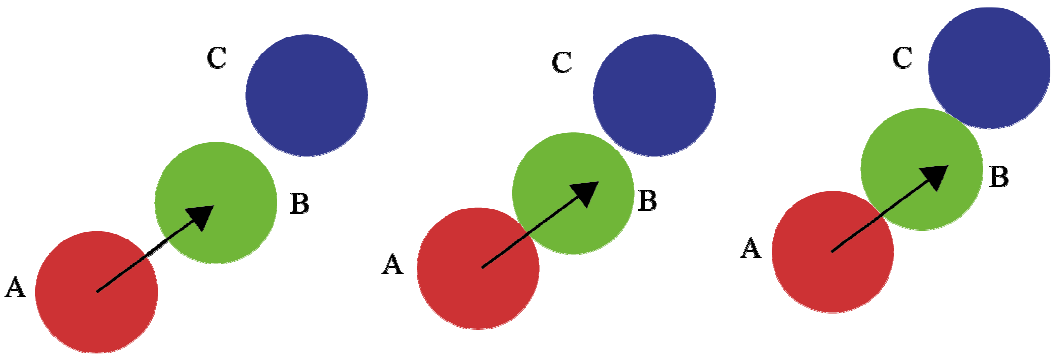


Figure 3.5: In a collision by proxy, entity A (user-controlled) collides with entity B (physics-aware), and during this collision, entity B contacts entity C (also physics-aware)

A proposed and intuitive solution to this is to recursively check both the algorithm and bodies that it is in contact with for collisions. That is to say that on each tick, if the controlled entity is in contact with another body, that body is checked for contacts, and if any are found, the other bodies in the collision are checked, and so forth. A record is kept of these contacts, and if a contact ends, then any bodies no longer in contact are updated. If the controlled entity loses its contact with the first object, then all objects can be updated, as the playout will be deterministic with the possibility of user input having been removed from the system.

3.4 Minimising physics-consistency costs

With an authority scheme for physics-aware entities in a DIA with dead reckoning models of controlled entities, variation of physics-aware inconsistency similar to that illustrated in Figure 3.2 is achievable. The next step was to try to minimise the difference between the two measures of inconsistency in this graph, specifically by minimising the amount by which the physics-aware inconsistency increases.

It has already been observed that this discrepancy, or physics-consistency-cost, develops as a result of the inconsistency in controlled entities being transferred to physics-aware entities. An authority scheme as outlined above ensures that the two different measures of inconsistency reconverge, and that the inconsistency introduced to the physics-aware entities is controlled, in the same manner that dead reckoning permits a controlled level of inconsistency in entities.

In Section 3.1.1 the subject of anticipating physics-consistency-costs is discussed, and suggestions as to how to achieve this are detailed. In this thesis, a forecasting method will be utilised to try to anticipate the occurrence of physics-consistency-costs. Since physics-consistency-costs arise as a result of the error in the dead reckoning model of entities, it is proposed that peers should attempt to minimise the error present in these models at the time of collisions. As has previously been explained, any dead reckoning model has an error threshold, and when the error present in the model compared to the true state exceeds this threshold, an update is generated. Consequently a model with a variable, or adaptive, threshold is proposed. Adaptive threshold dead reckoning models have been described in the literature previously (Cai et al., 1999, Lee et al., 2000), but in this instance it is specified that the threshold be varied in response to physics considerations (i.e. a physics-aware adaptive threshold algorithm) where previous applications have sought to reduce the number of updates generated by an application. The combination of this adaptive-threshold model with an authority scheme for physics-aware entities is the main proposal of this thesis, and the function of such a combined algorithm is detailed in the following section.

3.5 Adaptive threshold DR with Authority

In this section, the combination of a dynamic authority scheme for control of physics-aware entities, and the use of a physics-aware, adaptive threshold dead reckoning model for peer-controlled entities are proposed. In implementing this, the physics-awareness of the model threshold comes from the use of a forecasting method to anticipate physics-consistency-costs.

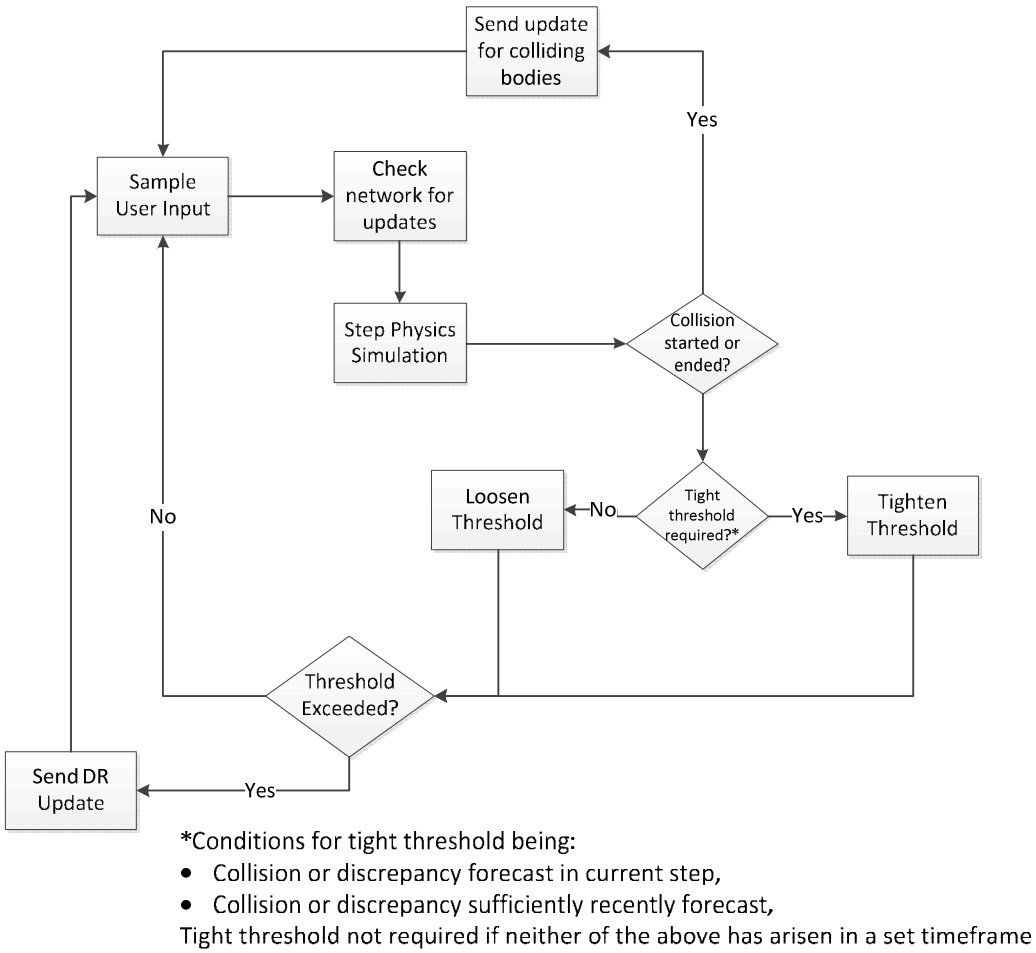


Figure 3.6: Flow of execution for proposed authority scheme with adaptive threshold dead reckoning and a forecasting mechanism for anticipating physics-consistency-costs.

Figure 3.6 illustrates the process followed by each peer for the purposes of state management under the proposed state management mechanism. Considering each loop to commence at the "Sample User Input" block, the sequence of steps would be as follows:

- Sample the user's input from their input device(s) (e.g. keyboard/mouse), and apply these inputs to the user's controlled entity.

- Check the network connection for state updates from other peers yet to be applied to the local world database.
- Advance the local world simulation by one frame based on existing state, and the newly applied information (inputs and updates).
- Check the controlled entity's list of contacts to see if it has begun a new collision, or ended an existing collision.
    1. If it has, send a state update for both the controlled entity, and the entity that it collided with. Then return to the beginning of the loop.
- Check whether the tight or the loose threshold should be applied for this simulation tick. If the forecast mechanism is to look N ticks ahead then
    1. **Tight threshold** should be applied if:
        - the forecast mechanism detects either a future collision, or
        - the forecast mechanism detects a disagreement between forecasts based on true and dead reckoned state, or
        - either of the above occurred within the previous N simulation ticks.
    2. **Loose threshold** should be used if no collision or disagreement is forecast, and N simulation ticks have elapsed since one was last forecast.
- If the error threshold in use for dead reckoning in this simulation tick has been exceeded, send a state update for the controlled entity to remote peers.
- Return to the beginning of the loop and start again.

For clarity and simplicity of presentation, Figure 3.6 does not include such tasks as rendering the world to the user.

Having outlined a state management algorithm incorporating dead-reckoning models for controlled entities, and a dynamic authority scheme for physics-aware entities, the next section outlines the development of a testbed application used in the testing and validation of the techniques outlined so far.

3.6 Testbed Application

3.6.1 Code Development

Code::Blocks

Code::Blocks (www.codeblocks.org) is a cross-platform IDE for C and C++, with support for multiple compilers, including GCC, Microsoft's Visual C++ and the Intel C++ compiler. The software is available and officially supported for Windows, Linux, and Mac OS X. In addition to code editing and compilation, the IDE provides access to debugging functionality to monitor internal variables during execution. It also serves to organise and manage source files, libraries, and other resources needed for compiling programs.

MATLAB

MATLAB (www.mathworks.co.uk/products/matlab/) is a numerical computing environment that was used to develop functions and scripts for analysis of simulation output. These were used to calculate performance metrics to compare algorithms, as well as to trace the loci of entities over the course of the simulation based on logged traces of their positions and velocities.

3.6.2 Tools and Middleware

Box2D

Box2D is an open-source, two-dimensional physics simulation engine that can be integrated into applications as middleware (Catto, 2007). It is written in C++, and has been ported to a number of other languages and environments including Java, Adobe Flash and C#. In its native form, it has also been incorporated into the Torque 2D game engine, and is used in a number of successful two-dimensional games, including Angry Birds (Kumparak, 2011), and Crayon Physics Deluxe (Catto, 2011). It provides facility for simulation of rigid body dynamics with bodies of various shapes, including convex polygons and circles, as well as applying gravity, friction and restitution forces to the bodies.

3.6.3 Testbed Structure

As a means to both frame the problem, and to measure and compare the performance of algorithms, a testbed application was developed to simulate the function of a physics-aware, peer-to-peer DIA. It was determined that the testbed would have to contain two separate "copies" or instances of a single world, with entity state being shared from one (the "local" environment) to the other ("remote"). The facility to simulate latency and jitter in the connection between these two environments would also be necessary, as these are real world scenarios and problems affecting DIAs.

With this in mind, a `packetPipe` class was implemented, which could be extended and modified, depending on the nature of the updates being transmitted. This class would serve as a link between the two simulation instances, and be capable of simulating the previously mentioned network conditions of latency and jitter.

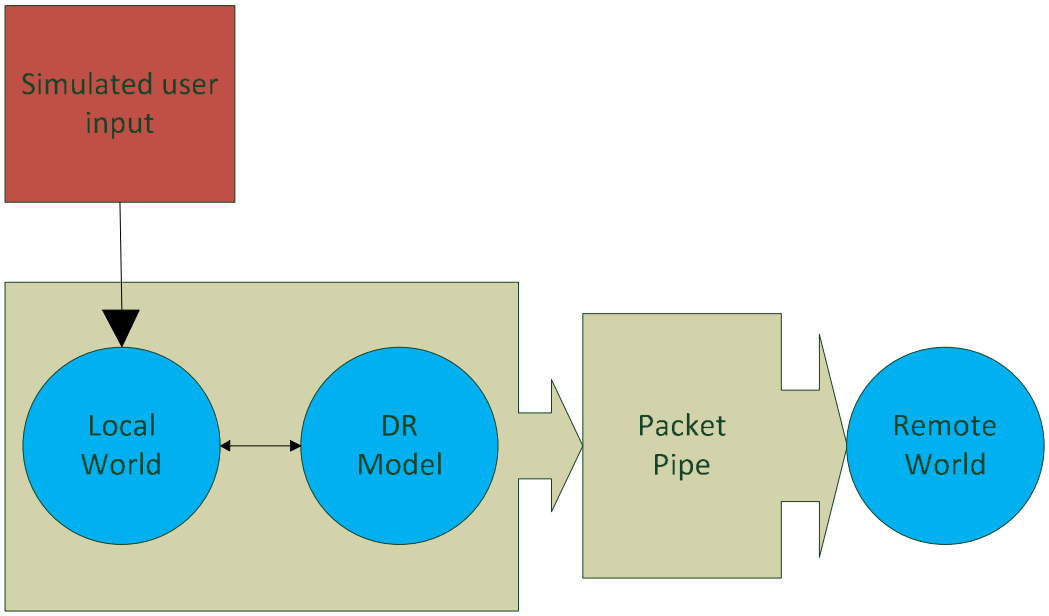


Figure 3.7: Basic structure of testbed for simulating dead reckoning scenarios and algorithms.

For simulation of a controlled entity's motion in the environment, a path was read from comma-separated values (CSV) files, with a force being applied to the controlled entity in order to move it to the next point along its path.

In applying updates to the state of the entity in the simulated remote world, the decision was made to "snap" the avatar to the correct state, with the potential for convergence by means of visual smoothing to be applied at a rendering level (Fiedler, 2006). An alternative approach used in some dead reckoned applications is to utilise a convergence algorithm on the actual entity state (Singhal and Zyda, 1999), but in the context of physics simulation, this was judged to be unsuitable, as it would preserve an error in the simulation, which could cause additional "knock on" effects.

Figure 3.8 illustrates an example of an environment which could be simulated in the testbed application. An arrangement of physics-aware entities (squares) and a single controlled entity (circle) are enclosed within a bounded area, and the controlled entity moved along a path. The physics state resulting from the controlled entity's motion is then simulated by the Box2D engine.

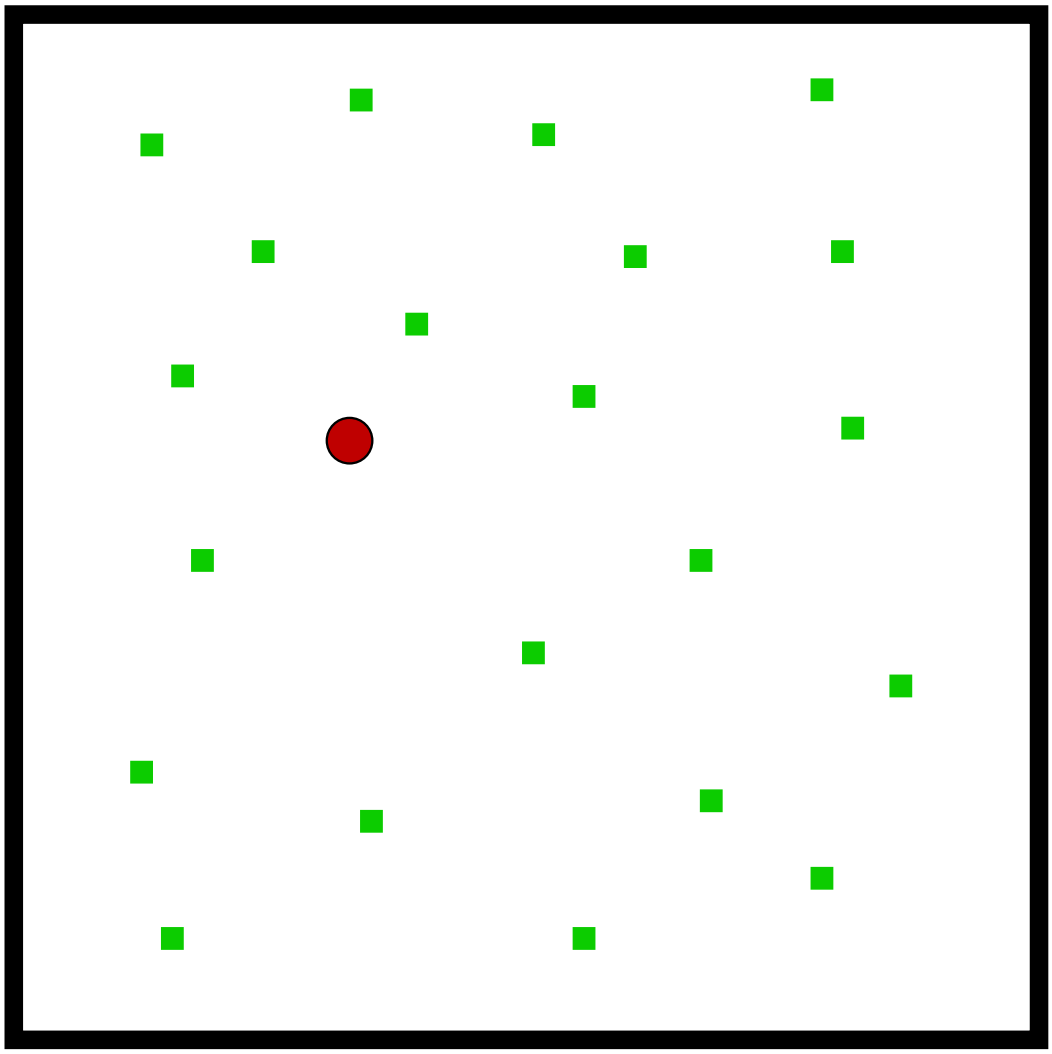


Figure 3.8: Example arrangement of physics-aware entities (squares), and a controlled entity (circle) in a "walled-in" world within the testbed application.

3.7 Summary

In this chapter the development of the algorithms contributed by this thesis was outlined, as well as the development of a testbed for validating and comparing the performance of these algorithms. The specific problems facing a peer-to-peer application utilising dead reckoning were also further detailed.

In the following chapter, the actual testing and validation of these algorithms is described, as well as the performance of these algorithms under both ideal and real-world conditions.

Chapter 4

Results and Performance

Having proposed a physics-aware state management algorithm in the previous chapter, a means of testing and validating this algorithm is now necessary. An application for this has already been described in Chapter 3, but it is now necessary to establish performance criteria for the algorithm. Additionally, the data to be used in the testing is introduced and explained.

Initially all algorithms are compared for performance under the ideal network conditions of zero latency, and consequently zero jitter, with these influences being included in later analyses. The reason for this is to ascertain a limit on the best performance achievable by each method, as a frame of reference when later comparing performance under more realistic conditions.

Both classified types of motion (Lee et al., 2000), and recorded paths from past experiments (McCoy, 2007) will be used in this testing process. Further information about both the classified and recorded motion is given in Section 4.2. Before that, the criteria and metrics for testing are outlined.

4.1 Testing and Performance Metrics

4.1.1 Physics-aware inconsistency, and physics-consistency-costs

Since a goal of the proposed mechanisms is to reduce the physics-aware inconsistency in the application, the first metric examined will be the variation of physics-aware inconsistency (i.e. the inconsistency present in both the user-controlled entity, and the physics-aware entities) over time in the application.

4.1.2 Magnitude of Corrections to Physics-aware Entities

An indirect, but potentially more relevant, means of examining the effectiveness of the mechanism in reducing the physics-inconsistency-costs is to consider the magnitude of the corrections that must be applied to physics-aware entities by the

authority scheme. It has already been stated that physics-consistency-cost is defined for the purposes of this thesis as the sum of the inconsistency present in the states of physics-aware entities. Thus the means of removing, or correcting, these inconsistencies, and the physics-consistency-cost, is the application of updates to physics-aware entities by the authority scheme.

It can be inferred from this that the magnitude of the change in state, or correction, induced in a physics-aware entity is related to the inconsistency present in that entity. In fact, it is the instantaneous inconsistency present in the entity's state at the time that the correction is applied. Thus, the magnitude of corrections applied to physics-aware entities provides a metric of the physics-aware inconsistency present in an application, and consequently the physics-consistency-costs incurred by each interaction or collision.

4.1.3 Update Rate

The update rate of a DIA has already been explained in Chapter 2 as having a bearing on the scalability of the application. Thus the effect of any algorithm on the update rate of an application must be considered. It could be argued that bandwidth has become less of a concern in recent years, as networking hardware has improved, and higher capacity connections to the Internet are provided to users, however, the proliferation of internet-ready devices in homes is again putting pressure on bandwidth requirements. The update rates were calculated by recording the generation of updates in the testbed, and using a sliding window filter to examine the number of updates generated for each simulation tick.

4.2 Motion Data for Testing

4.2.1 Use of Classified Motion Data

From the literature, entity motion in DIAs can be classified according to three descriptive types (Lee et al., 2000): smooth, bounce, and jolt. Each of these types of motions characterises different rates of change of entity state, as well as the nature of the change of the state, e.g. continuous or abrupt. Both of these

elements of an entity's motion are relevant to the generation of a dead reckoning model, as they can affect the frequency and promptness of the generation of state updates. Thus as part of the testing of the algorithms outlined in the previous chapter, simulations were executed for an entity exhibiting each of the three classes of motion individually, in order to identify issues arising in the algorithms from the presence of any single type.

Two-dimensional examples of each of these types of motions are illustrated in Figure 4.1 (a)-(c), to accompany the descriptions, below.

Smooth Motion

Part (a) of Figure 4.1 illustrates an example of smooth motion. Such a two-dimensional example can be produced by an entity moving in a circle of fixed radius with constant angular velocity (Lee et al., 2000). The stated example utilises a period of 32 seconds, and amplitude (i.e. circle radius) of 50 metres, with the circle being centred at the origin, (0,0). In general terms, smooth motion occurs when the entity's velocity may change continuously, but with no sudden, or jump, variation.

Bounce Motion

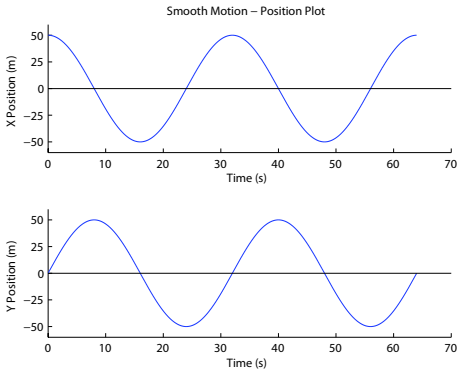
Figure 4.1 part (b) shows a two-dimensional example of bounce motion. This example is produced by a hypothetical entity moving back and forth through a 90 degree arc traced around a fixed centre at the origin, (0,0). The radius of the arc is again 50m, and the period of the motion is 16s. Bounce motion, in general terms, is a motion with sawtooth features as might occur when an entity is involved in a collision.

Jolt Motion

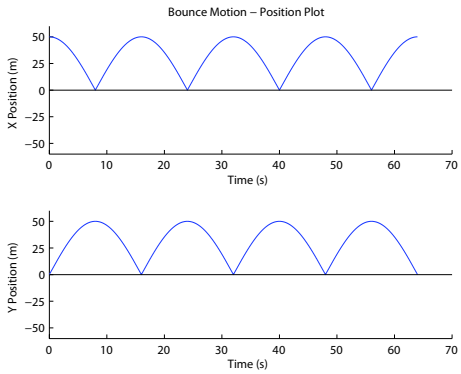
Finally, part (c) of Figure 4.1 shows an example of jolt motion. This trajectory was produced by an entity spinning itself in a circle while moving in a larger circle around a fixed point within the two-dimensional environment (Lee et al.,

2000). The radius and period of the larger circle were 50m and 32s respectively, with the smaller circle having a radius of 10m, and a period of 4s. This is representative of motion with frequent sudden changes of direction.

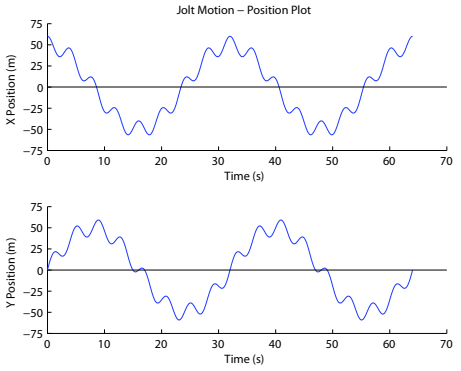
In general an entity's motion will not be classifiable as being excusively of one type, but rather it will contain elements of each. For example, were an entity with smooth motion to be involved in a collision, its motion could exhibit a bounce characteristic as a result of rebounding from the collision. Figure 4.1 (d) shows an example of how the x-value of an entity's position vector might vary over time as it exhibits a mixture of motion types, generated similarly to the the individual traces in Figure 4.1 (a)-(c). However, since each of these classes of motion may present different challenges to a given DIA, it is important to consider each type in isolation.



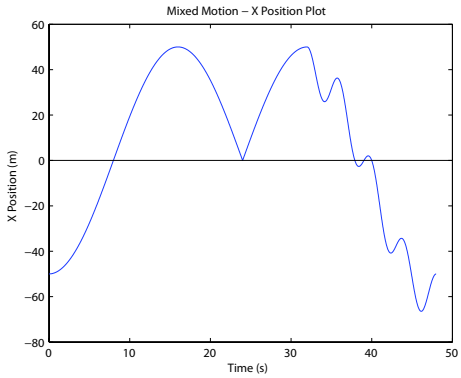
(a) Smooth Motion (Simulation time = 64s, period = 32s, amplitude = 50m).



(b) Bounce Motion (Simulation time = 64s, period = 16s, amplitude = 50m).



(c) Jolt Motion (Simulation time = 64s, period = 32s / 4s, amplitude = 50m / 10m)



(d) X component of the position vector of an entity exhibitin a mix of smooth, bounce and jolt motion

Figure 4.1 (a)-(d): Examples of the three distinct classifications of entity motion (Lee et al., 2000). Simulations were performed at a rate of 50Hz over a time of 50s.

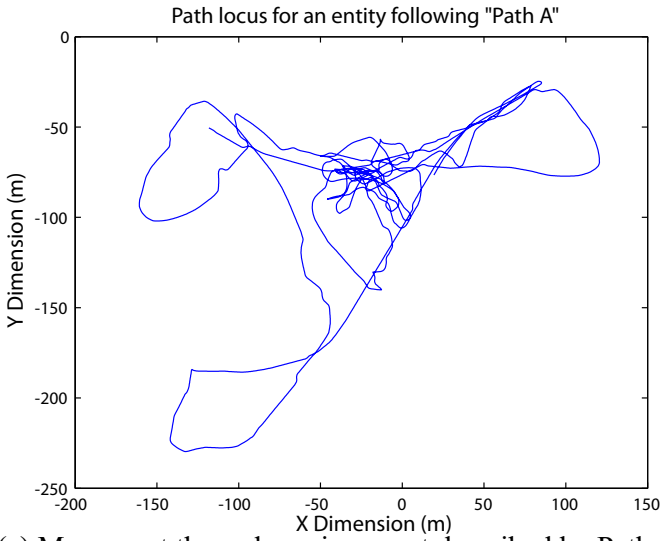
4.2.2 Use of Recorded Motion Data

As already mentioned, the application was also tested for datasets of real motion. These datasets were gathered from previous experiments in the Distributed Applications Group in NUI, Maynooth (McCoy, 2007) using the Torque Game Engine.

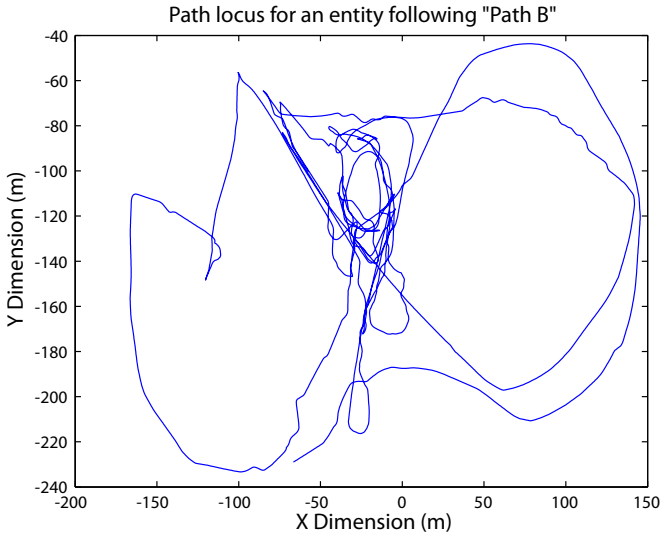
The Torque engine is a three-dimensional simulation engine, originally developed by Dynamix for the Tribes series of games. It was later purchased and released by Garage Games (www.garagegames.com), a company founded by ex-employees of Dynamix. Licensees of the engine get access to both the engine's source code and scripting language (Torque Script), allowing them to modify and adapt how the engine functions which facilitates the implementation and testing of algorithms. Additionally, the engine provides access to content creation tools, and implements networking and rendering, reducing the development workload on licensees. Because of the comprehensive functionality outlined above, as well as the availability of source code to licensees, the engine has been used for a number of experiments within the Distributed Interactive Applications Group in NUI, Maynooth in the past (Delaney, 2004, Marshall, 2004, Marshall, 2008, McCoy, 2007).

The datasets utilised in this set of experiments originally represented motion in three dimensions in a first person shooter application, but for use in the two-dimensional application being tested, the third "Z" dimension was removed. These sets of data were collected for four different environments, or "missions", with several trials of data collected for each mission.

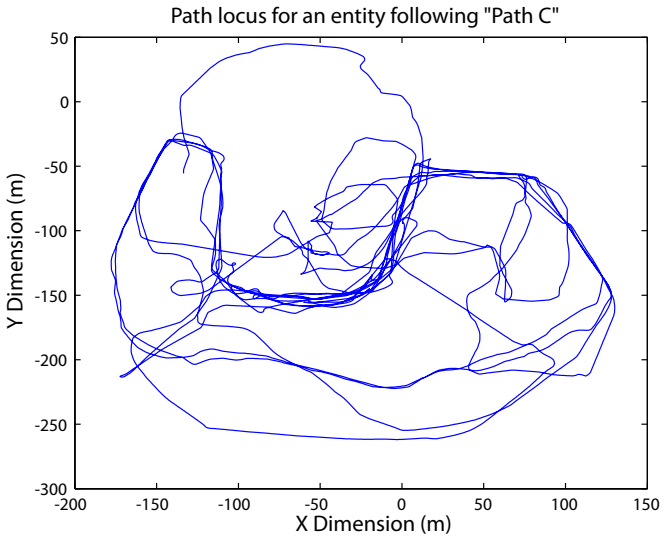
These sets of data, or paths, were stored in comma separated variable (CSV) files, and read by the application at run time. Movement between each point was accomplished by a force being applied to the locally-controlled entity. Three separate paths will be considered in detail, and these are named paths A, B and C. Figure 4.2 illustrates the course that each of these paths follows through the world.



(a) Movement through environment described by Path A



(b) Movement through environment described by Path B



(c) Movement through environment described by Path C

Figure 4.2 (a)-(c): Routes traced through an environment by each of Paths A, B and C

4.2.3 Arrangement of simulated physical environment in testing

For the purposes of comprehensive testing, a variety of arrangements of environmental objects were utilised, but all environment configurations shared a common basis, as outlined:

- The world was a square of side 800m with immovable and non-deformable walls. This was done to contain entities within a finite environment, as would usually be the case in a DIA.
- The locally controlled entity was represented as a circle (two-dimensional representation of a sphere) of radius 2.5m. The modelled motion in the remote environment was also represented by a similar circle.
- The environmental objects were boxes or squares (two-dimensional representation of a cube) of side 6m. For the experiments presented, there were 180 such boxes in the 800m x 800m environment. Taking into account the area of each box, approximately 1% of the environment was occupied by physics-aware entities.

Beyond this, environmental entities (those not owned or controlled by a specific peer and whose mechanics are simulated according to physical laws) were placed in the environment in random positions. To ensure that these random arrangements were identical in the local and remote environments at the beginning of the simulation, the random number generator was seeded before populating the local environment, and then reseeded with the same value for populating the remote environment. Additionally it was possible for this seed to be passed as an argument to the testbed at runtime to run multiple algorithms and entity paths for a single arrangement.

For each of the examples considered in detail here, the same arrangement of physics-aware entities will be used in order to ensure that different sets of data remain comparable.

Having introduced and explained the data, environments and criteria for testing of the algorithm, the next section examines the performance of the authority

scheme alone in conjunction with a fixed threshold. This will serve the purpose of verifying the ability of an authority scheme to control the inconsistency present in the environment, as well as providing a baseline for later comparison of the dead reckoning model with an adaptive threshold.

4.3 Authority Scheme with Fixed Threshold

4.3.1 Requirement for the Authority Scheme

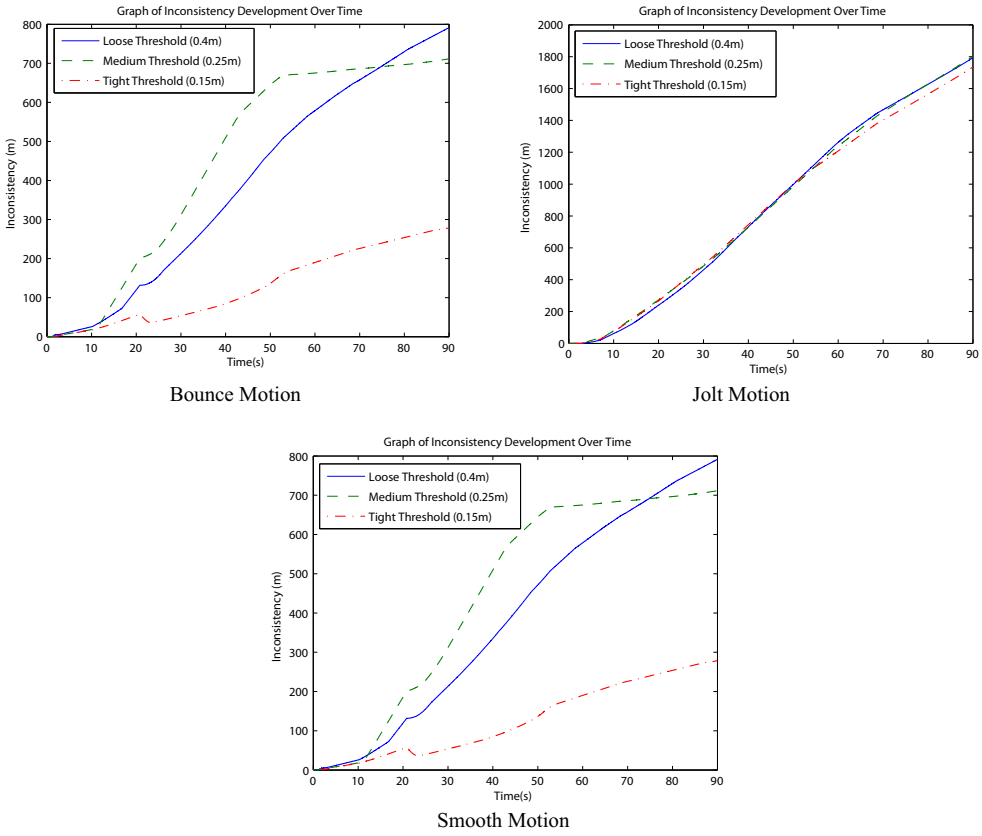


Figure 4.3: Graphs illustrating that collisions occurred for each classification of motion (bounce, jolt and smooth) and that inconsistency would be uncontrolled without an authority scheme

In order to verify the effectiveness of the authority scheme, it is first necessary to prove that the examples being examined have a need of the authority scheme, i.e. that the examples involve the controlled entity colliding with physics-aware entities. This meant that the first set of simulations were run for a variety of fixed thresholds, but with only dead reckoning models of the controlled entity

being supplied to the simulated remote peer. Figure 4.3 and Figure 4.4 show the inconsistency graphs for the classifications of motion, and the recorded paths respectively, and it can be observed that in all cases uncontrollable inconsistency occurs in the absence of an authority scheme. The tendency of these plots to show an increase over time reflects the fact that physics aware entities which have been disturbed are not corrected and therefore errors tend to accumulate.

The data gathered from this set of simulations is not considered or examined in detail, as it represents an unrealistic scenario, whereby entities would go uncorrected, and remain inconsistent. Instead it is presented to illustrate the need for, and scope for improvement by, an authority scheme in the examples to be considered.

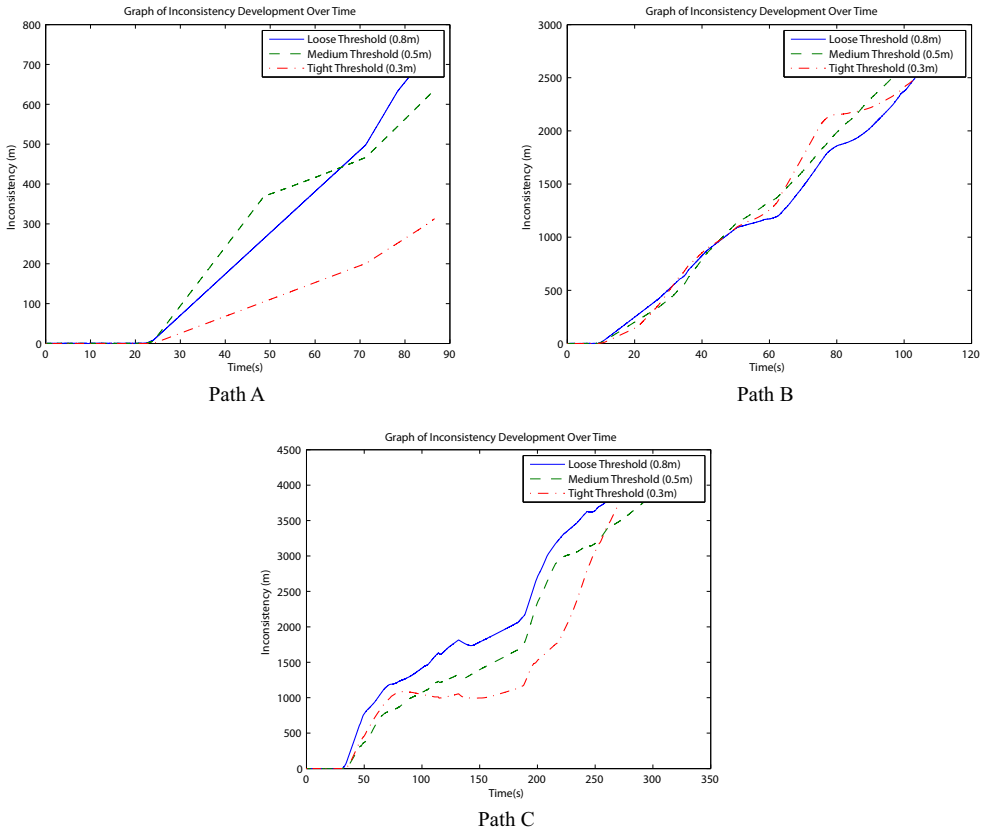


Figure 4.4: Graphs illustrating that collisions occurred for each path and that inconsistency would be uncontrolled without an authority scheme

Having established that each example to be presented has need of the authority scheme, the next section examines the performance of the authority scheme when

applied to the examples for fixed threshold dead reckoning models, using a range of thresholds.

4.3.2 Authority Scheme with Fixed Thresholds

In examining the performance of the authority scheme with fixed thresholds, each of the metrics listed in Section 4.1 is considered in turn. The same approach is taken when considering the variable threshold approach later in this chapter.

Physics-aware Inconsistency

Figure 4.5, Figure 4.6, and Figure 4.7 show the variation of physics-aware inconsistency over time for experiments with entities exhibiting bounce, smooth, and jolt classified motions respectively. For each classification of motion, only the first 20 seconds of the simulation is examined, as each simulation only had a single collision, which occurred in this timeframe. Due to the periodic and repeating nature of the motions, it is difficult to ensure multiple interactions, as the path was generally empty after a single period completed.

In Figure 4.5 (bounce) it can be observed that the peak present in the inconsistency of physics-aware entities alone is reduced as the threshold is tightened. This peak is representative of the inconsistency introduced to the state of a physics-aware body during a collision. The peak in Figure 4.6 (jolt) behaves similarly, with the implementation of the 0.15m threshold producing the largest reduction. Finally, Figure 4.7 (smooth) produces the same decrease in magnitude of the peak as in Figure 4.5. This is because the jolt and smooth motions followed the same path initially, and no collisions occurred in either simulation after they diverged. Additionally, the total physics-aware inconsistency present in the application reduces for tighter thresholds, as the error permitted within the dead reckoning model of entity motion is reduced.

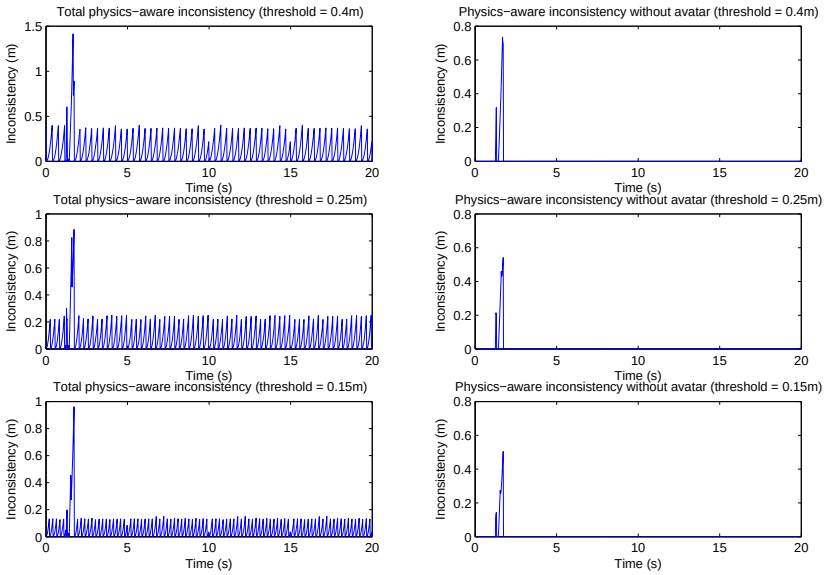


Figure 4.5: Physics-aware inconsistency for entity exhibiting bounce motion with three fixed threshold dead reckoning models, using thresholds of 0.4m, 0.25m and 0.15m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

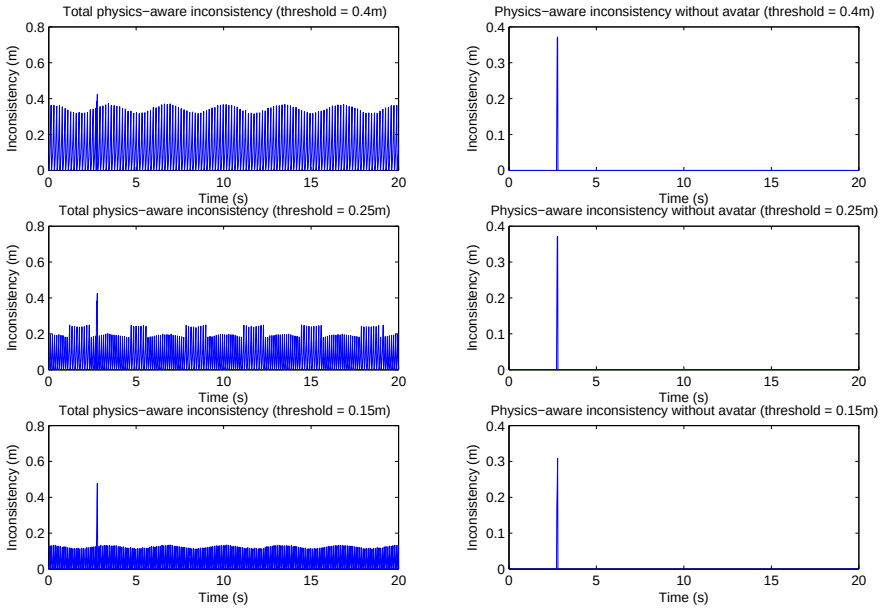


Figure 4.6: Physics-aware inconsistency for entity exhibiting jolt motion with three fixed threshold dead reckoning models, using thresholds of 0.4m, 0.25m and 0.15m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

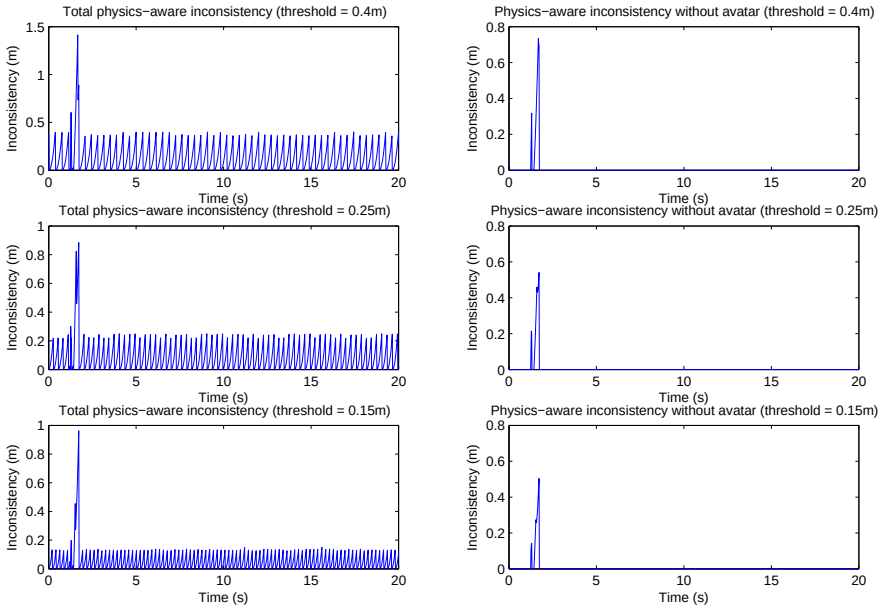


Figure 4.7: Physics-aware inconsistency for entity exhibiting smooth motion with three fixed threshold dead reckoning models, using thresholds of 0.4m, 0.25m and 0.15m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

Figure 4.8, Figure 4.9 and Figure 4.10 show the variation of physics-aware inconsistency over time for an entity following each of paths A, B and C respectively, with fixed threshold dead reckoning and the authority scheme in place. The contributions of physics-aware entities to this inconsistency are also isolated, and results are presented for three different fixed thresholds: 0.8m, 0.5m and 0.3m. This can be considered a graph of physics-consistency-cost over time as per the definition of physics-consistency cost in this thesis.

It can be observed in Figure 4.9 and Figure 4.10 in each of the graphs considering only the inconsistency present in physics-aware entities that the use of a tighter threshold produces a reduction in the magnitude of the peaks of physics-aware inconsistency. This is also exhibited in the graphs that include the contribution of the controlled entity, or avatar, but it is less clear in many cases being obscured by the variation in the avatar contribution.

Figure 4.8 is less clear, and indeed appears to present a less favourable result. For example, in the case of a 0.5m threshold in this instance, an additional peak

is visible in the inconsistency of physics-aware entities. This peak is still present in the other two graphs, but to a much smaller extent. A more detailed examination of the results suggested that the reason for the peak in this instance was the timing of dead reckoning updates; by chance the 0.8m and 0.3m threshold models both generated an update closer in time to the collision than the 0.5m threshold model, resulting in a lower level of inconsistency being present in the entity's state for the 0.8m and 0.3m models. Aside from this aberration, however, there is an observable trend that utilising a tighter threshold for a dead reckoning model produces less inconsistency in physics-aware entities at the time of collisions.

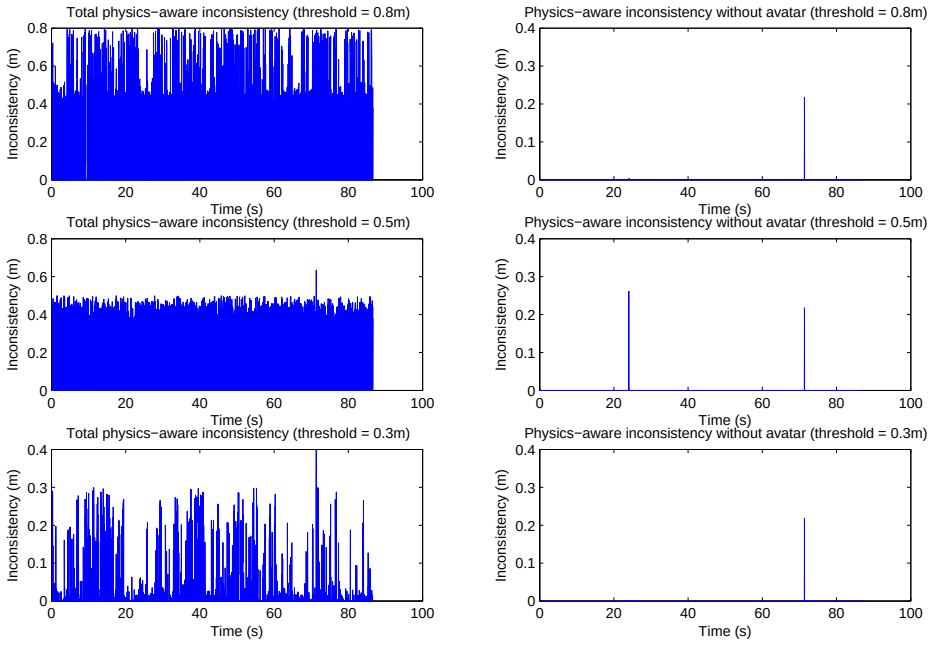


Figure 4.8: Physics-aware inconsistency for entity following path A with three fixed threshold dead reckoning models, using thresholds of 0.8m, 0.5m and 0.3m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

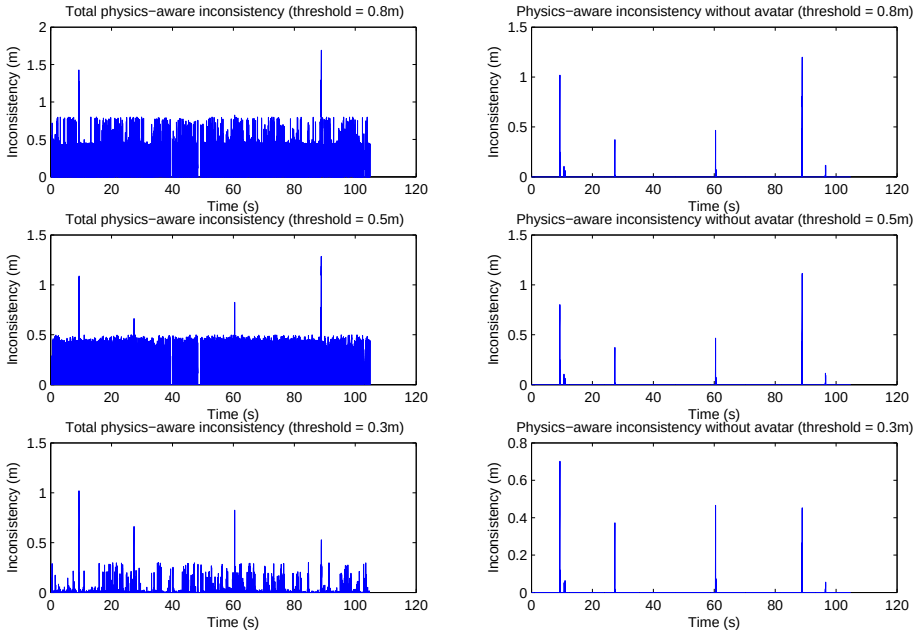


Figure 4.9: Physics-aware inconsistency for entity following path B with three fixed threshold dead reckoning models, using thresholds of 0.8m, 0.5m and 0.3m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

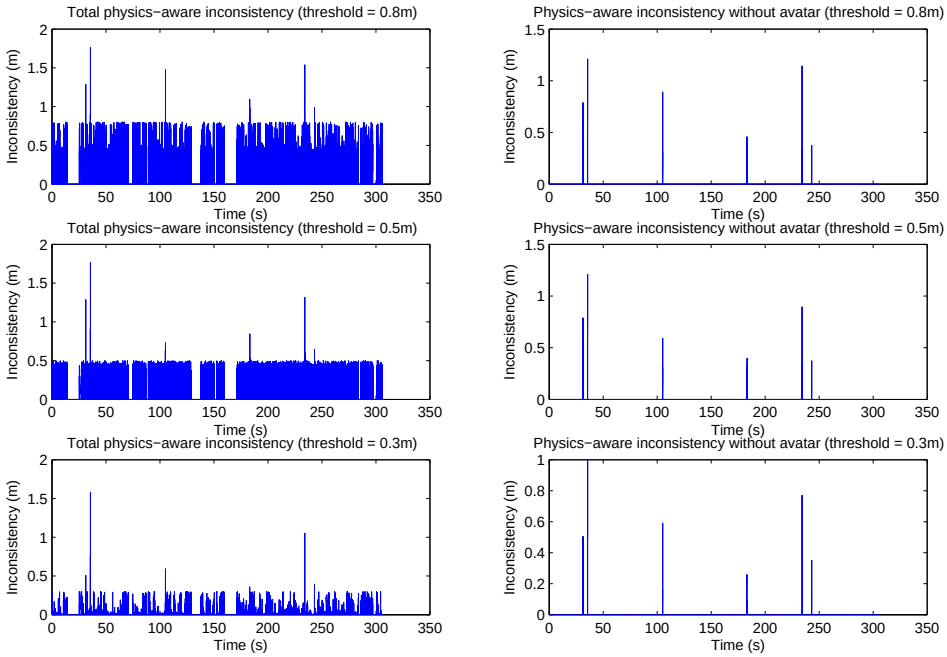


Figure 4.10: Physics-aware inconsistency for entity following path C with three fixed threshold dead reckoning models, using thresholds of 0.8m, 0.5m and 0.3m respectively. The left column includes contribution of avatar to overall physics-aware inconsistency, while the right isolates the contributions of physics-aware entities.

Correction Magnitudes

Figure 4.11 shows the average magnitudes of corrections applied to physics-aware entities in experiments involving entities exhibiting each of the classifications of motion, i.e. bounce, jolt and smooth. It is visible in each of these graphs that a tighter threshold in each case leads to smaller corrections to the state of physics-aware entities being necessary. It is more pronounced in the bounce and smooth examples than in the case of jolt motion, but this is likely related to the proximity of the last update to the collision.

Figure 4.12 presents the same information for each of the recorded path experiments. An aberration, however, is present in the case of path A, whereby the medium threshold produces the largest average correction magnitude. This is as a result of correcting the "extra" peak in Figure 4.8, and is therefore related to poor timing of updates, rather than disproving the otherwise present trend.

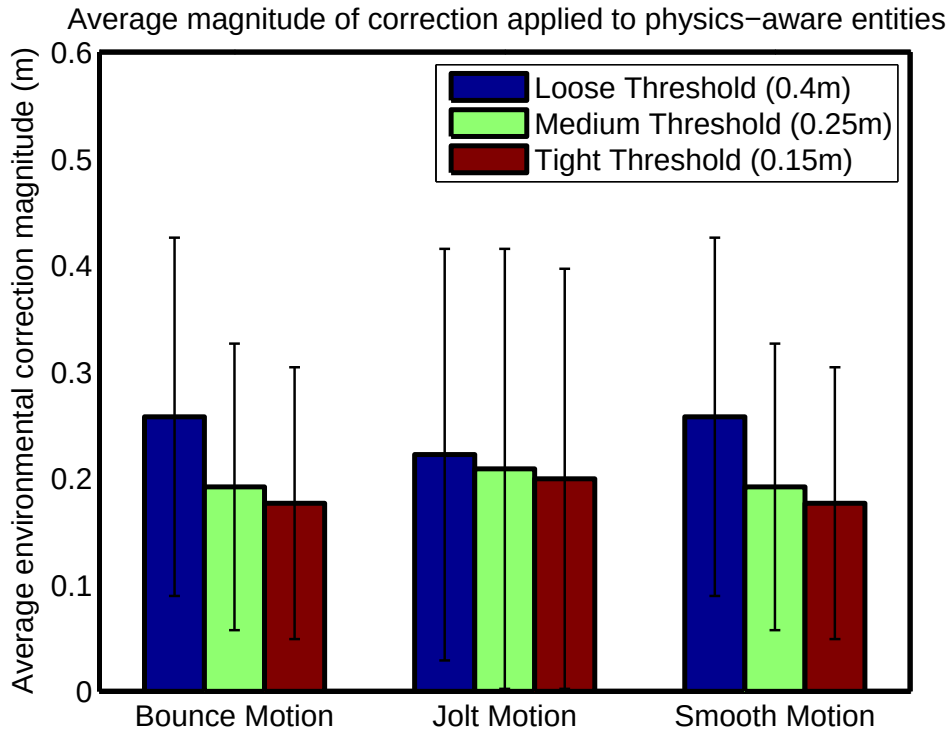


Figure 4.11: Average magnitude of corrections (in metres) applied to physics-aware entities for three fixed threshold dead reckoning models for each of bounce, jolt, and smooth motions.

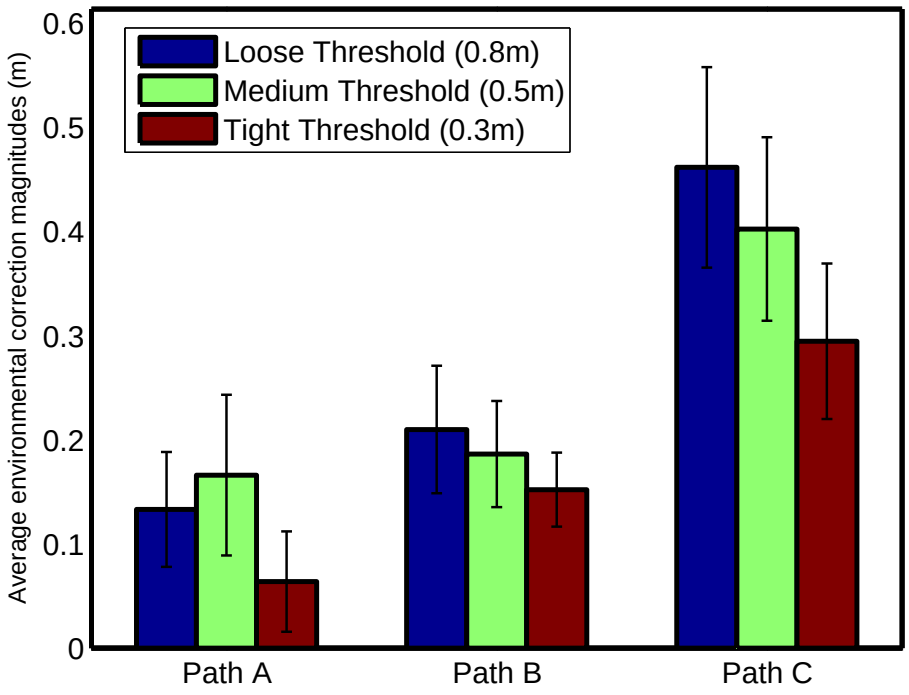


Figure 4.12: Average magnitude of corrections (in metres) applied to physics-aware entities for three fixed thresholds for each of paths A, B and C.

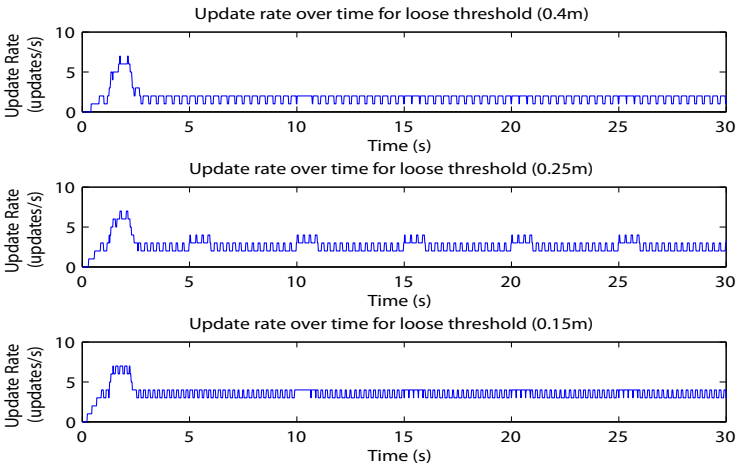
Update Rates

It can be observed from Figure 4.13 and Figure 4.14 that the improved consistency and accuracy gained by utilising a tighter threshold to calculate the dead reckoning model comes at a cost, both for the classifications of motion, and the recorded paths. Specifically, the tighter the threshold employed, the higher the update rate. This is because tighter error thresholds are violated or exceeded more frequently, triggering the transmission of a state update each time. Jolt motion tends to have a markedly higher average update rate for a given threshold than either of the bounce or smooth classes of motion. This is because the direction of jolt motion varies much more frequently than in either of the other classes, meaning that the rate of divergence of the model from the true motion is higher between updates. A similar characteristic arises periodically in the graph for bounce motion, where the entity turns back on its path. In Figure 4.13 this manifests as either the update rate remaining high for additional time (0.4m and 0.15m models), or rising to a higher rate (0.25m model).

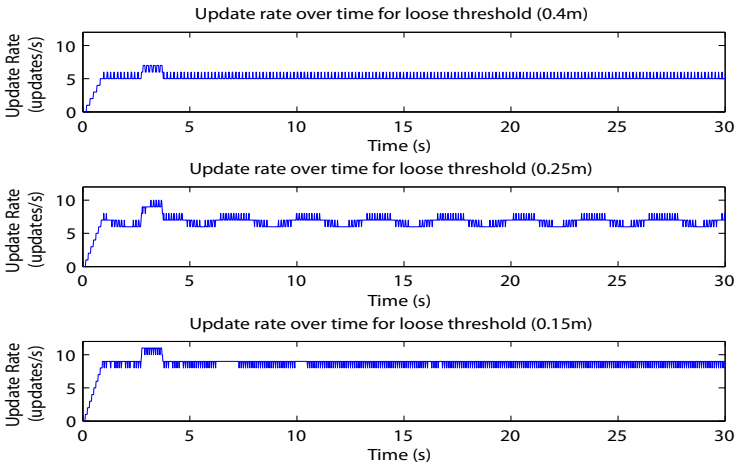
In the case of the recorded paths in Figure 4.14, it can be noted that the use of a 0.3m threshold (tightest threshold used in this experimentation) in the recorded path data causes a particularly high average update rate. The shape of the update rate curve for each model of a given path remains broadly similar, but as can be observed from the change in y-axis limits, from 20 updates per second to 50 updates per second, the numerical values involved are significantly higher.

4.3.3 Conclusions

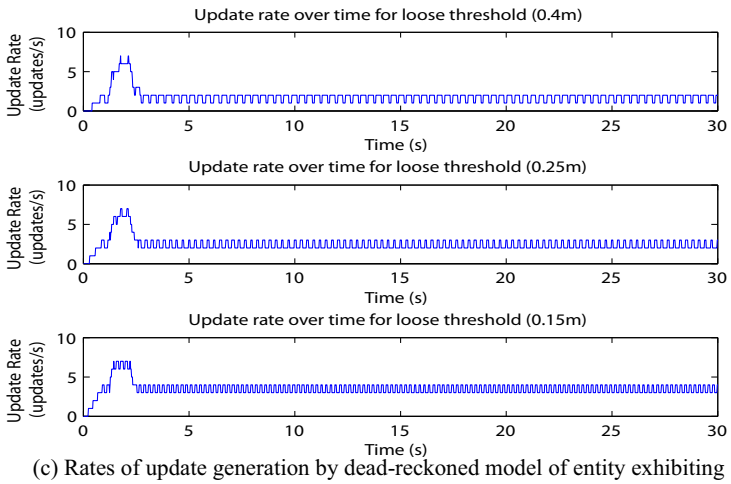
The results so far illustrate that in a simple two dimensional application with zero latency and collisions between the controlled entity and a single physics-aware entity, the authority scheme as proposed can control the amount of physics-aware inconsistency present. Improved levels of peak physics-aware inconsistency can, in general, be achieved by the use of a tighter error threshold in the calculation of dead reckoning models. The use of tighter thresholds reduces both the contributions of the controlled entity and physics-aware entities to the total physics-aware inconsistency. The cost imposed by utilising tighter thresholds, however, is an increased update rate, and hence usage of bandwidth.



(a) Rates of update generation by dead-reckoned model of entity exhibiting bounce motion, with authority scheme for thresholds of 0.4m, 0.25m and 0.15m

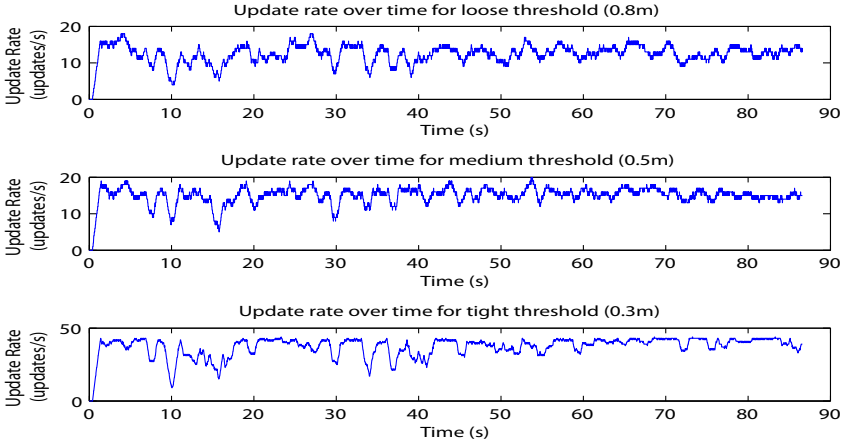


(b) Rates of update generation by dead-reckoned model of entity exhibiting jolt motion, with authority scheme for thresholds of 0.4m, 0.25m and 0.15m

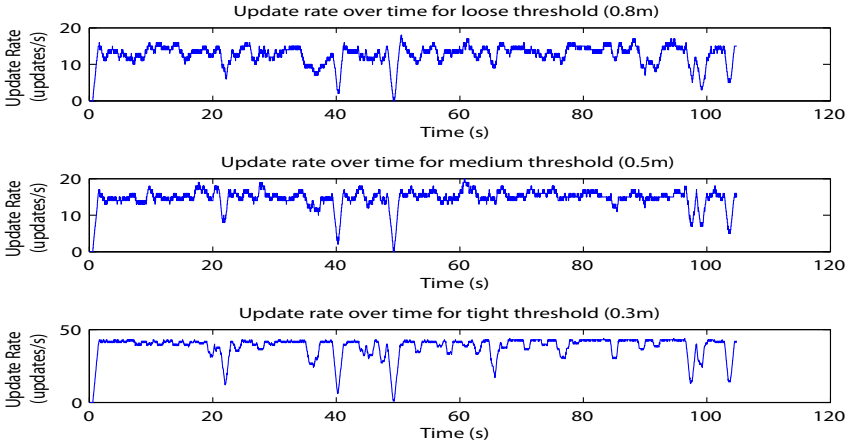


(c) Rates of update generation by dead-reckoned model of entity exhibiting smooth motion, with authority scheme for thresholds of 0.4m, 0.25m and 0.15m

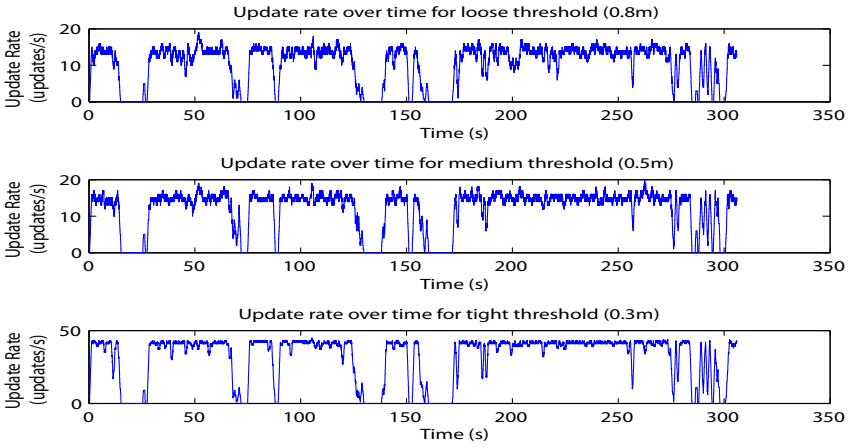
Figure 4.13: Graphs illustrating update rates in experiments with an entity exhibiting each of bounce, jolt and smooth motions, and fixed threshold dead reckoning models of entity motion with authority scheme for thresholds of 0.8m, 0.5m and 0.3m.



(a) Graph showing rate of update generation for entity following sample path A using authority scheme and dead reckoning for three thresholds, 0.8m, 0.5m and 0.3m respectively



(b) Graph showing rate of update generation for entity following sample path B using authority scheme and dead reckoning for three thresholds, 0.8m, 0.5m and 0.3m respectively



(c) Graph showing rate of update generation for entity following sample path C using authority scheme and dead reckoning for three thresholds, 0.8m, 0.5m and 0.3m respectively

Figure 4.14: Graphs illustrating update rates in experiments with an entity following each of paths A, B and C, and fixed threshold dead reckoning models of entity motion with authority scheme for thresholds of 0.8m, 0.5m and 0.3m.

4.4 Adaptive Threshold Dead Reckoning

Thus far the function of the authority scheme with fixed threshold dead reckoning models has been verified, and the broad effects of different threshold values for calculation of dead reckoning models on the mechanism has also been established. In this section, the adaptive threshold (in which the threshold is adapted between two values in response to a forecast of collisions for both the present entity state and the state of its dead reckoned model as described in detail in Chapter 3) dead reckoning model is tested, and compared to each of the previously tested scenarios. It should be noted at this point that due to the similarities observed in the bounce and smooth classifications of motion (only differing occasionally in their update rates), these will be considered as one for the purposes of this section. Thus the classified motion will be presented as bounce/smooth motion, and jolt motion.

4.4.1 Zero Latency

As in the previous set of tests, the adaptive threshold mechanism is tested first for performance under "ideal" conditions, i.e. zero latency and, as a result, jitter. A subsequent section will present an example of performance under such conditions.

Physics-aware Inconsistency

Figure 4.15 shows the variation of physics-aware inconsistency over time for entities following paths of classified motions as in Section 4.3.2. The only exception to this is that, as already noted in the previous section, only bounce motion is considered in relation to both the smooth and bounce classifications, as both of them were so similar, and this is labelled "smooth/bounce".

The noteworthy elements of this set of graphs are as follows:

- In the case of both classifications, the peak in the right hand graphs, i.e. the inconsistency present in physics-aware entities only, the magnitude of

the peak is the same as that of the tight (0.15m) threshold in the fixed threshold authority scheme.

- The total physics-aware inconsistency, which is usually dominated by the controlled entity's inconsistency, is mostly the same as for the loose (0.4m) threshold in a fixed threshold scheme. This is acceptable, and in fact desired, as it means that the looser threshold, which would be determined as sufficient for times when the controlled entity is not interacting with a physics-aware entity, is in use most of time
- The threshold can be observed as tightening at the time of collision, when the total physics-aware inconsistency drops briefly to a lower value, before rising to the previous value after the threshold is relaxed again.

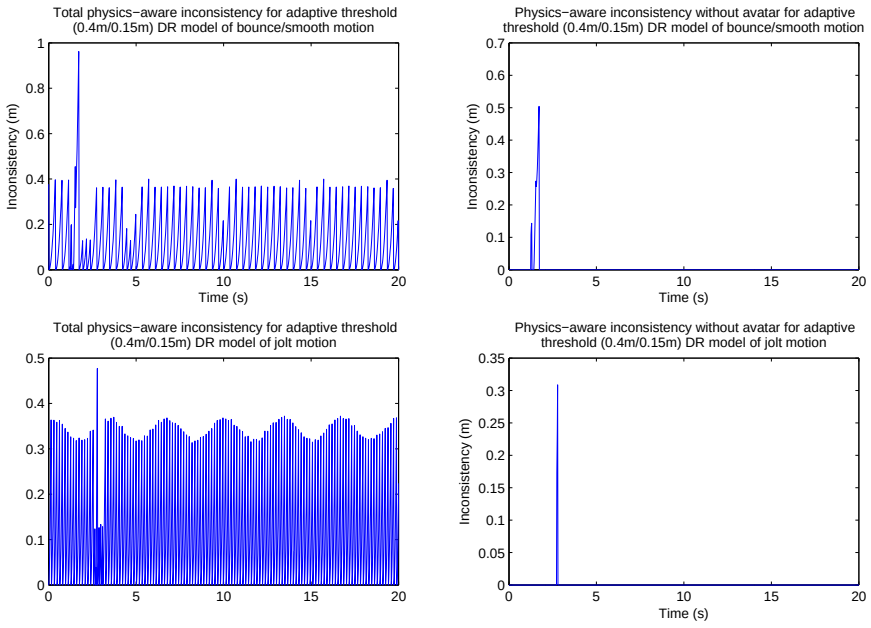


Figure 4.15: Variation of physics-aware inconsistency over time for entities exhibiting smooth/bounce motion, and jolt motion respectively, with models utilising physics-aware adaptive dead reckoning. Thresholds employed are 0.4m and 0.15m.

Figure 4.16 presents a similar set of experiments for the recorded paths A, B and C, and similar observations can be made for each of these:

- In the right column of graphs for each motion, where the contribution to inconsistency of the controlled entity has been removed, the profiles of the peaks of inconsistency present in physics-aware entities match those

of the tight threshold (0.3m) presented in Figure 4.8, Figure 4.9, and Figure 4.10. Specifically, the largest peak for path B is approximately 0.7m, and for Path C it is approximately 1m.

- While the total physics-aware inconsistency (left column of graphs) for each path is less regular than that presented in Figure 4.15, and indeed in the fixed threshold scheme earlier (Figure 4.8, Figure 4.9, and Figure 4.10), it can be seen to fluctuate between 0.8m and 0.3m at different times, aside from peaks when collisions arise.
- Similar to the graphs in Figure 4.15, the physics-aware inconsistency is seen to fall as the threshold tightens due to predicted collisions. There are, however, more visible instances where the threshold tightens, and no collision actually takes place. This may lead to wasted bandwidth or network traffic when considering the performance of the mechanism in terms of its update rate.

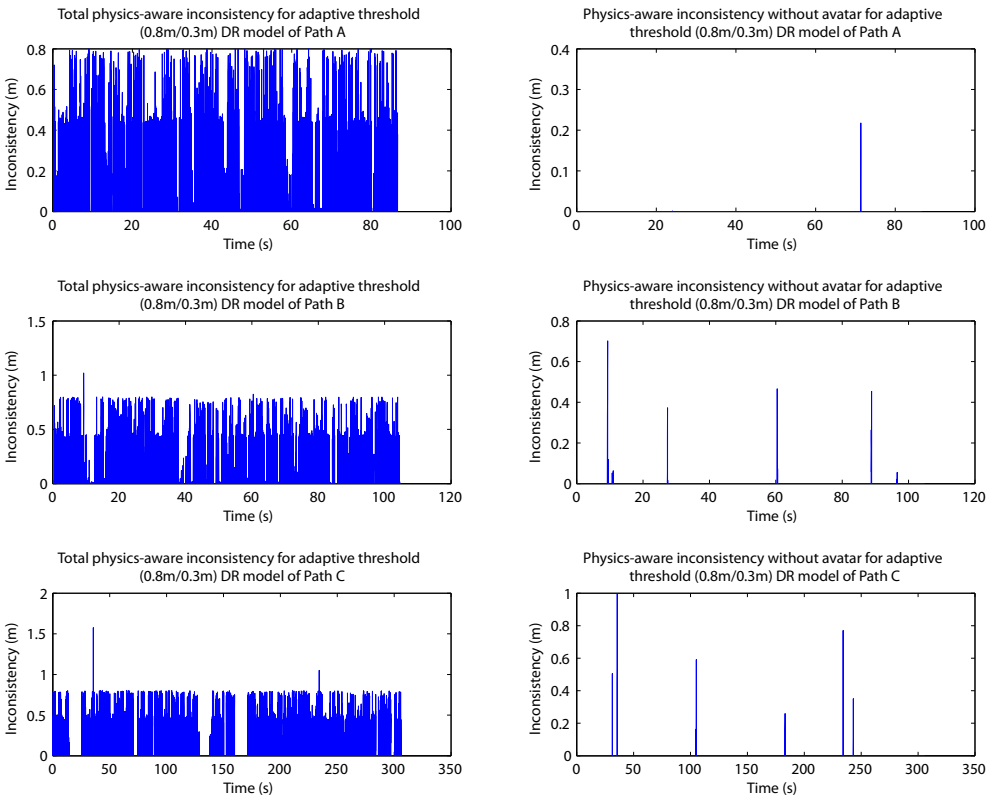


Figure 4.16: Variation of physics-aware inconsistency over time for an entity following paths A, B and C respectively, with models utilising physics-aware adaptive dead reckoning. Thresholds employed are 0.8m and 0.3m.

Correction Magnitudes

Figure 4.17 shows the average magnitude of the corrections applied to physics-aware entities in simulations for both bounce/smooth motion, and jolt motion. For reference, the average magnitudes of the equivalent corrections for each of the fixed threshold models presented earlier are also presented. In the case of bounce/smooth motion, the adaptive threshold model actually results in a smaller magnitude correction being applied than even the tight (0.15m) fixed threshold model.

In contrast to this, however, the adaptive threshold model performs slightly worse than the tight fixed threshold model for jolt motion. It still results in smaller corrections to environmental entities than either the 0.4m or 0.25m thresholds, however. The reason for the variation in this instance appeared to be the difference in the time elapsed since the last dead reckoning update was generated before the collision occurred. Previous examination of update rates for the classified motions has shown relatively low update rates, meaning that updates of the controlled entity's model can be as long as 200ms, or 10 simulation steps, apart. This means that a collision could be anywhere between 1 and 10 simulation ticks after a dead reckoning update. Therefore the timing of the final dead reckoning update before a collision relative to the collision itself can influence this test. Coupling this with the low sample size (a single collision and associated updates in each case) means that this influences the graph significantly.

This influence is less observed in the case of the recorded paths, however, and the average correction magnitudes applied to physics-aware entities by the adaptive threshold mechanism are presented in Figure 4.18. In this graph, the proposed mechanism results in much more similar magnitudes to the tight (0.3m) threshold, with the corrections of both the adaptive threshold and tight threshold being consistently similar. These experiments generated updates at a higher rate than the classified motions, and as can be seen in Figure 4.14, the 0.3m threshold generated updates at a rate only slightly lower than the simulation update rate of

50Hz. As a result of this frequency of updates, the simulations are less sensitive to the time at which the final model update before the collision is generated.

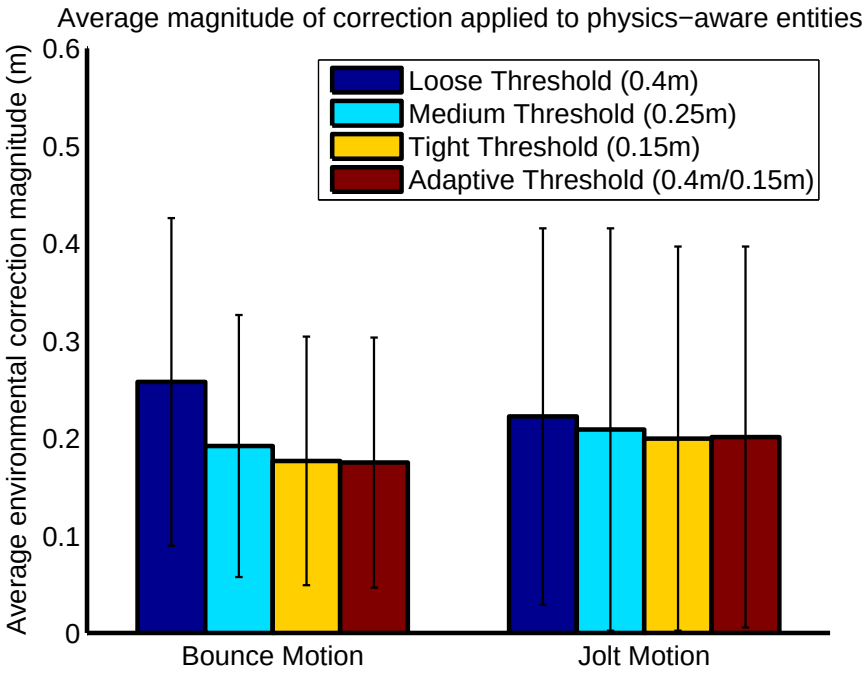


Figure 4.17: Average magnitude of corrections applied to physics-aware entities in experiments for controlled entities exhibiting bounce/smooth motion, and jolt motion respectively. Three fixed threshold models (0.4m, 0.25m and 0.15m), and an adaptive threshold (0.4m/0.15m) model are compared.

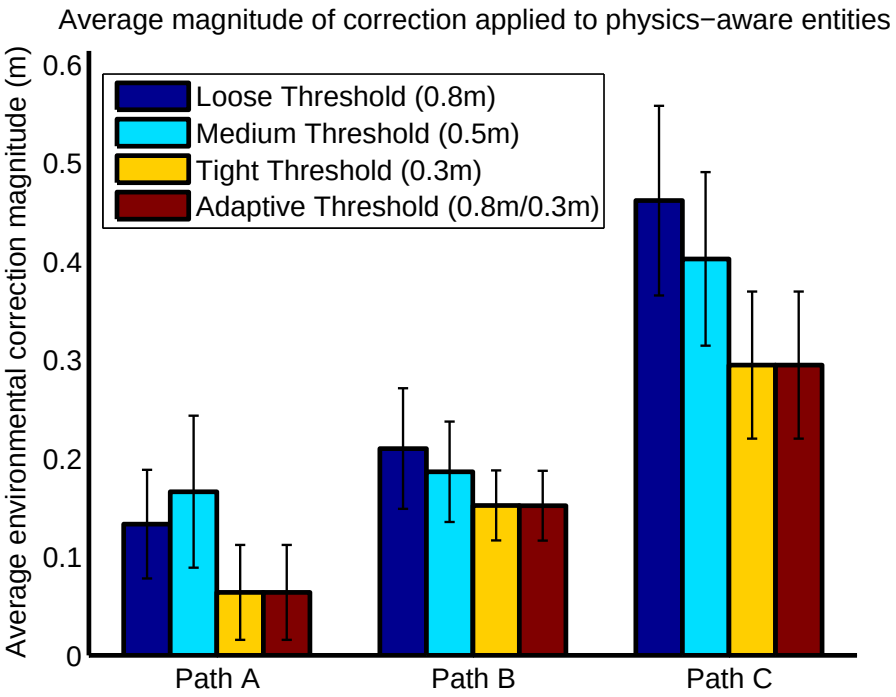


Figure 4.18: Average magnitude of corrections applied to physics-aware entities in experiments for controlled entities following paths A, B and C respectively. Three fixed threshold models (0.8m, 0.5m and 0.3m), and an adaptive threshold (0.8m/0.3m) model are compared.

Update Rates

Figure 4.19 and Figure 4.20 show the variation of update rate over time for the adaptive threshold dead reckoning models for classified and recorded motion respectively. To consider Figure 4.19, if the update rate is compared to those presented in Figure 4.13, it can be seen that for each of the classes of motion, the rates remain at the value of the loose threshold model most of the time, with the exception being the times when collisions occur, at which point they rise to similar heights to the tight threshold. This is to be expected, as the tight threshold should impose an upper bound on the update rate, with a similar lower bound being imposed by the looser threshold.

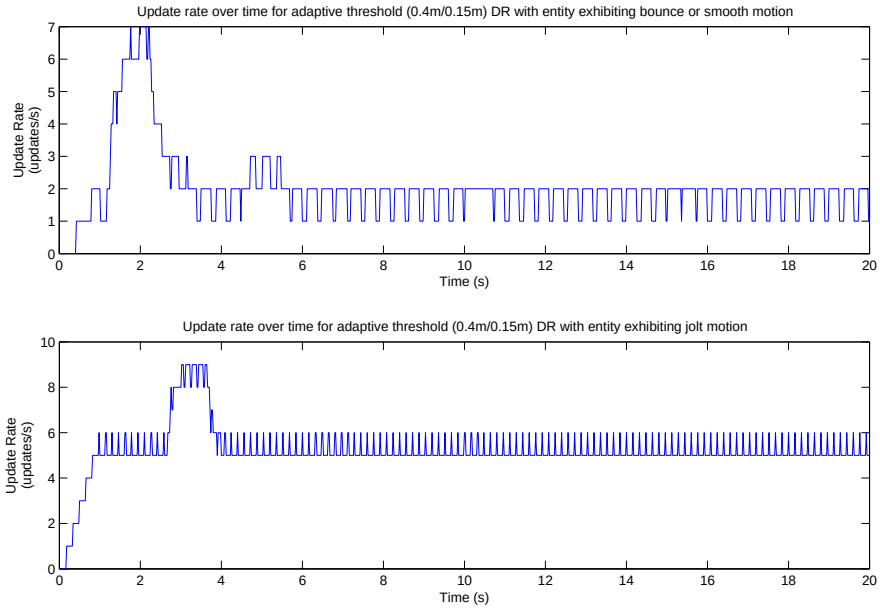


Figure 4.19: Update rates for adaptive threshold (0.4m/0.15m) dead reckoning for entity exhibiting bounce (or smooth) and jolt motions respectively.

In examining Figure 4.20, it is to be compared to Figure 4.14. Similar to the above classified motion examples, it can be observed for each path that the tight and loose threshold models already examined provide an upper and lower bound respectively on the update rate of the adaptive threshold mechanism. In this set of graphs, the update rate varies significantly more often than in the case of the simulated classified motions, but this is also consistent with the graphs already examined for the fixed threshold models.

Just as the inconsistencies in Figure 4.16 were observed to vary and the thresholds tighten in the absence of collisions, so too do the update rates presented here. This is due to the prediction mechanism perhaps predicting too far ahead, and observing collisions that the entity motion will in fact avoid, but tightening the threshold regardless. This suggests that the parameters of the anticipation, or forecasting, mechanism need further adjustment. For example, a mechanism that predicted ahead for a shorter time would not anticipate collisions until closer to the time of occurrence, at which point they may have a greater likelihood of occurring.

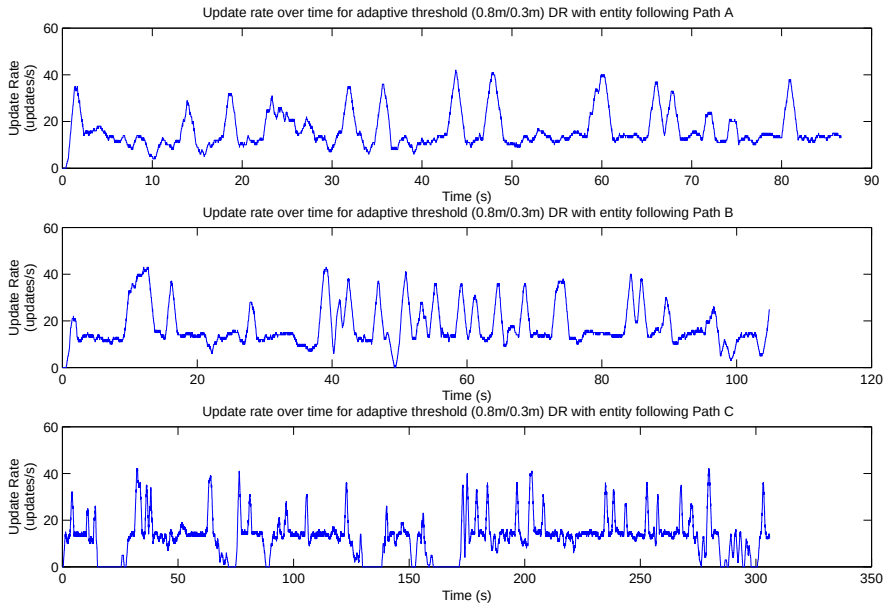


Figure 4.20: Update rates for adaptive threshold (0.8m/0.3m) dead reckoning for entity exhibiting bounce (or smooth) and jolt motions respectively.

Having examined this mechanism in detail for the ideal conditions of zero latency, in the next section, the same simulations as have been examined are conducted for both constant latency, and latency with jitter, or variation.

4.4.2 Non-zero Latency and Jitter

The final test applied to this algorithm is to analyse how it responds to real-world network influences such as latency and jitter, in contrast to previous experiments which have featured "ideal" conditions, i.e. zero latency and jitter. In order to achieve this, latency and jitter was applied to the simulated connection between the local and remote environments in the testbed, and results recorded as in the previous experiments. The three previously used sample paths will be examined in detail for the purposes of this section, with classified motion omitted, as the low numbers of interactions, and periodic nature of the motion limit the insight that can be gained from them.

Due to the absence of any feedback mechanism between the local and remote peers in the algorithm and testbed, inconsistency is the only quantity examined in

this section. Update generation rates have a similar profile to those acquired during the previous experiments, and so are omitted from this section.

4.4.3 Fixed Latency

Figure 4.21 shows graphs of remote inconsistency for paths A, B and C. For the purposes of this experiment, a fixed latency of 100ms was incorporated into the simulation. The graphs illustrate that inconsistency levels in the presence of latency are higher than in previous sections, but this is to be expected, as the delay in application of updates to the remote peer means that the remote view of the dead reckoning model, and of the interactions "lags" the local view. In spite of this increased inconsistency being present, it is still limited, or controlled when taken in comparison to the absence of an authority scheme, as in Figure 4.4.

In the case of path A, inconsistency is reasonably constant, experiencing reductions occasionally. Between 70 and 80 seconds the average inconsistency seems to increase, likely due to a collision occurring at this point, and another physics-aware entity set in motion means that the remote simulation experiences a delayed view of this entity as well.

Similar increases in inconsistency are evident in the graphs for paths B and C at the times of collisions, suggesting that each physics-aware entity set in motion makes a measurable contribution to the overall inconsistency. At some instances in all three simulations the inconsistency drops to zero, or near zero, associated with times where the recorded motion indicates that the player stopped moving. This allows the remote peer's model to "catch up" with the local peer's state.

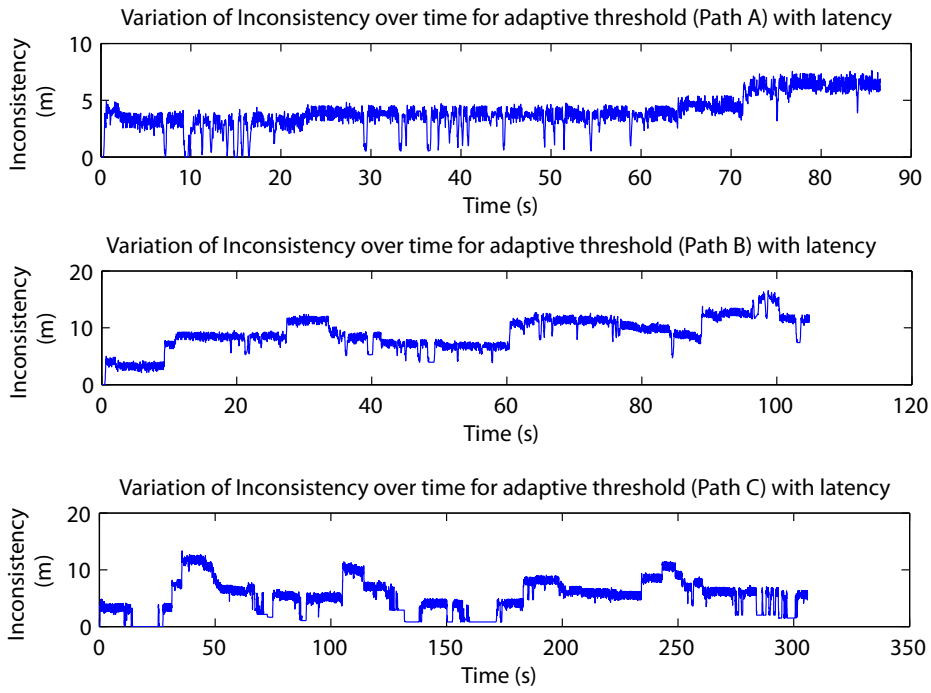


Figure 4.21: Graphs of inconsistency variation over time with constant latency of 100ms present between local and remote simulated peers for entities following paths A, B and C respectively. An adaptive threshold (0.8m/0.3m) is employed for the dead reckoning model.

4.4.4 Latency with Jitter

Figure 4.22 illustrates the inconsistency observed for 3 simulations of entities following paths A, B and C respectively, with average latency of 100ms, and jitter of 10ms simulated between the local and remote peers. Similar to the previous section dealing with constant latency simulation, we can observe that the collision notification algorithm still provides a bound on the inconsistency present in the view observed by a remote peer. Figure 4.22 presents a broadly similar performance to that of Figure 4.21, suggesting that the algorithm is reasonably robust to the presence of a moderate amount of jitter or variation in latency, provided out of order packets are discarded appropriately.

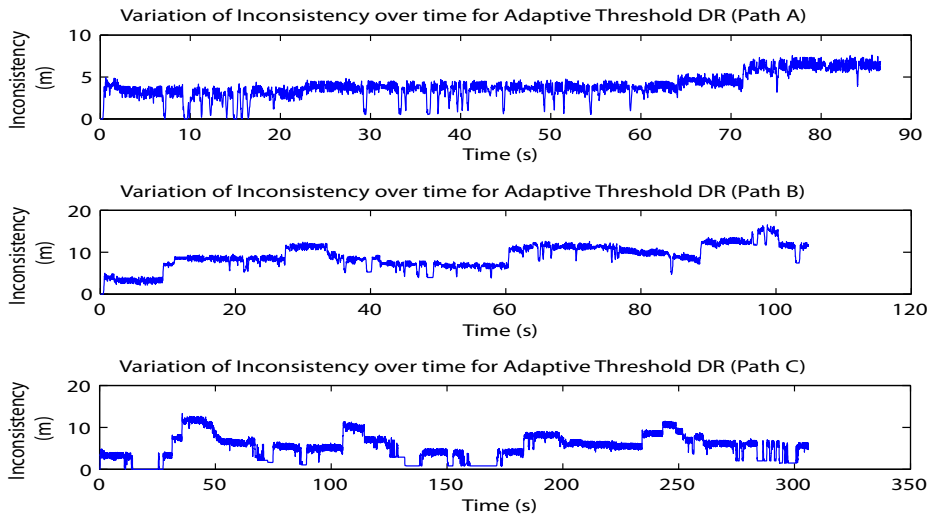


Figure 4.22: Graphs of inconsistency variation over time with latency of 100ms and jitter of 10ms present between local and remote simulated peers for entities following paths A, B and C respectively. An adaptive threshold (0.8m/0.3m) is employed for the dead reckoning model.

4.5 Summary

In this chapter, a series of tests have been outlined to evaluate the performance of both a notification scheme, and a physics-aware, adaptive threshold dead reckoning model. The results have indicated that adapting the dead reckoning model to utilise a tighter threshold for the motion of controlled entities in advance of collisions can lead to more accurate simulation of those collisions. This was measured by considering the overall physics-aware inconsistency present in the application, or the sum of the spatial inconsistencies in both the local peer's controlled entity and all physics-aware entities.

The inconsistency present in the states of physics-aware entities with the controlled entity removed was also considered, so that the effects of the mechanism on these entities alone could be determined. In this analysis, peaks in the inconsistency present in these entities (as a result of collisions) were observed to be reduced by the use of an adaptive threshold algorithm, similar to the manner in which the same peaks were reduced by using a tighter fixed threshold.

In order to verify the accuracy with which collisions and contacts between the controlled entity and physics-aware entities were simulated, the magnitudes of the state corrections induced at the remote peer by applying updates were measured. It was found that generally the adaptive threshold model produced similar performance in this metric to the tighter of the two thresholds employed. However, if the update rate of the tighter threshold is sufficiently low, then the relationship between the magnitudes of corrections applied to physics-aware entities under the adaptive threshold model and the tight fixed threshold model appeared to be less consistent. This was observed to be due to the error in the model assuming a greater range of values between zero and the error threshold, as it approached the threshold.

The improvements in physics-aware consistency granted by the adaptive threshold model come at a cost, however. Depending on the choice of thresholds in the application, the update rate associated with the tighter threshold can approach the simulation frequency of the application. Thus if the bandwidth available to an application is a concern, this must be taken into account at the design phase, when selecting the thresholds to be used in the adaptive model. Upper and lower bounds for the update rate of an adaptive threshold model can be ascertained by considering the two thresholds in use, as the update rate becomes a mix of these, and varies between the two.

Finally, the performance of the adaptive threshold model and authority scheme under a realistic set of network conditions was evaluated, and it was observed that while greater physics-aware inconsistency developed in the application than under ideal conditions, a bound on this inconsistency was still imposed.

Chapter 5
Conclusions and Future Work

In this thesis a state management mechanism for physics-rich, peer-to-peer DIAs using dead reckoning modelling of controlled entities was proposed. Additionally, the design and implementation of a testbed application for testing the effectiveness of that algorithm was described. The development of this mechanism was motivated by a lack of published physics-aware or "physics-friendly" DIAs and state management algorithms, specifically for peer-to-peer DIAs utilising dead reckoning to model remote entities.

5.1 Algorithm Evaluation

The state maintenance mechanism as outlined in Chapter 3 consisted of two main elements, or components; an authority scheme for physics-aware entities, and a physics-aware, adaptive threshold dead reckoning model. Each of these components was intended to serve a different purpose. The goal of the authority scheme was to ensure an overall bound on the level of inconsistency in physics-aware entities by facilitating corrections to the state of such entities at remote peers, while also providing a responsive simulation to a local peer. The adaptive threshold dead reckoning model extended this idea in an attempt to minimise the physics-consistency-cost of collisions or interactions. Physics-consistency-cost was introduced in this thesis as a flexible concept to describe the loss of consistency induced in such interactions. While the state management mechanism considered it as a sum of spatial inconsistencies, it can also be adapted to consider the complexity of, or number of bodies involved in, a physics-aware inconsistency.

For the scenarios examined in Chapter 4, the authority scheme was successful in ensuring that physics-aware inconsistency in the environment is controlled, and resulted in a situation where, for zero latency, peaks in physics-aware inconsistency were observed, but the inconsistency quickly settled to the previously observed levels.

Similarly, the adaptive threshold dead reckoning model was successful in providing a more accurate model of the motion of the controlled entity prior to and during collisions. In the model utilising two thresholds, a level of accuracy similar to that of the tighter threshold could be achieved in the simulation of collisions at remote peers.

5.2 Limitations

While the state synchronisation mechanism presented in this thesis has addressed a number of the challenges associated with designing and implementing a physics-aware DIA, both components of the mechanism have limitations and consequently scope for improvement.

5.2.1 Authority Scheme

The authority scheme as detailed in Chapter 3 stipulates that it can only account for collisions whereby the controlled entity collides directly with a physics-aware entity. In the same chapter, it is explained that collisions-by-proxy, whereby the controlled entity causes the physics-aware entity to touch a second physics-aware entity at the same time as the controlled entity, can result in the second physics-aware entity becoming inconsistent. This may not be sufficient in some applications with high densities of physics-aware entities, and wider ranging authority may be necessary. A suggested implementation for this, in the form of recursive authority, is included in the same chapter.

The authority scheme is also dependent on state updates to physics-aware entities being reliably transmitted. The underlying UDP protocol used for data transmission in many DIAs does not guarantee delivery however, which means that this must be manually handled within the application, or by networking middleware. Retransmission of such updates (in the event of one being lost) can introduce additional latency, and thus jitter to that update. As a consequence of this, an update to the dead reckoning model of the controlled entity that was sent after an update regarding a collision could in fact be received and applied before

the physics-aware entity has been updated. Thus the movement of the physics-aware entity could be observed to occur after the dead reckoned model of the controlled entity has moved past the collision. By contrast, an ordered series of unreliable state updates for the physics-aware entity could see the lost update ignored, and a slightly later one applied before a reliable update would have been retransmitted.

It was stipulated in Chapter 3 that the authority scheme as outlined depends on the deterministic simulation of physics at all peers. While many physics engines guarantee deterministic results and outputs, this is still a limitation of the scheme should an application be unable to ensure determinism. Both this and the previous issue of reliability could potentially be improved by further adapting the authority scheme proposed by Fiedler (2010b), whereby peers maintain dead reckoning models for physics-aware entities that they have authority over, and transmit updates to this model until the entities come to rest.

5.2.2 Adaptive Threshold Dead Reckoning Model

In introducing the physics-aware, adaptive threshold dead reckoning model, it was stated that a forecasting method would be used to determine when to adapt, or tighten, the threshold in anticipation of a collision. An alternative approach based on "entity population density" was also suggested, which may be more suitable for some applications or scenarios. A forecasting method as proposed depends on an entity having a certain amount of inertia or momentum, in that it should not be able to deviate significantly from its predicted path by the end of the prediction. This may not be applicable in all applications, such as first-person shooters where users can change either the speed or direction of their velocity very quickly and suddenly. Examining the density of physics-aware entities nearby, or simply the proximity of the user to any physics-aware entity, may be more reliable for such applications.

The model is also sensitive to choices in the size of error thresholds, and in the number of steps to predict ahead when anticipating collisions. If an excessively tight threshold is used in calculating the model at any point (i.e. less than or

equal to the distance that a user can move in a single simulation tick), then the rate at which updates are sent to remote peers may equal the simulation rate at times. Similarly a collision forecasting mechanism predicting too far ahead in time increases the risk of unnecessary tightening of the model threshold, as the likelihood of the collisions actually occurring (i.e. the accuracy of the forecast) decreases with each timestep.

5.3 Future Work

5.3.1 Cheat prevention

As already referred to in Section 2.2.1, peer-to-peer applications are susceptible to cheating due to the lack of a single central authority. As the algorithm presented in this thesis makes no special provisions to prevent such cheating, it is equally vulnerable, and potentially more so, as allowing peers to update the state of entities not owned by any peer grants them a limited form of authority. A rudimentary mechanism to improve the robustness would be for peers to only accept state updates to an environmental entity when their model of the originating peer is within a certain distance of that entity (so as to ascertain some measure of the genuine need for this authority transfer), but this could still be abused.

Consequently, a preferred approach would be to integrate this algorithm with one of the established anti-cheat systems for peer-to-peer DIAs, such as the Referee Anti Cheat System (RACS) (Webb et al., 2007).

5.3.2 "N-tier" hierarchy of thresholds

In this thesis, an adaptive dead-reckoning algorithm was proposed that varied its model between two thresholds. An observed issue with this is that the change in update rate when a collision *may* occur can be significant if the thresholds employed are sufficiently different in magnitude. This can create a significant increase in the overall traffic generated by the application, which can lead to scalability issues. Thus there is scope to develop a more bandwidth efficient

algorithm. Specifically this work could extend the algorithm to select the threshold to be used from a range of available values, based on a physics-consistency-cost anticipation mechanism returning one of a range of values. This would allow a host to provide a variety of models of differing fidelities, and only employ the most bandwidth intensive one when the entity is very close to a collision or discrepancy, and thus it is more likely to arise, necessitating the most accurate model.

5.3.3 Reduced traffic prediction model

In the adaptive threshold model proposed, it was suggested that the error threshold of the model be tightened in the event of either

1. A disagreement between collisions occurring for forecasts of true state and remote state, or
2. A collision being forecast for the true state.

This potentially induces periods with a high rate of update generation even though no collision may actually occur. An alternative implementation to be considered in the future is to

1. Use a tighter threshold if a collision is forecast when considering true state, regardless of the forecast for the state of the dead reckoning model.
2. Send a single update of controlled entity state if a forecast using dead reckoning state predicts a collision, while the forecast for true motion does not.

This modification would mean that periods of higher traffic are reserved for instances where the controlled entity's true state leads to a collision being forecast, further optimising the use of bandwidth for physics-awareness.

5.4 Conclusions

The authority scheme and physics-aware dead reckoning model proposed in this thesis are two components of a state management mechanism that may be useful for physics aware distributed interactive peer-to-peer applications. An authority

scheme of some form is necessary in any physics-aware, peer-to-peer DIA to ensure that the physics-aware inconsistency present is bounded. The scheme as proposed, or other similar schemes, can function with either a fixed threshold or adaptive threshold dead reckoning model. Similarly, the physics-aware dead reckoning model as proposed operates independent of the specific authority scheme, to improve the accuracy of the dead reckoning model in the area around collisions.

While the exact authority scheme and implementation of a physics-aware dead reckoning model as outlined in this thesis are limited in their functionality and effectiveness, the general concepts introduced are more flexible. Physics-consistency-costs can consider a simple sum of spatial inconsistencies, but it can also be generalised to consider the complexity of the collision. Similarly, the criteria for varying the threshold of a physics-aware dead reckoning model are flexible, and can be chosen to suit the specific application.

In summary, while this thesis demonstrated a specific authority scheme in use alongside a forecast-based adaptive threshold dead reckoning model, these two concepts, along with physics-aware inconsistency, are flexible components and considerations in physics-aware DIAs, with scope for tailoring to an individual application.

REFERENCES

Ageia (2005a). *Advanced Gaming Physics: Defining the New Reality in PC Hardware* http://www.ageia.com/pdf/wp_advanced_gaming_physics.pdf

Ageia (2005b). *Physics, Gameplay and the Physics Processing Unit* http://www.datasheetarchive.com/indexdl/Datasheet-029/DSA00514337.pdf

Aldridge, D. I Shot You First! Gameplay Networking in Halo Reach. Game Developers Conference, February 28th - March 4th 2011 San Francisco, CA.

Assiotis, M. & Tzanov, V. (2006). A distributed architecture for MMORPG. *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games.* Singapore: ACM.

Barrus, J. W., Waters, R. C. & Anderson, D. B. Locales and beacons: efficient and precise support for large multi-user virtual environments. Virtual Reality Annual International Symposium, 1996., Proceedings of the IEEE 1996, 30 Mar-3 Apr 1996 1996. 204-213, 268.

Bassiouni, M. A., Chiu, M.-H., Loper, M., Garnsey, M. & Williams, J. (1997). Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation. *ACM Trans. Model. Comput. Simul.,* 7**,** 293-331.

Bettner, P. & Terrano, M. (2001). *1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond* [Online]. Gamasutra. Available: http://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php [Accessed September 26th 2011].

Bhola, S., Banavar, G. & Ahamad, M. (1998). Responsiveness and consistency tradeoffs in interactive groupware. *Proceedings of the 1998 ACM*

conference on Computer supported cooperative work. Seattle, Washington, United States: ACM.

Bishop, B., Kelliher, T. P. & Irwin, M. J. (2000). SPARTA: Simulation of Physics on a Real-Time Architecture. *Proceedings of the 10th Great Lakes symposium on VLSI.* Chicago, Illinois, United States: ACM.

Blow, J. (1998). A Look at Latency in Networked Games. *Game Developer,* 5**,** 28-40.

Bouillot, N. & Gressier-Soudan, E. (2004). Consistency models for distributed interactive multimedia applications. *SIGOPS Oper. Syst. Rev.,* 38**,** 20-32.

Brown, E. & Cairns, P. (2004). A grounded investigation of game immersion. *Extended abstracts of the 2004 conference on Human factors and computing systems - CHI '04***,** 1297.

Bullet (2008). *Determinism* [Online]. Available: http://bulletphysics.org/mediawiki-1.5.8/index.php/Determinism [Accessed 26/09 2011].

Cai, W., Lee, F. B. S. & Chen, L. (1999). An auto-adaptive dead reckoning algorithm for distributed interactive simulation. *Proceedings of the thirteenth workshop on Parallel and distributed simulation.* Atlanta, Georgia, United States: IEEE Computer Society.

Cairns, P., Cox, A., Berthouze, N., Dhoparee, S. & Jennett, C. (2006). Quantifying the experience of immersion in games. *Proc Cognitive Science of Games and Gameplay workshop at Cognitive Science 2006***,** 7.

Calvin, J., Dickens, A., Gaines, B., Metzger, P., Miller, D. & Owen, D. The SIMNET virtual world architecture. Virtual Reality Annual International Symposium, 1993., 1993 IEEE, 18-22 Sep 1993 1993. 450-455.

Capps, M., Mcgregor, D., Brutzman, D. & Zyda, M. (2000). NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments. *IEEE Comput. Graph. Appl.,* 20**,** 12-15.

Catto, E. (2007). *Box2D User Manual* [Online]. Available: http://www.box2d.org/manual.html [Accessed September 23rd 2011].

Catto, E. (2011). *Box2D FAQ* [Online]. Available: http://code.google.com/p/box2d/wiki/FAQ [Accessed September 26th 2011].

Cheng, K. & Cairns, P. A. (2005). Behaviour, realism and immersion in games. *CHI '05 extended abstracts on Human factors in computing systems.* Portland, OR, USA: ACM.

Cronin, E., Filstrup, B., Kurc, A. R. & Jamin, S. (2002). An efficient synchronization mechanism for mirrored game architectures. *Proceedings of the 1st workshop on Network and system support for games.* Braunschweig, Germany: ACM.

Dahmann, J. S., Fujimoto, R. M. & Weatherly, R. M. (1997). The Department of Defense High Level Architecture. *Proceedings of the 29th conference on Winter simulation.* Atlanta, Georgia, United States: IEEE Computer Society.

Davis, T. (2008). *Dedicated Or Peer, That Is The Question* [Online]. Available: http://www.thebitbag.com/2008/09/03/dedicated-or-peer-that-is-the-question/ [Accessed September 30th 2011].

Delaney, D. (2004). *Latency Reduction in Distributed Interactive Applications using Hybrid Strategy-Based Models.* Ph.D. Dissertation, NUI, Maynooth.

Delaney, D., Ward, T. & Mcloone, S. Reducing Update Packets in Distributed Interactive Applications using a Hybrid Approach. In Proceedings of 16th International Conference on Parallel and Distributed Computing Systems, 2003. 417-422.

Diot, C. & Gautier, L. (1999). A distributed architecture for multiplayer interactive applications on the Internet. *Network, IEEE,* 13**,** 6-15.

Durbach, C. & Fourneau, J. M. Performance evaluation of a dead reckoning mechanism. Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings. 2nd International Workshop on, 19-20 Jul 1998 1998. 23-29.

Enhua, W. & Youquan, L. Emerging technology about GPGPU. Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on, Nov. 30 2008-Dec. 3 2008 2008. 618-622.

Fiedler, G. (2006). *Networked Physics* [Online]. Available: http://gafferongames.com/game-physics/networked-physics/ [Accessed September 20th 2011].

Fiedler, G. (2008). *Reliability and Flow Control* [Online]. Available: http://gafferongames.com/networking-for-game-programmers/reliability-and-flow-control/ [Accessed September 26th 2011].

Fiedler, G. (2010a). *Floating Point Determinism* [Online]. Available: http://gafferongames.com/networking-for-game-programmers/floating-point-determinism/ [Accessed September 26th 2011].

Fiedler, G. Networking for Physics Programmers. Game Developers Conference, March 9th - 13th 2010b San Francisco, CA.

Frécon, E. & Stenius, M. (1998). DIVE: A Scalable Network Architecture for Distributed Virtual Environments. *Distributed Systems Engeneering Journal,* 5**,** 91-100.

Frohnmayer, M. & Gift, T. (2000). The TRIBES Engine Networking Model. *Game Developers Conference.* San Jose, CA.

Glimberg, S. & Engel, M. (2007). Comparison of ragdoll methods - Physics-based animation.

Greenberg, S. & Marwood, D. (1994). Real time groupware as a distributed system: concurrency control and its effect on the interface. *Proceedings*

of the 1994 ACM conference on Computer supported cooperative work. Chapel Hill, North Carolina, United States: ACM.

Greenhalgh, C., Purbrick, J. & Snowdon, D. (2000). Inside MASSIVE-3: flexible support for data consistency and world structuring. *Proceedings of the third international conference on Collaborative virtual environments.* San Francisco, California, United States: ACM.

Hardt, J. & White, K. (1998). *Distributed Interactive Simulation (DIS)* [Online]. Available: http://www-ece.engr.ucf.edu/~jza/classes/4781/DIS/project.html.

Hecker, C. (2000). Physics in computer games. *Communications of the ACM,* 43**,** 34-39.

IEEE (1998). IEEE Standard for Distributed Interactive Simulation - Application Protocols. *IEEE Std 1278.1a-1998*.

Jefferson, D. (1990). Virtual time II: storage management in conservative and optimistic systems. *Proceedings of the ninth annual ACM symposium on Principles of distributed computing.* Quebec City, Quebec, Canada: ACM.

Jennett, C., Cox, A., Cairns, P., Dhoparee, S., Epps, A., Tijs, T. & Walton, A. (2008). Measuring and defining the experience of immersion in games. *International Journal of Human-Computer Studies,* 66**,** 641-661.

Jones, D. (1998). *What is a CAVE?* [Online]. Available: http://www.sv.vt.edu/future/vt-cave/whatis/ [Accessed September 21st 2011].

Kumparak, G. (2011). *Creator of Angry Birds' Physics Engine Calls Out Rovio For Not Giving Him Credit* [Online]. Available: http://techcrunch.com/2011/02/28/creator-of-angry-birds-physics-engine-calls-out-rovio-for-not-giving-him-credit/ [Accessed September 26th 2011].

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM,* 21**,** 558-565.

Lee, B.-S., Cai, W., Turner, S. J. & Chen, L. (2000). Adaptive Dead Reckoning Algorithms for Distributed Interactive Simulation. *International Journal of Simulation: Systems, Science & Technology,* 1**,** 21-34.

Leigh, J., Johnson, A. & Defanti, T. (1997). CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments. *Virtual Reality: Research, Development and Applications,* 2**,** 217-237.

Lui, J. C. S. (2001). Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems. *Knowledge and Data Engineering, IEEE Transactions on,* 13**,** 778-792.

Macedonia, M. R., Zyda, M. J., Pratt, D. R., Barham, P. T. & Zeswitz, S. (1994). NPSNET: A Network Software Architecture for Large Scale Virtual Environment. *Presence Teleoperators and Virtual Environments,* 3**,** 265-287.

Marshall, D. (2004). *A Platform for Testing Consistency Maintenance Methods in Highly Interactive Distributed Applications.* M. Sc Thesis, NUI, Maynooth.

Marshall, D. (2008). *Improving Consistency in Distributed Interactive Applications.* PhD Thesis, NUI, Maynooth.

Marshall, D., Delaney, D., Mcloone, S. & Ward, T. (2004). Challenges in modern Distributed Interactive Application design (NUIM-CS-TR-2004-02). *Technical Report Series 2004*.

McCoy, A. (2007). *Data-Driven Modelling Approaches to Network Traffic Reduction in Distributed Interactive Applications.* PhD Thesis, NUI Maynooth.

Millington, I. (2007). *Game Physics Engine Development,* San Francisco, Morgan Kaufmann.

Mulley, G. & Bittarelli, M. (2007). *Ragdoll Physics* http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S07/final_projects/mulley_bittarelli.pdf

Nourian, S., Shen, X. & Georganas, N. (2006). *XPHEVE: An Extensible Physics Engine for Virtual Environments*, IEEE.

Nourian, S., Xiaojun, S. & Georganas, N. D. Role of extensible physics engine in surgery simulations. Haptic Audio Visual Environments and their Applications, 2005. IEEE International Workshop on, 1-2 Oct. 2005. 6 pp.

Page, E. H. & Smith, R. (1998). Introduction to military training simulation: a guide for discrete event simulationists. *Proceedings of the 30th conference on Winter simulation.* Washington, D.C., United States: IEEE Computer Society Press.

Pullen, J. M. & Wood, D. C. (1995). Networking technology and DIS. *Proceedings of the IEEE,* 83**,** 1156-1167.

Qvist, I. (2009). *Simulators And Determinism* [Online]. Available: http://ianqvist.blogspot.com/2009/08/simulators-and-determinism.html [Accessed September 25th 2011].

Roberts, D. (2004). Communication Infrastructures for Inhabited Information Spaces

Inhabited Information Spaces. *In:* SNOWDON, D., CHURCHILL, E. & FRÉCON, E. (eds.). Springer London.

Roehle, B. (1997). Channeling the data flood. *Spectrum, IEEE,* 34**,** 32-38.

Singhal, S. & Zyda, M. (1999). *Networked virtual environments: design and implementation*, ACM Press/Addison-Wesley Publishing Co.

Smed, J., Kaukoranta, T. & Hakonen, H. (2002). Aspects of networking in multiplayer computer games. *The Electronic Library,* 20**,** 87-97.

Sun, C., Jia, X., Zhang, Y., Yang, Y. & Chen, D. (1998). Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.,* 5**,** 63-108.

Swing, E. (2000). Adding immersion to collaborative tools. *Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML).* Monterey, California, United States: ACM.

Tanenbaum, A. S. (1996). *Computer Networks*, Prentice Hall.

Vaghi, I., Greenhalgh, C. & Benford, S. (1999). Coping with inconsistency due to network delays in collaborative virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology.* London, United Kingdom: ACM.

Valve (2005a). *Networking Entities* [Online]. Valve Software. Available: http://developer.valvesoftware.com/wiki/Networking_Entities [Accessed September 23rd 2011].

Valve (2005b). *Source Engine Features* [Online]. Valve Software. Available: http://developer.valvesoftware.com/wiki/Source_Engine_Features [Accessed September 23rd 2011].

Valve (2005c). *Source Multiplayer Networking* [Online]. Valve Software. Available: http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking [Accessed September 23rd 2011].

Waters, R., Anderson, D., Barrus, J., Brogan, D., Casey, M., Mckeown, S., Nitta, T., Sterns, I. & Yerazunis, W. (1996). *Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability* [Online].

Watts, S. (2010). *Gears of War 3 Multiplayer Beta, Dedicated Servers Coming in 2011* [Online]. Available: http://www.1up.com/news/gears-3-multiplayer-beta-dedicated-servers [Accessed September 30th 2011].

Webb, S. D., Soh, S. & Lau, W. (2007). RACS: A Referee Anti-Cheat Scheme for P2P Gaming. *17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV'07).* Urbana-Champaign, IL, USA Association for Computing Machinery (ACM).

Yan, J. & Randell, B. (2005). A systematic classification of cheating in online games. *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. Hawthorne, NY: ACM.

Yardi, S., Bishop, B. & Kelliher, T. (2006). HELLAS: a specialized architecture for interactive deformable object modeling. *Proceedings of the 44th annual Southeast regional conference.* Melbourne, Florida: ACM.

Yeh, T. Y., Faloutsos, P., Patel, S. J. & Reinman, G. (2007). ParallAX: an architecture for real-time physics. *SIGARCH Comput. Archit. News,* 35**,** 232-243.

Zhou, S., Cai, W., Lee, B.-S. & Turner, S. J. (2004). Time-space consistency in large-scale distributed virtual environments. *ACM Trans. Model. Comput. Simul.,* 14**,** 31-47.