Matching Algorithms for Interest Management in Distributed Virtual Environments

Sze Yeung Liu

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
The University of Birmingham
September 2011

“*Reason is immortal, all else mortal.*”

- Pythagoras

“*There is only one good, knowledge, and one evil, ignorance.*”

- Socrates

To my beloved parents ...

Acknowledgements

First of all I would like to express my greatest gratitude towards my supervisor Dr. Georgios K. Theodoropoulos for his guidance during my Ph.D. studies. Without his constant support and valuable comments, this thesis would not have been successful. I am also very grateful to my co-supervisor Dr. Iain B. Styles and thesis group member Dr. Eike Ritter, who gave me useful advice and enthusiastic help.

I also want to thank my sister Serena and my brother-in-law Huw, who provided me a family at a far place from home.

Last but not least, I would like to thank my parents for their unconditional love and untold sacrifice. I would have never made it through the hard times without them.

Abstract

Interest management in distributed virtual environments (DVEs) is a data filtering technique which is designed to reduce bandwidth consumption and therefore enhances the scalability of the system. This technique usually involves a process called "interest matching", which determines what data should be sent to the participants as well as what data should be filtered. This thesis surveys the state of the art in interest management systems and defines three major design requirements. Based on the requirement analysis, it can be summarised that most of the existing interest matching approaches are developed to solve the trade-off between runtime efficiency and filtering precision. Although these approaches have been shown to meet their runtime performance requirements, they have a fundamental disadvantage - they perform interest matching at discrete time intervals. As a result, they would fail to report events between discrete time-steps. If participants of the DVE ignore these missing events, they would most likely perform incorrect simulations. This thesis presents a new approach called space-time interest matching, which aims to capture the missing events between discrete time-steps. Although this approach requires additional matching effort, a number of novel algorithms are developed to significantly improve its runtime efficiency through the

exploitation of parallelism.

Contents

List of Figures

Chapter 1

Introduction

A virtual environment, also called a virtual reality, is a computer simulated environment that provides sensory information (e.g., sight, sound, and others) in such a way that humans can readily visualize, explore, and interact with the virtual entities in the environment. A distributed virtual environment (DVE) is intended for multiple users to interact in a virtual environment in real-time even though they are at geographically different locations. In recent years, high bandwidth and low latency network environments have facilitated the development of large-scale DVE applications such as Massively Multiplayer Online Games (MMOGs) [1] which aim to support hundreds of thousands, if not millions, of participants. Solving the real-time rendering problem is only part of the effort in building such compelling DVEs. Today, one of the major challenges for large-scale DVE development is to provide scalable data distribution mechanisms.

The simplest data distribution approach for DVE would be to have each host broadcast the state of each virtual entity (e.g., position of avatar) that it maintains. This might include, however, data that are not of interest to some receivers.

Consider a DVE with n virtual entities and m participants, using state broadcasting would result in the system regularly sending $O(nm)$ entity states to the network. As the scale and complexity of DVE grows, this approach could consume significant network resources.

To solve this problem and satisfy the scalability requirement of DVE, a technique called "interest management" has been introduced. The basic idea of interest management is simple: all participants should only receive data that are relevant to them. In order to meet this goal, the interest management system filters unneeded data before delivering updates to the appropriate participant for processing. This typically involves a process referred to as "interest matching", which matches the interests of data senders and receivers and hence determines what data should be sent to the participant as well as what data should be filtered.

If the "interests" of most of the data senders and receivers are static, the computational overhead of interest matching would be insignificant. However, in real-time DVEs, participants' "interests" are frequently changed. The interest matching process should be carried out frequently which introduces significant computational overhead. For instance, using a brute-force approach for interest matching is an $O(nm)$ process. This is obviously time consuming and is not suitable for real-time DVEs for which runtime performance is important.

Over the last two decades, interest management and interest matching have been studied extensively in many fields such as military simulations, academic and social DVEs, and commercial applications such as MMOGs. Numerous schemes have been proposed which sought to reduce the computational overhead and, at the same time, to maintain the high precision of message filtering. These algorithms have been shown to meet their runtime performance requirements;

however, they have a fundamental disadvantage - they perform interest matching at discrete time intervals. This might lead to missing interactions in large-scale DVEs that contain virtual entities of greatly varying types. For example, a military simulation may involve virtual entities such as infantry men, fighter aircrafts, artillery shells and warships which are all different in speed and size. If an entity moves around the virtual space at a high speed such that the distance travelled is sufficiently large per time-step, it might move across a small entity without notice. In other words, the events between two consecutive time-steps are ignored. Since DVE participants rely on the interest matching process to determine what data to receive, if missing interactions are ignored, the participants would most likely render incorrect scenes or perform incorrect simulations. Hence, it would not be unusual to see avatars walk through walls and bullets penetrate soldiers without wounding them. Obviously, the illusion of presence in virtual reality would break down seriously when this happens.

Furthermore, existing interest matching algorithms are designed for serial processing which is supposed to be run on a single processor. As the problem size grows, using these algorithms does not satisfy the scalability requirement of DVE since the single processor may eventually become a bottleneck. On the other hand, as shared-memory multiprocessors are common today, most of the commercial DVE applications such as MMOGs employ dual-core or quad-core machines as servers. If existing algorithms are deployed directly on these machines, the performance gain would not be guaranteed.

The research described within this thesis takes a number of decisive steps towards developing several interest matching algorithms which aims to efficiently solve the "missing-event" problem, preserve filtering precision, and enhance runtime efficiency through the exploitation of parallelism.

1.1 Contributions

The main contributions of this thesis include:

1. An efficient algorithm for interest matching, which is developed based on an existing algorithm [2]. The proposed algorithm aims to improve the space efficiency of [2] while maintaining the same computational efficiency.

2. A space-time interest matching approach, which is able to capture the missing events between discrete time-steps. Although this approach requires additional matching steps, an approximate solution and an efficient sorting algorithm are developed to minimise the runtime overhead.

3. A parallel interest matching algorithm, which enhances the runtime efficiency of discrete and space-time interest matching by dividing the workload among shared-memory multiprocessors where communication among processors take places via shared data variables.

4. A second parallel interest matching algorithm, which is suitable to apply on distributed-memory systems, where communication among processors take places via message passing. Two load-balancing approaches are developed to assist workload redistribution.

1.2 Thesis Structure

The remainder of this thesis is structured as follows:

- Chapter 2 presents a full review of existing work in interest management and interest matching algorithms.

- Chapter 3 presents a novel sorting algorithm for interest matching. It begins by reviewing basic theories of dimension reduction and temporal coherence. It then presents the algorithm and compares it with the existing approaches. Experimental evaluations are also presented, which demonstrate the effectiveness of the proposed algorithm.

- Chapter 4 tackles the problem of missing events. It presents formal definitions, theorems, and proofs of using swept volumes for space-time interest matching. An approximate solution and an efficient sorting algorithm are introduced to minimise the runtime overhead of the matching process. Finally, experimental evaluations are presented to demonstrate the effectiveness of space-time interest matching in terms of the ability to capture missing events.

- Chapter 5 introduces the parallel interest matching algorithm for shared-memory multiprocessors. It begins by presenting the basic theories of spatial decomposition through hashing. It then discusses how the discrete and space-time matching algorithms can be integrated into the parallel framework. Experimental evaluations are presented, followed by an analysis of speed-up and efficiency of parallelism.

- Chapter 6 presents the second parallel algorithm, which is designed for distributed-memory systems. It also introduces two load-balancing approaches to maximise the utilisation of the processors. Finally, experimental evaluations are presented, followed by the analysis of speed-up and efficiency.

- Chapter 7 summarises the results and contributions of this thesis and discusses the future work.

1.3 Publications

Four conference papers have been published as a consequence of the work that has culminated in this thesis. In addition, two manuscripts have been submitted to peer-reviewed journals and are, at the time of writing, under review for publication.

The preliminary design of the parallel interest matching for shared-memory multiprocessors presented in Chapter 5 was first presented at the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT) in October 2009 [3]. This paper has been nominated for the Best Paper Award of the conference.

The preliminary design of the space-time interest matching algorithm presented in Chapter 4 was first presented at the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS) in May 2010 [4]. A more detailed description of the theoretical concepts of this algorithm has been organised as a manuscript [5] and submitted for publication to the IEEE Transactions on Parallel and Distributed Systems (TPDS).

The application of parallelism in space-time interest matching presented in Chapter 5 was first presented at the Winter Simulation Conference (WSC) in December 2010 [6].

The preliminary design of the parallel interest matching for distributed-memory systems presented in Chapter 6 was first presented at the 15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT) in September 2011 [7]. This paper has received the Best Paper Award of the conference.

The review of existing and historical interest management systems presented in Chapter 2 has been organised as a manuscript [8] and submitted for publication to the ACM Computing Surveys.

Chapter 2

Literature Review

2.1 Distributed Virtual Environments

The development of DVE started in as early as 1970s. Various related topics, such as data distribution, synchronisation, rendering, and network security have been studied by the DVE research communities throughout the years. A general overview can be found in [9] and [10], which provide comprehensive details on all aspects of designing DVE systems. This section briefly describes two basic aspects: resources and communication architectures, which form the basis for the discussion of interest management schemes in the subsequent sections.

2.1.1 Resources

Scalability is perhaps the most important requirement for DVE development as it has attracted much research interest since the beginning of this field. A DVE system is said to be scalable if it would remain effective when there is a significant increase in the number of resources and the number of participants. Common

DVE resources include network bandwidth and processing power which, as discussed in Singhal and Zyda's book [9], are also the two most significant bottlenecks in a DVE. Therefore, minimising the demands on these resources is essential to achieve improved scalability and performance.

To describe the relationship between different resources, Singhal and Zyda [9] defined a resource equation as follows:

$$Resources \approx M \times H \times B \times T \times P$$

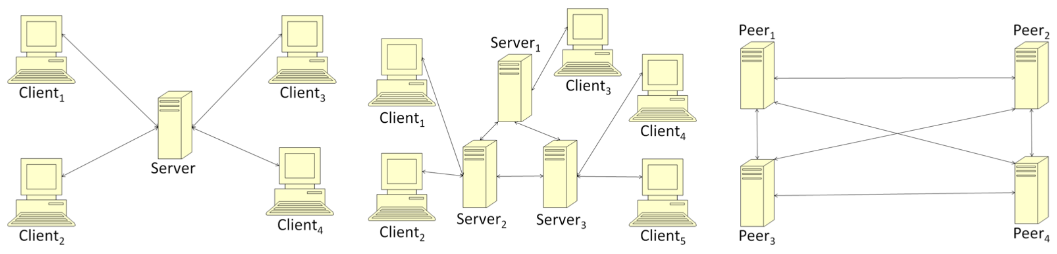
where

- M = number of messages transmitted in the DVE
- H = average number of destination hosts for each message
- B = average amount of network bandwidth required for a message to each destination
- T = timeliness with which the network must deliver packets to each destination
- P = number of processor cycles required to receive and process each message

Based on this equation, they presented various approaches to managing resources in DVEs, such as the interest management technique. They asserted that interest management effectively reduces H but at the cost of increasing M, P, and T. This introduces a second problem: if the filtering process consumes too much resources, it would be impractical to apply on real-time systems. Therefore, minimising the use of resources, such as processor cycles, is one of the primary design requirements for a practical interest management scheme. Section 2.2.1 gives a detailed discussion of these design requirements.

2.1.2 Communication Architectures

Communication architecture is an important component of a DVE since it controls how data is distributed and how participants are synchronised. Numerous architectures have been proposed throughout the years. The following are the three most typical models.



(a) Client-Server (b) Multiple-Server (c) Peer-to-Peer

Figure 2.1: Communication Architectures

2.1.2.1 Client-Server

Most of the commercial applications such as MMOGs adopt the client-server architecture. In this model, a single master server plays the most important role in the system. Each client is connected to the server and can only communicate to other clients via the server. The server is responsible to collect participant's actions from the clients, perform simulations, compute entity states, and send updates to the clients. If interest management is applied, all filtering processes would be carried out at the server since it has full knowledge of entity states. Figure 2.1(a) illustrates the client-server architecture. Obviously, this is the least scalable model among the three since the master server may eventually become a bottleneck. Moreover, the server is a potential single point of failure - if it fails, the entire DVE would become unavailable to the participants.

2.1.2.2 Multiple-Server

Figure 2.1(b) illustrates the multiple-server architecture. This model involves multiple servers, with each server serving a number of clients. A common practice is to partition the virtual world into several space subdivisions and assign one or a few subdivision(s) to each server. Hence, the workload of simulations and state updates can be shared among the server cluster. Participants that are frequently communicating with each other are supposed to connect to the same server. If a partitioning approach is used, the participants connecting to a particular server usually have avatars residing in the subdivision(s) that are controlled by the server. This model is more scalable than the client-server architecture. It is also desirable for DVE designers to apply the zone-based interest management schemes, which are reviewed in Section 2.3.

2.1.2.3 Peer-to-Peer

Figure 2.1(c) illustrates the Peer-to-Peer (P2P) architecture. In this model, there is no central repository of entity states. Instead each peer (client) maintains its own copy of the entity state based on its own simulation results and update messages from all other peers. The virtual world is usually partitioned into a number of space subdivisions which are controlled by peers. This is similar to the multiple-server architecture with the exception that servers are also clients. The P2P architecture is more fault tolerant than the two server-based models, since the failure of some peers may not render the whole system unavailable to the participants. However, putting all the data and simulations on geographically distributed clients increases the difficulty of maintaining the DVE and makes the system more vulnerable to attacks, cheats, and information exposure.

Examples of running a DVE on P2P can be found in Singhal and Zyda's book and [11]. In recent years, building DVE on a structured P2P network based on distributed hash table (DHT) has become increasingly common. Representative examples can be found in [12] and Colyseus [13].

2.2 Overview of Interest Management

Although there is no clear consensus on the origin of the interest management technique, developers have long been aware that limiting interaction and communication between participants is essential for scaling up the DVE system. In fact, this idea existed even in one of the earliest DVEs in history - Multiuser Dungeon (MUD) [14], which was created in 1978 and is regarded by many others as the origin of today's MMOGs. To enhance scalability, MUD partitions the virtual world into "rooms" and restricts the participants to see only those virtual entities that are in the same room. This approach was later classified as an example of the zone-based schemes, which are reviewed in Section 2.3.

Much of the work in large-scale DVE for military simulations began with the Simulator Networking (SIMNET) project [15] in 1980s. To enhance scalability, SIMNET addresses the $O(nm)$ state broadcasting problem by embedding a software component called "dead-reckoning", which employs extrapolation models to reduce the frequency of position update of the virtual entities. This technique, however, induces a trade-off between scalability and data consistency. The successor of SIMNET, Distributed Interactive Simulation (DIS) [16], inherits many of SIMNET's features including the dead reckoning models. In addition, it defines a more generic protocol for data communication - the protocol data units (PDUs). The entity state PDU (ESPDU) is the most common type of PDU that describes the format of an entity state, which includes fields such as entity type, position, and velocity. Although DIS itself does not provide a systematic approach for interest management, the ESPDU forms a basis for the design of message filtering mechanisms of the future DIS-based systems, such as [17] and [18].

In the literature, the first academic paper describing the use of message filtering in DVE simulation that we can find is [19]. This paper presents a filtering scheme that is based on the circular visibility of simulated vehicles. It asserts that the filtering process can be carried out either at the receiver's or the sender's network gateway. A more generalised version of this approach, the aura-based schemes, were later used in some early DVE systems such as Distributed Interactive Virtual Environment (DIVE) [20] and Model, Architecture, and System for Spatial Interaction in Virtual Environments (MASSIVE) [21]. This class of schemes are reviewed in Section 2.4.

The development of High-Level Architecture (HLA) [22, 23] was another major milestone in the evolution of DVE technology and has attracted a vast amount of research work in interest management. HLA provides a generic framework for data filtering, which includes two filtering mechanisms: value-based filtering through data distribution management (DDM) services and class-based filtering through declaration management services. The DDM services were designed to be flexible, as they can be employed for zone-based and/or aura-based schemes. Over the years, a vast amount of DDM implementations have been proposed, which are reviewed in the subsequent sections.

Although early efforts in DVE were pioneered by the defence and the academic research communities, large-scale commercial applications, such as MMOGs, have tended to be dominated since the late 1990s. Popular MMOGs like EverQuest [24], Final Fantasy XI [25], and World of Warcraft [26], all use zone-based interest management to reduce bandwidth consumption. Recently, Guild Wars [27] and Second Life [28] employed interest management schemes for content streaming, which distributes not only the dynamic entity states to the relevant clients, but also the static data such as geometry models, scenes, audio files, textures, and animations. This starts a new trend of creating very large-scale, fully dynamic DVEs. The approaches of content streaming are reviewed in Section 2.7.

2.2.1 Requirements of Interest Management

There are primarily three design requirements of interest management:

2.2.1.1 Filtering Precision

As the scale of DVE grows in terms of participants and virtual entities, using state broadcasting could consume significant network resources such as bandwidth. Therefore, interest management schemes should provide precise filtering mechanisms, which ensure the participants receive the minimal set of data that are relevant to them, in order to reduce bandwidth consumption. This is in fact the primary goal of interest management.

Apart from the network resources issue, filtering precision is also important for controlling information exposure, especially in commercial applications. For example, in a certain MMOG, a player is not supposed to know the hit point (HP) value of other characters even though he can see them in the virtual environment. If the interest management system is not carefully designed, all states of nearby characters, including their HP value and position, would be sent to the player's client computer. Although the official game client does not show the HP values on the screen, the player may still use third-party programs to access such information and thus gaining an unfair advantage.

2.2.1.2 Runtime Efficiency

As described in Chapter 1, interest management systems have to match the "interests" between data senders and receivers and hence determine what data should be sent to the participants. If the "interests" of most of the data senders and receivers are static, the computational overhead of interest matching would be insignificant. However, in real-time DVEs participants "interests" are frequently changed. The interest matching process should be carried out frequently which introduces significant computational overhead. If the cost of matching is too high, it would be unsuitable for real-time applications such as MMOGs for which runtime performance is important. Therefore, interest management schemes should provide a way to efficiently minimise the computational overhead of the matching process.

Complex filtering mechanisms may provide good filtering precision, however, their matching process usually introduces more computational overhead. This creates a "trade-off" between runtime efficiency and filtering precision. Over the years, a number of efficient interest matching algorithms have been proposed which sought to solve this trade-off. Section 2.8 gives a detailed review of these algorithms.

2.2.1.3 Event-Capturing Ability

Since DVE participants rely on the interest matching process to determine what events (i.e., messages) to receive, if the process fails to report some of the events, the participants would most likely render incorrect scenes or perform incorrect simulations. Therefore the interest management schemes should have the ability to capture and report most of the events generated by the DVE.

As simulation in most of the DVEs is based on discrete time-steps, the "missing events" usually occur between two discrete interest matching processes. Section 2.8.1 gives a detailed description on the nature of this problem and discusses the existing solutions.

2.3 Zone-based Schemes

The zone-based filtering schemes[1] limit the participants' interactions and communications within a small number of space subdivisions, or zones. They usually partition the virtual world into a number of zones; each zone contains a number of entities. Participants in the DVE are connected to these zones in order to receive events and updates that are generated from them.

The zone-based schemes can be classified into the following two categories.

[1]Various other terminologies have been used in the literature, most noticeably "cell-based", "grid-based", and "region-based". For consistency, we refer all of them as the "zone-based" schemes throughout the rest of this thesis.

2.3.1 Disjoint Zones

The disjoint zone-based approach is the most widely used interest management approach for MMOGs. The most important feature of this approach is that each participant is restricted to *one* zone at a time. Therefore he is only allowed to interact with the entities or other participants within his zone of residence, which greatly reduces the update messages he receives and generates. In addition, this partitioning approach allows the size and shape of the zones to be freely chosen by DVE developers, which are usually application dependent. Each zone has a number of exit points that allow the participants to change their zone of residence. When a participant's avatar moves into an exit point, he would be disconnected from the old zone and transported to the corresponding exit point of a new zone. Since this is a static one-to-one mapping process, no interest matching is required.

The primary advantage of using disjoint zones is its simplicity of implementation as the DVE system does not need to perform interest matching at runtime. However, this approach has a very poor filtering precision. Once a participant is connected to a zone, he must receive all events and updates from that zone; this might include a lot of irrelevant messages that are not of interest to him. In addition, this approach also has poor migration transparency. When the participant changes his zone of residence, he would be halted for a few seconds and would see a "loading screen". This seriously breaks down the illusion of presence of virtual reality.

As discussed in the previous section, MUD is the first multiplayer online game to adopt disjoint zones (which are called "rooms" in the game). Participants access the game database from inside the rooms, seeing only those entities that are in the same room and moving between rooms via the “exits” that connect them [29]. Until today, many of the most subscribed MMOGs such as Everquest [24] and Final Fantasy XI [25] still use the same idea to partition the virtual world, in order to limit the data communication between the participants and entities.

In academia, the MASSIVE system [21] is one of the earliest DVE systems that adopts this partitioning approach. The MASSIVE universe is structured as a set of disjoint zones called “worlds”. Each world defines an infinitely large virtual space which may be inhabited by many concurrent participants. The worlds are connected by portals, which allow participants to jump from one world to another.

Another example can be found in [12], which uses a structured P2P infrastructure for MMOG. It divides the game world into disjoint zones (termed “region” in the paper). Each player in the game can be a member of only one zone at a time, receiving all updates which occur within the zone. Each zone is hashed (by unique, globally known, game-assigned handles) into a Pastry overlay [30]. The node in the overlay to whom a zone is assigned becomes that zone’s coordinator, synchronising all updates to shared objects in the zone. It also becomes responsible for dissemination of updates to members of the zone using a Pastry-based application level multicast infrastructure called Scribe.

2.3.1.1 Instancing

In recent years, a number of popular MMOGs such as World of Warcraft [26], Guild Wars [27], and Final Fantasy XIV [31] adopt the use of instance zones, which are special areas, typically dungeons, that generate a new copy of the location for certain amount of participants that enter the area. This technique is called *instancing*, which can be categorised as a kind of disjoint zone-based approach.

The advantage of using instancing is similar to the original disjoint zones. Having participants in instances tends to spread out the population, instead of concentrating them, which limits the the number of potential interactions between participants and virtual entities. Since the participants in the instance do not need to receive updates that are generated outside the instance, and vice versa for the participants outside the instance, there is an overall reduction in network communications.

The disadvantage of this technique is also similar to the original disjoint zones, when participants enter or leave a zone instance, they would inevitably suffer from the "loading screen" problem.

2.3.2 Seamless Zones

Apart from disjoint zones, some DVEs such as NPSNET [32, 33] allow participants to specify an area of interest (AOI), in order to subscribe to multiple zones. A typical AOI consists of a radius of zones where the participant is joining new zones at the leading edge and leaving old zones at the trailing edge as his avatars moves around the virtual world. In Section 2.3.2.1, different types of AOIs are discussed in more detail.

The primary advantage of using seamless zones over disjoint zones is that they provide a "seamless" view of the virtual world. In other words, this approach has a better migration transparency - the participant would not see a "loading screen" when he joins a new zone. Interest matching, however, is required for this approach. Whenever the AOI moves, the system needs to determine which zone(s) it overlaps. If the number of zones is constant, the computational complexity of the matching process would be $O(m)$ where m is the number of AOIs.

Many systems that are compliant to HLA DDM [22] adopt this type of schemes. The DDM provides a flexible mechanism for publishing, and subscribing to the interest of participant through multidimensional routing space. The basic structure of the routing space is defined as follows:

- *Routing space*: A routing space is a collection of dimensions.
- *Dimension*: Dimensions are used to define regions.
- *Extent*: An extent is a bounded range defined along each dimension of a routing space.
- *Region*: Each region is defined in terms of a set of extents.

An object is said to be of interest to a federate (i.e., an individual simulation) if and only if at least one update region associated with the object overlaps with at lease one subscription region specified by the federate. A typical DDM implementation for the seamless zone-based approach is to define the AOI as subscription regions and the zones as static update regions, and carry out overlap tests between the subscription and update regions during the matching process. [34] provides an example of such an implementation.

Unlike disjoint zones, dividing the virtual world into seamless zones requires a proper partition approach. Although most of the virtual worlds are three-dimensional, partition shapes are usually two-dimensional, and can be classified into two main categories: uniform and non-uniform shapes. These partition shapes are discussed in Section 2.3.2.2 and Section 2.3.2.3.

2.3.2.1 AOI Types

NPSNET adopts the circular AOI, which is the most common type of AOI used for seamless zone-based approaches. Typically, the centre of the circular AOI is determined by the coordinates of the avatar's position. If the distance between the centre and a zone is smaller than the radius of the AOI, the participant who controls the avatar subscribes to the updates of the zone. For systems that are compliant to HLA DDM, rectangular AOIs (and zones) are often in use since regions in DDM are all axis-aligned. Rak and Van Hook have published an experimental comparison between circular and square AOI in [35]. Based on the results, they argued that smaller zones more closely approximate the circular AOI, resulting in improved filtering precision.

Other AOI shapes such as field of view (FOV) are discussed in [36]. A FOV is more refined than a circular AOI that it only subscribes to what the participant can actually see (i.e., entities in front). The area within the FOV are defined as a circular sector, which provides higher filtering precision than the traditional circular AOI. However, this approach has a disadvantage that when the avatar turns 180^o, a "lag" effect might occur to the participant's screen. Since all the things he can see now were outside his FOV at the previous moment, a large amount of entity states would suddenly be subscribed by his AOI. As a result, the DVE might not have enough time to deliver all needed data to the participant due to latency and bandwidth limitation. This "lag" effect would become more serious if the participant keeps turning his avatar around very quickly.

Bezerra et al. [36] also proposed the A^3 AOI, which is a combination of circular AOI and FOV. The A^3 AOI consists of a small circular AOI which subscribes to events within a close area around the avatar, and a large FOV which subscribes to the events that the avatar can actually see. If the avatar is turned 180^o quasi-instantly, the only entities would be affected negatively due to abrupt turning are the more distant ones. Therefore, the "lag" effect on screen would only happen to those distant entities. The authors argued that this is acceptable for most games.

Figure 2.2 illustrates the four kinds of AOIs that are described in this section. Experimental comparisons of circular, FOV, and A^3 AOIs can be found in Bezerra et al.'s paper [36].

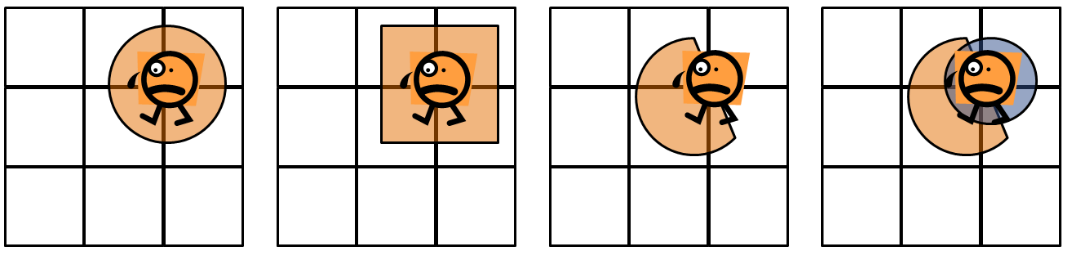

(a) Circular AOI (b) Rectangular AOI (c) FOV (180^o) (d) A^3

Figure 2.2: AOI shapes for seamless zone-based interest management

2.3.2.2 Uniform Partitioning

Schemes adopt uniform partitioning divide the virtual world into zones that are static, regular, have a uniform orientation, and have uniform adjacency. The most common shapes adopted by the existing approaches are rectangles, hexagons, and triangles, which are illustrated in Figure 2.3. For different DVEs, the reasons for adopting one of these shapes are varied, and are usually application dependent.

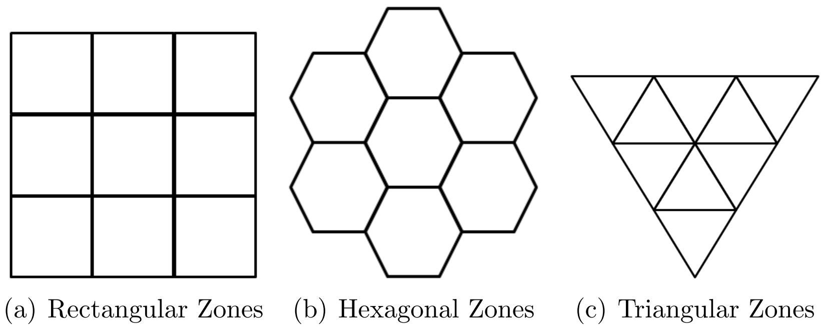


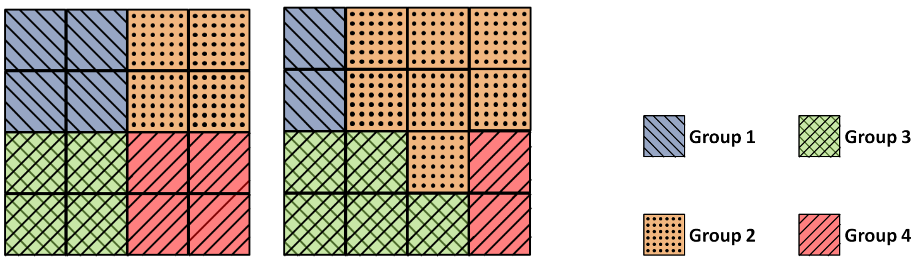
Figure 2.3: Uniform partition shapes for seamless zone-based interest management

In an early paper [37], Van Hook et al. introduced the concepts of using square zones (which is termed grid-based filtering in the paper). In this approach, multicast addresses are associated with zones defined by a square system overlaid on the terrain. State updates are transmitted to the multicast address of the zone in which an entity is located. Updates may be received from an area by joining the multicast groups for the zones that are in the area. Relevant data is selected indirectly by joining groups for the square zones that fall within an AOI. In a later paper [35], Rak and Van Hook published a detailed experimental evaluation of this approach, focusing on variables such as the size of zones, zone alignment, multicast group join rates, and AOI shapes.

Simth et al. proposed a multicast implementation for Modular Semi-Automated Forces (ModSAF) [38], which is a set of DIS-compliant software modules and applications used to construct advanced distributed simulation and computer generated forces applications. This approach creates seamless zones at two update resolutions. A given ESPDU is updated through either the high-fidelity zone address or the low-fidelity zone address, depending on the amount of elapsed time since the last ESPDU was transmitted. All other types of PDUs are updated through the low-fidelity address. Depending on the application, a simulator can subscribe to either or both of high- and low-fidelity information. Smith et al. [38] argued that this approach is especially suitable for wide area viewers (AOIs), such as a long-range radar. Since a wide area AOI may be able to subscribe to all zones, if additional filtering is ignored, it could cause a flood of traffic due to its promiscuous subscriptions. However, by only subscribing to the low-fidelity version of traffic, as Smith et al. proposed, the wide area AOI receives a greatly reduced set of data.

Srinivasan et al. [17] presents an approach which defines static multicast groups that are overlaid on square zones. Regular and irregular overlays are discussed in the paper, which are illustrated in Figure 2.4. Srinivasan et al. asserted that using regular overlays might be wasteful when the groups are being allocated to areas where little or no activity will take place. On the other hand, the irregular overlays can be arbitrarily shaped groups of adjacent square zones, which allow the developers to define multicast groups according to the terrain information. For example if an impassable mountain separates two groups of entities, the two sides of mountain can be assigned separate multicast groups. Moreover, if a large area have little activity, it can be assigned a single multicast group.

One drawback of this approach, as discussed in the paper, is that using irregular overlays may require relatively more computation overhead. This is due to that fact that there is no trivial solution for the overlap test between the circular AOI and the multicast groups of arbitrary shape. This problem can be solved by partitioning the virtual world with hierarchy data structures, which offer similar advantages as the irregular multicast groups with efficient matching algorithms



(a) Regular Overlays (b) Irregular Overlays

Figure 2.4: Multicast Group Overlays

(traversal algorithms). We describe these algorithms in more detail in Section 2.3.2.3.

In [39], Liu et al. proposed a tracking-needless grouping approach for uniform square zones. This approach assumes that a zone can actually "see" other zones through a circular vision domain. Zones that are overlapped with the vision domain form a visible set, which is precomputed before simulation begins. Similarly, each zone also has an influence domain (and a precomputed influence set) that indicates the area that the zone can influence. During runtime, each participant only subscribes to the visible set of his avatar's residing zone and the avatar's state update is sent to all members of corresponding group in the influence set. With this concept, the system does not have to frequently find out what zones are overlapped with the AOIs as the entities move around the virtual world. As a result, the computational overhead of the interest matching process can be saved. However, one assumption of this approach is that all AOIs are the same size as the vision domains. This would be inflexible and might generate many irrelevant update messages. Figure 2.5 illustrates this approach in two-dimensional space.

Zhai et al. proposed a slightly different approach called adaptive grouping scheme [40]. Unlike tracking-needless grouping, this scheme divides the visible

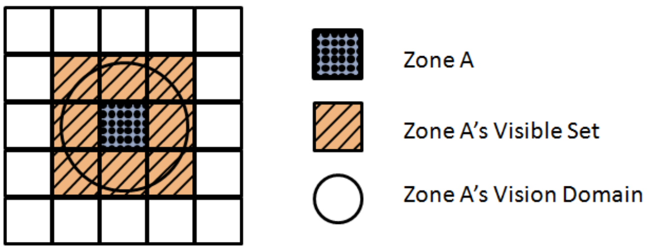


Figure 2.5: Tracking-needless Grouping

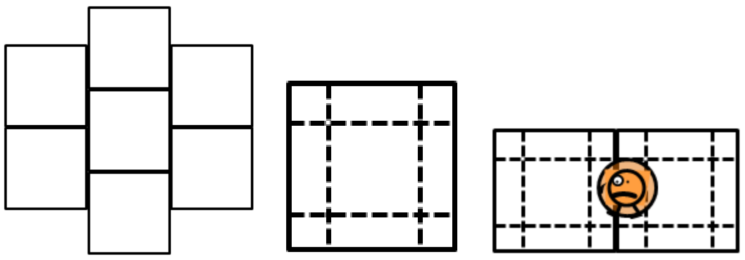
set of a zone into several subsets called "subscription sets". The zones in the subscription set are visible to participants whose AOI radius is within the range when their avatar resides in any position of the zone. Zhai et al. argued that this approach can reduce the irrelevant messages by lessening the visible zones of the participants.

The developers of NPSNET prefer hexagonal zones over square [33]. Macedonia et al. explained that since the AOI is typically defined by a radius, if squares were used, it would either need to include more area than was necessary (and thus include more entities in the AOI) or use smaller zones (which requires more multicast groups) and compute which zones the AOI should be associated with. On the other hand, using hexagons can more closely approximate a round shape AOI. When the AOI moves through the virtual, it uniformly joins and leaves the same number of hexagon zones, which would increase the filtering precision. However, Prasetya and Wu pointed out in [41] that the interest matching process with hexagons is more complicated than squares, since the overlap test between AOI and hexagons usually involves point-to-line distance formula.

In MOPAR [42], a seamless zone-based scheme for MMOG is proposed, which combines a DHT overlay and direct P2P connections. Three types of nodes (peers) are introduced in this approach: master nodes, slave nodes, and home nodes. The scheme divides the virtual world into hexagonal zones. Each zone has a corresponding home node via the DHT mapping. Master nodes and slave nodes have a position in the same zone. Each zone has at most one master node, but can have multiple slave nodes. The home nodes are virtual nodes, which means they are not necessarily positioned in the same zone as master or slave nodes. The master node registers itself to the home node of the zone. The connections between the master nodes act as the backbone for distributing data to the neighbours. Slave nodes in each zone receive the neighbourhood update from the master nodes. If there is no joining or leaving node, the correspondence between a zone and its home node is bounded, however, master and slave nodes can change their belonging zones while they are moving in the virtual world. The author argued that this approach can minimise the use of the DHT, which has overhead for the $O(\log\ n)$ hops from the source node to the destination node, where n is the number of nodes. In addition, they proposed a neighbourhood dissemination algorithm which can further minimise the overhead of message dissemination.

In [43], Lu et al. proposed a variation of the hexagonal partitioning which employs a hierarchy structure to organise the zones. The authors argued that this would enhance the scalability and managing efficiency of interest management. Moreover, this work addresses the problem of an avatar frequently bouncing in and out at the border of two zones by introducing buffer borderlines.

Prasetya and Wu proposed a Brickwork with Internal Partition (BIP) approach [41], which is designed specifically for mobile gaming. BIP begins with brickworks pattern as illustrated in Figure 2.6(a). The authors argued that this can reduce the number of neighbour zones of the original rectangular partitioning approach (see Figure 2.3(a)) and thus achieve the same result of hexagonal partitioning (see Figure 2.3(b)). Each single zone in BIP contains internal partitions which divide the cell into 9 rectangles (see Figure 2.6(b)). Similar to a typical seamless zone-based approach, participants receive state updates of the whole zone as their avatar moves around the virtual world. However, when the avatar is going to change zones, the owner participant would not receive the whole information of possible future zones. Instead, it will receive only part of it as it is crossed by the participant's AOI (Figure 2.6(c)). In this way, bandwidth consumption would be lower because there are less data to transfer during zone crossing.



(a) Brickworks Pattern (b) Internal Partitions (c) Zone Crossing

Figure 2.6: Brickwork with Internal Partition (BIP)

Prasetya and Wu also defined a metric for performance analysis specifically for virtual world partitioning. They presented detailed experimental comparisons between BIP and other typical partition shapes such as uniform triangles, rectangles, and hexagons.

2.3.2.3 Nonuniform Partitioning

Apart from partitioning the virtual world by some regular pattern, a number of schemes create zones with different shape, size, and even relative orientation.

Individual zones can be chosen freely and may be modified dynamically during runtime based on whatever is most convenient from the perspective of designing the individual zones themselves. These approaches often employ a hierarchy data structure for world partitioning.

In [44], Broll argued that the uniform partitioning model of NPSNET does not apply well to general purpose virtual environments, where objects are complex and self-contained. Broll instead proposed a hierarchy structure, which supports zones with arbitrary shapes and individual boundaries. Depending on the "camera view" of the participant's avatar, connections are established between the participant and all visible zones. This provides a seamless view of the virtual world. However, Broll's approach does not allow virtual entities other than the avatars to travel between different zones. This is somewhat similar to the restriction of the disjoint zones. Another problem of this approach is that, since it supports zones with arbitrary shapes, interest matching between the camera view (or AOI) and the zones is not straightforward and may be computationally intensive. This problem, however, is not discussed in Broll's paper.

The Spline platform [45, 46] decomposes the virtual world into various space subdivisions called "locales", which may have an arbitrary shape and may be linked together by arbitrary transformations. Each virtual entity resides in exactly one locale, where the participant's interactions are limited to the current locale and its immediate neighbours. Spline employs a binary space partitioning tree (BSP tree) to describes the boundary of a partition.

The basic concepts of BSP tree are presented in [47]. While constructing a BSP tree, the system recursively uses a partition plane to split the virtual space into two subdivisions. Each node of the BSP tree represents a partition in virtual space. Each child represents one subdivision of its parent. At the bottom of the hierarchy are the smallest size subdivisions called "leaf nodes". Choosing proper partition planes is important for BSP tree construction. It is always desirable to partition the virtual space into two subdivisions that contain the same number of virtual entities, in order to keep the tree balanced. A balanced BSP tree allows us to process a query of interest matching in $O(\log n)$ time, where n is the number of entities in the virtual space. The query algorithm starts by comparing the AOI with the root node, in order to determine which child(ren) the AOI lie on. It then visits recursively the corresponding child(ren), when it reaches a leaf node, it checks each of the associated entities for intersection with the AOI in question. For m AOIs, the computational complexity of the matching process is $O(m$ log $n)$. However, if the tree is unbalanced, in the worst case, each query could be processed in $O(n)$ time. Therefore the whole matching process would become $O(nm)$.

If there is only one participant moving in a static environment, the interest matching process would be very fast. However, a problem with hierarchy is updating it for multiple moving entities, since it changes the structure of the hierarchy. This update process would become time consuming when the number of moving entities is large. In the worse case, this may lead to a total reconstruction of the BSP tree, which takes $O(n$ log $n)$ time.

In [48], Steed and Abou-Haidar presented four partition schemes for interest management, including: quadtree, k-dimensional tree (k-d tree), constrained k-d tree, and region growing. Both BSP tree and k-d tree share similar properties, except the partition planes of the latter must be axis-aligned. In other words, after the partition process, all edges of the space subdivisions in the k-d tree would be parallel or perpendicular to the coordinate axes. Although this is less flexible and may be harder to keep the tree balanced than the BSP tree approach, the query processes can be faster, since comparing an AOI with axis-aligned rectangles are simpler than unregulated shapes.

Another feature of Steed and Abou-Haidar's approach that differs from Spline is that the partition process is only performed when the density (i.e., number of entities per node) of a leaf node is greater than a certain threshold. The choice of threshold is in fact a trade-off and is application dependent. Choosing a large threshold can reduce the height of the k-d tree and make the tree more balanced. As a result, the runtime efficiency of tree construction and matching query would be increased. However, in doing so a leaf node may contain more entities, which may reduce the filtering precision.

Steed and Abou-Haidar also pointed out that, in a k-d tree, very thin partitions are potentially problematic since the entities may frequently move between partitions. They instead proposed a constrained k-d tree, which specifies that no partition will have one edge longer than a given multiple of the other edge. This is implemented in the recursion by constraining the axis of the partition plane and also the positioning of that plane.

The region growing approach constructs the virtual world from bottom-up, which is based on a standard image processing algorithm for image partitioning. It starts from a seed point and adds adjacent points until a threshold is reached. Once the threshold is reached, a new seed point is chosen. The choice of partitions is according to the actual structure of the virtual world, which is of irregular pattern and arbitrary shape.

Typical k-d tree partition scheme can also be found in [49], which supports

FOV type AOI and places the space partitions on a P2P overlay. Zones with too many entities inside can be split and one part can be sent to another server. Rieche et al. [49] proposed five different splitting algorithms for the k-d tree:

- *SplitCenter* splits areas along the centre horizontally or vertically in exchange (i.e., if area A was split into areas B and B‘ horizontally, then area B will be split into C and C‘ vertically)

- *MaxDistToBorders* splits along the centre. This algorithm decides whether to split horizontally or vertically by maximising the average distance of the entities to the newly created border. The idea behind this is to minimise internal traffic (i.e., messages exchanged between servers) as looking and walking over borders always creates overhead

- *IntelliDistance* splits along the centre and decides whether to split horizontally or vertically, so that as few entities as possible can see the other side of the new border

- *EqualNumbers* splits along the centre and decides whether to split horizontally or vertically in a way that makes the number of entities in the new zone is as equal as possible. The expectation is that this algorithm will perform better than *SplitCenter* in terms of load-balancing

- *VarAreas* splits horizontally or vertically as *SplitCenter* does. However it places the border not along the centre but at the centroid of the entities. As with *EqualNumbers* the expected result is better load-balancing

In addition to splitting, adjacent zones with low numbers of entities can be merged for a lower number of internal messages. The experimental results presented in the paper show that the *VarAreas* algorithm can effectively minimise internal traffic, where *IntelliDistance* performs best on balancing the load (i.e., number of entities) between servers.

In [50], Van Hook et al. proposed a clustering approach that employs quadtrees to reduce the computational cost of the interest matching process. A quadtree is used to partition two-dimensional space by recursively subdividing it into four equal-size, axis-aligned subdivisions. Since the partition planes are fixed, partitioning by quadtree is less flexible than the BSP tree and k-d tree. In Van Hook et al.'s approach, a maximum depth of partitioning is used to limit the height of the tree. This type of control is stricter than Steed and Abou-Haidar's threshold approach [48]; the processing time of tree construction and matching query would be bounded by a constant, even if the tree is unbalanced.

In [51], a partition approach based on N-tree (N-tree is the generalisation of quadtree) is proposed. Its underlying partition mechanism is similar to Van Hook et al.'s approach. In addition, a predictive method is employed to calculate the level of awareness between entities, in order to reduce the frequency of update messages.

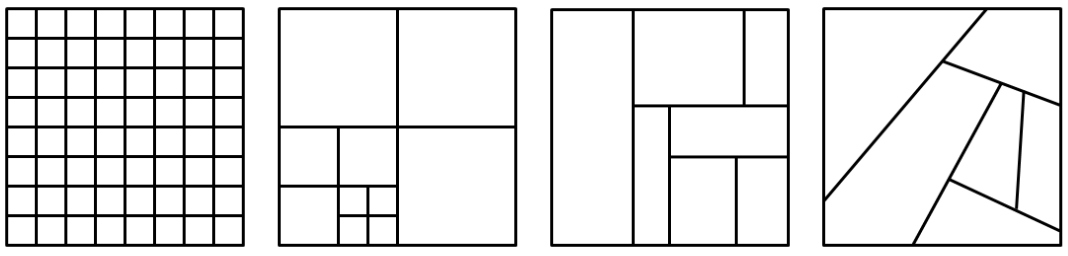

(a) Uniform Squares (b) Quadtree (c) k-d Tree (d) BSP Tree

Figure 2.7: Partition Patterns

Figure 2.7 illustrates the partition pattern of uniform square (***US***), quadtree

(***QT***), k-d tree (***KT***), and BSP tree (***BT***). In general, their relationships can be expressed as:

$$\boldsymbol{US} \subseteq \boldsymbol{QT} \subseteq \boldsymbol{KT} \subseteq \boldsymbol{BT}$$

Although hierarchy structures are common, they are not the only variable partitioning approaches presented in the literature. In Voronoi-based Overlay Network (VON) [52], a Voronoi diagram (see Figure 2.8) is used to defined the virtual world and solve the data distribution problem. Given a number of points, or "sites", in a two-dimensional world, a Voronoi diagram partitions the world into the same number of zones (Voronoi regions), such that each Voronoi region contains all the points closer to the region's site than to any other site. For a given participant, a "AOI neighbour" is defined as the Voronoi region whose position are within the participant's AOI. Each Voronoi region is enclosed by a set of other Voronoi regions, known as its "enclosing neighbours", a participant's AOI may also partially overlap with some set of Voronoi regions, this set is termed its "boundary neighbours". This Voronoi diagram is continually recomputed by each avatar.

When an avatar moves, the update is broadcast to all connected neighbours. In addition to rendering the update to the participant, each enclosing neighbour will perform checks to determine if the move requires adding new boundary neighbours (of which it may not currently be aware). In this manner no individual participant need ever be aware of the entire diagram but only that covered by his boundary and enclosing neighbours. This scheme is an elegant way of localising update traffic between participants in line with the geometric relationships

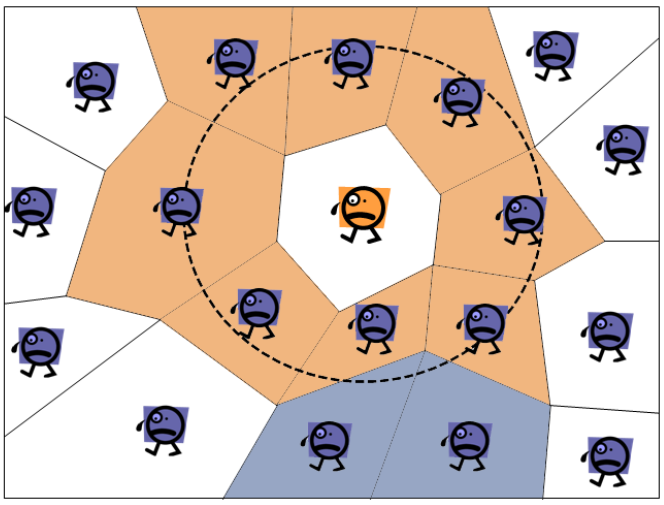


Figure 2.8: Voronoi Diagram

between avatars. However, it does come at the significant overhead of continual re-computation of the Voronoi diagram itself, an operation requiring $O(m \log m)$ time, for m neighbours in the boundary set. Once again in this approach we can see that the basic trade-off encountered in interest management repeated: filtering precision vs. runtime efficiency.

In [53], Buyukkaya and Abdallah presented another Voronoi diagram approach that is similar to VON. In addition to the partitioning scheme, this work focused more on the discovery and interaction of both static and dynamic entities, making it a more practical solution for data distribution.

2.3.3 Granularity

Choosing a proper granularity is one of the major considerations for all zone-based schemes. For a static partitioning of the virtual world, a significant trade-off must be made. If the zones are large, each zone would contain a large number of virtual entities and thus the participants might receive a large amount of irrelevant data. On the other hand, if the zones are small, the number of zones as well as the number of multicast groups would become large, and therefore the entity movement between zones would be more frequent. This increases the chance of subscribing to and unsubscribing from multicast groups as the participants move around the virtual world, resulting in an increase in management overheads.

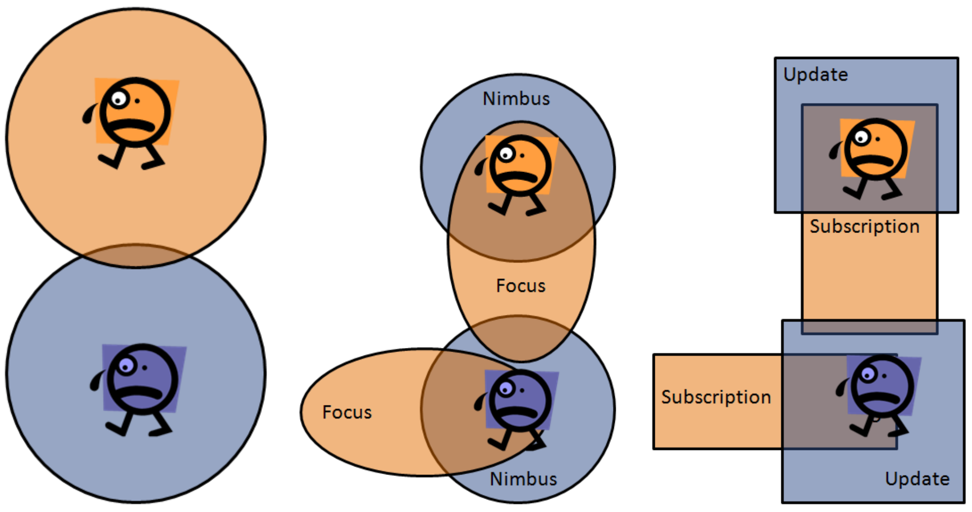
The hierarchical structures reviewed in Section 2.3.2.3 also suffer from the same problem but in a different form - a trade off must be made when choosing a proper granularity of the leaf nodes or a proper height of the hierarchy. Van Hook et al. [50] and Steed and Abou-Haidar [48] tried to maintain a balanced hierarchy by setting a maximum height or a population threshold. These are practical solutions, however, one should also consider the characteristic of the application, and the processing power and communication speed of the entire DVE system when choosing the optimal granularity.

A study presented in [34] argued that in a system with fast CPUs and slow communication network, the optimal zone size would be rather small. On the other hand, in a DVE with slower CPUs and faster communication the optimal zone size would be rather large. Moreover, in [54] the authors suggested that adaptive reassignment of zones during runtime may be the best way to achieve and maintain an optimal use of resources. Detailed experimental evaluations on the optimal granularity of certain types of seamless zone-based schemes can be found in [35] and [55].

2.4 Aura-based Schemes

The aura-based approach is another large class of interest management approaches. It was originally proposed by Bassiouni et al. [19] (see Section 2.2) and was later used in the DIVE system [56, 20]. The basic idea of this approach is to use auras to represent the interests of each entity or participant. An aura can be defined as a spatial scope which is similar to the AOI described in the previous section. When two auras overlap, a connection between the two owners of the auras is established and messages are exchanged through the connection. This approach provides a much more precise data filtering mechanism than the zone-based approaches; however, since the system needs to periodically test the overlap status of the auras, more computational effort is required for interest matching. The computational complexity of the matching process is $O(nm)$ by brute-force, where n is the number of entities and m is the number of participants. A detailed experimental comparison between a seamless zone-based scheme and an aura-based scheme can be found in [57], which focuses on evaluating the cost of leaving/joining multicast groups as well as the filtering precision.

The MASSIVE system [21] adopts an alternative version of the aura-based approach called "focus and nimbus" [58]. The "focus" represents the allocation of attention of the participant (or his avatar), while the "nimbus" represents the observed entity's manifestation or observability. A participant (or his avatar) is aware of an entity if and only if his focus overlaps its nimbus. During runtime, focus and nimbus are attached to the entity or the avatar's current position and therefore moves dynamically. The system needs to perform overlap tests frequently to keep track of their overlap status. In addition, MASSIVE also takes into account the level of awareness. The more an entity's nimbus is within a participant's focus the higher the level of awareness of him to it. The system hence is able to calculate the awareness value of information such as graphics and audio, which is used as the basis for managing the interaction between the entity and the avatar.



(a) Pure Auras (b) Focus-Nimbus (c) HLA DDM

Figure 2.9: Aura-based schemes

Much of the existing work in aura-based interest management has focused on developing robust interest matching algorithms. These algorithms are reviewed in Section 2.8.

2.5 Class-based schemes

Zone-based and aura-based schemes do not consider filtering aspects other than spatial information. For example, in military simulations, a radar can detect all nearby aircraft except stealth fighters. Therefore, the position of the class of stealth fighters should not be sent to the radar. Normally, the previous three types of schemes do not support this kind of filtering. One exception is the third-party objects [59] introduced in MASSIVE-2. These objects are created on the basis of spatial information, but can also be used for other aspects such as attribute of entities.

A filtering approach based on message type can be found in e-Agora [60, 61], which is a testbed DVE system aimed at social interaction and culture content dissemination. In this system, participants can see each other by the help of avatars and communicate by chat and gestures. An approach that applies "domains" on top of an aura-based scheme is used. It defines eight domains including (W)orld, (S)pace, (R)ooms, logical (G)roups, (C)hat, (N)avigate, (P)lay games, and (E)dit objects. The first three domains are related to spatial information, where the others are based on message or action type. A participant can issue the sets: {C, G} for a chat to a group of participants, {E, G, W} for editing within a specific group and a world, or {E, W} for cross-group editing. The combination of domains restricts the scope of the data variable, and thus reducing the data communication within the DVE.

Ding and Zhu [62] proposed another approach based on participant's behaviours. Typical behaviours such as *point*, *grasp*, *touch*, *gaze* and *talk* are predefined which describe the interaction between the participants. An interest degree $I \in [0, 1]$ is used to represent the degree by which an entity is interested, which is somewhat similar to the "level of awareness" of the MASSIVE system. According to certain application, a user defined threshold $\delta \in [0, 1]$ is used to judge whether $I > \delta$. If this is the case, the status update of the corresponding entity-participant pair would be communicated. When there is more than one kind of behaviour between two participants, an overall influence is calculated as $I^* = \sum I$.

The declaration management services of HLA [22, 63] provide a more generic class-based/attribute-level filtering scheme. Specifically, a federate can subscribe to attributes of an object class, indicating it wishes to receive notification whenever that attribute of any object instance of that class is modified. Furthermore, a federate may subscribe at any point in the class hierarchy. Attributes of a class are inherited by the subclasses of that class. Subscribing to an attribute of a class at a certain level in the hierarchy automatically subscribes the federate to that attribute as it is inherited by all the subclasses of that class. One important difference between the class-based schemes and the DDM schemes described in the previous sections is that interest matching of the former can be precomputed, and thus the matching results can be cached and are not modified dynamically during runtime. The DDM schemes, on the other hand, should perform interest matching at runtime due to frequent region updates.

The Interest Operator (IO) [64] is another class-based/attribute-level scheme. An IO is an operator that accepts parameters, performs a calculation and returns results. Results are whether interest criteria are met. One key difference between IO and the attribute of HLA is that different classes (with no common class ancestry other that the hierarchy root) may support the same IO. Moreover, in the HLA an object cannot specify its interest in any attribute or interaction, where the IOs allow developers to establish criteria that may be of interest to both objects or participants.

2.6 Hybrid Schemes

Many systems attempt to combine different interest management schemes for fine-grained filtering as well as reducing computational overheads. In fact, *pure* zone-based or aura-based schemes are rare and only exist in early work such as [19] and [37]. Many schemes reviewed in the previous four sections are actually hybrid schemes. For example, the latest version of DIVE [65] uses "world hierarchy" to provide zone-based filtering and performs aura-based filtering within each world. The MASSIVE system first combined disjoint zones and aura-based filtering; in its third generation, MASSIVE-3 [66] integrated the "locales" method of Spline with the original system to facilitate seamless zone-based filtering. Abrams et al. proposed a three-tiered architecture [67] which provides three tiers of message filtering hierarchically. The first and second tiers perform zone-based and aura-based filtering; in the third tier, protocol dependent filtering is carried out. Other similar zone/aura-based hybrid approach can also be found in [68]. In [69], Pan et al. proposed a slightly different zone/aura-based hybrid approach for P2P DVE. This work focuses on reducing not only the computational overhead of aura-based interest matching, but also the communication overhead of exchanging aura (or AOI) information between peers.

The HLA itself is in fact a good example of a hybrid interest management system, as it provides class-based and value-based filtering as the same time. In the DDM interest matching process, an object is said to be of interest to a federate if and only if the following two conditions are satisfied:

1. At least one of the object's attributes is subscribed to by the federate (through declaration management services); and

2. at least one update region associated with the object overlaps at least one subscription region of the federate (through DDM).

The use of "region" in DDM is flexible: it can be employed for both zone-based and aura-based filtering. HLA DDM implementations of zone-based and aura-based hybrid can be found in prior work such as [70, 71, 72, 73, 74].

The DARPA Synthetic Theater of War (STOW) programmes that began in mid-1990s were intended to support military training exercises with tens of thousands of entities. The hierarchical filtering approach [75] is one of the interest management approaches that has been used to improve the scalability of the STOW exercises. In this approach, filtering schemes are organised into three tiers. The first tier uses multicasting network technology which routes only the relevant entity states to the simulations that might need that data. The second tier filters execute a Modelling Interest Language (MIL) which carries out filtering on the receiving host before passing the entity states to the local simulation. The third tier filters are a set of user defined boolean functions, which further optimise the data set that passes the MIL filters. The first generation of STOW exercises use DIS protocols with customised PDUs. After DIS was succeeded by the HLA standard, an HLA-compliant implementation called "RTI-s" was developed [76].

The DIS Filter-Analyzer (DFA) [18] developed by The Air Force Distributed Mission Operations Center of Excellence (DMOC) is a set of software filters for DIS-based exercises. It reads DIS PDUs from both LANs and determines if they pass or fail the filter criteria set by the user. Several filter types are available within the DFA: DIS PDU type, DIS Site, Application, and Entity ID, Enumeration filter, and the range filter. The other capabilities include DIS Version translation, a teleport function, a frequency filter, and a delay for PDU processing. Multiple filters are being applied at the same time. For a PDU to be forwarded, it must pass all filters.

The Push-Pull framework [77] derives the DVE data distribution process at the infrastructure level in a "bottom-up" manner. In this framework, data are represented by a set of distributed shared variables (DSVs). The DVE data distribution process can be translated into read and write operations to the DSVs. Each DSV is associated with a particular owner node (e.g., a server). All writes to a variable are sent to that variable's owner, thus ensuring consistency via this master copy. All nodes who are not the owner of a particular variable are termed "replicators" of that variable. These maintain a proxy locally which can be read and written by the hosted application in a transparent way. The state of proxies can be maintained in one of two processing modes:

- *Push* wherein each write at the master results in an update message sent to the proxy
- *Pull* wherein each read at the proxy results in a read-request/read-response exchange initiated by the proxy

Under the Push-Pull framework, several algorithms were proposed which sought to minimise the number of message passing of the read-write operation. A later paper [78] shows that how the Push-Pull framework can be combined with a zone-based interest management approach, which partitions the virtual world into uniform three-dimensional grids. A quantitative study of the behaviour of this combined approach when it is used in a load imbalanced DVE is given in [79].

2.7 Content Streaming

The discussion of this chapter so far has focused on how interest management systems distribute dynamic data (e.g. state of moving entities) to the virtual world participants. The static data, such as geometry model of the virtual entities, scenes of the virtual world, pre-recorded audio files, textures and cutscenes, are beyond the concern of these approaches. For most of the existing DVEs, if the developers want to modify the static data, they usually do it in an offline manner. They would either send out a new copy of software to the participants, or simply ask them to download an update patch. Therefore, the participants' machine is required to pre-install all up-to-date static data before simulation starts. It is important to note that, however, most of these data are not of interest to the participants during most of the runtime. As the scale of virtual world grows, storing a large amount of irrelevant static data may consume significant storage space.

The content streaming technique was introduced to address this problem. This technique distributes the content of the virtual world, including both static and dynamic data, to the participants in a real-time manner. It allows the participants to enter the virtual world without a complete installation of content. During runtime, content is only transmitted to the participants that are interested in it. This filtering process is based on various interest management schemes. Moreover, one important characteristic of static data is that their size is usually many times larger than dynamic data, resulting in a serious bandwidth consumption. Therefore, interest management schemes with high filtering precision are often preferred by the content streaming systems.

In an early paper [80], Schmalstieg and Gervautz presented a strategy for managing network transmission of geometry data in DVEs. This approach adopts a client-server architecture, which allows the participant (client) to request geometry from the server based on individual levels of detail (LOD) instead of downloading the complete geometry model of virtual entities or even the entire scene. A typical aura-based interest management scheme is used to filter irrelevant contents. When the participant's avatar approaches an entity and their auras overlap, the LODs of the entity's geometry model are transmitted to the participant from coarser to finer resolution. This avoids the transmission of high-resolution geometry data that are never actually used, and thus reduces the bandwidth and rendering costs.

In [81], Teler and Lischinski presented a content streaming approach that supports not only the geometry models, but also image-based entity representations. In addition, this approach includes an online optimisation framework for remote rendering, which uses path prediction and a cumulative benefit (i.e., transmission priority) function in order to more efficiently exploit the available bandwidth. The online optimisation algorithm is based on a greedy optimisation strategy. The algorithm computes an added benefit integral for all relevant representations of entities that are predicted to become visible, and transmits the representation with the best benefit to cost (i.e., duration of transmission) ratio.

The CyberWalk system [82] is a web-based DVE based on on-demand transmission of content. It employs multiresolution modelling techniques for caching and prefetching entities in a client at various granularities, with nearby entities at higher resolution and distant entities at lower one. Similar to Schmalstieg and Gervautz's approach, CyberWalk uses an aura-based interest management scheme to filtering irrelevant geometry models. However, there are some major differences between these two approaches. Firstly, Schmalstieg and Gervautz's approach transmits multiple models of the same object at different resolution, redundant information will be sent. CyberWalk solves this problem by applying a progressive mesh technique for model transmission. Therefore, no redundant information needs to be sent across the network. Moreover, in Cyberwalk an entity is calculated based not only on the distance of the entity from the participant's avatar, but also on the size of the entity concerned, the depth of sight of the participant, and the resolution of the display device, allowing the entity models to be transmitted at a low cost. Finally CyberWalk employs a prefetching mechanism which predicts entities that may be accessed in the near future.

In the social DVE Second Life [28, 83], the content of its virtual worlds is user-created and real-time editable. Therefore, updating static data by releasing new patch or new copy of software is nearly impossible. Second Life solves this problem by employing content streaming as its key feature. Besides the geometry model of the virtual entities, texture and audio data are also delivered to participants through streaming. Progressive techniques are exploited to allow participants to put thousands of entity models, large textures and a large number of audio sources into a scene, and then to stream only the LOD that are needed. In addition, Second Life employs a seamless zone-based scheme to partition the world into uniform square zones. Each zone is managed by a "simulator machine", which handles not only data distribution, but also different kind of simulations within the zone. The simulator machine communicates only to the four nearest neighbours, therefore there would be no transactional scaling problem as the world becomes large. As the participant moves around the virtual world, he maintains a streaming connection only to the nearby simulator machines. The simulators compute the entities and information that are relevant to him, and only transmit the data of those entities that are either newly created or that have changed. This allows a thin client software to be the only thing a participant needs to download and install.

In recent years, some popular MMOGs have begun to use P2P techniques to distribute game content during runtime. This provides large-scale real-life testbeds for the content streaming technology. In Guild Wars [27, 84], players only need to download a 90KB thin game client (launch program) before starting the game. When they join the game world, the relevant data continue to stream to them in the background. Generally, players can play the game without seeing a "loading screen". Traditional MMOGs such as Everquest [24] and Final Fantasy XI [25] update game content by performing scheduled update or server maintenance. During the time the players are forced to exit the game and have to download an update patch after the maintenance is finished. Guild Wars, on the contrary, update the game content dynamically without interrupting the players' game-play experience or causing the normal patching delays. This provides very good "patching transparency" and the ability to update only the specific files that need to be changed. It is necessary to note that, unlike the DVE systems that were mentioned previously in this section, in Guild Wars once a player has received the files on his hard drive he will not have to reacquire the data unless they are changed. In this way the size of the game on the computer will grow as the game content grows but he will only download the relevant data once.

The HyperVerse [85] is a web-based DVE system that uses a Torrent-based protocol for data distribution. It consists of a highly structured federated backbone and a loosely structured P2P client overlay. The client is similar to a web browser which is heterogeneous and contains no predistributed information of the virtual world. Data are hosted by a massive amount of public servers, which are organised in a completely distributed manner. Interest management in HyperVerse is somewhat similar to an aura-based scheme, which is illustrated in Figure 2.10. Each participant has three circular auras with radii d, $d + \Lambda$ ($\Lambda < \Delta$), and $d + \Delta$ ($\Delta \geq 0$), respectively, around the position of his avatar. The inner aura represents the visibility of the avatar. Initially, all entity and terrain data within the outer aura are delivered to the participant. The primary goal of this scheme is to mitigate the effects of message latency. It allows the avatar to move within a distance Λ without requesting further data. Whenever the avatar has moved more than Λ, the auras must be re-calculated and all entities and terrain data within the new outer aura need to be delivered to the participant. The introduction of the threshold Λ allows for more time for requesting these data since the participant's inner aura (i.e., visibility) is still $\Delta - \Lambda$ away from areas for which no information has been prefetched. Δ and Λ are chosen by participants' computers according to their individual capabilities. If they are chosen properly, message latency can be tolerated without having visual effects. HyperVerse also exploits data locality by applying caching techniques. Entity and terrain data with high "hit rate" can be cached locally, in order to prevent needless message communication.

Flowing Level-of-Details (FLoD) [86, 87] is a P2P streaming framework that allows participants to retrieve relevant content of the DVE. All content is initially stored at a server, and clients obtain it by streaming from either the server or other clients through P2P overlay. FLoD adopts a typical seamless zone-based

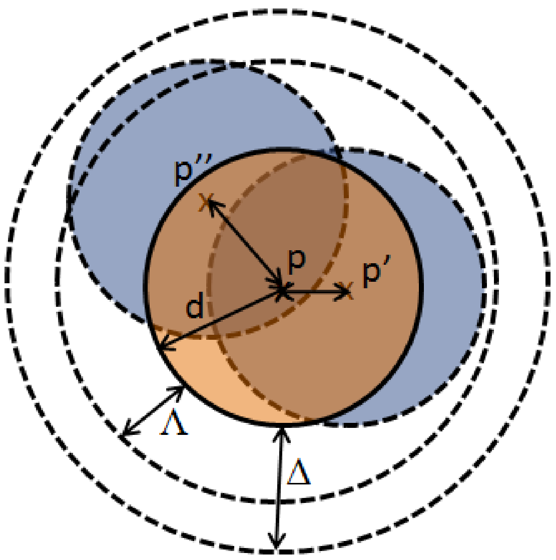


Figure 2.10: Auras of HyperVerse

interest management approach which partitions the virtual world into uniform squares. Interest matching between AOI and the zones is done in a fully distributed manner, as each peer is able to carry out this process locally. An important assumption of the design of FLoD is that avatars tend to see each other or crowd at certain hotspots. Therefore, a participant may have overlapped visibility with his neighbours. Content data may therefore already be possessed by neighbouring participants and by requesting data from the neighbours first, the server can be relieved from sending the same data repetitively. FLoD exploits this property by using a Voronoi-based approach that is similar to VON [52] as its P2P overlay. This approach organises the virtual space into a Voronoi diagram in order to support neighbour discovery, which has been described in Section 2.3.2.3.

2.8 Interest Matching Algorithms

The interest matching algorithms are designed to solve the “trade-off” between runtime efficiency and filtering precision. They have usually been applied on high precision filtering schemes, such as HLA DDM, which ensures the participants receive the minimal set of data that are of interest to them. In addition, they provide a way to efficiently reduce the computational overhead of the matching process.

In an early paper [37], Van Hook et al. pointed out that the matching process of the aura-based approach (referred to as “object-based approach” in the paper) could be computationally intensive. To solve this problem, one might use crude grid-based filtering to cull out many irrelevant entities before a more compute-intensive procedure is carried out for finer discrimination. In a later paper [50], Van Hook et al. proposed a clustering approach through the use of multidimensional binary trees to reduce the computational cost of the matching process.

Experimental comparisons for different matching approaches in terms of computational overheads and filtering precision have been presented in [88, 89]. [90] have provided a theoretical analysis of the computational complexity of the interest matching process. The paper showed by reduction from Binary Search that the matching process requires a total time with a lower bound in $\Omega(nlogn)$ and a upper Bound in $O(n^2)$, where n is the number of runtime data declarations made by the federates.

Morgan et al. [91] proposed a collision detection algorithm for aura-based interest matching. The algorithm uses aura overlap for determining spatial subdivision. The authors argued that it more accurately reflects the groupings of entities that may be interacting than existing collision detection algorithms and provided performance figures to demonstrate its scalability.

Recently, more robust matching algorithms [92, 93, 94] based on dimension reduction were proposed. These algorithms are designed specifically for HLA-compliant systems, and thus adopt the use of rectangular auras (i.e., regions of the HLA). The basic idea of dimension reduction is to reduce the multidimensional overlap test to a one-dimensional problem, which is defined in Section 3.2.

In Raczy et al.'s paper [92], a sort-based DDM matching algorithm based on dimension reduction is proposed. It projects the regions on each axis and uses heap-sort to sort the projections in order to find out the overlap information. The computational complexity of such sorting is $O((n+m)\log(n+m))$, for n is the number of update regions and m is the number of subscription regions.

Liu et al.'s approach is adopted by the Lucid Platform [2], which is a commercial middleware for MMOG development. It employs a HLA-compliant matching algorithm based on caching. Similar to Raczy et al.'s approach, it projects the regions on each axis and performs sorting. However, instead of finding the overlap information every time, it caches the matching results of the previous time-steps. In environments where entities make relatively small movement between consecutive time-steps, interest matching can be processed in linear time.

Pan et al. [94] also designed a sort-based matching algorithm for HLA-compliant systems. The efficiency of this approach relies on the assumption that only a small portion of the entities are updated at each time-step of simulations. The authors argued that it has better storage and computational scalability than Raczy et al.'s algorithm in many cases.

2.8.1 The Missing Event Problem

The interest matching algorithms discussed so far focus on enhancing the computational efficiency of the matching process; they might, however, lead to a "missing event". This problem occurs when an entity moves at a high speed such that the distance travelled is sufficiently large per time-step, it might move across another entity without notice.

Figure 2.11 illustrates the "missing event" problem in two-dimensional space. An aura U moves across a static aura S over the time interval [0,1]. However, neither at time $t = 0$ nor at $t = 1$ can a discrete interest matching algorithm determine that they indeed overlap each other. The event is thus *missed*.

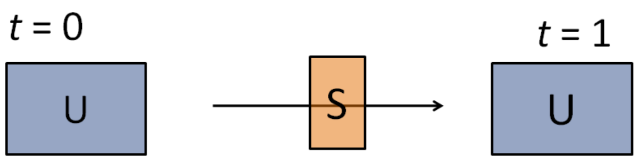


Figure 2.11: Missing Event

There are two simple general solutions to this problem, but each of them must make a trade-off. The first solution is to specify *large-enough* auras. For example, an avatar's original line-of-sight is 500 metre radius. If we extend it to 1 km, many of the missing events (generated by fast moving auras) could be captured. However, extending the line-of-sight may result in the avatar seeing things he is not supposed to see. Therefore, the participant controlling the avatar may receive a large amount of irrelevant messages resulting in a bandwidth overheard. In Fujimoto's book [63], a similar solution is discussed in Section 8.6.5.

The second simple solution is to reduce the time-step of simulation and increase the frequency of performing discrete interest matching. For example, suppose that the original time-step is 1 second. If we reduce it to 0.1 second and carry out interest matching 10 times within 1 second, we could avoid many of the missing events. This is however a very time consuming process. Furthermore, it would be meaningless to perform extra matching without aura updates. Additional overhead would be introduced when the frequency of aura update is increased.

Morgan and Lu proposed a predictive interest management approach [95] to address the missing event problem, which is a message exchange policy based on predictive modelling of entity movements. It aims to vary message exchange between nodes based on the likelihood that entities will influence each other in the near future. Hence, it can avoid missing interactions involving aura and regionalisation by enabling the DVE to ensure message exchange occurs at an appropriate frequency before, during and after overlap of auras. This paper, however, has not evaluated the performance of predictive interest management (in terms of runtime efficiency and the ability to capture missing events).

2.9 Summary

This chapter has seen how the problem of interest management has been independently encountered and explored in various related areas of DVE research, such as military simulations, academic and social DVEs, and commercial applications such as MMOGs. Though approaches differ with application scenarios, most of them have been shown to solve a simple but fundamental trade-off: between filtering precision and runtime efficiency. As we have described previously, the hybrid approach is perhaps the most widely used approach in interest management.

Many schemes first partition the virtual world into zones, in order to reduce the runtime overhead of matching. They then carry out a interest matching process to preserve the precision of filtering. These approaches are particularly common among HLA-compliant systems.

Most of the existing interest matching algorithms in the literature, as we have reviewed in Section 2.8, focus on improving the runtime efficiency of the compute-intensive matching process. They however ignore the missing events as they perform overlap tests for auras and AOIs at discrete time intervals. This leads to the problem of missing interactions as well as incorrect simulations. To solve this problem, this thesis present a "space-time" approach, which adds "time" as an additional dimension of interest matching, in order to capture the missing events. Although the new approach requires additional matching steps, an approximate solution and an efficient sorting algorithm have been developed to minimise its computational overhead.

Furthermore, the existing matching algorithms can be considered as "serial algorithms" since they are designed for single processor or are developed with serial processing in mind. As the problem size grows, the single processor may eventually become a bottleneck. In addition, the performance gain of the serial algorithms would be limited if they are deployed on multiprocessor computers. To solve these problems, this thesis presents a parallel framework for interest matching, which facilitates workload sharing by dividing the matching process among multiple processors. We will also show that the space-time approach can be easily integrated into the parallel framework, and thus enhancing the overall runtime efficiency of space-time interest matching.

Chapter 3

An Efficient Sorting Algorithm for Interest Matching

This chapter presents an efficient sorting algorithm for interest matching, which is designed for HLA-compliant interest management systems and aims to enhance the runtime efficiency of the filtering process. The proposed algorithm is developed based on the concept of dimension reduction, which has been used in [2, 92, 93, 94]. It improves the design of [2], in order to achieve a better space efficiency.

The proposed algorithm was originally published as a part of [3]. It also forms the basis of several interest matching algorithms that are proposed in this thesis.

3.1 Region

For the sake of consistency, "auras" will hereafter be referred to as "regions" based on the terminology of HLA. There are two kinds of regions in DDM: an update region is associated with a virtual entity which indicates the effective area of the entity's state update; where a subscription region represents the area that is of interest to a participant. To make the algorithms compatible with the HLA-based DVEs we adopt the use of axis-aligned auras since all regions in the HLA are axis-aligned. Other aura shapes can be converted to axis-aligned auras by computing their orthogonal projection on the Cartesian coordinate axes.

Formally, a region can be regarded as a set of points $\boldsymbol{R}$, which is defined in **Definition** 1.

Definition 1. Let MIN_d, MAX_d be real numbers with $MIN_d \leq MAX_d$, for $d = 1, 2, ..., n$. A region $\boldsymbol{R}$ can be defined as

$$\boldsymbol{R} = \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge MIN_d \leq x_d \leq MAX_d, \\ for \quad d = 1, 2, ..., n\}.$$

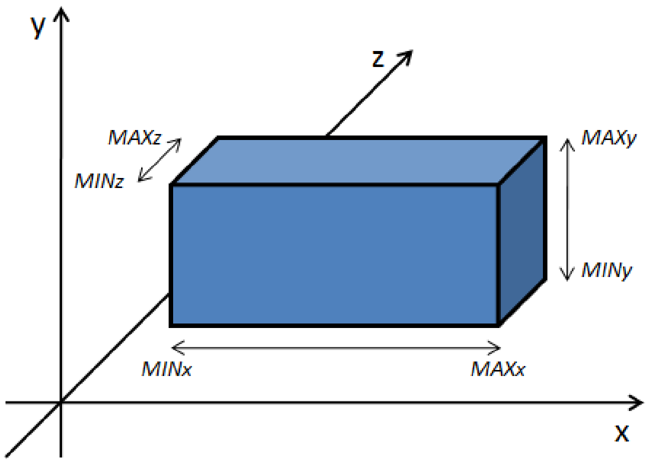


Figure 3.1: A Region in Three-Dimensional Space

Figure 3.1 illustrates a region in three-dimensional space. $[MINx, MAXx]$,

$[MINy, MAXy]$, and $[MINz, MAXz]$ are the three orthogonal projections of the region on the Cartesian coordinate axes. The orthogonal projections are defined in **Definition 2**.

Definition 2. Let MIN_d, MAX_d be real numbers with $MIN_d \leq MAX_d$, for $d = 1, 2, ..., n$. An orthogonal projection of region $\boldsymbol{R}$ on d-axis can be defined as

$$P_d = [MIN_d, MAX_d] = \{x_d \in \mathbb{R} \mid MIN_d \leq x_d \leq MAX_d\}$$

for $d = 1, 2, ..., n$.

A region can also be defined as the Cartesian product of its orthogonal projections, which is given in **Definition 3**.

Definition 3. Let P_d denote the orthogonal projection of region $\boldsymbol{R}$ on d-axis. $\boldsymbol{R}$ can be defined as Cartesian product of all P_d, for $d = 1, 2, ..., n$, such that

$$\boldsymbol{R} = P_1 \times P_2 \times ... \times P_n = \prod_{d=1}^{n} P_d.$$

In order to find out whether two regions overlap, we need to perform a set intersection operation for both sets of points, which is given in **Theorem 1**.

Theorem 1. Let $\boldsymbol{R_U}$, $\boldsymbol{R_S}$ be two regions. $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ if and only if there exists a common point that lies within both regions, such that

$$\boldsymbol{R_U} \cap \boldsymbol{R_S} \neq \emptyset \Leftrightarrow \exists(\vec{p})(\vec{p} \in \boldsymbol{R_U} \wedge \vec{p} \in \boldsymbol{R_S}).$$

Proof. By set intersection, we derive

$$\boldsymbol{R_U} \cap \boldsymbol{R_S} = \{\vec{p} \mid \vec{p} \in \boldsymbol{R_U} \wedge \vec{p} \in \boldsymbol{R_S}\}.$$

Hence,

$$\boldsymbol{R_U} \cap \boldsymbol{R_S} \neq \emptyset \Leftrightarrow \{\vec{p} \mid \vec{p} \in \boldsymbol{R_U} \wedge \vec{p} \in \boldsymbol{R_S}\} \neq \emptyset.$$

Therefore, $\boldsymbol{R_U} \cap \boldsymbol{R_S} \neq \emptyset$ if and only if a common point $\vec{p}$ exists.

□

3.2 Dimension Reduction

Sorting is the key to all sort-based interest matching algorithms. However, it is not intuitively obvious how to sort regions in three-dimensional space. As described Section 2.8, the sort-based algorithms exploit a property called "dimension reduction", which reduce the multidimensional interest matching problem to a one-dimensional problem. The basic idea is based on the fact that

> *Two regions overlap in n-dimensional space if and only if their orthogonal projections on* 1^{st}*,* 2^{nd}*,..., and* n^{th} *dimension overlap.*

Figure 3.2 shows how the concept of dimension reduction works in two-dimensional space. In the figure, B-C overlap on x-axis; A-C, A-B, B-C, B-D, and C-D overlap on y-axis; hence, B-C overlap in two-dimensional space.

Formally, the one-dimensional overlap test is given in **Theorem 2**.

Theorem 2. Given two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, the orthogonal projections of the two regions are denoted by O_d and P_d, respectively, on d-axis. Then, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ if and only if

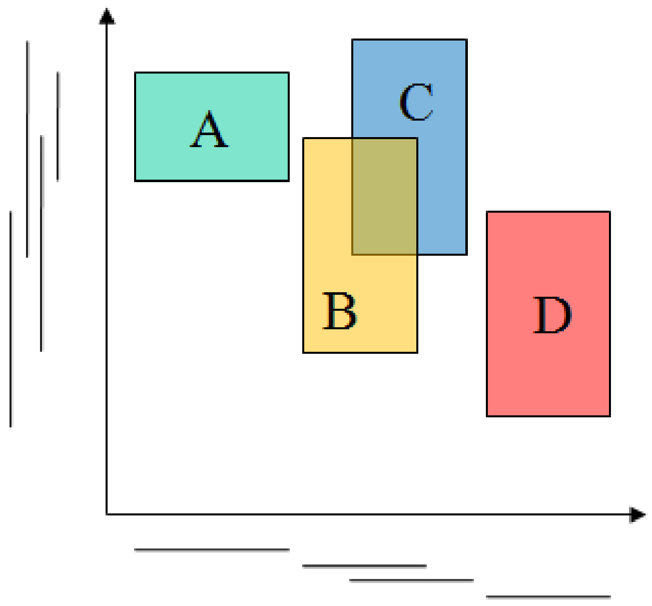


Figure 3.2: Dimension Reduction

$$\boldsymbol{R_U} \cap \boldsymbol{R_S} \neq \emptyset \Leftrightarrow O_d \cap P_d \neq \emptyset, \qquad for \quad d = 1, 2, ..., n.$$

Proof.

The Cartesian product satisfies the following properties, for all sets A, B, C, and D

$$A \times \emptyset = \emptyset \tag{3.1a}$$

$$(A \times B) \cap (C \times D) = (A \cap C) \times (B \cap D). \tag{3.1b}$$

Now, by **Definition 3**, we derive

$$\begin{aligned}
\boldsymbol{R_U} \cap \boldsymbol{R_S} &= \prod_{d=1}^{n} O_d \cap \prod_{d=1}^{n} P_d \\
&= (O_1 \times O_2 \times ... \times O_n) \cap (P_1 \times P_2 \times ... \times P_n) \\
&= (O_1 \cap P_1) \times ((O_2 \times O_3 \times ... \times O_n) \cap (P_2 \times P_3 \times ... \times P_n)) \quad by(3.1b) \\
&= (O_1 \cap P_1) \times (O_2 \cap P_2) \times ((O_3 \times O_4 \times ... \times O_n) \cap (P_3 \times P_4 \times ... \times P_n)) \\
&= ... \\
&= (O_1 \cap P_1) \times (O_2 \cap P_2) \times ... \times (O_n \cap P_n) \\
&= \prod_{d=1}^{n} (O_d \cap P_d).
\end{aligned}$$

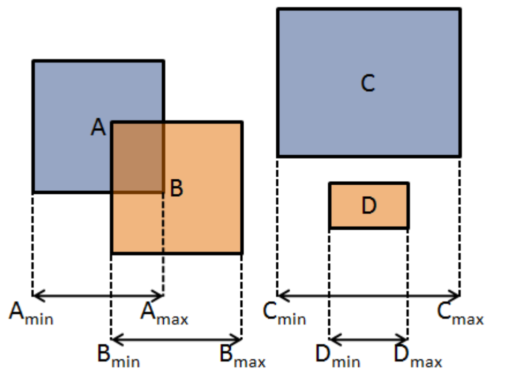
Hence,

$$\boldsymbol{R_U} \cap \boldsymbol{R_S} \neq \emptyset \Leftrightarrow \prod_{d=1}^{n} (O_d \cap P_d) \neq \emptyset.$$

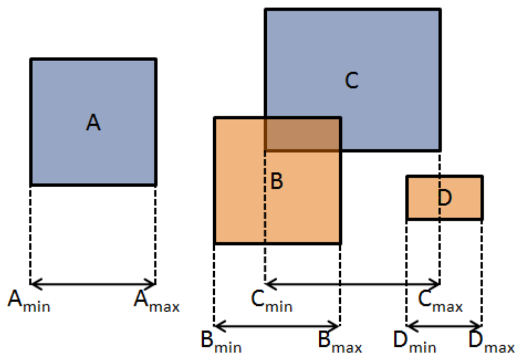
Therefore, by **Theorem 1** and (3.1a), $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ if and only if $O_d \cap P_d \neq \emptyset$ for $d = 1, 2, ..., n$. □

In the sort-based approaches, the endpoints of the orthogonal projections are maintained in d lists (i.e., three lists for three-dimensional space). In each time-step of simulation, the coordinates of each region as well as its projection endpoints are updated. These approaches carry out various sorting process for the lists, in order to find out the change of overlap status of the orthogonal projections. Figure 3.3 illustrates how changes in overlap status of the projections can be detected. In the figure, Region B moves and causes two swaps in the list of endpoints. The swap of A_{max} and B_{min} indicates that the projections A and B cease to overlap. The swap of C_{min} and B_{max} indicates that the projections B

and C begin to overlap. The move of projection D causes a swap of C_{max} and D_{max}. The overlap status of projections C and D does not change.



(a) Position of Projection Endpoints (t = 0)



(b) Position of Projection Endpoints (t = 1)

Figure 3.3: Changes of Overlap Status

3.3 Temporal Coherence

Similar to [93], the efficient sorting algorithm proposed in this thesis exploits temporal coherence, which is the property that the application state does not change significantly between consecutive time-steps of simulation. For a sparse and dynamic DVE, we anticipate that each entity (and the regions that are associated with it) would not make a relatively large movement between time-steps. Therefore, the sorted list of projection endpoints from a previous time-step is likely to be nearly sorted at the current time-step, in which case sorting will take only linear time using insertion-sort or bubble-sort. The property of temporal coherence is used originally in the field of computational geometry and was first introduced in Baraff's PhD thesis [96]. Baraff exploits this idea in an incremental algorithm for maintaining the set of overlapping Axis-Aligned Bounding Boxes (AABBs) during a simulation. This is perfectly suited for HLA-compliant systems since all regions in DDM are axis-aligned.

3.4 The Algorithm

At the initialisation stage of the proposed algorithm, we first construct a list of projection endpoints for each dimension. By sorting these lists, we can determine which projections overlap. Since the lists are presumed to be in random order, quick sort would be a good choice for this stage. The complexity of the sorting process is $O((m+n) \log (m+n))$ where m is the number of subscription regions and n is the number of update regions.

During runtime, the algorithm re-sorts the lists in order to find out the change of matching results. We can reduce the computational complexity of this process by caching the sorted lists and matching results from the previous time-step. This process exploits temporal coherence, which asserts that the coordinates of extent endpoints would change minimally between consecutive time-steps. Therefore, the lists are already nearly sorted before the re-sort operation is performed.

We use insertion-sort to re-sort the lists at runtime. During this process, the overlap statuses are only modified when the insertion-sort performs a *swap*. When this happens, the algorithm carries out pairwise overlap test for the two regions in question. Since temporal and geometric coherence is exploited, the chance of swap would be extremely small. The computational complexity of re-sorting the lists using insertion-sort is $O(n+m+s)$ for each dimension, where s is the number of swaps.

Details of sorting process are given in Algorithm 1. During the sorting pro-

Algorithm 1: An Efficient Algorithm for Interest Matching

```
Data: $EList$: a list of extent endpoints of the corresponding dimension
Data: $\boldsymbol{R}[i]$: a region containing the endpoint $EList[i]$, such that
        $Elist[i] \in \boldsymbol{R}[i]$
Data: $\boldsymbol{R}[j]$: a region containing the endpoint $EList[j]$, such that
        $Elist[j] \in \boldsymbol{R}[j]$
Data: $\boldsymbol{U}$: the set of all update regions
Data: $\boldsymbol{S}$: the set of all subscription regions
Result: $\boldsymbol{RS}$: a result set storing the pairs of regions that overlap each
          other
begin
    foreach dimension do
        for $i \leftarrow 1$ to $Size(EList) - 1$ do
            $temp \leftarrow EList[i]$;
            $j \leftarrow i - 1$;
            while $j \geq 0$ AND $EList[j] > temp$ do
                if ($\boldsymbol{R}[i] \in \boldsymbol{U}$ AND $\boldsymbol{R}[j] \in \boldsymbol{S}$) OR ($\boldsymbol{R}[i] \in \boldsymbol{S}$ AND $\boldsymbol{R}[j] \in \boldsymbol{U}$)
                then
                    if $\boldsymbol{R}[i] \cap \boldsymbol{R}[j] \neq \emptyset$ then
                        $\boldsymbol{RS} \leftarrow \boldsymbol{RS} \cup (\boldsymbol{R}[i], \boldsymbol{R}[j])$;
                    else
                        $\boldsymbol{RS} \leftarrow \boldsymbol{RS} - (\boldsymbol{R}[i], \boldsymbol{R}[j])$;
                    end
                end
                $EList[j+1] \leftarrow EList[j]$;
                $j \leftarrow j - 1$;
            end
            $EList[j+1] \leftarrow temp$;
        end
    end
end
```

cess, every iteration of the algorithm removes an endpoint from the $EList$ (line 4), inserting it into the correct position in the already-sorted list (line 17), until no endpoint remains. If insertion (swap) occurs, the algorithm then determines whether the regions that contain the two endpoints in question are one subscription region and one update region (line 7). If this is the case, and the two regions overlap (line 8), it stores this region pair into $\boldsymbol{RS}$ (line 9). Otherwise, it removes the region pair from $\boldsymbol{RS}$ (line 11). The whole process is repeated until all $EList$ are sorted.

3.4.1 Comparisons with Existing Sort-based Algorithms

If temporal coherence is exploited, the proposed algorithm would be theoretically faster than the sort-based algorithms presented in [92] and [94]. These approaches adopt heap-sort or quick sort for the matching process, which have a computational complexity of $O((m+n) \log (m+n))$. Furthermore, Pan et. al's approach [94] relies heavily on the frequency of regions updates, for which the authors assumed that only a small portion of the virtual entities are updated at each time-step of simulations. Our approach does not make any assumptions about the frequency of region updates. The property of temporal coherence allows all regions to be updated at each time-step. As long as the order of each endpoint list is not significantly changed, the sorting process can be very efficient.

Although both our algorithm and the insertion sort algorithm adopted by the Lucid Platform [93, 2] exploit temporal coherence to speed up the matching process, there are two important differences between the two algorithms. First, the Lucid Platform applies the approach of I-Collide [97], which maintains an overlap status for each pair of regions. The overlap status consists of a boolean flag for each dimension. Whenever all three of these flags are set, the corresponding region pair overlaps. These flags are only modified when insertion sort performs a swap. The decision of whether or not to toggle a flag is based on whether the coordinate values both refer to projection minima, both refer to projection maxima, or one refers to a projection minimum and the other a maximum. Although this approach is computationally efficient, it introduces significant storage overhead. For m subscription regions and n update regions, the number of potentially overlapped region pairs is $n \times m$. Therefore, the storage complexity of this approach is $O(dnm)$, where d is the number of dimensions. In contrast to the Lucid Platform's approach, our approach does not use a multidimensional table to store the pairwise overlap status. Instead, we perform an overlap test for the two regions in question whenever a swap occurs. This greatly reduces the storage overhead of the sorting algorithm of Lucid Platform. The second difference is that in Lucid Platform, a region can be used as an update region, a subscription region, or both. This might lead to wasteful overlap tests between two subscription regions or two update regions. Our approach follows the original HLA DDM framework, which carries out overlap tests only for a subscription and a update region. Therefore, the runtime efficiency can be further increased.

3.5 Performance Evaluation

This section describes the evaluation of the interest matching algorithm presented in this chapter. Based on the requirements of interest management described in Section 2.2.1, the evaluation focuses on three primary metrics:

- *Runtime Efficiency*: The computational performance of the interest matching process.
- *Filtering Precision*: The ability of filtering irrelevant messages.
- *Space Efficiency*: The computer memory required for the matching process.

Several sets of experiments were carried out to compare the performance of three approaches, namely:

1. Brute-force
2. Insertion sort algorithm of Lucid Platform [2]
3. Efficient sorting algorithm proposed in this chapter

The brute-force approach performs overlap tests for all update and subscription regions. Since no matching result is cached, its matching process is carried out at every time-step. The matching algorithm of Lucid Platform is described in Section 2.8. It is chosen as a comparison to the proposed algorithm because it is theoretically the fastest (linear time in the general case) among all existing interest matching algorithms.

3.5.1 Implementation and Experimental Set-ups

There is no generally accepted benchmark for the evaluation of interest matching algorithms. Of all the interest matching algorithms that have been reviewed in Section 2.8, a number of approaches such as Liu et al. [93], Lucid Platform [2], and Pan et al. [94] used random entity distribution and movement in the experiments.

Since Lucid Platform is the primary comparison target, its evaluation approach was adopted for our experimental set-ups.

The three algorithms were implemented in C++. All of the experiments were conducted on an Intel Core2 Duo E6850 3.0GHz with 4GB main memory and based on the following experimental set-up:

- *Entity Distribution*: The virtual entities were distributed randomly across the virtual space.
- *Entity Movement*: All entities move in a random direction.
- *Entity Speed*: Average speed of movement was equal to 50% of the region length per time-step.
- *Number of Dimensions*: All simulations were performed in three-dimensional space.
- *Number of Regions* An update region and a subscription region were associated with each moving entity.
- *Execution Time Measurement*: Average execution time of the matching algorithms was measured over 10,000 time-steps.

3.5.2 Runtime Efficiency

The first set of experiments compares the runtime efficiency of the three algorithms with the number of virtual entities extending from 100 to 1000.

Figure 3.4 shows the average execution time of the three approaches. It is not difficult to see that the brute-force approach, which is of quadratic complexity, had the poorest performance. The insertion sort algorithm of Lucid Platform and the proposed efficient sorting algorithm had similar performance and they both required much less computational effort than the brute-force approach. Such difference becomes significant when the number of entities is gradually increased.

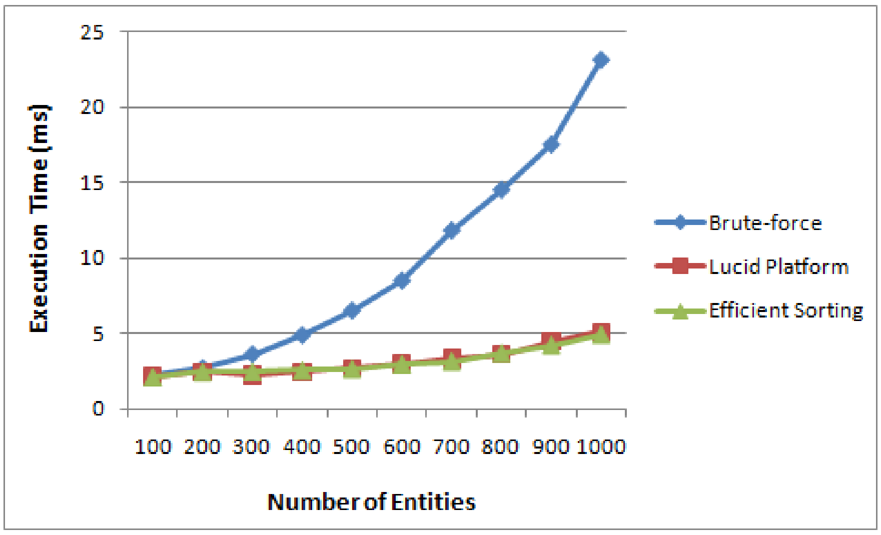


Figure 3.4: Runtime Efficiency of Interest Matching Approaches (Number of Entities varies)

3.5.3 Filtering Precision

The second set of experiments compares the filtering precision of the three algorithms with number of entities extending from 100 to 1000. In order to do so, an assumption is made that whenever the update region of an entity overlaps the subscription region of a participant, its state would be sent to him. Hence, the total number of entity states that the DVE system sends to the network after the filtering process is measured; this is regarded as the system throughput. For the sake of comparison, the performance of state broadcasting (i.e., no filtering) is also included as a reference.

The results of the experiments are shown in Figure 3.5. The graph clearly shows that the three algorithms had similar performance. They all filtered more than 90% of irrelevant entity states. In fact, since all three algorithms carry out aura-based filtering, under the same region size, their filtering accuracy should be similar.

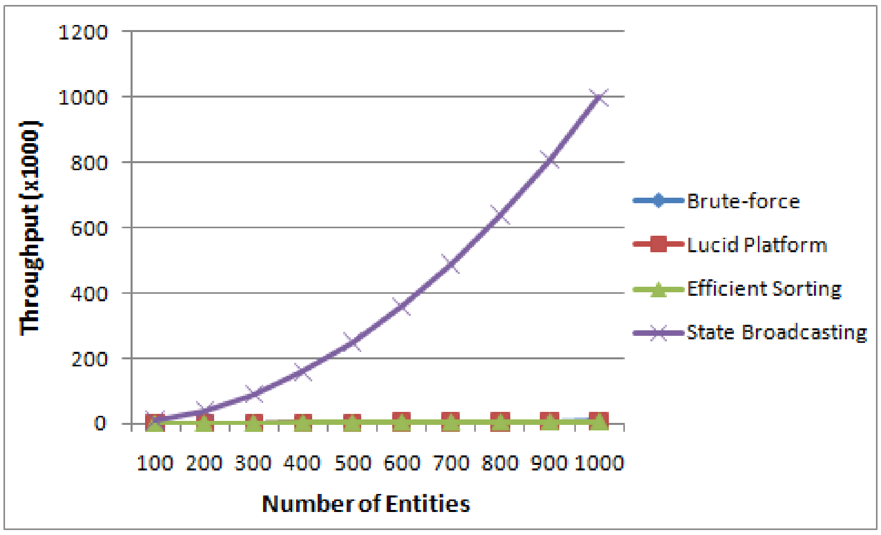


Figure 3.5: Filtering Precision of Interest Matching Approaches (Number of Entities varies)

3.5.4 Space Efficiency

The third set of experiments aims to compare the space efficiency of Lucid Platform and the proposed algorithm. The memory usage of the two approaches was measured with the number of virtual entities extending from 100 to 1000. The measurement was made by using Windows Task Manager. Since there was no memory allocation and deallocation after the initialisation stage of simulation, the memory usage of each time-step of simulation would remain unchanged.

Therefore, we can pause the process during runtime and measure its memory usage.

The results are shown in Figure 3.6. We can see from the graph that the memory consumption of Lucid Platform's matching algorithm is quadratic. This is due to the fact that it maintains a $n \times m$ table to store the pairwise overlap statuses. When the scale of the DVE becomes large, this approach would not be scalable in term of space efficiency.

On the other hand, the results suggest that the memory consumption of the proposed algorithm is linear. It scales much better than the Lucid Platform's approach when the number of entities is gradually increased.

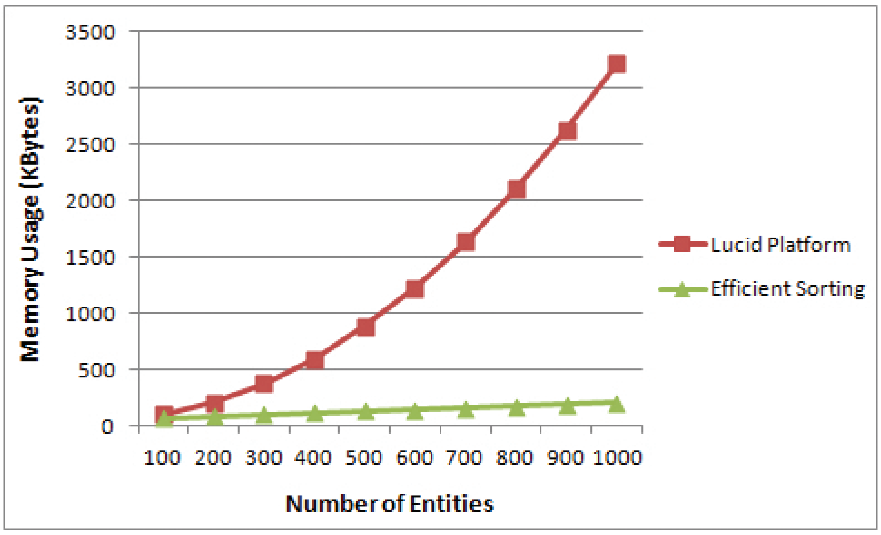


Figure 3.6: Memory Usage of Interest Matching Approaches (Number of Entities varies)

3.6 Summary

This chapter has presented and evaluated a new interest matching algorithm, which improves the design of the algorithm adopted by the Lucid Platform [2]. The proposed algorithm is developed based on two existing concepts: dimension reduction and temporal coherence. Dimension reduction is an approach which reduces the multidimensional overlap test to a one-dimensional problem. We have given formal definitions, theorems, and proofs of performing DDM-compatible overlap test based on this approach. Although the proposed algorithm has similar runtime efficiency as [2], experimental results presented in this chapter have demonstrated that its space efficiency is much better.

The proposed algorithm can still be considered as a “discrete algorithm”, which performs matching at discrete time-steps and ignores the missing events. In the next chapter, we will demonstrate how this algorithm can be extended to support space-time interest matching, and thus significantly reducing its computational overhead.

Chapter 4

Space-Time Interest Matching Algorithms

This chapter introduces a new approach to solve the missing event problem discussed in Section 2.8.1. This approach is called space-time interest matching, which involves using "time" as an additional dimension for the matching process. In contrast to the existing approaches, it does not increase region size and thus can preserve the filtering precision of interest management. Moreover, it does not increase the frequency of interest matching for all auras. Therefore, the runtime overhead can be reduced.

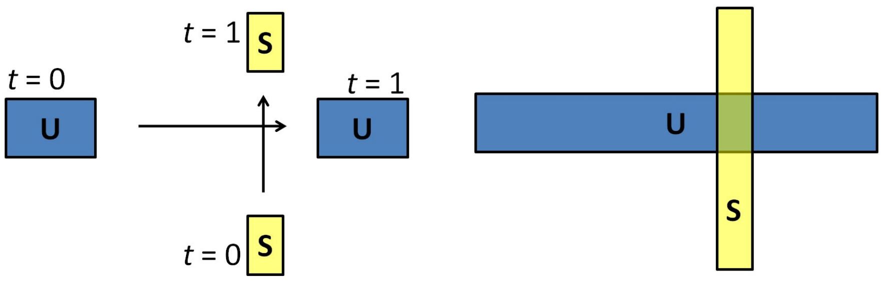
In the next section we present an algorithm for pairwise space-time interest matching. This algorithm forms part of the final multi-region space-time matching scheme which is described in Section 4.2.

4.1 Pairwise Space-Time Interest Matching

This section presents a pairwise space-time interest matching approach, which is designed for aura-based filtering and aims to find out the interaction of a pair of regions between two consecutive time-steps of simulation. The proposed approach uses swept volumes [98] to enclose the trajectory of the regions over each time interval. If the swept volumes overlap, it then carries out a divide-and-conquer algorithm to efficiently find out whether the two auras in question actually overlap at a certain time.

4.1.1 Swept Volume

A swept volume is a bounding volume that bounds the motion of an arbitrary virtual entity along an arbitrary path over a time interval. The use of swept volume has been studied extensively in computer graphics, robotics, and in particular in proximity queries [99]. Some formal definitions of swept volume can be found in an early paper [100].



(a) Positions of U and S (b) Swept Volume of U and S

Figure 4.1: Swept Volume of a Pair of Regions

Figure 4.1(a) illustrates the two-dimensional positions of two regions, U and S, at time $t = 0$ and $t = 1$. Figure 4.1(b) illustrates the two-dimensional swept volume generated by the motion of U and S over the time interval [0,1]. We can see that U and S neither overlap at $t = 0$ nor at $t = 1$. However, their swept volumes overlap each other. This indicates that they potentially overlap within the time interval [0,1].

Exact computation of swept volumes could be time consuming, especially when entities undergo rotations and are geometrically complex. However, this can be greatly simplified if we apply swept volumes on DDM-based interest matching, since all regions in DDM are axis-aligned and do not rotate. In the rest of this subsection, we extend and modify the formal definitions of [100], in order to apply them on axis-aligned regions.

We assume the existence of a polynomial function $\vec{\boldsymbol{\tau}}(t)$, which describes the motion of $\boldsymbol{R}$ such that it transforms a point $\vec{q} \in \boldsymbol{R}$ to a new position $\vec{p}$ at time t. For example, in three-dimensional Cartesian space, $\vec{\boldsymbol{\tau}}(t)$ can be expressed as

$$\vec{\boldsymbol{\tau}}(t) = x(t)\hat{\boldsymbol{\imath}} + y(t)\hat{\boldsymbol{\jmath}} + z(t)\hat{\boldsymbol{k}} \tag{4.1}$$

where $\hat{\boldsymbol{\imath}}$, $\hat{\boldsymbol{\jmath}}$ and $\hat{\boldsymbol{k}}$ are the unit vectors; and the x, y, and z coordinates of $\boldsymbol{R}$ vary with time t according to the polynomial functions $x(t)$, $y(t)$ and $z(t)$, respectively.

Definition 4. Given a polynomial function $\vec{\boldsymbol{\tau}}(t)$, which describes the motion of $\boldsymbol{R}$ such that it transforms a point $\vec{q} \in \boldsymbol{R}$ to the position $\vec{p}$ at time t. $\boldsymbol{R}(t)$ is defined as a set of points occupied by $\boldsymbol{R}$ at any time t, such that

$$\boldsymbol{R}(t) = \{\vec{p} \mid (\exists \vec{q})(\vec{q} \in \boldsymbol{R}, \vec{p} = \vec{q} + \vec{\boldsymbol{\tau}}(t))\}.$$

Definition 5. Let $\boldsymbol{SV}([a, b])$ denote the swept volume generated by the motion of a region $\boldsymbol{R}$ over the time interval $[a,b]$. $\boldsymbol{SV}([a,b])$ can be regarded as the infinite union of all $\boldsymbol{R}(t)$ within $[a,b]$, such that

$$\boldsymbol{SV}([a,b]) = \bigcup_{t\in[a,b]} \boldsymbol{R}(t) = \{\vec{p} \mid \vec{p} \in \boldsymbol{R}(t), \exists t \in [a,b]\}.$$

Definition 6. Given a time interval $[a,b]$. It can be expressed as the union of subintervals, such that

$$[a,b] = \bigcup_{i=1}^{n} I_i$$

where $I_i = [t_{i-1}, t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

Lemma 1. Let $\boldsymbol{SV}([a,b])$ denote the swept volume generated by the motion of a region $\boldsymbol{R}$ over a time interval $[a,b]$. By **Definition** 6, $\boldsymbol{SV}([a,b])$ can be expressed as the union of smaller swept volumes, such that

$$\boldsymbol{SV}([a,b]) = \bigcup_{i=1}^{n} \boldsymbol{SV}(I_i).$$

Proof. By **Definition 5**, we derive

$$
\begin{aligned}
\boldsymbol{SV}([a,b]) &= \bigcup_{t\in[a,b]} \boldsymbol{R}(t) \\
&= (\bigcup_{t\in[a,t_1]} \boldsymbol{R}(t)) \cup (\bigcup_{t\in[t_1,t_2]} \boldsymbol{R}(t)) \cup \ldots \\
&\quad \cup (\bigcup_{t\in[t_{n-1},b]} \boldsymbol{R}(t)) \\
&= \boldsymbol{SV}([t_0,t_1]) \cup \boldsymbol{SV}([t_1,t_2]) \cup \ldots \\
&\quad \cup \boldsymbol{SV}([t_{n-1},t_n]) \\
&= \bigcup_{i=1}^{n} \boldsymbol{SV}(I_i).
\end{aligned}
$$

□

Details of swept volume computation can be found in [98]. The basic idea, as described in **Definition 4** and **Definition 5**, is to interpolate a set of points between consecutive time-steps by a polynomial function $\vec{\boldsymbol{\tau}}(t)$. If the degree of the polynomial, *deg* $\vec{\boldsymbol{\tau}}(t)$, is 1, then the region undergoes linear translational motion and thus its swept volume can be computed easily. However, if *deg* $\vec{\boldsymbol{\tau}}(t) > 1$, solving higher order polynomials can be expensive and results in numerical inaccuracies. The following equations describe two common approaches of interpolation:

$$
\vec{\boldsymbol{\tau}}(t) = \begin{cases} \vec{v}t & \text{with constant velocity } \vec{v} \\ \vec{v}(a)t + \frac{1}{2}\vec{a}t^2 & \text{with constant acceleration } \vec{a} \end{cases}
$$

where $\vec{v}(a)$ is the initial velocity of the set of points at $t = a$.

To determine whether two swept volumes overlap, we have to perform a set intersection operation for the two sets of points that are occupied by them. This overlap test is given in **Theorem** 3.

Theorem 3. Let $\boldsymbol{SV_U}$, $\boldsymbol{SV_S}$ denote the swept volumes generated by two moving regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, over a certain time interval. Then, $\boldsymbol{SV_U}$ overlaps $\boldsymbol{SV_S}$ if and only if there exists a common point that lies within both of them, such that

$$\boldsymbol{SV_U} \cap \boldsymbol{SV_S} \neq \emptyset \Leftrightarrow \exists(\vec{p})(\vec{p} \in \boldsymbol{SV_U} \wedge \vec{p} \in \boldsymbol{SV_S}).$$

Proof. Similar to **Theorem** 1, by set intersection we derive

$$\boldsymbol{SV_U} \cap \boldsymbol{SV_S} = \{\vec{p} \mid \vec{p} \in \boldsymbol{SV_U} \wedge \vec{p} \in \boldsymbol{SV_S}\}.$$

.

Hence,

$$\boldsymbol{SV_U} \cap \boldsymbol{SV_S} \neq \emptyset \Leftrightarrow \{\vec{p} \mid \vec{p} \in \boldsymbol{SV_U} \wedge \vec{p} \in \boldsymbol{SV_S}\} \neq \emptyset.$$

Therefore, $\boldsymbol{SV_U} \cap \boldsymbol{SV_S} \neq \emptyset$ if and only if a common point $\vec{p}$ exists.

□

The result $\boldsymbol{SV_U} \cap \boldsymbol{SV_S} \neq \emptyset$ also indicates that $\boldsymbol{R_U}$ potentially overlaps $\boldsymbol{R_S}$ in $[a, b]$. However, it is not sufficient to say that they actually overlap each other at a certain time $t \in [a, b]$.

In order to find out whether $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$ overlap in $[a, b]$, we need to carry out a space-time overlap test for the sets of points which each region occupies along its path of motion. Specifically, this test involves performing set intersection operations for all $\boldsymbol{R_U}(t)$ and $\boldsymbol{R_S}(t)$ along the regions' path of motion in $[a,b]$, which is given in **Theorem 4**.

Theorem 4. Let $\boldsymbol{R_U}(t)$, $\boldsymbol{R_S}(t)$ denote the two sets of points occupied by two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, at some time $t \in [a,b]$. Then, during the motion, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ in $[a,b]$ if and only if there exists a common point that lies within both $\boldsymbol{R_U}(t)$ and $\boldsymbol{R_S}(t)$ at some time $t \in [a,b]$, such that

$$\begin{aligned}&\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \neq \emptyset \\ \Leftrightarrow &\exists(\vec{p},t)(\vec{p} \in \boldsymbol{R_U}(t) \wedge \vec{p} \in \boldsymbol{R_S}(t), t \in [a,b]).\end{aligned}$$

Proof. From the L.H.S. of the equivalence, we derive

$$\begin{aligned}&\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \neq \emptyset \\ \Leftrightarrow &\exists t\, (t \in [a,b] \wedge \boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t) \neq \emptyset),\end{aligned}$$

and the assertion

$$\begin{aligned}&\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t) \neq \emptyset \\ \Leftrightarrow &\exists \vec{p}\, (\vec{p} \in \boldsymbol{R_U}(t) \wedge p \in \boldsymbol{R_S}(t)).\end{aligned}$$

Then, for any propositions P and Q, the assertion

$$\exists t\,(P(t) \wedge (\exists \vec{p}\, Q(\vec{p}, t))$$

$$\Leftrightarrow \exists (t, p)\,(P(t) \wedge Q(p, t)).$$

Substituting $t \in [a, b]$ as $P(t)$ and $\vec{p} \in \boldsymbol{R_U}(t) \wedge p \in \boldsymbol{R_S}(t)$ as $Q(p, t)$, hence, the L.H.S. is equivalent to

$$\exists (\vec{p}, t)(\vec{p} \in \boldsymbol{R_U}(t) \wedge \vec{p} \in \boldsymbol{R_S}(t), t \in [a, b]).$$

□

Since the infinite union of **Theorem 4** cannot be easily computed numerically, an approximate solution is used to convert the infinite union to a finite union. Specifically, according to **Definition 6** and **Lemma 1**, $\boldsymbol{SV}([a, b])$ can be split into some smaller swept volumes $\boldsymbol{SV}(I_i)$ by splitting $[a, b]$ into some subintervals I_i. Hence $\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)$ can be approximated by $\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)$ if I_i are small. This approximate solution is given in **Lemma 2**.

Lemma 2. *Let $\boldsymbol{R_U}(t)$, $\boldsymbol{R_S}(t)$ denote the two sets of points occupied by two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, at some time $t \in [a, b]$;and let $\boldsymbol{SV_U}$, $\boldsymbol{SV_S}$ denote the swept volumes generated by the motion of $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, over the time interval $[a, b]$. Then,*

$$\bigcup_{t \in [a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \subseteq \bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i))$$

where I_i represents the subinterval of $[a, b]$, such that $I_i = [t_{i-1}, t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

Proof. Let $\vec{p}$ be an element of the L.H.S, such that

$$\begin{aligned}
&\vec{p} \in \bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \\
\Leftrightarrow &\exists t\, (t \in [a,b] \wedge \vec{p} \in \boldsymbol{R_U}(t) \wedge \vec{p} \in \boldsymbol{R_S}(t)) \\
\Leftrightarrow &\exists (i,t)\, (t \in I_i \wedge \vec{p} \in \boldsymbol{R_U}(t) \wedge \vec{p} \in \boldsymbol{R_S}(t)).
\end{aligned}$$

.

Likewise,

$$\begin{aligned}
&\vec{p} \in \bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)) \\
\Leftrightarrow &\exists i\, (\vec{p} \in \boldsymbol{SV_U}(I_i) \wedge \vec{p} \in \boldsymbol{SV_S}(I_i)).
\end{aligned}$$

Since, by **Definition 5**, $\vec{p} \in \boldsymbol{SV_U}(I_i)$ is equivalent to $\exists u(u \in I_i \wedge \vec{p} \in \boldsymbol{R_U}(u))$ and $\vec{p} \in \boldsymbol{SV_S}(I_i)$ is equivalent to $\exists s(s \in I_i \wedge \vec{p} \in \boldsymbol{R_S}(s))$, therefore the R.H.S. is equivalent to

$$\exists (i,u,s)(u \in I_i \wedge s \in I_i \wedge \vec{p} \in \boldsymbol{R_U}(u) \wedge p \in \boldsymbol{R_S}(s)).$$

Take $u = s = t$. Hence,

$$\forall \vec{p}(\vec{p} \in \bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \implies$$
$$\vec{p} \in \bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)))$$

which means

$$\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \subseteq \bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)).$$

□

By using the approximate solution given in **Lemma 2**, the overlap test of **Theorem 4** can be converted into **Theorem 5**.

Theorem 5. *Let $\boldsymbol{SV_U}$, $\boldsymbol{SV_S}$ denote the swept volumes generated by the motion of two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, over the time interval $[a, b]$. Then, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ in $[a, b]$ only if*

$$\bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)) \neq \emptyset$$

where I_i represents the subinterval of $[a, b]$, such that $I_i = [t_{i-1}, t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

Proof.

From **Lemma 2**,

$$\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \subseteq \bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i))$$

which means

$$\begin{aligned} &\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \neq \emptyset \\ \Longrightarrow &\bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)) \neq \emptyset. \end{aligned}$$

Applying **Theorem 4**, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ if and only if

$$\bigcup_{t\in[a,b]} (\boldsymbol{R_U}(t) \cap \boldsymbol{R_S}(t)) \neq \emptyset$$

i.e., it can be determined that $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ only if

$$\bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)) \neq \emptyset.$$

□

4.1.2 Divide-and-Conquer Algorithm

It is not difficult to see that the accuracy of the approximate overlap test described in **Theorem 5** is dependent on the size of subintervals I_i. For $I_i = [a, b]$, the overlap test is essentially equivalent to testing whether $\boldsymbol{SV_U}([a, b]) \cap \boldsymbol{SV_S}([a, b]) \neq \emptyset$. As mentioned previously, this would be insufficient to determine whether $\boldsymbol{R_U}$ actually overlaps $\boldsymbol{R_S}$ in $[a, b]$. Decreasing the size of I_i would increase the accuracy of the approximate overlap test. However, this also results in increasing the number of I_i, and thus much overhead in computing swept volumes and performing set intersection operations. Therefore, instead of sequentially computing all

$\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)$ over the entire time interval $[a, b]$, we use a divide-and-conquer algorithm to avoid unnecessary set intersections.

Algorithm 2: Pairwise Space-Time Interest Matching

```
Data: [a, b]: a time interval
Data: $\boldsymbol{SV_U}([a,b])$, $\boldsymbol{SV_S}([a,b])$: two swept volumes generated by the motion
      of two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, respectively, over $[a,b]$
Data: Stack: a stack storing a pair of swept volumes and a time interval
Data: $\boldsymbol{SV_1}, \boldsymbol{SV_2}$: two temporary variables to hold the instance of the swept
      volumes
Data: $I$: a temporary variable to hold the instance of the time intervals
Result: isOverlapped: a boolean variable indicating whether $\boldsymbol{R_U}$ overlaps
        $\boldsymbol{R_S}$ at a certain time
begin
    $isOverlapped \leftarrow false$;
    Stack.push ($\boldsymbol{SV_U}([a,b])$,$\boldsymbol{SV_S}([a,b])$,$[a,b]$);
    while Stack is not empty do
        $\boldsymbol{SV}_1 \leftarrow Stack.top.\boldsymbol{SV}_1$;
        $\boldsymbol{SV}_2 \leftarrow Stack.top.\boldsymbol{SV}_2$;
        $I \leftarrow Stack.top.I$;
        Stack.pop();
        if $\boldsymbol{SV}_1 \cap \boldsymbol{SV}_2 \neq \emptyset$ then
            if $I$ is smaller than a threshold then
                $isOverlapped \leftarrow true$;
                break;
            else
                Split $I$ into two equal subintervals $I_1$ and $I_2$;
                Stack.push ($\boldsymbol{SV}_1(I_1)$,$\boldsymbol{SV}_2(I_1)$,$I_1$);
                Stack.push ($\boldsymbol{SV}_1(I_2)$,$\boldsymbol{SV}_2(I_2)$,$I_2$);
            end
        end
    end
end
```

The divide-and-conquer algorithm employed by the proposed pairwise space-time interest matching approach is given in Algorithm 2. The algorithm first tests whether $\boldsymbol{SV_U}([a,b])$ overlaps $\boldsymbol{SV_S}([a,b])$ (line 9). If this is the case, it then splits the time interval $[a, b]$ into two equal subintervals (line 14) and splits each of the swept volumes into two smaller ones accordingly (lines 15, 16). This process continues recursively until no overlap occurs or the tested subinterval is smaller than a user defined threshold (line 10).

Essentially, the choice of threshold value controls the size of the smallest subintervals. As we have discussed previously, this is in fact a trade-off and is application dependent. The performance of the divide-and-conquer algorithm for different threshold values is evaluated in Section 4.3.

4.2 Multi-Region Space-Time Interest Matching

This section presents the design principles of a space-time interest matching algorithm for multiple regions. This algorithm aims to support matching between n update regions and m subscription regions, and capture their interactions between discrete time-steps. It first uses a sorting algorithm based on dimension reduction, which reduces the multidimensional problem to a one-dimensional problem and efficiently finds out the pairs of swept volumes that are potentially overlapped. It then carries out the pairwise matching algorithm presented in Section 4.1, in order to determine whether the regions in question actually overlap each other at a certain time.

4.2.1 Orthogonal Projection of Swept Volume

In order to make use of the concept of dimension reduction, the orthogonal projections for each set of points on all coordinate axes need to be calculated. This can be easily done with the regions, since all regions are axis-aligned. Computing the orthogonal projections for swept volumes, however, would be more complex. For example, according to (1), $x(t)$ is the polynomial function that describes the motion of a region's x coordinate. To find the orthogonal projection of a swept volume on x-axis within the time interval $[a, b]$, we must find the greatest and least values of $x(t)$ on $[a, b]$. This requires finding all the relative extrema there and then compare all these values together with $x(a)$ and $x(b)$ (since greatest and least values may sometimes occur at the endpoint of the time interval). Finally we choose the greatest and least values among them. A formal definition of the orthogonal projection of a swept volume is given in **Definition 7**.

Definition 7. Given a polynomial function $\vec{\boldsymbol{\tau}}(t)$, which describes the motion of a region $\boldsymbol{R}$ at any time $t \in [a, b]$. Let MIN_d, MAX_d be the two endpoints of the orthogonal projection of $\boldsymbol{R}$ on d axis at $t = a$, where $MIN_d \leq MAX_d$, and $x_d(t)$ be the d-component of $\vec{\boldsymbol{\tau}}(t)$. Then, the orthogonal projection of the swept volume generated by the motion of $\boldsymbol{R}$ in $[a, b]$ on d axis can be defined as

$$\begin{aligned} SP_d = [&min\{MIN_d, \{MIN_d + x_d(t) \mid t \in [a, b]\}\}, \\ &max\{MAX_d, \{MAX_d + x_d(t) \mid t \in [a, b]\}\}]. \end{aligned}$$

Figure 4.2 illustrates the orthogonal projections of a swept volume in two-dimensional space.

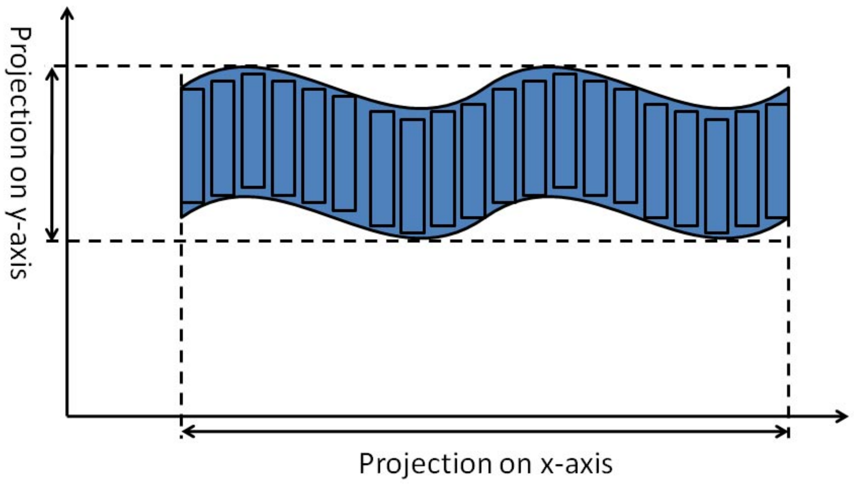


Figure 4.2: Orthogonal Projections of Swept Volume

As discussed previously in Section 4.1, if $deg\ \vec{\tau}(t) = 1$, the region undergoes linear translational motion. The computation of SP_d can, therefore, be greatly simplified, which is given in **Corollary 1**.

Corollary 1. Given a polynomial function $\vec{\tau}(t)$, which describes the motion of a region $\boldsymbol{R}$ at any time $t \in [a, b]$. Let MIN_d, MAX_d be the two endpoints of the orthogonal projection of $\boldsymbol{R}$ on d axis at $t = a$, where $MIN_d \leq MAX_d$, and $x_d(t)$ be the d-component of $\vec{\tau}(t)$. If $deg\ \vec{\tau}(t) = 1$, the orthogonal projection of the swept volume generated by the motion of $\boldsymbol{R}$ in $[a, b]$ on d axis can be simplified as

$$\begin{aligned} SP_d = [&min\{MIN_d, MIN_d + c\}, \\ &max\{MAX_d, MAX_d + c\}] \end{aligned}$$

where $c = x_d(t)$, $\forall c \in \mathbb{R}$ and $\forall t \in [a, b]$.

Proof. If $deg\ \vec{\tau}(t) = 1$, $x_d(t)$ becomes constant $\forall t \in [a, b]$, such that $c = x_d(t)$, $\forall c \in \mathbb{R}$. Then, $min\{MIN_d + x_d(t) \mid t \in [a, b]\}$ is equivalent to $\{MIN_d + c\}$, and $max\{MAX_d + x_d(t) \mid t \in [a, b]\}$ is equivalent to $\{MAX_d + c\}$. Hence, SP_d can be simplified as

$$\begin{aligned} SP_d = [&min\{MIN_d, MIN_d + c\}, \\ &max\{MAX_d, MAX_d + c\}]. \end{aligned}$$

$\square$

For $deg\ \vec{\tau}(t) > 1$, solving higher order polynomials would be more complex. Finding all the relative extrema of $x(t)$ on $[a, b]$ involves computing its first derivative such that, solving $x'(t) = 0$ for all roots within [a,b]. Hence, we can determine the relative extrema by substituting these roots into $x(t)$. Alternatively, [101] describes a numerical solution without computing the first derivative of $x(t)$. For the details of this solution, the reader can be referred to Appendix A.

4.2.2 Axis-Aligned Swept Volume

The orthogonal projections of the swept volume actually form an axis-aligned swept volume (AASV) of a region's trajectory. Although it is a loose bound, using AASV approximation has two benefits. First, computing AASV is much faster than computing the actual swept volume. Second, the AASV can be easily integrated with the dimension reduction approach. **Definition 8** gives a formal definition of AASV.

Definition 8. Let SP_d denote the orthogonal projections of a swept volume on d-axis. Then, the AASV $\boldsymbol{AV}$ of the swept volume can be defined as the Cartesian product of all SP_d, for $d = 1, 2, ..., n$, such that

$$\boldsymbol{AV} = SP_1 \times SP_2 \times ... \times SP_n = \prod_{i=1}^{n} SP_i.$$

Lemma 3. Let $\boldsymbol{SV}([a,b])$ denote the swept volume generated by the motion of a region over the time interval $[a,b]$, and **AV**([a,b]) denote the AASV formed by the orthogonal projections of $\boldsymbol{SV}([a,b])$, then

$$\boldsymbol{SV}([a,b]) \subseteq \boldsymbol{AV}([a,b]).$$

Proof. Assume by contradiction that $(\exists \vec{p})(\vec{p} \in \boldsymbol{SV}([a,b]) \wedge \vec{p} \notin \boldsymbol{AV}([a,b]))$. By **Definition 8**, $\vec{p} \notin \boldsymbol{AV}([a,b])$ implies that there exists a component x_d of $\vec{p}$ that lies outside the interval SP_d, which is the d-component of $\boldsymbol{AV}([a,b])$. This contradicts $\vec{p} \in \boldsymbol{SV}([a,b])$ since by **Definition 7**, SP_d is the orthogonal projection of $\boldsymbol{SV}([a,b])$. Therefore, $\vec{p}$ does not exist and $\boldsymbol{SV}([a,b]) \subseteq \boldsymbol{AV}([a,b])$. □

We can now rewrite the approximate solution of **Lemma 2** to as **Lemma 4**.

Lemma 4. Let $\boldsymbol{SV_U}$, $\boldsymbol{SV_S}$ denote the swept volumes generated by the motion of two different regions over a certain time interval, and $\boldsymbol{AV_U}$, $\boldsymbol{AV_S}$ denote the AASVs formed by the orthogonal projections of $\boldsymbol{SV_U}$ and $\boldsymbol{SV_S}$, respectively. Then,

$$\bigcup_{i=1}^{n} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \subseteq \bigcup_{i=1}^{n} (\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i))$$

where I_i represents the subinterval of $[a,b]$, such that $I_i = [t_{i-1}, t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

Proof.

It follows from **Lemma 3** that $\boldsymbol{SV_U}(I_i) \subseteq \boldsymbol{AV_U}(I_i)$ and $\boldsymbol{SV_S}(I_i) \subseteq \boldsymbol{AV_S}(I_i)$ for any time interval I_i. Then

$$\begin{aligned}
\forall p \in \boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) &\implies p \in \boldsymbol{SV_U}(I_i) \wedge p \in \boldsymbol{SV_S}(I_i) \\
&\implies p \in \boldsymbol{AV_U}(I_i) \wedge p \in \boldsymbol{AV_S}(I_i) \\
&\implies p \in \boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i).
\end{aligned}$$

Therefore $\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \subseteq \boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)$ holds for any time interval I_i.

Let $P(n)$ be the proposition that

$$\bigcup_{i=1}^{n}(\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \subseteq \bigcup_{i=1}^{n}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)).$$

It is shown that $P(1)$ is true. Assume $P(k)$ is true, $\exists k \in \mathbb{Z}^+$, such that

$$\bigcup_{i=1}^{k}(\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \subseteq \bigcup_{i=1}^{k}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)).$$

When $n = k + 1$, let us shorten the conditions into

$$\begin{aligned}
A &= \bigcup_{i=1}^{k} (\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i)) \\
B &= \boldsymbol{SV_U}(I_{k+1}) \cap \boldsymbol{SV_S}(I_{k+1}) \\
C &= \bigcup_{i=1}^{k} (\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)) \\
D &= \boldsymbol{AV_U}(I_{k+1}) \cap \boldsymbol{AV_S}(I_{k+1}).
\end{aligned}$$

i.e., $P(k + 1)$ is equivalent to

$$A \cup B \subseteq C \cup D.$$

Since by $P(k)$, $A \subseteq C$ and $B \subseteq D$, we derive

$$\begin{aligned}
\forall p \in A \cup B &\implies p \in A \vee p \in B \\
&\implies p \in C \vee p \in D \\
&\implies p \in C \cup D.
\end{aligned}$$

Therefore, $A \cup B \subseteq C \cup D$ holds and thus $P(k + 1)$ is also true. Hence by induction $P(n)$ is true, $\forall n \in \mathbb{Z}^+$.

□

Hence, by approximation, the overlap test of **Theorem 5** can be converted into **Theorem 6**.

Theorem 6. Given two regions $\boldsymbol{R_U}$ and $\boldsymbol{R_S}$, the AASVs generated by the motion of the two regions over the time interval $[a, b]$ are denoted by $\boldsymbol{AV_U}$ and $\boldsymbol{AV_S}$, respectively. Then, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ in [a,b] only if

$$\bigcup_{i=1}^{n}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)) \neq \emptyset$$

where I_i represents the subinterval of $[a, b]$, such that $I_i = [t_{i-1}, t_i]$ for $a = t_0 < t_i < ... < t_n = b$.

Proof.

From **Lemma 4**,

$$\bigcup_{i=1}^{n}(\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \subseteq \bigcup_{i=1}^{n}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i))$$

which means

$$\begin{aligned} &\bigcup_{i=1}^{n}(\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \neq \emptyset \\ \Longrightarrow &\bigcup_{i=1}^{n}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)) \neq \emptyset. \end{aligned}$$

Applying **Theorem 5**, $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ only if

$$\bigcup_{i=1}^{n}(\boldsymbol{SV_U}(I_i) \cap \boldsymbol{SV_S}(I_i) \neq \emptyset$$

i.e., it can be determined that $\boldsymbol{R_U}$ overlaps $\boldsymbol{R_S}$ only if

$$\bigcup_{i=1}^{n}(\boldsymbol{AV_U}(I_i) \cap \boldsymbol{AV_S}(I_i)) \neq \emptyset.$$

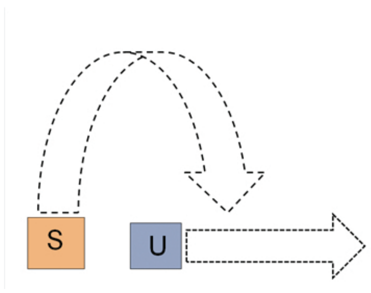
□

Similar to **Theorem 5**, the accuracy of the approximate overlap test described in **Theorem 6** is dependent on the size of subintervals I_i. Choosing a large size for I_i would be insufficient to determine whether $\boldsymbol{AV_U}$ actually overlaps $\boldsymbol{AV_S}$ in $[a, b]$. Decreasing the size of I_i would increase the accuracy of the test; however, it also results in increasing the number of I_i, and thus much overhead in computing swept volumes and performing set intersection operations.

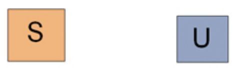
Figure 4.3 illustrates how the divide-and-conquer algorithm can be used with AASVs. Consider two regions, U and S, which move along the direction of the arrows at $t = a$ (Figure 4.3(a)). They arrive to a new position at $t = b$ (Figure 4.3(b)). Although the two regions neither overlap at $t = a$ nor at $t = b$, their swept volumes, SV_U and SV_S, overlap in the time interval $[a, b]$ (Figure 4.3(c)), which indicates that the two regions potentially overlap at a certain time in $[a, b]$. To carry out the divide-and-conquer overlap test, the algorithm first computes two AASVs, $AASV_U$ and $AASV_S$, which are formed by the orthogonal projections of SV_U and SV_S, respectively (Figure 4.3(d)). Since $AASV_U$ overlaps $AASV_S$ in $[a, b]$, the algorithm splits the time interval into two equal subintervals. It then checks recursively whether or not the two AASVs overlap in $[a, (b-a)/2]$ (Figure 4.3(e)) and $[(b-a)/2, b]$ (Figure 4.3(f)).

4.2.3 Sorting Algorithm

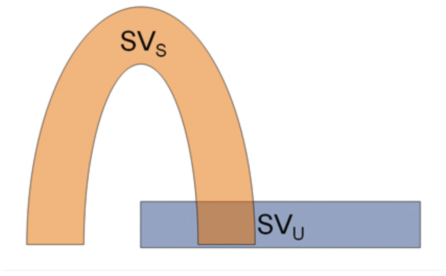
Based on the approximate solution for AASV overlap test, a robust culling method can be developed which is extended from the efficient sorting algorithm presented in the previous chapter (Algorithm 1). This new algorithm first culls



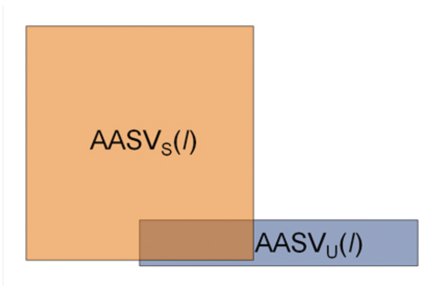
(a) t = a



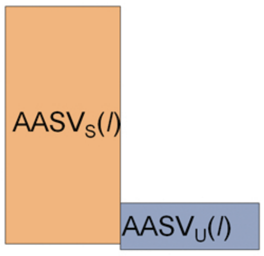
(b) t = b



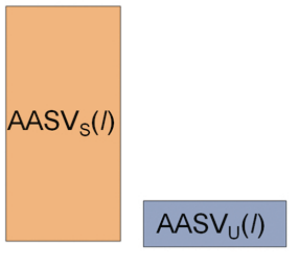
(c) Swept Volumes, I = [a,b]



(d) AASVs, I = [a,b]



(e) AASVs, I = [a,(b-a)/2]



(f) AASVs, I = [(b-a)/2,b]

Figure 4.3: Divide and Conquer with AASVs

out the AASV pairs that are unlikely to overlap. The remaining pairs are then processed by the divide-and-conquer algorithm given in Algorithm 2.

At the initialisation stage, the algorithm constructs a list of AASV's projection end-points for each dimension. By sorting these lists, it can determine which projections overlap. Since the lists are presumed to be in random order, quick-sort would be a good choice for this stage. The complexity of the sorting process is $O((m+n)\log(m+n))$ where m is the number of subscription regions and n is the number of update regions.

During runtime, the algorithm re-sorts the lists in order to find out the change in matching results. We can reduce the computational complexity of this process by exploiting temporal coherence, which caches the sorted lists and matching results from the previous time-step. Hence, the coordinates of the AASV end-points would change minimally between consecutive time-steps, and the lists would be already nearly sorted before the re-sort operation is performed.

Insertion-sort is used to re-sort the lists at runtime. During this process, the overlap statuses are only modified when the insertion-sort performs a *swap*. When this happens, the algorithm carries out pairwise space-time overlap test for the two AASVs in question. Since temporal coherence is exploited, the chance of swap would be extremely small. The computational complexity of re-sorting the lists using insertion-sort is $O(n+m+s)$ for each dimension, where s is the number of swaps.

Details of sorting process are given in Algorithm 3. During the process, every iteration of the algorithm removes an end-point from the $EList$ (line 4), inserting it into the correct position in the already-sorted list (line 33), until no end-points remain. If insertion (swap) occurs, the algorithm then carries out the divide-

Algorithm 3: Re-Sort the Lists of End-points

```
Data: $[a, b]$: a time interval
Data: $EList$: a list of extent end-points of the corresponding dimension
Data: $\boldsymbol{R}[i]$, $\boldsymbol{R}[j]$: two regions that contain the endpoints $EList[i]$ and $EList[j]$,
      respectively
Data: $\boldsymbol{AV}_1$, $\boldsymbol{AV}_2$: two temporary variables to hold the instance of AASVs
Data: $Stack$: a stack storing a pair of AASVs and a subinterval $I$
Data: $isOverlapped$: a boolean variable indicating whether the two regions overlap at
      a certain time
Result: $\boldsymbol{RS}$: a result set storing the pairs of regions that overlap with each other
begin
    foreach dimension do
        for $i \leftarrow 1$ to $Size(EList) - 1$ do
            $temp \leftarrow EList[i]$;
            $j \leftarrow i - 1$;
            while $j \geq 0$ AND $EList[j] > temp$ do
                $isOverlapped \leftarrow false$;
                $Stack.push\ (\boldsymbol{R}[i].GetAASV(),\boldsymbol{R}[j].GetAASV(),[a, b])$;
                while Stack is not empty do
                    $\boldsymbol{AV}_1 \leftarrow Stack.top.\boldsymbol{AV}_1$;
                    $\boldsymbol{AV}_2 \leftarrow Stack.top.\boldsymbol{AV}_2$;
                    $I \leftarrow Stack.top.I$;
                    $Stack.pop()$;
                    if $\boldsymbol{AV}_1 \cap \boldsymbol{AV}_2 \neq \emptyset$ then
                        if I is smaller than a threshold then
                            $isOverlapped \leftarrow true$;
                            break;
                        else
                            Split $I$ into two equal subintervals $I_1$ and $I_2$;
                            $Stack.push\ (\boldsymbol{AV}_1(I_1),\boldsymbol{AV}_2(I_1),I_1)$;
                            $Stack.push\ (\boldsymbol{AV}_1(I_2),\boldsymbol{AV}_2(I_2),I_2)$;
                        end
                    end
                end
                if isOverlapped then
                    $\boldsymbol{RS} \leftarrow \boldsymbol{RS} \cup (\boldsymbol{R}[i], \boldsymbol{R}[j])$;
                else
                    $\boldsymbol{RS} \leftarrow \boldsymbol{RS} - (\boldsymbol{R}[i], \boldsymbol{R}[j])$;
                end
                $EList[j + 1] \leftarrow EList[j]$;
                $j \leftarrow j - 1$;
            end
            $EList[j + 1] \leftarrow temp$;
        end
    end
end
```

and-conquer approach to determine whether the two regions that contain the two end-points in question overlap (lines 7-24). The whole process is repeated until all *EList* are sorted.

One important difference between this algorithm and Algorithm 1 is that we use swept volumes instead of regions. The size of update and subscription regions is rarely changed during runtime; however, the size of their AASVs may change frequently due to the arbitrariness of the regions' trajectory. This does not affect the precision of finding potentially overlapped AASVs, but it might increase the chance of swap during the insertion sort process. The runtime performance of this approach is evaluated in the next section.

4.3 Performance Evaluation

This section presents an evaluation of the proposed space-time interest matching algorithms. Several sets of experiments were carried out to compare the performance of four approaches, namely:

1. Discrete interest matching by brute-force (DIM)

2. Space-time interest matching by brute-force (CIM)

3. Discrete interest matching by efficient sorting algorithm (SDIM)

4. Space-time interest matching by efficient sorting algorithm (SCIM)

The DIM approach performs interest matching for all n update regions and m subscription regions at discrete time intervals. Therefore, the events between two consecutive time-steps are ignored. The CIM approach also performs interest matching for all region pairs; however, instead of testing the overlap status of the actual regions at discrete time intervals, it carries out the pairwise space-time matching presented in Section 4.1 to test the overlap status of their swept volumes. The SDIM approach is the efficient insertion sort algorithm presented in Section 3. It is chosen as a comparison target because it is theoretically one of the fastest (linear time in the general case) discrete algorithms. The SCIM approach uses the sorting algorithm presented in Section 4.2, which efficiently culls out the region pairs that are unlikely to overlap with each other. It then carries out the divide-and-conquer algorithm similar to CIM, in order to find out the missing events.

The filtering precision of the four approaches is similar as they are all aura-based schemes. Therefore, the evaluation only focuses on their runtime efficiency and event-capturing ability.

4.3.1 Implementation and Experimental Set-ups

The four algorithms were implemented in C++. All of the tests were run on an Intel Core2 Duo E6850 3.0GHz with 4GB main memory and used the following experimental set-up:

- *Entity Movement*: All entities move in a random direction and undergo linear translational motion (i.e., the degree of $\vec{\tau}(t)$ is equal to 1).
- *Entity Speed*: The speed factor (SF) represents the average speed of the entities in proportion to its region length. The value of SF varies in different sets of experiments.

- *Entity Distribution*: The entities are distributed randomly across the virtual space.
- *Number of Dimensions*: All simulations were performed in three-dimensional space.
- *Number of Regions* An update region and a subscription region were associated with each moving entity.
- *Execution Time Measurement*: Average execution time of the matching algorithms was measured over 10,000 time-steps.
- *Threshold of Space-Time Interest Matching*: For all sets of experiments, except for Section 4.3.4, the threshold of space-time interest matching algorithms was set to $\delta t/64$ (see the evaluation of threshold in Section 4.3.4).

4.3.2 Event-Capturing Ability

The first set of experiments compares DIM, CIM, SDIM and SCIM for their ability to capture missing events with the number of virtual entities extending from 100 to 1000. In order to do so, we counted the number of region overlaps detected by the four approaches. The SF is an important factor in these experiments, because the higher the average speed of the entities, the greater the chance of region overlap between consecutive time-steps. The exact value of the entity SF should be dependent on the application domain. In our simulations, three different values representing low (1), medium (10) and high (20) SF were used.

Figure 4.4 shows the overlap count of the four approaches. It is not difficult to see that when the number of entities increases, the overlap count of all approaches

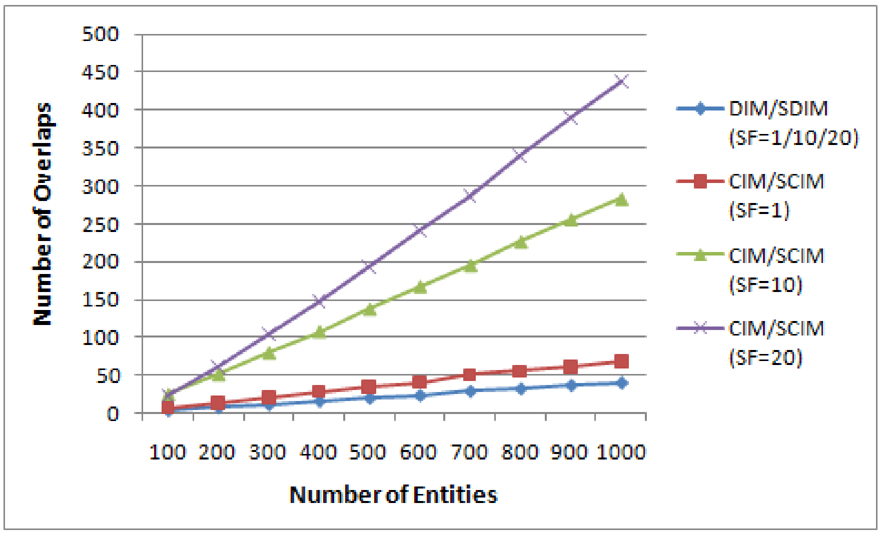


Figure 4.4: Comparing the Event-Capturing Ability of Interest Matching Approaches (Number of Entities varies)

also increases. This is due to the increase in the chance of region overlap when we add more entities to the scene.

The results also suggest that the number of overlaps detected by both DIM and SDIM is independent of the value of SF. As mentioned previously, we expected that when the entity speed becomes higher (e.g., SF = 20), more regions would overlap each other. However, since DIM and SDIM can only perform interest matching at discrete time intervals, no matter how many overlaps occur between discrete time-steps, they would not be able to detect them. Therefore, the number of overlaps detected by discrete algorithms remains constant under various conditions of entity speed.

Another observation is that CIM and SCIM have similar overlap count under the same value of SF. As they both employ swept volumes and the divide-and-conquer algorithm to perform interest matching, their event-capturing ability should be the same.

Most importantly, the results show that both CIM and SCIM capture more overlaps than the discrete algorithms; and the difference becomes significant when SF increases. This indicates that when the entities move at high speed (e.g. SF=20), many overlaps are ignored by discrete algorithms where the space-time algorithms are still able to capture them. The difference of overlap count between discrete and space-time algorithms can be regarded as the "missing events".

4.3.3 Runtime Efficiency

The second set of experiments compares the runtime efficiency of the DIM, CIM, SDIM and SCIM with the number of virtual entities extending from 100 to 1000.

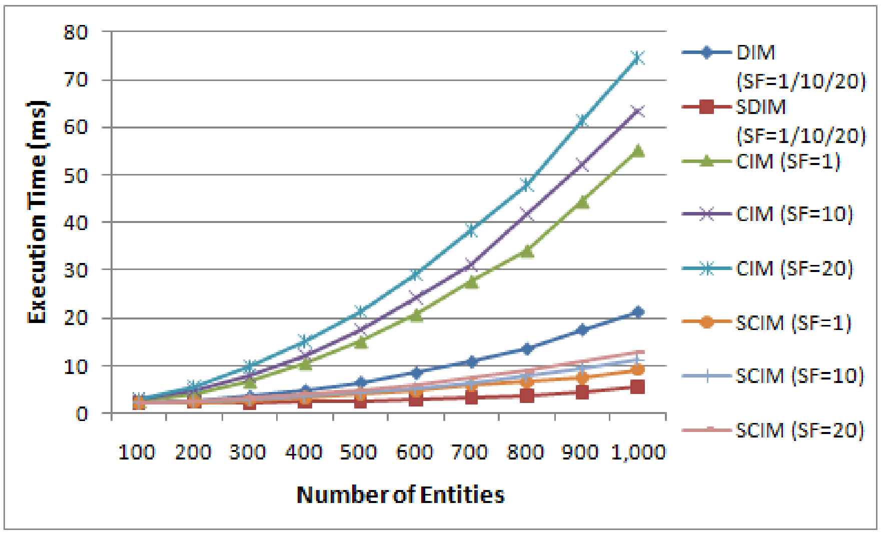


Figure 4.5: Comparing the Execution Time of Interest Matching Approaches (Number of Entities varies)

Figure 4.5 shows the execution time of the four approaches. The first observation is that although both DIM and CIM perform interest matching by brute-force, the CIM approach requires more computational effort than DIM. Such difference becomes significant when the number of entities is gradually increased. Similarly, SCIM also requires more computational effort than SDIM. This discrepancy is due to the overhead introduced by the divide-and-conquer algorithm as well as the computation of swept volumes.

The second observation is that the approaches based on sorting (SDIM and SCIM) perform much faster than the brute-force approaches (DIM and CIM). This confirms that the sorting algorithms, which cull out the region or AASV pairs that are unlikely to overlap with each other before the pairwise overlap tests are carried out, are very efficient in reducing the computational overhead of interest matching.

The results also show that when the value SF increases, both CIM and SCIM would spend slightly more time on the matching process. This is due to the increase in size of the swept volumes when the entities move at high speed, resulting in an increase of the chance of overlap, and thus more computational effort is spent on the divide-and-conquer process. However, the execution time of DIM and SDIM is not affected by the change of SF value since they can only perform interest matching at discrete time intervals.

4.3.4 Threshold of Space-Time Interest Matching

The third set of experiments evaluate the performance of the space-time algorithms with the user defined threshold of the divide-and-conquer algorithm extending from $\delta t/2$ to $\delta t/64$, where δt is the time-step of simulation. Moreover, the number of entities and the value of SF were set to constant (1000 and 20, respectively). We also include the performance of discrete algorithms as a reference.

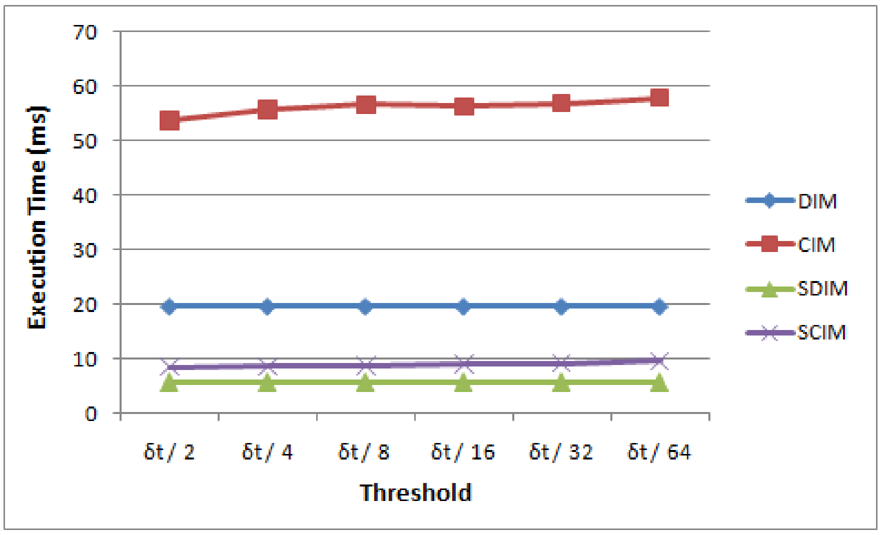


Figure 4.6: Comparing the Execution Time of Interest Matching Approaches (Threshold varies)

Figure 4.6 shows the execution time of the four approaches. The results indicate that when the value of the threshold decreases, the CIM approach would spend slightly more time on the matching process. This is because choosing a smaller threshold would cause the divide-and-conquer algorithm to perform more recursions. It was expected that the same effect would be applied on the SCIM too, but the significance cannot be shown on the figure. Furthermore, the performance of discrete algorithms is not affected by the change of threshold value since they do not employ the divide-and-conquer approach.

Figure 4.7 shows the overlap count of the four approaches. As can be seen the overlap count of the space-time algorithms gradually drops to a steady number ($\approx$ 95) as the threshold decreases. This is due to the behaviour of the divide-and-conquer approach described in Section 4.1.2. When a large threshold is chosen, most of the potentially overlapped region pairs would be considered as overlapped, which might also include a large number of false positives. Decreasing

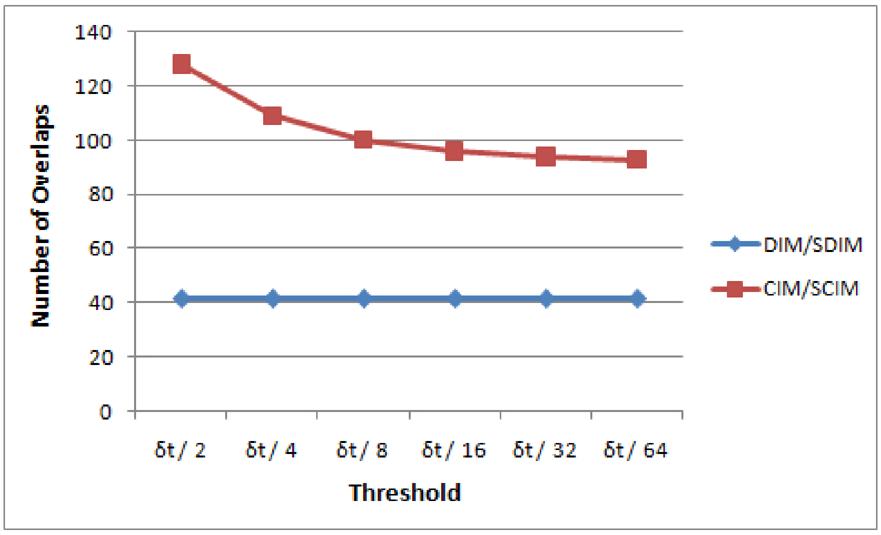


Figure 4.7: Comparing the Event-Capturing Ability of Interest Matching Approaches (Threshold varies)

the threshold would cause the algorithm to perform more recursions, but thus increases the accuracy of the test. However, it is obvious that choosing a threshold smaller than $\delta t/64$ would not significantly increase the accuracy. We therefore used this threshold value in all other experiments described in this chapter.

As mentioned earlier, the performance of discrete algorithms is not affected by the change of threshold value since they do not employ the divide-and-conquer approach.

4.3.5 Space-time Interest Matching vs. Frequent Discrete Interest Matching

As described in Section 2.8.1, one simple approach to solve the missing event problem is to reduce the time-step of simulation and perform frequent discrete interest matching. The last set of experiments aimed to investigate whether the proposed SCIM algorithm has better performance than this simple approach. In order to do so, we only picked the SDIM approach as the comparison target since it has the best runtime efficiency. We reduce the time-step of discrete interest matching from δt to $\delta t/i$ and carry out SDIM i times per δt, for $i = 1, 2, ..., 10$. During the experiments, the following variables were set to constant: SF (20), number of entities (1000).

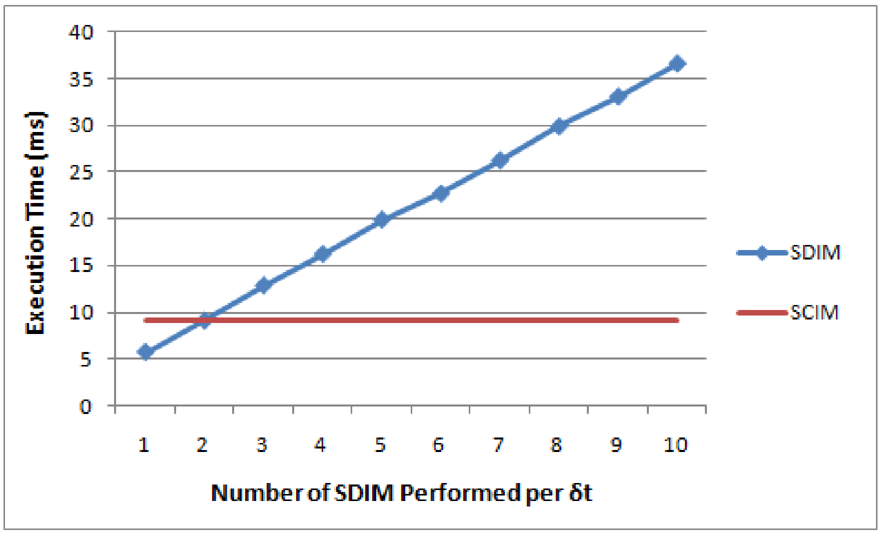


Figure 4.8: Comparing the Execution Time of SCIM and SDIM (Frequency of SDIM varies)

The execution time and overlap count of the two approaches are given in Figure 4.8 and Figure 4.9, respectively. The results clearly suggest that the SCIM approach is a better solution to the missing event problem. If we perform SDIM more than 2 times per δt, it would spend more execution time than the SCIM approach. The difference becomes significant when we increase the frequency of SDIM. Nevertheless, even if we perform SDIM 10 times per δt, its event-capturing ability is still worse than SCIM.

In a sense, both approaches enhance their event-capturing ability by increas-

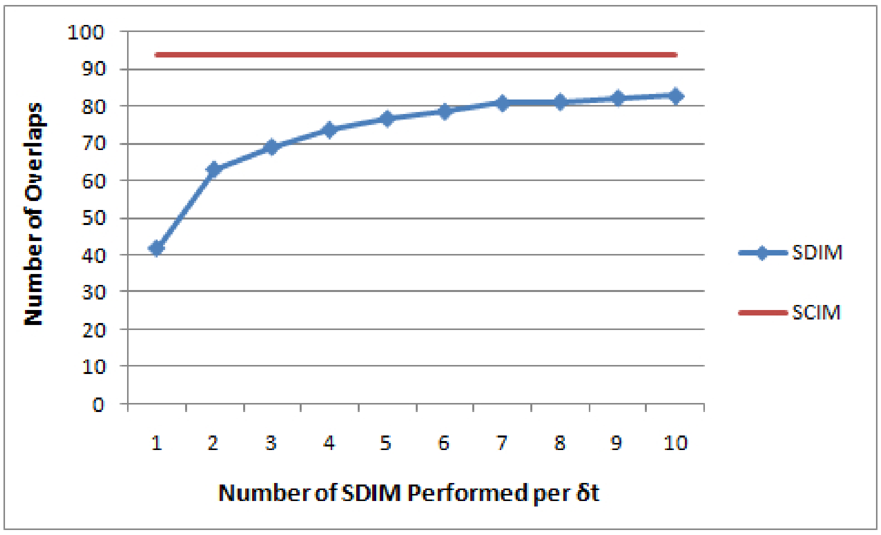


Figure 4.9: Comparing the Event-Capturing Ability of SCIM and SDIM (Frequency of SDIM varies)

ing the frequency of overlap test. The frequent SDIM approach incurs large computational overhead because it performs extra overlap tests for *all* regions per δt. The proposed SCIM approach, on the other hand, culls out the region pairs that are unlikely to overlap before it increases the test frequency for potentially overlapped pairs. This is obviously more computationally efficient than the frequent SDIM approach. Furthermore, instead of performing pairwise overlap tests for all subintervals $\delta t/i$, SCIM carries out the divide-and-conquer algorithm described in Section 4.1, which allows it to avoid much of the unnecessary pairwise comparisons. This further enhances the runtime efficiency of SCIM.

4.4 Summary

This chapter has presented formal definitions, theorems and proofs of using swept volume for space-time interest matching. This approach aims to capture the missing events and interactions that traditional discrete interest matching approaches would fail to report. Although an exact swept volume is difficult to compute, we have proposed a new model called AASV, which can be used as an approximate solution for swept volume. Furthermore, we have demonstrated how to integrate the AASV model into the efficient sorting algorithm presented in the previous chapter. The new sorting algorithm can efficiently cull out the AASV pairs that are unlikely to overlap, before a more computationally intensive pairwise overlap test is carried out. Lastly, this sorting algorithm can be combined with the parallel approach presented in the next chapter, which can further minimise the computational overhead of space-time interest matching.

Experimental evidence presented in this chapter has demonstrated that the space-time interest matching approach can capture most of the events that are ignored by the discrete algorithms. It has also shown that the sorting algorithm can significantly reduce the computational overhead of space-time interest matching. Most importantly, the results have demonstrated that even if we increase the frequency of discrete interest matching, it is still outperformed by the proposed algorithm in terms of runtime efficiency and event-capturing ability.

Chapter 5

Parallel Interest Matching Algorithms for Shared-Memory Multiprocessors

In this chapter, a new approach of interest matching is presented which facilitates parallelism by distributing the workload of the matching process across multiple processors. Since it is increasingly common to deploy commercial DVE applications on shared-memory multiprocessors, using the parallel algorithm for these applications is expected to be more suitable than the existing serial algorithms.

The algorithm divides the matching process into two phases. In the first phase it employs a spatial data structure called uniform subdivision to efficiently decompose the virtual space into a number of subdivisions. We define as work unit (WU) the interest matching process within a space subdivision. In the second phase, WUs are distributed across different processors and can be processed concurrently.

The parallel algorithm is suitable to apply for traditional discrete interest matching as well as the space-time interest matching approach presented in Chapter 4. For the sake of simplicity, the discussion of this chapter is first focused on the former. The integration of the latter is straightforward, which is described in Section 5.2.1.

5.1 Spatial Decomposition

Uniform subdivision is a common spatial data structure which has long been used as a mean of rapid retrieval of geometric information. Over the years, it has been studied extensively in many fields such as computer graphics and robotics. The idea of using hashing for subdivision directory was first described in an early article written by Rabin [102] and was later discussed more generally in Bentley and Friedman's survey [103]. This section presents the formal definitions of uniform subdivision, which leads to the discussion in the subsequent sections where they are used for hash indexing and rapid WU distribution.

Formally, the virtual space $\boldsymbol{S}$ can be define as a multidimensional point set that contains all entities in the virtual world. Therefore, all update or subscription regions can be regarded as the subsets of $\boldsymbol{S}$.

Definition 9. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1, 2, ..., n$.

$$\boldsymbol{S} = \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d \leq x_d < SMAX_d, \\ for \quad d = 1, 2, ..., n\}.$$

Alternatively, $\boldsymbol{S}$ can be expressed as the Cartesian product of its one-dimensional boundaries.

Definition 10. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension, for $d = 1, 2, ..., n$.

$$\begin{aligned}\boldsymbol{S} &= [SMIN_1, SMAX_1) \times [SMIN_2, SMAX_2) \times ... \times [SMIN_n, SMAX_n) \\ &= \prod_{d=1}^{n} [SMIN_d, SMAX_d).\end{aligned}$$

The hashing approach requires decomposing $\boldsymbol{S}$ into uniform subdivisions. Each subdivision represents a slot in the hash table, which is labelled by a multidimensional hash table index.

Definition 11. Let $[SMIN_d, SMAX_d)$ be the boundary of a space $\boldsymbol{S}$ in d dimension. The boundary can be uniformly divided into N_d sub-boundaries with unit length L_d, such that

$$L_d = \frac{SMAX_d - SMIN_d}{N_d}$$

$\forall N_d \in \mathbb{Z}^+$, $\forall L_d \in \mathbb{R}^+$, for $d = 1, 2, ..., n$.

Definition 12. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ is labelled by a multidimensional hash table index $(z_1, z_2, ..., z_n)$, such that

$$\boldsymbol{Z}(z_1, z_2, ..., z_n) = \{(x_1, x_2, ..., x_n) \mid x_d \in \mathbb{R} \wedge SMIN_d + z_d L_d \leq x_d < SMIN_d + (z_d + 1)L_d, \quad for \quad d = 1, 2, ..., n\}$$

for $z_d = 0, 1, ..., N_d - 1$.

Similar to all axis-aligned point sets, the uniform subdivision can be express as the Cartesian product of its one-dimensional boundaries, which is given in **Definition 13**.

Definition 13. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The uniform subdivision $\boldsymbol{Z}$ of $\boldsymbol{S}$ can be defined as

$$\begin{aligned}\boldsymbol{Z}(z_1, z_2, ..., z_n) &= [SMIN_1 + z_1 L_1, SMIN_1 + (z_1 + 1)L_1) \\ &\quad \times [SMIN_2 + z_2 L_2, SMIN_2 + (z_2 + 1)L_2) \times ... \\ &\quad \times [SMIN_n + z_n L_n, SMIN_n + (z_n + 1)L_n) \\ &= \prod_{d=1}^{n} [SMIN_d + z_d L_d, SMIN_d + (z_d + 1)L_d)\end{aligned}$$

for $z_d = 0, 1, ..., N_d - 1$.

Theorem 7. Given a set of all hash table indices

$$\boldsymbol{HI} = \{(z_1, z_2, ..., z_n) \mid z_d = 0, 1, ..., N_d - 1 \wedge d = 1, 2, ..., n\}$$

where N_d is the number of subdivisions of space $\boldsymbol{S}$ in d dimension. Then, $\boldsymbol{S}$ can be expressed as the union of all uniform subdivisions, such that

$$\boldsymbol{S} = \bigcup_{k \in \boldsymbol{HI}} \boldsymbol{Z}(k).$$

Proof. By **Definition 10**, we derive

$$\begin{aligned}
\boldsymbol{S} &= [SMIN_1, SMAX_1) \times [SMIN_2, SMAX_2) \times ... \times [SMIN_n, SMAX_n) \\
&= \bigcup_{z_1=0}^{N_1} [SMIN_1 + z_1 L_1, SMIN_1 + (z_1+1)L_1) \\
&\quad \times \bigcup_{z_2=0}^{N_2} [SMIN_2 + z_2 L_2, SMIN_2 + (z_2+1)L_2) \times ... \\
&\quad \times \bigcup_{z_n=0}^{N_n} [SMIN_n + z_n L_n, SMIN_n + (z_n+1)L_n) \\
&= \bigcup_{k \in \boldsymbol{HI}} \boldsymbol{Z}(k).
\end{aligned}$$

□

5.2 First Phase: Hashing

During the simulation, regions are hashed into the hash table. The algorithm uses the coordinate of a region's vertex as a hash key. Given a key k, a hash value $H(k)$ is computed, where $H()$ is the hash function. The hash value is an n-dimensional index which can be matched with the index of a space subdivision, and therefore indicating that which subdivision the vertex lies in. Hence, the regions with hash key k are stored in slot $H(k)$. The hash function is given in

Definition 14.

Definition 14. Let $[SMIN_d, SMAX_d)$ be the boundary of a space in d dimension, for $d = 1, 2, ..., n$. The boundary is uniformly divided into N_d sub-boundaries with unit length L_d. The hash function for transforming a key k_d into a hash value is defined as

$$H : \mathbb{R}^n \to \mathbb{Z}^n, H(k_d) = \lfloor \frac{k_d - SMIN_d}{L_d} \rfloor$$

There are two important properties of using a hash table for spatial decomposition. First, hash table collision means that regions in the same slot are potentially overlapped with each other; therefore, further investigation on their overlap status is required. This process will be left to the second phase of the algorithm. Second, if a region lies in multiple space subdivisions, it would be hashed into all of them. The algorithm assumes that the size of region is much smaller than a space subdivision. Therefore, a region would exist in at most four slots in the two-dimensional space (at most eight slots in the three-dimensional space). This assumption ensures that the computational complexity of the hashing process would be bounded by a constant.

Figure 5.1 illustrates the basic concept of the spatial hashing for two-dimensional space. In the figure, region A is hashed into slot (0,1); region B is hashed into slots (0,0), (0,1), (1,0) and (1,1); region C is hashed into slots (1,1) and (1,2); region D is hashed into (1,0), (1,1), (2,0) and (2,1).

The basic steps to construct a hash table are given in Algorithm 4. Note that if not all vertices of a region are hashed into the same slot, then the region exists in multiple subdivisions.

Algorithm 4: Algorithm for Hash Table Construction (Region)

```
Data: $\boldsymbol{S}$: a $n$-dimensional virtual space
Data: $\boldsymbol{Z}$: an uniform subdivision of $\boldsymbol{S}$
Data: $k$: a $n$-dimensional hash table index
Data: $\boldsymbol{R}$: a region
Data: $v$: a vertex of $\boldsymbol{R}$
Data: $H()$: a hash function
Data: $HashTable$: a hash table
1  begin
2      Decompose $\boldsymbol{S}$ into a list of $\boldsymbol{Z}$;
3      foreach $\boldsymbol{Z}$ do
4          Determine the index $k$ for $\boldsymbol{Z}$;
5      end
6      foreach $k$ do
7          $HashTable.AddSlot(k)$;
8      end
9      foreach $\boldsymbol{R}$ do
10         foreach $v$ of $\boldsymbol{R}$ do
11             $HashTable.Slot[H(v)].AddRegion(\boldsymbol{R})$;
12         end
13     end
14 end
```

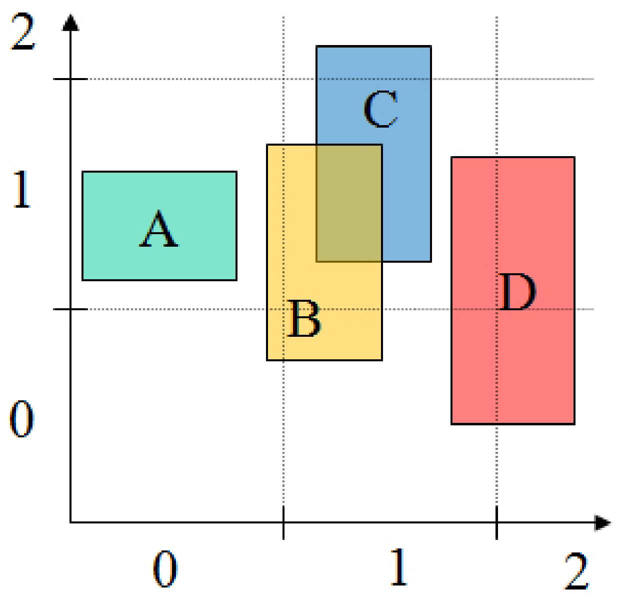


Figure 5.1: Hashing for Space Subdivisions

The hash table is constructed at the initialisation stage. During runtime, the position and size of regions may be frequently modified. Therefore, the algorithm needs to perform rehashing for the regions (line 9-13) at every time-step. The complexity of this process is $O(n + m)$ where m is the number of subscription regions and n is the number of update regions.

5.2.1 Conversion to Space-Time Interest Matching

Applying the parallel algorithm to space-time interest matching is straightforward. All that needs to be changed is to replace regions with AASVs. Therefore, hashing would be performed for the AASVs' vertices instead of the regions'. Algorithm 5 describes the basic steps of constructing a hash table for the AASVs.

Algorithm 5: Algorithm for Hash Table Construction (AASV)

```
Data: $\boldsymbol{S}$: a $n$-dimensional virtual space
Data: $\boldsymbol{Z}$: an uniform subdivision of $\boldsymbol{S}$
Data: $k$: a $n$-dimensional hash table index
Data: $\boldsymbol{AV}$: an AASV
Data: $v$: a vertex of $\boldsymbol{AV}$
Data: $H()$: a hash function
Data: $HashTable$: a hash table
begin
    Decompose $\boldsymbol{S}$ into a list of $\boldsymbol{Z}$;
    foreach $\boldsymbol{Z}$ do
        Determine the index $k$ for $\boldsymbol{Z}$;
    end
    foreach $k$ do
        $HashTable.AddSlot(k)$;
    end
    foreach $\boldsymbol{AV}$ do
        foreach $v$ of $\boldsymbol{AV}$ do
            $HashTable.Slot[H(v)].AddAASV(\boldsymbol{AV})$;
        end
    end
end
```

5.3 Second Phase: Sorting

After the hashing stage, each slot of the hash table represents a WU which will be distributed across different processors. The algorithm then places the WUs on a task queue. Each processor fetches WUs from the queue and performs interest matching for the corresponding space subdivisions. Since only one processor has the authority to manage each space subdivision, there will be no ambiguous matching result. Figure 5.2 illustrates how the processors and task queue interact during runtime.

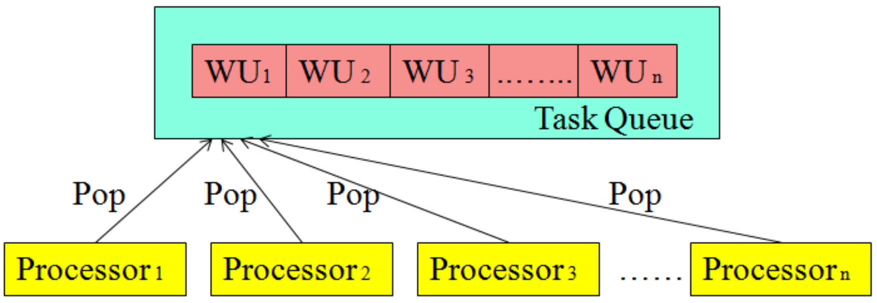


Figure 5.2: Task Queue for WU Distribution

Since every subdivision is isolated during the matching process, lock mechanisms for each WU are unnecessary. Locking is only required when a processor tries to fetch a WU from the task queue.

5.3.1 Sorting Algorithms

The spatial decomposition approach essentially transforms the large-scale interest matching process into several individual sub-problems. When a WU is being processed, each processor carries out a matching process only for the regions or AASVs within the WU. As discussed in the previous chapters, using a brute-force approach to determine the overlapping status of the regions or AASVs is computationally expensive. Therefore, a sorting algorithm based on dimension reduction would help to increase the computational efficiency. For discrete interest matching, Algorithm 1 is employed, where for space-time interest matching, Algorithm 3 is employed. The integration of these algorithms into the parallel interest matching framework is simple and straightforward. No additional matching steps are required.

5.3.2 Load Balancing

As discussed in [104], the task queue approach is desired for task distribution and provides very good load sharing for shared-memory multiprocessor systems. When a processor finishes processing a WU, it would fetch another WU from the task queue immediately unless the queue is empty. Therefore, no processor would be idle until all WUs are fetched.

The worst case happens only when all regions or AASVs reside in a single space subdivision. In this situation, a single processor would be responsible for the matching of all of them.

5.4 Theoretical Analysis on Computational Complexity

This section presents the theoretical analysis of the proposed parallel interest matching approach. As described in Chapter 1, the computational complexity of interest matching by brute-force is $O(nm)$ where m is the number of subscription regions and n is the number of update regions. Other approaches such as [92, 94, 93, 2] have the complexity of $O((n+m)\log(n+m))$ or even $O(n+m)$ in the general case. The parallel algorithm, as we expected, would outperform the existing approaches in theory and in practice when running on multiprocessor machines. According to Algorithm 4, the following are the major steps of the algorithm:

- In the first phase, the expected time to compute the hash value $H(k)$ for a vertex k is $O(1)$. Assume the simulation is performed in three-dimensional space, each region would only have eight vertices, thus the hashing process for each region is still $O(1)$.

- Computing the hash value $H(k)$ for all regions requires $O(n+m)$ time. However, since each computation of $H(k)$ is independent, this can be shared among the processors. Therefore the computational complexity of the hashing process is $O(\frac{n+m}{p})$, for p number of processors.

- In the second phase, each WU is being processed and can be treated as an independent sub-problem. Assume the regions are distributed uniformly throughout the virtual space, assume further that temporal coherence is exploited; hence, during runtime, the complexity of re-sorting the list of endpoints using insertion sort is $O(\frac{n+m+s}{p})$ for each dimension, where s is the number of swaps. Since most of the DVEs are in two- or three-dimensional space, the number of dimensions is constant.

- The worst case happens when all $n+m$ regions reside in a single space subdivision. In this situation, the workload cannot be shared. A single processor would be responsible for performing matching for all regions. The computational complexity of the second phase would become $O(n+m+s)$.

5.5 Performance Evaluation

This section presents the evaluation of the proposed parallel algorithms. Several sets of experiments were carried out to compare the performance of six approaches, namely:

1. Discrete interest matching by brute-force (DIM)
2. Space-time interest matching by brute-force (CIM)
3. Discrete interest matching by scalable insertion-sort algorithm (SDIM)
4. Space-time interest matching by scalable insertion-sort algorithm (SCIM)
5. Discrete interest matching by parallel algorithm (PDIM)
6. Space-time interest matching by parallel algorithm (PCIM)

The DIM approach performs interest matching for all n update regions and m subscription regions at discrete time intervals. The CIM approach also performs matching for all region pairs; however, instead of testing the overlap status at discrete time intervals, it carries out pairwise space-time interest matching, as given in Algorithm 2, in order to capture the missing events. The SDIM approach is the sorting approach presented in Chapter 3. It was chosen as a comparison target because it is theoretically the fastest (linear time in the general case) algorithm among all serial matching algorithms. The SCIM approach is an implementation of Algorithm 3. It performs sorting to efficiently cull out the region pairs that are unlikely to overlap with each other, and carries out space-time matching for the remaining pairs. PDIM and PCIM are the two parallel algorithms presented in this chapter, which exploit parallelism by distributing the workload across multiple processors. The PDIM approach is designed for discrete interest matching, while the PCIM approach performs space-time interest matching similar to the CIM approach.

Of the three major design requirements described in Section 2.2.1, this evaluation only focuses on the runtime efficiency of the six approaches. As discussed in Section 3.5.3, the filtering precision of the matching approaches would be similar if they all adopt an aura-based scheme. Therefore, performing further experiments to evaluate the filtering precision of these six approaches is unnecessary. Furthermore, the event-capturing ability of PCIM is similar to CIM and SCIM since they all employ swept volumes and the divide-and-conquer algorithm to perform space-time interest matching. In Section 3.5.3, it has already been shown experimentally that the space-time algorithms (CIM and SCIM) have better event-capturing ability than the discrete algorithms.

5.5.1 Implementation and Experimental Set-ups

The six algorithms were implemented in C++ with the use of multi-threading. All of the tests were run on a workstation, which has two Intel Xeon E5634 2.4Ghz 6-core CPUs with 48GB main memory and used the following experimental set-up:

- *Entity Distribution*: The entities are distributed randomly across the virtual space.

- *Entity Movement*: All entities move in a random direction and undergo linear translational motion (i.e., the degree of $\vec{\boldsymbol{\tau}}(t)$ is equal to 1).
- *Entity Speed*: The speed factor (SF) represents the average speed of the entities in proportion to its region length. The value of SF varies in different sets of experiments.
- *Number of Dimensions*: All simulations were performed in three-dimensional space.
- *Number of Regions* An update region and a subscription region were associated with each moving entity.
- *Execution Time Measurement*: Average execution time of the matching algorithms was measured over 10,000 time-steps.
- *Number of WUs*: The number of WUs is dependent on the granularity of spatial decomposition, which was assigned statically. The optimal granularity value was determined through experiments, which are presented in Section 5.5.2.

5.5.2 Granularity of Spatial Decomposition

The spatial decomposition approach described in Section 5.1 requires an optimal granularity to achieve an optimal use of resources. This is similar to the virtual world partitioning problem as we have discussed in Section 2.3.3. In this section, we present the results of a set of experiments that we have conducted to determine the optimal granularity of partitioning.

Since uniform subdivisions are employed for the parallel algorithm, the granularity of spatial decomposition is dependent on the number of sub-boundaries per dimension (denoted by N_d in **Definition** 11). We take $N_1 = N_2 = N_3$, which implies that each subdivision is a cube in shape. We measured the execution time of PDIM and PCIM, with N_d extending from 2 to 10. The number of entities was set to 1000 and the SF was set to 20. Throughout the experiments, all 12 cores of the two Intel E5634s were enabled.

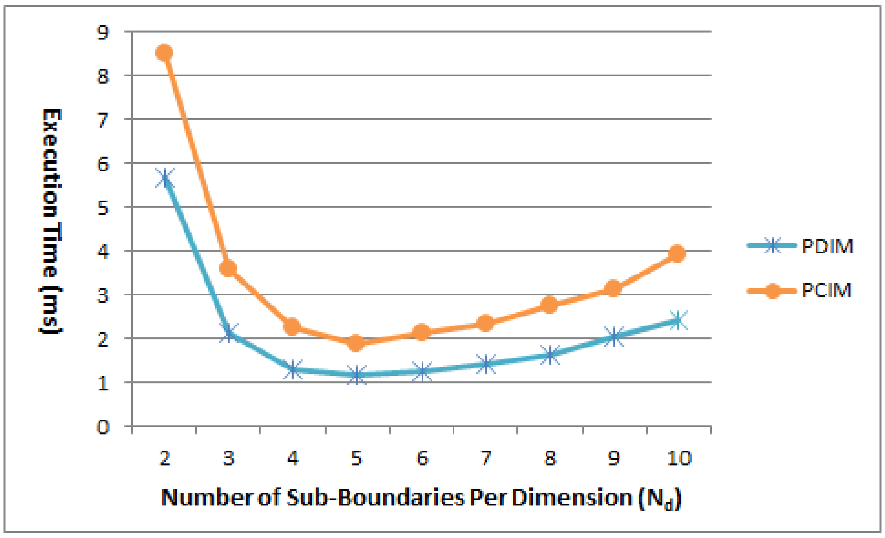


Figure 5.3: Comparing the Execution Time of Parallel Interest Matching Approaches (Number of Sub-Boundaries varies)

The results are given in Figure 5.3. As we can see in the graph, there is some significant runtime overhead when N_d is equal to 2 (i.e., the number of WUs is equal to 8). This is due to the fact that the number of WUs is less than the number of physical processors (12), which implies four physical cores would be idle at each time step of simulation, resulting in a poor utilisation of computational resources. However, if a large N_d is chosen, the size of job queue becomes large and thus overhead would be introduced due to the increase in the frequency of job queue access. According to results shown in Figure 5.3, we can conclude that, for the current experimental set-up, the optimal value of N_d for both PDIM and PCIM is 5. This value would be used in all other experiments described in this chapter.

5.5.3 Number of Entities

The second set of experiments compares the runtime efficiency of the DIM, CIM, SDIM, SCIM, PDIM and PCIM with the number of virtual entities extending from 100 to 1000. The value of SF was set to 20 and the threshold of space-time algorithms was set to $\delta t/32$, where δt is the time-step of simulation. Throughout the experiments, all 12 cores of the two Intel E5634s were enabled.

Figure 5.4 shows the execution time of the six approaches. It is not difficult to see that PDIM has the best performance among discrete algorithms, where DIM has the poorest performance. The difference becomes significant when the number of entities is gradually increased. Similarly, PCIM has the best performance among space-time algorithms, where CIM has the poorest performance. These observations indicate that the parallel approaches (PCIM and PDIM) scale much better than the serial approaches when running on a shared-memory multiprocessor machine. Furthermore, the space-time algorithms require more computational effort than discrete algorithms under the same matching approach (i.e., brute-force, sorting, or parallelism). The discrepancy is due to the overhead introduced by the divide-and-conquer algorithm as well as the computation of AASVs.

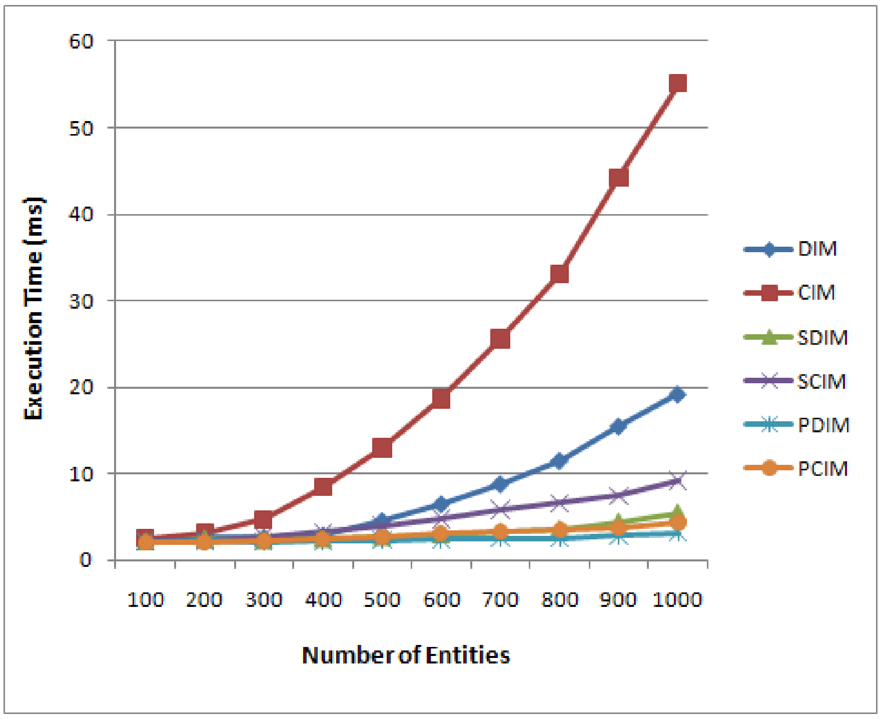


Figure 5.4: Comparing the Execution time of Interest Matching Approaches (Number of Entities varies)

5.5.4 Speed Factor

The third set of experiments compares the runtime efficiency of the six algorithms with the value of SF extending from 2 to 20. The number of entities was set to 1000 and the threshold of space-time algorithms was set to $\delta t/32$. Throughout the experiments, all 12 cores of the two Intel E5634s were enabled.

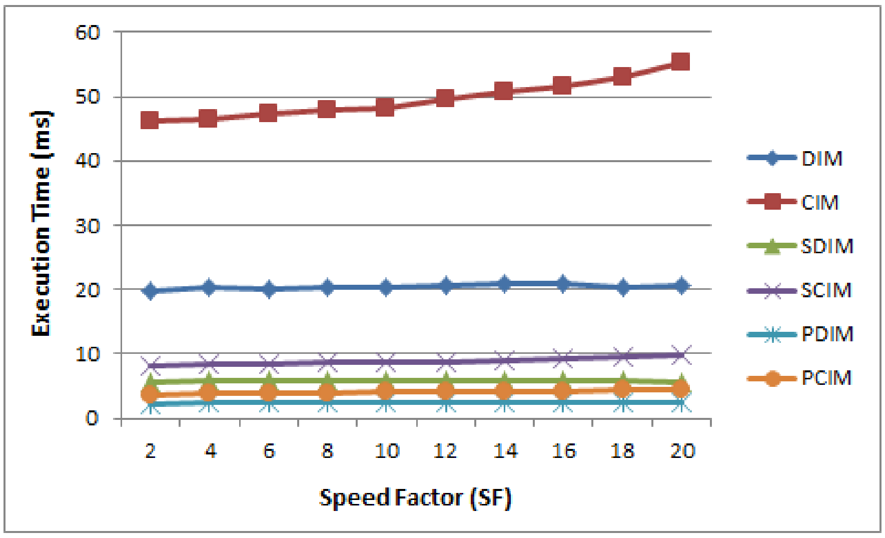


Figure 5.5: Comparing the Execution Time of Interest Matching Approaches (Speed Factor varies)

Figure 5.5 shows the execution time of the six approaches. The results indicate that when the value SF increases, CIM would spend slightly more time on the matching process. This is because the AASVs become bigger when the entities move at higher speed, resulting in an increase of the chance of AASV overlap, and thus more computational effort is spent on the divide-and-conquer process. It is expected that the same effect would be seen in SCIM and PCIM too. However, since these two algorithms are much faster than CIM, the results show that the computational overhead generated by increasing SF is insignificant for these two algorithms.

The execution time of discrete algorithms is not affected by the change of SF value since they only perform interest matching at discrete time intervals.

5.5.5 Threshold of Space-Time Interest Matching

The fourth set of experiments compares the runtime efficiency of the six algorithms with the threshold of space-time algorithms extending from $\delta t/2$ to $\delta t/64$. The number of entities was set to 1000 and the SF was set to 20. Throughout the experiments, all 12 cores of the two Intel E5634s were enabled.

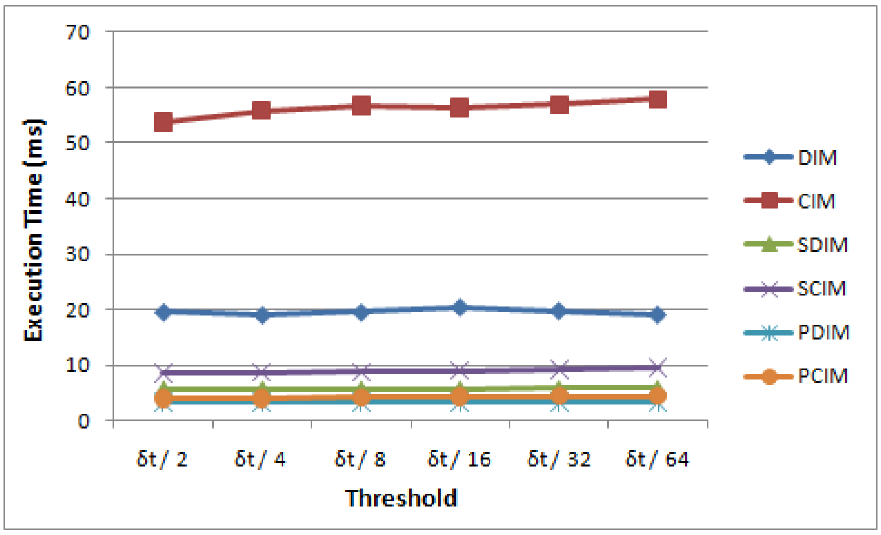


Figure 5.6: Comparing the Execution Time of Interest Matching Approaches (Threshold varies)

Figure 5.6 shows the execution time of the six approaches. The results indicate that when the threshold decreases, the CIM approach would spend slightly more time on the matching process. This is because choosing a smaller threshold would cause the divide-and-conquer algorithm to perform more recursions, resulting in an increase of computational overhead. We expected that the same effect would be applied on SCIM and PCIM too; however, since they are much faster than

CIM, the results show that the computational overhead generated by decreasing the threshold is insignificant for these two algorithms.

The execution time of discrete algorithms is not affected by the change of threshold value since they do not employ the divide-and-conquer method.

5.5.6 Number of Processors

The last set of experiments compares the runtime efficiency of the six approaches when running on different number of processors. In the case of number of required processors less than 12, some of the physical cores of the two Intel E5634s were disabled. The number of working threads was equal to the number of available cores. The number of entities, the speed factor, and the threshold of space-time algorithms were set to the constant value of 1000, 20, and $\delta t/64$, respectively.

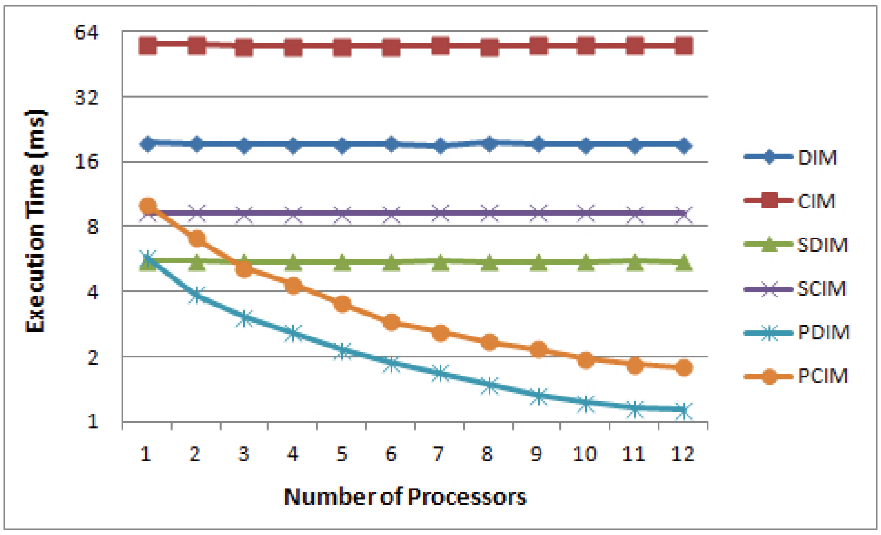


Figure 5.7: Comparing the Execution Time of Interest Matching Approaches (Number of Processors varies)

Figure 5.7 shows the execution time of the six approaches. Since the brute-force and sorting approaches are designed for serial processing, the execution time of these algorithms did not change significantly when the number of active processors increased. The execution time of parallel algorithms, on the other hand, decreased gradually with increase in the number of active processors; this suggests that the proposed algorithm is scalable when running on a multiprocessor machine.

5.5.6.1 Analysis of Speed-up and Efficiency

As Lee has observed that [105], a parallel algorithm rarely attain its maximum efficiency, this is due to both logical and physical constraints. For example, logical constraints may include intrinsic data-dependencies, control dependencies and operator precedences in the parallel algorithm, which force a serial chain of execution amongst the dependent operations, and thus limit the number of operations which may be executed in parallel. Physical constraints may include the control restrictions on the different types of operations which may be executed simultaneously, and the delays due to the communication and competition amongst the interacting components in the computer. Lee therefore defined certain measures to compare the effectiveness of various parallel algorithms, such as speed-up, efficiency, redundancy, utilisation, and quality. The first two measures are widely used today for analysis of parallelism [106].

In Lee's paper, the term "speed-up" refers to how much faster a parallel algorithm is compared to a corresponding serial algorithm, such that

$$S(n,p) = \frac{T(n,1)}{T(n,p)} \tag{5.1}$$

where n is the problem size, p is the number of processors, $T(n,1)$ is the execution time of the serial algorithm, and $T(n,p)$ is the execution time of the parallel algorithm with p processors.

An ideal speed-up (or linear speed-up) is obtained when $S(n,p) = p$. When running an algorithm with ideal speed-up, doubling the number of processors doubles the speed. Therefore it is considered extremely good scalability.

The term “efficiency” represents how well-utilised the processors are in solving the problem, compared to how much effort is wasted in communication and synchronisation. It is defined as

$$E(n,p) = \frac{S(n,p)}{p} = \frac{T(n,1)}{pT(n,p)}. \tag{5.2}$$

Note that a parallel algorithm with ideal speed-up and algorithms running on a single processor have an efficiency of 1.

The experimental results presented in this chapter allow us to analyse the speed-up and efficiency of proposed parallel algorithms based on the calculation of these two measures. Let T_C, S_C and E_C denote the execution time, speed-up and efficiency of PCIM, respectively; and let T_D, S_D and E_D denote the execution time, speed-up and efficiency of PDIM, respectively. The results of the analysis are given in Table 5.1. Note that the problem size n is set to constant (1000).

The results show that the efficiency of both PDIM and PCIM drops gradually when more processors are involved in the matching process. In particular, we can see that when all 12 processors are used in the experiments, the efficiency of both algorithms drops below 50%. Therefore, even if we add more processors to the experiments, the performance gain may not be significant.

p	$T_D(n,p)$	$S_D(n,p)$	$E_D(n,p)$	$T_C(n,p)$	$S_C(n,p)$	$E_C(n,p)$
1	5.784	1	100%	10.142	1	100%
2	3.874	1.493	74.65%	7.102	1.428	71.40%
3	3.041	1.902	63.40%	5.127	1.978	65.94%
4	2.589	2.234	55.85%	4.318	2.349	58.72%
5	2.142	2.700	54.01%	3.542	2.863	57.27%
6	1.876	3.083	51.39%	2.894	3.504	58.41%
7	1.685	3.433	49.03%	2.601	3.899	55.70%
8	1.486	3.892	48.65%	2.356	4.305	53.81%
9	1.325	4.365	48.50%	2.176	4.661	51.79%
10	1.225	4.723	47.23%	1.952	5.196	51.96%
11	1.152	5.023	45.64%	1.844	5.500	50.00%
12	1.129	5.123	42.69%	1.784	5.685	47.37%

Table 5.1: Speed-up and Efficiency of the Parallel Algorithms

5.6 Summary

This chapter presents a parallel appropach for both discrete and space-time interest matching. As reviewed in Section 2.8, over the years, many efficient matching algorithms were proposed to speed up the interest matching process. However, they are all designed for serial processing. The proposed parallel algorithm facilitates workload sharing by dividing a large-scale interest matching problem into several smaller sub-problems and distributing them across multiple processors. Experimental evidence presented in this chapter has demonstrated that the parallel algorithm is more computationally efficient than the existing serial algorithms when running on a shared-memory multiprocessor machine. Moreover, the parallel algorithm can be easily combined with the space-time interest matching algorithm presented in the previous chapter. Although the space-time approach requires additional effort to compute the swept volumes, running it within the parallel framework can further minimise its computational overhead.

The discussion so far has been centred on shared-memory multiprocessor machines, however, the parallel algorithm can be easily extended and applied on cluster of distributed-memory computers. In this case, the processors can no longer exchange data through the shared-memory and therefore additional synchronisation protocols will be required. The application of the parallel algorithm on distributed-memory systems is described in the next chapter.

Chapter 6

Parallel Interest Matching Algorithms for Distributed-Memory Systems

This chapter presents an extension of the parallel algorithm proposed in the previous chapter (which is hereafter referred to as the "shared-memory algorithm"). The new parallel algorithm (which is hereafter referred to as the "distributed-memory algorithm") can be run on a cluster of computers that enables them to work simultaneously and thus increasing the overall runtime efficiency of the matching process. The distributed-memory algorithm is suitable to apply for both discrete and space-time interest matching. For the sake of simplicity, our discussion is first focused on the former.

6.1 Spatial Decomposition

Similar to the shared-memory algorithm, the distributed-memory employs the uniform subdivision approach to efficiently decompose the virtual world into a number of static space subdivisions. However, in contrast to shared-memory multiprocessors, all processors (which are hereafter referred to as *nodes*) participating in the matching process have their own private memory. Therefore, the task queue is no longer available for data sharing. As a result, each node must maintain a WU-node map, which contains the information of the space subdivisions that are currently being processed by the nodes.

Figure 6.1 illustrates an example of space subdivisions in two-dimensional space. In the figure, $Node_A$ is responsible for WU (0,2), (1,1), (1,2), and (2,2); $Node_B$ is responsible for WU (0,0 and (0,1); $Node_C$ is responsible for WU (1,0), (2,0), and (2,1).

At the initialisation stage of the distributed-memory algorithm, the WUs are evenly divided between the working nodes. Algorithm 6 describes the initial distribution of WUs in the two-dimensional space.

The regions are distributed to different nodes according to the space subdivisions they reside in. If a region lies in multiple space subdivisions that are owned by different nodes, it would be distributed to all of them. Again, we assume that the size of region is much smaller than a space subdivision. Therefore, a region would exist in at most four nodes for the two-dimensional space (at most eight nodes for the three-dimensional space).

The position and size of a region may be modified dynamically during simulation. Whenever a region is modified, its owner node is responsible for determining

Algorithm 6: Initial Distribution of WUs in Two-Dimensional Space

```
Data: $Z$: a two-dimensional table storing the space subdivisions
Data: $N_1$: number of sub-boundaries of the first dimension
Data: $N_2$: number of sub-boundaries of the second dimension
Data: $P$: number of nodes
begin
    $Count = \lfloor (N_1 \times N_2)/P \rfloor$;
    $Remainder = (N_1 \times N_2) \bmod P$;
    $CurrentNode = 0$;
    for $i \leftarrow 0$ to $N_1 - 1$ do
        for $j \leftarrow 0$ to $N_2 - 1$ do
            if $Count \neq 0$ then
                $Z[i, j].Node = CurrentNode$;
                $Count \leftarrow Count - 1$;
            end
            else if $Remainder \neq 0$ then
                $Z[i, j].Node = CurrentNode$;
                $Remainder \leftarrow Remainder - 1$;
                $CurrentNode \leftarrow CurrentNode + 1$;
                $Count = \lfloor (N_1 \times N_2)/P \rfloor$;
            end
            else
                $CurrentNode \leftarrow CurrentNode + 1$;
                $Count = \lfloor (N_1 \times N_2)/P \rfloor$;
            end
        end
    end
end
```

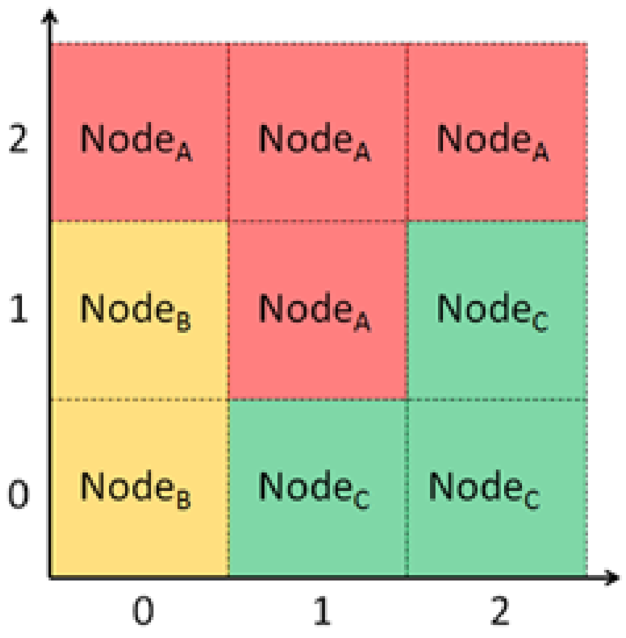


Figure 6.1: Space Decomposition in 2D

whether the region in question is entering a space subdivision that is owned by another node. This is done by hashing the vertices of the region with the hash function given in **Definition** 14. The hash value is an n-dimensional key which can be matched with the index of space subdivision, and therefore indicating which subdivision the vertex lies in.

6.2 Sorting and Matching

At the matching stage, each node processes all of the WUs that are assigned to it. If the WU has more than one region (i.e., hash table collision), Algorithm 1 would be applied to determine the overlap status of the regions.

Figure 6.2 illustrates an example of this process in two-dimensional space. In the figure, a node is responsible for processing the WU (0,0), (2,0), (1,1), (1,2), and (2,2), which are marked by dotted lines. The orthogonal projections of B-C and D-E overlap on x-axis; the orthogonal projections of C-D, C-E, and D-E overlap on y-axis; hence, the region pair D-E overlap in two-dimensional space.

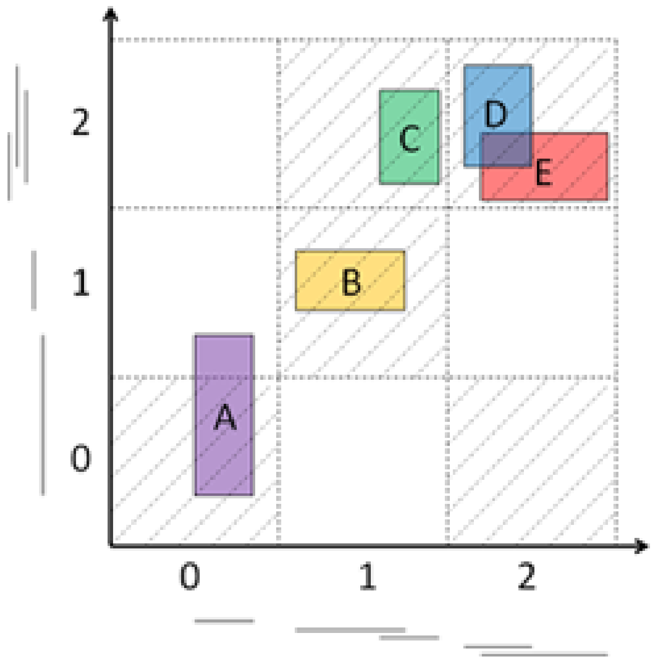


Figure 6.2: Dimension Reduction in Two-Dimensional Space

The shared-memory algorithm proposed in the previous chapter requires each WU maintains one endpoint list per dimension at the sorting stage. In contrast to this approach, in the distributed-memory algorithm the endpoint lists of different WUs are combined, as long as the WUs are owned by the same node. As a result, each node only maintains one endpoint list per dimension (for example, in Figure 6.2 only two endpoint lists in total are required for the node that owns WU (0,0), (2,0), (1,1), (1,2), and (2,2)). Combining the endpoint lists does not affect the order of the regions' vertices, due to the fact that the regions are all in the same space.

6.3 Load-Balancing

Due to the arbitrariness of the entities' movement pattern, the workload of the nodes may become uneven during runtime. Since the shared task queue is no longer available as a mean of load-balancing, a load-balancing scheme should be carried out to redistribute the workload, in order to maximise the utilisation of computational resources.

The workload of interest matching is defined by the elapsed time to process Algorithm 1. After the matching process, each node broadcasts its workload to all other nodes. The node with the heaviest load then carries out a load-balancing algorithm to redistribute its workload to another node. Two load-balancing algorithms have been developed, which are discussed in the following subsections.

6.3.1 Workload Redistribution (The Least Loaded Node)

The first load-balancing algorithm allows the most heavily loaded node to distribute a portion of its workload to the least loaded nodes in the simulation. This approach is given in Algorithm 7.

Algorithm 7: Workload Redistribution (The Least Loaded Node)

```
  Data: $WU$: a list of WU owned by the current node
  Data: $leastLoadedNode$: the least loaded node in the system
  Data: $\lambda$: the threshold of workload difference
1 begin
2     Sort $WU$ in descending order by the number of regions they contain;
3     if $(Load - leastLoadedNode.Load)/Load > \lambda$ then
4         $TransferOwnership(WU[1], leastLoadedNode)$;
5     end
6 end
```

The algorithm begins by sorting the list of WU in descending order by their crowdedness. If the workload difference between the current node and the least loaded node is bigger than a threshold λ, the algorithm transfers the ownership of the most crowded WU to the least loaded node. Note that the load-balancing process is performed in an adaptive manner that only the ownership of one WU would be transferred at each time-step.

6.3.2 Workload Redistribution (The Least Loaded Neighbour)

The second algorithm allows the heaviest loaded node to distribute a portion of its workload to a neighbour node. The term "neighbour node" is defined as the owner of an adjacent space subdivision. For example, in Figure 6.1, $Node_C$ is a neighbour node of WU (0,0).

Algorithm 8: Workload Redistribution Algorithm (The Least Loaded Neighbour

```
   Data: $WU$: a list of WU owned by the current node
   Data: $\lambda$: the threshold of workload difference
1  begin
2      Sort $WU$ in descending order by the number of regions they contain;
3      for $i \leftarrow 1$ to $Size(WU) - 1$ do
4          if $WU[i].HasNeighbourNode()$ then
5              $node \leftarrow WU[i].LeastLoadNeighbour()$;
6              if $(Load - leastLoadedNode.Load)/Load > \lambda$ then
7                  $TransferOwnership(WU[i], node)$;
8              end
9          end
10     end
11 end
```

Algorithm 8 begins by sorting the list of WU in descending order by their crowdedness. It then selects a crowded space subdivision (or WU). If the WU has a neighbour node, the algorithm calls the function $LeastLoadNeighbour()$ which returns a neighbour node with the lightest workload. Finally, if the workload difference between the current node and the neighbour node is bigger than a threshold λ, the algorithm transfers the ownership of the selected WU to the neighbour node by calling $TransferOwnership()$. Again, this load balancing process is performed in an adaptive manner that only the ownership of one WU would be transferred at each time-step.

The reason for only transferring the ownership of WU to a neighbour node is for reducing the communication overhead. When a region moves across a border that the WUs on both sides of the border are owned by different nodes, the two nodes should exchange messages for the region's migration. If the algorithm transfers the ownership of WU to arbitrary nodes, it might increase the chance of creating isolated WUs (i.e., all adjacent WUs are owned by different nodes), and therefore increase the chance of region migration. On the other hand, if the WU is transferred to a neighbour node, after the transfer there would be at least one adjacent WU is owned by the same node. This reduces the communication overhead of border crossing. The effectiveness of this approach and Algorithm 7 is evaluated in Section 6.6.

6.4 Conversion to Space-Time Interest Matching

Similar to the shared-memory algorithm, applying the distributed-memory algorithm to space-time interest matching is straightforward. All that needs to be changed is to replace regions with AASVs and replace Algorithm 1 with Algorithm 3. Therefore, hashing would be performed for the AASVs' vertices instead of the regions' and the sorting process would be carried out for the AASVs' projection endpoints instead of the regions'.

6.5 Application Scenarios

The communication architecture, as reviewed in Section 2.1.2, is an important component of a DVE. The proposed parallel algorithm is designed to be generic and suitable for all architectures. This section briefly describes some typical application scenarios.

6.5.1 Client-Server

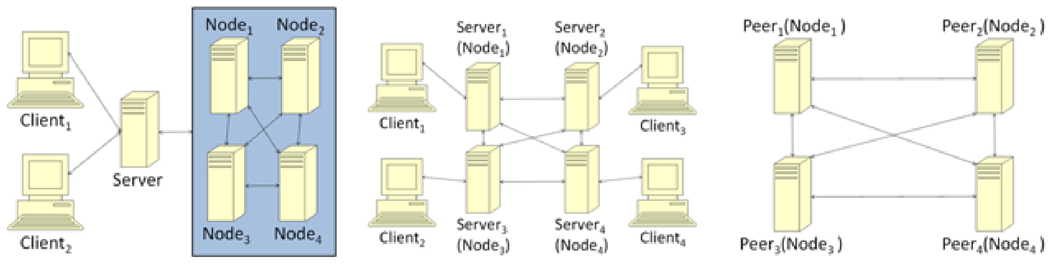
In this scenario, the master server has the most important role in the system. It is responsible for collecting participant actions from the clients, performing simulations, computing entity states, and sending updates to the clients. Figure 6.3(a) illustrates the client-server architecture. When applying the proposed algorithm, the nodes are used to assist the master server in the matching process. At each time-step, the master server would send the region modifications, if any, to the nodes. The nodes then send back the result set $\boldsymbol{RS}$ to the master server after the matching process. All ***RS*** would be merged on the master server to create a final result set of overlapped region pairs, which guides the master server through the client update process.

6.5.2 Multi-Server

Figure 6.3(b) illustrates the multi-server architecture. In this scenario, the virtual world is usually partitioned into a number of space subdivisions, and the workload of simulations and state updates are shared among a cluster of servers. The proposed algorithm can be fitted in this scenario if the virtual world adopts the same partition strategy as the proposed algorithm. In this case, each server also plays the role of a node. It is responsible to update the entities it owned as well as perform interest matching. Since regions are usually associated with entities, when an entity migrates to another server its regions usually follow. This reduces the communication overhead of region migration. In addition, after the matching process, the servers do not have to synchronise or merge the ***RS***. They can update the clients based on their own ***RS***.

6.5.3 Peer-to-Peer

Figure 6.3(c) illustrates the Peer-to-Peer (P2P) architecture. In a typical scenario, the virtual world is partitioned into a number of space subdivisions which are controlled by some of the peers (participants). Other peers are connected to them in order to receive updates. This is similar to the multi-server scenario with the exception that servers are also participants. The proposed algorithm can be fitted in this scenario by using the same model as multi-server architecture.



(a) Client-Server (b) Multi-Server (c) Peer-to-Peer

Figure 6.3: Communication Architectures

6.6 Performance Evaluation

This section presents an evaluation of the proposed parallel interest matching algorithm. Several sets of experiments were carried out to compare the performance of four approaches, namely:

1. Discrete interest matching by scalable insertion-sort algorithm (SDIM)
2. Space-time interest matching by scalable insertion-sort algorithm (SCIM)
3. Discrete interest matching by parallel algorithm (PDIM-DM)
4. Space-time interest matching by parallel algorithm (PCIM-DM)

The SDIM approach is the efficient sorting approach presented in Chapter 3. It was chosen as a comparison target because it is theoretically the fastest (linear time in the general case) algorithm among all serial algorithms. The SCIM approach is an implementation of Algorithm 3. It performs sorting to efficiently cull out the region pairs that are unlikely to overlap with each other, and carries out space-time matching for the remaining pairs. PDIM-DM and PCIM-DM are the two parallel algorithms presented in this chapter, which perform interest matching in a parallel manner where workload is shared by a cluster of nodes.

The PDIM-DM approach is designed for discrete interest matching, while the PCIM-DM approach performs space-time interest matching similar to the SCIM approach.

Evaluation of brute-force approaches is excluded since they have already been evaluated in the previous chapters. This evaluation only focuses on using the robust serial matching approaches as the comparison to the proposed parallel algorithms. Moreover, an evaluation of filtering precision and event-capturing ability is also excluded. Due to the facts that all four approaches have the same filtering precision, and the event-capturing ability of the space-time approaches has already been evaluated in Chapter 4. Therefore, this evaluation only focuses on the runtime efficiency of the four algorithms.

6.6.1 Implementation and Experimental Set-up

All four algorithms were implemented in C++ and Message Passing Interface (MPI). Message communication of the two parallel algorithms was constructed based on MPI protocols, such as $MPI\_Bcast()$, $MPI\_Send()$, and $MPI\_Recv()$. Moreover, all processes were synchronised by the $MPI\_Barrier()$ call, which is a simple lock-step synchronisation protocol.

All of the experiments were executed on the eScience Cluster at the Midland e-Science Centre, University of Birmingham. Each worker node has an Intel Xeon 3GHz processor with 2GB main memory. A Myrinet backplane is used to give 2+2Gbps programmable interconnection between the worker nodes. The experiments were conducted based on the following set-up:

- *Entity Distribution*: The virtual entities were initially distributed randomly across the virtual space.

- *Entity Speed*: The speed factor (SF) represents the average speed of the entities in proportion to its region length. The value of SF is set to 20 for all sets of experiments.
- *Number of Dimensions*: All simulations were performed in three-dimensional space.
- *Number of Regions* An update region and a subscription region were associated with each moving entity.
- *Execution Time Measurement*: Average execution time of the matching algorithms was measured over 10,000 time-steps.
- *Threshold of Space-Time Interest Matching*: The threshold τ of space-time interest matching was set to $\delta t/64$ for all sets of experiments.
- *Number of Nodes*: All experiments of the parallel algorithms, with the exception of those of Section 6.6.5, were run on 10 nodes.
- *Number of WUs*: The number of WUs is dependent on the granularity of spatial decomposition, which was assigned statically. The optimal granularity value was determined through experiments, which are presented in Section 6.6.2.

6.6.2 Granularity of Spatial Decomposition

Similar to Section 5.5.2, a set of experiments was conducted to determine the optimal granularity of partitioning, which would be used in all other experiments described in this chapter. We again take $N_1 = N_2 = N_3$, which implies that each subdivision is a cube in shape. We measured the execution time of the two parallel algorithms (PDIM-DM and PCIM-DM), with N_d extending from 2 to 10. The number of entities was set to constant (10000). Moreover, Algorithm 8 was used as the load-balancing algorithm, and the threshold of load difference λ was set to constant (10%).

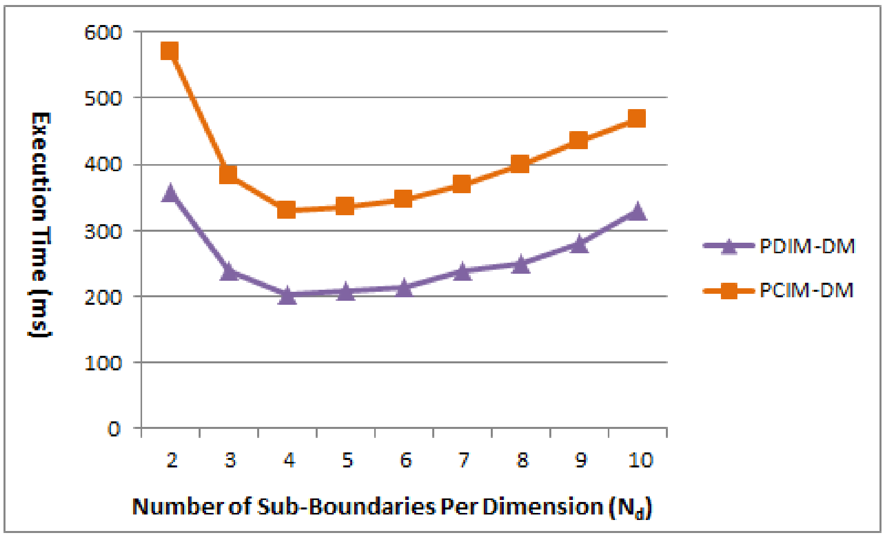


Figure 6.4: Comparing the Execution Time of Parallel Interest Matching Approaches (Number of Sub-Boundaries varies)

The results indicate that there is some significant runtime overhead when N_d is equal to 2. This is due to the fact that the number of WUs is less than the number of nodes, resulting in a poor utilisation of computational resources. However, if a large N_d is chosen, the number of WUs becomes large and the chance of region migration would be increased, and therefore more communication overhead would be introduced. According to results shown in Figure 5.3, we can conclude that the optimal value of N_d for both PDIM-DM and PCIM-DM is 4. This value would be used in all other experiments described in this chapter.

6.6.3 Random Entity Movements

One of the purposes of the evaluation is to investigate the performance of the parallel algorithms with different entity distribution models. In the second set of experiments, the entities were under random movements. The experiments were conducted to compare the runtime efficiency of the four approaches with the number of entities extending from 2000 to 10000. Since we also wanted to find out the message communication overhead of the parallel algorithms, the execution time without communication overhead (marked with \M) was recorded. Moreover, Algorithm 8 was used as the load-balancing algorithm, and the threshold of load difference λ was set to constant (10%).

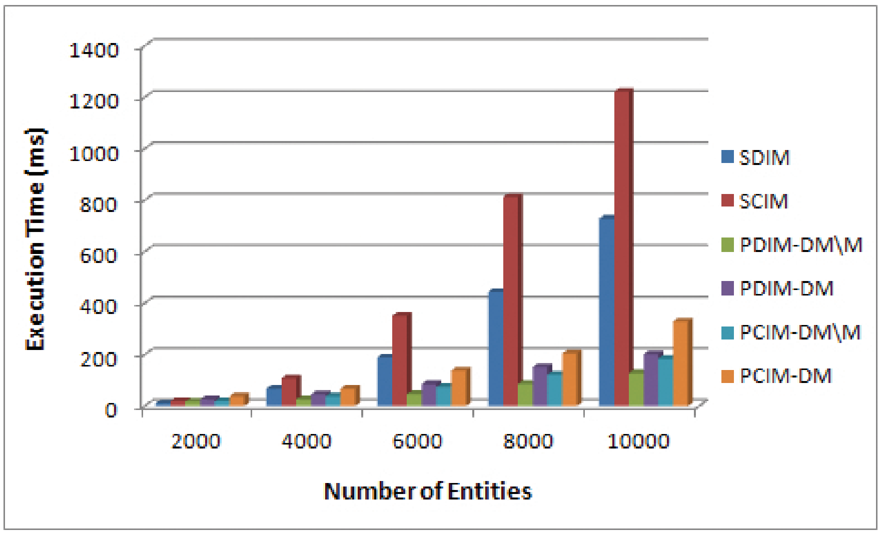


Figure 6.5: Comparing the Execution Time of Interest Matching Approaches (Random Entity Movements)

Figure 6.5 shows the execution time of the four approaches. The graph indicates that message communication of the parallel algorithms is a significant overhead as PDIM-DM requires about 30%-40% more execution time than PDIM-DM\M. The same observation also applies to PCIM-DM and PCIM-DM\M.

We can also see that when the number of entities is equal to 2000, the execution time of PDIM-DM is almost the same as SDIM. However, when the number of entities is equal to 4000, PDIM-DM starts outperforming SDIM; and their difference becomes significant when the number of entities is gradually increased. This suggests that the PDIM-DM approach, even with communication overhead, is more computationally scalable than SDIM when running on a cluster of nodes. Again, the same observation applies to the two space-time interest matching algorithms (i.e., PCIM-DM is more scalable than SCIM).

6.6.4 Skewed Environment

The third set of experiments aimed to investigate the performance of the matching algorithms under a load imbalance environment. In this environment, the entities always tend to move to the eight corners of the virtual world. After they reach a corner, they would stay for a period of time and then move to a random new corner. Similar to the previous set of experiments, we measured the execution time of the SDIM, SCIM, PDIM-DM, PCIM-DM, PDIM-DM\M, and PCIM-DM\M with the number of entities extending from 2000 to 10000. Algorithm 8 was again used as the load-balancing algorithm, and the threshold of load difference λ was set to constant (10%).

Figure 6.6 shows the execution time of the four approaches. The results are similar to the previous set of experiments, except the execution time spent on PDIM-DM, PCIM-CM, PDIM-DM\M, and PCIM-DM\M is slightly higher. This suggests that in a skewed environment, some of the nodes might be overloaded. The load imbalance environment causes poor utilisation of computational re-

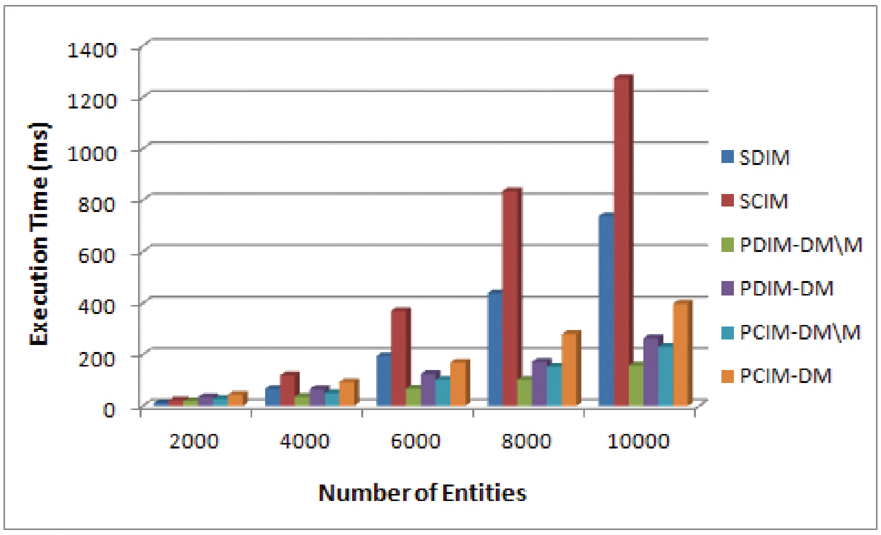


Figure 6.6: Comparing the Execution Time of Interest Matching Approaches (Skewed Environment)

sources, and the overloaded nodes have to redistribute the WUs they owned to some less loaded nodes. Therefore, the communication overhead of the parallel algorithms is also higher than the previous set of experiments. However, even with the extra overhead, the results clearly show that the parallel approaches are still more computationally scalable than the serial approaches.

6.6.5 Number of Processors

The fourth set of experiments compares the runtime efficiency of the four approaches when running on different number of processors (1-10). The number of entities was set to constant (10000). Algorithm 8 was again used as the load-balancing algorithm, and the threshold of load difference λ was set to constant (10%).

The results are shown in Figure 6.7. We can see from that graph that the execution time of the sorting approaches is constant as they are designed for serial

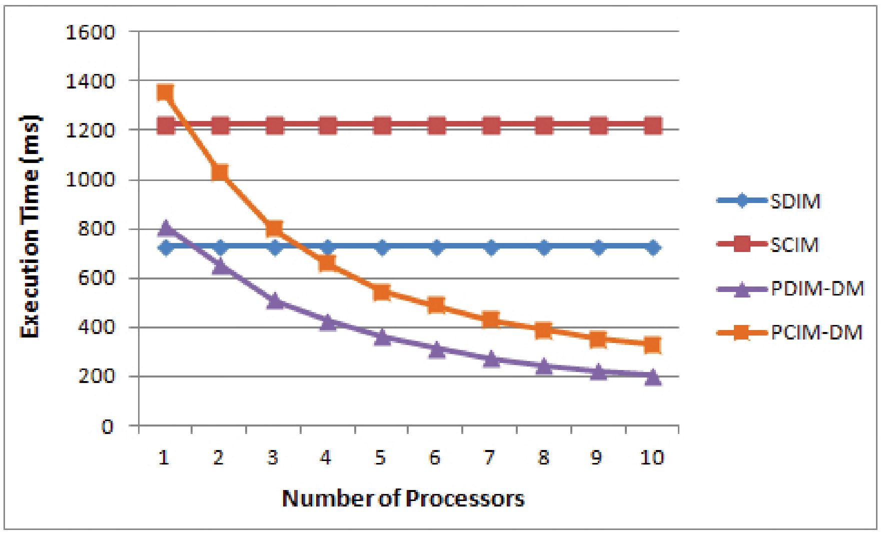


Figure 6.7: Comparing the Execution Time of Interest Matching Approaches (Number of Processors varies)

processing and are included only for reference. The execution time of the two parallel algorithms, on the other hand, decreased gradually with increase in the number of active processors. This suggests that the two parallel algorithms are scalable when running on a cluster of nodes. Nevertheless, similar to the shared-memory multiprocessors described in the previous chapter, the performance gain of PDIM-DM and PCIM-DM is not proportional to the number of processors. Their actual scalability of parallelism should be obtained through speed-up and efficiency analysis.

6.6.5.1 Analysis of Speed-up and Efficiency

The analysis of speed-up and efficiency is based on Lee's methodology described in Section 5.5.6.1. Let T_C, S_C and E_C denote the execution time, speed-up and efficiency of PCIM-DM, respectively; and let T_D, S_D and E_D denote the execution time, speed-up and efficiency of PDIM-DM, respectively. The results of the analysis are given in Table 6.1. Note that the problem size n is set to constant (10000).

p	$T_D(n,p)$	$S_D(n,p)$	$E_D(n,p)$	$T_C(n,p)$	$S_C(n,p)$	$E_C(n,p)$
1	810.53	1	100%	1352.18	1	100%
2	653.42	1.240	62.02%	1030.40	1.312	65.61%
3	512.25	1.582	52.74%	798.37	1.694	56.46%
4	425.13	1.907	47.66%	658.45	2.054	51.34%
5	364.18	2.226	44.51%	542.22	2.494	49.88%
6	315.24	2.571	42.85%	487.91	2.771	46.19%
7	275.94	2.937	41.96%	427.85	3.160	45.15%
8	244.18	3.319	41.49%	388.02	3.485	43.56%
9	221.57	3.658	40.65%	350.34	3.860	42.88%
10	202.99	3.993	39.93%	329.46	4.104	41.04%

Table 6.1: Speed-up and Efficiency of the Parallel Algorithms

The analysis shows that the efficiency of both PDIM-DM and PCIM-DM drops gradually to about 40% when the number of processors is increased to 10, and the change in efficiency becomes less apparent when more processors are involved in the experiments. If we compare PDIM-DM and PCIM-DM with the shared-memory algorithms presented in Chapter 5, we can see that the distributed-memory algorithms are about 10% less efficient than the shared-memory algorithms when the same number of processors is involved in the simulations. It is important to note that, however, this comparison is not entirely fair since the processing power, cache size, and memory bandwidth of the two testbeds (Intel E5645 and eScience Grid) are significantly different.

6.6.6 Load-Balancing

The fifth set of experiments was conducted to evaluate the two load-balancing algorithms proposed in Section 6.3. The first algorithm (LB1) is Algorithm 7, which allows the overloaded nodes to redistribute their WUs to the least loaded nodes in the system. The second algorithm (LB2) is Algorithm 8, which restricts the target of workload redistribution to the least loaded neighbours. To enable comparative analysis of the performance figures, an parallel algorithm without load-balancing (NoLB) is included. Specifically, this algorithm only carries out hashing and sorting. At its initialisation stage, an equal number of WUs is assigned to each node and the ownership of each WU would remain unchanged throughout the entire simulation process. In addition, to simplify the analysis, only space-time overlap test (i.e., Algorithm 3) was chosen for the sorting phase of the parallel algorithms. This is the due to the fact that, in terms of runtime efficiency, the only difference between discrete and space-time overlap tests is the computational overhead they incur, otherwise they have similar behaviour in the parallel interest matching framework.

Similar to the previous sets of experiments, we measured the execution time of PCIM-DM and PCIM-DM\M with the number of entities extending from 2000 to 10000. The threshold λ was again set to constant (10%). Lastly, since the experiments aimed to investigate the effectiveness of the load-balancing algorithms, performing simulations in a load imbalance environment would be more suitable than a random distribution environment. Therefore, the skewed environment described in the previous section was adopted for the simulations.

Figure 6.8 shows the results of the three approaches. The first observation to be made is that PCIM-DM (NoLB) is the slowest among all approaches we tested. Since PCIM-DM (NoLB) does not redistribute the workload even though some of the nodes are overloaded, there would be a significant difference in execution time between the nodes. Therefore, the overall performance of the matching algorithm

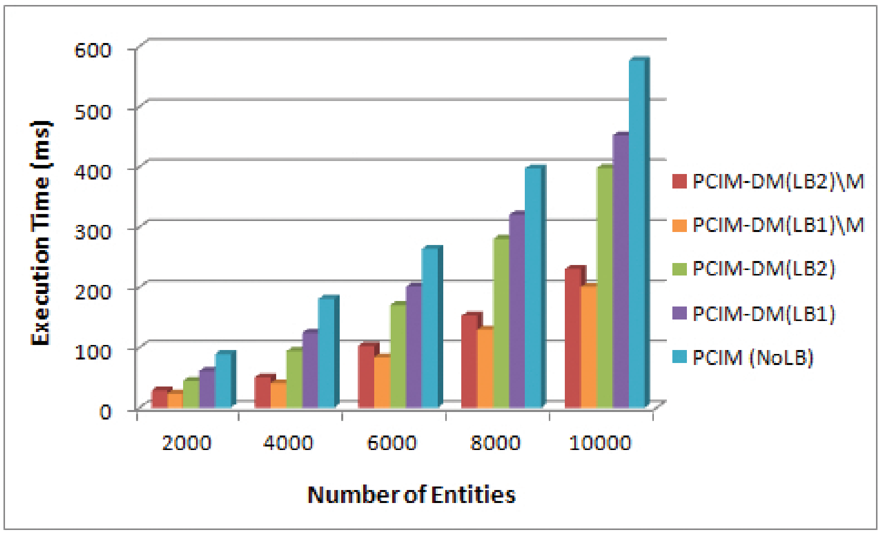


Figure 6.8: Workload Redistribution Algorithms

is degraded.

In addition, it is not difficult to see that PCIM-DM(LB1)\M is slighter faster than PCIM-DM(LB2)\M. Since the LB1 puts no restriction on how to redistribute the WUs, it would easier to balance the workload between the nodes than LB2. However, message communications of LB1 is a significant overhead since PCIM-DM(LB1) requires about 60% more execution time than PCIM-DM(LB1)\M. As a result, the total execution time of the PCIM-DM(LB1) is longer than the PCIM-DM(LB2). This overhead is caused by the increase of the chance of AASV migration. Since the LB1 transfers the ownership of WU to arbitrary nodes, after the simulation is performed for a period of time, a significant number of WUs become isolated (i.e., all adjacent WUs are owned by different nodes). This increases the chance of migration when the AASVs move from one space subdivision to another. As a result, the communication overhead of PCIM-DM(LB1) is also increased.

6.6.7 Threshold

The last set of experiments was conducted to investigate the performance of the two load-balancing algorithms with various threshold (λ) values. The execution time of PCIM-DM(LB1) and PCIM-DM(LB2) was measured with the threshold λ extending from 5% to 30%. The number of entities was set to constant (10,000). Lastly, since the experiments aimed to investigate the effectiveness load-balancing, the skewed environment was again adopted for the simulations.

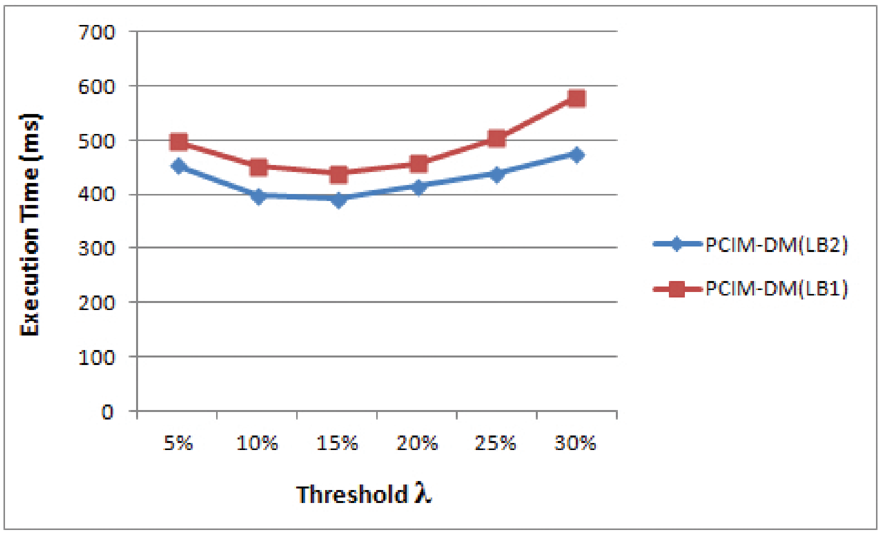


Figure 6.9: Execution Time of DIM (Threshold varies)

The experimental results are given in Figure 6.9. Before we discuss the figure, we must point out that the choice of threshold is in fact a trade-off and is application dependent. As we can see in the graph, there is some noticeable runtime overhead for both approaches when the value λ is less than 10% or larger than 20%. If a small threshold is chosen, the chance of load migration would be increased, and therefore more time would be lost to communication overhead. On the other hand, a large threshold makes the two algorithms more tolerant to workload imbalance, resulting in poor utilisation of computational resources. Hence we can conclude that, for the current experimental set-up, the optimal value of λ for both PCIM-DM(LB1) and PCIM-DM(LB2) is between 10% to 20%.

6.7 Summary

In the previous chapter, we have presented a parallel matching algorithm which is suitable to deploy on a shared-memory multiprocessors. In this chapter, we have presented an extension of the parallel algorithm that is suitable for distributed-memory systems. The proposed algorithm can be run on a cluster of computers to enable them to work simultaneously and thus enhance the overall runtime efficiency of the matching process. Due to the arbitrariness of the entities' movement pattern, the workload of the processors may become uneven during runtime. A load-balancing algorithm must be carried out to maximise the utilisation of the computational resources. However, unlike the shared-memory multiprocessors, in a distributed-memory system a shared task queue is no longer available as a mean of load-balancing. We thus have developed two algorithms (Algorithm 7 and 8) to assist workload redistribution.

As experimental evidence has suggested, using the load-balancing algorithms can better utilise the processors. In particular, we found out that Algorithm 8 has better overall performance since it only distribute the WU of the overloaded nodes to their neighbour nodes. In doing so it reduces the chance of creating isolated WUs, and therefore reducing the communication overhead incurred by the migration of regions or AASVs.

Chapter 7

Conclusions

Interest management is a technique which attempts to improve the scalability of large-scale DVE systems by selectively filtering the data which is sent in the system. This technique usually involves a process called interest matching, which determines what data should be sent to the participants of the DVE as well as what data should be filtered. Over the last two decades, interest management has been studied extensively in many fields such as military simulations, academic and social DVEs, and commercial applications such as MMOGs. Numerous schemes have been proposed for different application scenarios. The survey of interest management systems in Chapter 2 has led us to define three major design requirements, namely filtering precision, runtime efficiency, and event-capturing ability.

Based on the requirements analysis, it can be summarised that most of the interest matching approaches presented in the literature are designed to solve the trade-off between runtime efficiency and filtering precision. Although these approaches have been shown to meet their runtime performance requirements, they have a fundamental disadvantage - they perform interest matching at discrete time intervals. As a result, they would fail to report events between discrete time-steps when the virtual entities move rapidly or are small. Since DVE participants rely on the interest matching process to determine what data to receive, if missing interactions are ignored, the participants would most likely render incorrect scenes or perform incorrect simulations. This thesis has presented theoretical and practical solutions to this "missing event" problem. The new approach, as presented in Chapter 4, is called "space-time interest matching". It aims to capture the missing events by using swept volumes to enclose the trajectory of the update and subscription regions over a time interval. Although this approach requires additional effort to compute the swept volumes, we employ an efficient insertion sort algorithm, as presented in Chapter 3, to cull out the swept volume pairs that are unlikely to overlap with each other before a more computationally intensive pairwise matching process is carried out. Experimental evidence presented in Chapter 4 demonstrates that the sorting algorithm significantly reduces the overhead of space-time interest matching. It also shows that the space-time algorithm can capture most of the events that are ignored by the discrete algorithms. Most importantly, it demonstrates that even when we increase the frequency of discrete interest matching, it is still outperformed by the space-time algorithm in terms of runtime efficiency and event-capturing ability.

Another problem of existing interest matching approaches is that they are designed for serial processing which is supposed to be run on a single processor. As the problem size grows, using these algorithms does not satisfy the scalability requirement of DVE since the single processor may eventually become a bottleneck. On the other hand, as shared-memory multiprocessors are common today, most of the commercial DVE applications such as MMOGs employ dual-core or quad-core machines as severs. If existing algorithms are deployed directly on these machines, the performance gain would not be guaranteed. To tackle this problem, this thesis has presented two parallel interest matching algorithms. The first algorithm, as presented in Chapter 5, is designed for shared-memory multiprocessors such as multi-core CPUs; while the second algorithm, as presented in Chapter 6, is designed for distributed-memory systems. Both algorithms enable the processors to work simultaneously and thus enhance the overall runtime efficiency of the matching process. Experimental evidence has demonstrated that the parallel algorithms are much faster than the existing serial algorithm when running on multiprocessors.

Furthermore, this thesis has also shown that the space-time interest matching approach can be easily integrated into the parallel framework. The integration is simple and straightforward, which further minimises the computational overhead of space-time interest matching. The experimental results presented in the thesis lead to three important conclusions. Firstly, the combined algorithms have a very good filtering precision which is similar to all aura-based matching algorithms. Secondly, the proposed algorithms significantly enhance the runtime efficiency of space-time interest matching when running on shared-memory multiprocessors or distributed-memory systems. Since it is increasingly common to deploy commercial DVE applications such as MMOGs on the former, using the combined algorithm for these applications is more suitable than the serial algorithms. Finally, the results show that the combined algorithm can capture most of the events that are ignored by the discrete algorithms. Hence, it can be concluded that the combined algorithm satisfies all three major design requirements of interest management.

7.1 Future Work

This Section presents potential further development into the algorithms and systems covered by this thesis.

7.1.1 Real-Life Evaluation

The experimental evaluation of the interest matching algorithms presented in this thesis was conducted through simulations. Two approaches, namely random entity movement and skewed environment, have been used to model the distribution and behaviour of the entities. We have also used various speed factors to model the speed of entity movement. However, these approaches may not reflect the real-life behaviour of the participants. A "real-life behaviour" can only be obtained through recording the interactions of the participants in the DVE. It is worth noting that there is no standard accepted metric with which to characterise these interactions, since the behaviour of the entities and the participants is always application dependent. However, by getting the traces from certain types of large-scale applications such as MMOGs, a real-life evaluation would still be worth the effort and can be considered as a proof of concept experiment for the proposed algorithms.

7.1.2 Large-Scale Multi-Core Processor

Deploying the parallel algorithm to multi-core computers has a number of advantages. Firstly, the implementation is simple, synchronisation, communication, and explicit load-balancing algorithms are not required. Secondly, the communication overhead is small since the processors can communicate through the shared-memory. In addition, using the multi-core processors for commercial DVE applications such as MMOGs is increasingly common. As examples, the video game console Sony PlayStation 3 uses a IBM Cell processor which contains 9 cores while Microsoft's Xbox 360 uses a Xenon processor which is a custom PowerPC-based triple-core. This increases the potential applications of the parallel interest matching algorithm.

As we have discussed in Section 5.5.6.1, the analysis on the speed-up and efficiency of the parallel algorithm is limited by the double 6-core testbed. Further experiments on a large-scale multi-core processor should be considered as the future work of this research. However, at the time of writing, only six core processors are available in the PC market. Therefore an available experimental machine may be used as the platform to carry out further evaluation of the proposed algorithms. One potential testbed is the Single-Chip Cloud Computer (SCC) [107], which is an experimental machine developed by Intel Labs. It contains the most Intel cores ever integrated on a single chip - 48 cores. This would give us a better analysis on the scalability of parallelism. Moreover, another problem we faced in the evaluation is that we were unable to fairly compare the efficiency of the two parallel algorithms proposed in Chapter 5 and 6 since the processing power, cache size, and memory bandwidth of the two testbeds (Intel E5645 and eScience Grid) are significantly different. This may be solved by using SCC as the testbed, since apart from being used as a shared-memory multi-core processor, The SCC can also be configured as a distributed-memory system, with each core able to access up to 2GB private memory.

7.1.3 Dead Reckoning

Dead reckoning [63] is a technique used in DVEs. Similar to interest management, it decreases message transmissions in the network and thus increasing the scalability of the DVE system. The basic idea of the dead reckoning is that instead of receiving frequent state updates throughout the network, each participant in the DVE stores a mathematical model to predict the new position of the remote entities. The extrapolation is based on the entities' velocity, acceleration, and previous position.

Synchronising the dead-reckoned entities is a well-studied problem of DVE, but it would become more complex if this technique is applied together with interest management. Performing interest matching for dead-reckoned entities but not for the actual entities is possible in a DVE. This is the case where the participant updates an avatar's position on a local machine but does not update its corresponding dead-reckoned model on the remote machines. Other participants may decide that the dead-reckoned avatar's update region overlaps with a subscription region and send a message to the network indicating the result of this event. However, the real avatar is at a slightly different actual position, depending on the error tolerance in the dead reckoning algorithm, the actual position of its update region may not overlap with any subscription region. Therefore, solving the problem of dead-reckoned regions is a potential future continuation of this research.

Appendix A

Golden Section Search for Minimisation or Maximisation of Functions

For the sake of completeness and to provide the reader of this thesis easy access to the relevant information, this appendix provides a detailed description of Golden Section Search, which is an excerpt from the book Numerical Recipes [101].

Golden Section Search is a simple numerical method to find the extrema of function in a finite neighbourhood. Consider a function $f(t)$ and a boundary (a, b), if we want to find the local minimum of $f(t)$, we can apply a searching method by choosing a point x, such that$a < x < b$. If $f(x)$ is less than both $f(a)$ and $f(b)$, the function has a local minimum in $a, b)$. We then choose a new point y, if $f(x) < f(y)$, then the new bracketing triplet of points is (a, x, y); otherwise if $f(x) > f(y)$, then the new bracketing triplet is (x, y, b). This process continues until the distance between the two outer points of the triplet is small than a threshold.

The strategy for choosing the new point y is important in this method. Given (a, x, b), suppose that x is a fraction w of the way between a and b, i.e.

$$\frac{x-a}{b-a} = w \quad \frac{b-x}{b-a} = 1-w. \tag{A.1}$$

Also suppose that the next trial point y is an additional fraction z beyond x,

$$\frac{y-x}{b-a} = z \tag{A.2}$$

Then the next bracketing segment will either be of length $w + z$ relative to the current one, or else of length $1 - w$. If we want to minimise the worst case possibility, then we will choose z to make these equal, namely

$$z = 1 - 2w \tag{A.3}$$

The new point is the symmetric point to x in the original interval, namely with $|x - a|$ equal to $|y - b|$. Therefore y lies in the larger of the two segments (z is positive only if $w < 0.5$).

If z is chosen to be optimal, then so was w before it. This scale similarity implies that y should be the same fraction of the way from x to c (if that is the bigger segment) as was x from a to b, in other words,

$$\frac{z}{1-w} = w \tag{A.4}$$

Equation A.3 and A.4 give the quadratic euqation

$$w^2 - 3w + 1 = 0 \quad \textit{yielding} \quad w = \frac{3 - \sqrt{5}}{2} \approx 0.38197 \tag{A.5}$$

In other words, the optimal bracketing interval (a, x, b) has its middle point x a fractional distance 0.38197 from one end, and 0.61803 from the other end. These fractions are those of the so-called golden mean or golden section. This optimal method of function minimization, the analog of the bisection method for finding zeros, is thus called the golden section search, summarized as follows:

Given, at each stage, a bracketing triplet of points, the next point to be tried is that which is a fraction 0.38197 into the larger of the two intervals (measuring from the central point of the triplet). If you start out with a bracketing triplet whose segments are not in the golden ratios, the procedure of choosing successive points at the golden mean point of the larger segment will quickly converge you to the proper, self-replicating ratios. The golden section search guarantees that each new function evaluation will (after self-replicating ratios have been achieved) bracket the minimum to an interval just 0.61803 times the size of the preceding interval.

References

[1] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. A Review on Networking and Multiplayer Computer Games. Technical Report 454, Turku Centre for Computer Science, April 2002. 1

[2] Elvie S. Liu, M. K. Yip, and G. Yu. Lucid Platform: Applying HLA DDM to Multiplayer Online Game Middleware. *ACM Computers in Entertainment*, 4(4):9, 2006. 4, 51, 55, 64, 66, 71, 118

[3] Elvis S. Liu and Georgios K. Theodoropoulos. An Approach for Parallel Interest Matching in Distributed Virtual Environments. In *Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009)*, October 2009. 6, 55

[4] Elvis S. Liu and Georgios Theodoropoulos. A Continuous Matching Algorithm for Interest Management in Distributed Virtual Environments. In *Proceedings of the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2010)*, May 2010. 6

[5] Elvis S. Liu and Georgios Theodoropoulos. Space-Time Matching Algorithms for Interest Management in Distributed Virtual Environments. Manuscript submitted for publication, December 2011. 6

[6] Elvis S. Liu and Georgios Theodoropoulos. A Fast Parallel Matching Algorithm for Continuous Interest Management. In *Proceedings of Winter Simulation Conference (WSC) 2010*, December 2010. 6

[7] Elvis S. Liu and Georgios K. Theodoropoulos. A Parallel Interest Matching Algorithm for Distributed-Memory Systems. In *Proceedings of the 15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2011)*, September 2011. 7

[8] Elvis S. Liu and Georgios Theodoropoulos. Interest Management for Distributed Virtual Environments: A Survey. Manuscript submitted for publication, December 2011. 7

[9] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999. 8, 9

[10] Anthony Steed and Manuel Fradinho Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Morgan Kaufmann, 2010. 8

[11] Silvia Rueda, Pedro Morillo, and Juan M. Orduna. A Peer-To-Peer Platform for Simulating Distributed Virtual Environments. In *Proceedings of the 13th International Conference on Parallel and Distributed Systems - Volume 02*, pages 1–8, 2007. 12

[12] BjLorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *Proceedings of the Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, March 2004. 12, 18

[13] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. Colyseus: A Distributed Architecture for Online Multiplayer Games. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 12–12, 2006. 12

[14] Richard Allan Bartle. *Designing Virtual Worlds*. New Riders Games, 2003. 12

[15] James Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller, and Dan Owen. The SIMNET virtual world architecture. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, VRAIS '93, pages 450 – 455, September 1993. 12

[16] David L. Neyland. *Virtual Combat: A Guide to Distributed Interactive Simulation*. Stackpole Books, 1997. 13

[17] Sudhir Srinivasan and Bronis R. De Supinski. Multicasting in DIS: A Unified Solution. In *Proceedings of the 1995 Electronic Conference on Scalability in Training Simulation (ELECSIM 1995)*, April-June 1995. 13, 24

[18] Joe Sorroche and Jerry Szulinski. Bandwidth reduction techniques used in DIS exercises. In *Proceedings of the 2004 European Simulation Interoperability Workshop (EURO-SIW)*, June 2004. 13, 42

[19] M. Bassiouni, H. Williams, and M. Loper. Intelligent filtering algorithms for networked simulators. In *Proceedings of 1991 IEEE International Conference on Systems, Man, and Cybernetics. 'Decision Aiding for Complex Systems'*, October 1991. 13, 37, 41

[20] Christer Carlsson and Olof Hagsand. DIVE - A platform for multi-user virtual environments. *Computers & Graphics*, 17:663–669, 1993. 13, 37

[21] C. Greenhalgh and S. Benford. MASSIVE: A Collaborative Virtual Environment for Teleconferencing. *ACM Transactions on Computer Human Interactions*, 2(3):239–261, September 1995. 13, 18, 37

[22] DMSO. High Level Architecture Interface Specification Version 1.3, 1998. 13, 20, 40

[23] K. Morse and M. Petty. High Level Architecture Data Distribution Management migration from DoD 1.3 to IEEE 1516. *Concurrency and Computation: Practice and Experience*, 16(15):1–17, 2004. 13

[24] Sony Online Entertainment. Everquest, 1999. http://everquest.station.sony.com/. 14, 18, 47

[25] Square Enix. Final Fantasy XI, 2002. http://www.playonline.com/ff11us/. 14, 18, 47

[26] Blizzard Entertainment. World of Warcraft, 2004. http://www.worldofwarcraft.com/. 14, 18

[27] NCsoft. GuildWars, 2005. http://www.guildwars.com/. 14, 18, 47

[28] Linden Lab. Second Life, 2003. http://secondlife.com/. 14, 46

[29] Pavel Curtis and David A. Nichols. MUDs Grow Up: Social Virtual Reality in the Real World. In *Proceedings of the Thirty-Ninth IEEE Computer Society International Conference (COMPCON Spring '94)*, pages 193–200, February 1994. 18

[30] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Middleware '01, pages 329–350, London, UK, 2001. Springer-Verlag. 18

[31] Square Enix. Final Fantasy XIV, 2010. http://www.finalfantasyxiv.com/. 18

[32] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Paul T. Barham, and Steven Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. *Presence*, 3:265–287, 1994. 19

[33] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments. In *Proceedings of the 1995 IEEE Virtual Reality Annual International Symposium (VRAIS '95)*, pages 2–10, 1995. 19, 26

[34] Gary Tan, Rassul Ayani, YuSong Zhang, and Farshad Moradi. Grid-Based Data Management in Distributed Simulation. In *Proceedings of the 33rd Annual Simulation Symposium (SS 2000)*, Washington, DC , USA, April 2000. IEEE Computer Society. 20, 36

[35] Steven J. Rak and Daniel J. Van Hook. Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment. In *Proceedings of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pages 739–747, March 1996. 21, 23, 36

[36] Carlos Eduardo Bezerra, Fábio R. Cecin, and Cláudio F. R. Geyer. A3: A Novel Interest Management Algorithm for Distributed Simulations of MMOGs. In *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2008)*, pages 35–42. IEEE Computer Society, 2008. 21, 22

[37] D. J. Van Hook, S. J. Rak, and J. O. Calvin. Approaches to Relevance Filtering. In *Proceedings of the 11th Workshop on Standards for the Interoperability of Distributed Simulations*, pages 26–30, 1994. 23, 41, 50

[38] Joshua E. Smith, Kevin L. Russo, and Lawrence C. Schuette. Prototype Multicast IP Implementation in ModSAF. In *Proceedings of the 12th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, pages 175–178, March 1995. 23, 24

[39] Chia-Hao Liu, Chen-Hsing Wen, and Hsing-Lung Chen. Tracking-needless grouping: an efficient and scalable grouping scheme in networked virtual environments. In *Proceedings of the 1st IEEE Consumer Communications and Networking Conference (CCNC 2004)*, pages 477–482, January 2004. 25

[40] Zhengjun Zhai, Xiaomei Hu, and Xiaobin Cai. An Adaptive Grouping Scheme in Collaborative Virtual Environment Systems. In *Proceedings of the 2005 International Conference on Cyberworlds (CW'05)*, pages 311–315, November 2005. 25

[41] K. Prasetya and Z. D. Wu. Performance Analysis of Game world Partitioning Methods for Multiplayer Mobile Gaming. In *Proceedings of the 7th*

ACM SIGCOMM Workshop on Network and System Support for Games (Netgames 2008), pages 72–77, New York, NY, USA, 2008. ACM. 26, 27

[42] Anthony (Peiqun) Yu and Son T. Vuong. MOPAR: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 99–104, 2005. 26

[43] Yaping Lu, Yunjia Wang, and Houquan Liu. An Interest Management Architecture by ALM and Region Partition for Large-Scale Distributed Virtual Environment. *Journal of Computers*, 5(6):836–843, June 2010. 27

[44] Wolfgang Broll. Distributed virtual reality for everyone - a framework for networked VR on the Internet. In *Proceedings of IEEE 1997 Virtual Reality Annual International Symposium (VRAIS 1997)*, March 1997. 29

[45] John W. Barrus, Richard C. Waters, and David B. Anderson. Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments. In *Proceedings of IEEE 1996 Annual International Symposium on Virtual Reality (VRAIS '96)*, pages 204–213, 1996. 29

[46] Diamond Park, R. Waters, D. Anderson, J. Barrus, D. Brogan, S. Mckeown, T. Nitta, I. Sterns, and W. Yerazunis. Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability. Technical report, MERL - A Mitsubishi Electric Research Laboratory, 1996. 29

[47] Bruce Naylor, John Amanatides, and William Thibault. Merging BSP Trees Yields Polyhedral Set Operations. In *Proceedings of the 17th International Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 115–124, 1990. 29

[48] Anthony Steed and Roula Abou-Haidar. Partitioning Crowded Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST) 2003*, pages 7–14, 2003. 30, 33, 36

[49] Simon Rieche, Klaus Wehrle, Marc Fouquet, Heiko Niedermayer, Leo Petrak, and Georg Carle. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. In *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, pages 763–767, January 2007. 31, 32

[50] D. J. Van Hook, S. J. Rak, and J. Calvin. Approaches to RTI Implementation of HLA Data Distribution Management Services. In *Proceedings of the 15th Workshop on Standards for the Interoperability of Distributed Simulations*, 1997. 33, 36, 50

[51] Xiang-bin Shi, Yue Wang, Qiang Li, Ling Du, and Fang Liu. An Interest Management Mechanism Based on N-Tree. In *Proceedings of the Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD '08)*, pages 917–922. IEEE Computer Society, August 2008. 33

[52] Shun-Yun Hu, Jui-Fa Chen, and Tsu-Han Chen. VON: A Scalable Peer-

to-Peer Network for Virtual Environments. *IEEE Network*, 20:22–31, 2006. 34, 49

[53] Eliya Buyukkaya and Maha Abdallah. Data Management in Voronoi-Based P2P Gaming. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC 2008)*, pages 1050–1053, January 2008. 35

[54] G. Deen, M. Hammer, J. Bethencourt, I. Eiron, J. Thomas, and J. H. Kaufman. Running Quake II on a grid. *IBM Systems Journal*, 45:21–44, January 2006. 36

[55] Rassul Ayani, Farshad Moradi, and Gary Tan. Optimizing cell-size in grid-based DDM. In *Proceedings of the fourteenth workshop on Parallel and distributed simulation*, PADS '00, pages 93–100. IEEE Computer Society, 2000. 36

[56] Lennart E. Fahlén and Charles G. Brown. The use of a 3D aura metaphor for computer based conferencing and teleworking. In *Proceedings of the 4th MultiG Workshop*, Stockholm, May 1992. 37

[57] Li Zou and Mostafa H. Ammar Christophe Diot. An Evaluation of Grouping Techniques for State Dissemination in Networked Multi-User Games. In *Proceedings of the ninth Internation Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 33–40, 2001. 37

[58] Steve Benford, Adrian Bullock, Neil Cook, Paul Harvey, Rob Ingram, and

Ok-Ki Lee. From Rooms to Cyberspace: Models of Interaction in Large Virtual Computer Spaces. *Interacting with Computers*, 5:217–237, 1993. 37

[59] Steve Benford and Chris Greenhalgh. Introducing Third Party Objects into the Spatial Model of Interaction. In *ECSCW'97: Proceedings of the fifth conference on European Conference on Computer-Supported Cooperative Work*, pages 189–204. Kluwer Academic Publishers, 1997. 39

[60] Michal Masa and Jiří Žára. Generalized interest management in virtual environments. In *Proceedings of the 4th international conference on Collaborative virtual environments*, CVE '02, pages 149–150. ACM, 2002. 39

[61] Michal Masa and Jiří Žára. Interest Management for Collaborative Environments Through Dividing Their Shared State. In *Proceedings of Cooperative Design, Visualization, and Engineering International Conference (CDVE 2004)*, 2004. 39

[62] Dawei Ding and Miaoling Zhu. A model of dynamic interest management: interaction analysis in collaborative virtual environment. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '03, pages 223–230. ACM, 2003. 39

[63] Richard M. Fujimoto. *Parallel and Distributed Simlation Systems*. John Wiley and Sons, Inc., 2000. 40, 52, 160

[64] Robert Bartlett. Interest Operators: Facilitating Attribute Interest Criteria for Formula-Based Interest Management in Distributed Virtual Environments. In *Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles*

of Advanced and Distributed Simulation (PADS 2006), pages 111–118, May 2006. 40

[65] Steve Benford, John Bowers, Lennart E. Fahlen, and Chris Greenhalgh. Managing Mutual Awareness in Collaborative Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST) 1994*, pages 223–236, 1994. 41

[66] James Purbrick and Chris Greenhalgh. Extending Locales: Awareness Management in MASSIVE-3. In *Proceedings of the IEEE Virtual Reality 2000 Conference*, page 287, Washington, DC, USA, 2000. IEEE Computer Society. 41

[67] Howard Abrams, Kent Watsen, and Michael Zyda. Three-Tiered Interest Management for Large-Scale Virtual Environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST) 1998*, pages 125–129, 1998. 41

[68] A Boukerche, N.J. McGraw, and R.B. Araujo. A Grid-Filtered Region-Based Approach to Support Synchronization in Large-Scale Distributed Interactive Virtual Environments. In *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, ICPPW '05, pages 525–530, Washington, DC, USA, 2005. IEEE Computer Society. 41

[69] Ke Pan, Wentong Cai, Xueyan Tang, Suiping Zhou, and Stephen John Turner. A Hybrid Interest Management Mechanism for Peer-to-Peer Networked Virtual Environments. In *Proceedings of the 24th IEEE Interna-*

tional Parallel and Distributed Processing Symposium (IPDPS'10), April 2010. 41

[70] Azzedine Boukerche, Amber J. Roy, and Neville Thomas. Dynamic Grid-Based Multicast Group Assignment in Data Distribution Management. In *Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2000)*, pages 47–. IEEE Computer Society, 2000. 42

[71] Gary Tan, YuSong Zhang, and Rassul Ayani. A Hybrid Approach to Data Distribution Management. In *Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2000)*, pages 55–. IEEE Computer Society, 2000. 42

[72] Azzedine Boukerche and Amber J. Roy. Dynamic grid-based approach to data distribution management. *Journal of Parallel and Distributed Computing*, 62:366–392, March 2002. 42

[73] ChangHoon Park, Koichi Hirota, Michitaka Hirose, and Heedong Ko. A Flexible and Efficient Scheme for Interest Management in HLA. In *Proceedings of Asian Simulation Conference (AsiaSim 2004)*, pages 141–149, October 2004. 42

[74] Azzedine Boukerche and Kaiyuan Lu. Optimized Dynamic Grid-Based DDM Protocol for Large-Scale Distributed Simulation Systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, April 2005. 42

[75] Larry Mellon. Hierarchical Filtering in the STOW System. In *Proceedings of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, March 1996. 42

[76] Steven J. Rak, Marnie Salisbury, and Robert S. Macdonald. HLA/RTI data distribution management in the synthetic theater of war. In *Proceedings of 1997 Fall Simulation Interoperability Workshop (SIW)*, September 1997. 42

[77] Rob Minson and Georgios Theodoropoulos. An Adaptive Interest Management Scheme for Distributed Virtual Environments. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, PADS '05, pages 273–281. IEEE Computer Society, 2005. 43

[78] Rob Minson and Georgios Theodoropoulos. An Evaluation of Push-Pull Algorithms in Support of Cell-Based Interest Management. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '07, pages 39–47. IEEE Computer Society, 2007. 43

[79] Rob Minson and Georgios Theodoropoulos. Load Skew in Cell-Based Interest Management Systems. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 43–50. IEEE Computer Society, 2008. 43

[80] Dieter Schmalstieg and Michael Gervautz. Demand-Driven Geometry Transmission for Distributed Virtual Environments. *Computer Graphics Forum*, 5(3):421–433, 1996. 45

[81] Eyal Teler and Dani Lischinski. Streaming of Complex 3D Scenes for Remote Walkthroughs. *Eurographics*, 20(3):17–25, 2001. 45

[82] Jimmy Chim, Rynson W. H. Lau, Hong Va Leong, and Antonio Si. CyberWalk: A Web-Based Distributed Virtual Walkthrough environment. *IEEE Transactions on Multimedia*, 5:503–515, 2003. 45

[83] Philip Rosedale and Cory Ondrejka. Enabling Player-Created Online Worlds with Grid Computing and Streaming. *Gamasutra Resource Guide*, 2003. Also available as http://www.gamasutra.com/resource_guide/20030916/rosedale_pfv.htm. 46

[84] Jeff Strain and Dan Adams. The Tech of Guild Wars. *IGN*, 2005. Also available as http://uk.pc.ign.com/articles/534/534454p1.html. 47

[85] Jean Botev, Er Hohfeld, Hermann Schloss, Ingo Scholtes, and Markus Esch. The HyperVerse - Concepts for a Federated and Torrent-based "3D Web". In *Proceedings of the 1st International Workshop on Massively Multiuser Virtual Environments (MMVE)*, 2008. 47

[86] Shun-Yun Hu, Ting-Hao Huang, Shao-Chen Chang, Wei-Lun Sung, Jehn-Ruey Jiang, and Bing-Yu Chen. FLoD: A Framework for Peer-to-Peer 3D Streaming. In *Proceedings of the 27th IEEE International Conference on Computer Communications (INFOCOM 2008)*, pages 1373–1381, April 2008. 48

[87] Shun-Yun Hu, Jehn-Ruey Jiang, and Bing-Yu Chen. Peer-to-Peer 3D Streaming. *IEEE Internet Computing*, 14:54–61, 2010. 48

[88] M. Petty and K. Morse. Computational complexity of HLA data distribution management. In *Proceedings of the Fall Simulation Interoperability Workshop*, 2000. 50

[89] Azzedine Boukerche and Caron Dzermajko. Performance Comparison of Data Distribution Management Strategies. In *Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2001)*, page 67, Washington, DC, USA, 2001. IEEE Computer Society. 50

[90] M. Petty and K. Morse. The Computational Complexity of the High Level Architecture Data Distribution Management Matching and Connecting Processes. *Simulation Modelling Practice and Theory*, 12:217–237, 2004. 50

[91] G. Morgan, K. Storey, and F. Lu. Expanding Spheres: A Collision Detection Algorithm for Interest Management in Networked Games. In *Proceedings of the Entertainment Computing - ICEC 2004*, 2004. 50

[92] C. Raczy, G. Tan, and J. Yu. A Sort-Based DDM Matching Algorithm for HLA. *ACM Transactions on Modeling and Computer Simulation*, 15(1):14–38, 2005. 51, 55, 64, 118

[93] Elvis S. Liu, Milo K. Yip, and Gino Yu. Scalable Interest Management for Multidimensional Routing Space. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST) 2005*, pages 82–85, 2005. 51, 55, 61, 64, 66, 118

[94] Ke Pan, Stephen John Turner, Wentong Cai, and Zengxiang Li. An Efficient Sort-Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment. In *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS 2007)*, pages 70–82, Washington, DC, USA, 2007. 51, 55, 64, 66, 118

[95] Graham Morgan and Fengyun Lu. Predictive Interest Management: An Approach to Managing Message Dissemination for Distributed Virtual Environments. In *Proceedings of the First International Workshop on Interactive Rich Media Content Production: Architectures, Technologies, Applications, Tools 2003*, 2003. 53

[96] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies.* PhD thesis, Department of Computer Science, Ithaca, NY, USA, 1992. 61

[97] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, 1995. 64

[98] Karim Abdel-Malek, Denis Blackmore, and Kenneth Joy. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling*, 12(1):87–127, 2006. 73, 76

[99] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast Proximity Queries with Swept Sphere Volumes. Technical report, Department of Computer Science, UNC Chapel Hill, 1999. 73

[100] Stephen Cameron. A Study of the Clash Detection Problem in Robotics. In *Int. Conf. Robotics and Automation*, 1985. 73, 74

[101] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3 edition, 2007. 87, 161

[102] M.O. Rabin. Probabilistic algorithms. In *Algorithms and Complexity: New Direction and Recent Results*, pages 21–39. Academic Press Inc, New York, 1976. 108

[103] Jon Louis Bentley and Jerome H. Friedman. Data Structures for Range Searching. *ACM Computing Survey*, 11:397–409, December 1979. 108

[104] Sivarama P. Dandamudi and Philip S. P. Cheng. A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(1):1–16, 1995. 117

[105] Ruby Bei-Loh Lee. Empirical Results on the Speed, Efficiency, Redundancy and Quality of Parallel Computations. In *Proceedings of the 1980 International Conference on Parallel Processing*. IEEE Computer Society, August 1980. 128

[106] Behrooz Parhami. *Introduction to Parallel Processing: Algorithms and Architectures*. Springer, 1 edition, 1999. 128

[107] Intel Labs. Single-Chip Cloud Computer, 2009. http://techresearch.intel.com/ProjectDetails.aspx?Id=1. 159