Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sandeep Kishan Singhal
August 1996

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

David R. Cheriton
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Patrick M. Hanrahan

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Craig Partridge

Approved for the University Committee on Graduate Studies:

Abstract

The best Qualification of a Prophet is to have a good Memory.
– George Savile

A *Distributed Interactive Simulation* application provides a group of users with the illusion of a single, coherent virtual world, even though the users may be physically located at different machines connected by a network. These applications demand that each user sees a consistent virtual world view, that users are able to interact closely with one another and with other simulation entities in the virtual world, and that maximum realism is provided by hiding the distributed nature of the application from users.

Faster processors, more powerful graphics hardware, and higher-capacity networks are supporting the development of distributed simulation applications containing more simulation entities and more detailed models of entity appearance and behavior. Consequently, these networked virtual environments are seeing increased use for multiplayer video games, military and industrial training, and collaborative engineering (e.g. collaborative design and distributed simulation of complex engineering models). Large distributed interactive simulation applications are growing to include well over 100,000 dynamic entities. Achieving the size and detail required by future simulations is constrained by limits in network bandwidth, network latency, and host processing power.

This thesis describes network protocols and algorithms to support *remote modeling*, allowing a host to model and render remote entities in large-scale distributed simulations. These techniques require fewer network resources and simultaneously support a broader range of entity types than previous approaches. The thesis begins by describing the Position History-Based Dead Reckoning (PHBDR) protocol, a simple, efficient protocol which provides smooth, accurate remote position modeling and minimizes dependencies on network performance and entity representation. We use

PHBDR as a foundation for building three additional protocols:

- Axis Point Protocol: Models remote entity orientation by tracking the position of points in the entity's local coordinate system. This protocol is designed to extract information from a broad range of entity representations.

- Multiple-Detail Channels: A protocol architecture for remotely modeling complex entities—those having non-rigid structure or articulated parts—at different levels of detail depending on locally available computational and network resources. This technique allows each host to allocate its local network and computational resources toward entities that are of primary local interest.

- Projection Aggregation Entities: A protocol for dynamically bundling information from a group of remote entities based on their type and location. This protocol is used for remotely modeling distant or uninteresting entities at a low level of detail.

In presenting these techniques, this thesis shows that a simple, efficient protocol can provide smooth, accurate remote position modeling and that it can be applied recursively to support entity orientation, structure, and aggregation at multiple levels of detail; these protocols offer performance and costs that are competitive with more complex and application-specific approaches, while providing simpler analyses of behavior by exploiting this recursive structure. In support of this claim, this thesis shows that:

- PHBDR is a simple, efficient protocol that provides smooth and accurate remote modeling for a broad range of entities and explicitly recognizes network latency.

- PHBDR is still smooth and accurate when used to model entity orientation, entity structure at multiple levels of detail, and entity aggregations.

- The recursive protocol structuring provides better network performance and reduced software complexity when compared with the application-specific approaches deployed in previous systems.

Acknowledgements

Gratitude is the memory of the heart.
– Jean Baptiste Massieu

I have discovered that the process of writing a Doctoral thesis is much like a barn raising common in pioneer communities during the nineteenth century. In a barn raising, relatives, friends, neighbors, and even complete strangers would gather to help a family erect a new barn. The barn's completion was met with a community-wide celebration. That tradition reveals how the first pioneers, while being adventurous, recognized the need to exchange knowledge and experience with each other by developing and maintaining a strong sense of community. Probably the biggest lesson I have learned in graduate school is the importance of such collaboration among modern-day research pioneers. Dozens of people have offered suggestions and insight, information, and encouragement that contributed to this thesis and the research it describes. Indeed, for my own little thesis barn raising, I am indebted to relatives, friends, neighbors, and complete strangers.

I thank my advisor, Professor David Cheriton, for his advice and support in performing this research. He has a unique intuition for what will and will not work, and his on-the-spot suggestions often saved me hours of fruitless programming. David always demanded that my research yield high-impact results and "not embarrass the grandkids." He has taught me that real engineering is a series of design tradeoffs. Finally, he has provided an invigorating research environment in which people are expected to question any and all ideas put before them.

Professor Patrick Hanrahan and Consulting Professor Craig Partridge read this thesis critically, and it benefited greatly from their comments. Professors Marc Levoy and Gio Wiederhold also pointed me toward several related references, while Anthony Tomasic first observed the relationship between real-time aggregation and data warehousing applications.

I have become notorious for walking down the hallways and bothering everyone with my latest research problems. My work benefited greatly from discussions with numerous members of my research group, particularly Ken Duda, Michael Greenwald, Hugh Holbrook, and Matt Zelesko. I especially thank Hugh who, along with Eric Halpern, implemented most of the PARADISE distributed simulation system on which many of my protocols were deployed. Luis Gravano, David Hoffman, and Hugh McGuire were always willing to drop everything, listen to my ideas, and offer valuable suggestions, even though my research was wholly unrelated to their own.

Numerous other people took time out of their busy schedules to provide information about their (often unpublished) work: Judith Dahmann (The MITRE Corporation), John Hines and Adam Whitlock (Naval Research and Development (NRaD)), Marty Johnson (Science Applications International Corporation (SAIC)), Yoshifumi Kitamura (ATR Communication Systems Research Laboratories), Duncan Miller and Dan Van Hook (MIT Lincoln Laboratories), Nobutatsu Nakamura (NEC Information Technology Research Laboratories), Kenny Sato (Sumitomo Electric Industries Limited), Dan Schab (Naval Air Warfare Center–Training Systems Division (NTSC)) who also provided the F-16 flight traces used in Chapter 4, Greg Troxel (BBN), and Michael Zyda (Naval Postgraduate School (NPS)). Randy Garrett (Institute for Defense Analysis (IDA)/DARPA) first offered me the opportunity to become involved with the STOW program and learn about cutting-edge military simulation technology; he and Stuart Milner (also of DARPA) ensured the availability of any information I needed and could always direct me to people capable of answering my questions. This thesis has truly benefited from an understanding of real application demands that I gained courtesy of these individuals.

The Stanford U.S.–Japan Technology Management Center translated several technical papers from Japanese to support my research. I thank Richard Dasher, Tom Cunningham, and Gary Brown for making this valuable service available to me. Tadashi Nakatani graciously translated additional materials from Japanese for me.

This research was funded by a graduate fellowship from the Fannie and John Hertz Foundation, by DARPA grant DABT63–91–K–0001, and by an equipment grant from IBM.

Graduate school involves more than writing a thesis, for it is also a time of intellectual curiosity. Besides discussing a wealth of research-related matters with my officemates, Jonathan Stone and Stuart Cheshire, I shared discussions ranging from the efficacy of three-phase transaction commitment to the viability of airline deregulation in Europe. I have learned so much from interacting with these two individuals, and I only hope that they found our interactions equally rewarding.

For those times in the past year when I was frustrated, depressed, disillusioned, lonely, or upset, Risa, Brooke, Karen, and Derek always listened to my problems with a sympathetic ear. Your moral support allowed me to keep things in perspective. You are a wonderful group of people, and I wish each of you the very best.

Finally, my debt to my parents, Ram and Sushma, is one that I can never describe fully, let alone begin to repay. Your faith in me has always exceeded my own. I continually strive to match your own dedication, integrity, and strength. Thank you for not ever asking me what a Ph.D student does all day.

My biggest regret is that my younger sister, Nidhi, could not live to see the completion of this thesis. At the time of her death, I had not even graduated from high school, and a Ph.D was the last thing on my mind. However, I can imagine no other person who would have relished its completion more than I do. It is therefore in her memory that I dedicate this volume.

STANFORD, CALIFORNIA
30 AUGUST 1996

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A *Distributed Interactive Simulation* application, as illustrated in Figure 1.1, provides a group of users with the illusion of a single, coherent virtual world, even though the computations run on heterogeneous systems that may be dispersed over a physically large geographic area (e.g. around the real world). The virtual world consists of a set of *entities*, objects that participate in some way in the simulation. Simulation entities may include, for example, human-controlled and computer-controlled vehicles; a terrain (and associated features such as rocks, trees, and buildings), and even logical objects such as the current weather state or an object group. In this interactive environment, each host presents its local user with a realistic viewport onto the virtual world. For example, the rendered scene depicts all of the virtual world entities that would normally be visible to that user. As a result, the user is unaware of which users are controlling the various entities, which computers are performing detailed computational modeling of the various entities, and where those humans and computers are physically located. Besides supporting geographic separation of the users, the distributed nature of these applications supports fault-tolerant simulation environments and allows the creation of large systems at relatively low cost compared to large, centralized architectures.

Faster processors, more powerful graphics hardware, and higher-capacity networks are supporting the development of distributed simulation applications containing more simulation entities and more detailed models of entity appearance and behavior. Consequently, these networked virtual environments are seeing increased use for multiplayer video games, military and industrial training, and collaborative engineering (e.g. collaborative design and distributed simulation of complex engineering models). These applications demand that each user sees a consistent virtual world view, that users are able to interact closely with one another and with other entities in the virtual world, and maximum realism is provided by hiding the distributed nature of the application from users.

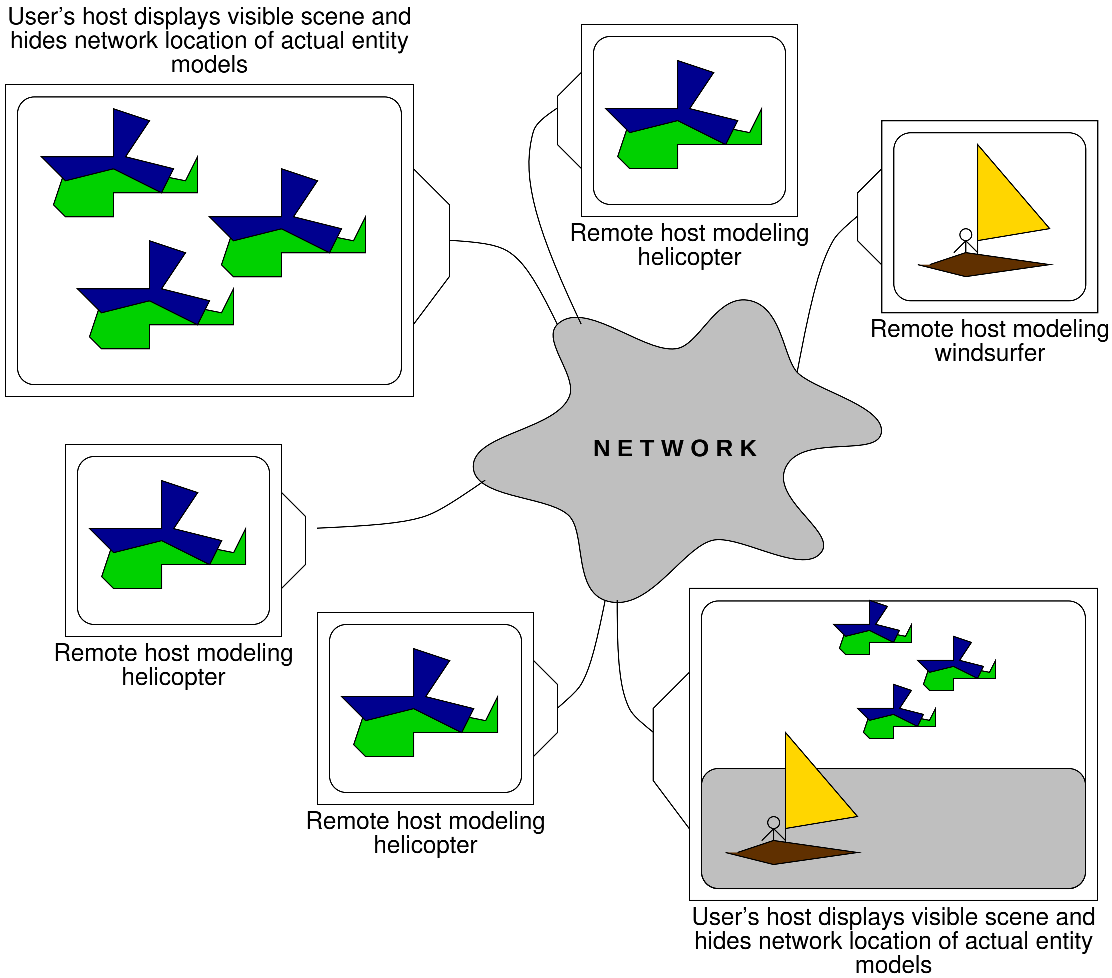

Figure 1.1: Distributed Interactive Simulation Application Presents a User with a Coherent Virtual World

To allow a local user to move about the virtual world and interact with remote participants, the host must accurately display the position, orientation, and structure of all virtual world entities visible to that user. For each entity, each host maintains an entity *model*—some state information along with a function that takes the state information as input and can generate the entity's current position, orientation, and structure on each frame. We designate one host's model as the "source" (also known as the "local" or "true") model representing the most accurate current state of the entity. Because the state of an entity model is updated in response to user input, the source model typically resides at the host whose local user is directly controlling the entity[1] to minimize the delay between

[1]Non-interactive models do exist; such models are either autonomous—in which case the source model may reside on

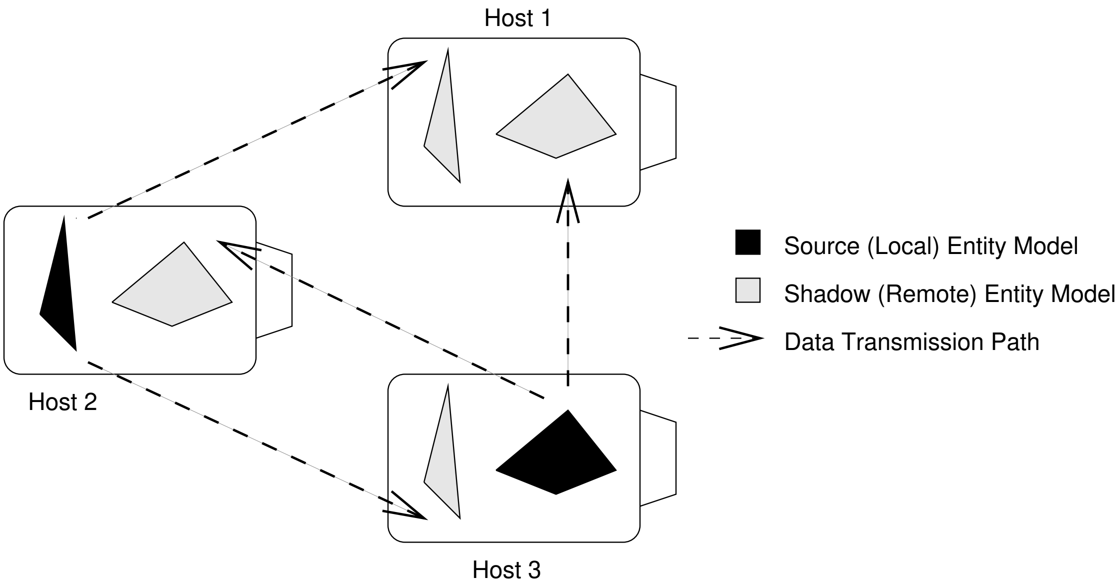

Figure 1.2: Simulation Nodes Exchange Information About Locally Modeled Entities

user actions and source model updates. All other simulation hosts maintain “remote” (also known as “shadow” or “ghost”) models providing less accurate versions of the entity’s current state. The source host must transmit information across the network to update the remote entity models at other hosts, as shown in Figure 1.2. The transmitted state information may take many forms, depending on the type of models maintained at the remote hosts. Every host periodically receives information from source models at other hosts and updates the corresponding entity model maintained locally. On each frame, each host queries its entity models (both source models and shadow models) for current entity position, orientation, and structure information. It then displays all visible entities on the screen, according to the local viewer’s position in the virtual world. To provide smooth, seamless animation, the host should update all entities—both local and remote—at the local frame rate. The user thus notices no differences between local and remote entities on the display.

1.1 Remote Modeling and Remote Rendering

Remote modeling is the task of maintaining a shadow representation of the position, orientation, and other attributes of an entity modeled at another host. *Remote rendering* is the task of smoothly

any host—or derived from information received from an actual physical object—in which case the source model would typically reside on a host located near the physical object.

integrating remote entity models with local entity models on a single display. Effective remote modeling and rendering should provide the local user with positional, behavioral, and structural fidelity about the entity.

Interactive users expect a positionally accurate view of remote entities. *Positional fidelity* means that the average position and orientation error of each remote model should be small. For example, in an auto-racing simulation, the user expects an accurate visual positioning of nearby competitors' cars in order to steer his own car and avoid collisions. Any significant perceived inaccuracy leads to confusion, incorrect actions, and inconsistent responses.

Animation of remote participants should also exhibit *behavioral fidelity*, meaning that the velocity and acceleration of each remote model should reflect the true entity's behavior. Even when positional fidelity is impossible, the scene should still aim to provide some level of behavioral fidelity. For example, the user wants to see that a distant car is swerving, even if the actual rate of motion is incorrect.

As distributed simulations become more sophisticated, remote rendering also needs to provide structural fidelity. *Structural fidelity* means that the remote model reflects real-time changes to the shape of non-rigid entities. For example, a swimming octopus does not retain a static form, and a piece of jello dynamically changes shape as it wiggles. Without providing structural fidelity, a distributed simulation cannot aim to provide a good sense of reality, because real entities are usually not rigid. Even an otherwise rigid entity might undergo a structural change after a sudden collision. An entity may also have *articulated parts*—attachments exhibiting relatively independent motion—such as a retractable radio antenna on an automobile or a rotating satellite receiver on a surveillance vehicle. Future simulations must transmit and process information regarding structural changes to the entity, although these changes may occur at a frame-rate granularity.

Existing distributed simulations have used a broad range of techniques for providing positional and behavioral fidelity in remote models, but these systems have only made limited efforts at supporting structural fidelity; most approaches in this arena have only handled statically-configured articulated parts [37].

Distributed simulation systems are evolving along several dimensions. First, they are growing in size to include dozens of sites and well over 100,000 entities [1]. Second, each simulation is expected to include a growing variety of entity types. Newer simulations also include entities that support dynamic terrain, weather, and electromagnetic transmissions. Third, the entity models themselves are becoming more detailed and dynamic. Entity models describe fine-grain motion, structural change, and values of numerous state attributes besides position and orientation. As a result of these trends, the remote modeling and remote rendering problems are growing in importance, size, and complexity. However, in meeting these demands, simulation designers must contend with several resource constraints.

1.2 Barriers to Remote Modeling/Rendering in Large-Scale Distributed Simulation

Within a year, we expect a top-of-the-line general-purpose workstation to contain a 500 MHz processor [19], contain 500 MB of memory, and be connected to a 100 Mbps Ethernet. We can reasonably expect that future machines will provide an order of magnitude more computational power and memory. Furthermore, gigabit-per-second Ethernet is already seeing experimental deployment.

Despite these developments, today's commercially available hardware cannot support the remote modeling and rendering requirements of a high-detail, 100,000 entity simulation. Moreover, the next generation of hardware is unlikely to keep up with the growing requirements on distributed simulation. These simulations face—and will continue to face—three resource limitations: network bandwidth, network latency, and host processing power.

1.2.1 Network Bandwidth

Distributed simulation design is constrained by the network bandwidth that is available for exchanging information among simulation nodes. This information includes updates about entity position and orientation, updates about other attributes of entity state such as color and infrared emissions, and data supporting simulation tasks such as entity collision detection and agreement, audio/video communications among participants, and clock synchronization.

For example, each entity must provide frequent updates (i.e. between two and 5 per second) to minimally support display and collision detection computation for close-range entities. In a large simulation containing 100,000 entities, the resulting wide-area bit rate would range between 96 Mbps (assuming a tiny 480-bit update packet and two updates per second per entity) and 375 Mbps (assuming a more likely 750-bit update packet and five updates per second per entity). Indeed, a traffic model [92] for a 100,000-entity military exercise predicts a sustained wide-area load of 230 Mbps with peak load of up to 700 Mbps. Even using multicast to prune the distribution of data, the data requirements still exceed the capacity of 100 Mbps Ethernet LANs [39]. Even worse, these bandwidth models assume controlled interactions between entities, update rates averaging only one per second, and the absence of video data. Notably, network tail circuits, which typically range in bandwidth between 1 Mbps and 45 Mbps, represent the most severe bandwidth bottleneck.

Network bandwidth will continue to be a limited resource for large-scale simulation. As we increase the number of entities and the level-of-detail of the individual entity models, more information must be exchanged among simulation nodes, so the simulation's network bandwidth requirements increase. For example, tight interactions between entities may require up to ten updates per second from each entity, particularly for distributed engineering models. Each entity can be reasonably expected to dynamically update at least four vertices (which is a conservative estimate and does not consider updates of non-position attributes). Even using multicast, a LAN bandwidth requirement of over 2 Gbps is not at all far-fetched, and again, we have not made any allowance for video data.

Finally, even if network technology could meet the bandwidth demands of a large distributed simulation, the bandwidth problem would remain. We have completely ignored the bandwidth demands of other applications, including other distributed simulations, sharing the same network. If network bandwidth were to become freely available, we can reasonably expect that most networked applications will have correspondingly high bandwidth requirements of their own. Therefore, it will always be advantageous to minimize the bandwidth demands of a particular simulation.

1.2.2 Network Latency

Distributing entity models onto multiple machines allows for greater scalability, but it also forces simulation hosts to contend with network *latency*, or delay. This delay includes several components, including transmission delay across the physical network medium, queueing delay in intermediate switches and routers, and processing delays at the endpoint hosts in preparing the packet for transmission and processing the incoming packet. Average one-way packet latencies between well-connected hosts (with at least T-3 connectivity) range from 45 ms (across the United States) to 85 ms (between the western United States and Europe or Asia); hosts with worse connectivity, including low-bandwidth SL/IP connections, can see latencies in the hundreds of milliseconds. Even worse, the *jitter* or variation in latency is considerable: latencies five times worse than the average case are common [78].

This latency means that simulation hosts cannot ever expect to maintain a *perfectly* consistent view of the virtual world. Any transmitted entity state update will be delayed by the network, and each remote site will see a different latency. Consequently, simulations must be explicitly designed to handle inconsistencies. For example, when two entities modeled at different hosts collide, no single host generally has enough information to immediately determine whether the collision actually happened and be sure that its decision is consistent with the perception at other hosts. Making such a consistent decision would require zero-latency data transmission.

Network latency simply cannot be eliminated. For example, speed-of-light propagation delays alone demand that the latency be at least 13 ms across the United States or 45 ms between the western United States and Europe or Asia. Though work is being done to improve packet processing latency at endpoint hosts [62, 63], these optimizations are addressing a relatively minor component of the end-to-end latency, which is dominated by router queueing and physical medium delays. Moreover, latency is a growing concern because large distributed simulations are incorporating sites separated by longer distances. At the same time, the interactions between simulation hosts are becoming more complex and, therefore, time-sensitive.

1.2.3 Computational Power

Simulation hosts incur computational overhead to receive and process incoming network packets about remote entities as well as to prepare and transmit updates about local entities. Current platforms cannot meet the networking demands of large-scale distributed simulations.

Even after using extrapolation techniques at remote hosts to reduce the required packet rate generated by a single entity, 100,000-entity simulations can still deliver a sustained 72,000 packets per second to each host [103] with burst rates of 480,000 packets per second [92]; again, these numbers are extremely conservative and assume controlled interactions between entities, update rates averaging only one per second, and the absence of video data. We assume that the packet reception interrupt and link-layer processing costs roughly 1 μs [2] or 500 cycles. UDP/IP processing accounts for approximately 4,125 cycles (roughly 1,500 instructions at a tightly optimized CPI of nearly 2.75)[3] Finally, running applications require roughly 1,000 cycles to recover from the cache pollution introduced during the interrupt handler.[4] Packet reception and UDP/IP processing

[2]Mogul and Ramakrishnan [62] report 142 μs for these operations. We conservatively use a considerably lower estimate to account for their previous-generation Alpha workstation and the delays introduced by their system instrumentation.

[3]Mosberger, et al [63] report 510 instructions for IP processing of a 160-bit payload (equivalent to application data payload of 128 bits after accounting for UDP header) on an Alpha workstation; the true instruction count should be considerably higher for any non-trivial payload. They report 1,594 instructions for TCP processing, and Kay and Pasquale [45] report only a 10% performance difference between TCP and UDP for small packets. We conservatively allocate 1,000 cycles for UDP processing. The 2.75 CPI estimate is probably low for our top-of-the-line workstation, because the processor speed has increased considerably relative to the memory speed.

[4]The results reported in [61] for a 500 Mhz processor are still valid because reading data from the second-level cache still costs on the order of 15 cycles. The reported cost ranges from 142 cycles to 15,000 cycles, depending on the workload. Again, we choose a conservative value.

therefore requires at least 5,625 cycles per packet, for a total CPU load of 80% to process the sustained 72,000 packets per second. This load processes almost no payload data, includes no application-level processing, excludes required machine tasks such as clock interrupts, and does not account for the burst packet rates. Moreover, we have intentionally underestimated our parameters by at least a factor of two. Clearly, today's general-purpose workstations are not equipped to handle the packet delivery demands of large simulations.

The packet delivery demands of distributed simulations are unlikely to be satisfied by future generations of workstations. First, the packet processing requirements increase with the number of participating entities, complexity of their models, and interactions between users. Second, packet processing—and particularly interrupt handling—is memory-bound. Memory speeds are not increasing at a rapid rate.

Even if the endpoint host could sustain the simulation's packet load, minimizing the number of cycles that are diverted toward packet reception and processing would remain as a fundamental goal. Packet reception and generation are not the only computational demands on simulation hosts: simulation hosts are required to render a high-resolution scene on a graphical display at an interactive frame rate, detect and resolve entity collisions, monitor user input devices and process their inputs, and model both local and remote entity state. The complexity of these other tasks is also increasing rapidly as simulations become more complex. For example, high-resolution graphical rendering by itself already demands the full resources of today's workstations, and collision detection complexity can increase roughly quadratically with the number of entities. Moreover, these non-packet-processing tasks have far greater impact on the user's perception of the simulation's quality. Therefore, we ultimately wish to devote as many CPU resources as possible toward these interaction tasks.

1.3 The Need for Fidelity Control and Aggregation

Given that workstations and networks cannot satisfy the requirements of current and future large-scale simulation, we must seek to reduce the bandwidth and computational demands of distributed simulation without introducing additional latency for information dissemination. Three basic approaches are available:

- Transmit less information about each entity and/or transmit entity updates less frequently.
- Limit the number of entities that are of interest to each host.

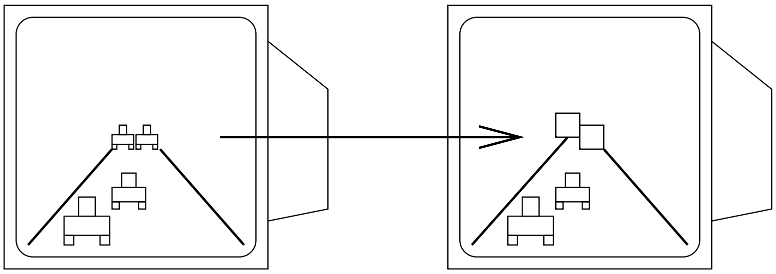

Figure 1.3: Reducing the Modeling Detail and Fidelity of Distant Entities To Reduce Local Packet Rate and Computational Requirements

- Have each packet provide information about multiple entities.

Our first option is to transmit less information about each entity or transmit that information less frequently. However, completely dispensing with transmitting certain types of information could have a detrimental effect upon the quality of the simulation, particularly for users who are interacting closely with the source entity. Consequently, to support tight interactions, the source must still supply all of the available data. We observe, however, that some receivers may be able to selectively ignore some of this information. For example, if an entity is far from the local viewer in the virtual world, then that host may not require updates to that entity's color or structure, and it might be able to accept less frequent position updates. Figure 1.3 illustrates a scene in which distant cars receive lower positional and structural fidelity and are therefore rendered as simple "blobs." The graphical detail and modeling fidelity requirements on each entity are likely to change during the simulation. As the viewer accelerates, the distant cars should receive increased graphical detail and modeling fidelity, and the host must receive more frequent and more descriptive updates for those cars. Similarly, as the viewer moves away from the cars, the host can receive less frequent and less descriptive packet updates. We conclude, therefore, that reducing the amount of received data is possible, but we cannot reduce the amount of transmitted data: Only the receiver is capable of determining how to reduce modeling detail without sacrificing the realism of the local display.

Our second option for reducing packet and computational load is to limit the number of entities from which each host receives update information. For example, a host might only subscribe to information from the N closest entities, where N is some constant. However, this approach has the greatest potential for distorting the simulated world presented to the user because it effectively alters the user's natural visibility range. We conclude that because of the loss in realism, this approach is inappropriate for simulations involving humans.

Our third option is to *aggregate*, or group, information from multiple updates into a single packet. For example, updates about all of an octopus' legs might be combined into a single packet. Similarly, a single packet might update the location of an entire school of fish, or it might summarize information about all entities located in a particular region of the virtual world. Remote hosts can then use this aggregated information to update their models of each entity. However, aggregation poses three challenges. First, the simulation designer must decide what information to aggregate within each packet. If a receiver is only interested in a fraction of the information within a particular aggregation, then the bandwidth and computation required for a large aggregation packet may exceed the bandwidth and computation required for the one or two small update packets that the receiver actually required. Because each receiver is potentially interested in a completely different set of entities, designing optimal aggregations is a daunting task. Second, different components of the aggregation update are likely to be generated at different times. Collecting these components and packaging them into an aggregation packet necessarily introduces an additional delay on the transmission of each data component; aggregated information is less timely than direct information transmission. Consequently, aggregation is not an appropriate technique for close-range interactions requiring high detail and low latency. Third, the simulation designer must designate which host is responsible for collecting and packaging the data components of the aggregation update. This host must be located near the information sources to minimize the latency introduced by aggregation. This task is made difficult because aggregation membership may change during the course of the simulation. We conclude, therefore, that aggregation is a viable approach to reducing bandwidth and computational load, yet it cannot replace direct per-entity transmission and must be designed with care.

To implement large-scale simulations, each host needs the ability to receive lower-detail and lower-frequency updates for a selected set of entities while retaining the ability to receive full-detail and high-frequency updates for the remaining entities. Furthermore, hosts should have the ability to receive packets containing aggregated updates for those entities where information latency is not the primary constraint; again, the host must retain the ability to receive low-latency updates directly from the entity. This thesis addresses these needs for large-scale distributed simulations.

1.4 Original Contribution of This Thesis

This thesis shows that a simple, efficient protocol can provide smooth, accurate remote position modeling and that it can be applied recursively[5] to support entity orientation, structure, and aggregation at multiple levels of detail; these protocols offer performance and costs that are competitive with more complex and application-specific approaches, while providing simpler analyses of behavior by exploiting this recursive structure.

In support of this thesis, I showed that:

1. **The Position History-Based Dead Reckoning (PHBDR) protocol is a simple, efficient protocol that provides smooth and accurate remote modeling for a broad range of entities and explicitly recognizes network latency.** I demonstrate a method for systematically analyzing remote modeling protocol performance over a broad range of expected entity behaviors by means of mathematical analysis, controlled simulation on a representative set of entity paths, and deployment experience. Analysis of PHBDR shows that within the expected available bandwidth, it can support the remote modeling requirements of the next generation of large-scale simulation.

2. **PHBDR is still smooth and accurate when used to model entity orientation, entity structure at multiple levels of detail, and entity aggregations.** I define the Axis Point protocol for modeling entity orientation, Multiple-Model Channels for representing dynamic entity structure, and Projection Aggregation Entities for representing groups of entities. I use results from the basic PHBDR protocol analysis to analyze the performance of these recursively-defined protocols. This approach reduces the amount of analysis that would otherwise be necessary had each protocol been developed independently.

3. **The recursive protocol structuring provides better network performance and reduced software complexity when compared with the application-specific approaches deployed in recent systems.** The recursive protocol design promotes code re-use at the source and remote hosts. The recursive structuring permits each source host to efficiently support multiple remote modeling protocols, each providing a different representation of the entity's behavior. The availability of multiple protocols yields a net reduction in the simulation's

[5] By "recursively" here, I mean that it may be used as a component within a more complex protocol. Although the term "component architecture" might be more appropriate, we use the term "recursive structuring" as it is consistent with existing work in the area [16, 106].

aggregate bandwidth requirements because each remote host locally determines its own data requirements and only subscribes to high-bandwidth information when justified by bandwidth availability, computational resources, and user interests.

The next chapter surveys how existing distributed simulation systems have solved the remote modeling problem, how they have used aggregation to support scalability, and the limitations of those approaches. Chapter 3 introduces Position History-Based Dead Reckoning (PHBDR), our basic protocol and algorithms to model the position of rigid entities. The protocol makes minimal assumptions about the simulation environment and is computationally simple. Chapter 4 presents a systematic approach to analyzing dead reckoning protocol behavior and applies that approach to evaluating PHBDR's network bandwidth requirements and remote modeling error. The evaluation shows that PHBDR provides smooth, accurate modeling of a broad range of entity curves. The next three chapters discuss how the simple PHBDR protocol may be applied recursively to address move complex remote modeling requirements: Chapter 5 discusses the Axis Point protocol that recursively employs the PHBDR protocol to remotely model the orientation of rigid entities. Chapter 6 describes how PHBDR is used to model the structure of complex entities at different levels of fidelity, depending on local computational and network availability. Chapter 7 employs PHBDR to support low-fidelity remote modeling of entity groups based on the entity type and location. Chapter 8 concludes this thesis with a discussion of the successes and limitations of this work, as well as an indication of what work remains in this area.

Chapter 2

Related Work

Distributed simulation applications have adopted widely disparate approaches for disseminating information about entity motion and modeling those entities at remote hosts. To some extent, this broad range of techniques reflects the relative lack of experience in developing this class of applications. The heterogeneity also arises because simulation systems are designed to operate in network environments providing different latency and bandwidth characteristics. For example, local-area networks offer low-latency communication best suited for small simulations requiring tight coherence among the participating hosts, but a wide-area network offers high-latency communication best suited for larger simulations able to tolerate larger discrepancies between hosts. Furthermore, a system designed for a telephone dialup network has traditionally been incompatible with a system designed for a private leased-line network or the Internet.

This chapter describes how existing distributed simulation systems transmit updates about entity state over the network. We begin by discussing the variety of deployed approaches to remote modeling and rendering, allowing each host to maintain a real-time representation (position, orientation, and other state information) for each entity and present a smooth, accurate graphical view to the user. We then discuss techniques used to reduce the bandwidth and computational demands of large-scale simulations. We conclude by summarizing the limitations of the prior work and discussing how the techniques described in this thesis provide a scalable solution to remote modeling and rendering in a variety of network environments.

2.1 Tradeoffs in Real-Time Remote Modeling

As illustrated in Figure 2.1, the design of the data dissemination system introduces a fundamental tradeoff between remote modeling accuracy and the flexibility of the simulation system. At one

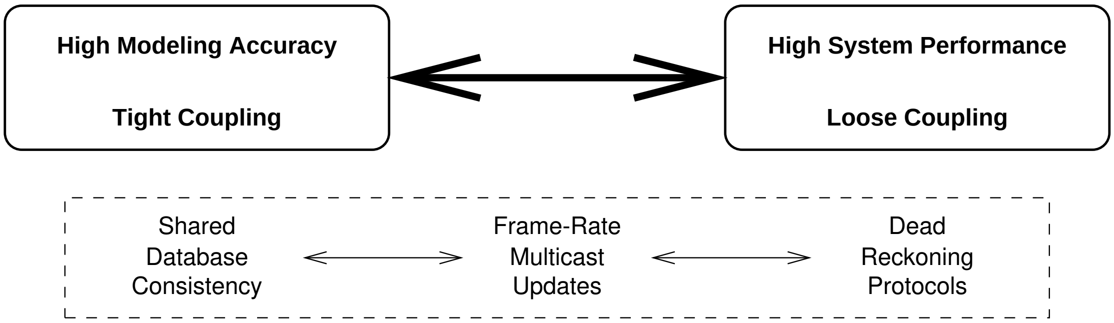

Figure 2.1: Tradeoffs in Remote Rendering for Distributed Simulation Applications

extreme, distributed simulation systems are tightly coupled and therefore guarantee perfect accuracy in the remote modeling. To achieve this accuracy, such systems demand high network bandwidth to support frequent updates, low latency and limited jitter, and a limited number of participants; consequently, tight coupling is impractical for large-scale interactive systems. At the other extreme, systems are loosely coupled and therefore tolerate some inaccuracy in the remote modeling. Such systems, though typically adding computational complexity, require less network bandwidth and less frequent updates, are more resilient to network latency and jitter, and support better real-time performance across a larger number of heterogeneous platforms.

Three remote modeling techniques are in use by most current systems: shared database consistency, frame-rate updates, and dead reckoning protocols. Shared database consistency represents one extreme of the design tradeoff by potentially providing the greatest accuracy but the least scalability. Dead reckoning protocols fall into the other extreme by accepting imperfect remote modeling to support larger simulations. Frame-rate updates represent an intermediate solution. In the following sections, we describe each of these approaches, highlight how they have been implemented in existing distributed simulation systems, and discuss their advantages and disadvantages.

2.1.1 Shared Database Consistency

Shared database consistency provides absolute consistency between the different hosts participating in a distributed simulation application. The consistency approach guarantees that hosts share the same information about each entity's position, orientation, structure, and other state. As a result, all hosts display identical views of each entity, thereby eliminating any possibility of inconsistency or confusion between interacting players.

Architecturally, this technique strives to hide the network environment from the application. From an application developer's perspective, all simulation entities appear to be maintained locally. The resulting simplicity of the application programming model contributes to the popularity of this technique, particularly for developers who are modifying existing uniprocessor simulations for networked operation.

A central server provides the simplest implementation of shared consistency. In the Shastra system [2] from Purdue University, for example, a central server provides the infrastructure for collaborative design applications. The first node to start a collaborative application is designated as the application's group leader. This group leader communicates with a Shastra session manager process to control the admission of additional clients into the distributed application environment. All communication between application nodes flows through the session manager, which provides reliable and ordered delivery of update messages to all clients. Because any user may modify the state of any simulation entity, the session manager prevents multiple clients from simultaneously modifying the same data. Hosts communicate with the session manager to obtain access, modify, and copy permissions to the shared entities; the session manager uses a token-passing mechanism to provide fair access to the shared entities.

Instead of using a centralized server to provide shared database consistency, the DIVE virtual reality system [14] utilizes a distributed data management model. The system is built above the ISIS communications library [7] that provides reliable, in-order delivery of network data. Using this network software base, DIVE creates the abstraction of a shared database, replicated at each participating host, that stores all entity information. DIVE allows a user to see, move, and interact with any entity inside the virtual world. When a user wishes to update the position of an entity, his local host obtains a distributed shared lock on that entity in the database, updates the local copy, reliably broadcasts the change throughout the shared database, and finally releases the lock.

Some recent systems have attempted to maintain the illusion of a shared database while allowing receiving hosts to independently determine the desired data consistency level for remote entity state information. With the BrickNet toolkit [82], for example, hosts transmit updates to the central server asynchronously. Data subscribers can then select from a broad range of approaches, ranging between absolute and loose consistency, for receiving updates. The Networked VR virtual reality project [66] from NEC Research Laboratories carries this idea one step further by allowing each remote host to independently choose when to actually receive updates. The server (referred to as a "Dialogue

Manager") maintains two outgoing buffers for each client. A FIFO buffer stores sequential data and holds incremental update information. This buffer is used when transmitting a complete entity description would be prohibitively expensive. For example, the FIFO buffer is typically used to transmit incremental changes to the entity's shape model. The second buffer, an overwrite buffer, is used for most other types of data, such as position, orientation, color, etc. Because data in the overwrite buffer is self-contained and completely describes the associated attributes, data in this buffer is overwritten whenever more up-to-date information becomes available. Clients explicitly request buffer information from the server, thereby applying a "pull" rather than a "push" model for communications [65].

Although the easy programming model and guaranteed consistency are desirable features, the shared database consistency approach does not easily scale to large-scale simulations containing thousands of entities over a wide-area network. Absolute consistency imposes significant communication overhead. In a DIVE application, for instance, each host much perform several message exchanges on each frame to obtain the lock on the entity state, reliably transmit the update, and release the lock. Moreover, because the host must wait until the transaction is complete before rendering the new frame, the local frame rate is limited by network latency and remote host response time. The network also cannot exhibit high jitter; hosts display updates to an entity's position only after they become available through the shared database, variable network latencies for updating the database destroy the illusion of smooth motion at remote hosts. These bandwidth, latency, and jitter issues make absolute consistency only practical over local-area networks. Finally, all participating hosts must possess comparable computational power, because the local frame rate is determined by the response time of the slowest host.

2.1.2 Frame-Rate Updates

By transmitting *frame-rate updates*, simulation designers relax the state consistency between hosts in order to achieve greater scalability than that provided by shared database consistency techniques. Under the frame-rate update approach, entity state updates are transmitted (typically in broadcast mode) unreliably at the source host's frame rate. Although updates may be lost (or delayed long enough to be subsumed by another update), the simulation designer assumes that the high update frequency will make such inconsistencies unnoticeable. Because the application maintains no shared locks over simulation entities, multiple hosts could attempt to modify the same entity simultaneously. Systems eliminate this conflict by explicitly associating each entity with a particular host from which update packets must originate; update packets for that entity originating from any other host are ignored.

By relaxing the absolute consistency requirement, frame-rate updates permit permits greater decoupling between participating hosts. Each host simulates local entities and transmits updates without regard to how or where those updates are being delivered. The local simulation is less directly affected by network latency or remote host performance. In particular, sending hosts do not need to wait for lock acquisition/release or update acknowledgements on each frame. At the receiving host, the application simply waits for update packets to arrive and modifies the local representation of the entity accordingly. The frame-rate update approach keeps the underlying communication library quite simple.

The SGI Flight Simulator represents the most significant use of the frame-rate update technique. In this multi-player "dogfight" application, each player attempts to shoot down opponents' planes. On each frame, each host simply broadcasts the complete 140-byte data structure describing the local airplane. New players are recognized at remote hosts when an update packet arrives with a previously unseen player name. Hosts eliminate players if no update packet with that player's name arrives within a suitable timeout period.

Another implementation of the frame-rate update technique was demonstrated at the IMAGINA'93 conference [72]. In this two-player system, participants could see and talk to one another while interacting with a virtual environment. In this system, two machines—one in Paris and the other in Monte Carlo—were connected by a 64 Kbps data link over which the hosts exchanged an inventory listing the position of 100 entities (400 bytes in all). Each host transmitted 10 updates per second, resulting in an approximate 100 ms response time that the participants felt to be inobtrusive and, therefore, acceptable. A similar system to support intravascular tele-surgery has also been demonstrated [3]. In this system, the source host transmitted an appropriate viewpoint position, along with the position of the surgical catheter device. At the remote host, the appropriate image was re-constructed on each frame. The source and destination hosts were separated by nearly 220 miles and were connected by a 156 Mbps ATM link. In this case, frame-rate updates were appropriate given the availability of a dedicated point-to-point link and the need for high data consistency within a medical application.

While the SGI Flight Simulator, the IMAGINA'93 demonstration, and the intravascular tele-surgery demonstration limited themselves to modeling rigid entities, ATR Communication Systems Research Laboratories has developed a virtual space teleconferencing system for the remote modeling of human faces [105, 64]. This task poses additional complexity because of the dynamic structure of the entities being modeled. Each host stores a prototype wire-mesh model of a Japanese face. Before the teleconference starts, multiple photographs of each speaker are taken, and those images are processed to customize the prototype for that person's particular facial features. These revised prototypes are exchanged. To generate update packets during the teleconference, the system captures multiple images of the speaker and processes the images to determine the current physical location of facial features such as the tip of the chin, corner of the eyes, nose position, etc. The system transmits the position of these facial features over the network. The remote host uses the update packet data to anchor corresponding points on the wireframe model and then apply a texture mapping before displaying the image. A newer version of the system [67] uses a 3D digitizer to generate a wireframe model for each speaker without the need for a prototype model. The teleconferencing system tracks the position of blue tape marks on the speaker's face, thereby reducing the image processing complexity and achieving higher frame rates with the same computational resources. Update packets in the current system describe 11 facial points, 4 body points, and 10 points on a hand-held dataglove for a total of 528 bits. Using a 1,000 node wireframe model, the teleconference is rendered at 10 frames per second [47].

In contrast to broadcast-based systems, the VEOS System [9] from the University of Washington uses a so-called "epidemic" approach to distributing frame-rate updates. VEOS entities are organized hierarchically. During each iteration of the entity's event loop (or "frame"), the entity stores updated state information into a local buffer known as the "boundary view." At the end of each iteration, a communications daemon pushes information from this buffer into an output FIFO for each sibling and parent entity that has expressed interest in that data. Those recipient entities, in turn, export that information to other entities during their next frame. If a FIFO already contains an old version of this information—for example, if the information has not yet been absorbed into the destination entity's "external view"—then the stale version in the buffer is simply overwritten. As a result, all entities receive updates in-order and always receive the most recent update available.

To reduce network traffic in frame-rate update systems, a *relevance filtering* mechanism eliminates traffic from entities that are irrelevent to the local user. The RING system [29], for example, implements relevence filtering by sending all entity updates to servers that maintain a record of the current location of each viewer and forward updates only to viewers that can see that entity. Alternatively, one can implement relevence filtering by transmitting updates to a different multicast address for each entity, with remote hosts only subscribing to updates from entities of local interest.

Techniques similar to frame-rate update approaches are also used in virtual reality systems which must use information obtained from a Head-Mounted Display (HMD) to remotely model the user's position and orientation and render an appropriate image. The position tracker device samples a set of sensors and computes the position and orientation from these sensor measurements; it then transmits the position values (usually over a dedicated line) to the rendering computer, typically at a rate of 20–60 Hz. The position determination technique employed by the tracker may be divided into several categories [57]:

- **Mechanical:** The user is physically connected to a point of reference by a series of jointed linkages. The sensor triangulates the user's exact location based on the joint angles.
- **Optical:** The sensors detect the direction and intensity of a set of fixed-position LED transmitters, the appearance of a pattern placed in a fixed location, or the distortions of reflected laser light.
- **Magnetic:** The system uses electric current to create a low-frequency magnetic field which the sensors detect and measure.
- **Acoustic:** Sensors detect the direction and intensity of high-frequency sound waves transmitted from a fixed source.
- **Inertial:** Accelerometers and gyrometers respectively measure changes to the entity's velocity and orientation; the sensors use these changes to maintain a model of the entity's current velocity (and, therefore, position) and orientation. Often, information from these two devices are combined to produce a more robust estimate of orientation [97, 26].

The frame-rate update technique has several limitations. First, the assumption of low-latency data dissemination is often unachievable. Latencies of 250 ms are not uncommon over wide-area networks. HMD position tracking typically requires maximum latencies of 50–170 ms, and studies of multimedia applications [4, 21] indicate that humans become uncomfortable with inconsistencies on the order of 100 ms; the inconsistencies become intolerable at 200 ms. Indeed, the IEEE communications standard for distributed simulation [38] demands maximum end-to-end latency of under 100 ms. (Notably, the IMAGINA'93 demonstration, linking Paris to Monte Carlo—roughly 600 miles—represents the point at which the latency problems start exceeding these limits.) Second, the delay jitter over wide-area networks means that packets do not arrive at a fixed rate, and they may not even arrive in order. If a packet is delayed, the remote host must redisplay the entity at its previous position because it cannot wait for the data to arrive. Third, most existing networks cannot support the necessary bandwidth for frame-rate updates. Transmitting 128-byte update packets at 60 frames per second, only 520 entities are needed to impose a 40% load on a 100 Mbps Ethernet.

Finally, the sender's frame rate may be slower than that of a receiver, in which case the remote entity on that receiver's display does not exhibit the same smoothness as local entities. For example, ModSAF and other computer-generated forces in military simulations often only run at 2–4 Hz [92], hardly enough to effectively feed a remote manned simulation which typically runs at over 15 Hz.

Faced with these problems of latency, jitter, bandwidth, and frame-rate correspondence, large-scale simulations need to transmit information at a lower frequency [70], to relax the sensitivity to network delay, and to permit receivers to update the remote model at their own frame rate.

2.1.3 Dead Reckoning Protocols

Dead reckoning protocols represent the other extreme of the remote rendering technique design space, for they sacrifice coherence to maximize system scalability. In a dead reckoning system, hosts model the state (position, velocity, and acceleration) of each remote entity by applying predictive extrapolation based on update information sent from each entity's local host. Dead reckoning at remote hosts typically reduces bandwidth requirements because update packets can be transmitted at lower-than-frame-rate frequencies. Because remote entities are updated at a slower rate than local entities (whose models are driven by user input, rather than incoming packets), receivers must use extrapolation to provide seamlessly integrate remote and local entities on the display. Each receiver performs this extrapolation independently, so it can provide a smooth rendering despite a slow source host.

On the other hand, dead reckoning does introduce several limitations. First, dead reckoning does not guarantee that all hosts share identical state about each entity. Instead, dead reckoning protocols require hosts to tolerate and adapt to potential discrepancies. Second, simulations that rely on dead reckoning protocols are usually more complex to develop, maintain, and evaluate. The application developer must be aware of the network's behavior and typically tailors the simulation software and algorithms to operate within a wide-area network environment. For example, because the extrapolation of entity position and orientation is imperfect, collision detection requires a distributed agreement protocol. To avoid presenting a jerky view of the entity's position on the display, the host must also apply apply some smoothing or convergence algorithm to correct the extrapolated model after a new update arrives.

Dead reckoning protocols have evolved over the past several years, both in terms of their sophistication and in terms of how well their modeling behavior is understood.

2.1.3.1 Early Dead Reckoning Systems: Amaze

The Amaze multiplayer game [5] represents one of the earliest implementations of a dead reckoning protocol in distributed simulation. Hosts transmit position and velocity updates about local player(s) roughly once per second. Remote hosts use the velocity information to predict the entity's future position, though they ignore network latency effects because of the low latencies of the LAN environment on which the system is deployed. Given a packet specifying position point x_0 and velocity vector x'_0, the dead reckoned position at time t after packet arrival is:

$$x(t) = x_0 + x'_0 t \tag{2.1}$$

In this game, the slow speed of entities relative to update rate simplifies the remote modeling problem considerably because the potential error in the remote model is always small. Therefore, the Amaze system did not integrate a convergence algorithm, so the rendered position of the remote entity "jumps" immediately to the new position when an update arrives.

2.1.3.2 Commercial Systems: SIMNET, DIS, and STOW

The U.S. Army's Simulation Networking (SIMNET) system [69] first introduced dead reckoning protocols on a large scale. SIMNET implements a networked battle simulation system in which up to fifteen participants manipulate tanks and planes within a virtual battlefield. As in the SGI Flight Simulator, update packets contain a complete description of the entity, including its position and velocity (but not acceleration). However, SIMNET moves away from the fixed-rate update approach used by earlier systems. The transmitting host maintains two models of each entity: the true model representing the entity's actual position (updated by user input, autonomous control, and external forces) and the remote model (that applies dead reckoning to the entity's transmitted update packets using the same algorithm used in Amaze). The host transmits an update packet either when the true and dead reckoned models differ by some error threshold or when no update has been otherwise sent within a five second timeout period. This timeout allows remote hosts to recognize when an entity is no longer on-line, and it allows a new entity to enter the simulation simply by sending an update packet.

The Distributed Interactive Simulation (DIS) protocol [37], IEEE standard 1278, targets larger virtual environments populated by hundreds, if not thousands, of participants. The DIS dead reckoning protocol [36] is similar to the SIMNET protocol in most respects, though its update packets also include acceleration, orientation (specified in Euler angles [76]), and angular velocity information. Therefore, with an update packet specifying position point x_0, velocity vector x'_0, and acceleration vector x''_0, the dead reckoned position at time t after packet arrival is:

$$x(t) = x_0 + x'_0 t + \frac{1}{2} x''_0 t^2 \tag{2.2}$$

Entities may adopt different timeouts for update transmissions, although five seconds remains the default; moreover, particular entities may be configured to use the first-order dead reckoning employed by SIMNET and Amaze. The DIS dead reckoning algorithms have been adopted by successor programs such as STOW (Strategic Theater of War) [1].

The relatively wide deployment of the DIS protocol has encouraged some initial efforts to analyze its performance in terms of both remote modeling accuracy [79, 101] and network bandwidth requirements [51]. This work has used flight simulator traces to compare the performance of various dead reckoning protocols with the goal of validating the second-order positional and first-order orientation dead reckoning algorithms used by the DIS protocol. A more control-theoretic analysis was undertaken by Foster and Massel [25] using regression analysis to produce a formula predicting the average dead reckoning error as a function of dead reckoning threshold. This work also analyzed the effects of numerical precision on dead reckoning error. Other work has concentrated on improving the effectiveness of modeling entity orientation, either by using better implementation techniques [91] or by using quaternions to represent orientation [11].

2.1.3.3 Recent Research: Moving Toward Specialized Dead Reckoning Algorithms

Recognizing the difficulty of the remote modeling problem, most recent research systems have strived to develop specialized algorithms for a particular class of entity or motion. For example, the NPSNET system [70], developed at the Naval Postgraduate School, provides an experimental land-based virtual battlefield environment over LAN network environments (with recent support added for Wide-Area Networks [53]). NPSNET optimizes the DIS dead reckoning protocol for tank motion by transmitting only the entity position, a two-dimensional velocity vector, and an “align to ground” flag that causes remote hosts to adjust the entity’s pitch and roll to ensure that it is drawn on the ground. This latter feature eliminates the need to explicitly transmit orientation information in most cases.

Amnon Katz and Kenneth Graham at the University of Alabama have optimized a dead reckoning algorithm for aircraft that move with a constant “angle of attack” [43]. This condition arises when the aircraft moves with a constant acceleration vector when described in its body coordinate system;

the resulting path is a spiral (with circular paths representing a special case). The remote host receives the initial acceleration vector x_0'', which may be decomposed into a constant component magnitude A_{0t} representing acceleration in the direction of the entity's motion and a constant component magnitude A_{0n} representing acceleration normal to the entity's motion. Furthermore, the entity's initial velocity vector x_0' may be decomposed into unit vectors tangential to the body (e_{0t}') and normal to the body (e_{0n}'). With the entity's initial speed $s_0 = |x_0'|$, the forward speed over time is therefore

$$s(\tau) = s_0 + A_{0t}\tau \tag{2.3}$$

They define the following constants:

$$\alpha = \frac{A_{0t}}{A_{0n}} \tag{2.4}$$

$$c = \frac{A_{0n}s_0^2}{4A_{0t}^2 + A_{0n}^2} \tag{2.5}$$

and the following expressions:

$$\theta = \frac{A_{0n}}{A_{0t}} \ln \left| \frac{s(\tau)}{s_0} \right| \tag{2.6}$$

$$x_p = ce^{2\alpha\theta}(2\alpha \cos\theta + \sin\theta) \tag{2.7}$$

$$y_p = ce^{2\alpha\theta}(2\alpha \cos\theta - \sin\theta) \tag{2.8}$$

and hence the dead reckoned entity position is:

$$x(\tau) = x_0 + (x_p - 2\alpha c)e_{0t}' + (y_p - c)e_{0n}' \tag{2.9}$$

Furthermore, their "phugoid scheme" recognizes that for these curves, an aircraft's orientation is determined solely by the spiral's radius and the aircraft's speed. Consequently, accurate dead reckoning can be performed using only position, velocity, and acceleration updates; the airplane's position and velocity determine its orientation.

Dead reckoning is commonly used for tracking an entity's position and orientation based on inputs from remote sensors while accommodating for noise in the sensor readings, low frequency in the sensor updates, or high latency in sensor information. The common approach to these problems, particularly that of inaccurate sensor readings, is to employ a form of Kalman filtering [42, 87, 41]. A state matrix encodes the current position and velocity estimate and is used to derive position estimates at any given (present or future) time. When a new sensor reading arrives, the Kalman filter updates the current position estimate by convolving the position sample derived from sensor/radar with the current estimate provided by the state matrix; the relative weights given to these two values is determined from an estimate of the error variances in the sensor or radar readings. A similar weighting is used to update the velocity estimate. Though the Kalman filter provides a general-purpose framework for predicting and smoothing updates, researchers have continued to customize the Kalman filter algorithm based on the application.

Early specialization was done at the Naval Research Laboratories where dead reckoning algorithms were optimized to model ship motion along the earth's spherical surface based on inaccurate position estimates transmitted from a remote sensor. With regard to specializing the Kalman filter, Willman [104] wrote:

> The tracking algorithms are based on the Kalman filter and Bayesian smoother for a specific motion model in which a ship's motion is approximated as the vector sum of a constant (average) velocity and a two-dimensional (random) Brownian motion. The intensity of the Brownian motion ... is selected to correspond to the extent of maneuvering performed by the ship with respect to a constant-speed, great-circle course.

Trunk and Wilson [93] specialized the Kalman filter to track ship motion based on signals received from multiple radars; each radar signal provided both range and direction information, though only one component of each measurement was accurate. Moreover, because the basic Kalman smoother would not quickly detect when a ship changes direction, Trunk modified his Kalman smoother to lend more weight to recent radar measurements when the filter's current position estimate and the radar's current position estimate diverge significantly. These Kalman filter approaches share the disadvantage of being computationally complex (requiring on the order of 110 addition and multiplication operations on each time step), and their state matrix update cannot accommodate out-of-order updates.

The MIT Media Lab used a specialized Kalman filter to predict the position of drumsticks as they were played above a sensor pad [27]. Whenever the sensor was hit by a drumstick, a computer attached to this sensor was responsible for generating the sound of a drum shot. However the latency between the sensor impact and the sound generation was noticeable, so the researchers attempted to synchronize the drum sound with the sensor impact by predicting the position of the drumsticks. The computer received position updates from the drumsticks 30 times per second and used the Kalman filter to generate a second-order approximation of the drumstick position for 33 ms in the future. The approach was effective for most cases; however, the researchers discovered that the predictor would often "overshoot" and predict a drum shot because drummers regularly move the sticks downward toward the drum without actually touching it. To counter this effect, the researchers introduced an "emergency" signal that allowed the sound generator to disregard the previous prediction when a sudden change in the drumstick behavior is detected; the emergency signal was effective because of the relatively low latency between the physical sensors generating updates and the computer system processing that information. The dead reckoning approach was finely tuned to the particular application and required the generation of a specific noise model for user motions. Moreover, the approach did not provide general applicability because the predictive time interval was short (33 ms) compared to that needed over wide-area networks, the relative drumstick motion distances were short (250 mm), and updates occurred regularly and at high frequency.

Most recently, position tracking has become increasingly common with Head-Mounted Displays (HMD) which, as discussed above, can exhibit position generation latencies of up to 170 ms—enough to cause "simulator sickness" among users of virtual reality systems. To compensate for this latency, the graphics system applies a customized dead reckoning algorithm to predict the participant's current location based on the delayed position and orientation information provided by the sensor system. An early helmet-mounted display system developed for the Air Force [74] used a simple dead reckoning algorithm that treated acceleration as a random number between -1.0 and 1.0. To achieve better performance, Liang, Shaw, and Green [49] employed an algorithm more specifically tuned to head motion:

$$x'' = -\beta x' + \sqrt{2\sigma^2\beta}\,w(t) \tag{2.10}$$

where x' and x'' are the angular velocity and acceleration, $w(t)$ is a noise function, β is a time factor representing the smoothness of the prediction, and σ is a variance factor that controls how aggressively the orientation is changed. This dead reckoning algorithm assumes that the user's viewing direction is generally fixed and changes infrequently. Though it provides good results in this application, this dead reckoning algorithm is not generally applicable. The experimental Virtual Laboratory system at IBM [102] provides another example of this move toward specialized dead reckoning protocols for sensor tracking. The Virtual Laboratory employs a custom dead reckoning algorithm to predict the position of the participants' hands in a bouncing ball simulation.

To support this trend toward specialized dead reckoning algorithms, the VERN system [8] aims to provide an entity-oriented development framework into which specialized dead reckoning algorithms can be integrated. The system defines an `AbstractVERNObject` class, subclassed by `AbstractPlayer` and `AbstractGhost` classes. To create an entity, a developer subclasses from `AbstractPlayer` and implements a `computeNextState` member function that returns the current state of the entity. At the remote host, the developer subclasses from `AbstractGhost` and implements the `processMsg` (process an incoming message) and `computeNextState` (compute the current state for the entity) member functions. The VERN system also allows the user to change the dead reckoning protocol error threshold in real-time.

The current state-of-the-art in dead reckoning algorithms raises several limitations. First, all existing protocols are tightly coupled to their underlying network environment. Most systems have been designed for use over a local-area network providing high reliability and predictable latency characteristics. Even the DIS protocol design, targeted for use over wide-area networks, does not directly address the variable performance of long-haul communication networks. Second, existing simulation protocols do not accommodate the variable modeling fidelity needs at each remote simulation host but instead associate a single dead reckoning error threshold with the source transmissions. As simulations contain increasing numbers of entities, hosts cannot afford to model all entities in full detail. Instead, the simulation needs to support a continuum from low-fidelity modeling to high-fidelity modeling so that individual hosts can select the appropriate level-of-detail based on local requirements. Finally, analyses of dead reckoning protocol behavior have concentrated almost exclusively on single entity types (tanks, fighter aircraft, drumsticks, etc.) These analyses do not offer a general-purpose technique for assessing the protocol's behavior over more general entity motion.

2.1.3.4 Related Techniques From Signal Processing

Dead reckoning in distributed simulation essentially represents a generalized signal prediction problem. Signal prediction is commonly used in speech processing (to extrapolate a speech pattern between discrete samples or to accommodate sample loss), economic prediction and forecasting, and medicine. We have already seen special-purpose uses of signal processing in sensor prediction for HMD position tracking.

The basic signal prediction problem can be formulated as follows [12]: A signal f is band-limited to the range $[-W\pi, W\pi]$ for some $W > 0$ (that is, f can be decomposed into the sum of sine waves with frequencies ranging up to $W\pi$). The signal f is sampled at regular intervals $\frac{1}{W}$. Based on n samples taken within the time interval $[t_0 - \frac{n}{W}, t_0]$, we wish to extrapolate the signal's behavior for time $t > t_0$. To address this problem, we construct a convolution series estimate for f:

$$(S_W^{\varphi} f)(t) \equiv \sum_{k=-\infty}^{\infty} f\left(\frac{k}{W}\right) \varphi(Wt - k) \tag{2.11}$$

In this expression, $\varphi(x)$ is a kernel function such that $\varphi(Wt - k) \neq 0$ only for $k \in [t_0 W - n, t_0 W]$, namely the interval for which samples of $f(\frac{k}{W})$ are available.

The simplest set of kernel functions can be derived [68] by applying the Fourier transform of the signal. In this case,

$$\varphi(Wt - k) = \frac{\sin((Wt - k)\pi)}{(Wt - k)\pi} \tag{2.12}$$

Observe that $\varphi(Wt - k)$ is 1 for $t = \frac{k}{W}$ and is 0 at all other sample points $t = \frac{j}{W}$ $(j \neq k)$. For example, with $n = 3$ samples, we estimate the curve as:

$$(S_W^{\varphi} f)(t) = \frac{\sin((Wt-1)\pi)}{(Wt-1)\pi} f\left(\frac{1}{W}\right) + \frac{\sin((Wt-2)\pi)}{(Wt-2)\pi} f\left(\frac{2}{W}\right) + \frac{\sin((Wt-3)\pi)}{(Wt-3)\pi} f\left(\frac{3}{W}\right) \tag{2.13}$$

However, the extrapolation error increases rapidly as t increases from $\frac{3}{W}$ toward $\frac{4}{W}$ because all terms approach zero expecting to rely on a $f\left(\frac{4}{W}\right)$ term to provide support.

A better set of functions φ can be derived by convolving a set of central B spline curves, with convolution coefficients computed by solving a linear system of equations (see [12] for a more complete description of the rather complex procedure). For example, to extrapolate a signal over a period of $\frac{1}{W}$ given $n = 3$ samples,

$$\varphi(Wt - k) = \begin{cases} 0, & t \leq \frac{k+1}{W} \\ 3(Wt - k) - 3, & \frac{k+1}{W} < t \leq \frac{k+2}{W} \\ 13 - 5(Wt - k), & \frac{k+2}{W} < t \leq \frac{k+3}{W} \\ 2(Wt - k) - 8, & \frac{k+3}{W} < t \leq \frac{k+4}{W} \\ 0, & \frac{k+4}{W} < t \end{cases} \tag{2.14}$$

If f'' is continuous and bounded, then we can derive a maximum error bound on the estimator:

$$||S_W^{\varphi} f - f|| \leq 15||f''||W^{-2} \tag{2.15}$$

In general, an estimator that relies on n samples has an error bound that is $O(W^{-n})$.

These approaches to signal estimation pose a number of limitations when applied to the simulation domain. First, the estimators rely on a uniform sampling frequency from the source and assume that we can derive some estimate for W (representing the maximum underlying frequency of the entity's motion, which is correlated to the motion's overall complexity). With dynamic entity motion, any fixed sampling frequency is liable to either oversample the entity position—and hence generate high network traffic—or undersample the entity position—and hence sacrifice accuracy in the remote model. In reality, the true value of W changes over time and cannot easily be estimated. Second, when compared to competing dead reckoning algorithms, evaluating the estimator function $S_W^\varphi f$ requires roughly twice the computation at each timestep t because all of the coefficient terms $\varphi(x)$ must be recomputed. Finally, the signal estimator assumes that the signal's second derivative is bounded and continuous. In many simulation domains, this assumption is not valid because the acceleration of physical objects may change almost arbitrarily.

One other approach to signal prediction employs a gradient descent method for finding the locally optimal coefficients in the prediction equation [56]. With this technique, however, computation time can vary dramatically on each time step. Consequently, this approach has limited utility in real-time systems, such as distributed simulation, which must process many entity updates within each frame interval.

2.2 Approaches to Supporting Large Distributed Simulations

Over the past decade, distributed simulations have grown in size, starting with the Amaze system (which included roughly four hosts connected over an Ethernet) to the STOW 97 system (originally envisioned to include 100,000 entities connected over a wide-area network). Larger simulations require more network bandwidth because each entity independently transmits state updates. In addition, each host requires more computational resources to receive the entity state update packets, model those simulation entities, and perform tasks such as scene rendering and collision detection. As discussed in Chapter 1, multicasting update packets to allow receiving hosts to filter data streams from entities that are not of local interest only partially addresses the bandwidth and computation problems. Each unfiltered entity must still provide updates at the rate needed by the host requiring the maximum remote modeling fidelity.

Simulation designers have attempted to reduce network bandwidth requirements and received packet rates in two ways: protocol-specific optimizations and entity aggregation.

2.2.1 Protocol Optimizations

Simulation protocols can often be optimized or specialized to reduce the bandwidth or packet rate. These optimizations are typically derived by analyzing the traffic generated in prior simulations to detect data duplication and other inefficiencies.

For example, the DIS protocol requires each simulation entity to periodically broadcast its complete state; this periodic broadcast makes DIS simulation resilient to packet loss or intermittent connectivity. However, analyses of DIS and STOW exercises have shown that up to 50% of the wide-area network traffic is generated by dead or otherwise inactive entities transmitting periodic state updates [98, 99]. Moreover, even for active entities, only a portion of the state (position, orientation, etc.) changes frequently, so the traffic contains considerable information that was previously transmitted.

To address these issues, newer simulations such as the STOW program, optimize the basic DIS protocol by placing an *Application Gateway* (AG) [13] on each LAN. The AG is responsible for managing the flow of information between the LAN and the tail circuit/backbone. For example, because must of the traffic in DIS simulations consisted of redundant updates from inactive entities, the AG provides a Quiescent Entity Service (QES). The AG detects when a local entity has become inactive and reliably informs the other AGs. Until the entity becomes active again, the local AG blocks its update packets from the WAN, and each AG is responsible for locally generating state updates on the entity's behalf.

Most aspects of an entity's state do not change frequently, so successive entity update packets contain redundant information. Early versions of STOW AGs implemented the Protocol Independent Compression Algorithm (PICA) [100] to eliminate this redundant state information from the DIS packets. Each receiver maintains a numbered "reference" state for each entity. The AG receives the state updates generated by each entity and only transmits the bitwise difference between the entity's current state and its reference state over WAN. When the length of these difference packets exceeds a threshold, the AG transmits a new reference state with a new sequence number. Because each difference packet includes the sequence number of its corresponding entity reference state, receivers can detect lost reference state packets and request retransmission from the AG.

The Log-Based Receiver-Reliable Multicast (LBRM) [33] protocol is optimized to support inactive entities that generate occasional updates. While inactive, each entity generates low-frequency heartbeat packets, but after transmitting an update packet, it transmits heartbeat packets at a high rate to ensure rapid packet loss detection. Other optimizations, such as statistical acknowledgement and distributed logging, allow LBRM to quickly detect large-scale loss and provide faster packet loss recovery. Based on its update frequency, an entity would therefore either use periodic unreliable multicast or an occasional reliable multicast. Zelesko and Cheriton [106] present a framework for optimizing protocols to enable such optimizations based on functionality and performance requirements.

2.2.2 Entity Aggregation

Entity aggregation attempts to merge a group of entity updates into a single update packet, thereby reducing packet header overhead in the network and reducing packet-processing overhead at receivers. An *aggregation* is a logical simulation entity representing a group of other entities. For example, an aggregation may be used to represent a battalion of tanks. Alternatively, an aggregation entity might represent all simulation entities located within a particular region of the virtual world. The merged update packet transmitted by an aggregation entity may either simply bundle the component entity updates [13] or summarize the component entities using an entirely new representation.

A key challenge in entity aggregation is determining which entities to group together. Three approaches have been used in previous distributed simulation systems: network-based, organization-based, and grid-based.

2.2.2.1 Network-Based Aggregations

Network-Based Aggregations group simulation entities by their physical location in the network [13]. For example, all entities located at a single site or on a single LAN may form the basis for an aggregation. This aggregation approach is best suited for environments in which the wide-area network or network tail-circuits represent the primary bandwidth bottleneck. However, entities on a LAN need not share any relationship to one another, either in terms of entity type or entity location within the virtual world. A receiver who subscribes to the aggregation would typically receive a considerable volume of information from entities that are of no local interest. Consequently, network-based aggregations are most beneficial only when there is some correspondence between the entity locations in the virtual world and their physical locations.

2.2.2.2 Organization-Based Aggregations

Organization-Based Aggregations group simulation entities by their organizational hierarchy (armies, brigades, battalions, platoons, etc.) [23]. Though easy to construct and maintain, this aggregation structure offers limited value because each organization's member entities may travel within different regions of the virtual world. For example, a battalion might divide into two sub-units. Even if undivided, large aggregations might be spread over a broad region. Also, destroyed tanks become separated from the live platoon members as the battle advances: In military simulations, for example, up to half of the simulation entities are destroyed. However, for common operations such as collision detection and scene rendering, each host wants data about all entities located within a nearby region of the virtual world. If only organization-based aggregations are available, the host must subscribe to information from all organizations represented within that region, even though most of the organizations' member entities may actually be far from the viewer. Consequently, organization-based aggregations are most beneficial only when there is some correspondence between the static entity organization and the dynamic entity location within the virtual world.

2.2.2.3 Grid-Based Aggregations

Grid-Based Aggregations group simulation entities by their location within the virtual world. The virtual world is divided into rectilinear or hexagonal grids[1] whose associated aggregation transmits packets bundling information about entities in that region. Most existing implementations of grid aggregations [54, 52, 86, 60] dispense with a designated aggregation entity and instead simply associate a multicast address to each grid. Each entity transmits updates to the multicast group associated with its current virtual world location, so although the data is not bundled into the same packet, the multicast group allows remote hosts to select the virtual world region(s) of interest.

Grid-based aggregations pose several disadvantages. They mask the organizational relationships between the various entities. For example, if a host only provides summary views of a tank battalion to a commander, then it must subscribe to information from all regions that potentially contain one of those tanks, even though each grid contains numerous entities that are not of local interest. Grid-based aggregations and similar *Area of Interest* (AOI) techniques do not allow remote hosts to receive information at different levels-of-detail depending on the entity type—a capability that can be desirable for rendering regions containing many heterogeneous entities. Moreover, establishing an optimal grid size for use by all simulation hosts is difficult because the ideal grid size depends on the amount of inter-host interaction in the simulation scenario and on the number of entities running on each host [73]; a poor grid size selection can affect network bandwidth requirements by up to 150%, and even the alignment of grids with respect to the coordinate system origin can affect data

[1] Ideally, these regions would be determined dynamically based on a "clustering analysis," but this approach is infeasible in real-time simulations because of the NP-hard nature of the dynamic clustering problem with constantly moving entities.

traffic by 15% or more. Because grid-based aggregations do not allow hosts to access entities by their organization or type and because of the configuration complexities, they have limited value for reducing network traffic or computational load.

A simple dynamic grid-based aggregation scheme was deployed by Schilit and Theimer [80] to address data distribution requirements in mobile computing environments. In this environment, servers are responsible for providing continuous information to clients who have subscribed to various types of data. The server dynamically monitors the destination for each piece of data, detects when multiple data streams are sending information to the same client set, and creates a multicast group to aggregate those transmissions. Ideally, a distributed simulation might apply a similar technique to aggregate information from multiple entities when the receiver sets are substantially similar. However, their implementation has limited value in our domain because it only detects exact matches among the data distribution groups and because the linear searching required for each data transmission does not not scale effectively.

2.2.2.4 Multidimensional Data Cubes

Database vendors are starting to offer tools to support *On-Line Analytical Processing* (OLAP), a method for navigating and analyzing complex data [88, 46]. OLAP systems present the user with the illusion of a multi-dimensional spreadsheet, with each dimension representing a different category for grouping information. However, unlike traditional spreadsheets whose axes are linear in nature, OLAP dimensions are hierarchical. For example, in an inventory database, one dimension may represent stores which are organized into a hierarchy of territories, regions, etc., while a second dimension would represent product type organized into another hierarchy. Users access a particular "cube" in the database by selecting a node along each dimension. For each cell, the OLAP tool provides summary information for the underlying data. For example, a user might select "Western Region" and "Casual Shoes" to receive a summary of the casual shoe inventory in the western region. Finally, the OLAP tool provides "drill down" and "roll up" capabilities along each dimension. Hence, the user can expand the current cube to study the casual shoe inventory within each sales region of the western territory, or he may expand the cube to explore the inventory of different types of casual shoes.

Implementations of OLAP systems are quite disparate [20]. Some systems compute the aggregate information for all cubes at an OLAP server, so that client requests can be serviced immediately. However, this approach is not scalable because of the exponential number of possible cubes in a complex data set. Consequently, such cube generation is typically done nightly on a batch basis, so the aggregation does not necessarily accommodate real-time information sources. Newer systems slated for release throughout 1996 are attempting to generate cubes dynamically, either by collating information from a data cache at the client host or by transmitting the cube request to an OLAP server host that generates an SQL query to the database server and then collates the resulting records. The performance of these dynamic approaches is generally poor (response time is measured in minutes). Moreover, they do not support dynamic updating of the aggregated information as data values in the underlying database change.

Despite the relative infancy of the deployed implementations, the "multi-dimensional cube" model is a powerful model for aggregating complex information, and in many ways, it addresses many of the limitations imposed by network-based, organization-based, and grid-based aggregations. In particular, it offers clients the ability to select the dimension(s) of interest, rather than demanding a single criterion for grouping data.

2.3 Conclusion

We have described a representative sample of approaches to remote modeling and rendering, revealing the considerable variety of techniques used by existing distributed simulation systems, We have also described how recent systems attempt to support high entity counts within the bandwidth, latency, and computational constraints imposed by a wide-area network environment.

Our survey of existing simulation systems reveals the following general characteristics:

- Most simulation systems are designed for networks providing high bandwidth and low latency and jitter, and they usually operate among a small number of homogeneous hosts. Such systems, which rely on shared database consistency or frame-rate updates, are not suitable for large simulations on wide-area networks.

- Support for high entity counts is typically added only after a system has been built and deployed on a LAN environment. This support has often involved ad hoc techniques, such as application gateways and network-based aggregation, to reduce the bandwidth demands imposed by the original system design.

- The current trend is toward remote modeling protocols that are tailored for the particular entity types being modeled. These algorithms are rarely usable in other systems.

- No common methodology has emerged for evaluating the performance of remote modeling protocols. Instead, previous analyses have only concentrated on a small class of entity behaviors

- Existing aggregation techniques often group unrelated entities together, thereby limiting the bandwidth reduction gained by combining entity updates.

The techniques described in this thesis take a different approach from most existing work: Develop a general-purpose remote modeling protocol for large-scale simulations over wide-area networks, and provide support for entity-specific customization within this basic framework. The Position History-Based Dead Reckoning (PHBDR) protocol provides fast, accurate remote modeling of a scalar value based only on a history of previous updates to that value. It makes no assumptions about the value being modeled other than that it is continuous,[2] though entity-specific constraints can always be introduced to optimize the remote modeling accuracy. By modeling three scalars, hosts can represent the remote entity's position in the virtual world. To provide scalability, PHBDR packets are small, and its computational requirements are minimal. Finally, its remote modeling tolerates network latency and jitter.

The PHBDR protocol provides a base for developing more sophisticated remote modeling protocols. The Axis Point protocol uses PHBDR to model remote entity orientation. Multiple-Detail Channels (rigid-body, approximate-body, and full-body) use PHBDR to model non-rigid entity structure at different levels-of-detail. Projection Aggregation Entities use PHBDR to model entity groups. As demonstrated in [16] and [106] with transport-level protocols, this *recursive protocol structuring* keeps the design simple, eases testing of protocol implementation, and simplifies the analysis of protocol effectiveness.

Table 2.1 shows how the techniques in this thesis are used to support different entity types at different fidelities. In summary, Position History-Based Dead Reckoning and its derivative protocols provide an integrated architecture for remote rendering in large distributed simulation environments. Moreover, we demonstrate the effectiveness of these protocols in arbitrary simulation environments by analyzing their network behavior and modeling accuracy. This domain-independent analysis represents a significant departure from previous work, which has either neglected analysis altogether or chosen to focus on particular entity behaviors.

[2] That is, the scalar's value, when graphed as a function of time, is C^0.

Entity Type	**Remote Modeling Fidelity**		
	Low	Medium	High
Rigid	PHBDR	PHBDR Axis Point	PHBDR Axis Point
Semi-Rigid	PHBDR Axis Point Rigid-Body Channel	PHBDR Axis Point Approximate-Body Channel	PHBDR Axis Point Full-Body Channel
Non-Rigid/ Entity Group	Projection Aggregation Entities	Projection Aggregation Entities	PHBDR Axis Point Full-Body Channel

Table 2.1: Use of the Techniques Presented in this Thesis

Chapter 3

Position History-Based Dead Reckoning (PHBDR)

In this chapter, we describe a protocol for accurately modeling the real-time position of remote entities and for generating smooth graphical representations of those entities on the user's display. The Position History-Based Dead Reckoning (PHBDR) protocol [83, 84] provides remote modeling of a scalar value (and by extension, the position of a single vertex). The protocol implementation only assumes that the value being modeled is continuous, hosts have near-synchronous clocks, and the network offers a unidirectional datagram service. To derive the greatest benefit from the protocol in a large system, the network should provide some sort of multicast service, but this feature is not absolutely essential. The protocol makes no other assumptions about the simulation domain, the type of entity being represented, or the network performance.

Furthermore, the PHBDR protocol reduces the real-time dependencies between hosts as much as possible by transmitting less time-sensitive information over the network than existing techniques. Receivers process the received information independently based on the locally perceived latency. This decoupling is essential for supporting large-scale simulations operating over wide-area networks.

We begin by providing an overview of the PHBDR protocol. We describe how source hosts generate update packets from the true entity model. We then describe how remote hosts use those update packets to produce a dead reckoned representation of entity position. We conclude the chapter with a discussion of how a simulation designer selects appropriate values for various PHBDR protocol parameters

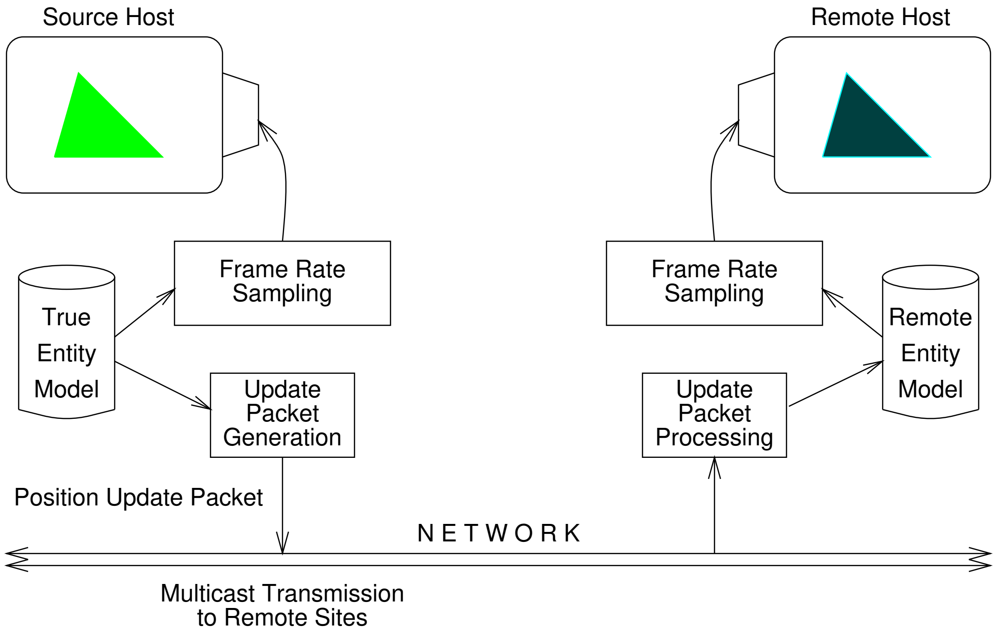

Figure 3.1: Hosts Multicast Position Update Packets for Local Entities

3.1 Overview of the PHBDR Protocol and Algorithm

The position history-based protocol transmits timestamped packets containing three scalar coordinates describing the entity position along the x, y, and z axes. The protocol is supported by an algorithm at the source host for generating update packets and a dead reckoning algorithm at the remote host for processing update packets, as illustrated in Figure 3.1. The source host maintains an entity model which is sampled to generate its local frame-rate display. The host also periodically transmits the entity's current position over an associated multicast address to remote sites across the network. Based on information in these update packets, the remote host maintains a remote entity position model which is sampled at the local frame rate. This dead reckoning algorithm allows remote hosts to generate a visually accurate animation of remote entities in spite of typical network delays.

3.2 Source Generation of Updates

Figure 3.2 depicts the processing performed at the source host. The host concurrently maintains two models of each entity: 1) the true position, which is determined by user input, autonomous control, and external forces; and 2) the remote tracking position (described in the next section)

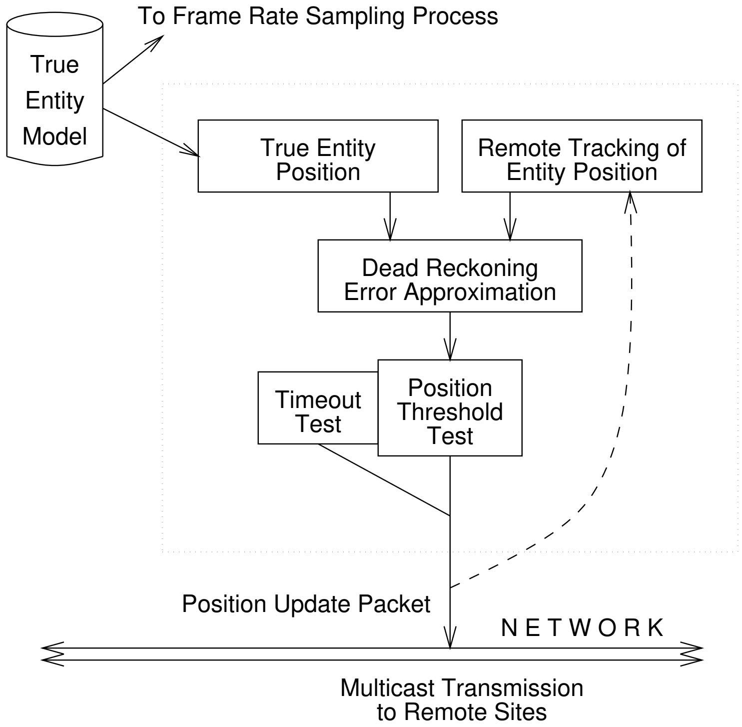

Figure 3.2: Update Packet Generation at the Source Host

which estimates the position based on previous network updates. By calculating the difference between these two positions, the source host therefore approximates the dead reckoning error at remote hosts. A maximum error threshold is associated with each entity, and the host transmits an update packet for the entity whenever the estimated dead reckoning error exceeds this threshold. The source transmits an update for the entity if none is otherwise generated within a timeout period to limit the time a remote host relies on old information when a packet is lost. This use of an error threshold is similar to that used by SIMNET [69], DIS [36], and NPSNET [70].

An update packet only reports the entity's current position along each axis. Remote sites use this absolute state information to estimate the entity's position, velocity, and acceleration. Each packet includes a timestamp which is used by receivers to account for transmission latency. The algorithm assumes accurate clock synchronization between all participating hosts. Implementations of distributed clock synchronization algorithms, such as NTP [58], are widely available and provide accuracy to less than one millisecond [59]—enough for most real-time simulations.

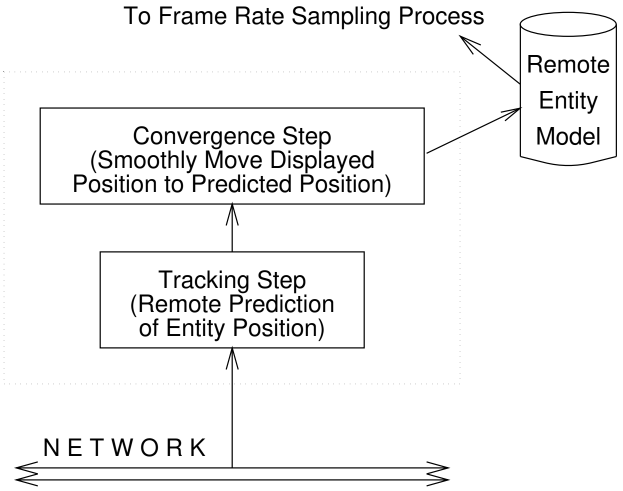

Figure 3.3: Dead Reckoning of Entity Models at the Remote Host

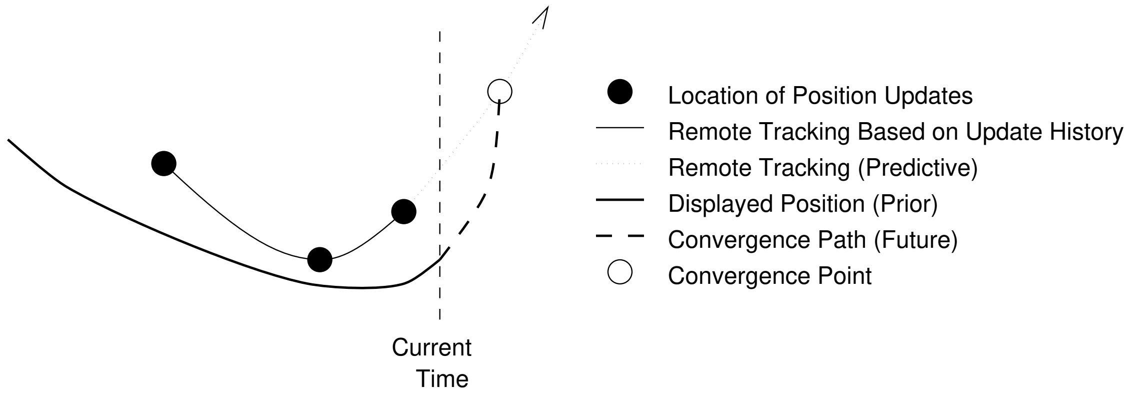

Figure 3.4: Tracking and Convergence Steps at Remote Sites

3.3 Receiver Processing of Updates

Upon receiving an update packet, a remote host performs the steps pictured in Figure 3.3 to update its model of the entity. The first phase, the *remote tracking step*, uses a short history of updates to predict the entity's position, velocity, and acceleration until the next update arrives. This step compensates for network latency, network jitter (variation in latency), and the inability to transmit update packets at high frequencies. The dotted line in Figure 3.4 indicates the predicted entity path generated by the tracking step. The second phase, the *convergence step* adjusts the local estimate of velocity and acceleration so that the entity's current displayed position smoothly converges to the predicted path at the *convergence point*, as shown by the dashed line in Figure 3.4. Because

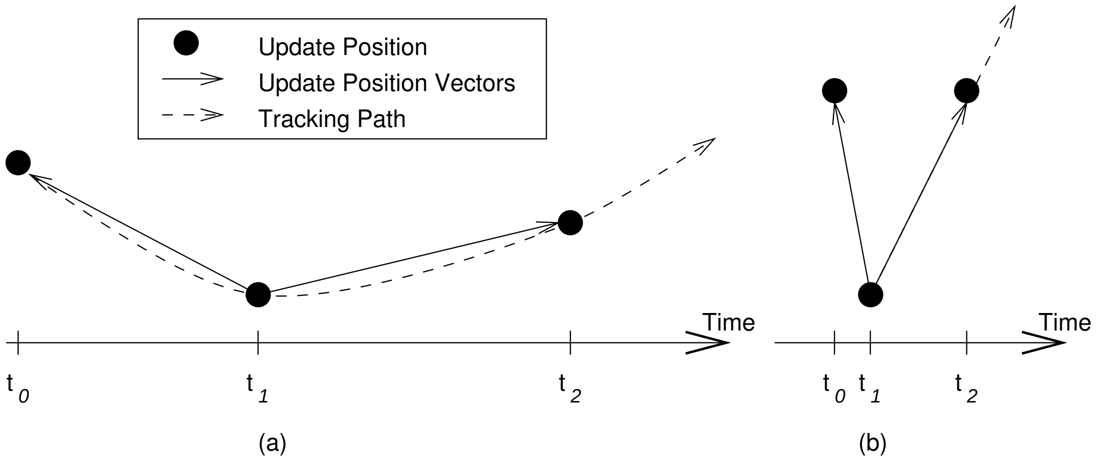

Figure 3.5: Angle of Embrace Determines Adaptive Tracking

the host predicts the entity's future location, the current displayed position usually differs from the true position provided in the update packet. Rather than directly "jumping" to the correct position, smooth convergence provides the viewer with a more natural animation of remote entities.

Instead of using fixed tracking and convergence algorithms, the position history-based protocol adapts the algorithms to the recent behavior of the remote entity.

3.3.1 Adaptive Tracking Algorithm

The tracking algorithm adapts to the entity's behavior by using either a second-order (parabolic) estimation between the three most recent updates or a first-order (linear) estimation between the two most recent position updates, as shown in Figure 3.5. To determine which of the two tracking techniques to use, the remote host calculates the angle between the three most recent update positions. This angle, defined in differential geometry as the *angle of embrace*, estimates the local Gaussian curvature of the entity's path [17].[1] A large angle of embrace—implying that the entity is not changing direction significantly—invokes second-order (quadratic) tracking along each dimension, as shown in Figure 3.5a. On the other hand, a small angle—implying that the entity has recently made a significant change in direction—invokes first-order (linear) tracking along each dimension,

[1]For three updates x_0, x_1, and x_2, the angle of embrace is computed by computing the vectors $\vec{x_1 x_0}$ and $\vec{x_1 x_2}$, normalizing them, taking their dot product, and applying the arc-cosine:

$$\cos^{-1}\left(|\vec{x_1 x_0}| \cdot |\vec{x_1 x_2}|\right)$$

as shown in Figure 3.5b.

The second-order tracking technique is used when the entity is not changing direction rapidly. If the three most recent positions along a particular dimension are x_0 at time t_0, x_1 at time $t_1 = t_0 + d_{01}$, and (the most recent update) x_2 at time $t_2 = t_1 + d_{12}$, then the remote entity tracking traces a parabola joining those three position points. Using Aitken's algorithm [22][2] to interpolate these points, we derive an initial position at time t_2 of $x(t)\Big|_{t=t_2} = x_2$, initial velocity

$$x'(t)\Big|_{t=t_2} = \frac{d_{12}}{(d_{01}+d_{12})d_{01}}x_0 - \left(\frac{12}{d_{01}} + \frac{1}{d_{12}}\right)x_1 + \left(\frac{1}{d_{12}} + \frac{1}{d_{01}+d_{12}}\right)x_2 \tag{3.1}$$

and acceleration of

$$x''(t) = \frac{2}{d_{01}(d_{01}+d_{12})}x_0 - \frac{2}{d_{01}d_{12}}x_1 + \frac{2}{(d_{01}+d_{12})d_{12}}x_2 \tag{3.2}$$

By common subexpression elimination, these parameters are computed in eight multiplications and four additions. This formulation has the advantage of being invariant to affine transformations (in particular, coordinate system translation and rotation).

We assume that the host processes the most recent update at time $t_2 + d_{delay}$, where d_{delay} represents the latency introduced by the network. To achieve a smooth visual effect, we set the displayed position to converge with the tracked position after a convergence period $d_{cp} = d_{12}$ seconds. We therefore treat the interval between the two most recent update packets as a crude estimate of packet rate with the intention that path correction will be complete when the next update packet arrives (assuming a roughly constant update rate).[3] The convergence point (at time

[2]The Aitken algorithm operates by recursively generating lower-order polynomials for subsets of the given points and then convolving those interpolating polynomials together to construct an interpolation for the entire point set. In the case of three points x_0, x_1, and x_2 at respective times t_0, t_1, and t_2, we produce two functions:

$$f_{01}(t) = \frac{t_1 - t}{t_1 - t_0}x_0 + \frac{t - t_0}{t_1 - t_0}x_1$$

and

$$f_{12}(t) = \frac{t_2 - t}{t_2 - t_1}x_1 + \frac{t - t_1}{t_2 - t_1}x_2$$

and then convolves them to get the interpolating polynomial:

$$f_{012}(t) = \frac{t_2 - t}{t_2 - t_0}f_{12}(t) + \frac{t - t_0}{t_2 - t_0}f_{01}(t)$$

The independence to affine coordinate transforms results from the fact that all curves are rooted (at the base case) on linear interpolation, which is invariant to coordinate system rotation.

[3]One can envision using a more complex packet rate estimation technique, possibly based on a smoothed average of the inter-arrival times between the last few packets. However, because of the dynamic nature of the entity motion, it is not

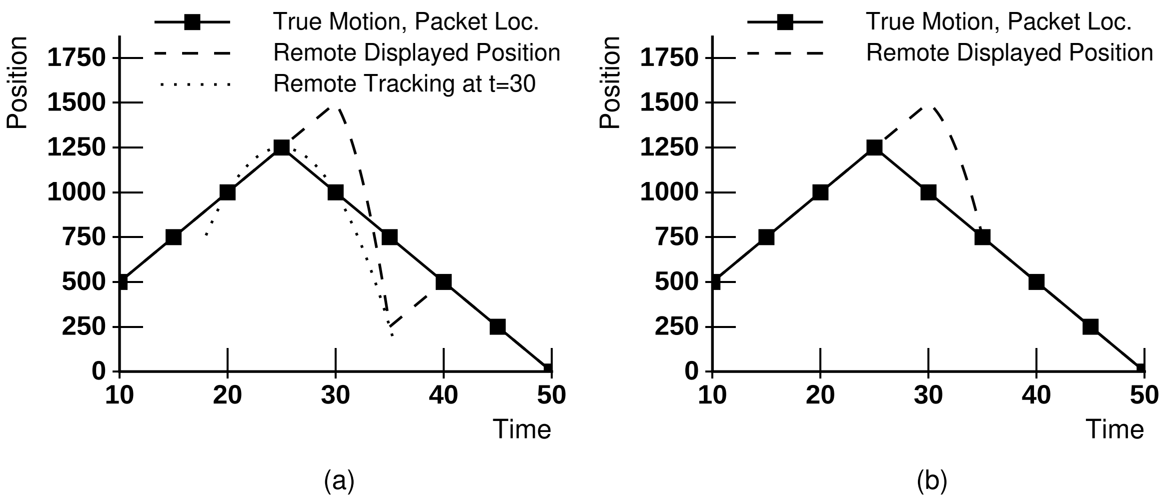

Figure 3.6: Using Adaptive Tracking to Model Sudden Path Changes Using (a) Three and (b) Two Previous Position Updates

$t_{cp} = t_2 + d_{delay} + d_{cp}$) for second-order tracking is therefore

$$\begin{aligned} x(t)\Big|_{\tau = t_{cp}} &= (d_{delay} + d_{12}) \left(\frac{d_{delay} + 2d_{12}}{d_{01}(d_{01} + d_{12})} x_0 \right. \\ &\quad \left. - \frac{d_{01} + 2d_{12} + d_{delay}}{d_{01}d_{12}} x_1 + \frac{d_{01} + 3d_{12} + d_{delay}}{(d_{01} + d_{12})d_{12}} x_0 \right) + x_0 \quad (3.3) \end{aligned}$$

The first-order tracking technique is used after the entity makes a sudden turn, as might happen after a collision. In this situation, second-order tracking as described above is inaccurate, because the older update position provides little information for predicting the future position of the entity. The dotted line in Figure 3.6a shows that a second-order curve over-compensates for the turn and introduces new error in the remote tracking; the resulting displayed path, represented by the dashed line, does not reflect the entity's true behavior. A first-order approach, on the other hand, is more accurate because it ignores information from before the turn, as shown in Figure 3.6b. The resulting displayed path converges to the true position more rapidly, as reflected in the dashed line. To generate a first-order tracking equation, we apply straightforward linear interpolation between the

clear whether such effort would yield a packet arrival estimate that is better enough to justify the additional computational costs. The use of the most recent packet arrival distance seems to provide a reasonable engineering tradeoff between estimation and computation.

two most recent position updates; the tracking curve therefore has initial position at time t_2 of

$$x(t)\Big|_{t=t_2} = x_2 \tag{3.4}$$

and velocity

$$x'(t) = \frac{1}{d_{12}}x_2 - \frac{1}{d_{12}}x_1 \tag{3.5}$$

These parameters are computed in one multiplication and one addition. Once again, the tracking path is invariant to affine transformations in to coordinate system.

We set the displayed position to converge with the tracked position after a convergence period of $d_{cp} = \min(d_{12}, \Delta_{max-cp})$ seconds after the latest update is processed at the host, where Δ_{max-cp} represents an upper bound on the convergence period. This upper bound is necessary because the most recent update packet may have resulted from an unusual event (such as a collision), so the inter-packet arrival time does not necessarily provide any information about when another update is likely to arrive. Moreover, in the case of a collision, the correction should occur reasonably quickly because discrepancies in the visual effect are more important to the viewer. The resulting convergence point (at time $t_{cp} = t_2 + d_{delay} + d_{cp}$ is therefore either

$$x(t)\Bigg|_{t\,=\,t_2 + d_{delay} + d_{12}} = -\left(\frac{d_{delay}}{d_{12}} + 1\right)x_1 + \left(\frac{d_{delay}}{d_{12}} + 2\right)x_2 \tag{3.6}$$

or

$$x(t)\Bigg|_{t\,=\,t_2 + d_{delay} + \Delta_{max-cp}} = -\frac{d_{delay} + \Delta_{max-cp}}{d_{12}}x_1 + \left(\frac{d_{delay} + \Delta_{max-cp}}{d_{12}} + 1\right)x_2 \tag{3.7}$$

The adaptive tracking algorithm accounts for network latency by processing position updates as if they had arrived at the time they were actually sent. In effect, the receiving host rolls back the tracking step to the packet transmission time. This use of timestamps reduces the real-time dependencies between remote host dead reckoning and network performance by effectively providing loose "eventual consistency" semantics on the entity state information at remote hosts. For example, although different hosts might encounter different latencies on each update packet or some hosts might receive update packets out-of-order, all hosts eventually track the entity in the same manner (modulo packet loss) because the tracking algorithm processing is determined by the packet's absolute timestamp. Furthermore, the effects of packet loss are eliminated over time as that information is outdated by new updates. Because of this common tracking at all remote sites, the source host can reasonably approximate error in the remote tracking position and therefore generate appropriately timed update packets. Convergence, on the other hand, begins from the packet arrival time; hence, although all of the remote tracking models are identical, each host applies a different convergence model based on the packet delays observed locally.

3.3.2 Adaptive Convergence Algorithm

While the tracking algorithm described in the previous section allows remote hosts to model the entity's actual position, the convergence algorithm ensures that the locally displayed entity position is smooth. The convergence algorithm also adapts to the entity's recent behavior. Based on the angle of embrace calculated during the tracking step, the convergence algorithm selects between a first-order and second-order path. A smaller angle indicates that the entity motion is curved, and a second-order (constant acceleration) path is therefore generated. A large angle indicates that the entity motion is nearly linear, so a first-order (constant velocity) path is used.

Second-order convergence uses the Aitken algorithm described earlier to generate a smooth parabolic curve between the entity's previous absolute position, the current displayed position, and the convergence point on the tracked path,[4] as shown in Figure 3.7a. If the previous absolute position is x_1 at time $t_1 = t_0 + d_{01}$, the current displayed position is $\overline{x_{2+delay}}$ at time $t_2 + d_{delay}$, and the convergence point (determined in the previous section) is $\overline{x_{cp}}$ at time t_{cp}, then convergence has initial position $\left.\overline{x(t)}\right|_{t=t_2+d_{delay}} = \overline{x_{2+delay}}$, initial velocity

$$\begin{aligned}\left.\overline{x'(t)}\right|_{t = t_2 + d_{delay}} &= \frac{-d_{cp}x_1}{(d_{12} + d_{delay})\,(d_{12} + d_{delay} + d_{cp})} \\ &+ \left(\frac{1}{d_{12} + d_{delay}} - \frac{1}{d_{cp}}\right)\overline{x_{2+delay}} \\ &+ \frac{(d_{12} + d_{delay})\,\overline{x_{cp}}}{(d_{12} + d_{delay} + d_{cp})\,d_{cp}} \end{aligned} \tag{3.8}$$

[4]We could have used the Aitken algorithm to generate a smooth convergence path between the currently displayed path and the tracking path; this would result in a third-order path preserving a smooth velocity. However, should an update packet arrive before convergence completes, we would need to generate a fourth-order curve to continue providing a smooth convergence (i.e. between the current convergence path and the new tracking path). Ultimately, either the degree of the convergence path must remain unbounded, or the user must still accept an occasional discontinuity in the velocity. Credit goes to Hugh Holbrook for this observation.

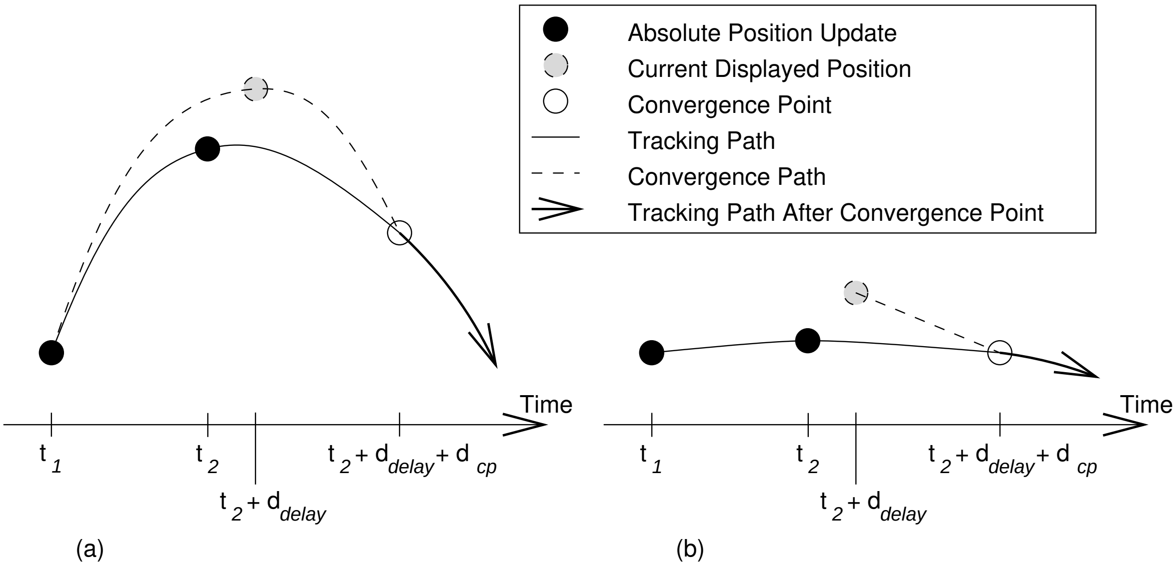

Figure 3.7: Convergence Smoothly Corrects the Current Displayed Position

and acceleration

$$\overline{x''(t)} = \frac{2x_1}{(d_{12} + d_{delay})\,(d_{12} + d_{delay} + d_{cp})} - \frac{2\overline{x_{2+delay}}}{(d_{12} + d_{delay})\,d_{cp}} + \frac{2\overline{x_{cp}}}{(d_{12} + d_{delay} + d_{cp})\,d_{cp}} \tag{3.9}$$

By common subexpression elimination, the parameters are computed in nine multiplications and six additions.

When the motion is almost linear, first-order convergence joins the current displayed position to the convergence point as shown in Figure 3.7b. Using first-order convergence, the initial position is $\overline{x(t)}\Big|_{t=t_2+d_{delay}} = \overline{x_{2+delay}}$ and the velocity is

$$\overline{x'(t)} = \frac{\overline{x_{cp}} - \overline{x_{2+delay}}}{d_{cp}} \tag{3.10}$$

These parameters are calculated in one multiplication and one addition.

The convergence process terminates once the entity's displayed position reaches the convergence point. Until the next update arrives, the displayed entity follows the position, velocity, and acceleration predicted by the remote tracking algorithm.

Table 3.1 summarizes how angle of embrace determines the adaptive tracking and convergence algorithms.

Angle Between Three Recent Updates	Curve Characterization	Tracking Model	Convergence Model
Small	Rapid Turn	First-Order	Second-Order
Medium	Smooth Curve	Second-Order	Second-Order
Large	Straight Line	Second-Order	First-Order

Table 3.1: Adaptive Algorithms for Extrapolation and Convergence

Parameter	Source/Remote Agreement Needed	Characteristic	Setting
Small–Medium Angle Threshold	Yes	Entity changes behavior frequently	High
		Entity behavior stable	Low
Medium–Large Angle Threshold	No	Ample CPU, entity of local interest	High
		Limited CPU, entity not of local interest	Low
Δ_{max-cp}	No	Entity cannot make sharp turns	High
		Entity motion may be arbitrary	High

Table 3.2: Criteria for Setting PHBDR Tracking and Convergence Parameters

3.4 Setting Dead Reckoning Parameters

The PHBDR protocol requires that four parameters be set for each entity: an error threshold (used by the source to decide when to transmit an update), small–medium and medium–large angle thresholds (used by the receiver to select between first- and second-order tracking and convergence, and Δ_{max-cp} (used by the receiver to bound the convergence time during linear extrapolation). The appropriate values for these parameters depend on the physical dynamics of the particular entity being modeled. In this section, we provide some guidance on setting the small–medium and medium–large angle thresholds and Δ_{max-cp}. These parameters are summarized in Table 3.2. The remaining protocol parameter, the error threshold, is discussed in depth in the next chapter in conjunction with discussions of the network load and remote modeling error produced by PHBDR.

3.4.1 Setting the Small–Medium Angle Threshold

The "small" angle category signals to the remote host's tracking algorithm that the entity's motion represents a collision rather than smooth motion. Consequently, entities capable of experiencing frequent changes in behavior (as a result of rapidly changing acceleration, for instance) require a high small–medium angle threshold, thereby forcing most of the motion to fall within the first-order collision tracking category. For example, a bee, which changes direction frequently, should have a high small–medium angle threshold to ensure that first-order dead reckoning is used on almost all packets; a passenger aircraft, which changes direction rarely, should have a low small–medium angle threshold to ensure that second-order (smooth) dead reckoning is used on almost all packets.

The particular entity type determines the appropriate threshold value, and because its value affects the tracking path, the source and remote hosts must agree upon its value. Typically, therefore, remote hosts would obtain its value from a simulation database which also provides, for example, graphical descriptions of the entities. Empirically, we have found that setting the small–medium angle threshold at 90 degrees provides a conservative parameter that minimizes the number of collisions assumed by the tracking algorithm.

3.4.2 Setting the Medium–Large Angle Threshold

The "large" angle category signals that the convergence algorithm at the remote host may conserve computational processing by replacing a smooth curve path with a straight line path without distorting the visual effect. The value of this threshold therefore depends on the entity's distance from the viewer and the entity type (together representing its importance to the viewer), as well as the locally available computational resources.

Because this value is only used in convergence, each remote site may independently adjust the threshold for each entity based on local requirements. When computational resources are not an issue, setting the medium–large angle threshold at 175 degrees eliminates almost all visual distortion.

3.4.3 Setting Δ_{max-cp}

The parameter Δ_{max-cp} bounds the convergence period after the dead reckoning algorithm detects a sudden change in the entity motion; this parameter is necessary because sharp turns and collisions often occur without warning, so the normal packet inter-arrival time potentially yields a long convergence period and an unrealistic path. Consequently, smaller values for Δ_{max-cp} allow the convergence algorithm to generate sharper turns on the display, while larger values force the convergence algorithm to generate smoother convergence paths.

The appropriate value for Δ_{max-cp} therefore depends on how sharp are the turns that the entity is capable of making. A ball, for instance, can move almost arbitrarily, so a small value of Δ_{max-cp} is appropriate. On the other hand, a human typically does not make sharp turns (like a complete reversal of direction), so a larger value of Δ_{max-cp} is desirable. Empirically, we have found that setting Δ_{max-cp} to 0.25 seconds produces a reasonable visual effect in most cases, though this value depends on the particular entity type.

3.5 Conclusion

We have presented the Position History-Based Dead Reckoning (PHBDR) protocol which provides a simple, efficient protocol for remote modeling of entity position. The PHBDR protocol exhibits several novel characteristics:

- Source hosts only transmit position information, and remote sites use position history from multiple updates to accurately track entity position.
- A timestamp indicating time of generation is included in the protocol packet, allowing the receiver's dead reckoning process to compensate for variable delay in packet delivery.
- The adaptive tracking and convergence characterize the entity's overall behavior to determine how many updates to include in the curve-fitting process.

The protocol makes minimal assumptions about the simulation environment. It only assumes that the entity's position is continuous, simulation hosts have synchronized clocks, and the network provides a unidirectional datagram service. It only relies on entity position information which, unlike velocity and acceleration, is universally available from entity models, including models derived from sensors attached to physical entities. These assumptions allow the PHBDR protocol to be used in a broad range of simulation environments.

Having provided a simple, efficient dead reckoning protocol, we now proceed to evaluate its network bandwidth utilization and positional fidelity. The next chapter demonstrates that PHBDR provides smooth and accurate remote modeling for a broad range of entities while using minimal network bandwidth.

Chapter 4

Analyzing PHBDR Network and Error Performance

We have presented the Position History-Based Dead Reckoning (PHBDR) protocol, a simple, efficient protocol to support remote modeling in distributed simulations. However, the protocol is only useful if it requires minimal network bandwidth utilization and offers smooth, accurate remote modeling.

Assessing the behavior of a dead reckoning algorithm is difficult because it depends heavily on the type of entity motion, the network latency between the source and remote hosts (which is affected by the distance between hosts), and network *jitter*—the variation in network latency (which is affected by the variability in network congestion).

To evaluate a dead reckoning protocol systematically, we use a three-step process:

1. Classification of curves into groups representing the different types of entity behavior of interest.

2. Mathematical analysis of the protocol to understand its worst-case modeling behavior on each curve type, followed by controlled simulation of the protocol over selected entity paths from each curve category to evaluate its performance over the expected set of entity behaviors.

3. Deployment of the protocol in a distributed simulation application to validate the evaluation and confirm the visual quality of the remote modeling from a user's perspective.

This chapter applies this procedure to analyze and evaluate the PHBDR protocol. We begin by describing the curve classification used in the evaluation and argue that this classification is broad

Jerk Magnitude	Jerk Smoothness	Sample Motion	Algorithm(s) Responsible For Providing Most Accurate Remote Display
None	N/A	Line, Parabola	Tracking
Low	No Spikes	Near Parabola	Tracking, Convergence
High	No Spikes	Oscillation	Tracking
Low	Spikes	Bouncing	Convergence
High	Spikes	Random	Neither

Table 4.1: Summary of Five Curve Classes

enough to describe most behaviors likely to arise in a simulation. After analyzing the PHBDR performance on each curve type, we incorporate experience in a deployed simulation environment to summarize the key advantages and limitations of the PHBDR protocol.

4.1 Entity Path Classification

To systematically evaluate the PHBDR protocol, we divide entity behavior into five categories, as listed in Table 4.1, This classification is based on the third derivative, or *jerk*—the change in acceleration exhibited by the entity. The third derivative most significantly affects the protocol's positional fidelity because PHBDR generates first-order and second-order curves for both tracking and convergence. Our classification covers all combinations of jerk magnitude and smoothness and is therefore all-inclusive. Although an entity's motion is generally complex, its behavior can at least locally be described by one of these cases. For each curve category, we explore how the dead reckoning protocol works under different network conditions.

At the remote host, the PHBDR protocol accurately tracks any entity exhibiting constant acceleration (zero jerk) because it uses second-order curves to locally model smooth motion. In this case, the remote tracking model matches the true position exactly. For these curves, network latency is irrelevant because updates only serve to signal that the entity's behavior has not changed.

PHBDR is also highly accurate for smooth curves whose acceleration changes slowly (small magnitude jerk) because those paths locally exhibit near-parabolic behavior. In these cases, the parabolic remote tracking model closely approximates the true motion. Consequently, the dead reckoning error at each position update is relatively small (linearly proportional to jerk), so the convergence algorithm quickly corrects the error by slightly exaggerating the acceleration applied between position updates. As perceived by remote hosts, network latency increases the effective error threshold proportionally to the cube of network latency; this behavior is identical to that experienced with high magnitude jerk and is discussed further in Section 4.2.1.

The next section discusses the three remaining entity curve types, whose behavior is more complex and less intuitive:

- **High magnitude jerk with no spikes:** For example, a bus traveling along a winding road exhibits smooth jerk.
- **Low magnitude jerk with occasional spikes:** Jerk may spike when an external force is temporarily applied to the entity. For example, a bouncing entity or an entity moving along rough ground exhibits near-zero jerk except that when it hits the ground, the jerk spikes and changes the velocity. In this case, the convergence algorithm must quickly recover from errors resulting from the unpredicted collision with the ground [97].[1]
- **High magnitude jerk with frequent spikes:** This case includes all remaining types of motion, including random motion. For example, a person in a crowd moves around smoothly at varying velocity but is occasionally pushed by someone else. Tracking and convergence algorithms are of little benefit in predicting or correcting the remote entity display.

4.2 Evaluating PHBDR On Complex Curve Types

In this section, we evaluate the PHBDR performance over the three remaining curve categories: high magnitude jerk with no spikes, low magnitude jerk with occasional spikes, and high magnitude jerk with frequent spikes. Based on measurements of the average packet rate and remote tracking error generated by the PHBDR protocol on these three types of curves, we make the following observations:

Curve Type Tradeoff: Although a higher packet rate generally produces higher positional fidelity in remote models, the proportion of smooth versus sudden changes in jerk exhibited by the entity motion determines the precise relation between these two variables. Moreover, different curve types exhibit different sensitivity to network latency.

Threshold Estimation: The curve type also determines the appropriate protocol error threshold required to achieve a reasonable balance of positional fidelity and network bandwidth: the

[1]We assume here that the entity is tracked in isolation, without advance knowledge of the impending collision. Section 4.4 discusses extending the protocol to leverage such information.

appropriate protocol error threshold lies between one and two times the average tolerable visualization error.

Balanced Degradation: A higher protocol threshold degrades positional and behavioral fidelity equally, while retaining a faithful representation of the true motion.

4.2.1 High Magnitude Jerk With No Spikes

If the jerk is continuous, then we may bound the error of the interpolating curve generated by the second-order tracking model. In particular, the tracking error at time $t = t_0 + d$ equals [40]:[2]

$$\text{Error}[x(t)] = \frac{x^{(3)}(\xi)}{3!}(d)(d - d_{01})(d - d_{01} - d_{12}) \tag{4.1}$$

where ξ is some time between time t_0 and time t_2. We replace $x^{(3)}(\xi)$ with j_{max}, signifying the maximum jerk. Also, because we are interested in the future error, let $d = d_{01} + d_{12} + d_{future}$. We get:

$$\begin{aligned} \text{Error}[x(\tau)] &\leq \frac{j_{max}}{6}(d_{01} + d_{12} + d_{future})(d_{12} + d_{future})d_{future} \\ &= \frac{j_{max}}{6}\left(d_{future}^3 + 2d_{12}d_{future}^2 + (d_{01}d_{future} + d_{01}d_{12} + d_{12}^2)d_{future}\right) \end{aligned} \tag{4.2}$$

With a target steady-state packet rate D between packets, we can make a substitution for d_{01}, d_{12}, and d_{future}:

$$\begin{aligned} \text{Error}[x(t)] &\leq \frac{j_{max}}{6}6D^3 \\ &= j_{max}D^3 \end{aligned} \tag{4.3}$$

From this equation, we can draw a number of conclusions. First, each incremental reduction in the error threshold causes a larger increase in the packet rate (reduction in D); conversely, each increase of the error threshold has incrementally smaller effects on the packet rate (increase in D). This result signals in modeling this class of curves, packet rate, rather than average error, is likely to be the limiting constraint. This behavior represents one endpoint in the Curve Type Tradeoff. Second, we observe that the remote tracking error increases only linearly with jerk. The PHBDR

[2] We retain the same variables used in defining the protocol in Chapter 3. the three most recent updates x_0, x_1, and x_2 were transmitted at times t_0, $t_1 = t_0 + d_{01}$, and $t_2 = t_1 + d_{12}$, respectively. The remote host receives the last update at time $t_2 + d_{delay}$ where d_{delay} denotes the network latency.

protocol therefore has limited sensitivity to the dynamics of the entity motion. For example, to be competitive with traffic [103] exhibited by many entities under the existing Distributed Interactive Simulation (DIS) protocol (IEEE Standard 1278) [37], PHBDR must achieve a target packet rate of one per second ($D = 1$). Under these conditions, the maximum tracking error (modulo network latency) equals the jerk, and the average tracking error equals only $\frac{j}{4}$. Third, because network latency delays the arrival of update packets, effectively increasing the error threshold seen at remote hosts, it can have a significant impact on the remote tracking error. By the point that a remote host processes the update, the network latency d_{delay} has increased the tracking error by an additional

$$j_{max}(3D^2 d_{delay} + 3Dd_{delay}^2 + d_{delay}^3) \tag{4.4}$$

which is dominated by the $3D^2 d_{delay}$ term because we assume that $d_{delay} \ll D$; We expect d_{delay} to equal 0.1 seconds, reflecting typical latencies across the Internet in the United States, and as we have seen, D should equal roughly 1 second. Substituting these values, the network latency raises the maximum error by $0.3 j_{max}$ (or 30%) and the average error by $0.11 j_{max}$ (or 46%).

To validate the conclusions derived from this analysis, we run simulations on selected curves from this curve category

4.2.1.1 Oscillatory Motion

Oscillation can be regarded as one of the worst case situations for dead reckoning among curves having smooth jerk. In this case, the jerk smoothly changes within the range $[-(2\pi f)^2 A, (2\pi f)^2 A]$, where A represents the amplitude of the oscillatory motion and f represents its frequency. Non-oscillatory motion whose jerk increases or decreases monotonically also exhibits similar remote modeling characteristics.

At tight protocol thresholds, the PHBDR algorithm yields tight positional fidelity for high-speed oscillatory motion by transmitting update packets soon after the entity's behavior changes, as shown in Figure 4.1a. Table 4.2 lists the average remote modeling error and packet rates corresponding to each curve in the figure. We observe that tight error thresholds produce high positional fidelity at the expense of a higher update rate. When the protocol error threshold is increased, remote sites model the oscillation amplitude with less positional fidelity because they receive fewer updates for each oscillation period. Despite the lower positional fidelity, we observe that the remote model still exhibits oscillatory motion, thus maintaining reasonable behavioral fidelity. This behavioral fidelity arises because at higher error thresholds, PHBDR is simply sampling the motion at a rate lower

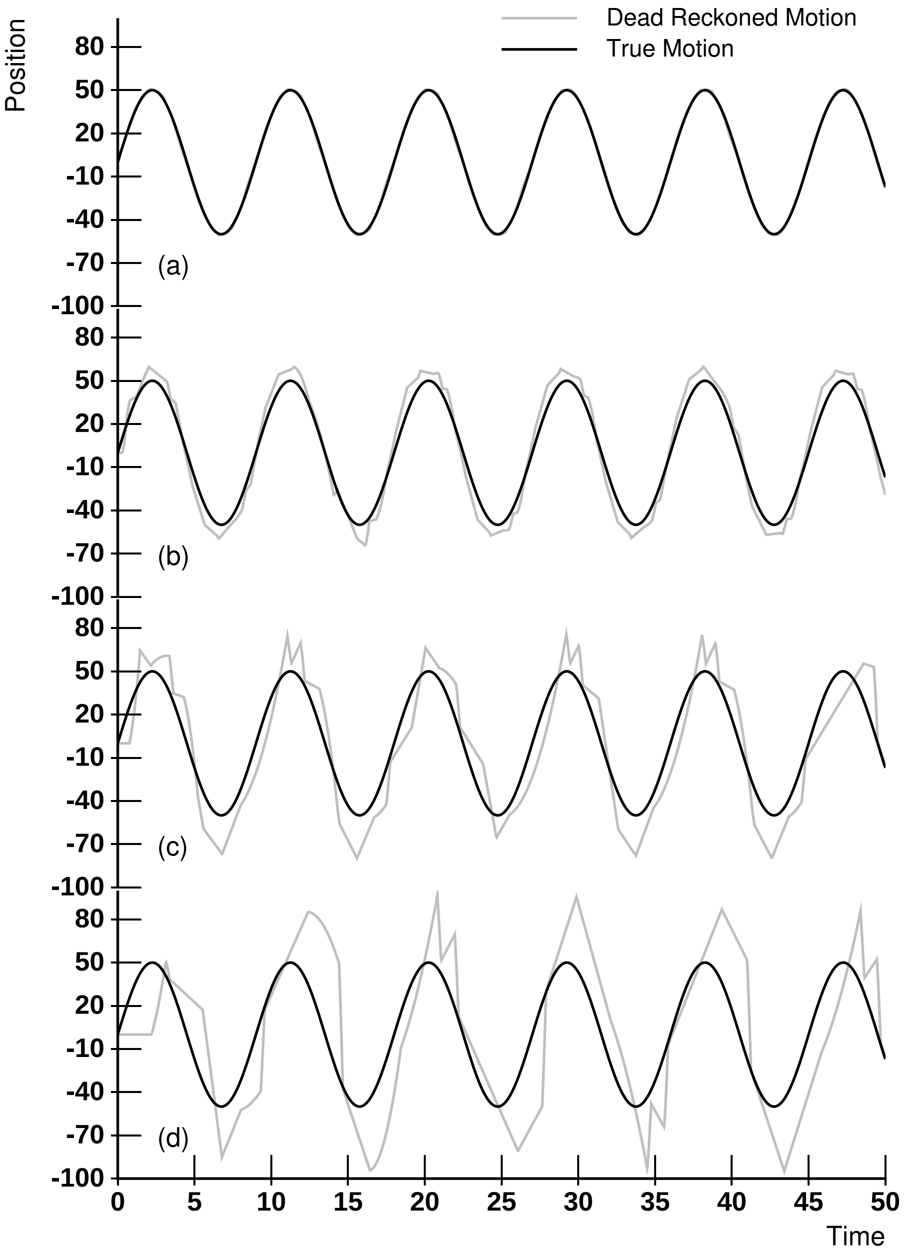

Figure 4.1: PHBDR Rendering of Sinusoidal Oscillation (amplitude 50 meters, period 9 seconds, timeout 5 seconds, zero latency, threshold (a) 1, (b) 10, (c) 25, and (d) 50 meters)

Error Threshold		Average Error Between True Model and				Packet Rate Per	
		Remote Modeling		Remote Rendering		Oscillation	Second
1	(2%)	0.38	(0%)	0.84	(2%)	19.44	2.16
10	(20%)	3.85	(8%)	7.58	(15%)	9.36	1.04
25	(50%)	11.56	(23%)	16.11	(32%)	5.40	0.60
50	(100%)	19.59	(39%)	28.63	(57%)	3.60	0.40

Table 4.2: Summary of Average Error (Absolute and As Percentage of Amplitude) and Packet Rate for Oscillatory Motion

than its natural Nyquist frequency. Although sub-sampling an oscillatory motion yields an incorrect oscillation frequency (and, therefore, incorrect position), it is still guaranteed to demonstrate some sort of oscillatory behavior.

We also observe from Table 4.2 that the rendered position (incorporating both tracking and convergence) exhibits a higher error than the remote position model (incorporating only entity tracking). This discrepancy arises because the convergence algorithm artificially sustains an error in order to retain the illusion of smooth entity motion on the display, while the tracking algorithm maintains an up-to-date view of the entity's real position based on provided update packets. Because we are most interested in the visual effect produced by the simulation, we only consider the remote rendering error throughout the rest of this chapter.

4.2.1.2 Circular Motion

Circular or spherical motion results when jerk smoothly changes in direction but retains a constant magnitude. Analysis of circular motion has broader applicability, however, for a large family of curves can be locally approximated as circles [101]. Figure 4.2a shows the average remote rendering error as a function of the protocol error threshold, and Figure 4.2b shows the corresponding packet rate as a function of the protocol error threshold. Although the data is based on circles of radius 50, the results generalize for circles of any radius: a smaller radius simply means that the entity must move with a higher jerk in order to maintain the same speed.

The graph reveals that the average displayed error increases linearly with increasing protocol threshold. On the other hand, the packet rate rises rapidly with tighter threshold. These results confirm our mathematical analysis that remote rendering on these curves is more sensitive to bandwidth usage than to remote modeling error. If all other factors are equal, the protocol error

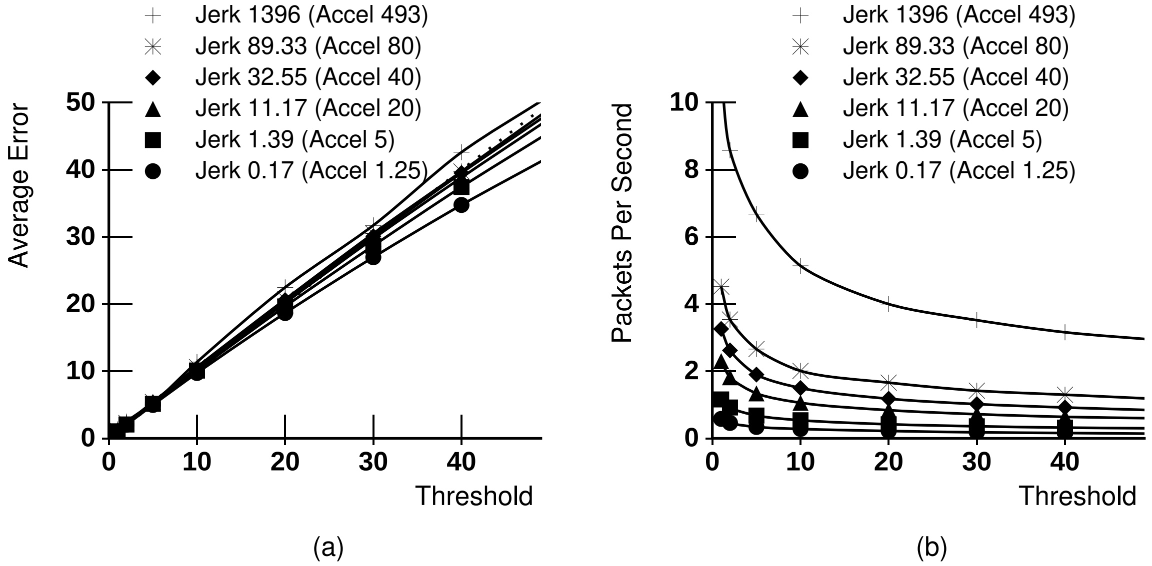

Figure 4.2: Average PHBDR Protocol (a) Rendering Error and (b) Packet Rate for Circular Motion (radius 50 meters) With Zero Network Latency

threshold value for this class of curves must be determined primarily by bandwidth limitations, rather than visual error requirements. This characteristic represents one endpoint of the Curve Type Tradeoff.

The dotted line of Figure 4.2a supports the Threshold Estimation observation, which sets the protocol error to between one and two times the average visual error tolerance (or between $\frac{j}{2}$ and j, based on the prior analysis). For this endpoint in the Curve Type Tradeoff, the protocol error threshold should be set equal to the average tolerable error. As shown by the dotted line, this threshold provides a remote model that provides the minimum acceptable positional fidelity and therefore optimizes the network bandwidth, which is the sensitive resource for this curve class.

4.2.2 Low Magnitude Jerk With Occasional Spikes

In the second class of entity motion, the acceleration remains nearly constant most of the time but occasionally changes suddenly. For example, when two entities collide, they instantaneously exhibit high acceleration as momentum is reversed. The velocity consequently undergoes an almost instantaneous change. After changing direction, each entity's acceleration returns to a stable state.

A bouncing entity offers one example of this class of behavior, as illustrated in Figure 4.3. The jerk is zero as the entity is moving, but it exhibits a positive spike as the entity changes direction and a negative spike as the entity returns to a stable acceleration. The corresponding acceleration

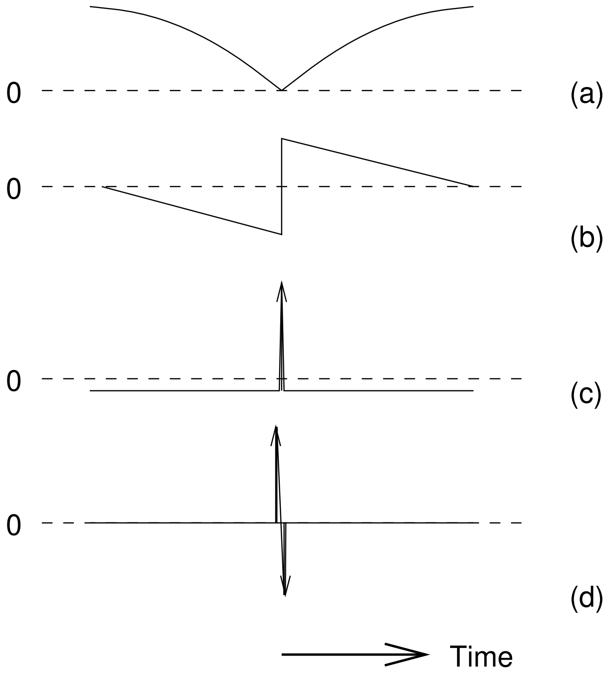

Figure 4.3: Entity (a) Position, (b) Velocity, (c) Acceleration, and (d) Jerk During a Collision

remains constant as the entity is moving, but it spikes as the entity bounces and velocity reverses direction.

Prior position information provides insufficient information for predicting a collision and the subsequent spike in jerk. We must therefore rely on the convergence algorithm to quickly recover from the error after the behavior change is reported. Because the acceleration changes instantaneously at the collision point, the remote modeling error increases quadratically after the collision until it exceeds the protocol error threshold. Therefore, the protocol error threshold does not have a significant effect on packet rate. Instead, a larger protocol error threshold only serves to delay the update packet transmission by allowing the source to tolerate a larger divergence between the true and remote models.

Moreover, suppose that the entity acceleration and velocity respectively change in magnitude by a_Δ and v_Δ during the collision, where a_Δ and v_Δ are calculated from the instantaneous acceleration and velocity before and after the collision. We ignore the effect of jerk, because by assumption, its magnitude is near-zero before and after the collision. Therefore, we can apply the basic physics equations of motion to derive the remote tracking error at time $t = t_2 + d_{future}$:

$$\text{Error}[x(t)] = \left|\frac{1}{2}a_\Delta d_{future}^2 + v_\Delta d_{future}\right| \tag{4.5}$$

In the case of an "elastic bounce," where the entity's acceleration does not change, this expression reduces to:

$$\text{Error}[x(t)] = |v_\Delta d_{future}| \tag{4.6}$$

This linear relationship between error and packet rate is fundamentally different from that exhibited with paths exhibiting high jerk. To achieve better remote modeling of bouncing motion, we should transmit packets more aggressively but not necessarily more often. We conclude, therefore, that the error threshold value for this class of curves should be determined primarily by the visual error tolerances, rather than network bandwidth limitations. This behavior represents the second extreme in the Curve Type Tradeoff.

As we have seen with other types of curves, network latency increases the protocol error threshold perceived at remote hosts. However, Equation 4.6 indicates that latency has a less dramatic effect on the average error for bouncing motion than it does for oscillatory or circular motion. The network delay d_{delay} introduces an average additional error of $|v_\Delta d_{delay}|$, or $|0.1v_\Delta|$ based on our Internet latency estimates. Assuming a maximum one second delay on the packet transmission after the bounce motion (corresponding to the target one packet per second update rate), we see that the network latency increases the maximum remote tracking error by only 10% and the average remote tracking error by only 21%, considerably less than the respective 30% (maximum error) and 46% (average error) effects seen with oscillatory and circular motion.

We use simulations on bouncing motion to confirm these analyses.

Figure 4.4a shows the PHBDR rendering error performance for bouncing motion as a function of the protocol error threshold. The figure confirms that the average rendering error rises nearly linearly with increased protocol threshold. Figure 4.4b shows the relationship between packet rate and protocol error threshold. The figure confirms that the number of transmitted packets is almost independent of the protocol threshold. The large drop in packet rate after the threshold equals 20 meters occurs as transmission of update packets from one bounce begin to overlap with the occurrence of the next bounce. Figure 4.4 therefore confirms our mathematical analysis indicating that changes to the protocol error threshold affects average error more significantly than network bandwidth utilization. If all other factors are equal, the protocol error threshold should be determined primarily by the visual error tolerances.

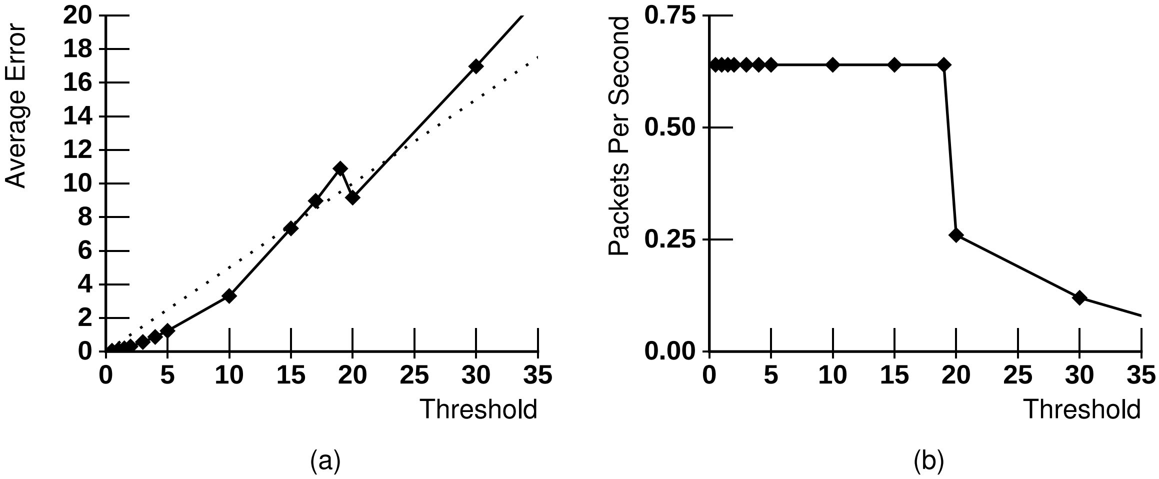

Figure 4.4: Average PHBDR Protocol (a) Error and (b) Packet Rate for Bounce Motion of Height 25

The dotted line of Figure 4.4a shows that this curve class represents the other extreme for the Threshold Estimation observation. For bounce motion, the protocol error threshold should be set to twice the average visual error tolerance to achieve the desired positional fidelity. However, the average modeling error does depend on how frequently the entity jerk exhibits sharp spikes. More frequent acceleration changes demand lower thresholds to achieve the same average positional fidelity.

Figure 4.5 shows the protocol's visual behavior on bouncing motion. Figure 4.5a shows that for tight thresholds, the remote modeling supports the desired positional fidelity. With an increased protocol error threshold, the remote model loses positional fidelity but still retains considerable behavioral fidelity, as shown by Figure 4.5b. In particular, the remote model still exhibits the same bounce-like motion and frequency as the true path. In this case, however, the remote model has switched to a first-order approximation of the entity's motion because the algorithm detects a sudden change in direction (small angle of embrace) between successive updates.

The degradation of positional and behavioral fidelity becomes more pronounced when the threshold is increased further, as in Figure 4.5c. Although positional fidelity is degraded, the remote model retains significant similarities to the true motion. The remote entity model exhibits the bounce-like behavior present in the true motion, though it has been degraded by presenting a lower-frequency bounce with varying height. Moreover, although the positional fidelity is degraded, the remote entity position generally remains within its true positional range. This remote model is therefore still usable by users who are far from the entity in the virtual world.

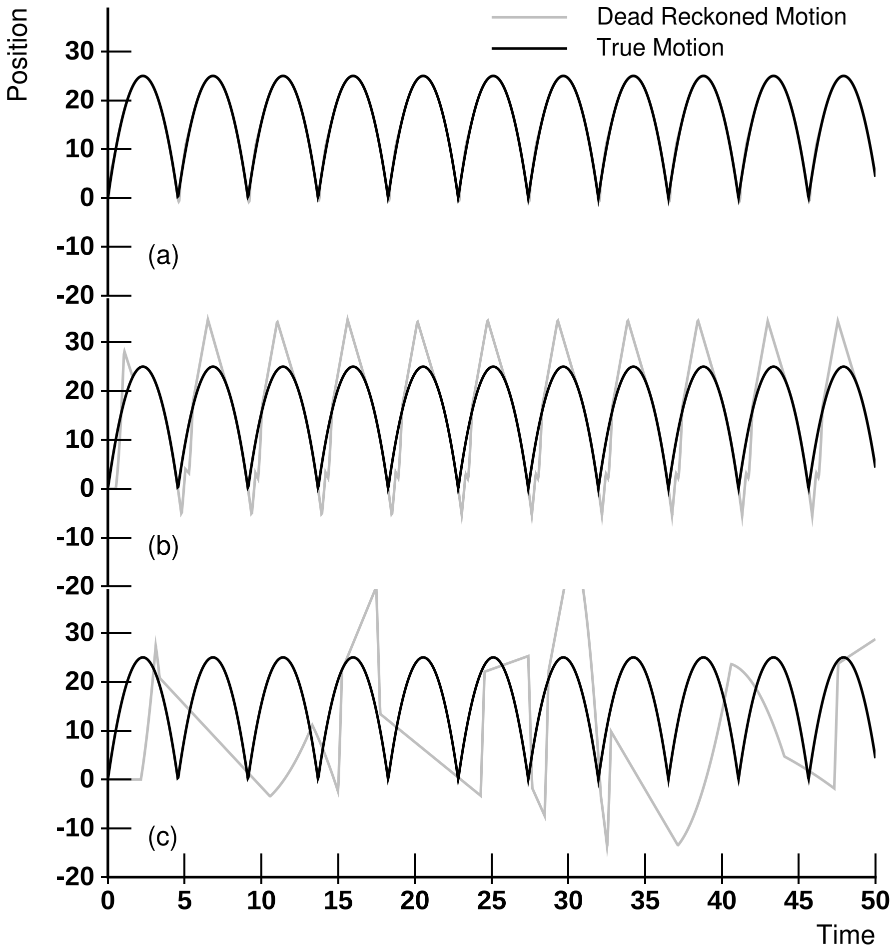

Figure 4.5: PHBDR Protocol Rendering of Bounce Motion (height 25 meters; timeout 5 seconds; threshold (a) 1, (b) 10, and (c) 25 meters)

4.2.3 High Magnitude Jerk With Frequent Spikes

If jerk has high magnitude and exhibits frequent changes, the resulting entity motion is characterized by both smooth, rapid acceleration changes and sharp, unpredictable changes. Such behavior includes most random motion, such as a person weaving through a crowd or a particle traveling through a wind tunnel. Accurate remote modeling of these curves relies on a high update rate from the source host. The tracking step cannot accurately predict the future position because the behavior is likely to change at any moment, and the convergence step us unlikely to recover from a display error before the entity's behavior changes again.

Despite the complexity of this class of curves, the PHBDR protocol provides reasonable support for such erratic behavior. Figure 4.6 demonstrates the algorithm's behavior on a sample path from this curve class. The curve trace was generated by randomly perturbing the entity's acceleration on each frame by a small amount (up to one percent) and introducing large acceleration jumps (by reversing the direction of acceleration) after 0.5% of the frames (on average, once every 3.33 seconds). In this example, we observe that the error and packet rate behavior represent a hybrid of that seen for circular and bounce motion in Figures 4.2 and 4.4. In fact, all curves that we have studied from this class generate an intermediate behavior between the other two curve classes, depending on the proportion of smooth and sudden changes in jerk. If the curve is dominated by smooth jerk changes, then it exhibits oscillation-like behavior, while a dominance of sudden jerk changes results in more bounce-like behavior. Based on our analyses of the oscillation and bouncing motion, we conclude that the Threshold Estimation observation similarly allows us to trade off positional fidelity and network utilization across the spectrum represented by this class of curves.

In summary, no remote modeling algorithm can accurately support random motion without introducing a packet rate approaching the frame rate. However, by sampling only position and smoothing between these periodic samples, the PHBDR protocol provides good behavioral fidelity with an acceptable positional fidelity.

4.3 Advantages of the PHBDR Protocol

Having systematically studied the behavior of PHBDR, we now extract the key advantages offered by the protocol. To place these advantages in context, we consider them in relation to the current Distributed Interactive Simulation (DIS) protocol (IEEE Standard 1278) [37] that sees wide use in deployed distributed simulation systems. The DIS dead reckoning protocol [36] transmits position, velocity, and acceleration information whenever the remote entity model exceeds a threshold or a five second timeout elapses. Using data from the most recent packet, DIS dead reckoning algorithms generate a second-order model to predict the future entity location.

In comparing the PHBDR and DIS protocols, we consider three criteria: remote model stability, dependencies between hosts, and network and computational load.

4.3.1 Remote Model Stability

The *stability* of a remote model refers to how it is affected by short-term changes to the entity's behavior. For example, we re-consider the rapid bouncing motion illustrated in Figure 4.5. Figure 4.7

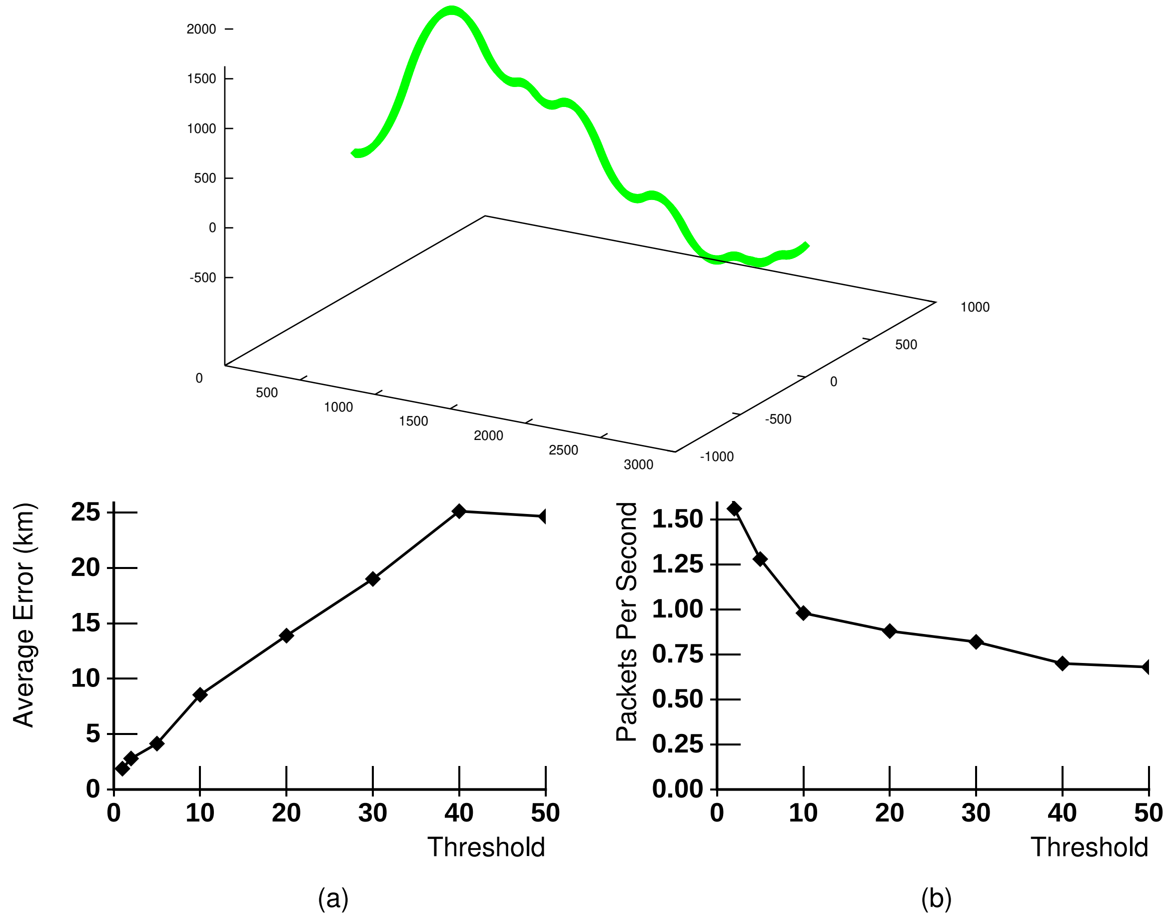

Figure 4.6: Average PHBDR Protocol (a) Error and (b) Packet Rate for Sample Motion Exhibiting High Jerk with Frequent Spikes

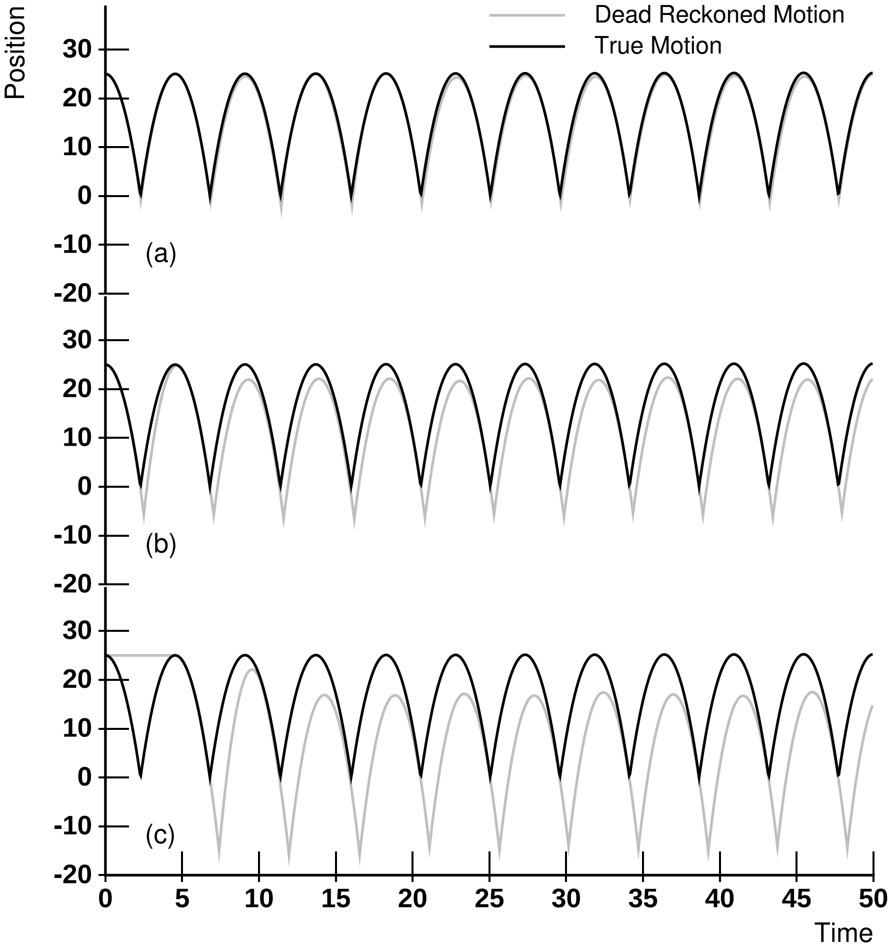

Figure 4.7: DIS Rendering of Bounce Motion (height 25 meters; timeout 5 seconds; threshold (a) 1, (b) 10, and (c) 25 meters)

illustrates how the DIS protocol performs on the same entity motion. Figures 4.5a and 4.7a reveal that at tight protocol error thresholds, both protocols perform comparably. At wider thresholds, as illustrated in Figures 4.5b–c and 4.7b–c, the algorithms behave quite differently. We observe that at a high error threshold, the DIS remote model faithfully preserves the bounce frequency. However, it regularly positions the entity outside the range of its true motion, so the user sees the entity in locations that it never exists. Although hosts would typically employ these high error thresholds when the entity is far from the local viewer in the virtual world, this unbalanced degradation of behavioral and positional fidelity is potentially confusing to users.

On the other hand, the PHBDR protocol, by using several position updates instead of extra derivatives, generally degrades positional and behavioral fidelity more evenly. Behaviorally, the protocol smoothes the entity motion and hence presents a lower-frequency bounce. More importantly, the remote entity model generally stays within the positional range of the actual entity motion, so although the displayed position is inaccurate, it almost always places the entity in a realistic position. Users effectively see less fidelity in both position and behavior, but the presented model is still constrained by the true motion. This balanced fidelity reduction is more meaningful for users.

The smoothing characteristics of the PHBDR protocol arises because it relies on information—namely entity position—that changes least rapidly and least randomly. For example, a physical entity's position must be continuous, and the position is generally an *indirect* and, therefore, delayed response to velocity and acceleration changes. By relying on position information alone, PHBDR is consequently less sensitive to short-term changes in an entity's velocity and acceleration. Furthermore, the adaptive tracking and convergence techniques allow the PHBDR protocol to simply ignore acceleration altogether when its value seems to be changing rapidly. As a result, the protocol accurately models a variety of entity behaviors, including straight lines, sharp turns, and smooth curves.

The stability of the PHBDR protocol contrasts with the DIS protocol which relies on more transient entity attributes. Velocity and acceleration can change rapidly and even instantaneously, and errors in their representation have second- and third-order effects on position. If a source host happens to send an update during a velocity or acceleration spike, receivers extrapolate the entity using inaccurate information that exaggerates the entity's motion. These factors potentially affect the remote modeling of a broad set of entities. Most entities, though they appear to be moving smoothly, are constantly subjected to external forces which rapidly change the velocity and acceleration. For instance, as a car drives along a road, the vertical motion changes rapidly because road surfaces are not perfectly smooth.

We observe that our techniques for achieving remote model stability in the PHBDR protocol have broader applicability. For example, DIS-style protocols could be re-engineered to detect and ignore instantaneous velocity and acceleration values that appear to represent short-term fluctuations. The details of implementing this hybrid approach represent an area for future research. However, even with this optimization, DIS protocols still face a significant disadvantage, namely that not all entities can provide accurate velocity and acceleration information. For example, entity models derived from sensors attached to physical entities can usually generate position samples but cannot always generate velocity and acceleration. PHBDR, which relies only on position information, therefore still supports a more general class of entities.

4.3.2 Decoupling Receivers from Network and Source Host Performance

By using timestamps to synchronize the remote tracking at receiving hosts, the PHBDR protocol is effective at addressing network latency and jitter issues in real-time visualization systems. Although each host receives the packet with a different latency, it assures "eventual consistency" semantics whereby all remote hosts eventually share the same tracking model for the entity. Furthermore, the protocol supports out-of-order packet arrival, because the receiver simply inserts the update in the correct order in the dead reckoning state for the entity and discards the update if it predates all of the updates currently held in the entity's position history. This approach departs from the published DIS dead reckoning protocol, under which remote hosts process update packets when they arrive, without regard for intervening network latency. Under the DIS protocol, therefore, each remote host may have a significantly different model of the entity's current position.

However, the eventual consistency provided by timestamps represents a general-purpose technique that has greater applicability than the PHBDR protocol. Figure 4.8 shows the DIS protocol performance when it is modified to use timestamps in a manner similar to that of PHBDR. The figure shows two traces of F-16 aircraft performing "Air Combat Maneuvering" (ACM) turns, representing typical motion in one-on-one aircraft combat.[3] For a given packet rate, an increase in network latency produces a near-linear reduction in the average positional fidelity of the PHBDR remote model. The graph reveals that the modified DIS protocol exhibits nearly identical behavior when faced under network latency. Without timestamp information, DIS dead reckoning error is higher by an order of magnitude. From these graphs, we see that using timestamps to provide eventual consistency semantics in DIS is warranted.

Moreover, timestamps eliminate almost all real-time dependencies between the source and destination host frame rates. Each remote host generates positional information for each frame by sampling its entity model at the local frame rate. Consequently, the remote host can generate a smooth visual representation even though its frame rate may be faster than the source frame rate. Timestamps therefore provide an element of decoupling between simulation hosts. As we shall see in later chapters, decoupling represents a significant architectural direction for developing scalable simulation systems.

Using timestamps does not eliminate all real-time dependencies, however. To be effective,

[3]These traces are provided courtesy of Dan Schab at the Naval Air Warfare Center–Training Systems Division (NTSC).

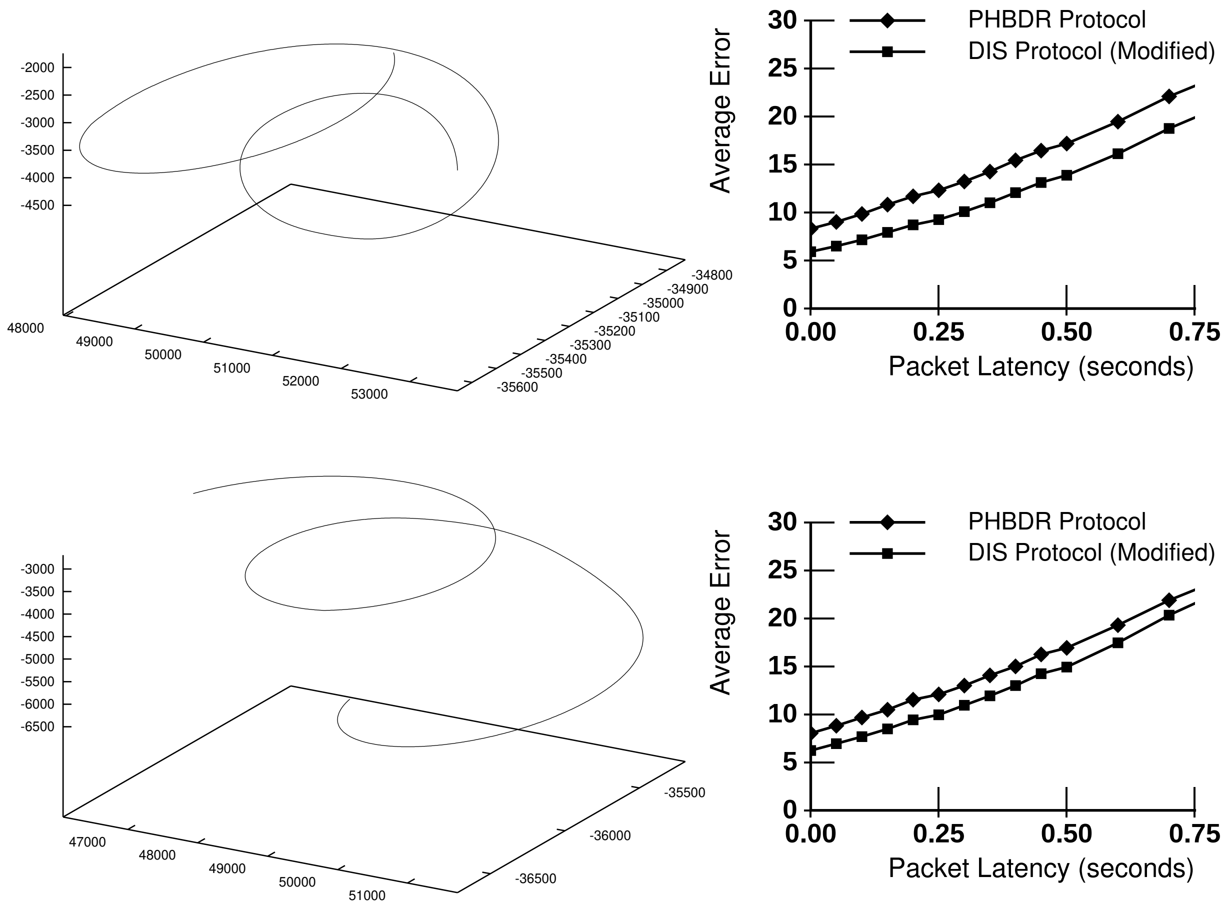

Figure 4.8: Remote Rendering Error on Two Traces of F-16 Turning Maneuvers With Network Latency (protocol error threshold 20 feet)

PHBDR requires synchronized clocks at all simulation hosts, and it assumes that the source host is executing the position error threshold check at a reasonably high frequency to assure that update packets are quickly transmitted when remote tracking error arises.

4.3.3 Bandwidth and Computational Load

By transmitting only position information, the PHBDR protocol transmits smaller packets than competing algorithms such as DIS. The protocol transmits three 64-bit position parameters, a 32-bit timestamp, a 32-bit entity identifier, and a 224-bit UDP/IP header for a total of 480 bits.[4]

[4] In this discussion, we ignore the size of any other entity state information contained in update packets.

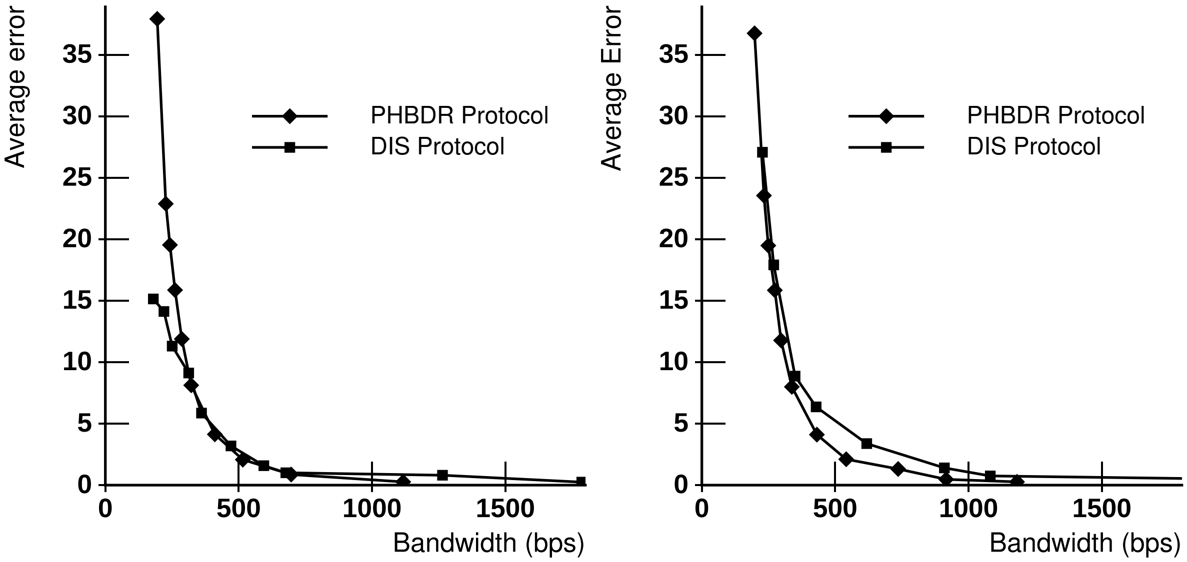

Figure 4.9: Bandwidth–Error Comparison Between PHBDR and DIS Protocols on F-16 Traces Shown in Figure 4.8

For comparison, the DIS protocol transmits three 64-bit position parameters, three 32-bit velocity parameters, three 32-bit acceleration parameters, a 32-bit timestamp, a 32-bit entity identifier,[5] and a 224-bit UDP/IP header for a total of 672 bits. We see, therefore, that the PHBDR protocol can transmit 1.4 times as often as DIS and maintain the same bandwidth. If a single update packet aggregates information from multiple entities or vertices, the packet size difference approaches a factor of 2.

DIS performs best when generating remote models for entities moving along smooth curves with monotonically changing acceleration because the most recent instantaneous acceleration information received by DIS is more accurate than the smoothed acceleration computed by PHBDR. Figure 4.8 demonstrate the superior modeling behavior of DIS on two examples of such curves. However, if we account for the smaller packets generated by the PHBDR protocol, the comparison is quite different. Figure 4.9 plots the bandwidth requirements of each protocol against the resulting positional fidelity (assuming no network latency). We see that for these curves which represent the best case for DIS, the PHBDR protocol performs comparably in the worst case and outperforms DIS in many

[5]The DIS protocol actually uses a 48-bit entity identifier that includes a 16-bit site identifier, 16-bit exercise identifier, and 16-bit entity identifier. Because the site information is already encoded by the source host address contained in the IP header, we presume that a 32-bit exercise/entity identifier will be sufficient for future versions of the DIS protocol.

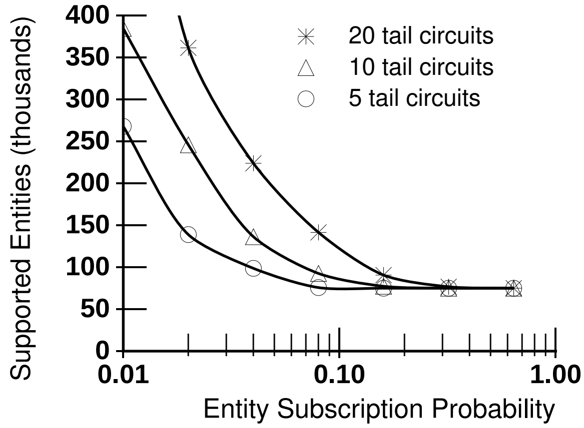

Figure 4.10: Expected Number of Supported Entities Using PHBDR in a Multicast Environment with 200 Hosts

situtations.

The bandwidth reduction provided by PHBDR has a significant effect on the scalability of a distributed simulation. In a broadcast-based simulation, for example, PHBDR protocol updates would saturate the 45 Mbps tail circuits (which, given the emergence of gigabit-per-second LAN capacity and the rapid investment in high-bandwidth WANs, we expect will continue to be the primary bandwidth bottleneck in future networks) at 80% utilization with 75,000 entities, each transmitting one update per second. An equivalent DIS simulation, including a 10% lower update rate per entity to assure equivalent remote modeling error, would saturate the tail circuits with fewer than 59,000 entities. In a large simulation, the use of multicast for information dissemination potentially prunes some of the entity updates from various tail circuits, thereby increasing the number of potential entities that the system can support. For example, assuming a network configuration with 200 hosts, Figure 4.10 shows the expected number of entities that the simulation can support as a function of the probability that a given host is interested in a given entity. These numbers are quite conservative, because they fail to consider any locality exhibited by the entity subscription patterns among hosts at the same site. As one would expect, having fewer hosts behind each tail circuit allows multicast to have a greater effect on reducing bandwidth. Additionally, reducing the probability that a host subscribes to a given entity's data allows the simulation to support more entities overall. We see that to support the STOW 97 program target of 100,000 entities [1], each host should only see between 5% and 15% of the available entities, or 5,000–15,000 entities. Given our target of one update per second per entity, these entity counts translate into 5,000–15,000 packets

Parameter	PHBDR Values	DIS Values
Current displayed position, velocity, and acceleration	9	9
Most recent position, velocity, and acceleration update and its timestamp		10
Two previous position updates and their timestamps	8	
Convergence time	1	1
Velocity and acceleration at convergence point	6[a]	
Small–medium and medium–large angle thresholds	2	
Δ_{max-cp}	1	
Total	27	20

[a] Alternatively, this information can be recomputed from the position history information at the end of the convergence period, thereby saving these six floats of storage per entity.

Table 4.3: State Values Stored Per Entity Under the PHBDR and DIS Protocols

per second, which is achievable on modern high-end workstations. Chapter 7 describes the effects of additional bandwidth reduction techniques to improve the scalablity of distributed simulations.

The scalability of the PHBDR protocol is also enhanced because it introduces minimal computational load on sending or receiving hosts when compared with protocols that only use the most recent packet for modeling remote entities, and the PHBDR can actually reduce computational load. For smooth curvature motion (which requires the most computation), the protocol requires 51 multiplication and 30 addition operations to update the entity's tracking and convergence models in response to an incoming update. By using local curve fitting to characterize the entity's path, the algorithm automatically shifts computation away from entities exhibiting "uninteresting" behavior (such as linear motion) or about which it has the least information (such as those that have just exhibited a sharp turn or change in behavior). In the case of almost linear motion, the PHBDR requires only 27 multiplication and 15 addition operations to process an update; after the entity undergoes a sudden change in direction, PHBDR requires 30 multiplication and 21 addition operations to process the update.

The PHBDR protocol also has the desirable property of storing per-entity state information that is comparable in size to that required by the existing DIS protocol. Maintaining too much state can adversely affect computation speed because entity state must typically reside in the host's cache to achieve maximal real-time performance. The PHBDR protocol stores 27 values for each entity, as listed in Table 4.3; at the expense of some additional computation, this information can be reduced to 21 values. As a comparison, the DIS protocol stores 20 values for each entity.

4.4 Addressing Limitations in the PHBDR Protocol

The PHBDR protocol uses position information from multiple update packets to provide stability in the remote model, but providing this stability also introduces some disadvantages, some of which we observed while deploying the algorithm in the PARADISE distributed simulation system developed in the Distributed Systems Group at Stanford University. We propose potential protocol extensions to address these limitations, though studying their effectiveness is an area for future research.

4.4.1 Delayed Reaction to Sudden Behavior Changes

The PHBDR protocol maintains a history of three updates; after a sudden change to the entity's behavior—such as after a collision—*three* new updates are required before all information about the previous path is discarded from remote hosts. This problem is most visible, for example, when an entity stops moving. Until three update packets indicate that the entity has kept the same position, the remote model oscillates around the entity's true location. Similar problems have been faced by previous prediction algorithms [74]. On the other hand, using the DIS protocol, remote hosts use only information from the most recent update packet, so when an entity stops moving, the remote host reacts immediately because the update packet shows a zero velocity and acceleration.

To address this issue, a host can transmit *state-replace* packets based on domain knowledge about a local entity's behavior. The state-replace packet includes three new position updates with corresponding timestamps. In response to a state-replace packet, remote hosts replace any position information previously stored about the entity with the new position updates from the packet. For example, after an entity stops moving, the entity would transmit a state-replace packet containing three updates showing the entity in its fixed position. Similarly, when an entity changes direction after a collision, a state-replace packet would include three position samples taken soon afterward. In order to transmit state-replace packets, however, the source must be able to recognize sudden behavior changes and determine that the new behavior will persist. Detecting such behavioral change requires entity-specific information that is not always available from an entity model at the source host. We therefore do not include state-replace packets as part of the basic PHBDR protocol because doing so would contradict our goal of constructing a general-purpose dead reckoning algorithm that can be used on all source entity models. Instead, extensions to support particular entity behaviors can be built on top of the general-purpose PHBDR base when better remote modeling is required.

4.4.2 Reaction to Packet Loss

Because remote hosts store information from multiple update packets, the PHBDR protocol can also be more sensitive than competing algorithms to packet loss. If a remote host does not receive an update packet, then its model will not be completely recovered until up to three new updates arrive. In the DIS protocol, however, each packet encapsulates all of the desired entity dead reckoning state, so a host can completely recover from a packet loss when the next update packet arrives.

To a large extent, the protocol's transmission timeout and use of position history limit the effect of packet loss on the PHBDR protocol. The timeout guarantees that a single lost packet can only disrupt the remote entity model for a bounded period of time. Furthermore, the path stability provided by the position history, as discussed in Section 4.3.1, additionally mitigates the effects of packet loss: After missing a packet, the host continues to generate its entity model based on a long-term history of information. DIS-style protocols that only store one packet must continue to provide an entity model based only on instantaneous information.

Simulations that use PHBDR can take additional steps to further mitigate the effect of packet loss. The position history protocol can be modified so that besides including the current position, each update packet includes the previous one or two transmitted position samples. In effect, each position sample is transmitted multiple times over the network, thus increasing the chance that it eventually arrives at each receiver. Under this approach, all update packets effectively behave like state-replace packets. However, this optimization does impose a bandwidth cost and more computation at the receivers. Alternatively, the simulation designer can employ a scalable reliable multicast protocol, such as Log-Based Reliable Multicast [33], that allows remote hosts to locally detect and recover missing position updates. This approach is best suited for entities that produce more occasional updates.

When the remote modeling error is large, particularly when there is a high loss rate for position updates, the convergence algorithm can generate entity motion that might appear “unnatural” to viewers [48]. For example, when rendering an F-16 aircraft, the convergence algorithm might cause the aircraft to move faster than physically possible, or it might cause the aircraft to take an unexpected path.

These convergence problems can be solved in two ways. First, when the remote host finds that its model of the entity and the displayed position differ too much, then it can choose to simply avoid convergence altogether and “jump” the entity to the correct location. In many situations, the viewer would prefer to see this jump—which would be ascribed to network problems—rather than see a smooth, but inappropriate maneuver. Determining whether jumping or convergence is an appropriate display technique is a decision that must be made based on the particular entity type, the type of viewer, and the type of simulation application. To further assure that the convergence does not produce an unrealistic display, the remote host may apply some entity-specific information to constrain the path generated by the basic convergence algorithm. For example, the remote host can ensure that the generated F-16 path never exceeds the aircraft's maximum physical speed. One might also employ constraints based on terrain topography, similar to those used by Talluri and Aggarwal for remote robot navigation [90], or based on the location of terrain features, similar to those used by Tsutsui for car navigation [94]. In a production simulation, entity-specific information may be available, though requiring such information is inconsistent with the goals of the basic PHBDR protocol.

4.4.3 Quiescent Entity Traffic

To provide resilience against packet loss, the transmission timeout requires all entities to generate at least one update packet within any given timeout period. Consequently, most of the traffic generated by inactive entities is timeout updates. In extreme cases, a stationary entity must transmit updates after every transmission timeout even though no state has changed. Even though no single entity generates timeout-based updates at a significant rate, the aggregate number of these updates might represent a significant overhead for large simulations in which most entities are idle at any given time. Indeed, this problem arises with all protocols having transmission timeouts, including DIS.

To reduce the amount of extra traffic generated by stationary entities, we can extend the PHBDR protocol to support variable transmission timeouts. Variable timeouts introduce some overhead at remote hosts to maintain dynamic timeout information for each known entity, but this minor computational overhead is dwarfed by the potential reduction in packet rates. We consider two variable timeout schemes: a tiered-timeout approach and a exponential-backoff approach.

The tiered-timeout approach defines a small number of specific timeout levels that are selected *explicitly* by the entity based on its behavior. For example, when the entity becomes idle, it transmits an *timeout-length-announce* message, announcing that it will use a longer transmission timeout with its (less frequent) update packets. When the entity's behavior becomes more dynamic, a corresponding timeout-length-announce message restores the transmission timeout to the original (short) duration. To further extend this technique, an entity may transmit a timeout-length-announce packet with a flag representing an infinite timeout value, effectively signalling that it will no longer need to transmit any updates. To begin transmitting updates again, the entity must reliably multicast a new timeout-length-announce packet, possibly along with a state-replace packet to provide an initial dead reckoning state for remote hosts.

Using the exponential-backoff approach, the entity's timeout value is determined *implicitly* based on its recent update behavior. Under this approach, the entity applies exponential backoff to increase the transmission timeout if it has not transmitted any update packets within the previous timeout period. The timeout is bounded by some maximum value, and any position update transmission immediately restores the timeout to the original (short duration) value. This approach can be further optimized with an explicit packet to set the timeout to infinity when the maximum timeout value has been in place for some period of time.

4.4.4 Dependence on Time Synchronization

One disadvantage of the PHBDR protocol is a reliance on time synchronization among the simulation hosts. Though time synchronization is typically not a serious concern on Unix hosts connected to the Internet, it does pose a challenge for distributed simulation applications on non-Unix operating systems. In these environments, network time synchronization protocols are not ubiquitous.

Where clock synchronization is not possible, hosts can implement a simplified version of the NTP [58] protocol by adding additional information to position update packets. In each outgoing packet, host h includes a tuple (h_i, t_i, d_i) for each host h_i from which an update packet has arrived since host h last transmitted an update. In this tuple, t_i is the timestamp contained in the update received from h_i, and d_i is the elapsed time since that packet arrived. If host h_i receives the update from h at time t'_i, it can estimate the one-way delay between h and h_i as $\frac{t'_i - t_i - d_i}{2}$. Using this delay information, the host can estimate the current time at each of the other hosts and set the local clock to an average value. Though this technique does not allow hosts to account for network jitter effects, it does allow them to account for network latency. This latency estimation technique is a multicast adaptation of the technique proposed by Floyd, et al [24].

4.4.5 Uniformly High Network Latency

Because the source host does not model the effects of network latency on its transmitted update packets, it does not have an accurate real-time representation of the entity model stored at remote hosts. Although the protocol's timestamp does provide eventual consistency semantics, the network does introduce a minimum transmission latency, thereby assuring that the shadow model at the source is temporarily more accurate than the model actually maintained by remote hosts. This problem becomes most significant when all receivers perceive a high latency, such as when the source's outbound tail circuit is slow.

To address this issue, the source can maintain an estimate of the minimum latency between it and the receiving hosts and uses this estimate to dynamically reduce the protocol error threshold that it uses. This approach effectively allows the source to generate update packets before the receiving hosts actually encounter the error threshold. One potential method for maintaining this latency estimate would involve having the source include the current estimate inside transmitted update packets. If a receiver perceives a significantly lower latency, then it sends the current perceived latency to the source, and the source applies an exponential smoothing algorithm

$$d_{delay:new} = \alpha d_{delay:old} + (1-\alpha) d_{delay:sample} \tag{4.7}$$

on any latency updates that it receives. Moreover, to ensure that the latency estimate adapts to increases in network latency, the source automatically increases the current latency estimate by a fixed factor β on each update transmission. Intuitively, the source continually tests higher latency estimates, and receivers provide feedback when the estimate grows too large. To achieve stability, the relationship between α and β is important. To maintain the latency estimate within a factor of ϵ of the true minimum latency,

$$\beta = \frac{\alpha + \epsilon}{\alpha(1+\epsilon)} \tag{4.8}$$

4.5 Conclusion

In this chapter, we have presented a systematic approach for evaluating dead reckoning protocols. To perform this evaluation, we classify curves that represent different types of entity behavior of interest in the simulation. We then apply a combination of mathematical analysis, controlled simulation, and deployment experience to measure the protocol's performance in the worst case and common case. By applying this evaluation approach across different protocols, simulation developers can gain valuable insight into a protocol's overall behavior over a broad variety of entity paths, and they have a better framework for comparing different protocols.

We have demonstrated this evaluation approach by applying it to the PHBDR protocol. In performing this analysis, we have observed that PHBDR provides accurate entity modeling over a broad range of entity behaviors. Our analysis has demonstrated that PHBDR offers the following additional benefits:

- Stable curve modeling despite transient changes in entity velocity and acceleration.

- Smaller packet size and lower bandwidth requirements than protocols that rely on velocity and acceleration information, as well as memory and computation requirements that are comparable to these existing protocols.
- An effective mechanism using timestamps to provide "eventual consistency" semantics on remote modeling state across all remote hosts.

We have also shown that the proportion of smooth and sudden jerk changes in the entity motion determines the relationship between bandwidth requirements and remote modeling fidelity, as well as the effects of network latency on fidelity. Information about the entity behavior, desired packet rate, and desired positional fidelity can therefore be used to determine an appropriate protocol error threshold.

Moreover, our evaluation has demonstrated that many of the techniques used by PHBDR are generally applicable and could be incorporated in existing protocols such as DIS. For example, DIS can easily be enhanced to make effective use of timestamps to account for network latency. Furthermore, DIS could be engineered to ignore suspicious instantaneous velocity and acceleration values, thereby achieving many of the desirable path stability properties exhibited by the PHBDR protocol.

Our evaluation has revealed some limitations of the PHBDR protocol arising mostly from its caching of multiple updates. In particular, the protocol exhibits longer-term sensitivity to packet loss and is slow to react to changing entity behavior. However, we have shown that such limitations can be removed by requiring the source to transmit redundant information and to explicitly signal significant changes in entity behavior, when such changes can be determined accurately. Implementation of these and other extensions is an area for future research.

As discussed in the previous chapter, a significant strength of PHBDR is its simplicity. By modeling the location of a vector using only a short history of that vector, the PHBDR algorithm provides a building block to address more complex remote modeling problems. In the next three chapters, we show how the PHBDR protocol can be used recursively to remotely model the orientation of a rigid-body entity, the structure of semi-rigid and non-rigid entities, and the behavior of entity groups. In presenting these recursive protocols, we demonstrate how their analysis is simplified by applying results from the analysis in this chapter.

Chapter 5

Using the PHBDR Recursively to Model Entity Orientation

The PHBDR protocol provides remote modeling of the position of a single vertex and is therefore sufficient for modeling the position of a rigid entity having fixed orientation. In this chapter, we describe how PHBDR can be applied recursively to support remote modeling of entity orientation.

5.1 The Axis Point Protocol

The Axis Point protocol, illustrated in Figure 5.1, uses position history to dead reckon the orientation of the entity's local coordinate system. As the entity rotates about any axis passing through its origin, the points $X(1,0,0)$, $Y(0,1,0)$, and $Z(0,0,1)$ in the local coordinate system move along the surface of the unit sphere centered at $(0,0,0)$. To model the entity's orientation, the Axis Point protocol simply applies PHBDR to local coordinates $X(1,0,0)$ and $Y(0,1,0)$ as they move over the unit sphere. Together, these two points completely determine the entity's rotated coordinate system, because their cross product defines the position of $Z(0,0,1)$.

5.1.1 Source Packet Generation

On each frame, the entity's source host determines the entity's origin $O(0,0,0)$, X-axis $X(1,0,0)$, and Y-axis $Y(0,1,0)$ in world coordinates. The vectors $\vec{OX}$ and $\vec{OY}$ then represent the axis point positions, which are passed to instances of the PHBDR protocol. (Alternatively, if the source is maintaining a rotation matrix for rendering the entity model, then the vectors $\vec{OX}$ and $\vec{OY}$ can

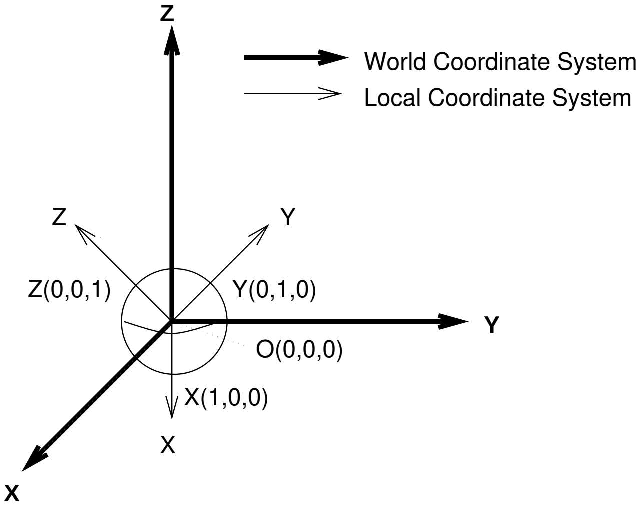

Figure 5.1: The Axis Point Protocol Tracks Rotation of $X(1, 0, 0)$, $Y(0, 1, 0)$, and $Z(0, 0, 1)$ Along the Unit Sphere Surface

be extracted directly without any additional computation.) When either of the PHBDR protocol instances desires to transmit an update packet (either because the remote modeling error of the axis point has exceeded a protocol error threshold or because a transmission timeout has arrived), the source host transmits an orientation update packet.

Orientation update packets contain a timestamp and entity identifier along with a six-tuple containing the $\vec{OX}$ and $\vec{OY}$ vector coordinates. As discussed in the next section, remote hosts dead reckon the position of these two vectors to model the entity's orientation. To improve network efficiency, the orientation and position updates for a single entity may be merged, so that all entity updates include nine coordinates (three position coordinates and six axis point coordinates).

By transmitting the axis point vectors directly, the protocol imposes minimal computational overhead on the source and destination hosts. However, the entity's orientation could be described using other formats to optimize bandwidth or provide compatibility with existing code.

- Euler/Tait-Bryan angles [76]: This representation involves transmitting a three-tuple of rotation angles (θ, ϕ, ψ). To determine the local coordinate system orientation from these angles, one first rotates it about the Z-axis by angle θ, then rotates it about the new Y-axis by angle ϕ, then finally rotates it about the new X-axis by angle ψ. The vectors $\vec{OX}$ and $\vec{OY}$ can be computed from this information by multiplying the three rotation matrices described by the

Euler angles and extracting the first and second columns of the resulting matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -\sin(\psi) \\ 0 & \sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} \cos(\phi)\cos(\theta) & -\cos(\phi)\sin(\theta) & \sin(\phi) \\ \sin(\psi)\sin(\phi)\cos(\theta)+\cos(\psi)sin(\theta) & -\sin(\psi)\sin(\phi)\sin(\theta)+\cos(\psi)cos(\theta) & -\sin(\psi)\cos(\phi) \\ -\cos(\psi)\sin(\phi)\cos(\theta)+\sin(\psi)sin(\theta) & \cos(\psi)\sin(\phi)\sin(\theta)+\sin(\psi)cos(\theta) & \cos(\psi)\cos(\phi) \end{bmatrix} \tag{5.1}$$

- A quaternion [81]: This representation involves transmitting a four-tuple (w, x, y, z) encoding a rotation vector $\vec{R}(r_x, r_y, r_z)$ and angle θ, such that $w = \cos\left(\frac{\theta}{2}\right)$ and (x, y, z) represents $\vec{R}$ normalized to length $\sin\left(\frac{\theta}{2}\right)$. To determine the local coordinate system orientation from this information, one would rotate the coordinate system by angle θ about vector $\vec{R}$. The vectors $\vec{OX}$ and $\vec{OY}$ are extracted from the first and second columns of the resulting matrix:

$$\begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{5.2}$$

These formats require the remote host to extract axis point coordinates from the information provided in the update packet, thereby effectively trading off bandwidth for additional computational complexity.

5.1.2 Receiver Packet Processing

As shown in Figure 5.2, each remote host first recovers the vectors $\vec{OX}$ and $\vec{OY}$ from the information contained in the orientation update packet, and each of these vectors is dead reckoned using PHBDR's adaptive tracking and convergence algorithms as described in Chapter 3.

On each frame, the host queries the appropriate PHBDR modules for the current extrapolated position of the axis points $X(1, 0, 0)$ and $Y(0, 1, 0)$ which are being modeled in the entity's local coordinate system. It computes the cross-product $X \times Y$ to produce the current position of $Z(0, 0, 1)$. These three vectors directly provide the three columns for the 3×3 rotation matrix, which after being orthogonalized, is used to orient the entity's local coordinate system. The host applies this rotation matrix to each entity polygon and then applies a translation within the world coordinate system using the dead reckoned position of the entity's origin. Finally, the host renders the polygon

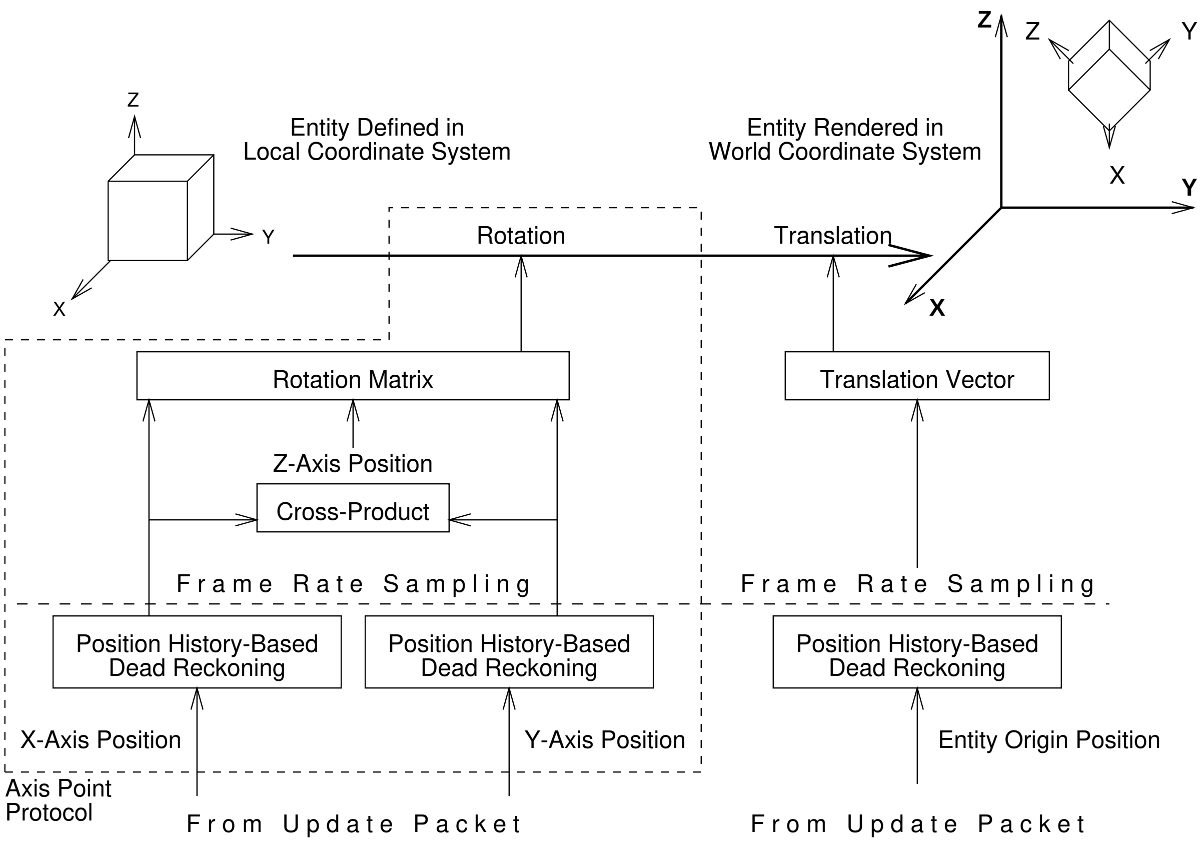

Figure 5.2: Entity Rotation and Translation in World Coordinate System

onto the screen.

5.2 Analyzing Axis Point Protocol Behavior

Suppose the true coordinate system A and the dead reckoned coordinate system A' differ by a rotation of angle θ about unit vector $\vec{R}(r_x, r_y, r_z)$. The Axis Point protocol transmits an orientation update when the remote tracking error for $X(1,0,0)$ or $Y(0,1,0)$ exceeds a protocol error threshold δ, reflecting how much the true and dead reckoned coordinate systems differ. Therefore, when an

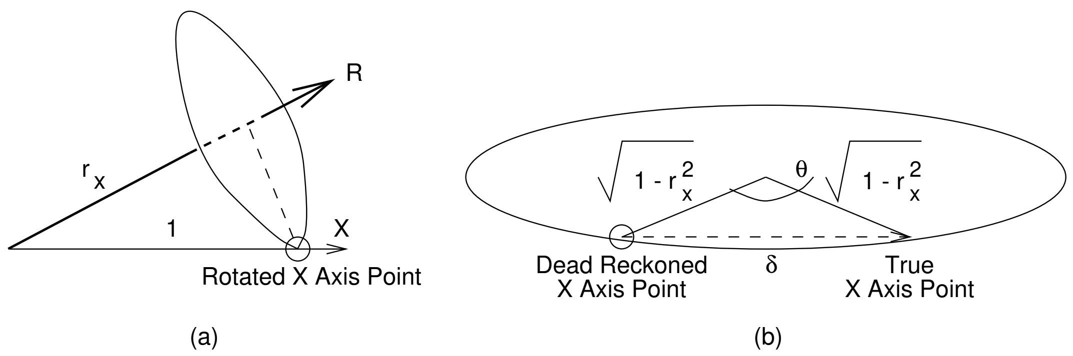

Figure 5.3: Relationship Between Rotation and Position Threshold on X Axis Point: (a) As coordinate system rotates, axis point moves along circle on surface of unit sphere; (b) Position error is the length of a chord through this circle. The behavior of the Y axis point is similar.

update is transmitted,[1]

$$\theta = 2\sin^{-1}\left(\frac{\delta}{2\sqrt{1-\min(r_x^2, r_y^2)}}\right) \tag{5.4}$$

Figure 5.3 illustrates the derivation of this expression. As the coordinate system rotates about $\vec{R}$, the X axis point maps out a circle of radius $\sqrt{1-r_x^2}$ along the surface of the unit sphere. The true and dead reckoned coordinate systems therefore place the X axis point at different locations along this circle, so the position error measured by the PHBDR protocol is actually the length of a chord through that circle. Similarly, the Y axis point moves along a circle of radius $\sqrt{1-r_y^2}$.

By expanding the above equation, we see that $\theta \in \left[2\sin^{-1}\left(\frac{\delta}{2}\right), 2\sin^{-1}\left(\frac{\delta}{\sqrt{2}}\right)\right]$ with the exact value determined by the direction of axis $\vec{R}$. If $\vec{R}$ is perpendicular to one of the axis point vectors $X(1,0,0)$ or $Y(0,1,0)$, then θ is at the bottom of its range, and θ is maximized when $\vec{R}$ is parallel to $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$. Assuming no axis bias in the entity's rotation, θ averages

$$\frac{2\int_0^{\frac{1}{\sqrt{2}}} \sqrt{1-r^2}\cos^{-1}\left(\frac{r}{\sqrt{1-r^2}}\right)\sin^{-1}\left(\frac{\delta}{2\sqrt{1-r^2}}\right)dr}{\int_0^{\frac{1}{\sqrt{2}}} \sqrt{1-r^2}\cos^{-1}\left(\frac{r}{\sqrt{1-r^2}}\right)dr} \tag{5.5}$$

[1]More generally, if the axis points are $\vec{P}_0, \vec{P}_1, \ldots, \vec{P}_k$, then when an update is transmitted,

$$\theta = 2\sin^{-1}\left(\frac{\delta}{2\sqrt{1-\min((\vec{R}\cdot\vec{P}_0)^2, (\vec{R}\cdot\vec{P}_1)^2, \ldots (\vec{R}\cdot\vec{P}_k)^2)}}\right) \tag{5.3}$$

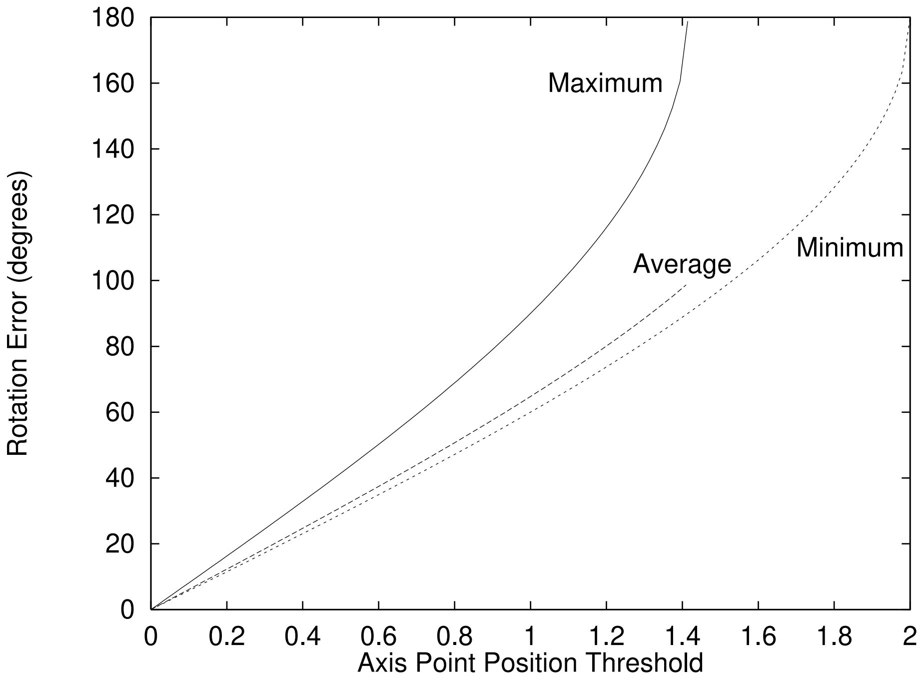

Figure 5.4: Relation Between Orientation Threshold and Rotation Angle

We computed values of this average by approximating the integrals as a sum with dr intervals of 10^{-6}. Figure 5.4 plots the minimum, average, and maximum values of θ when an orientation update is transmitted. Although the axis point error thresholds do not provide for a fixed coordinate system error, the graph shows that the error range of θ is relatively small for expected orientation thresholds under 45 degrees.

As we have seen, steady-rate motion about a fixed rotation axis causes each axis point to follow a circular path along the surface of the unit sphere. Moreover, for a given rotation axis $\vec{R}$, only one of the axis points will cause update packet generation, namely the axis point $\vec{P_i}$ yielding the smallest value of $(\vec{R} \cdot \vec{P_i})^2$. Consequently, analysis of the Axis Point protocol is reduced to understanding PHBDR performance on simple circular motion, an entity behavior that we studied in Section 4.2.1.2. Figure 5.5 simply uses the data from Figure 4.2, combined with the position threshold information from Figure 5.4, to describe the packet rate behavior of the Axis Point protocol. For various rotation rates, the figure shows the axis point update rate required to produce a given average rotation error in the remote model. We can make several important observations from this data. First, we observe that the update rate rises linearly with the rotation rate; this linear relationship is expected, based on the data in Figure 4.2b, because acceleration increases with the

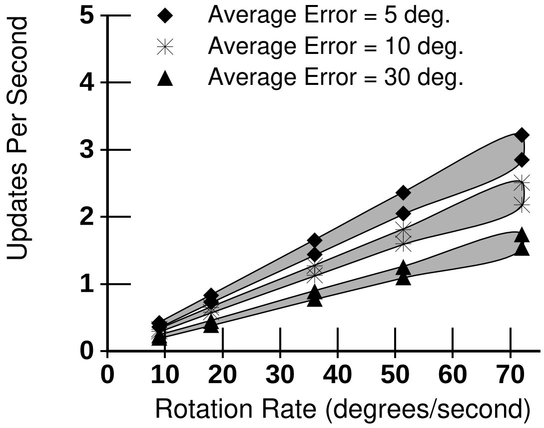

Figure 5.5: Packet Rate to Maintain Desired Average Rotation Error Under Steady Rotation

square of the rotation rate. Second, as the error tolerance is reduced, the packet rate increases more rapidly. This result is simply a reflection of the Curve Type Tradeoff observation seen for circular motion. Finally, the variable rotation angle tolerance represented by axis point positional error (as depicted in Figure 5.4) only translates into a 10–15% variability in update rate.

5.3 Evaluating the Axis Point Protocol

The Axis Point protocol provides several advantages over competing orientation dead reckoning protocols, such as Euler (Tait-Bryan) angles [76] used by the Distributed Interactive Simulation (DIS) dead reckoning protocol [36] and quaternions [81] used in newer distributed simulation systems such as NPSNET [18]. These advantages fall into three categories: minimal assumptions about the source entity model, reduced code complexity and size, and improved numerical performance. After discussing these advantages in turn, we describe some open issues with the effective use of the Axis Point protocol.

5.3.1 Decoupling From Source Entity Model

The biggest advantage of the Axis Point protocol lies in its minimal assumptions about the entity model being used by the source host. In particular, the protocol only requires the source to provide the position of certain vertices in the entity's local coordinate system. The Axis Point protocol effectively decouples the orientation dead reckoning protocol from the actual orientation

Representation of Entity Orientation	**Computation To Obtain Axis Points**
Rotation Matrix	None
Euler Angles	6 sin/cos, 6 multiplications, 2 additions
Quaternion	14 multiplications, 7 additions

Table 5.1: Computation Required to Obtain Axis Point Data

representation in the entity model.

In fact, as shown in Table 5.1, axis point information is readily available no matter how the entity's orientation is actually represented at the source host. Notably, most common graphic systems already use rotation matrices for rendering. Axis point coordinates can be extracted directly from the source host's rotation matrix, and at the remote host, they directly produce a rotation matrix for rendering. On the other hand, quaternions do not provide an efficient representation of orientation if the entity model uses rotation matrices or Euler angles to model the entity. A numerically stable conversion from a rotation matrix to a quaternion requires eight multiplications, 17 additions, and two square roots [28], and converting Euler angles to a quaternion requires six trigonometric functions, four additions, and 15 multiplications. Corresponding inverse conversions are equally expensive at the remote host. This computation is significant because it must be incurred on each frame at the source in order to test whether an orientation update is required.

The minimal assumptions made by the Axis Point protocol on the source entity model differ significantly from the high data requirements of Euler angles used by the DIS protocol. The DIS protocol requires the source host to explicitly model the entity's angular velocity and acceleration and transmit those values in update packets. However, accurate angular velocity and acceleration information is not readily available for all entities—particularly for live entities connected to the simulation via motion sensors. Consequently, the Axis Point protocol is easier to implement over a broader variety of entity models.

As we have seen, the Axis Point protocol does not require the source host to explicitly model the entity's angular velocity and acceleration. The source host is also not required to explicitly model the axis point velocity and acceleration within the entity's local coordinate system. The Axis Point protocol is recursively structured above the PHBDR protocol that only requires the vertex position to perform extrapolation and convergence. Notably, the Axis Point protocol would be less effective if it relied on the DIS protocol to track the axis points, because axis point velocity and acceleration information is certainly not ordinarily available from an entity model.

5.3.2 Code Complexity and Size

By employing recursive protocol structuring on top of the PHBDR protocol, the Axis Point protocol implementation does not need to provide significant new functionality and is, therefore, relatively easy to implement. At the source host, the Axis Point module simply samples the entity model to obtain the axis point representation and pass those coordinates to two instances of the PHBDR modules. When one of the PHBDR modules desires to transmit an update packet, the Axis Point module constructs a packet containing information about both axis points. At the remote host, the Axis Point module simply needs to unbundle the orientation update packet and pass the contained axis points to two instances of the PHBDR modules. It must also sample the dead reckoned axis point positions on each frame and construct a rotation matrix. In summary, the Axis Point software is only responsible for sampling entity orientation at the source host, packaging and unpackaging orientation update packets, and providing the entity orientation upon request at the remote host. By reusing the already tested algorithms provided by the PHBDR modules, we significantly reduce the development time and ease the software debugging when compared to a orientation dead reckoning protocol that requires an entirely new set of supporting algorithms for extrapolation and convergence. Notably, quaternion protocols are privy to subtle programming errors.

Because of its limited additional functionality, the Axis Point protocol requires almost no new code at either the source or destination hosts. In our implementation, the Axis Point protocol introduces approximately 100 new lines of C ++ code on top of the PHBDR implementation (not counting general-purpose support routines for manipulating matrices). As a point of comparison, the basic PHBDR protocol requires about ten times as much code to implement packet generation, tracking, and convergence. If we were to implement the Axis Point protocol without the benefit of the PHBDR protocol base, we would have needed to re-implement most of this functionality. Indeed, as we will see in the next section, dead reckoning protocols that use Euler angles and quaternions end up requiring far more complex tracking and convergence algorithms than those used by the PHBDR protocol.

The recursive protocol structuring greatly simplified the complexity of the protocol analysis. We were able to analyze the Axis Point protocol by simply expanding on our analysis of the PHBDR protocol performance on circular vertex motion. Without the benefits of a base protocol, the protocol analysis would have required an evaluation of all rotation behaviors.

Finally, the simplicity of the Axis Point protocol gives it the same advantage offered by the PHBDR protocol, namely the flexibility to be customized to support specialized entity requirements. First, different thresholds may be introduced for each axis point to reflect the remote model fidelity

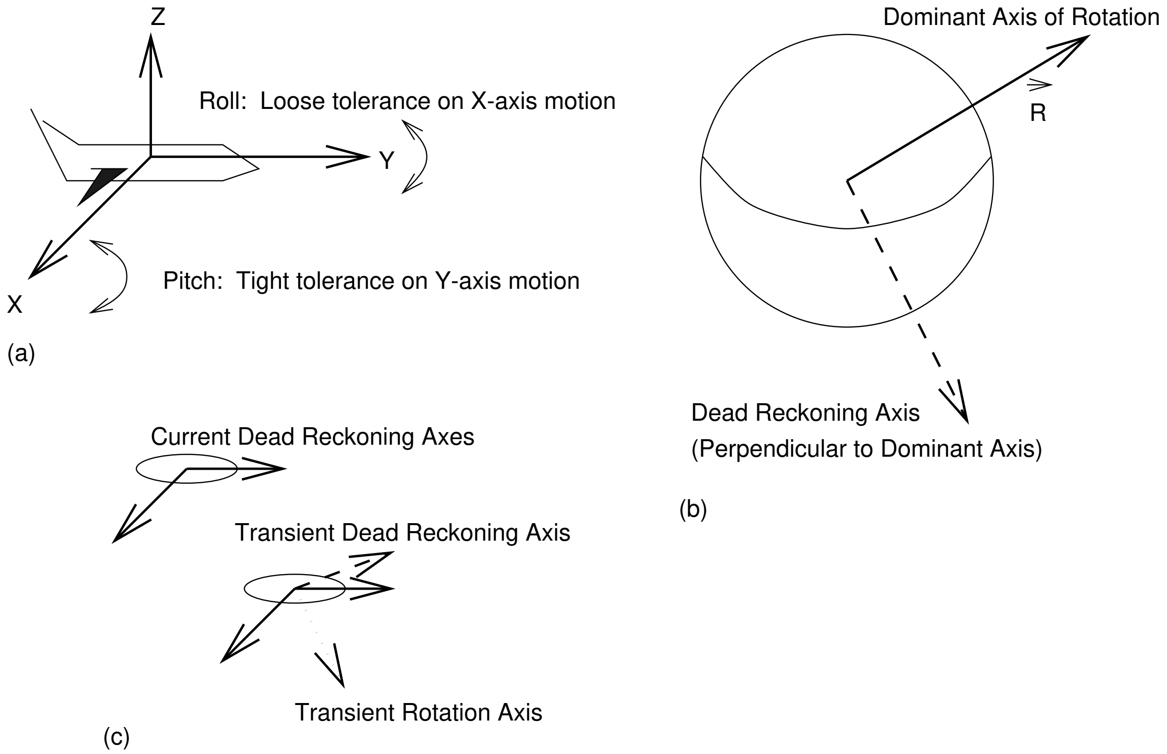

Figure 5.6: Extending Axis Point Dead Reckoning for (a) Dominant Rotation Axis, (b) Non-Aligned Rotation Axis, and (c) Multiple Rotation Axes.

required for different types of rotations. For example, when modeling an airplane in flight, remote viewers may be more sensitive to inaccuracies in the plane's pitch than to inaccuracies in the plane's roll. As shown in Figure 5.6a, a tighter threshold would be introduced about the Y axis point (whose motion represents rotation about the X axis point). Second, axes other than $\vec{X}$ and $\vec{Y}$ may be used, depending on the entity's dominant rotation behavior. In changing the axis point vectors, we exploit our earlier observation that the Axis Point protocol generates update packets more aggressively for rotations that are perpendicular to the axes being dead reckoned. For example, if the entity usually rotates about axis $\vec{R}$, then modeling an axis perpendicular to $\vec{R}$ allows more accurate remote orientation modeling by causing update packet generation to occur consistently at the minimum end of the allowable rotation range, as illustrated in Figure 5.6b. Third, the Axis Point protocol can support more than two axes for complex entities, and axes may be added or removed dynamically from the remote model. Figure 5.6c shows an entity which is about to undergo a transient rotation about a fixed axis. By introducing an additional axis point for dead reckoning, the simulation can guarantee precise modeling of that rotation. This real-time flexibility allows the simulation to adapt

Operation	Axis Point Cost		Quaternion Cost		
	Add	Mult	Add	Mult	Cos/Acos
Packet Arrival: Tracking					
Second-Order	32	24	49	109	12
First-Order	6	6	11	23	2
Packet Arrival: Convergence					
Second-Order	54	36	48	102	10
First-Order	6	6	11	23	2
Per Frame:					
Generate Rotation Matrix	6	3	15	9	

Table 5.2: Comparison of Packet Processing Computation in Axis Point and a Quaternion-Based Protocols

to changing entity behavior.

5.3.3 Numerical Performance

The Axis Point protocol provides a computationally efficient technique for providing second-order dead reckoning of entity orientation. Based on our analysis in Chapter 4, the protocol only requires between 84 and 162 arithmetic operations to process an orientation update, and it requires almost no computation between updates. The Euler angle approach, however, is far more complex because the rotation angles are not independent of one another. As a result, first-order orientation dead reckoning requires roughly 180 operations to process a packet and an additional 100 operations on each frame [91]. Second-order orientation dead reckoning with Euler angles is rarely done because of its computational complexity. Moreover, although first-order dead reckoning using quaternions is not computationally expensive, a second-order protocol using quaternions requires roughly twice the computation as the Axis Point protocol. Table 5.2 compares the computational requirements of the two protocols. Quaternions are considerably more costly than axis point vectors to manipulate. Quaternion addition involves a 4-dimensional vector cross product, while multiplying a scalar with a quaternion requires a renormalization of the result.

The Axis Point protocol also does not rely on trigonometric operations whose operations are notoriously inaccurate and vary considerably across platforms. By using only arithmetic operations, the Axis Point protocol minimizes the propagation of floating point errors. This characteristic is critical to systems that send packet updates infrequently. On the other hand, the first-order Euler angle protocol effectively solves a differential equation on each frame—an operation that relies on numerically unstable trigonometric and inverse-trigonometric functions [11]. As shown in Table 5.2, second-order dead reckoning with quaternions also requires trigonometric operations to renormalize the quaternions.

Finally, the Axis Point protocol has the advantage that as the entity rotates, the orientation parameters that it produces are continuous. That is, the axis points physically move along the surface of the unit sphere, so they do not exhibit any "jumps" or other anomalies. Furthermore, if the rotation is constant, then the axis points move in a uniform (circular) manner. However, as an entity rotates, the Euler angles are discontinuous or ill-defined. For example, as an entity rotates about the Z-axis, the Euler rotation angle increases from 0 degrees to 359 degrees and then returns to 0 degrees. There is also a singularity point at which the entity's orientation may be defined by an infinite number of Euler angle tuples. Remote hosts must compensate for these discontinuities, but the problem is typically underconstrained because of insufficient information between updates.

5.3.4 Limitations of the Axis Point Protocol

The biggest drawback of the Axis Point protocol is the independence of the six vector coordinate models. The basic PHBDR protocol imposes no constraints on the vector coordinates, so the dead reckoned axis points are not guaranteed to lie on the unit sphere. Consequently, each axis point vector must be renormalized during each frame. Similarly, because PHBDR does not ensure that the dead reckoned axis point vectors will always be orthogonal, the X-axis and Y-axis vectors may need to be made orthogonal by projecting one vector onto a plane perpendicular to the other. The normalization and orthogonalization operations are straightforward and do not introduce much computational overhead, but they may cause occasional anomalies in the orientation modeling, particularly when an entity is rotating rapidly. These anomalies are short-lived, however, and rare enough to be shadowed by the other advantages of the Axis Point protocol.

The other significant drawback of the protocol lies in the ambiguity of the axis point error thresholds in determining the tolerable rotation error. As shown in Figures 5.4 and 5.5, a given axis point error represents a *range* of rotation errors which may translate into a 10%–15% variability in the orientation update rate. We do not deem this variation to be significant, however, and expect simulation designers to set axis point error thresholds based on the associated maximum rotation error.

5.4 Conclusion

In this chapter, we have presented the Axis Point protocol which supports the remote modeling of entity orientation. The protocol shares many of the advantages of the PHBDR protocol, most notably its minimal dependencies on the entity model itself. The Axis Point protocol only relies on the entity model to provide the local coordinate position of particular vertices and, unlike alternative protocols, does not require the model to provide the angular velocity or acceleration. The dead reckoning of orientation is also decoupled from the actual representation of orientation used by the model itself. Finally, because it recursively uses the PHBDR protocol, the Axis Point protocol does not require velocity or acceleration information about the axis point vectors, so it does not impose new modeling requirements on the source host.

By recursively using the PHBDR protocol's tracking and convergence algorithms, the Axis Point protocol also provides the following benefits:

- Reduced implementation complexity and size, resulting in faster development time and easier debugging.
- Greatly simplified analysis by reusing results from the PHBDR protocol analysis to understand the Axis Point protocol.
- Minimal computational complexity and numerical stability while providing a second-order dead reckoning model.

We have also seen that a recursive protocol structure for entity orientation does come with some costs. For example, because PHBDR does not constrain the axis point vectors, the remote host is responsible for normalizing and orthogonalizing the dead reckoned vertex positions on each frame. Moreover, the protocol's precise behavior depends on the particular axis around which the entity is rotating; positional error on one of the axis points therefore represents a range of possible rotation angle error values, rather than a single rotation angle error. Clearly, a dead reckoning protocol designed specifically to handle entity orientation could eliminate such inefficiencies, but by providing such optimizations, we would lose the simplicity provided by the recursive protocol structure.

Together, the Position History-Based protocol and the Axis Point protocol provide a mechanism for remote modeling a rigid entity's position and orientation. In the next chapter, we continue our exploration of recursive protocol structure on the PHBDR protocol by considering protocols that support remote modeling of non-rigid and semi-rigid entity structures at variable fidelities.

Chapter 6

Multiple-Detail Channels for Modeling Non-Rigid Entities

Up to this point, we have associated each entity with a single multicast address over which it transmits position and orientation updates, and we have targeted an average of one update packet per second. However, position updates with a single error threshold cannot satisfy the data requirements of all hosts in a large-scale simulation. On one hand, nearby viewers expect to see the entity rendered with full graphical detail and with maximum structural and positional fidelity. These users require update rates approaching the frame rate, and they expect the update packets to provide information about the entity's dynamic structure and articulated parts that move independently from the entity's body motion. On the other hand, distant viewers can tolerate rendering the entity with less graphical detail and with less structural and positional fidelity. Each viewer may see hundreds of entities, and receiving high-frequency updates and detailed structural information from each one imposes an excessive bandwidth and computational demand, thereby limiting the scalability of the simulation.

In this chapter, we address this need for entities to provide high-frequency updates with structural detail for nearby viewers without burdening all other hosts that only require low-frequency updates with minimal structural detail. In the next section, we describe our solution, multiple-detail channels, which recursively uses the PHBDR and Axis Point protocols described in Chapters 3 and 5. We then present an extended example of one implementation of this multiple-detail channel architecture.

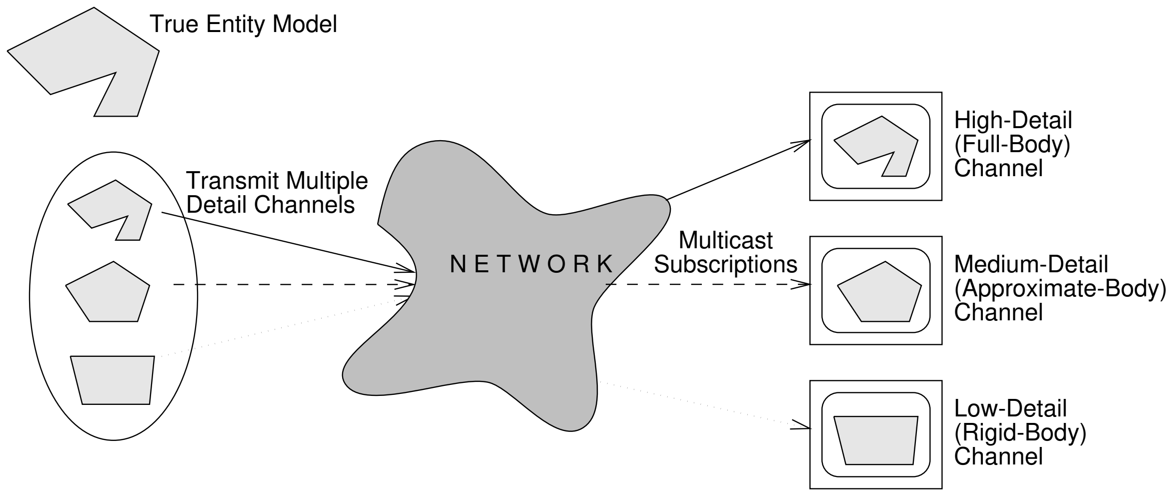

Figure 6.1: Multiple-Detail Channels Provide Independent Streams of Entity Update Information

6.1 The Multiple-Detail Channel Architecture

Each entity provides *multiple-detail channels*, separate data streams, each providing data to support remote modeling at a different level-of-detail, as shown in Figure 6.1. The source host associates each channel with its own multicast address, a different model of the entity's structure, and different PHBDR error thresholds. For example, a high-detail channel satisfies nearby viewers by describing the motion of all vertices in the entity's structure and by using small error thresholds on each of those vertices. A low-detail channel, on the other hand, satisfies distant viewers by describing very little about the entity's dynamic structure and by using high error thresholds for the entity's motion. The source host transmits an update packet to a channel's multicast group whenever the remote modeling error for a vertex exceeds its error threshold for that channel.

Each remote host independently subscribes to the channel(s) whose updates support the graphical detail and modeling fidelity required by the local viewer without exceeding the locally available bandwidth and computational resources. To allow remote hosts to locate and select the channels that are available for an entity, the distributed simulation must provide a directory service. In response to a query identifying an entity, the directory should provide the address of the source host for that entity, the multicast address for each channel, a description of the structural model associated with each channel, a list of PHBDR error thresholds associated with each vertex in the structural model, and an estimate of the target update rate for the channel.

Source hosts face a tradeoff in deciding how many channels to provide for an entity. By offering more channels, the source increases the chances that each remote host can select a channel that

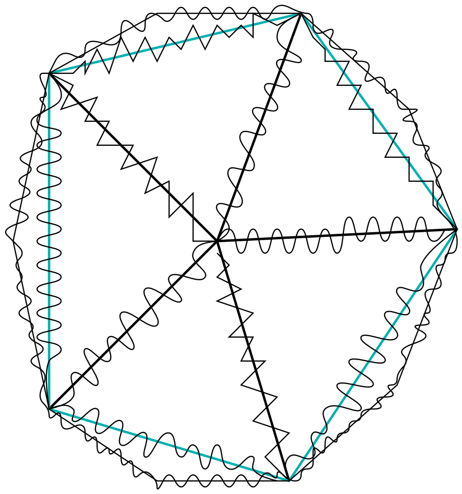

Figure 6.2: Sample Entity for Modeling Multiple-Detail Channels: Jello Icosahedron With Vertices Connected by Springs

closely matches the local modeling fidelity, bandwidth, and computation requirements. However, each supported channel also imposes a cost, both in terms of computation at the source host (it must model vertices for each channel independently) and in terms of bandwidth in the network links near the source (because the first hops must carry traffic for most of the channels).

To satisfy this tradeoff between supporting a large variety of remote modeling needs while controlling computation and bandwidth near the source, the source only provides three channels for each entity. The channels provide order-of-magnitude differences in structural and positional fidelity and also provide order-of-magnitude differences in packet rate. The three channels—which we refer to as the rigid-body channel, approximate-body channel, and full-body channel—respectively support far-range, mid-range, and near-range viewers.

In the next three sections, we define each of these channel types and describe an example implementation based on the SGI "jello" application. As illustrated in Figure 6.2, the jello is modeled as an icosahedron whose vertices are connected by elastic springs. The application places the jello entity inside a rotating cube. As the jello bounces onto the sides of the cube, the collisions deform the springs between the jello's vertices and cause changes to the jello's physical structure. The viewer consequently sees a "wiggling" jello entity.

6.2 Rigid-Body Channel for Far-Range Viewers

Over a *rigid-body channel*, the source host transmits information that allows remote hosts to model the entity as a rigid body, hence ignoring all changes to the entity's structure. Because it does not provide dynamic structural information, rigid-body channels require the least network bandwidth. In addition, because remote hosts do not need to model the entity's dynamic structure, the models supported by this channel impose the least computational demands on remote hosts.

Remote hosts subscribe to the rigid-body channel when the limited computational and network resources are better allocated toward modeling other entities at higher structural and positional fidelity. Consequently, this channel is used when the entity is distant from the local viewer, so structural changes would be imperceptible or uninteresting. In a large simulation in which hundreds or thousands of entities are visible to each host, hosts must subscribe to rigid-body channels for most visible entities to avoid consuming excessive bandwidth and computational resources.

The channel is also appropriate for entities whose structure changes rarely. For example, if an entity's structure only changes after it is involved with a significant collision, then the rigid-body channel does not burden hosts with real-time structural information that is usually unchanging. Instead, over a rigid body channel, the source host can simply disseminate a completely new structural representation after a rare significant structural change occurs.

Rigid-body channels can support two types of rigid body entity models: Position-Only and Position-and-Orientation.

6.2.1 Position-Only Model

To support a *position-only model*, the source applies the basic PHBDR protocol to the entity and only transmits the entity's position. For rigid entities, the entity position is simply the location of the entity's local coordinate system origin within the virtual world's global coordinate system. For non-rigid entities whose local coordinate system is ill-defined, the entity's position either be the location of a designated entity vertex or the entity's center-of-mass. After the entity undergoes a significant structural change, the source host transmits a new structural model along the channel.

For a position-only model of the jello entity, for example, the source transmits the location of the jello's center vertex. The remote model ignore the motion of the other vertices and instead represents the entity with its default structure and default orientation. Simulating the behavior of the position-only model of the jello application, we derive the data in Figure 6.3a which shows the average remote modeling error for the 12 surface vertices in the jello model, as a function

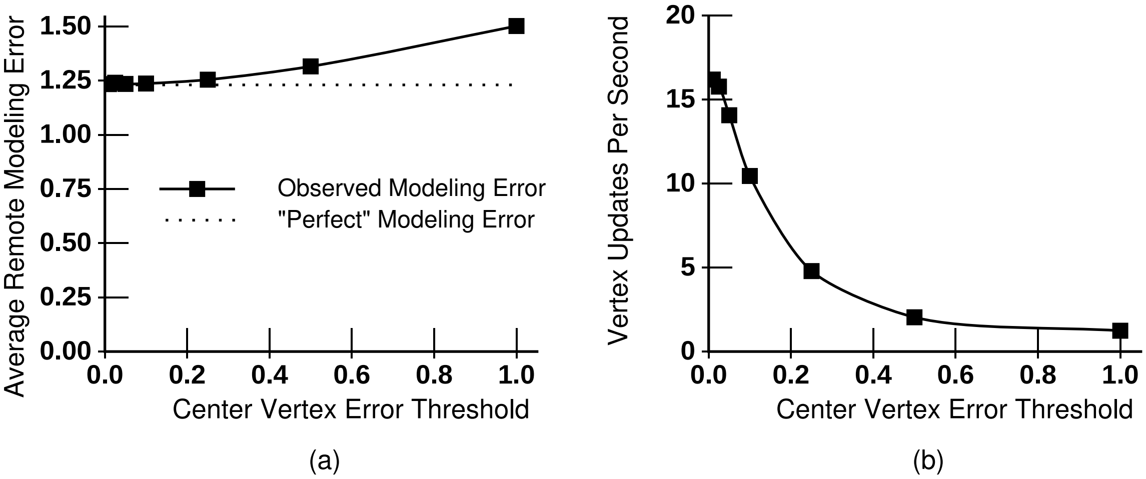

Figure 6.3: Position-Only Model for the Jello Application: (a) Modeling Error and (b) Vertex Update Rate

of the error tolerance on the jello's center point position. Most of this error arises because the remote host does not model the orientation of the entity. Because the jello has a diameter of two, a "perfect" position-only model of the jello would produce an average error of $\frac{4}{\pi} \approx 1.27$ on the surface vertices.[1] Any error above this minimum arises from inaccuracies in modeling the jello's center point. Figure 6.3b shows the relationship between the error threshold and the resulting update rate produced by the channel. The graph reveals that accepting a 20% increase in modeling error reduces the packet rate by over 90%. For the jello, we see that the observed error and packet rate behavior are similar to that observed for PHBDR with "bouncing" motion in Figure 4.4.

6.2.2 Position-and-Orientation Model

To support a *position-and-orientation model*, the source host uses the PHBDR protocol to transmit entity position information and uses the Axis Point protocol to transmit entity orientation information. For rigid entities, the source can directly extract axis point vectors from the entity model. However, as illustrated in Figure 6.4, estimating the orientation of non-rigid entities is harder because the individual vertices do not have a fixed location with respect to each another. To compute the entity's orientation in this case, the source must approximate its dynamic structure in terms of the entity's rigid structure model. As shown in Figure 6.4c, the source ideally would calculate

[1]In reality, the observed "perfect" error is closer to 1.23 because the jello's rotation is not perfectly uniform and is instead slightly biased toward its initial orientation.

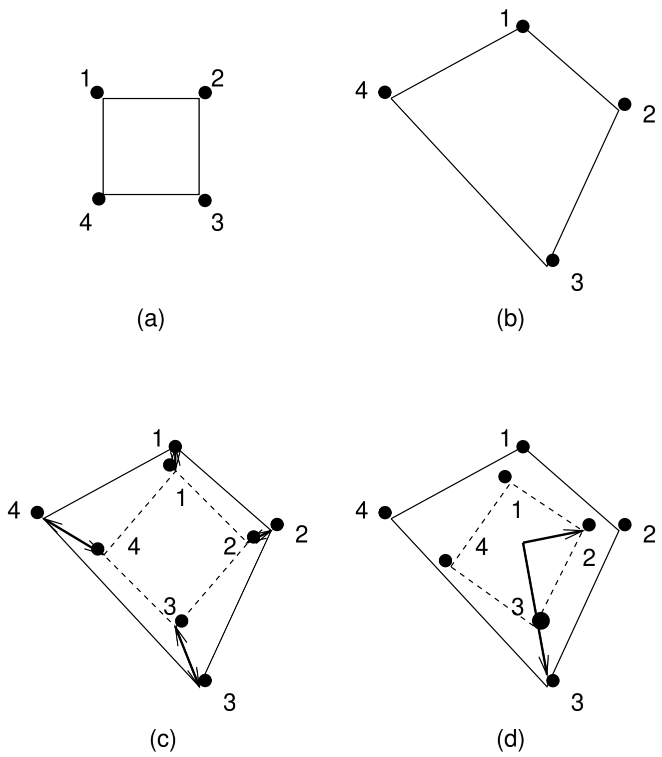

Figure 6.4: Estimating Orientation of Non-Rigid Entities: (a) Rigid structure of entity; (b) Dynamic structure of entity; (c) Optimally orienting the rigid model to minimize error between rotated rigid vertices and actual entity vertices; (d) Approximation using one axis point vector directly and orthogonalizing other axis point vector.

the orientation of the rigid entity model that minimizes the total (or maximum) error between the rotated rigid body vertices and the entity's dynamic vertices. This computation would be performed on each frame to determine the appropriate axis point vectors that the Axis Point protocol module would process and later transmit in update packets. However, this orientation computation is effectively a non-linear optimization problem and is therefore infeasible for real-time use. To achieve real-time performance, we instead rely on an approximation to the true axis point vectors, as shown in Figure 6.4d. The source host first computes the vectors joining the entity's center point to the axis points. One of the axis points (point 3 in the figure) is selected as a reference point, and the other axis point vector (corresponding to point 4 in the figure) is adjusted to form the appropriate angle with the reference vector, as required by the rigid model of the entity. Using this approximation, the reference axis point vector is more accurate and the second axis point vector is less accurate than the vectors produced using the ideal optimization approach. However, we deem that the loss of positional fidelity is well justified by the savings in computational complexity.

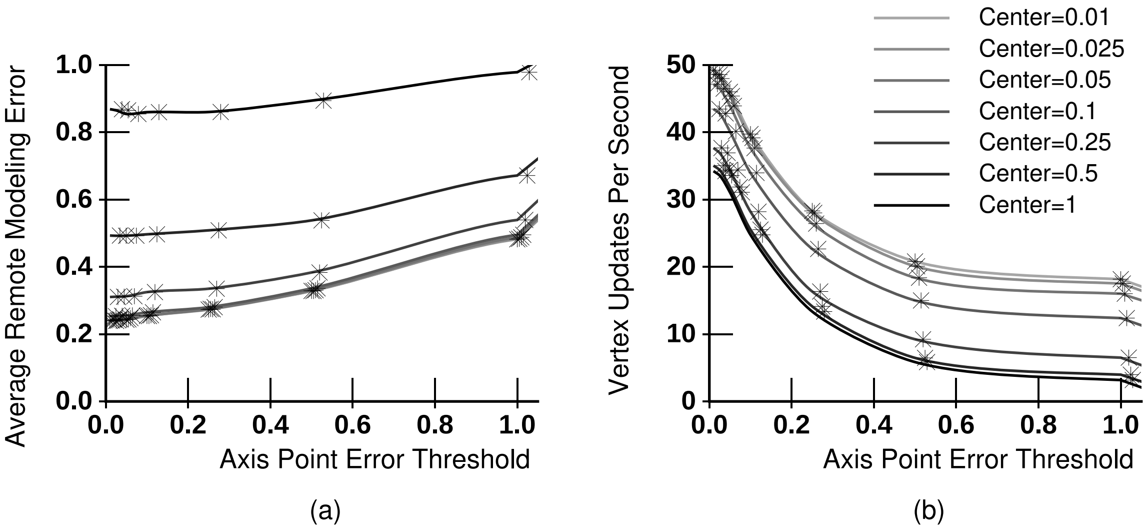

Figure 6.5: Position-and-Orientation Model for the Jello Application: (a) Modeling Error and (b) Vertex Update Rate

For example, our implementation of the jello application uses the rigid-body vertices $V_1(0,0,1)$ and $V_2(0.900665, 0, 0.434515)$ as axis points. The source first computes the vectors $O\vec{V}_1$ and $O\vec{V}_2$ formed by joining the entity's center $O(0,0,0)$ to V_1 and V_2 respectively. The first axis point vector is formed by simply normalizing $O\vec{V}_1$. The second axis point vector is computed by normalizing $O\vec{V}_2$ and rotating it about the vector $(O\vec{V}_1 \times O\vec{V}_2)$ until it forms an angle $\cos^{-1}(0.434515)$ from $O\vec{V}_1$. The resulting axis point vectors are then provided to the Axis Point protocol module to determine whether an orientation update packet should be transmitted.

Simulating the behavior of the position-and-orientation model of the jello application, we obtain the data in Figure 6.5 which shows the average remote modeling error of the 12 surface vertices in the jello model and the vertex update rate, as a function of the error tolerance for the axis points. Curves are shown for different error tolerances on the center vertex position, corresponding to the values shown in Figure 6.3. Whenever either axis point requires an update, we transmit both axis points, so each axis point update is treated as two vertex updates for the purposes of measuring the packet rate. Figure 6.5a reveals that a position-and-orientation model yields an average surface vertex error that is 35%–80% lower than that produced by a position-only model. Figure 6.5b reveals that the resulting packet rate is 12%–272% higher than that of a position-only model. We see that simply providing orientation information provides a significant reduction in modeling error, but as the error threshold is tightened, the significant bandwidth requirements are not balanced by correspondingly large improvements in modeling fidelity.

6.3 Approximate-Body Channel for Mid-Range Viewers

Over an *approximate-body channel*, the source host transmits information that allows remote hosts to model an approximation of the entity's dynamic structure. Remote hosts still model the entity using a rigid structure, but they dynamically adjust that rigid structure in a controlled manner according to some transmitted parameter(s). Such models attempt to isolate the entity's overall motion within the global coordinate system from the structural motion made by individual vertices in the local coordinate system.

Remote hosts subscribe to the approximate-body channel for non-rigid entities that are close enough so that the local viewer can notice structural changes but far enough for the viewer to tolerate some inaccuracy in the structural representation. Because it provides some information about the entity's structure, this channel consumes more bandwidth than a rigid-body channel, and receivers must dedicate more computational resources to handle the higher update rate and dead reckon the additional entity attributes.

The channel may also be appropriate for entities whose structural change is small compared to its overall translational and rotational motion. For example, approximate-body channels are appropriate for modeling a human because the attached arms and legs have limited range of movement [71] or for modeling a tank whose turret only rotates about a fixed attachment point. In these cases, the source host only needs to transmit the articulated part's angle of rotation about its attachment point.

To develop a good entity model for an approximate-body channel, the simulation developer must consider entity-specific information—particularly regarding what types of structural changes are commonly experienced by the entity—to determine how the rigid-body model may be dynamically adjusted. However, the radial-length model and local-coordinate-vertex model are two example models that do not require significant a priori information about the entity. We now consider them in turn.

6.3.1 Radial-Length Model

The *radial-length model* is best suited for entities whose vertices primarily move radially toward and away from the entity's center point. To support a radial-length model, the source transmits the position of the entity's center point and axis points, much like the rigid-body position-and-orientation model. In addition, the source transmits the entity's current average radius, as shown in Figure 6.6a. The source computes the average radius by measuring the distance between the center point and each of the entity's exterior vertices.

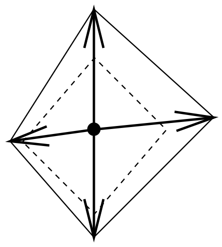

Figure 6.6: Computing Average Radius for a Dynamic Entity Structure: The solid line represents the entity's dynamic structure, and the dashed line represents the entity's rigid structure model.

The remote host maintains a rigid-body representation of the entity structure. It uses the PHBDR and Axis Point protocols to respectively model the entity's position and orientation based on information provided in the update packets. It also uses the PHBDR protocol to dead reckon the entity's average radius. This value is used to dynamically scale the entity's rigid-body structure before rendering.

To apply the radial-length model to the jello entity, the remote host maintains PHBDR proxies for the jello's center point in global coordinates, two axis points on the surface of the unit sphere, and the average radius of the jello's 12 exterior vertices. Simulating the behavior of the Radial-Length Channel on the jello application, we obtain the data in Figure 6.7, which shows the resulting average remote modeling error for the 12 surface vertices in the jello model and the vertex update rate, as a function of the PHBDR protocol error threshold for the average radius. The curves demonstrate behavior with twelve different error tolerance configurations for the center vertex and axis points. As in Figure 6.5, each axis point update is treated as two vertex updates for the purposes of measuring the packet rate.

The graphs reveal that radial information only offers a modest reduction in the average error provided by the position-and-orientation model: on average, the radial length information introduces an 11% increase in packet rate to obtain a 3% improvement in structural fidelity. For the jello, the radial-length model's effectiveness is limited for several reasons. First, each collision between the jello and the rotating cube only displaces a subset of the jello's exterior vertices. Therefore, by computing the average radius and treating all surface vertices as equidistant from the jello's center, the model only loosely approximates the actual behavior of the jello's structure. Second, the individual vertex displacements are relatively small, not exceeding 10% of the jello's radius, so displacements to a single vertex affect the average radius by less than 1%. Such a small change to the parameter does not affect the entity's structural model significantly. Third, because the jello's vertices are connected by springs, vertex displacements exhibit oscillatory motion with high

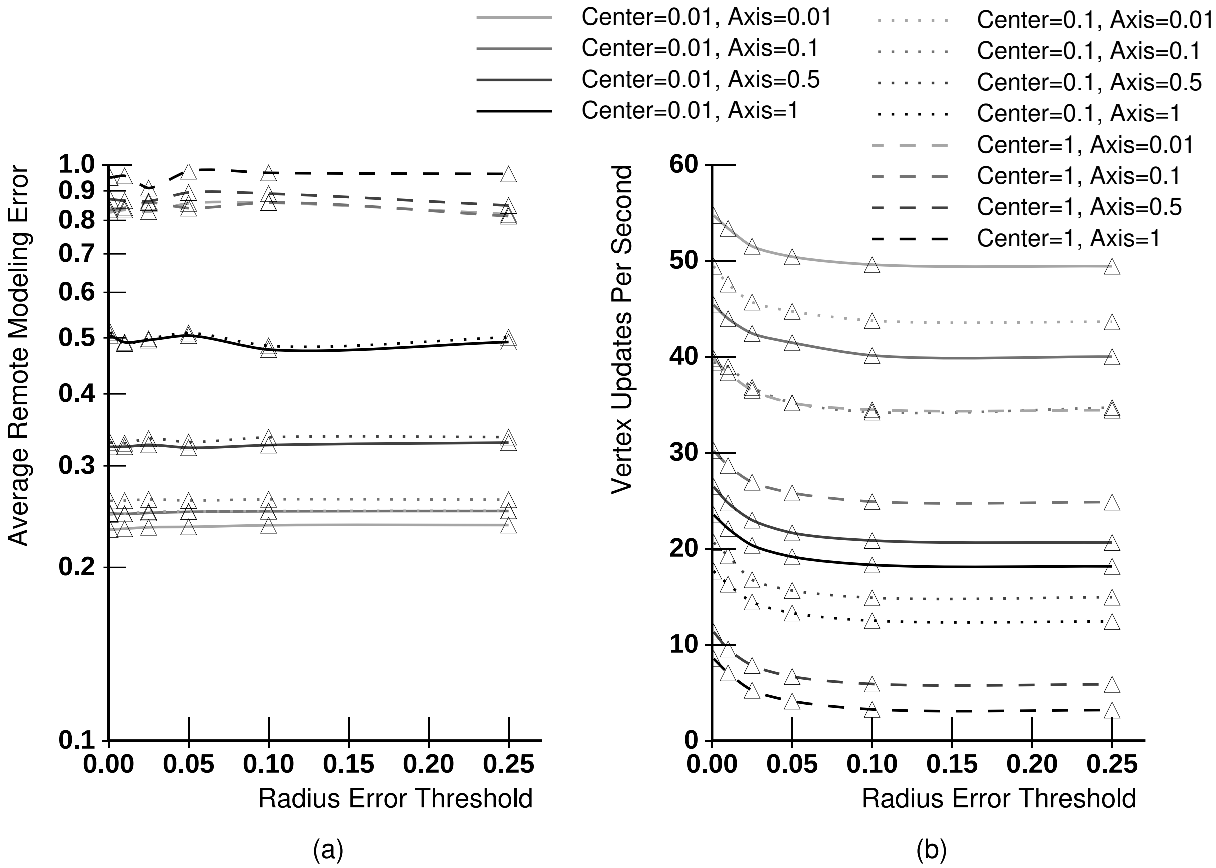

Figure 6.7: Radial-Length Model for the Jello Application: (a) Modeling Error and (b) Vertex Update Rate

frequency which, as discussed in Section 4.2.1, requires high packet rates to model in detail.

We conclude that radial-length models are most appropriate when the entity's exterior vertex motion is closely correlated (e.g. the entity's structure expands and contracts radially as a single unit) and when the entity's structural change is significant when compared to its rigid-body radius. Approximate structural modeling is also easier when the entity's structure does not change too rapidly.

6.3.2 Local-Coordinate-Vertex Model

To support a *local-coordinate-vertex model*, the source transmits the entity's center point and axis points, much like the rigid-body position-and-orientation model. In addition, the source transmits the position of each of the entity's structural vertices within their local coordinate system. The procedure for generating this information is illustrated in Figure 6.8. The source first computes

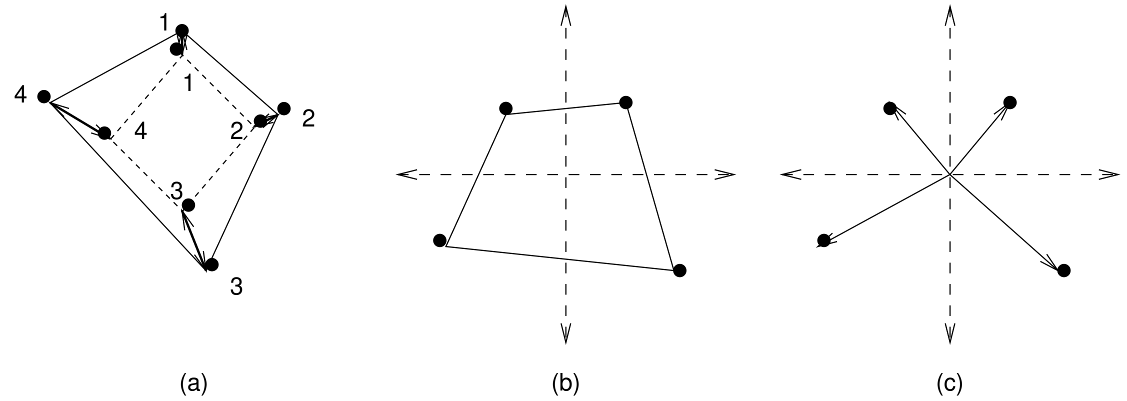

Figure 6.8: Computing Local Coordinate System Position of Entity Vertices: (a) Computed entity position and orientation; (b) Applying translation and rotation in reverse to center at origin; (c) Sampling vertex positions within local coordinate system.

the entity's position and orientation as if it were a rigid body, using the procedure illustrated in Figure 6.4d. Having computed the position and orientation, the source reverses the translation and rotation to center the entity at the origin of its local coordinate system. The resulting vertex positions describe the entity's new structure and are transmitted in the update packet.

The remote host uses the PHBDR and Axis Point protocols to respectively model the entity's position and orientation based on information provided in the update packets. It also dead reckons the position of each structural vertex within the entity's local coordinate system. By tracking the motion of these vertices, the host maintains an approximation of the entity's dynamic structure.

To render the entity, the host effectively reverses the steps of Figure 6.8. It constructs an entity geometry by retrieving the current dead reckoned position of the structural vertices. It then applies the rotation indicated by the dead reckoned axis points and finally translates the entity based on the dead reckoned center vertex position.

For example, to apply a local-coordinate-vertex model to the jello entity, the remote host maintains PHBDR proxies for the jello's center point in global coordinates, two axis points on the surface of the unit sphere, and the 12 surface vertices in the jello's local coordinate system. Within the local coordinate system, each of these surface vertices moves within a small tolerance governed by the elasticity of the springs attaching it to adjacent vertices. Because each collision between the jello and the rotating cube introduces a force onto those springs, the motion of each vertex initially resembles a series of collision (as discussed in Section 4.2.2) and then exhibits oscillatory motion (as discussed in Section 4.2.1.1).

Simulating the behavior of the local-coordinate-vertex model on the jello application, we derive

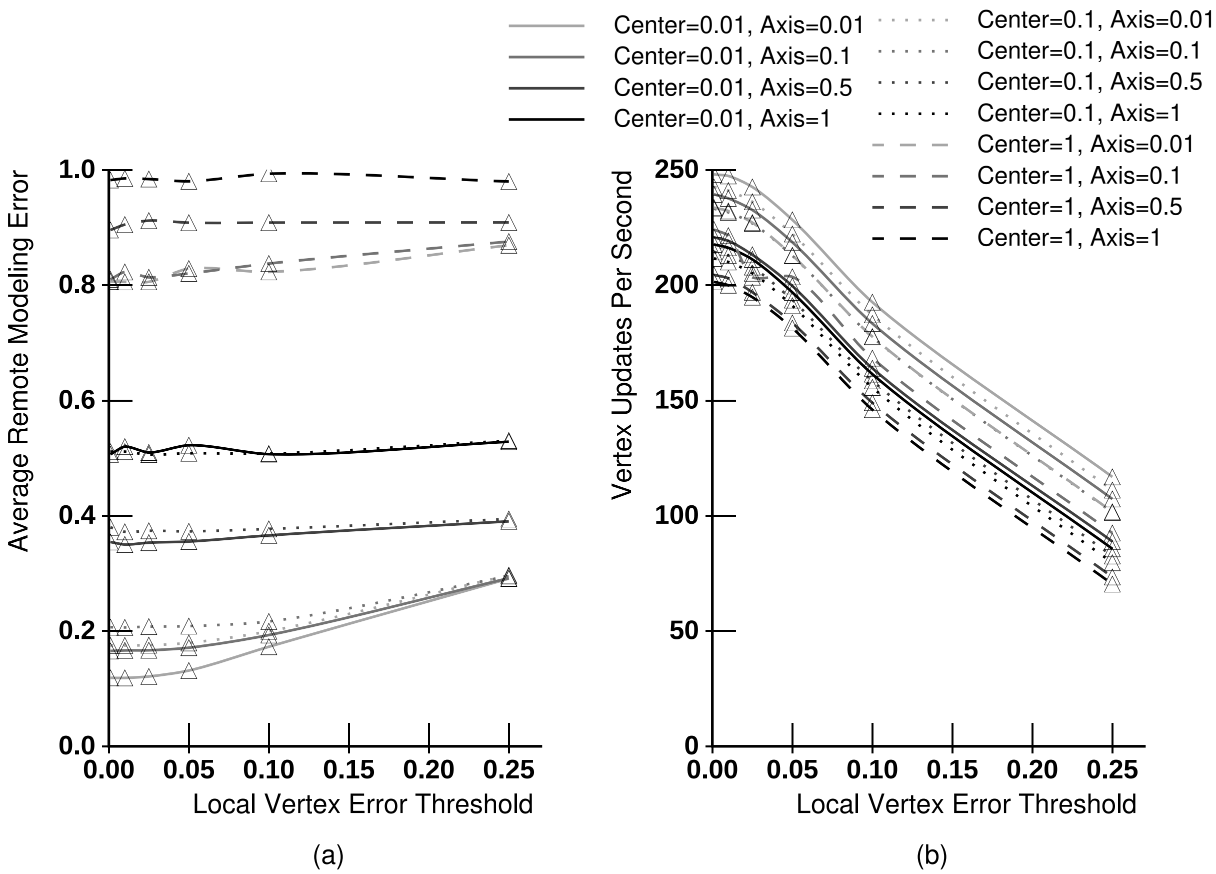

Figure 6.9: Local-Coordinate-Vertex Model for the Jello Application: (a) Modeling Error and (b) Vertex Update Rate

the data in Figure 6.9 which shows the resulting average remote modeling error of the 12 surface vertices in the jello model and the vertex update rate, as a function of the error tolerance for the component vertices in their local coordinate systems. The curves demonstrate behavior with twelve different error tolerance configurations for the center vertex and axis points. As in Figure 6.5, each axis point update is treated as two vertex updates for the purposes of measuring the packet rate. The figure reveals that the local-coordinate information is most valuable in remote models whose position and orientation are already being modeled accurately. For example, local vertex information has the greatest effect when the center vertex and axis point error thresholds are small (around 0.01, in the case of the jello). For these parameters, the local coordinate system information introduces a 300%–400% increase in packet update rate to obtain an error reduction of 28%–50% below that of a position-and-orientation model. Furthermore, the packet rate numbers are overstated because each packet only carries on vertex update (except for axis point updates). In reality, the source could bundle multiple vertex updates into a single update packet to reduce the overhead introduced by packet headers. For the jello entity, the local-coordinate-vertex model is more effective than the radial-length model because it models the independent motion of each surface vertex.

Overall, the local-coordinate-vertex model's structural fidelity is determined by the nature of the structural vertex motion in the entity's local coordinate system. Though it can handle independent behavior by each structural vertex, the model still requires the motion to be relatively simple. For example, like the radial-length model, it cannot model high-frequency oscillatory motion without imposing a high update rate.

6.4 Full-Body Channel for Close-Range Viewers

Over a *full-body channel*, the source host transmits the highest level of detail about the entity's dynamic position, orientation and structure. To provide this level of information, these channels impose higher bandwidth requirements than either the rigid-body or approximate-body channels. Furthermore, because the remote host must receive each of the updates and must model the detailed structure of the entity, full-body channels require the greatest amount of computational resources. Consequently, at any given time, only a small fraction of the remote entities visible to a host can receive this level of modeling.

Remote hosts subscribe to the full-body channel for non-rigid entities that are located near the viewer within the virtual world. These entities require the highest positional and structural fidelity because their accurate visualization most directly affects user actions. By allocating network bandwidth and computational resources toward these close-range entities, hosts force more distant entities to use fewer resources by using a lower-detail channel. Full-body models are also most appropriate for modeling entities that exhibit no structural constraints. Such entities include entity groups, like a flock of birds, where each member moves independently of the others.

6.4.1 Selecting Marker Vertices

In providing the full-body channel, the source uses the PHBDR protocol to transmit the position of the entity's *marker vertices* in the global coordinate system. The marker vertices are selected by the source to describe the entity's structure, and they may simply correspond to the vertices in a triangular mesh representation of the entity's geometry. For each marker, the source computes the set of adjacent marker vertices within the entity's structure and distributes this information to remote hosts via the simulation directory service.

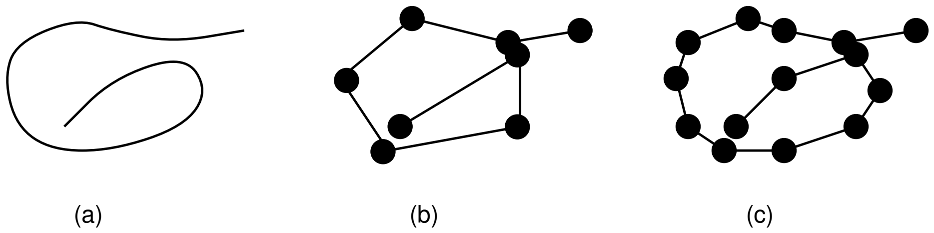

Figure 6.10: Number of Marker Vertices Determines Structural Fidelity Supported by Full-Body Channel: (a) Source model of a string; (b) Remote model with few marker vertices; (c) Remote model with more marker vertices

The receiver uses the PHBDR protocol to independently dead reckon each marker vertex. Upon receiving an update packet for the entity, it applies the marker position information to the appropriate PHBDR proxy(ies). To display the entity, the host determines the current dead reckoned position for the marker vertices and then uses the marker adjacency information to construct and render the entity's current geometry.

As shown in Figure 6.10, the number of marker vertices associated with the entity determines the structural fidelity supported by the full-body channel. By selecting more marker vertices, the source provides more information about the entity's structure, thereby improving the achievable structural fidelity. On the other hand, each marker vertex imposes a bandwidth and computation cost, and beyond a certain point, the additional information offers diminishing returns for conveying structural information.

To dynamically add or remove a marker vertex during the simulation, the source host reliably multicasts an announcement over the entity's full-body channel. This announcement specifies which marker point is being added or deleted, as well as any changes in marker adjacency. The source also updates the simulation directory service so that new subscribers can obtain the updated entity structure. By changing the set of marker vertices, the source can adopt to lasting changes to the entity's underlying structure. For example, to support remote modeling of a growing plant, the source must periodically add additional marker vertices to account for the increasing distance between the existing markers.

The source may also provide "ephemeral" marker vertices in its update packets. When transmitting an ephemeral marker vertex, the source provides three position values with corresponding timestamps; the remote host uses these three updates to dead reckon the ephemeral marker vertex, but that proxy is only kept until the next update packet arrives for the entity. (Each ephemeral marker update is effectively a type of state-replace packet, as discussed in Section 4.4.) Ephemeral markers are appropriate for entities undergoing transient structural changes that do not justify the

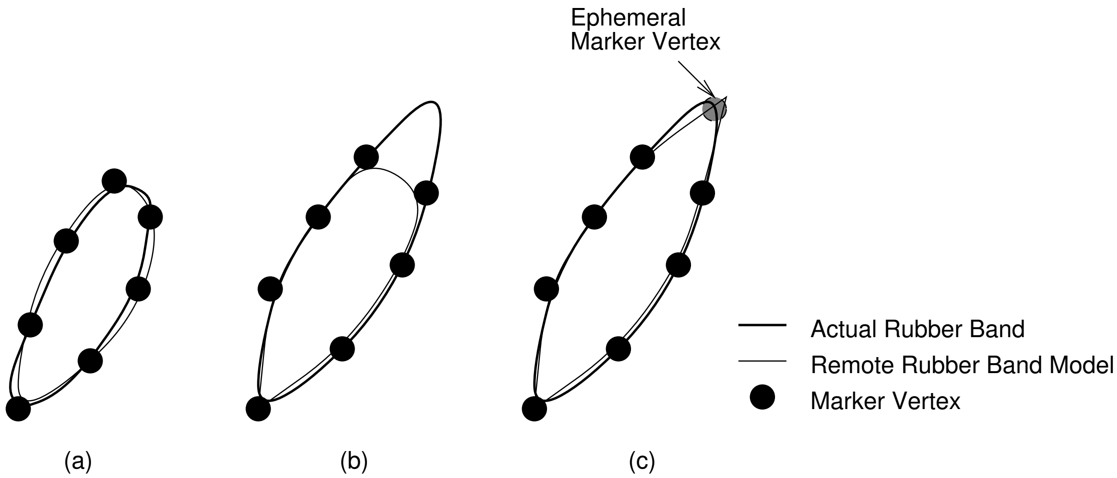

Figure 6.11: Remote Modeling of a Rubber Band: (a) Before being stretched; (b) After being stretched; (c) With an added ephemeral marker vertex

overhead of creating a new permanent marker vertex. For example, when a rubber band stretches, it creates an area containing no marker vertices, as shown in Figure 6.11. Changing the marker set each time the rubber band is stretched or released is generally impractical because the structure potentially changes too rapidly. However, if the ephemeral marker persists for some period of time (e.g. if the rubber band is stretched around an object), then the source may choose to add it to the permanent marker set for the entity, thereby eliminating the need to transmit multiple updates for the marker inside each update packet.

To provide a full-body channel for the jello entity, the source host designates the twelve vertices forming the icosahedron as the marker vertices. It uses the PHBDR protocol to update these vertices in the global coordinate system.[2] Modeling the jello based on information provided by the full-body channel, we derive the data in Figure 6.12 which shows the resulting average remote modeling error for the 12 surface vertices in the jello model and the vertex update rate, as a function of the PHBDR error tolerance on each marker. Because the motion of each vertex is dominated by the jello's rotation rather than its translation, the error and packet rate numbers closely resemble those observed for circular motion, for example, as shown in Figure 4.2.

[2]To facilitate re-use of much of the jello's original rendering code, we also transmit the the position of the jello's center point, but this information could be eliminated by re-writing the rendering code.

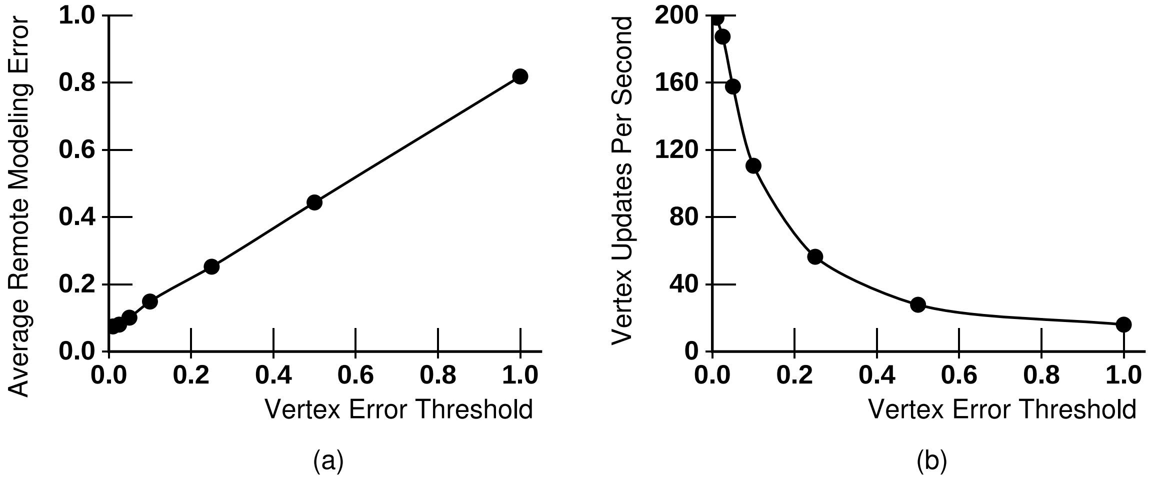

Figure 6.12: Modeling the Jello Using Data From a Full-Body Channel: (a) Modeling Error and (b) Vertex Update Rate

6.5 Relationship Between the Different Channels

Figure 6.13 illustrates the relationship between the three channels for the jello application. The figure shows where each channel falls within the space of the fidelity–bandwidth tradeoff. The curves reveal that the models exhibit order-of-magnitude differences in bandwidth and structural fidelity, thereby satisfying a broad range of possible requirements at remote hosts.

Although we have only demonstrated these order-of-magnitude distinctions for the jello application, we fully expect similar results for other complex entities. For example, we expect that a full-body channel will describe on the order of ten marker vertices for a complex entity, while a rigid-body channel only transmits information about one or three vertices. Moreover, the full-body channel employs tighter PHBDR error thresholds than the rigid-body channel. These factors together assure order-of-magnitude differences in update rate. Similarly, as we have seen, ignoring orientation and structure information significantly reduces the modeling fidelity for entities that rotate freely or undergo dynamic structural change.

We have not addressed how the source host selects appropriate protocol as the PHBDR protocol parameters are determined by the entity's Typically, the source host determines these error thresholds statically based on the expected modeling fidelity requirements of remote hosts. When those requirements cannot be determined in advance, the source host may apply two approaches: provide multiple channels within each category or dynamic rethresholding.

If sufficient computational and network resources are available, a source may provide multiple

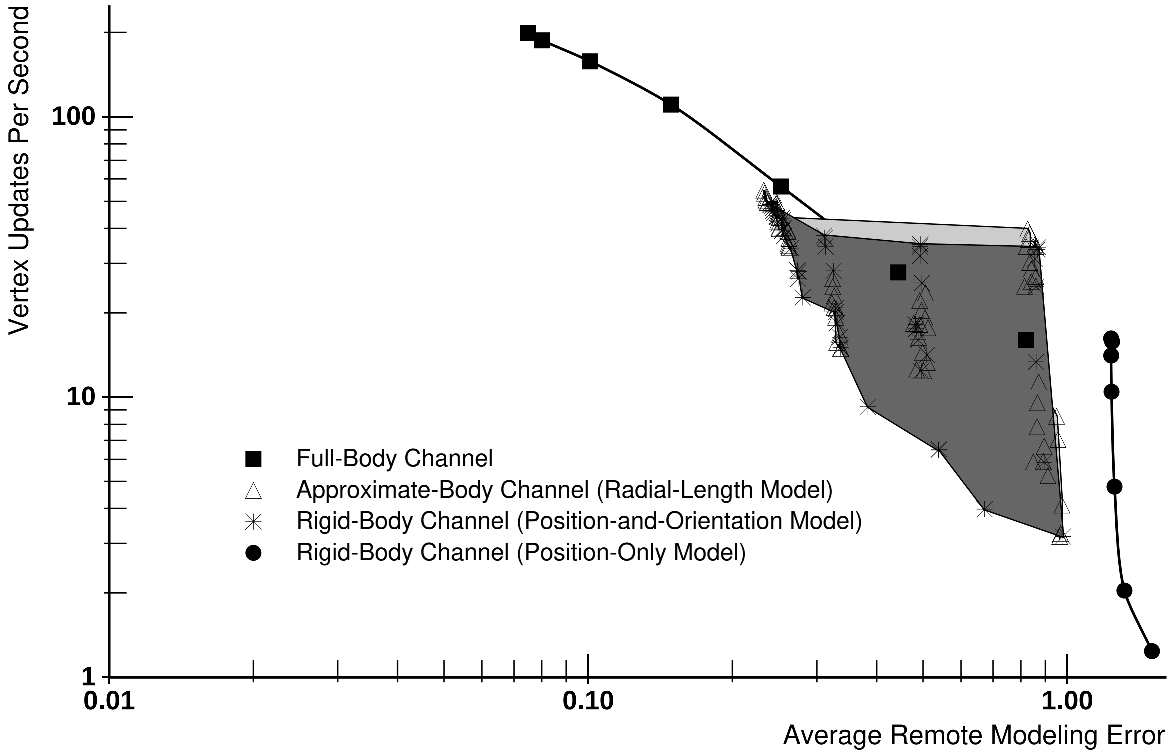

Figure 6.13: Comparison of Vertex Error and Packet Rate Ranges in Jello Application for Multiple-Detail Channels

channels for each category, with each channel offering a different protocol error threshold. This multiple approach alleviates the pressure of selecting an appropriate error threshold for each vertex, and it provides greater flexibility to remote hosts desiring to customize their entity models.

Alternatively, source hosts can support *dynamic rethresholding* based upon feedback provided by active subscribers to each channel. Each remote host subscribes to the channel providing the closest match to the locally desired bandwidth and structural fidelity. The remote host then periodically (e.g. once per minute) unicasts its ideal bandwidth utilization and modeling error to the source. Based on the set of recently-received requests, the source then chooses an appropriate error threshold, either by satisfying the median or average of the requests, by satisfying the highest fidelity requirement, or by satisfying the lowest bandwidth requirement. In environments that include dynamic rethresholding, source hosts must ensure that the channel information provided by the simulation directory service is kept up-to-date.

6.6 Conclusion

Multiple-detail channels demonstrate how the basic PHBDR and Axis Point protocols can be used to support the low-fidelity, medium-fidelity, and high-fidelity requirements simultaneously present within a simulation, as well as supporting a broad range of entity types from stationary and rigid to moving and non-rigid. Multiple channels improves scalability by decoupling the hosts in a distributed simulation, allowing each host to independently determine the level-of-detail at which to model remote entities. Each host is therefore free to allocate its own computational and bandwidth resources among entities to best meet the visualization needs of the local user. The dynamic rethresholding feature would further permits remote hosts to influence the remote modeling fidelity supported by the source.

In the worst case, multiple-detail channels introduce minimal additional overhead above the traditional single-detail approach, and in the common case, they reduces the aggregate traffic and computation throughout the simulation. Multiple-detail channels require more computation at the source host to maintain the additional representations and transmit updates. The network links near the source also see the additional traffic from the lower-detail channels. Finally, remote LANs may see extra traffic if local hosts subscribe to different channels for the same entity. However, if the source only provides one channel, it must provide high-detail information to support near-range viewers. Therefore, the additional resources to support lower-detail channels are proportionally minimal. Furthermore, because we expect that most remote hosts do not require high-detail channels for a given entity, they subscribe to channels containing less traffic. As a result, most network links (particularly the bottleneck tail circuits) no longer fall on the high-detail channel's multicast distribution tree.

One might argue that in outlining a multiple-detail channel architecture, we are permitting the development of specialized dead reckoning protocols, the very thing we were trying to avoid. However, the architecture tightly bounds the range of permitted entity models. For instance, model developers must build on top of the PHBDR module and the model must conform to the rigid-body, approximate-body, or full-body channel structure. Even the approximate-body channel, which offers the most design flexibility, only permits the developer to make controlled changes to a rigid entity structure. The local-coordinate-vertex model therefore be regarded as the most extreme implementation of this channel type.

Up to this point, each channel has supported the remote modeling of a single entity. Although a low-detail channel allows hosts to reduce the update rate and detail for a distant entity, it is often desirable to reduce the bandwidth and computation for a *group* of entities, particularly when their positions are either unimportant or imperceptible to the local viewer. The next chapter discusses how multiple-detail channels and PHBDR are further extended to support the remote modeling of entity groups.

Chapter 7

Entity Aggregation

When bandwidth and computational resources are limited, hosts must prioritize resource utilization toward those simulation entities that have the greatest importance to the local viewer. Multiple-detail channels allow hosts to selectively model entities with a lower level-of-detail, thereby reducing the local resources allocated toward those entities, while nearby entities continue to receive full-detail remote modeling of position, orientation, and structure. However, even over low-detail channels, each entity transmits update packets independently, so the receiving host must contend with per-entity network bandwidth and packet processing overheads. These costs can be prohibitive in large simulations in which each host may potentially be aware of thousands of entities.

In this chapter, we discuss *aggregation*, a technique that allows a single message to describe multiple entities. We begin by discussing the implementation of Projection Aggregation Entities (PAEs), which dynamically group entities by both their type and their location. After discussing the performance of PAEs when integrated with multiple-detail channels, we describe the implementation of PAE hierarchies. We conclude the chapter by describing potential uses for PAEs for optimizing simulation operations such as scene rendering and collision detection, as well as supporting evolution of simulation systems.

7.1 Projection Aggregation Entities and Their Implementation

A *Projection Aggregation Entity* (PAE) is a simulation entity that combines the organization-based and grid-based aggregation approaches [85] discussed in Chapter 2 to bundle update information from a group of entities. Each PAE includes entities from a single organization located within a single octtree grid of the virtual world. The organization therefore is effectively "projected" onto

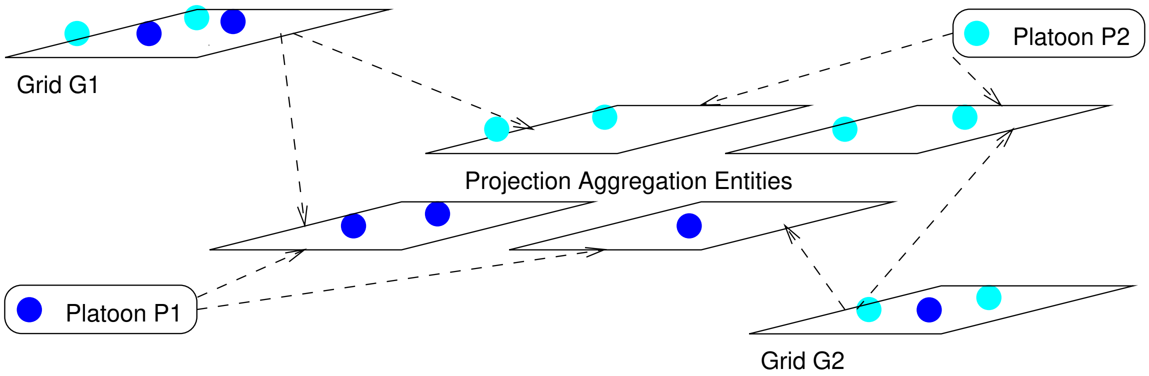

Figure 7.1: Projection Aggregation Entities Represent the Intersection of an Entity Organization and a Virtual World Grid Region

the virtual world grid, as shown in Figure 7.1. The figure represents a virtual world divided into two grids ($G1$ and $G2$) and contains two platoon organizations ($P1$ and $P2$). The entities are divided into four PAEs, namely $(P1, G1)$, $(P1, G2)$, $(P2, G1)$, and $(P2, G2)$. For example, PAE $(P2, G1)$ bundles updates from all members of platoon $P2$ that are currently located inside grid $G1$. A PAE's membership changes dynamically as entities move about the simulation virtual world.

By integrating organizational and location information, PAEs describe entity groups that are meaningful to remote hosts. For example, if the PAE were to group entities only by organization, it would bundle information about entities that may be scattered throughout the virtual world. Similarly, if the PAE grouped entities only based on virtual world grid location, it would bundle information from unrelated entity types. Fine-grain entity groups, as provided by PAEs, give receiving hosts better control over selecting the particular entity types and virtual world locations about which to receive aggregated information.

As shown in Figure 7.2, the PAE subscribes to the multicast groups carrying updates for its current member entities. The PAE simply collects those individual entity updates and bundles them into a single update packet. This update packet is transmitted along a multicast address assigned to the PAE. The PAE is associated with a *transmission policy* which determines how long to wait before transmitting the bundled data.

The PAE update packet consists of a UDP/IP header followed by a sequence of (entity/vertex identifier, timestamp, position) tuples extracted from the bundled entity update packets. Data bundling reduces the packet rate and associated computational overhead seen by remote hosts who subscribe to the PAE's multicast address. Bundled data also consumes less bandwidth than individual vertex updates because packet headers are eliminated.

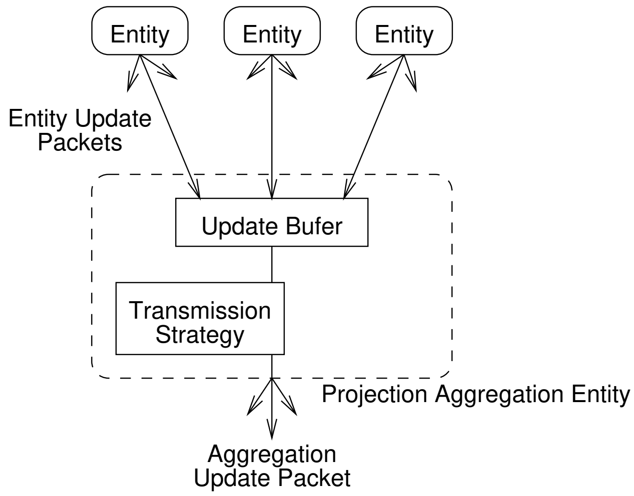

Figure 7.2: Projection Aggregation Entities Collect and Bundle Updates From Member Entities

On the other hand, bundled information is not appropriate for high-fidelity remote modeling because bundled updates are not as timely as the individual updates received directly from the individual entities. The PAE imposes an intermediate delay in order to collect multiple update packets. The length of this delay is determined by the transmission policy used by the PAE.

7.1.1 Transmission Policies for Projection Aggregation Entities

PAEs use a transmission policy to determine how long to wait before transmitting a bundled update packet. In choosing a transmission policy, the PAE trades off the amount of bandwidth reduction provided by the bundled packets against the amount of delay introduced on the bundled data. For transmission policy may minimize latency by transmitting frequent packets, each bundling fewer updates. On the other hand, the transmission policy may introduce more latency in order to transmit fewer packets, each bundling more updates.

We consider two transmission policies, timeout-based transmission and quorum-based transmission, each providing a different bandwidth and latency tradeoff.

7.1.1.1 Timeout-Based Aggregation Transmission

With a timeout-based transmission policy, the PAE collects individual entity updates and transmits them after waiting for some timeout period τ. Assuming that the member entities transmit updates independently of one another, the aggregation entity artificially delays each update on average by an extra $\frac{\tau}{2}$ plus the additional network transmission latency introduced because the update is transmitted twice—once to the aggregation entity and once to the final destination.

The timeout-based technique guarantees a bound on the delay seen by an individual update packet and, therefore, the error seen by remote hosts. On the other hand, the bandwidth and packet rate reduction provided by the timeout-based technique varies depending on the dynamic ratio between the timeout value and the member entity transmission rates. Suppose, for example, that the PAE represents n member vertices (i.e. entities), each generating updates (either by exceeding its respective threshold or by timing out) every t seconds, on average. Without aggregation, the entities transmit a total of $\frac{n}{t}$ packets per second. However, aggregation reduces the packet rate to a maximum of $\frac{1}{\tau}$ packets per second, with each packet bundling an average of $\frac{n\tau}{t}$ updates. Assuming that t and τ share the same order-of-magnitude, aggregation reduces the packet rate by at least a factor of n. Assuming further that each entity position update requires d bits of data and each packet header requires h bits. The total bandwidth requirement without aggregation is $\frac{n(h+d)}{t}$ bits per second. The PAE reduces this bandwidth to $\frac{h+\frac{n}{t}\tau d}{\tau}$ bits per second, a reduction factor of $\left(\frac{th+n\tau d}{n\tau h+n\tau d}\right)$. Assuming that a 224-bit UDP/IP packet header, 32-bit timestamps, 32-bit entity identifiers, and 192-bit position updates (three 64-bit values), the timeout-based transmission policy reduces bandwidth by a factor of $\left(\frac{7}{15}-\frac{7}{15n}\frac{t}{\tau}\right)$ ranging up to 53%.

7.1.1.2 Quorum-Based Aggregation Transmission

Under a quorum-based transmission technique, the PAE transmits an aggregate update packet when some minimum proportion p of the member vertices (the quorum) have provided an update. Intuitively, the value of p determines the overall level of consistency provided for the aggregate: p close to $\frac{1}{n}$ provides high consistency because the aggregation transmits an update when only a small number of member vertices are updated, while p close to one yields weak consistency because the aggregation entity must wait for most member vertices to change before transmitting an update.

Quorum-based aggregation transmissions reduce packet rate and bandwidth when the group contains heterogeneous entities because, unlike timeout-based transmission in which the packet rate is fixed, the quorum-based update rate adapts to reflect the packet rates of the component entities. Because each quorum-based update packet contains a pre-specified number of component updates, remote hosts are guaranteed to see a particular bandwidth and packet rate reduction; the value of p determines the level of network resource reduction. However, to achieve these guaranteed network savings, the quorum-based approach sacrifices the predictable positional fidelity offered by the timeout-based approach because the quorum-based transmission does not bound how long

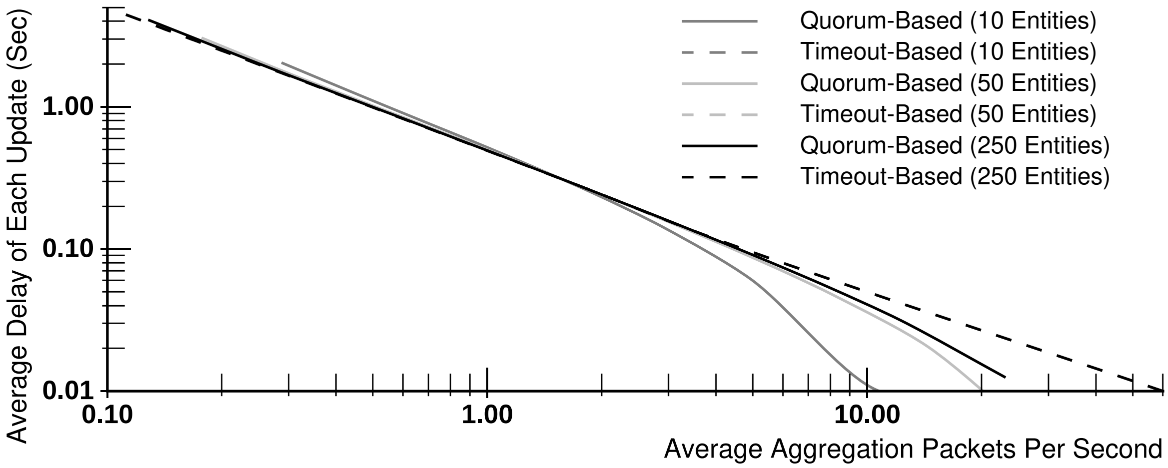

Figure 7.3: Average Packet Rate and Update Delay Produced by Timeout-Based and Quorum-Based Transmission for Aggregations

an individual update may wait before being transmitted in an aggregation packet.

Suppose that the aggregation contains n vertices which individually generate updates (either by exceeding their respective thresholds or by timing out) on average every $t_1 < t_2 < \cdots < t_n$ seconds. Without aggregation, the PHBDR protocol generates $\left(\frac{1}{t_1} + \frac{1}{t_2} + \cdots + \frac{1}{t_n}\right)$ packets per second. Using quorum-based transmission, t_{pn} represents the time in which pn vertices have transmitted updates.[1] Because a packet is transmitted whenever pn points exceed their thresholds, quorum-based transmission generates $\frac{1}{t_{pn}}$ packets per second. If each position update requires d bits of data and each packet header requires h bits, then the bandwidth requirement without aggregation is $(h + d)\left(\frac{1}{t_1} + \frac{1}{t_2} + \cdots + \frac{1}{t_n}\right)$ bits per second. Quorum-based transmission reduces this requirement to $\frac{h+pnd}{t_{pn}}$ bits per second.

7.1.1.3 Comparing the Transmission Approaches

Figure 7.3 compares the packet rates and delays produced by timeout-based transmission and quorum-based transmission for a mix of heterogeneous entities. The simulated aggregation contains a set of entities that one might expect to see within a single virtual world region in a ground-based military simulation [103]: 15% fixed-wing aircraft, 10% rotary-wing aircraft, 15% tanks, 10% trucks, and 50% dismounted infantry. The timeout-based curves were generated by changing

[1] Statisticians refer to t_{pn} as the (pn) order statistic of t_i.

the timeout period τ, while the quorum-based curves reflect the effect of changing the quorum proportion p.

The figure reveals the tradeoffs reflected by the two transmission policies. Timeout-based transmission bounds the delay placed on updates at the expense of a higher update rate. Moreover, the transmission behavior is independent of the aggregation size, which is a desirable property because the membership of an aggregation changes dynamically as entities move about the virtual world. On the other hand, the quorum-based approach transmits up to one-third fewer packets than timeout-based transmission, but its delay characteristics are less predictable. In particular, any individual update may be delayed indefinitely, and the delay characteristics depend on the aggregation size.

Based on these results, therefore, we conclude that timeout-based transmission is generally preferable for PAEs in highly dynamic simulation environments having ample wide-area network bandwidth. Quorum-based aggregations are more appropriate when the entity motion is more localized or when wide-area network bandwidth is a limited resource.

7.1.2 Optimizing PAEs for Scalability

We optimize the PAE implementation to minimize the computational and bandwidth overhead for managing large numbers of PAEs, for changing PAE membership, and for creating and destroying PAEs. These optimizations are needed to support fine-grain entity groups, such as those provided by PAEs, in large-scale simulations for several reasons:

- PAE membership changes frequently. In particular, an entity leaves one PAE and joins another PAE whenever it moves to a new grid region in the virtual world.

- The number of potential PAEs is large, bounded by the product of the number of grids and the number of entity organizations. This bound grows rapidly as simulations involve larger virtual worlds and involve more participants. Even if only non-empty PAEs exist in the simulation, the number of active PAEs can still be quite large.

- PAEs may be created and destroyed rapidly as a result of entity motion about the virtual world. For example, when members of platoon $P1$ first enter a grid area $G3$, a new PAE $(P1, G3)$ must be provided to bundle those entity updates. Similarly, when $G3$ no longer contains entities from $P1$, PAE $(P1, G3)$ should be destroyed.

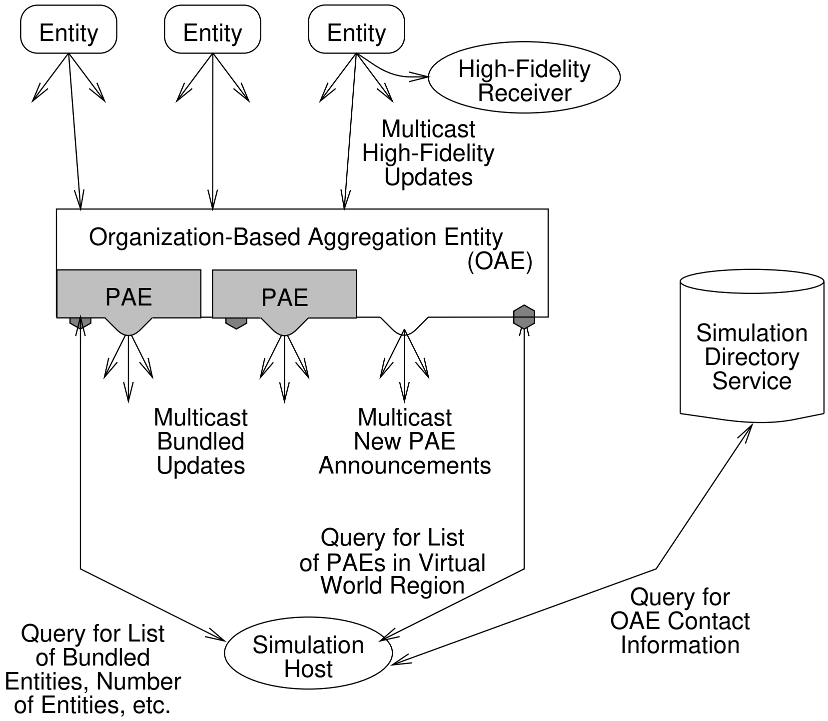

Figure 7.4: PAEs as Logical Entities Created and Managed by OAEs

These scalability barriers arise because PAEs mirror the dynamic nature of the entity motion. As entities move more rapidly, PAEs must also rapidly change to reflect new opportunities for bundling updates.

7.1.2.1 Creating and Locating PAEs

To support dynamic PAEs, we create a set of *Organization-Based Aggregation Entities* (OAEs)—simulation entities representing an entity organization such as a platoon of tanks—as shown in Figure 7.4. Because entity organizations are rarely created or destroyed, OAEs are long-lived entities that can be statically configured at the start of the simulation. Each OAE is associated with a multicast address and a port address which are registered in a simulation directory service.

PAEs are then implemented as *logical* entities created by the OAEs. The OAE subscribes to high-fidelity data from each of its members and organizes the entities into PAEs based on their locations within the virtual world. Each PAE is dynamically assigned a multicast address for transmitting bundled updates and a port address for receiving remote requests. The OAE announces the creation of the new PAE by transmitting a message to its own multicast address describing the

PAE's grid region, multicast address, and port address.

To start receiving bundled updates for a set of entities within a virtual world region, a remote host must first determine which PAE multicast group(s) provides the appropriate bundled data packets. It does this by querying the simulation directory service for the multicast and port addresses of the OAE. It then transmits a query to the OAE's port address asking for a list of PAEs covering the virtual world region of interest. In response to this query, the OAE returns a list of PAEs, their grids, multicast addresses, and port addresses. Finally, the host subscribes to the PAE multicast groups to receive bundled updates, and it subscribes to the OAE's multicast group to learn about new PAEs in the region of interest.

Our use of OAEs to create and manage logical PAEs has several advantages. First, OAEs can treat PAEs as lightweight entities that can be created quickly. The creation of PAEs is distributed and does not require a centralized process to determine the need for a new PAE, assign a host to execute the PAE software, and start the PAE at that host. Instead, each OAE is responsible for selecting, creating, and managing its own PAEs. Second, having OAEs disseminate information about PAEs reduces the latency for providing that information to potential subscribers. In particular, by not requiring information about the new PAE to be entered into the simulation directory service, we eliminate at least one network round-trip time. Third, the multi-level directory provided by the simulation directory and the OAEs provides better load balancing than could be provided by a single large directory service. Our implementation partitions static information (about OAEs) from the more dynamic information (about PAEs), thereby allowing each directory service to be optimized for its particular access patterns.

The OAE query mechanism might also be used to create a PAE only when a remote host actually expresses interest in the corresponding grid region. This "create-on-demand," or delayed binding, approach would have the advantage of eliminating unnecessary PAEs from the simulation and, consequently, reducing overhead at the OAE host. With the create-on-demand approach, however, remote hosts would need to periodically refresh their PAE queries, because a PAE would be destroyed if no matching query is received within some timeout period. The scalability of this approach is an area for future research.

7.1.2.2 Reducing the Number of PAEs Created and Destroyed

Although OAEs implement PAEs as logical entities, creating or destroying PAEs still represents a significant computational overhead throughout the distributed simulation for initiating or terminating a multicast summary data stream, announcing the group, and changing multicast subscription patterns at remote hosts. In particular, we want to minimize situations where a single PAE is repeatedly created and destroyed as entities enter and leave a particular grid region. To accomplish this goal, we employ two techniques: deferred PAE destruction and low-frequency entity sampling.

When a PAE's grid no longer contains entities from that organization, the managing OAE defers the destruction of that PAE (i.e. the release of the multicast address) for a timeout period. If an entity from the organization re-enters the grid before the timeout expires, then the PAE is not destroyed. This deferred destruction should be particularly effective in large-scale simulations such as the STOW 97 training system in which entity motion exhibits considerable locality within the virtual world [96]. In cases where entity motion does not exhibit locality, deferred destruction risks leaving a large number of multicast addresses assigned to empty PAEs that will not become active. To mitigate this problem, OAEs employ low-frequency entity sampling.

OAEs assess the location of member entities and re-organize PAEs at a low frequency (e.g. once every few seconds), rather than continuously. The lower frequency both reduces the number of PAEs created or destroyed as a result of short-term entity motion and significantly reduces the overhead required to manage the PAEs. However, the low-frequency sampling may cause an entity to be associated with the wrong PAE for some period of time. For example, suppose that the OAE provides PAEs for grids $G1$ and $G2$. If an entity moves from grid $G1$ to grid $G2$, it will still be bundled by the $G1$ PAE until the OAE next samples the member entity positions and re-organizes the PAE membership. Subscribers to the $G2$ PAE will therefore experience some delay before learning about the new PAE member. However, we deem short-term inaccuracies in the PAE membership to be acceptable because the duration of these inaccuracies is bounded by the latency introduced by the PAE bundling and because bundled updates are only used for low-fidelity entity modeling at remote hosts.

By grouping entities based on multiple criteria, PAEs give remote hosts fine-grain control in replacing low-latency, per-entity updates with higher-latency bundled updates. In the next section, we discuss how this flexibility enhances the effectiveness of multiple-detail channels discussed in the previous chapter.

7.2 Integrating PAEs With Multiple-Detail Channels

PAEs and the multiple-detail channel architecture described in Chapter 6 both aim to reduce bandwidth and computational load by providing lower-bandwidth alternatives to full-detail information. We now consider how effectively the two techniques integrate to reduce tail circuit bandwidth and

Channel Name	**Updates Per Sec.**	**Packet Contents**	**Bandwidth (bps)**	**Subscription Percentage**
Full-Body	8	UDP/IP packet header (entity ID, time, vertex position) [12 independent vertices]	46080	5
Approximate-Body (Radial-Length)	5	UDP/IP packet header (entity ID, time, center position) (entity ID, time, axis point positions) (entity ID, time, radius length)	7520	15
Rigid-Body (Position-Only)	1	UDP/IP packet header (entity ID, time, center position)	480	30
Projection Aggregation Entities (PAEs)	0.2	UDP/IP packet header (entity ID, time, center position) [25 tuples per packet]	265[a]	50
Total WAN	112.2		54345	100

[a] We assume that each PAE packet bundles five updates from each of five entities.

Table 7.1: Channel Options for Subscribing to Jello Entity Updates

to reduce packet rate and computation at remote hosts.

We consider a simulation consisting of complex entities such as the jello model described in the previous chapter. A subscriber may therefore subscribe to one of four channels, as shown in Table 7.1. The subscription percentages estimate the proportion of "visible" entities at each host likely to be serviced by that channel. For example, we expect that 50% of the entities visible to each host can be replaced by bundled PAE updates, while only 5% will require full-body modeling. As a conservative estimate, we assume that each PAE bundles only five vertex updates.

7.2.1 Tail Circuit Bandwidth Reduction

In current network environments, affordable off-the-shelf LAN technology is rapidly approaching gigabit-per-second capacity, while investment in high-bandwidth WANs is easily justified because those costs can be amortized over a large number of users. Consequently, we expect that tail circuit links will remain the bandwidth bottleneck over the next several years. With this in mind, we consider how PAEs, when combined with the multiple-detail channel architecture, affect tail circuit bandwidth requirements.

For the purposes of these experiments, we assume that all traffic must flow across the WAN backbone (i.e. each data channel has at least one subscriber somewhere in the simulation). Introducing lower-detail channels (including PAEs) increases the WAN bandwidth requirements by 11% over a system providing only a full-body channel.

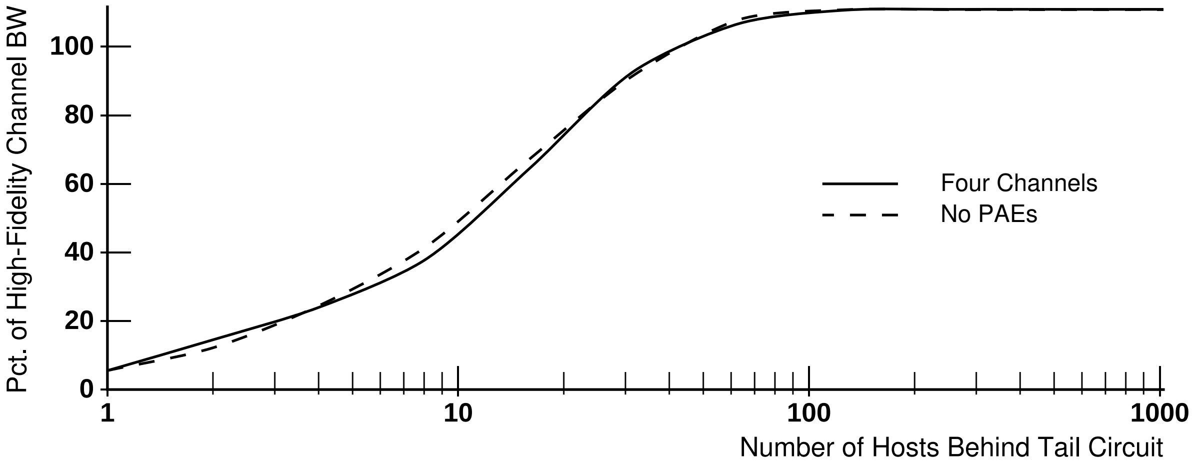

Figure 7.5: Tail Circuit Bandwidth Requirements of Multiple-Detail Architecture Relative to Single-Channel Architecture as a Function of Host Count Behind Tail Circuit

Figure 7.5 shows the tail circuit bandwidth requirements as a function of the number of simulation hosts located behind that tail circuit and interested in a given entity. These numbers represent worst-case results, because we assume that each host behind the tail circuit behaves independently; in reality, in most simulations, users on the same LAN tend to exhibit considerable locality in the virtual world too, so their entity subscription patterns exhibit some correlation.

This graph compares the bandwidth requirements from providing four channels (three of the multiple-detail channels plus a PAE) against the requirements from providing only a high-detail channel. When the number of interested hosts behind the tail circuit exceeds 125, the four-channel bandwidth reaches its maximum at 125% of the high-detail channel bandwidth. At this point, data from all of the channels is reaching the tail circuit. We expect to see a 50% reduction in tail circuit bandwidth when approximately fifteen hosts behind the tail circuit are independently subscribing to the entity. The multiple-detail channels clearly provide a desirable bandwidth reduction through the tail circuits.

The graph also depicts a curve "without PAEs" that considers the three multiple-detail channels alone, where all PAE subscriptions are replaced by subscriptions to the rigid-body channel. In comparing the "four channel" and "without PAEs" curves, we observe that adding PAEs does not significantly affect the tail circuit bandwidth when compared to the basic multiple-detail architecture without PAEs. This observation reveals that the tail circuit bandwidth is heavily dominated by the high-detail channel data, so incremental reductions from lower-bandwidth alternatives do not have a significant effect.

7.2.2 Host Packet Rate and Computation Reduction

Based on the numbers in Table 7.1, a host expressing interest in jello entities would see, on average, 7.37 packets per second for each entity of interest if PAEs bundles are available. On the other hand, without PAE bundles, the same host would see an average of 7.55 packets per second for each entity of interest, so PAEs reduce the packet rate by roughly 2.4%. However, the packet rate difference is more significant for hosts that do not require high-fidelity remote entity modeling. For example, if a wide-area viewer only employs rigid-body entity models, the packet rate without PAEs is approximately 1 per second, while the packet rate with PAEs is approximately 0.52 per second, representing a reduction of 48%.

We conclude that PAEs do not noticeably increasee the bandwidth requirements over bottleneck tail circuits. However, for hosts that can afford to further reduce the remote modeling fidelity for particular entities, PAE updates can significantly reduce the packet rate and consequent computation. Because this packet rate benefit imposes little cost to the simulation system, we observe that PAEs are a valuable tool for supporting scalability.

We have established the effectiveness of PAEs for reducing the packet rate seen by receiving hosts, and we have discussed optimizations to reduce the number of PAEs that are unnecessarily created and destroyed. However, the number of PAEs in a large simulation can still be substantial. In the worst case, a PAE would be created for each entity. In addition, remote hosts must subscribe to all PAEs of local interest. If the PAEs contain too few entities or cover too small a grid region, then the overhead for managing PAE subscriptions overwhelms the performance benefits of PAEs. To address these problems, we arrange PAEs into a hierarchy.

7.3 The PAE Hierarchy

PAEs are arranged in a hierarchy, much like the natural arrangement for entity organizations and an octtree of virtual world grid regions. To support this hierarchy, we provide OAEs representing organizations at each level of the organizational hierarchy. When a PAE is created by an OAE, it first registers itself with a parent PAE, and it unregisters itself before being deleted. The parent aggregation represents a broader organization and a grid of equal or larger size, as illustrated in Figure 7.6. For example, if a platoon is part of a company, then each of the platoon's PAEs associates itself as a descendant of the company PAE covering the same virtual world region. Consequently, each OAE provides PAEs for any virtual world regions containing any direct or indirect descendant entity. For example, if a tank is a member of a platoon organization, then the platoon's OAE

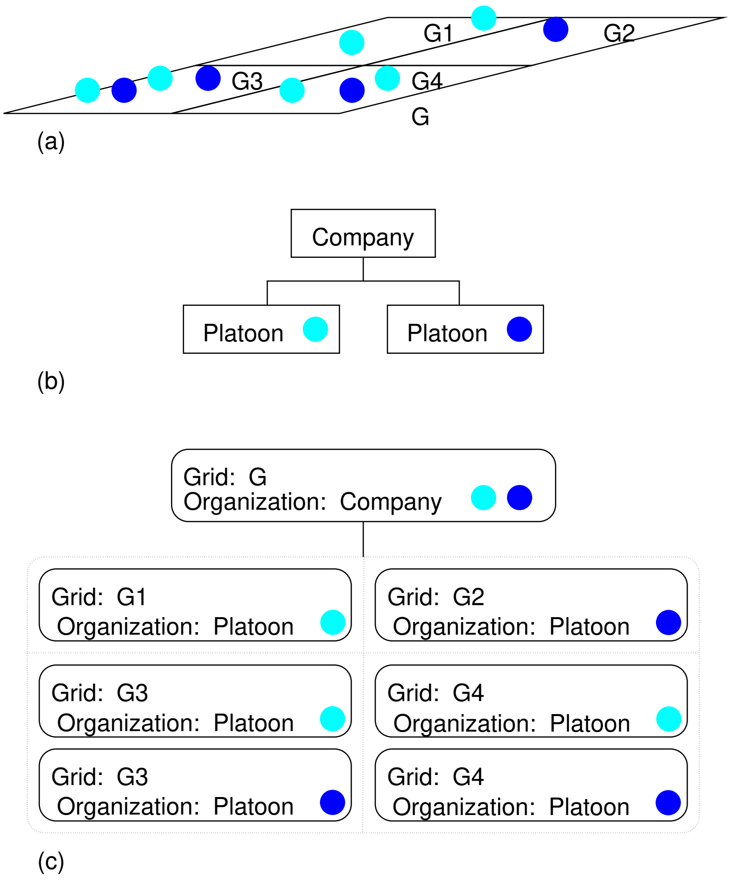

Figure 7.6: A PAE Hierarchy Describes Entities with Increasing Organization and Location Granularity: (a) Location of entities in the virtual world, (b) Organizational structure of entities, and (c) Corresponding PAE hierarchy.

provides a PAE whose associated grid covers the tank's location. Similarly, if that platoon is part of a company organization, then that company's OAE also provides a PAE whose associated grid covers the tank's location.

Figure 7.7 illustrates how the PAE hierarchy is integrated with the OAE hierarchy and a similar *Grid-Based Aggregation Entity* (GAE) hierarchy in a simulation. As we have seen, each OAE can be queried for the PAEs representing the locations of all descendant entities, and each PAE can be queried for the current set of member entities. With a PAE hierarchy, the list of a PAE's member entities may include both regular entities and sub-PAEs representing aggregations for sub-organizations in sub-grids. For example, the members of a tank company's PAE would include the PAEs for the platoons with members in the same grid as well as any tanks in the company that are not assigned to a platoon.

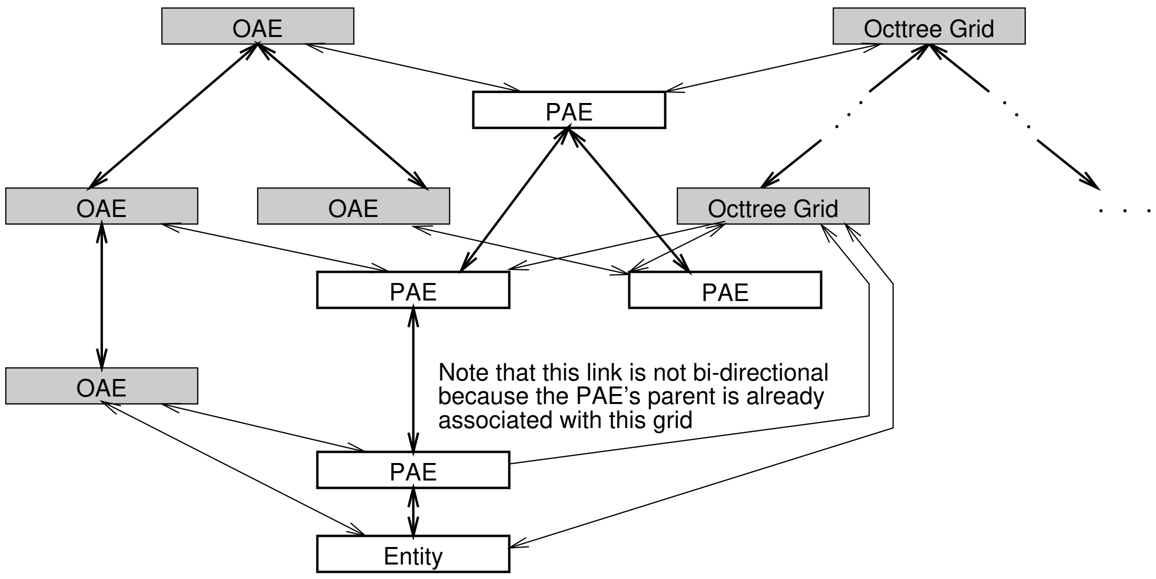

Figure 7.7: PAE Hierarchy is Integrated With OAE and GAE Hierarchies

The behavior of a PAE whose members include other PAEs is almost identical to that of a PAE whose members only include regular entities. A parent PAE simply subscribes to the bundled updates from its sub-PAEs, as well as the updates directly provided by its regular member entities. The parent PAE bundles all of the updates into a larger update packet. We observe that PAEs associated with high-level organizations provide the greatest level of bandwidth reduction because they potentially bundle the largest number of entities. At the same time, these PAEs also introduce the highest latency into their updates, partially because they incorporate the delay already introduced by the sub-PAEs.

Hierarchical PAEs maximize the ability of remote hosts to customize their incoming data flows. For example, the PAE hierarchy allows a host to easily query for and subscribe to bundled information about all tanks in the simulation or bundled information about all entities within a large region of the virtual world.

To improve the scalability of the PAE hierarchy, we introduce optimizations that restrict the grid size associated with the PAEs for each OAE and that reduce the number of changes to the PAE hierarchy. The next two sections discuss each of these optimizations in turn.

7.3.1 Restricting the Grid Size Associated With PAEs

Each OAE dynamically determines the appropriate grid size(s) associated with its PAEs based on the locations of its member entities. During the simulation, as an OAE's entities spread apart, the associated PAE grid size should increase, and as those entities move closer together, the PAE grid size should decrease. Dynamic PAE grid sizes are appropriate because they ensure that those streams provide meaningful data reduction. For example, if all PAEs in the simulation used the same fixed grid size, the simulation would either create many small PAEs (with high overhead to maintain) or just a few large ones (which do not offer the desired fine-grain filtering at remote hosts).

However, if OAEs independently determine their PAE grid sizes, then the PAE hierarchy can become inefficient. For example, if a company's PAEs use a smaller grid size than that used by a descendant platoon's PAEs, then a platoon PAE $p_{platoon}$ may need to have multiple parent PAEs. Therefore, each parent PAE would either transmit bundled updates including entities that do not fall within the associated grid or filter the bundled updates to discard information about entities outside the region.

Consequently, we require that PAEs must be associated with grids that are at least as large as the grids associated with the member PAEs. To implement this restriction, each PAE tracks the grid size associated with each of its member PAEs. The PAE registration packet includes the associated grid size. If this grid size exceeds the parent PAE's grid size, then the parent PAE signals to the OAE to perform a *PAE remapping* to increase the PAE's grid size, possibly merging it with other PAEs. When a descendant PAE unregisters itself, the parent PAE determines whether it can safely reduce its associated grid size, and if it can, it similarly signals the OAE for a PAE remapping which would potentially divide the PAE into smaller units.

A PAE remapping can be relatively expensive because it involves the creation and destruction of PAEs. However, it is an infrequent occurrence. In typical simulations, an organization's member entities tend to exhibit a relatively constant level of dispersion. To reduce the number of remappings further, we defer the PAE remapping operation.

7.3.2 Reducing Changes to the PAE Hierarchy

OAEs defer changes to their associated PAE grid sizes in case the change should be reversed shortly thereafter. This delay resembles the deferred destruction of individual PAEs, discussed in Section 7.1.2. Deferring a PAE remapping helps to reduce the number of PAEs created and quickly destroyed because of short-term entity motion. For example, before signalling to an OAE to change the PAE grid size, the PAE establishes a timeout. When the timeout arrives, the PAE signals the OAE to perform the PAE remapping. However, if conditions change before the timeout arrives so that the PAE's grid size change is no longer necessary, then the timeout is canceled.

Our use of timeouts attempts to localize changes to the lower levels of the PAE hierarchy. For example, when a PAE has no members, it defers its destruction. If that PAE turns out to be the last member of its parent PAE, then the parent PAE similarly defers its own destruction, and this deferred destruction proceeds recursively up the PAE hierarchy. The higher-level PAEs effectively employ longer timeouts than their descendant PAEs. A similar delay pattern arises when OAEs perform PAE remappings to change the associated grid sizes. Despite the potential brief inconsistencies in the PAE hierarchy, this delay has the desirable effect of isolating larger PAEs as much as possible from oscillatory behavior near the leaves of the hierarchy. Consequently, larger PAEs (associated with large organizations and grid areas and providing the lowest fidelity information) are more created and deleted less frequently. This design means that remote hosts more actively manage subscriptions for smaller PAEs, corresponding to entities of greatest local interest, and spend less time managing subscriptions to the larger PAEs of secondary local interest.

We evaluate the effectiveness of this deferred PAE remapping by using a 4096x4096x4096 virtual world with 10 top-level organizations, each containing five sub-organizations with 20 member entities (for a total of 1000 entities). Our simulations run for 5,000 time steps, divided into ten "epochs" of 500 steps. At the beginning of an epoch, each entity is associated with a unit direction vector that it follows during the epoch's time steps. During each time step, we apply a random scaling factor (ranging between 0 and 3) to the direction vector. In this simulation, each OAE associates the same grid size to each of its member PAEs and selects the grid size to limit the number of active PAEs to eight. We consider two extremes of entity behavior. In the first scenario, the "cohesive case," all entities in an sub-organization share the same direction vector (but independent vector scaling factors) during each epoch. Hence, members of each sub-organization tend to remain relatively clustered in the virtual world. In the second scenario, the "fragmented case," each entity receives its own direction vector. Hence, members of each sub-organization exhibit no clustering behavior as they move about the virtual world. Most real scenarios would fall somewhere between these two extremes.

Figure 7.8 shows how delaying PAE remappings by OAEs reduces how many remappings are actually performed. The data demonstrates that even a small quarter-second remapping delay can eliminate as much as 60% of the PAE remappings. This effect is more pronounced with "cohesive" entity motion because each organization's entities exhibit more constant dispersion during the

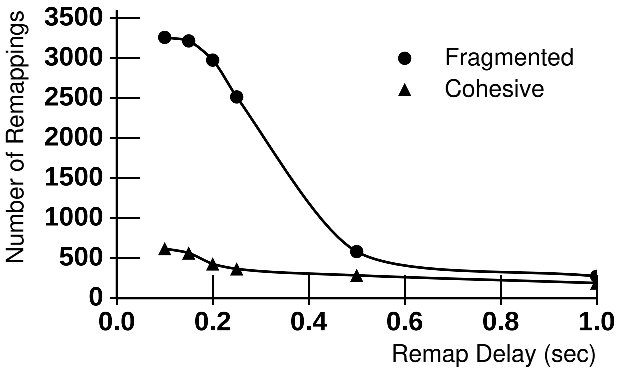

Figure 7.8: Number of PAE Remappings Performed by OAEs as a Function of Remapping Delay

simulation, so any changes are usually short-lived events. In the "fragmented" case, however, each organization's entities exhibit more variable dispersion, so a higher percentage of remappings are actually executed.

7.4 Potential Integration of PAEs With Other Simulation Tasks

Besides providing fine-grain entity groups that allow remote hosts to reduce their bandwidth and computational requirements, the PAEs can potentially be integrated with other simulation tasks, such as scene rendering and collision detection. Each of these operations can benefit from the ability to process multiple entities as a single unit. Group processing is used to merge information about the member entities, to prioritize or filter the handling of individual entities, or to summarize entity information that is not otherwise available.

In this section, we consider three potential uses for the PAE structure:

- Entity Rendering: Using PAE information to render an entity group directly rather than modeling the individual entities.

- Collision Detection and Scene Rendering: Using information from the PAE hierarchy to prioritize entities for processing and filtering entities that should not be processed at all

- Simulation Evolution: Using PAEs as proxies for more detailed representations of the simulation entities.

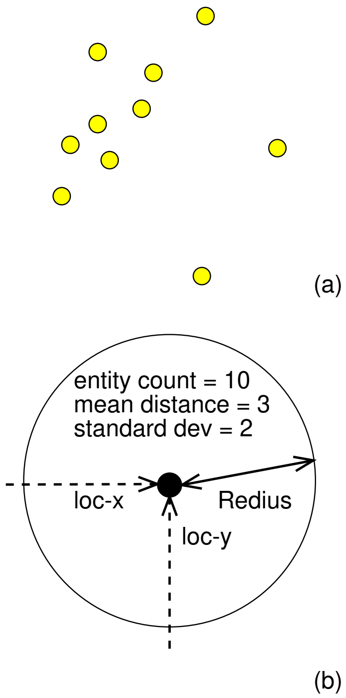

Figure 7.9: PAE Summary Updates: (a) Original entity locations; (b) Summary information transmitted in PAE summary update

7.4.1 PAE Summary Protocol for Rendering Entity Groups

Instead of simply bundling updates from individual entities, PAEs can multicast summary update packets about its members. The summary updates contain enough information for remote hosts to model those entities with extremely low positional fidelity. Remote hosts therefore use the PAE summary updates to represent groups of remote entities that do not locally merit high-fidelity modeling because they are far from the viewer or because the entity type is not of primary interest to that viewer.

7.4.1.1 PAE Summary Update Generation

PAE summary updates describe the location of the PAE's members, as shown in Figure 7.9. The summary packet contains a count of the number of entities represented by the PAE (including descendant PAEs), a single position point summarizing the location of those entities, and information regarding the distribution of those entities around that point.

The summary position point is calculated by averaging the positions of the PAE's member entities; each member PAE is treated as a regular entity except that its summary position is weighted by the number of entities that it represents. Rather than recomputing the summary position point whenever an entity moves, the PAE can incrementally update its summary position by weighting the change in each entity's position based on the overall number of member entities. The summary position must be periodically recomputed to bound the accumulation of round-off error.

The member entity distribution is encoded using the radius of a bounding sphere centered at the PAE's summary position and containing all of the PAE's member entities. The PAE describes the entity density by transmitting the mean and standard deviation of entity distance from the summary position. As with the summary position, member PAEs are treated like regular entities except for a weighting factor, and these density values can be updated incrementally as entities move.

As discussed in the next section, remote hosts use the PHBDR protocol to dead reckon the parameters included in the PAE summary update. The PAE transmits a summary update whenever it detects a discrepancy between the local values of these parameters and the values currently extrapolated by remote hosts. The actual error tolerances depend on the overall fidelity requirements of the simulation, but we set the tolerances to keep the update rate lower than that generated by individual entities and thus minimize the bandwidth overhead introduced by the PAE summaries. Remote sites needing more detailed or frequent information can subscribe to one of the multiple-detail channels provided by the entities themselves.

PAE summary updates containing entity count, summary position, and entity distribution information are sufficient for most simulations. Where additional information is needed, remote hosts can often rely on domain-specific information to generate approximate position models. For example, if a host models a radar, it might use a table of reflection characteristics based on entity type and distance to approximate the radar signature of a PAE's member entities.

7.4.1.2 PAE Summary Update Processing

Figure 7.10 illustrates how remote hosts might render the entity group from the PAE summary information in Figure 7.9b. The summary position determines the location of a local coordinate system used for the PAE group rendering, and the bounding sphere defines the range of locations for members of the entity group. Low-resolution renderings of the entities are then placed within the bounding sphere using the distribution mean and standard deviation values provided in the summary packet. To place each entity, the simulation generates a random direction vector (i.e. from 0^o to 360^o) and a random distance from the summary position, applying a normal distribution with the provided mean and standard deviation.

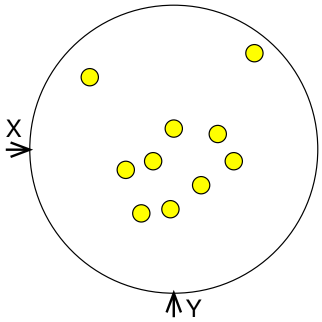

Figure 7.10: Rendering of an Entity Group Based on PAE Summary Information

Because the PAE's summary position, radius, and mean distance from the center point change over time, remote hosts dead reckon these parameters using the PHBDR algorithms. As these parameters are extrapolated, hosts update their models of the position of the PAE members. On each frame, the PAE's local coordinate system is translated to the extrapolated center point. Each entity is then translated along its associated direction vector according to the extrapolated change to the bounding sphere radius and mean distance. Consequently, dead reckoning allows remote hosts to present a dynamic view of the PAE's members despite receiving only occasional summary updates. Dead reckoning also lowers the PAE summary update rate by reducing the error between the true summary parameters and the parameters used at remote hosts. Like any other simulation entity, a PAE is represented as a set of attributes to dead reckon, along with a rendering routine.

The PAE summary protocol reduces the bandwidth and computation required at remote hosts. PAE summary packets, by only transmitting a fixed number of parameters about the entity group, are typically smaller than the bundled updates from the individual entities. Because summary packets are being used to generate low-fidelity positional models, they also can be transmitted less frequently than bundled updates. Moreover, receiving hosts no longer need to employ individual PHBDR protocol modules to extrapolate the position of each entity in the group. Instead, six PHBDR modules (three for the summary position and one each for radius, mean distance, and standard deviation) are shared among all of the group's member entities. For non-trivial groups, therefore, remote hosts need to perform less computation to process each update packet. This computational savings can be quite significant. In a large simulation, a host may be aware of hundreds, if not thousands, of entities, most of which are not of any significant interest to the local viewer. PAE summaries allow the host to maintain some basic information about the entity's general location, so that it can track when the entity later becomes more important to the viewer and merits a model with higher positional and structural fidelity.

Entity Type Collisions	Grid Overlap: None	Grid Overlap: Partial or Full
Not of Interest	No further evaluation	No further evaluation
Potentially of Interest	No further evaluation	Compute potential collisions between PAE members

Table 7.2: Pairwise PAE Analysis to Discard Impossible or Uninteresting Collisions

We have already seen how PAE summary updates might replace a group of entity models that are not of local interest. Because PAEs simultaneously group entities by both organization and location, remote hosts can also apply fine-grain criteria when selecting which entities to model in detail. Remote hosts can also use these fine-grain criteria to filter entities from other expensive simulation operations such as collision detection and scene rendering.

7.4.2 Using the PAE Hierarchy to Filter Entities

Arranging PAEs into a hierarchy enhances the entity filtering capability because top-level PAEs represent larger entity groups and larger grid regions. This top-level filtering is particularly useful in operations that otherwise require processing all entities in the simulation, regardless of their local model fidelities. Using the PAE hierarchy, a host can quickly filter those entities that do not need to be manipulated during the operation and thereby reduce its processing.

We assume that remote hosts cache information about the current PAE hierarchy, particularly the entity organizations and grid locations represented by each active PAE and the membership relationships among PAEs. We outline how hosts can use this cached hierarchy information to optimize two common simulation operations: collision detection and scene rendering, and we then describe a data structure that supports fast traversal of cached PAE hierarchy information.

7.4.2.1 Collision Detection

A collision detection algorithm can use PAEs to quickly filter both unlikely and uninteresting collisions involving local entities, as shown by Table 7.2. We assume that each host maintains a prioritized list of which types of collisions are important to detect accurately based on local user requirements. The PAE collision detection filtering discards entity pairs that either are not co-located within the same grid region in the virtual world or do not lie within the interest list for the local user. After filtering impossible or irrelevant collisions, the collision detection algorithm then attempts to order the processing of the remaining potential collisions based on the prioritized interest list for the local user.

```
Algorithm collDetect (PAEset1: PAEs including local entities
                      PAEset2: PAE list);
  for all p1 ∈ PAEset1 do begin
    for all p2 ∈ PAEset2 do begin
      if grid(p1) intersects grid(p2) and
        (p1, p2) type collisions may have local interest
        then begin
          if loadFunction(p1,p2)
            then collDetect(children(p1),children(p2))
            else collHandle(p1, p2)
        end;
    end;
  end;
end.
```

Figure 7.11: Collision Detection Algorithm With PAEs

The collision detection algorithm, which is fully specified in 7.11, begins by comparing PAEs that include local entities against each of the top-level PAEs in the hierarchy. For each pair, if the grids do not intersect or collisions between the corresponding entity types are not interesting, then further collision detection is not performed on that pair of PAEs. For example, although automobiles might be located in the same region as caterpillars, the interest comparison might prevent the host from testing collisions between local automobiles and PAEs only containing remote caterpillars. If the collision is possible and has potential interest, then a *load function* evaluates whether more detailed collision detection is desired, based on available computational resources. When computational resources are limited, the load function prioritizes collisions based on the entity types involved and an assessment of the possible effects of each collision. For example, the load function would select collisions among automobiles for more detailed processing than collisions between automobiles and grass. If the entity pair is accepted by the load function, then the PAEs' children (which partition entities by finer-grain type and smaller grid) are considered pairwise for further filtering; on the other hand, if the load function rejects the entity pair, then we simply assume that all entities within the two PAEs are colliding.

Collision detection using the PAE hierarchy turns out to be a natural extension to *time-critical collision detection* [35] that checks for collisions between successively tighter approximations to entities' surfaces. Unlike time-critical collision detection, whose iterations only consider entity location information, the PAE hierarchy successively provides both entity location (through smaller

Algorithm renderScene;
  $viewingPAEs \leftarrow$ list of viewing PAEs
  $PAEset \leftarrow$ list of PAE hierarchy root(s)
  collDetect($viewingPAEs, PAEset$)
end.

Algorithm collHandle ($p1$: viewing PAE,
  $p2$: colliding PAE);
  $iterator \leftarrow$ newIterator($p2$,
    grid and organization rendering
      priorities for $p1$,
    node prioritization function for $p1$)
  while $node \leftarrow$ nextItem($iterator$) **do begin**
    render($node$)
  end;
end.

Figure 7.12: Scene Rendering Algorithm With PAEs

grids) and entity type (through smaller organizations) approximations. Consequently, both impossible and *uninteresting* collisions can be discarded.

7.4.2.2 Rendering

To improve rendering, hosts can again use the PAE hierarchy to prioritize entities based on both their location and their type.

The rendering algorithm, fully specified in 7.12, constructs one or more temporary *viewing PAEs* representing a viewing frustrum entity (which might be treated as a top-level organization), as shown in Figure 7.13. To render the scene, we select a set of PAEs to display by using the collision detection algorithm of the previous section between the viewing PAEs and the simulation PAEs. A PAE is selected for display if it intersects (collides) with a viewing PAE and the entity type is of interest to that viewing PAE. When rendering the view from a car, the viewing PAEs would only collide with PAEs whose grids are also located in front of the car. Furthermore each viewing PAE might express a different entity interest set, depending on how far it is from the viewpoint. For example, viewing PAEs located more than ten feet from the viewer might have no interest in insects. As a further example, this rendering algorithm can generate a map of the tanks led by a commander simply by making the viewing PAEs only interested in the organizations that he commands.

Having filtered invisible or unimportant PAEs, the rendering algorithm then traverses each

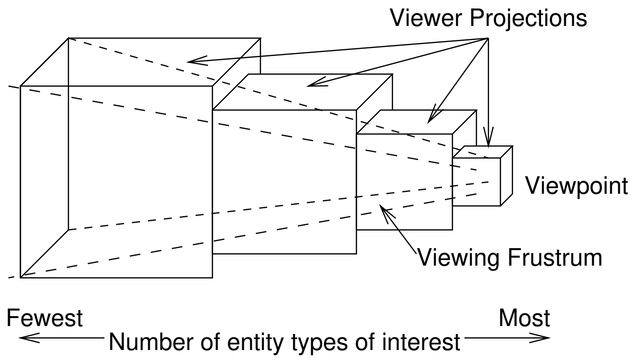

Figure 7.13: Viewing PAEs Bound the Viewing Frustrum and Filter the Types of Entities Displayed in Each Region

selected PAE to an appropriate level of detail. Computational resources can therefore be allocated to the more critical (closer and more interesting) PAEs. A potential implementation might assign a *rendering priority* to each organization and grid. Grids closer to the viewer receive a higher rendering priority, as do organizations that are of greatest interest to the viewer. For each selected PAE, a *node prioritization function* is applied to the rendering priorities for the associated organization and grid. If the prioritization function value exceeds a threshold, then the PAE's members are processed recursively; otherwise, the PAE summary is rendered. Some hysteresis is desirable to ensure that the positional fidelity employed for a particular entity does not change constantly and to ensure smooth transitions between different graphical representations for the entity.[2]

Scene rendering using the PAE hierarchy potentially provides greater flexibility than existing rendering algorithms. First, they extend the semantics of scene rendering to consider entity type as well as entity location. Traditional graphics applications represent the virtual world hierarchically using an octtree or other spatial data structure. The rendering routine then simply traverses the hierarchy and renders each node. Newer toolkits, such as SGI's IRIS Performer [75], only traverse the hierarchy to a level-of-detail based on the entity's distance from the viewer. However, existing rendering techniques do not directly consider entity type information. PAEs, on the other hand, allow early filtering of uninteresting entity types, and the node prioritization function uses both the entity type and entity location information to determine the rendering detail. For example, PAEs allow a renderer to prioritize the cars in a scene and only render caterpillars if sufficient computational resources remain. Second, by providing a low-fidelity model for entity groups,

[2]How to smoothly transition between different graphical models is a standard problem in computer graphics [44, 95, 77, 30, 34] and is therefore beyond the scope of this thesis.

PAEs may be used within a cost–benefit scheme for rendering multi-resolution data in interactive environments [30]. This algorithm selects entities that provide the lowest cost–benefit ratio, where "cost" is defined by the rendering complexity and "benefit" is some heuristic measure of the value of rendering to the user's perception. Once an entity has been selected for rendering (regardless of the chosen resolution), all other resolution models of the entity are removed from consideration. PAEs can be introduced into this algorithm by simply associating a cost corresponding to the number of polygons rendered in the group summary; the benefit measures the value of displaying those entities, reduced by some measure of the diminished resolution. Third, PAE summaries allow the generation of low-resolution graphical representations of entity groups. They contrast with traditional approaches to automated generation of low-resolution models [31] which only provide models on an entity-by-entity basis.

Despite these benefits, however, the effectiveness of using PAE summaries and the PAE hierarchy for scene rendering depends on whether the rendering algorithm adversely affects the user's perception of the scene. The algorithm's success depends on the ability to dynamically determine and adjust the various organization and grid node rendering priorities, as well as on the selection of an appropriate node prioritization function. These issues are all areas for future research.

7.4.2.3 Traversing the PAE Hierarchy Using Deep Iterators

Simulation operations, such as scene rendering, may need to traverse the cached PAE hierarchy information to a depth that depends on computational load and entity type. To implement this variable-depth traversal of the PAE hierarchy, the simulation implementor can use *deep iterators* that are parameterized with the node detail criteria (in our case, the selected PAE, the rendering priorities, and the node prioritization function). In response to a `nextItem()` call, the deep iterator returns the next PAE that should be rendered according to the prioritization criteria. Consequently, the traversal depth is transparent to the drawing routine, which simply needs to render the PAEs returned by the iterator.

Figure 7.14 compares the computational cost of deep iterators against a recursive function traversing the PAE hierarchy. The measurements were taken on an RS/6000 model 370, rated at 70 integer SPEC marks, running AIX 3.2.5 and using the native C ++ compiler on full optimization. Deep iterators introduce some overhead because they must explicitly store intermediate state information during the traversal and cannot rely on the program stack for this task. By allocating a state-history buffer whose length is doubled whenever the traversal depth requires more storage, we reduced the deep iterator overhead to within 5%. Although this overhead is dwarfed by the

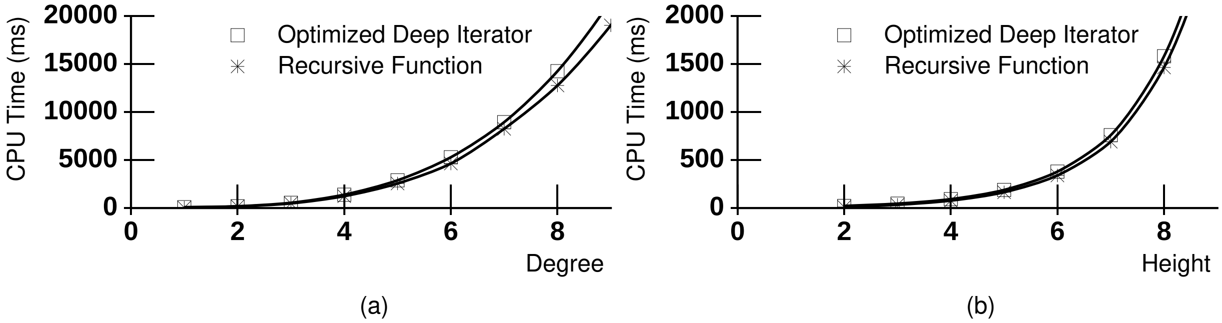

Figure 7.14: Performance of Deep Iterator Versus Recursive Function Over (a) Fixed-height PAE hierarchies (height 5) and (b) Fixed-degree PAE hierarchies (degree 2)

computational cycles for processing each node, we are confident that further tuning of the deep iterator implementation can eliminate most of the remaining overhead.

7.4.3 Supporting Simulation Evolution: Treating All Entities As PAEs

We can generalize the definition of PAEs to include *all* simulation entities, so that remote hosts process all entity updates as PAE summaries that differ only in their dead reckoning and rendering algorithms. Under this generalized definition, each PAE is parameterized with a list of attributes to dead reckon (the default being the summary position, radius, and distribution) and an appropriate rendering routine (the default being the randomized aggregation renderer described in Section 7.4.1). For regular entities, the associated dead reckoning attributes would include position, axis points, and structural vertices, while the associated rendering routine would be the one typically used for that entity.

For example, under this implementation, remote hosts treat an individual tank model as a low-detail replacement for full models of the tank's components, much like a PAE summary update replaces detailed models of each of its members. The tank is therefore a PAE corresponding to the organization "tank parts" and the grid region containing the tank. For example, a tank update (containing position and orientation information) is effectively just summarizing information about its wheels, turret, etc. Detailed models of the individual tank parts may not even be available anywhere in the distributed simulation, but remote hosts need not be aware of this fact.

This ability to unify all simulation entities as PAEs is important for supporting the evolution of distributed simulations. For example, after a simulation is deployed, the developer may choose to introduce models for the tank's components. We assume that the original tank continues to provide summary information about its position and orientation. However, because the tank is treated as a

PAE, the new component models can simply be inserted as its descendants in the PAE hierarchy. Each of these components, in turn, is also a PAe that can be expanded as the simulation becomes more sophisticated. Remote hosts can continue to rely on the tank PAE summaries and do not need to be aware of the new components. However, hosts that are capable of modeling the individual components can easily access that data by continuing to traverse the PAE hierarchy to the new depth.

7.5 Conclusion

In this chapter, we have considered the remote modeling problem for groups of entities. Like multiple-detail channels, which allow remote hosts to selectively change the model's structural and positional fidelity for a particular entity, aggregation allows remote hosts to selectively change the positional fidelity for a group of entities. Projection Aggregation Entities permit reductions in network and computational load in large distributed simulations. A host can dynamically locate and subscribe to a PAE instead of receiving individual updates from each group member. The PAE hierarchy extends the implementation by allowing hosts to fully customize the incoming data flows based on the desired modeling fidelity and tolerable data latency.

We have described a variety of implementation optimizations to support the use of PAEs in dynamic, large-scale simulation environments. We used Organization Aggregation Entities (OAEs) to provide a distributed management and directory service for PAEs and used aggressive deferred timeouts to limit the number of PAEs that must be created or destroyed. Our implementation of PAEs, OAEs, and GAEs is contained in a 4000-line C ++ class library which has been deployed within the PARADISE distributed simulation environment developed at Stanford University.

We have described several potential uses for PAEs to support various simulation operations. Using PAE summary updates, remote hosts can selectively replace a set of entity models with a PAE summary model that provides low-fidelity remote modeling for the group. By using PAEs to filter entities based on both their type and their location, hosts can improve the speed and effectiveness of collision detection and scene rendering, particularly when computational resources are limited. Finally, by providing a single abstraction for all simulation entities, PAEs can explicitly support the evolution of more detailed entity models within a deployed simulation. Ultimately, the effectiveness of these techniques depends on whether the resulting scene is credible to users, and therefore, evaluating these techniques is an area for future research.

Projection Aggregation Entities therefore can represent a foundation for remote modeling in distributed simulations. At the lowest level, a PAE might represent a single entity which, in turn, provides multiple-detail channels of its own. The PHBDR protocol ties everything together by supporting remote modeling of vertices at all levels-of-detail.

Chapter 8

Conclusion

The advent of fast processors and high-capacity networks has made large-scale distributed simulation applications feasible. However, most existing systems have been designed to operate in LAN environments characterized by high bandwidth, low latency and jitter, and a small number of participants running on homogenous hardware; this fact is demonstrated by the dominance of broadcast-based protocols and frame-rate transmission to support information dissemination in these applications. Traditionally, the migration to WAN networks—with the associated lower bandwidth, higher latency and jitter, and larger numbers of participants—has been an afterthought. The STOW program is a classic example of this migration, where ad hoc protocol optimizations are introduced on top of a LAN-based protocol (DIS).

The research presented in this thesis has taken a completely different approach to distributed simulation protocol design. We have attempted to design protocols and algorithms intended specifically for applications running in large-scale WAN environments. Large-scale WAN simulations represent a problem in their own right and are clearly more demanding than smaller LAN-based applications. The large-scale simulation must carefully trade off the timeliness and detail of state information at remote hosts against the network bandwidth, packet rate, and computational load imposed on the network and hosts.

8.1 PHBDR: An Accurate, Efficient Remote Modeling Protocol

This thesis first defined the Position History-Based Dead Reckoning (PHBDR) protocol, which is designed to provide accurate remote modeling of a vertex position. PHBDR employs several techniques to achieves good positional fidelity in the remote model:

- Using adaptive tracking and convergence to dynamically select an appropriate path approximation. This adaptive technique also permits the algorithm to rapidly react to sudden changes of direction in the entity motion.

- Relying only on position information rather than higher derivatives (velocity and acceleration) whose instantaneous values can be more volatile. Relying on position information permits the PHBDR to better model the long-term behavior of the entity and make it relatively independent of short-term oscillations. PHBDR effectively smoothes entity motion by applying a low-pass filter over the position samples.

- Accounting for network latency and jitter by rolling back the dead reckoning model to the update packet's transmission time. The timestamp in the update packet ensures that all remote hosts maintain the same tracking model despite variable network delays. The source host can therefore maintain an accurate representation of how the entity is being modeled (but not how it is being displayed) at remote hosts, and, using this information, it can transmit update packets at appropriate intervals. Moreover, the use of timestamps makes PHBDR more robust because the effects of failures (e.g. network congestion and host failure) are localized.

In demonstrating the positional fidelity of PHBDR, this thesis also represents the first attempt that we know of to systematically analyze the behavior of protocols used in distributed simulation applications. Previous analyses (such as those presented in [51, 101, 79]) have focused on a particular type of entity, typically high-speed military aircraft, and therefore do not assess how well a protocol performs when confronted with new types of entity behavior. Even more recent attempts to test a broader variety of entity behaviors [50] have provided results with limited applicability.

In contrast to these previous approaches, we have employed a combination of mathematical analysis, controlled simulation, and run-time experience to evaluate the PHBDR behavior. We selected and evaluated a small number of representative entity behaviors based on how quickly and smoothly their acceleration changes. We used experiments and analysis to validate the protocol's performance on the selected entity scenarios; because the behavior of any given entity demonstrates characteristics from each of the analyzed behaviors, we can feel reasonably confident in the protocol's overall success over a broader set of entities. Finally, actual deployment experience provided validation of our analytical and scenario-based results and exposed additional issues to consider.

PHBDR also offers reduced computational complexity and lower bandwidth requirements when compared with existing dead reckoning algorithms. These desirable characteristics result from the following design techniques:

- Transmitting only position information. Eliminating velocity and acceleration information represents a significant reduction in the size of update packets. For instance, the PHBDR protocol requires approximately 30% less bandwidth than existing DIS protocols.

- Relying on adaptive algorithms to shift computation away from "uninteresting" entity motion. Processing an update packet for simple entity motion (such as linear motion or a sharp turn) requires half the computation required for curved motion. Even in the worst case, the computational requirements are only 80 addition and multiplication operations. Furthermore, these operations are numerically stable.

- Maintenance of minimal state information per entity. A table maintains 27 state values per entity (or as few as 21, at the expense of some additional computation) which are passed to the PHBDR software module along with the incoming update packet. Consequently, the overall memory requirements are kept small, and the core PHBDR implementation is kept independent of the particular entity type.

Finally, the PHBDR protocol is general-purpose in that it makes minimal assumptions about the vertex behavior (other than that its motion is continuous) or about the simulation environment (other than the availability of synchronized clocks and a unidirectional datagram service). These assumptions indeed have little consequence, because entity motion is almost certainly continuous, fine-grain distributed clock synchronization is a well-understood problem with numerous deployed solutions, and most available networks provide datagram services. Moreover, hosts are relatively decoupled from the network behavior or the behavior of other hosts. These minimal assumptions permit PHBDR to be used in a wide variety of situations and, more importantly, to form the basis for constructing more complex protocols.

8.2 Effective Orientation, Structure, and Aggregation Modeling

Using the PHBDR protocol as a base, this thesis then presented a variety of other protocols: the Axis Point protocol for modeling entity orientation, multiple-detail channels for modeling entity structure, and Projection Aggregation Entities (PAEs) for modeling entity aggregations.

The Axis Point protocol is superior to both Euler angles and quaternions for modeling entity orientation. It offers computational simplicity and numerical stability, continuous parameters as the entity rotates, and easy translation to and from rotation matrices common in graphical systems.

Furthermore, axis point information is readily derived from Euler angles and quaternions, so the technique integrates easily into existing systems that employ other representations.

Multiple-detail channels allow source hosts to simultaneously support a variety of remote models for non-rigid entities. We have demonstrated experimentally in a case study that low-detail structural models can generate packet rates that are two orders of magnitude lower than those produced by high-detail structural models.

PAEs provide an effective way to group entities by simultaneously applying multiple aggregation criteria, particularly entity type and entity location. By supporting these multiple aggregation criteria, PAEs allow hosts to select entity groups based on the criteria that are most important locally, rather than based on some single criterion which might not offer the desired data reduction characteristics. Moreover, our design and implementation demonstrates that the potentially large number of PAEs can be effectively managed in a distributed environment. PAEs also potentially integrate effectively with other simulation operations such as collision detection and scene rendering. Finally, the PAE hierarchy potentially allows simulation developers to introduce more detail into entity models without disturbing existing simulation code. To support the additional detail, hosts simply need to know how to render the new entity components; the data management is otherwise unchanged.

This set of techniques demonstrate our underlying design philosophy to support the decoupling of simulation hosts by giving each host maximum autonomy. This autonomy allows each host to select the set of incoming data streams that most effectively utilize limited local bandwidth and computational resources to provide an accurate visualization for local users. As an example, for uninteresting entity types or entities located far from the viewer, the host subscribes to PAEs that require low bandwidth and minimal computation. For closer entities, the host uses more specific PAEs and rigid-body entity channels which support better entity models. Finally, for entities that are directly visible to the local viewer, the host subscribes to approximate-body and full-body channels that require high bandwidth and considerable computation to provide maximum positional and structural fidelity modeling. If the simulation does not provide this receiver-side flexibility, it cannot be scalable; to support close-range viewers, source hosts must always provide the highest resolution information, so if a host cannot selectively reduce the bandwidth/computation, it will receive an overwhelming amount of data from all of the visible entities.

Together, therefore, these techniques enable the scalability of distributed simulation. The adage "more [fine-grain data channels transmitted by the source] is less [data received by the remote hosts]" should become the guiding design principle for large-scale distributed simulations.

However, to provide this variety of information for all entities, source hosts must implement the entire family of dead reckoning protocols described in this thesis. This potential complexity is alleviated considerably by our recursive design of the underlying protocols.

8.3 The Value of Recursive Protocol Design

To meet the broad range of modeling requirements without introducing a large code complexity, this thesis has adopted a recursive protocol design approach. The simple PHBDR protocol provides a base from which other protocols are constructed. The Axis Point protocol, for instance, internally employs two instances of PHBDR. Each of the multiple-detail channels relies on the PHBDR protocol and the Axis Point protocol to model the motion of each entity vertex. If PAE summaries are used to render entity groups, then the PHBDR protocol is used to model the PAE summary position, bounding sphere radius, and entity distribution information.

The recursive protocol design eased the protocol analyses. Once a basic protocol has been validated, we used those results as a foundation for validating the more complex protocols. For example, having demonstrated the effectiveness of PHBDR on circular entity motion, we could confidently construct the Axis Point protocol, because vertices on the axis vectors trace circular paths along the unit sphere. Moreover, with the PHBDR and Axis Point protocols analyzed, their use in multiple-model channels to represent several vertices at once is a natural extension requiring minimal additional analysis.

The most obvious advantage of a recursive protocol design is in allowing code reuse at both the source and remote hosts. For example, we implemented the PHBDR protocol as a C ++ class library that is simply linked into each of the more advanced protocols. A single class, `TrueObjectSingle` (an implementation of the `TrueObject` interface), represents any source vertex that is being dead reckoned, while `DeadReckonObjectSingle` (an implementation of the `DeadReckonObject` interface) represents the remote model for that vertex. Advanced protocols such as multiple-model channels simply reuse the PHBDR protocol by creating other implementations of `TrueObject` and `DeadReckonObject` that respectively compose instances of `TrueObjectSingle` and `DeadReckonObjectSingle`. This code reuse also reduces the complexity of testing the correctness of a protocol implementation. Given the correctness of the PHBDR protocol library, testing efforts only need to be expended on the "glue" in the higher protocol that links PHBDR instances together.

Recursive structuring also provides greater flexibility for supporting evolution of the simulation system. In particular, the structure provides a well-defined framework for adding new protocols into the simulation or customizing existing protocols for particular entity types. New protocols are introduced by simply adding more implementations of the `TrueObject` and `DeadReckonObject` interfaces, while entity-specific customizations are performed by subclassing from the particular protocol implementation being customized. The biggest remaining issue is how to determine the appropriate protocol implementation to instantiate for a given entity. However, this determination is easily facilitated by providing the needed information in an entity directory service, and if the remote host does not have the necessary implementation available, it can substitute the corresponding implementation superclass which provides a more generic dead reckoning service. The use of loadable languages such as Java [89] for dynamically receiving dead reckoning protocol implementations represents an exciting an area for future research.

8.4 Future Work

Within the simulation domain, the techniques described in this thesis can be improved in several ways. First, additional work is needed to address the limitations to the PHBDR protocol, as discussed in Section 4.4. In particular, variable timeouts, state-replace packets, and minimum latency estimation appear to offer particularly effective protocol extensions. Second, more experience is needed with multiple-detail channels over a broader variety of entity types to provide better guidance on constructing approximate structural models. We envision a tool that can analyze an entity's geometric description and some specification of the entity's behavior to automatically construct a set of remote models for representing the entity's dynamic structure at different levels of detail. Third, PAEs currently rely on octtrees for classifying entity location, but octtrees are cumbersome to manipulate because small changes in an entity's position can cause rather significant changes to the entire octtree structure. Ultimately, one would want to replace "grid-based aggregations" with a hierarchy of bounding spheres, which are better suited for fast collision detection and computation; circular regions have been demonstrated to be at least 5% more effective than rectilinear regions in the two-dimensional case [73]. Finally, we have yet to understand how the ability to load software modules at run-time—a capability provided by languages such as Java [89]—may impact distributed simulations. At a minimum, such languages have the potential to permit hosts to support a broader range of remote modeling algorithms by simply downloading dead reckoning, rendering, and collision detection modules as needed.

However, distributed simulation is just one example of a broader class of applications that demand real-time state consistency across multiple hosts while minimizing bandwidth utilization and computational load. As high-bandwidth networks and cheap, high-performance machines become commonplace, more people will use applications involving real-time monitoring and control of remote devices, collaborative document editing, and distributed development and testing of engineering prototypes.

Although the challenges faced by these applications resemble those faced by distributed simulation, developers currently construct application-specific solutions. To a large extent, this thesis is guilty of the same error: it has concentrated on protocols to support the remote modeling of physical entities within a distributed simulation application. For example, the PHBDR protocol assumes that the values being modeled are continuous; moreover, it is particularly optimized to handle particular types of entity behavior, such as smooth motion and collisions. Although PHBDR does relax most of the assumptions made by previous dead reckoning protocols, clearly, one could go further in that direction. Similarly, the multiple-detail channel design assumes that the modeled entities have a physical structure, and PAEs assume that level-of-detail selection at hosts is based primarily on entity type and location.

In the long term, we envision a suite of basic protocols and techniques that can support a broader variety of dissemination-oriented applications. Besides supporting entity and other remote modeling, such a library would probably allow applications to query for available bandwidth and computational resources, select and manage data channel subscriptions based on these resource constraints, and perform information discovery. Effective subscription-based allocation and sharing of limited bandwidth resources is an area of active research [55, 32]. A key challenge in deploying a dissemination support library is determining to what extent it can provide general-purpose functionality without sacrificing the performance offered by application-specific code.

The decoupling techniques developed by this thesis also should be generally useful in large distributed applications, such as multimedia dissemination systems, information retrieval systems, or even the World-Wide Web [6]. In each of these applications, sources might be engineered to provide information at multiple levels of detail, and clients could be configured to select the appropriate detail based on user needs and resource availability. We expect that this approach might be particularly useful in addressing the variable bandwidth connectivity (ranging from kilobit-per-second mobile links to gigabit-per-second LANs) that data sources must contend with in today's network environment. Research remains to understand the appropriate levels of detail that sources would provide for these applications and to evaluate the effectiveness of the multiple channel approach when it is used in domains outside simulation. Initial work in this direction has begun in support of video dissemination over heterogeneous networks [10, 15].

Finally, prior work has well established the effectiveness of recursive protocol design for transport-layer protocols. As far as we know, the protocol family presented in this thesis represents the first example of this design technique for application-level protocols. Based on our positive experiences with its use for addressing the remote modeling problems, we are confident that recursive structuring will be useful not only in other areas of distributed simulation such as collision detection but also in other high-performance distributed applications. After the software development community gains additional experience with the technique, we envision the emergence of libraries and frameworks specifically aimed at supporting the development of recursive protocols.

8.5 Perspective

With increasing video resolution, network bandwidth, and processor speed, distributed simulation and visualization systems are becoming increasingly common in the scientific, industrial, and entertainment industries. At the same time, people's expectations of distributed interactive simulation have also increased considerably. These expectations—whether they be of entity counts, distance between participants, detail of entity models, or interaction level between entities—have always stayed a few years ahead of the available hardware, software, and network technology.

As an example of the expectations growth, the SIMNET system in 1989 supported up to 200 entities. The initial goal for the 1994 STOW system was 10,000 entities, and the initial goal for the 1997 STOW system was 100,000 entities. Moreover, though desired entity counts may not continue to grow at this pace, the complexity of those entity models is surely going to increase at a similar rate over the next few years. We therefore estimate that the complexity expectations for distributed simulations are increasing by a factor of ten every three years, as shown in Figure 8.1.

Until recently, meeting these expectations meant simply waiting for one or two years until the technology curve advanced. However, this "wait until we can do it" approach is no longer viable because the growth rate of expectations and the increasing widespread uses of distributed simulations are outpacing the technology growth rate. For example, processor performance only doubles every two-to-three years. Moreover, though a system might be developed to support current simulation demands, it is unlikely to immediately scale and be deployable in a wide-area environment where it is being used simultaneously by multiple independent user groups. To illustrate these limitations, the deployed 1994 STOW system supported only 1,500 entities, and the 1997 STOW system is expected to service 5,000 entities. As revealed by Figure 8.1, we estimate that the complexity of

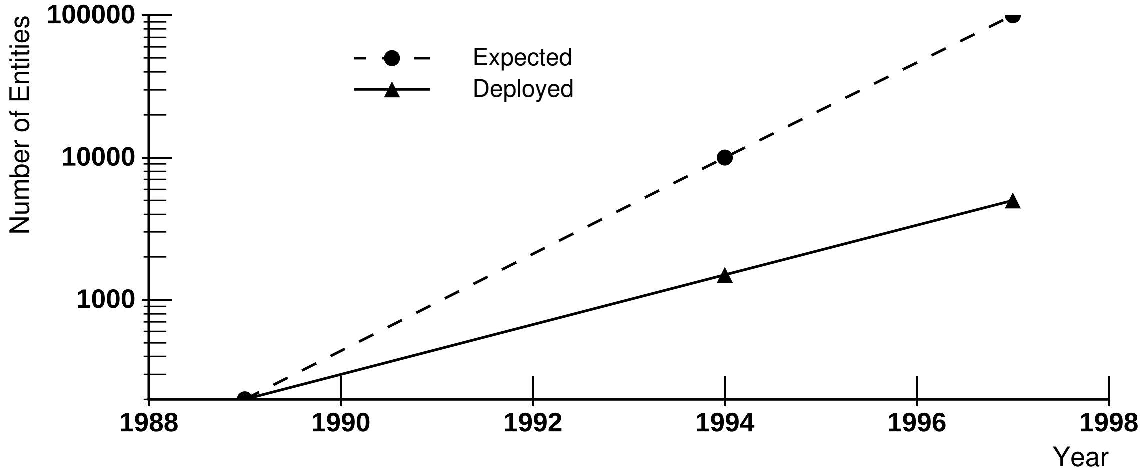

Figure 8.1: Divergence of Expected and Deployed Simulation Complexity in the STOW Program

deployed systems is only increasing by a factor of three every three years.

We conclude therefore that distributed simulation and visualization systems must now be *re-designed* to meet future demands; we can no longer afford to simply carry existing systems forward onto new hardware technology. Moreover, we expect this need for re-design to persist for several years.

With this perspective in mind, this thesis has attempted to take a fresh look at distributed simulation with the overriding goal of scalability. We believe that this approach is indicative of how future research into distributed simulations—and into large-scale and real-time distributed applications generally—will proceed. Considerable work remains to be done in this rapidly emerging area, for these applications still face numerous fundamental challenges. However, our new techniques do offer a promising basis for developing more advanced systems.

Bibliography

[1] Advanced Research Projects Agency (ARPA). "STOW 97 Program Plan." May 1994.

[2] Anupam, Vinod and Chandrajit L. Bajaj. "Distributed and Collaborative Visualization." *IEEE Multimedia*, 1(2):39–49, Summer 1994.

[3] Arai, Fumihito, et al. "Distributed Virtual Environment for Intravascular Tele-Surgery Using Multimedia Telecommunication." In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS)*, Pages 79–85, Santa Clara, California, March 1996. IEEE Neural Networks Council.

[4] Arthur, Kevin W., Kellogg S. Booth, and Colin Ware. "Evaluating 3D Task Performance for Fish Tank Virtual Worlds." *ACM Transactions on Information Systems*, 11(3):239–265, July 1993.

[5] Berglund, Eric J. and David R. Cheriton. "Amaze: A Multiplayer Computer Game." *IEEE Software*, 2(1):30–39, May 1985.

[6] Berners-Lee, Tim, et al. "World-Wide Web: The Information Universe." *Electronic Networking: Research, Applications, and Policy*, 1(2):52–58, Spring 1992.

[7] Birman, Kenneth P. and Keith Marzullo. "The ISIS Distributed Programming Toolkit and the META Distributed Operating System: A Brief Overview." In Agrawala, Ashok K., Karen D. Gordon, and Phillip Hwang, editors, *Mission Critical Operating Systems*, Pages 32–35. IOS Press, Amsterdam, 1992.

[8] Blau, Brian, et al. "Networked Virtual Environments." In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, Pages 157–160, Cambridge, Massachusetts, March 1992. ACM SIGGRAPH. Published as *Computer Graphics* 26 Special Issue.

[9] Bricken, William and Geoffrey Coco. “The VEOS Project.” *Presence: Teleoperators and Virtual Environments*, 3(2):111–129, Spring 1994.

[10] Brown, Tom, et al. “Packet Video for Heterogeneous Networks Using CU-SeeMe.” In *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society.

[11] Burchfiel, Jerry. “The Advantages of Using Quarternions Instead of Euler Angles for Representing Orientation.” In *Proceedings of the Third Workshop on Standards for the Interoperability of Defense Simulations*, Pages I:66–82, Orlando, Florida, August 1990. Published as Technical Report IST–CR–90–13, Institute for Simulation and Training, University of Central Florida, Orlando, Florida. Paper also published as White Paper ASD 91–001, Advanced Simulation Group, BBN Systems and Technologies, Cambridge, Massachusetts, July 1990.

[12] Butzer, P. L. and R. L. Stens. “Linear Prediction by Samples From the Past.” In Marks II, Robert J., editor, *Advanced Topics in Shannon Sampling and Interpolation Theory*, Pages 157–184. Springer-Verlag, New York, 1993.

[13] Calvin, James O., et al. “Application Control Techniques System Architecture.” Technical Report RITN–1001–00, MIT Lincoln Laboratories, Lexington, Massachusetts, February 1995.

[14] Carlsson, Christer and Olof Hagsand. “DIVE—A Platform for Multi-User Virtual Environments.” *Computers & Graphics*, 17(6):663–669, November–December 1993.

[15] Chaddha, Navin and Anoop Gupta. “A Frame-Work for Live Multicast of Video Streams Over the Internet.” In *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society.

[16] Cheriton, David R. “Exploiting Recursion to Simplify RPC Communication Architectures.” In *Proceedings of SIGCOMM 1988*, Pages 76–87, Stanford, California, August 1988. ACM SIGCOMM. Published as *Computer Communications Review* 18(4), August 1988.

[17] Colladine, C. R. “Gaussian Curvature and Shell Structures.” In Gregory, J. A., editor, *The Mathematics of Surfaces: The Proceedings of a Conference Organized by the Institute of Mathematics and Its Applications and Held at the University of Manchester, 17–19 September 1984*, Pages 179–196. Clarendon Press, Oxford, 1986.

[18] Cooke, Joseph M., et al. "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions." *Presence: Teleoperators and Virtual Environments*, 1(4):404–420, Fall 1992.

[19] Digital Equipment Corporation. "Newest Alpha Microprocessor Hits 500MHz, Alpha Tops for Windows NT Visual Computing." Press Release. July 1996.

[20] Eckerson, Wayne. "Approaches to OLAP: Making Sense of the Religious Wars Surrounding OLAP Implementations." *Open Information Systems*, 11(2):3–36, February 1996. Published by Patricia Seybold Group.

[21] Escobar, Julio, Craig Partridge, and Debra Deutsch. "Flow Synchronization Protocol." *ACM/IEEE Transactions on Networking*, 2(2):111–121, April 1994.

[22] Farin, Gerald. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, Pages 29–36, 83–86, 121–129. Academic Press, Boston, 1993.

[23] Feineman, Laura, et al. "Aggregate Level Simulation Protocol (ALSP) Project 1994 Annual Report." Technical Report MTR 95W0000017, The MITRE Corporation, McLean, Virginia, March 1995.

[24] Floyd, Sally, et al. "A Reliable Multicast Framework for Light-Weight Sessions and Application-Level Framing." In *Proceedings of SIGCOMM 1995*, Pages 342–356, Cambridge, Massachusetts, August 1995. ACM SIGCOMM. Published as *Computer Communications Review* 25(4), October 1995.

[25] Foster, Lester, Paul Maassel, and Dennis McBride. "The Characterization of Entity State Error and Update Rate for Distributed Interactive Simulation." In *Proceedings of the Eleventh Workshop on Standards for the Interoperability of Defense Simulations*, Pages I:61–73, Orlando, Florida, September 1994. Published as Technical Report IST–CF–94–02, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[26] Foxlin, Eric. "Inertial Head-Tracker Sensor Fusion by a Complementary Separate-Bias Kalman Filter." In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS)*, Pages 185–194, Santa Clara, California, March 1996. IEEE Neural Networks Council.

[27] Friedman, Martin, Thad Starner, and Alex Pentland. "Device Synchronization Using an Optimal Linear Filter." In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*,

Pages 57–62, Cambridge, Massachusetts, March 1992. ACM SIGGRAPH. Published as *Computer Graphics* 26 Special Issue.

[28] Funda, Janez, Russell H. Taylor, and Richard P. Paul. "On Homogenous Transforms, Quaternions, and Computational Efficiency." *IEEE Transactions on Robotics and Automation*, 6(3):382–388, June 1990.

[29] Funkhouser, Thomas A. "RING: A Client-Server System for Multi-User Virtual Environments." In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, Pages 85–92, Monterey, California, April 1995. ACM SIGGRAPH.

[30] Funkhouser, Thomas A. and Carlo H. Séquin. "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments." In *SIGGRAPH 1993 Conference Proceedings*, Pages 247–254, Anaheim, California, August 1993. ACM SIGGRAPH. Published as *Computer Graphics* 27 Annual Conference Series Special Issue.

[31] Heckbert, Paul S. and Michael Garland. "Multiresolution Modeling for Fast Rendering." In *Proceedings of Graphics Interface 1994*, Pages 43–50, Banff, Alberta, May 1994. Canadian Information Processing Society.

[32] Hoffman, Don and Michael Speer. "Hierarchical Video Distribution Over Internet-Style Networks." In *Proceedings of the IEEE International Conference on Image Processing*, Lausanne, Switzerland, September 1996. IEEE Signal Processing Society.

[33] Holbrook, Hugh W., Sandeep K. Singhal, and David R. Cheriton. "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation." In *Proceedings of SIGCOMM 1995*, Pages 328–341, Cambridge, Massachusetts, August 1995. ACM SIGCOMM. Published as *Computer Communications Review* 25(4), October 1995.

[34] Hoppe, Hughes. "Progressive Meshes." In *SIGGRAPH 1996 Conference Proceedings*, Pages 99–108, New Orleans, Louisiana, August 1996. ACM SIGGRAPH. Published as *Computer Graphics* 30 Annual Conference Series Special Issue.

[35] Hubbard, Philip M. "Real-Time Collision Detection and Time-Critical Computing." In *Proceedings of the First Workshop on Simulation and Interaction in Virtual Environments (SIVE)*, Pages 92–96, Iowa City, Iowa, July 1995. ACM SIGGRAPH.

[36] Institute for Electrical and Electronics Engineers. *Dead Reckoning Definitions and Algorithms*, Annex B. In *IEEE Std 1278.1–1995* [37], September 1995.

[37] Institute for Electrical and Electronics Engineers. *IEEE Standard for Distributed Interactive Simulation—Application Protocols*. IEEE Std 1278.1–1995. IEEE Standards Press, Piscataway, New Jersey, September 1995.

[38] Institute for Electrical and Electronics Engineers. *IEEE Standard for Distributed Interactive Simulation—Communication Services and Profiles*. IEEE Std 1278.2–1995. IEEE Standards Press, Piscataway, New Jersey, September 1995.

[39] Johnson, Marty P. "STOW 97 Bandwidth Estimate (Revision 6)." July 1996. Mr. Johnson is an enginner with the Simulation Technology Division, Systems Applications International Corporation (SAIC), Arlington, Virginia.

[40] Kahaner, David, Cleve Moler, and Stephen Nash. *Numerical Methods and Software*, Pages 87–95. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[41] Kailath, Thomas. *Lectures on Wiener and Kalman Filtering*. Springer-Verlag, New York, 1981.

[42] Kalman, Rudolph E. "A New Approach in Linear Filtering and Prediction Problems." *Transactions of the American Society of Mechanical Engineers [ASME], Journal of Basic Engineering*, 82D:35–45, March 1960.

[43] Katz, Amnon and Kenneth Graham. "Dead Reckoning for Airplanes in Coordinated Flight." In *Proceedings of the Tenth Workshop on Standards for the Interoperability of Defense Simulations*, Pages II:5–13, Orlando, Florida, March 1994. Published as Technical Report IST–CR–94–01, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[44] Kaul, Anil and Jarek Rossignac. "Solid-Interpolating Deformations: Construction and Animations of PIPs." In *EUROGRAPHICS'91: Proceedings of the European Computer Graphics Conference and Exhibition*, Pages 493–505, Vienna, Austria, September 1991. Eurographics Association, North-Holland.

[45] Kay, Jonathan and Joseph Pasquale. "The Importnace of Non-Data Touching Processing Overheads in TCP/IP." In *Proceedings of SIGCOMM 1993*, Pages 259–268, San Francisco,

California, September 1993. ACM SIGCOMM. Published as *Computer Communications Review* 23(4), October 1993.

[46] Kenan Systems Corporation. “An Introduction to Multidimensional Database Technology.” White Paper. 1995.

[47] Kitamura, Yoshifumi. Personal communication. July 1994. Dr. Kitamura is a developer of the Virtual Space Teleconferencing system at ATR Communication Systems Research Laboratories in Kyoto, Japan.

[48] Levoy, Marc. Personal communication. February 1994. Dr. Levoy is Assistant Professor of Computer Science at Stanford University.

[49] Liang, Jiandong, Chris Shaw, and Mark Green. “On Temporal-Spatial Realism in the Virtual Reality Environment.” In *Proceedings of the Fourth Annual Symposium on User Interface Software and Technology (UIST)*, Pages 19–25, Hilton Head, South Carolina, November 1991. ACM SIGGRAPH and ACM SIGCHI.

[50] Lin, Kuo-Chi and Daniel E. Schab. “The Performance Assessment of the Dead Reckoning Algorithms in DIS.” *Simulation*, 63(5):318–325, November 1994.

[51] Lin, Kuo-Chi and Daniel E. Schab. “Study on the Network Load in Distributed Interactive Simulation.” In *Flight Simulation Technologies Conference*, Scottsdale, Arizona, August 1994. American Institute of Aeronautics and Astronautics (AIAA).

[52] Macedonia, Michael R. *A Network Software Architecture for Large-Scale Virtual Environments*. Ph.D Dissertation, Naval Postgraduate School, Department of Computer Science, Monterey, California, June 1995. Available as `http://www-npsnet.cs.nps.navy.mil/npsnet/publications/Michael.Macedonia.thesis.ps.Z.`

[53] Macedonia, Michael R., et al. “NPSNET: A Network Software Architecture for Large-Scale Virtual Environments.” *Presence: Teleoperators and Virtual Environments*, 3(4), Fall 1994.

[54] Macedonia, Michael R., David R. Pratt, and Michael J. Zyda. “Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments.” In *Proceedings of the 1995 Virtual Reality Annual International Symposium (VRAIS)*, Pages 2–10, Research Triangle Park, North Carolina, March 1995. IEEE Neural Networks Council.

[55] McCanne, Steven, Van Jacobson, and Martin Vetterli. "Receiver-driven Layered Multicast." In *Proceedings of SIGCOMM 1996*, Pages 117–130, Stanford, California, August 1996. ACM SIGCOMM. Published as *Computer Communications Review* 26(4), August 1996.

[56] Mendel, Jerry M. *Discrete Techniques of Parameter Estimation: The Equation Error Formulation*, Chapter 4–6. Marcel Dekker, Inc., New York, 1973.

[57] Meyer, Kenneth, Hugh L. Applewhite, and Frank A. Biocca. "A Survey of Position Trackers." *Presence: Teleoperators and Virtual Environments*, 1(2):173–199, Spring 1992.

[58] Mills, David L. "Internet Time Synchronization: The Network Time Protocol." *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[59] Mills, David L. "Improved Algorithms for Synchronizing Computer Network Clocks." *IEEE/ACM Transactions on Networking*, 3(3):245–254, June 1995.

[60] Milner, Stuart. "STOW Real-Time Communications Architecture: Requirements, Approach, and Rationale." May 1995. Presentation. Dr. Milner is a Program Manager at ARPA.

[61] Mogul, Jeffrey C. and Anita Borg. "The Effect of Context Switches on Cache Performance." In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)*, Pages 75–84, Santa Clara, California, April 1991. ACM SIGARCH, SIGOPS, and SIGPLAN. Published as *Operating Systems Review* 25 Special Issue, April 1991.

[62] Mogul, Jeffrey C. and K. K. Ramakrishnan. "Eliminating Receive Livelock in an Interrupt-Driven Kernel." Research Report 95/8, Western Research Laboratory, Digital Equipment Corporation, Palo Alto, California, December 1995.

[63] Mosberger, David, et al. "Analysis of Techniques to Improve Protocol Processing Latency." In *Proceedings of SIGCOMM 1996*, Pages 73–84, Stanford, California, August 1996. ACM SIGCOMM. Published as *Computer Communications Review* 26(4), October 1996.

[64] Nagashima, Yoshio, Hiroshi Agawa, and Fumio Kishino. "3D Face Model Reproduction Method Using Multi View Images." In *Visual Communications and Image Processing '91: Image Processing*, Pages I:566–573, Boston, Massachusetts, November 1991. SPIE (International Society for Optical Engineering). Published as *Proceedings of the SPIE* 1606, Part 1.

[65] Nakamura, Nobutatsu. Personal communication. August 1994. Mr. Nakamura heads the "Networked VR" project at the Applied Information Technology Research Laboratory of NEC Corporation in Kawasaki, Japan.

[66] Nakamura, Nobutatsu, Keiji Nemoto, and Katsuya Shinohara. "Distributed Virtual Reality System for Cooperative Work." *NEC Research and Development*, 35(4):403–409, October 1994.

[67] Ohya, Jun, et al. "Real-Time Reproduction of 3D Human Images in Virtual Space Teleconferencing." In *Proceedings of the 1993 Virtual Reality Annual International Symposium (VRAIS)*, Pages 408–414, Seattle, Washington, September 1993. IEEE Neural Networks Council.

[68] Oppenheimer, Alan V. and Ronald W. Schafer. *Discrete-Time Signal Processing*, Chapter 2–3, Pages 39–91. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[69] Pope, Arthur. "The SIMNET Network and Protocols." Technical Report 7102, BBN Systems and Technologies, Cambridge, Massachusetts, July 1989.

[70] Pratt, David R. *A Software Architecture for the Construction and Management of Real-Time Virtual Worlds*. Ph.D Dissertation, Naval Postgraduate School, Department of Computer Science, Monterey, California, June 1993. Available as `http://www-npsnet.cs.nps.navy.mil/npsnet/publications/David.Pratt.thesis.ps.Z`.

[71] Pratt, David R., et al. "Insertion of an Articulated Human Into a Networked Virtual Environment." In *Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, Pages 84–90, Gainesville, Florida, December 1994. IEEE Computer Society Press.

[72] Quéau, Philippe. "Televirtuality: The Merging of Telecommunications and Virtual Reality." *Computers & Graphics*, 17(6):691–693, November–December 1993.

[73] Rak, Steven J. and Daniel J. Van Hook. "Evaluation of Grid-Based Relevance Filtering for Multicast Group Assignment." In *Proceedings of the Fourteenth Workshop on Standards for the Interoperability of Defense Simulations*, Pages II:739–747, Orlando, Florida, March 1996. Published as Technical Report IST–CF–96–03, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[74] Rebo, Robert K. and Phil Amburn. "A Helmet-Mounted Virtual Environment Display System." In *Helmet-Mounted Displays*, Pages 80–84, Orlando, Florida, March 1989. SPIE (International Society for Optical Engineering). Published as *Proceedings of the SPIE* 1116.

[75] Rohlf, John and James Helman. "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics." In *SIGGRAPH 1994 Conference Proceedings*, Pages 381–394, Orlando, Florida, July 1994. ACM SIGGRAPH. Published as *Computer Graphics* 28 Annual Conference Series Special Issue.

[76] Rolfe, J. M. and K. J. Staples, editors. *Flight Simulation*, Pages 45–47. Cambridge University Press, Cambridge, 1986.

[77] Rossignac, Jarek and Paul Borrel. "Multi-Resolution 3D Approximations for Rendering Complex Scenes." In Falcidieno, Bianca and Toshiyasu L. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, Pages 455–465. Springer-Verlag, New York, 1993. Also available as Research Report RC 17697 (77951), IBM, Yorktown Heights, New York, February 1992.

[78] Sanghi, Dheeraj, et al. "Experimental Assessment of End-to-End Behavior on Internet." Technical Report UMCP–CSD:CS–TR–2909, Department of Computer Science, University of Maryland, College Park, Maryland, June 1992.

[79] Schaffer, Robert. "The Applicability of Distributed Simulation Techniques to High Performance Aircraft." In *Flight Simulation Technologies Conference*, Hilton Head Island, South Carolina, August 1992. American Institute of Aeronautics and Astronautics (AIAA).

[80] Schilit, Bill N. and Marvin M. Theimer. "Disseminating Active Map Information to Mobile Hosts." *IEEE Network*, 8(5):22–32, September–October 1994.

[81] Shoemake, Ken. "Animating Rotation with Quaternion Curves." In *SIGGRAPH 1985 Conference Proceedings*, Pages 245–254, San Francisco, California, July 1985. ACM SIGGRAPH. Published as *Computer Graphics* 19(3).

[82] Singh, Gurminder, et al. "BrickNet: A Software Toolkit for Network-Based Virtual Worlds." *Presence: Teleoperators and Virtual Environments*, 3(1):19–34, Winter 1994.

[83] Singhal, Sandeep K. and David R. Cheriton. "Using a Position History-Based Protocol for Distributed Object Visualization." In *Designing Real-Time Graphics for Entertainment*, volume 14 of *SIGGRAPH 1994 Lecture Notes*, Chapter 10. July 1994. Also published as Technical Report STAN–CS–TR–94–1505, Computer Science Department, Stanford University, Stanford, California, February 1994. Available by anonymous FTP from `elib.Stanford.EDU` in `/pub/reports/cs/tr/94/1505/CS-TR-94-1505.ps`.

[84] Singhal, Sandeep K. and David R. Cheriton. "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality." *Presence: Teleoperators and Virtual Environments*, 4(2):169–193, Spring 1995.

[85] Singhal, Sandeep K. and David R. Cheriton. "Using Projection Aggregations to Support Scalability in Distributed Simulation." In *Proceedings of the 1996 International Conference on Distributed Computing Systems (ICDCS)*, Pages 196–206, Hong Kong, Hong Kong, May 1996. IEEE Computer Society.

[86] Smith, Joshua E., Kevin L. Russo, and Lawrence C. Schuette. "Prototype Multicast IP Implementation in ModSAF." In *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Defense Simulations*, Pages 175–178, Orlando, Florida, March 1995. Published as Technical Report IST–CF–95–01, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[87] Sorenson, Harold W. "Least-Squares Estimation: From Gauss to Kalman." *IEEE Spectrum*, 7(7):63–68, July 1970.

[88] Stamen, Jeffrey P. "Structuring Databases for Analysis." *IEEE Spectrum*, 30(10):55–58, October 1993.

[89] Sun Microsystems. *The Java Programming Language*. Addison-Wesley, Reading, Massachussets, 1996.

[90] Talluri, Raj and J. K. Aggarwal. "Positional Estimation for an Autonomous Mobile Robot in an Outdoor Environment." *IEEE Transactions on Robotics and Automation*, 8(5):573–584, October 1992.

[91] Towers, John and Jack Hines. "Highly Dynamic Vehicles in a Real/Simulated Virtual Environment (HyDy) Equations of Motion of the DIS 2.0.3 Dead Reckoning Algorithms." In

Proceedings of the Tenth Workshop on Standards for the Interoperability of Defense Simulations, Pages II:431–462, Orlando, Florida, March 1994. Published as Technical Report IST–CF–94–01, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[92] Troxel, Gregory. “Traffic Model for STOW 97.” December 1994. Distributed by BBN Systems and Technologies.

[93] Trunk, Gerard V. and J. Donald Wilson. “Tracking Filters for Multiple-Platform Radar Integration.” Technical Report 8087, Naval Research Laboratory, Washington, DC, December 1976. National Technical Information Service Document AD-A 034608; Government Document D 210.8:8087.

[94] Tsutsui, Kyoya. “Method of Car Navigation.” Patent (Japan) 250816. 1989. Translated from original.

[95] Turk, Greg. “Re-Tiling Polygonal Surfaces.” In *SIGGRAPH 1992 Conference Proceedings*, Pages 55–64, Chicago, Illinois, July 1992. ACM SIGGRAPH. Published as *Computer Graphics* 26(2), July 1992.

[96] United States Army, U.S. Atlantic Command (USACOM). “STOW 97 ACTD Playbook.” May 1996. Version 1.0.

[97] Vagany, J., M. J. Aldon, and A. Fournier. “Mobile Robot Attitude Estimation by Fusion of Intertial Data.” In *Proceedings of the IEEE International Conference on Robotics and Automation*, Pages 277–282, Atlanta, Georgia, May 1993. IEEE Robotics and Automation Society.

[98] Van Hook, Daniel J. “Simulation Tools for Developing and Evaluating Networks and Algorithms in Support of STOW 94.” Presented at Scaleability Peer Review, San Diego, California, 19–20 August 1993. Dr. Van Hook is a member of the research staff at MIT Lincoln Laboratories, Lexington, Massachussets.

[99] Van Hook, Daniel J. Personal communication. September 1994. Dr. Van Hook is a member of the research staff at MIT Lincoln Laboratories, Lexington, Massachussets.

[100] Van Hook, Daniel J., James O. Calvin, and Duncan C. Miller. "A Protocol Independent Compression Algorithm (PICA)." Advanced Distributed Simulation Project Memorandum 20PM–ADS–005, MIT Lincoln Laboratories, Lexington, Massachusetts, April 1994.

[101] Van Wechel, R. "Analysis of Dead Reckoning Update Time." In *Proceedings of the Eighth Workshop on Standards for the Interoperability of Defense Simulations*, Pages III:245–255, Orlando, Florida, March 1993. Published as Technical Report IST–CR–93–10, Institute for Simulation and Training, University of Central Florida, Orlando, Florida.

[102] Wang, Chu P., Lawrence Koved, and Semyon Dukach. "Design for Interactive Performance in a Virtual Laboratory." In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, Pages 39–40, Snowbird, Utah, March 1990. ACM SIGGRAPH. Published as *Computer Graphics* 24(2), March 1990.

[103] Whitlock, Adam H. "Draft Estimate of Bandwidth Demand for STOW 97." May 1995. Distributed by the Naval Research and Development Center (NRaD), San Diego, California.

[104] Willman, Warren W. "Recursive Filtering Algorithms for Ship Tracking." Technical Report 7969, Naval Research Laboratory, Washington, DC, April 1976. National Technical Information Service Document AD-A 024329; Government Document D 210.8:7969.

[105] Xu, Gang, et al. "Three-Dimensional Face Modeling for Virtual Space Teleconferencing Systems." *Transactions of the Institute of Electronics, Information and Communication Engineers (IEICE)*, E73(10):1753–1761, October 1990.

[106] Zelesko, Matthew J. and David R. Cheriton. "Specializing Object-Oriented RPC for Performance and Functionality." In *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS)*, Pages 175–187, Hong Kong, Hong Kong, May 1996. IEEE Computer Society.