A Distributed Implementation of an N-body Virtual World Simulation

Mark Parris, Carl Mueller, Jan Prins,
Adam Duggan, Quan Zhou, Erik Erikson

Department of Computer Science, Sitterson Hall
University of North Carolina
Chapel Hill, NC 27599-3175

Abstract

Meeting the challenge of real-time visualization of a complex physical simulation can require several specialized computers. The virtual-world simulation described in this paper combines a highly-parallel computer for physical simulation and a custom graphics engine for display communicating via TCP/IP over an ethernet. Application-level protocols to provide the high-throughput, low-latency communication required to support an interactive virtual world are investigated. Experiments demonstrate that a limited write-ahead protocol gives the best tradeoff between latency and throughput for this application.

1. Introduction

High performance graphics engines capable of generating virtual worlds offer an expanded opportunity for user interaction with computer generated simulations. In this paper we report on a prototype implementation of a real-time virtual-world N-body simulation. Such a simulation permits an observer to explore a volume of space occupied by spheres representing bodies of varying mass and charge whose motions are governed by forces due to, for example, gravitational and electrostatic interactions.

In general, a distributed implementation involving several different machines may be required to achieve real-time visualization. In our case, a custom graphics engine, Pixel-Planes 5 [HPE+89] can provide real-time display of the set of bodies in response to changes in viewpoint and orientation of the user, as well as changes in the locations of bodies. However, this rendering engine is not well-suited to the calculation of the N-body simulation at real-time rates. Instead, a general-purpose highly-parallel computer, the Maspar MP-1, is used for this component of the application.

The real-time visualization of and interaction with complex simulations is a research goal of a number of related projects. In the molecular docking experiment of the GRIP project [BOB+90], an operator viewing a display showing a drug molecule and a protein molecule tries to navigate the drug molecule to a docking site on the protein molecule. The forces experienced by the drug molecule as a result of its interaction with the protein molecule are calculated in real-time by the MP-1 and a corresponding actual force is applied to the hand grip controlling the drug position in the docking experiment. Compared to the application described in this paper, the docking application requires similarly low-latency but needs far less throughput.

The VISTAnet project [STB91] currently under development will use a remote Cray Y-MP as the simulation engine and Pixel-Planes 5 as the rendering engine in a radiation-therapy beam-placement application. In this case the volume of data passed between the Y-MP and Pixel-Planes 5 requires the use of a HIPPI [ANSI90] interface to a high-speed (OC-48) network. First high-speed operation is scheduled for 1993. Compared with our application this project requires higher throughput but tolerates far more latency, since the dose-calculation does not run in real-time.

The N-body application is representative of the sorts of applications we expect to be able to visualize in the future, and its implementation was undertaken to gain some insight on the possibilities and limitations of distributed real-time visualization systems. The key challenge is to find techniques that will increase the throughput and decrease the latency of communication between the distributed components. For the construction of this prototype we developed an application-specific high-level communication protocol that limits latency yet permits reasonable throughput.

The remainder of the paper is organized as follows. Section 2 describes the hardware components of the system. Section 3 gives a brief description of the application (a videotape of the system in operation is available to provide more detail for interested readers) and the structure of its implementation. Section 4 focuses on the high-level communication protocol for the distributed components of the application and its implementation using TCP/IP [Com91]. Section 5 summarizes the performance of the current system (a qualitative sense of the performance can be obtained from the videotape). We conclude with a discussion of other approaches to reducing latency and improving throughput for this application.

2. Hardware Components

Figure 1 shows the hardware components and their interconnection as used in this application. The primary components are the head-mounted display interface, Pixel-Planes 5 and the Maspar MP-1.

The head-mounted display (HMD) interface consists of three main pieces: the HMD itself, a hand-held mouse, and a tracking system. The HMD is a helmet that holds two LCD video displays and their associated optics in front of the user's eyes, plus headphones for audio feedback. The mouse is built from a modified multi-button joystick handle. For the tracking system we use a Polhemus magnetic tracker, consisting of an emitter suspended above the user with sensors attached to the helmet and mouse. The tracking system continuously reports the three-dimensional coordinates and orientation of the user and mouse within approximately a one-meter radius volume where the user may work.

Providing the display is UNC's custom graphics multicomputer, Pixel-Planes 5. This system consists of a number of Intel i860 microprocessors and several custom SIMD arrays of one-bit processors, all connected to a high-speed ring network. Also on the ring are frame-buffers to drive the video screens in the HMD and a host-interface attached to the Sun Microsystems 4/280 front-end computer which controls interaction with the user. The front-end contains an ethernet interface and is attached via serial lines to the Polhemus controller and to the sound server, an Apple Macintosh IIci configured to play back digitized sounds.

The computer we use to calculate the simulation is a 4096-processor Maspar MP-1 computer. The MP-1 consists of a back-end data parallel unit (DPU) attached to a Unix host. The parallel processors are controlled in SIMD fashion by sequential processor known as the array control unit (ACU). This back-end assembly is attached by a VME bus to a Digital Equipment Corporation VAXstation 3520 front-end host with the usual array of I/O devices including an ethernet connection.

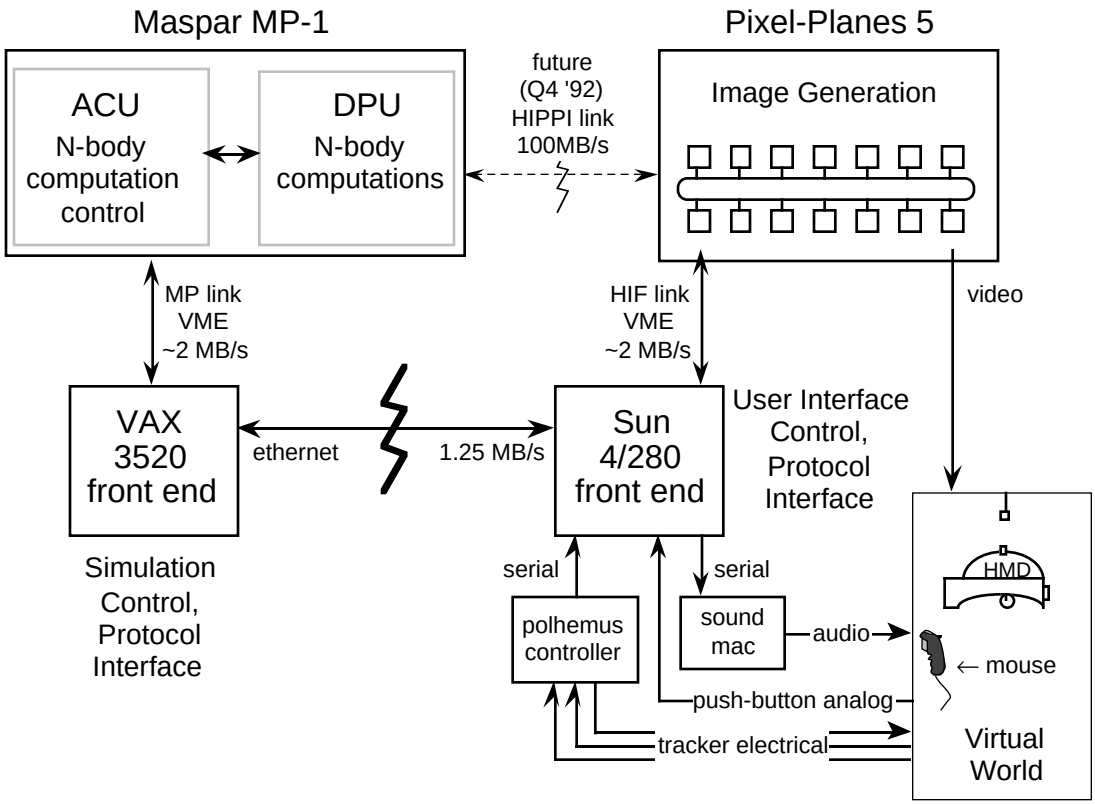

Figure 1. System Overview

3. Application

In our application, an observer looking around the virtual world while the simulation is running sees a collection of independently moving bodies represented by spheres of different colors and sizes. The bodies may optionally be displayed with a velocity vector on each body's surface giving the direction and magnitude of its motion. The observer can apply a force to a body, and can reposition himself by "flying" through the world, or by attaching himself to one of the spheres. When the simulation is stopped, various attributes of the bodies and the simulation can be modified using a 3-D control panel that appears in the virtual world.

The application consists of three major components: the virtual-world user interface, the simulation, and the protocol library which connects them. We first discuss the user's view of the user interface and simulations, and then consider the basics of their operation. The protocol is discussed in section 4 below.

3.1. User Interface

The design of the interface was limited by the fact that the user has only a multi-button mouse for input and the HMD for output. With these the user must be able to set up, alter, control, and save a simulation scenario. We chose a system of virtual control panels and tools similar to that provided by 3dm, a 3-D modelling program [BDH+92] A number of control panels are available to allow the user to

perform operations such as setting parameters, selecting options, saving the current setup, and selecting different tools. These panels appear as 3-D objects in the user's virtual world. A tool, indicated by a special cursor attached to the user's hand, allows the user to directly manipulate some global features of the environment, such as the environment's orientation or relative size with respect to the user, as well as features of individual bodies in the environment.

The interface is logically divided into two major modes: editing mode and run mode. This is done since it was deemed inconvenient to modify bodies while they are in motion, and in addition it simplifies the requirements of the simulation and the communications protocol. A tool selection panel provides easy access to certain common tools. The edit control panels feature dials and sliders for adjusting certain continuous parameters, a numeric display for feedback on the adjustments, and buttons for toggling specific features. From run mode user interaction is more limited; only the tool selection panel is still accessible, and it now provides some additional functions: a stop selection to exit run mode, a thrust tool to allow the user to apply an acceleration to the selected body, and a sample-ratio slider to change the apparent speed of the simulation.

3.2. Simulation

The N-body simulation is calculated by numerical integration of the differential equations describing each body's motion due to forces accumulating from all pairwise interactions with other bodies [Gre90]. Currently only gravitational forces are calculated; the calculation of electrostatic and other forces is simple to add. The numerical integration method used is the Euler-Heun Predictor-Corrector method. Although this yields reasonable accuracy with small time steps, large simulations, such as those of the solar system, accumulate large errors with time steps of interest (e.g . 1 sec real-time = 1 hour virtual world time). Therefore we partitioned the bodies into two classes: α bodies and β bodies. The motion of α bodies is described parametrically or by an ephemeris (table of times and locations). The motion of β bodies is simulated using the N-body calculation. All bodies participate in this calculation, but only β bodies react to the calculated forces. This is consistent with a model in which the α bodies are undergoing rigid motion or in which the α bodies are substantially more massive than β bodies.

3.3. Implementation

The user begins a simulation by running the user-interface, specifying a scenario to load. After initializing the simulation, the interface comes up in editing mode, where the user may adjust the scenario or start the simulation. The main control loop for the user interface involves these main actions:

- reading the tracker information and determining the user's viewpoint
- processing button-press and mouse-position information to see if the user has issued a command
- checking for any messages from the simulation (none are expected during edit mode)
- sending down to the graphics processors any updated body information
- drawing new views for the user's HMD (done synchronously by Pixel-Planes 5)

When the user presses the Go button, the user interface sends the following messages to the simulation: global scenario information, body information, and a start message. The user interface enters run mode and expects to receive a stream of body position information. This information is passed down to the graphics processors for proper display. Thrust commands are immediately sent to the simulation. Figure 2 illustrates the information flow.

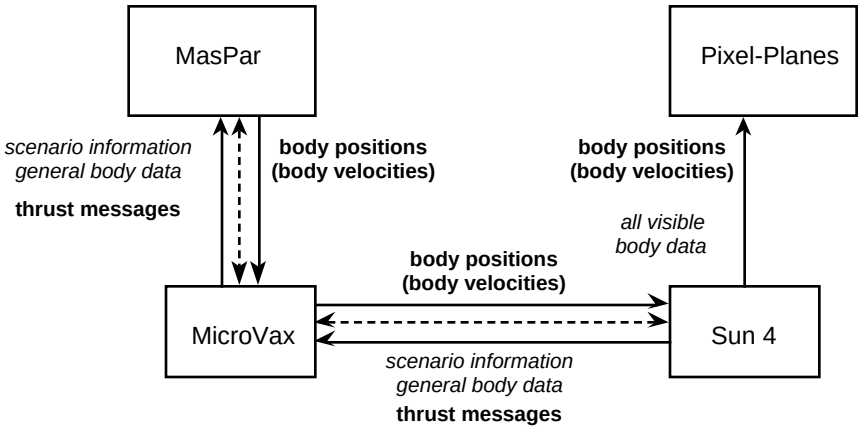

Figure 2. Communications Overview
Bold information is passed in Run mode
Italic information is passed in Edit mode
Dashed lines represent general control information

The simulation also arrives in editing mode once initialization is done. It waits in this mode and processes new scenario or body messages until it sees a start message. Upon entering run mode, the body information is transferred to the Maspar's DPU, the global scenario information is passed to the ACU, and the simulation computation is started. Each time a simulation step is completed the results are transferred into the font-end processor and the front-end processor is signalled. The front-end sends results from the simulation step to the user-interface processor whenever a result is ready. Double-buffering allows computation of body positions and the transmission of the computed positions to be overlapped. If a thrust message is received, it is passed to the back-end the next time the host tells the back-end to perform another computation.

The HMD display requires the generation of about twenty frames per second to maintain the illusion of reality. A full-size Pixel-Planes system can render as many as 85,000 spheres every 1/20th of a second. To transfer this many sphere descriptions would require a 20MB/s data rate into the Pixel-Planes system, far exceeding that of the HIF link to the Sun host. In fact, to even approach the 2 MB/s limit of this link, our application required some custom programming around the standard Pixel-Planes graphics software.

The N-body simulation using the naive $O(N^2)$ algorithm on a full-size MP-1 can generate body positions for about 1000 bodies every 1/20th of a second. Our 4K processor machine can simulate approximately 500 bodies at 20 hz. This represents about a 120 KB/sec data rate from the MP-1 to

Pixel-Planes, indicating that, under ideal circumstances, an ethernet connection should provide ample bandwidth for our application. As we found out, however, circumstances are often far from ideal.

4. Protocol

To support the communications between the user interface and the simulation we had to design and implement a protocol to exchange the initial configuration, editing information, and run mode interaction. Our design also had to be implemented with the constraints of figure 1 in mind. Providing a bandwidth of 1.25 MB/s shared among the computing resources of the department, the ethernet connection between the Vax and the Sun-4 is clearly the communications channel where dataflow will be most restricted. With this prototype we chose to implement the protocol on top of TCP/IP over ethernet to simplify our design. We were aware that this would lead to performance degradation and after analysis of preliminary timings, we chose to address this by choosing communications paradigms that would work suitably well under these constraints.

4.1. Performance characteristics of different modes

While the performance demands during initialization are very relaxed, an important contribution to the performance of the later phases occurs here. Since our system was designed to support a wide variety of simulations, a wealth of data could be exchanged in every update; however, most simulations are concerned with only a subset of the available data. In order to tailor the user interface to the particular simulation, feature lists are exchanged which indicate the features and properties affecting computation and display of the images. The protocol takes advantage of this information in formatting the editing and running messages to contain only the data required.

Like initialization mode, edit mode also has fairly relaxed performance constraints. However, because of the wide variety and possible orderings of operations, the protocol for this phase could be quite complex. Fortunately, a key insight greatly simplified this section of the protocol: the simulation is not interested in the sequence of editing commands that occurred, but only the end result. By keeping copies of the data on both hosts and re-establishing the consistency of the data at the beginning and end of edit mode, the number and complexity of the edit messages is greatly reduced. Thus separate messages are not needed for changes to each body parameter or even to each body, but rather a single message conveying the updated state of all of the bodies. This also avoids the overhead of message traffic which would slow down a highly interactive portion of the application.

Easily placing the highest demands on performance, run mode demanded high throughput and low latency. In order to achieve smooth updates at interactive rates we needed high throughput, but at the same time the application demands low latency for the interactive operation of applying thrust to a body. Like the simulation during edit mode, the user interface is not interested in receiving all of the updates to the bodies, only position and velocity updates, since those are the only parameters of the simulation affecting the display. Other variables such as mass (in a simulation where mass may be modified by the simulation) are not visible and thus need not be updated until edit mode is resumed. As such, the body updates contain only position and, conditionally, velocity (depending upon the feature exchange during initialization).

4.2. Implementation choices

Additionally, we considered implementation methods to reduce the application's susceptibility to network congestion in order to increase throughput. We considered four approaches for achieving high throughput. Simplest among these was completely unconstrained transmission of body updates by the simulation process. In this approach, once the simulation process receives a control message indicating that it should begin computation, it sends body updates to the display process unchecked until the user interface sends a stop message. In this approach the network itself is looked upon as a very large buffer. This simplifies the protocol since it doesn't require acknowledgements, but it also means the simulation can be an arbitrary number of time steps ahead of the image being displayed to the user, causing unpredictable and annoying results when the user attempts to apply thrust to a body. Since interactivity was required, we dismissed this approach.

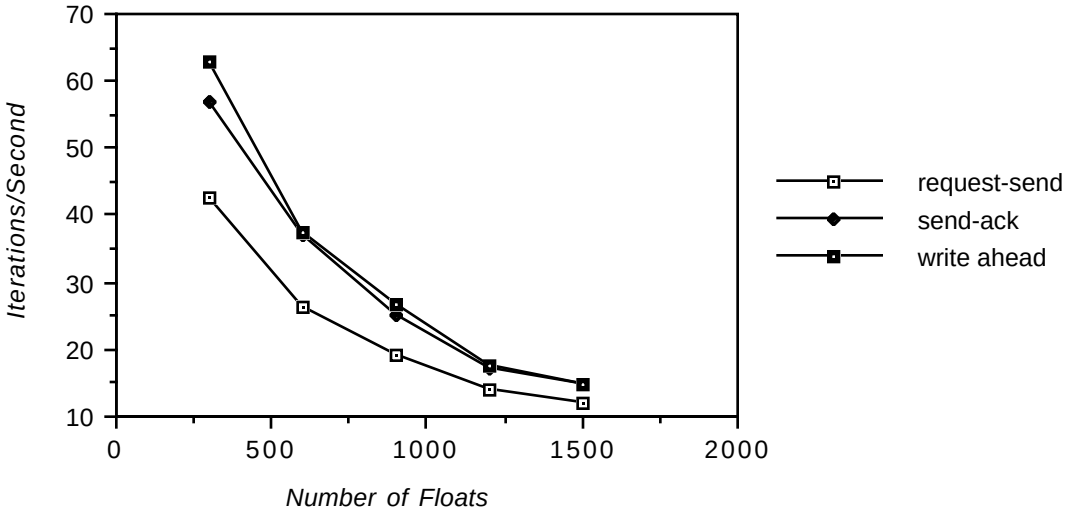

Figure 3. Preliminary Throughput Results

Each method was executed for 2000 iterations at least 10 different times over a 24 hour period with the average number of iterations/second displayed along the vertical axis. The number of floating point numbers sent in each test is indicated along the horizontal axis. The maximum number of unacknowledged messages for the write-ahead case is 10.

However, we did consider the other three alternatives more closely. We ran performance tests with one process acting as sender and another acting as receiver in order to evaluate three other protocol approaches and their performance tradeoffs. Here we define latency as the number of time steps that a user's view might be behind the current state of the simulation rather the amount of real time involved and this definition was our primary one in choosing an implementation. The methods we considered were: *request-send*, where the receiver sends a request for information and the sender then generates a message and sends it; *send-acknowledge*, where the sender generates a message and then waits for an acknowledgement before generating the next one; and finally, an extension of send-acknowledge, *limited write ahead*, where the sender can have up to N unacknowledged messages before blocking. While request-send obviously gives the lowest latency, write ahead would be expected to give the highest. However, we expected that with a sufficiently low limit on N we would be able

get reasonable latency and a higher throughput. The results of our throughput tests are shown in figure 3. From these results we confirmed our initial belief that limited write ahead would offer the best throughput, and thus chose to use it. The results of our choices are discussed below.

5. Performance Results

Once we had a working prototype we modified the user interface to record and playback user interaction so that we could run timing tests and get repeatable results, with the only expected source of variance being in the communication. Since interaction with the virtual world is event driven these modifications were simple to implement and offered substantial power both in the the repeatability of tests and savings in experimentation time since we were able to run a large number of tests without repeatedly donning the HMD and performing a repetitive sequence of actions.

For our latency tests we recorded the time between the application of thrust and the receipt of a body update where that thrust has been applied. We performed the tests repeatedly for 1, 16, 64, and 256 bodies and write-ahead limits of 1, 10, 20, 30, 40, 50 and 100. The observed latencies varied with little correlation to the number of bodies or the write ahead limit. After observing these results, we investigated further and were able to determine that after receiving notification that a message had arrived on the socket, the read call sometimes blocked for periods greater than 0.5 seconds. This variability in performance due to the shared nature of the ethernet is one of its fundamental shortcomings, making it less-than-ideally suited to this type of interactive application.

For our throughput tests we used the same set of variables and timed the number of body updates/second. Our results for four of the write-ahead limits are shown in figure 4. To compare with the earlier figure, note that each body represents six floats, three for position and three for velocity. Throughput increases as the write limit is increased for small numbers of bodies. However, as the number of bodies grows larger the effect of changing the write ahead limit becomes negligible, indicating that a small write ahead limit, providing more consistent interaction with the same throughput, is an effective implementation method.

We also recorded whether or not simulations reached the write ahead limit on positional updates and we found that even with write-ahead limits as large as 100 updates simulations with small numbers of bodies frequently reached this limit. From this we conclude that a write-ahead limit is necessary in order to bound the latency between the simulation and the user interface. Otherwise the latency would continue to grow unchecked.

6. Conclusions and Future Work

We were able to reduce latency and maintain sufficient throughput using the communication protocol described on top of TCP/IP to be able to create a convincing virtual world with many dozens of bodies moving according to the rules of gravity. However, the ethernet connection is still the limiting factor in the performance of the system. There are a number of other approaches that could be explored.

The first is to use the Internet User Datagram Protocol (UDP) for the body update messages during run mode. This separation of the run and edit mode messages should give us the performance benefits of UDP for run mode without sacrificing the reliability we need during editing.

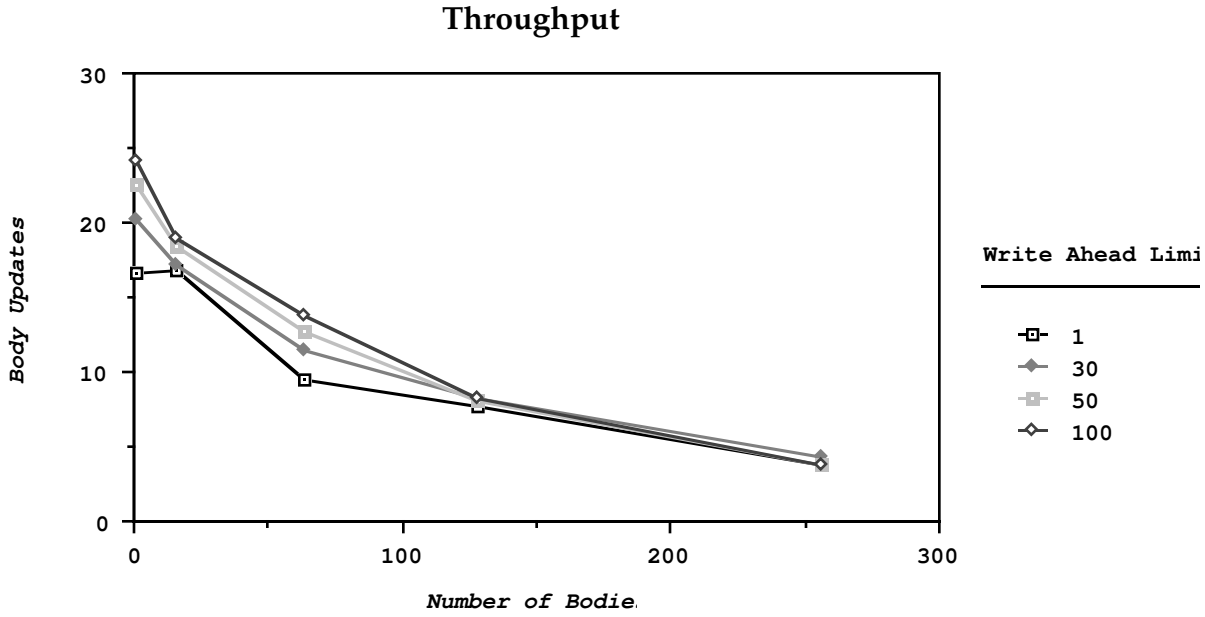

Figure 4. Application Throughput Results

In this figure we show the results for write ahead limits of 1, 30, 50, and 100. Each line indicates a different write-ahead limit. The body updates per second are indicated along the vertical axis, while the number of bodies in each message is indicated along the horizontal.

Additionally, other methods of dealing with current communications limits are possible. The visualization system could provide local interpolation of the paths of bodies and apply these interpolations to the bodies when network delays occur, providing continued interaction with some loss of accuracy. Another approach might be to increase buffering, giving higher throughput, but control latency using a rollback mechanism in the simulation process that assures that user interaction during run mode, such as thrusting, is applied to the currently displayed timestep by restarting the simulation engine in an appropriate previous state.

The other approach is to look toward a higher-bandwidth connection between the simulation and the rendering engine. We anticipate installation of a direct HIPPI connection (shown as a dashed line in Figure 1) between the Maspar and Pixel-Planes 5 by the end of the year. This connection will provide eighty times the bandwidth of a standard ethernet. This application would be well suited to use such a connection. We could take advantage of this high bandwidth connection by moving all of the communications to that data path, but this would probably be unwise since it would require moving a large base of general purpose code to a specialized architecture. A more appropriate course of action would move the body update messages (which are already simply forwarded to the graphics processors by the front-end) and their acknowledgements to this connection, leaving the other control and editing messages on the host machines, which are well suited to the general purpose computations needed for formatting and sending these messages.

This prototype demonstrates the feasibility of distributed interactive applications with sizable bandwidth requirements. Although the suitability of ethernet is perhaps marginal for these

applications, higher bandwidth connections open the door to even more ambitious projects. The experience gained building the virtual world system described here should be of use in such efforts.

7. Acknowledgements

We would like to thank the Pixel-Planes Team and the Head Mounted Display Team for supplying us with the graphics hardware and software environment needed to implement this project. We would also like to thank Russ Taylor for providing and graciously supporting the communications and data conversion libraries which were used in implementing the protocol. We would also like to thank Jan Prins and Jeff Butterworth, our clients for the software engineering class from which this project grew. Finally, we would like to thank David Harrison, Claire Gingell, John McHugh, Greg Turk, Terry Yoo and the review team from the software engineering course for their support and comments.

References

[ANSI90] ANSI Standard for Information System, "HIPPI Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH)", X3T9.3/88-023 Rev 7.2, August 2, 1990

[BDH+92] J. Butterworth, A. Davidson, S. Hench, and M. Olano, "3DM: A Three Dimensional Modeler Using a Head-Mounted Display", *Proceedings of the 1992 Symposium on Interactive Computer Graphics*, Cambridge, Massachusetts, pp. 135-138.

[BOB+90] F. P. Brooks, Jr., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick, "Project GROPE–Haptic Displays for Scientific Visualization", *Computer Graphics*, Vol. 24, No. 4, 1990, pp. 177-185.

[Com91] D. E. Comer, *Internetworking with TCP/IP*, Vol. I, 2nd edition, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[DEM+92a] A. Duggan, E. Erikson, C. Mueller, M. Parris, Q. Zhou, "An N-body Simulation in a Virtual Universe: User's Manual", Technical Report, Univ. of North Carolina, Chapel Hill, 1992.

[DEM+92b] A. Duggan, E. Erikson, C. Mueller, M. Parris, Q. Zhou, "An N-body Simulation in a Virtual Universe: Implementation Manual", Technical Report, Univ. of North Carolina, Chapel Hill, 1992.

[FPE+89] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, L Israel, "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories", *Computer Graphics*, Vol. 23, No. 3, 1989, pp. 79-88.

[Gre90] L. Greengard, "The Numerical Solution of the *N*-Body Problem", *Computers in Physics*, Mar. 1990, pp 142-152.

[STB91] R. Singh, S. Tell, and D. Becker, "VISTAnet Network Interface Unit: Prototype System Specification", Technical Report (TR91-017), Univ. of North Carolina, Chapel Hill