Adapting a Collaborative, Force-Feedback, Graphical User Interface to Best-Effort Networks

by
Thomas C. Hudson

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2004

Approved by

Advisor: Russell M. Taylor II

Advisor: Kevin Jeffay

Reader: Mary C. Whitton

Gary Bishop

F. Donelson Smith

Richard Superfine

ABSTRACT

THOMAS C. HUDSON: Adapting a Collaborative, Force-Feedback, Graphical User Interface to Best-Effort Networks.
(Under the direction of Russell M. Taylor II and Kevin Jeffay.)

Latency is an unavoidable fact of distributed systems, and an unrelenting foe of interface usability. I present methods for lessening the impact of latency on distributed haptic, graphic, and collaborative interfaces. These three interfaces are present in the distributed nanoManipulator, a shared tool for remote operation of Atomic Force Microscopes. The work is carried out in the context of the Internet, where best-effort service means that network performance is not guaranteed and that applications must function under a wide range of conditions.

The ability of a distributed control algorithm to tolerate latency is innately tied up with how data or operations in the algorithm are represented for transmission over the network. I introduce two new representations for haptics, the warped plane approximation and local environment sampling, with superior latency tolerance. I show how image-based rendering is a useful representation for transferring the output of a remote graphics engine across the network, yielding a tenfold reduction in mean response time over video-based approaches, and how optimistic concurrency control is a useful representation for transmitting the actions of a remote collaborator across the network, requiring 85% less concurrency control overhead than pessimistic approaches. All of these intermediate representations convey a meaning that is not just true at a single point space and a single point in time, but that is valid across a volume of space or across a range of times. These higher-order representations reduce both the amount of blocking necessary and the rate at which closed feedback loops must run in an algorithm.

I show how techniques used to adaptively deliver multimedia content – the User Datagram Protocol (UDP), Forward Error Correction (FEC), and Queue Monitoring (QM) – are applicable to the streams of data transmitted by force feedback teleoperators. UDP alone yields a 35% reduction in latency and a 75% reduction in jitter.

The new algorithms and adaptations for the distributed nanoManipulator's interfaces combined to make this collaborative tool feasible, leading to successful use by

experimenters more than 2000 miles apart.

For Challe.

Acknowledgments

This work was funded by the National Institutes of Health National Center for Research Resources (award RR02170), the National Science Foundation (grants CDA-9724662, ITR-0082870, and ITR-0082866), Intel Corporation, the North Carolina Networking Initiative, and the Board of Governors of the University of North Carolina. The University of North Carolina at Wilmington employed me as an Instructor during the latter years of this work.

The MIT Press and my coauthors kindly gave me permission to include portions of “Managing Collaboration in the nanoManipulator,” first published in *Presence: Teleoperators and Virtual Environments* 13(2), in Chapter 7.

While I wrote this dissertation, my advisors were Russ Taylor and Kevin Jeffay. Gary Bishop, Don Smith, Rich Superfine, and Mary Whitton served on my committee. Challe Hudson provided invaluable editorial and emotional support, and was responsible for many figures. Jason Smith and Alexandra Bokinsky provided valuable comments on partial drafts. Sharif Razzaque provided ideas for data analysis. Diane Sonnenwald, Mary Whitton, Kelly Maglaughlin, Eileen Kupstas Soo, and Ron Bergquist did much of the requirements development and interface design for and evaluation of the collaborative implementation. Members of the nanoManipulator (nM) Computer Science team helped with code and general support, especially Aron Helser and David Marshburn, while the nM science groups gave me a reason to build good tools. Hans Weber wrote clock synchronization code for the VRPN library; many other members of the department also contributed to this shared codebase. Dorian Miller helped me formulate the distinction between external and internal response time measures. Gene Tagliarini provided a sounding board when my committee was 150 miles away. Fred Brooks taught me helpful lessons about research, project management, and an academic career. Russ and Shelly Taylor, Andy and Cat Wilson, Byron Woods, Nick England and Mary Whitton, Ben and Conni Elgin, Paul and Kathryn Baerman, and Tanner and Janell Lovelace all provided me places to stay on my visits to Chapel Hill to wrap things up. Without the goodwill and encouragement of many members of the UNC Chapel Hill and UNC Wilmington Computer Science departments, my friends, and my family, I would not have persevered.

Contents

List of Tables

List of Figures

List of Abbreviations

ACE	ADAPTIVE Communication Environment
ACM	Association for Computing Machinery
AFM	Atomic Force Microscope
ARQ	Automatic Repeat Request
CVE	Collaborative Virtual Environment
DiffServ	Differentiated Services
DFF	Direct Force Feedback
DIS	Distributed Interactive Simulation
DOF	Degree of Freedom
DVE	Distributed Virtual Environment
FEC	Forward Error Correction
HeiTS	Heidelberg Transport System
IBR	Image-based rendering
IntServ	Integrated Services
IP	Internet Protocol
JND	Just-Noticeable Difference
LAN	Local Area Network
LEP	Local Environment Prefetching
MPEG	Motion Picture Experts Group
MVC	Model View Controller
NIH	National Institutes of Health
NCRR	National Center for Research Resources
NTP	Network Time Protocol
OSU	Ohio State University
OSI	Open Systems Interconnection
Qbone	Quality of Service Test Backbone
QM	Queue Monitoring
QoS	Quality of Service
RMS	Root Mean Square
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SEM	Scanning Electron Microscope

SGI	Silicon Graphics, Incorporated
SIGGRAPH	Special Interest Group on Graphics
SPM	Scanning Probe Microscope
STM	Scanning Tunneling Microscope
TCP	Transmission Control Protocol
TEP	Total Environment Prefetching
UDP	User Datagram Protocol
UNC	University of North Carolina at Chapel Hill
USC	University of Southern California
VE	Virtual Environment
VRPN	Virtual Reality Peripheral Network
WAN	Wide-Area Network

Chapter 1

Introduction

Many interesting applications have **feedback-critical user interfaces**. Response to user input must be "immediate" – subject to imperceptible delay – for the application to be effective. This immediate response to the user is necessary because these applications tie into the user's unconscious sensory-motor reflexes, such as proprioception or parallax. Feedback-critical user interfaces of interest today include haptic force feedback and head-tracked virtual environments.

At first glance, distributed feedback-critical user interfaces require a high level of service from the network: minimum bandwidth, bounded latency, bounded jitter, and bounded loss. Today's Internet guarantees reliability, but none of these other aspects of service. A great deal of effort has been expended to devise Quality of Service (QoS) guarantees for the Internet to support this class of applications, but no QoS implementation has been deployed (section 3.2).

An alternate approach to distributed feedback-critical user interfaces is to take the best-effort nature of the Internet as a given. Instead of trying to change the network, applications can be designed to adapt to whatever network conditions they encounter (Huitema 1996). Distributed algorithms can vary their use of the network depending on currently available network resources. Further, an application with several algorithms implemented to carry out a single task, each of which may provide a different interface to the user and require different performance from the network, can change between algorithms to suit the current performance provided by the network.

This dissertation explores adaptive and latency-tolerant algorithms for haptics, graphics, and collaboration through reports of user experience, experimental evaluation, new analysis techniques, and new algorithms. Adaptation and latency-tolerance provide effective means of building interactive distributed systems that would otherwise be constrained or rendered unusable by network delay.

The work was grounded in a single large project: the extension of the nanoManipulator into a distributed collaborative system. The nanoManipulator provides interactive 3-D visualization of data from a Scanning Probe Microscope (SPM) and gives a user force-feedback control of the microscope's tip (Taylor 1994). The distributed version enables collaborative experimentation and manipulation of microscope specimens by scientists hundreds of miles apart. This project is discussed in chapter 2.

1.1 Motivation

Four parameters are commonly used to characterize the performance of a network connection. The number of bytes that can be transmitted between two endpoints on the connection in a period of time is the connection's achievable **bandwidth**. The time (measured with a global clock) from the transmission of a packet to its receipt at the other endpoint is the network's **latency**. Latency can vary significantly over the life of a connection, and this variation – **jitter** – plays an important role in the suitability of a network connection for some applications. Finally, some packets sent are never received; **loss** is the rate at which this occurs. (Most protocols also attempt to detect data corruption and throw out corrupted packets, transforming corruption into loss.)

Groups working on virtual environments and other distributed interactive applications have stated that their work requires the network to provide some guarantees of QoS (Macedonia et al. 1994; Leigh et al. 1999; Greenhalgh et al. 1999). Many approaches to guaranteeing QoS have been proposed, each in some way guaranteeing network performance to applications. This guarantee is typically expressed as a minimum (or fixed) bandwidth, a maximum (or fixed) latency, a maximum jitter, and a maximum loss rate. Ferrari (1990) surveyed variations in these parameters and other useful network properties to define a design space for QoS schemes.

For the nanoManipulator, as for most interactive applications (Cheshire 1996), the critical problem is latency. The bandwidth required for distributed operation is modest, but humans are very sensitive to latency, which interferes with their work (Sheridan 1993; Wloka 1995). In a feedback control system, latency is a particularly severe problem, potentially causing system instability. Loss of messages transmitted over the network is also a problem; the standard Internet transport protocol Transmission Control Protocol (TCP) provides a reliable (apparently lossless) service but does so by dramatically increasing latency and jitter when loss occurs. Similarly, most approaches to reducing jitter do so by artificially increasing latency by introducing a buffer to smooth the arrival of data. The boundaries of human tolerance of loss and latency depend on the mode of interaction and the task attempted, and are for the most part not known exactly; approximate values from the current literature are given in subsequent chapters (4, 7, and 8).

A complete guarantee of network performance, such as that envisioned by the Integrated Services (IntServ) architecture for the Internet (chapter 3), will not necessarily support usable interactive applications. There is a lower bound below which any QoS approach cannot reduce the latency. This lower bound is dependent on such physical characteristics of the network as propagation delay. Many interactive applications have a “breakdown latency”; latency in excess of this limit makes the application unusable. If QoS approaches cannot guarantee a latency below the upper limit that the application can tolerate, the application will not be usable. Even latency below this limit can significantly impair interaction. Latency tolerance varies widely with the nature of the application; a survey of the literature is presented in chapter 3, and specific cases are examined in later chapters.

Our informal attempts at running the nanoManipulator across a wide-area network revealed that the standard network pathologies – loss, delay, and jitter – were bad enough to make the application unusable. A first trial, with the microscope at University of North Carolina at Chapel Hill (UNC) in Chapel Hill, NC and the interface at the ACM SIGGRAPH conference in Los Angeles, CA during late summer of 1997, showed us just how bad this could be. Loss on the commodity Internet was high enough that using User Datagram Protocol (UDP) as a transport protocol caused unacceptably poor performance (figure 1.1), while the error-recovery mechanisms in TCP caused unacceptably high latency and jitter. Something better than the best-effort Internet was needed to support this application across long distances.

We obtained access to the Internet2 Quality of Service Test Backbone (Qbone) for a trial from UNC to the Highway1 site in Washington, DC in the spring of 1999. The Qbone is a testbed for QoS mechanisms, particularly the Differentiated Services (DiffServ) architecture, which is discussed in chapter 3 (Teitelbaum et al. 1999). The Qbone could give us Premium service, the highest-quality form of DiffServ, which yielded a network connection with no loss, ample bandwidth, and roughly 30 ms of round-trip network latency – good operating conditions for our application. When the nanoManipulator did not receive Premium service, but had to compete with a high-bandwidth application for the same available network bandwidth, our application

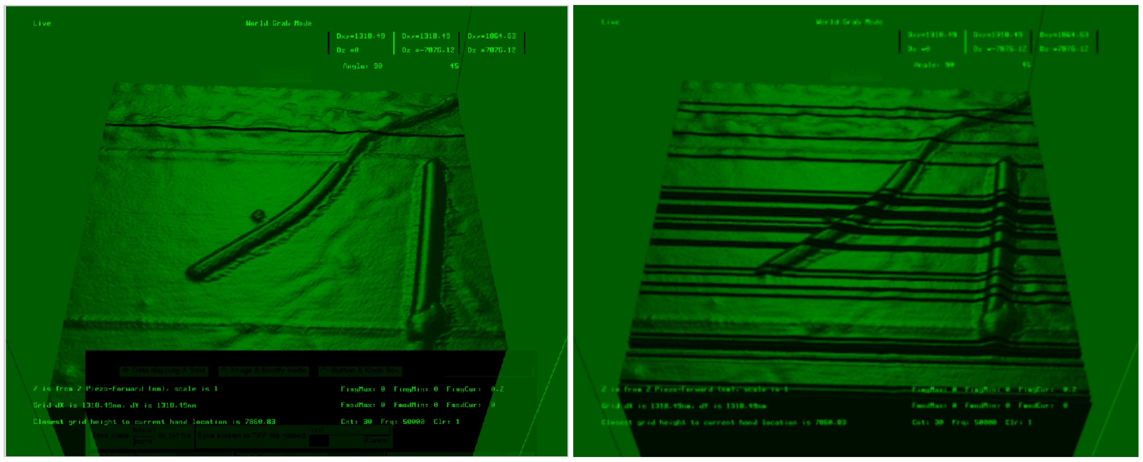

Figure 1.1: Two "frames" of surface data from the nanoManipulator in Los Angeles, California controlling a microscope in Chapel Hill, North Carolina. Horizontal black lines in the right-hand image of the surface are caused by data lost during the transcontinental Internet transmission by the UDP protocol.

became unusable due to latency, loss, and (loss-induced) jitter.

Although this Chapel Hill to Washington, DC connection was a demonstration for an Internet2 meeting and was strongly supported by a network vendor who wanted to use the distributed nanoManipulator to showcase their DiffServ support, the administrative and technical difficulty of establishing this Qbone connection was extremely high. We began planning and coordinating with the network service providers several months in advance, scheduling the Qbone connection to be operational one week before the conference for testing. Due to incompatibilities among implementations of the standard and miscommunications between service providers, our connection did not work until late the night before it was needed – and that it functioned at all was due to the heroic efforts of one of the vendor's network engineers. As of this writing, DiffServ appears to be no closer to wide deployment or effective standardization. Something easier and cheaper than guaranteed QoS was needed to support our application.

In 2000 and 2001 we used the Internet2 for test deployments of the interface to Ohio State University (OSU) in Columbus, Ohio and to Microsoft Research in Seattle, Washington, keeping the microscope in Chapel Hill. Although for this test the Internet2 did not provide QoS guarantees, it did give us our next-best case: a lightly-loaded, high-bandwidth network. Round-trip network latency to OSU was in the 30-40 ms range, and again our application was usable. Round-trip network latency to Microsoft Research was higher, averaging 70 ms, and the force-feedback was not sufficiently high-quality for manipulation of samples in the microscope. This implied that, although guaranteed QoS may have been sufficient to allow remote operation of the nanoManipulator from a site roughly 400 miles distant, it was insufficient for a site nearly 3000 miles away.[1,2] Something more effective than QoS methods was needed to support our application.

The nanoManipulator project has had a private network connecting the Physics and Computer Science buildings since soon after the project's inception. This network was required because our public network at the time was prone to bursts of congestion that seriously impaired experiments. When deploying the distributed nanoManipulator, we built another private network connecting the originating departments to our collaborators on-campus (Chemistry and Gene Therapy) and in the metropolitan area (NIEHS, the National Institute of Environmental Health Sciences). However, expanding this private network any farther would have been prohibitively expensive and logistically difficult. Something more widely accessible than a private network was needed to support our application.

In general, neither building a dedicated network nor gaining access to QoS facilities is feasible for most collaborations. To reach users at these other sites, we must be able to deploy the application over today's commodity Internet. In the original design, all of the nanoManipulator's interfaces had more stringent latency requirements than could be satisfied by the Internet, and remote rendering consumed very high

[1]There was no guaranteed QoS offered by the connection to OSU: however, this lightly-loaded, high-bandwidth network gave us performance comparable to what we would expect from a guaranteed, dedicated circuit. Thus we conclude that QoS methods may be able to solve the problem of collaborating with OSU.

[2]The distances I describe above are geographic straight-line distances. Messages sent over the Internet may travel significantly longer distances. For example, a packet from UNC to OSU passes through Washington, DC and New York City on its way. For terrestrial high-speed networks most delay occurs at routers, so the number of routers between two points can also serve as a useful distance measure. From UNC to OSU a packet may pass through a dozen routers, while to reach Microsoft Research it transits 18 or more. This is a significant difference, but much less than the geographic distances would lead one to expect, and helps explain why the delay only doubled when the geographic distance increased more than sixfold.

The Internet2 provides unusually direct connections between its sites. For messages to go from the nanoManipulator installation at Arizona State University to community colleges in Arizona over the regular Internet requires that they transit more than 14 routers, even though this is a relatively short geographic distance (Razdan et al. 2000).

bandwidth. (The calculations behind this bound are given in Appendix A). Thus, changing the application to deal with a high-latency, moderate-bandwidth environment was necessary.

1.2 Thesis Statement

Feedback-critical user interfaces must meet stringent requirements for their response time to user input. Network latency can easily increase response time beyond an acceptable level, rendering distributed applications unusable. Quality of Service guarantees cannot reduce latency below a hard limit, and that limit is too high for many distributed applications. Further, many QoS techniques, seeking to control loss or jitter, increase network latency. Thus, my thesis:

> Interactive applications can be successfully distributed across wide-area networks by selecting latency-tolerant operation semantics and intermediate representations to match the application's requirements and current network conditions. Applying these design approaches to the distributed nanoManipulator increases user satisfaction while reducing system response time and haptic display error.

I have combined and extended adaptive and latency-tolerant approaches to build a distributed application with three types of remote interaction, each with different requirements: teleoperation with force feedback, collaboration, and remotely rendered interactive graphics. These modes of interaction share one common trait that explains their high sensitivity to latency: they are all part of distributed control and feedback loops.

The two methods of latency tolerance used in this dissertation are **intermediate representations** and **operation semantics**. An intermediate representation is a representation of some data chosen for its suitability for a distributed application, and is not the "native" form in which the data was generated or will be displayed. By operation semantics I mean the precise definition of an action and the representation of that action *that will be transmitted across the network*. By making small changes to the definitions of operations afforded by the user interface we can make significant changes to the network performance of the distributed application, often without changing the user's perception of those operations.

1.3 Results

This dissertation presents five results: three in haptics, one in collaboration, and one in remote rendering.

- A novel metric for measuring the performance of force feedback systems (chapter 4)
- Two novel intermediate representations for networked force feedback (chapter 5)
- Evidence of the usefulness of network adaptations designed for multimedia applied to distributed force feedback systems (chapter 6)
- Evidence of the usefulness of particular operational semantics and architectural models for constructing networked collaborative applications (chapter 7)
- Measurements of the network performance of image-based rendering for remote rendering (chapter 8)

There are several distinct types of data transmitted across the network by the distributed nanoManipulator. Each has different characteristics and different requirements, so each must be considered separately when building an adaptive, latency-tolerant application. I have implemented adaptations for the three principal streams: haptics, concurrency control for collaboration, and interactive graphics, discussed in chapters 4-6, 7, and 8. First, in chapters 2 and 3, I introduce the nanoManipulator and examine other researchers' work in feedback-critical user interfaces.

Chapter 2

Background

In this chapter I discuss the original nanoManipulator, our goals for a distributed nanoManipulator, and early experiences in our attempt to deploy a distributed system. These experiences led to the thesis and experiments documented in this dissertation. In these experiences, across all of the nanoManipulator's interfaces, the chief problem was latency. I conclude this chapter with a discussion of models commonly used in the study of latency.

2.1 Scanning Probe Microscopes

Scanning Probe Microscopes (SPMs) allow scientists to see and manipulate nanometer-scale objects. These objects range from biological specimens, like viruses and DNA, to substances of interest to materials scientists, such as carbon nanotubes. We refer to whatever small piece of material is being imaged by a microscope as a **specimen**.

The current generation of the nanoManipulator system is built around an Atomic Force Microscope (AFM), one particular variety of SPM. Every experiment discussed in this dissertation was conducted with an AFM. Figure 2.1 shows a schematic diagram of an AFM: a stage that holds the specimen, a tip that probes the specimen, and a laser that measures the position of the tip The **stage** is a surface on which specimens are mounted, supported by a **scanner** – typically a hollow cylinder of piezoelectric crystal. Because piezoelectric materials change their size in response to electric voltage, the scanner (and thus the specimen) can be made to move small distances in any direction in response to precisely applied voltages. The **tip** is a microscopic, extremely sharp point projecting from the end of a small springy cantilever, both typically made of silicon.[1] The laser is aimed at the cantilever, and reflected onto a photodiode. Output from the photodiode can be interpreted to measure the bend and twist of the cantilever. Finally, the entire system is driven by circuitry that controls the voltage applied to the piezoelectric crystal based on the photodiode's output.

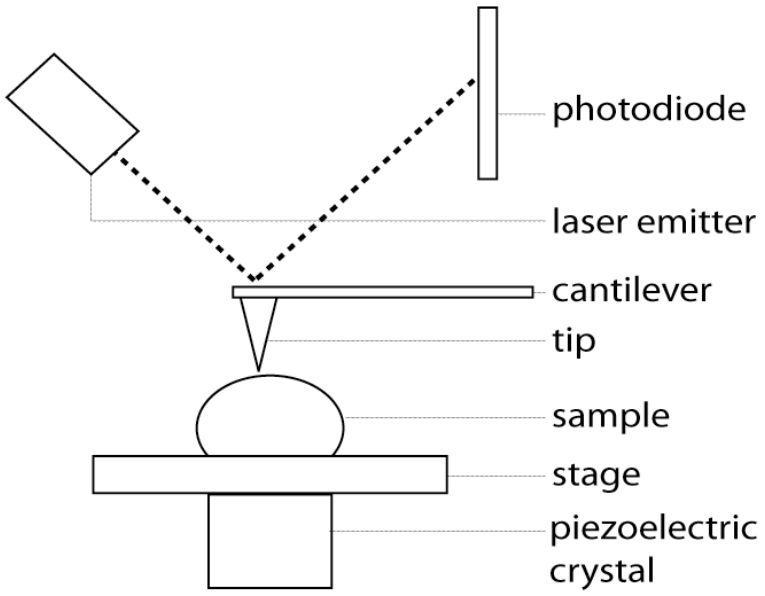

Figure 2.1: Schematic of an Atomic Force Microscope, showing the stage with its piezoelectric crystal, the tip, the laser, and the photodiode.

Typically, SPMs are operated under closed-loop (feedback) control.[2] The SPM controller moves the tip until it is very close to the specimen and measures the interaction between the tip and the specimen. The primary difference between different types of SPM is the choice of which interaction between tip and specimen to measure. For example, AFMs move the tip until it is in contact with the specimen and measure the vertical force exerted on the tip by the contact,[3] while Scanning Tunneling

[1]Tips are commonly manufactured as cones or square-based pyramids ending with a 10 nm radius of curvature.

[2]Feedback control systems attempt to maintain some specified output. To compensate for any errors, they measure the actual output and "feed back" the measured error into the system as a compensating factor (Nise 1995).

[3]This is a simplification of the physics of the interaction between tip and specimen. For more

Microscope (STM)s, another variety of SPM, apply a voltage to the tip, and measure the electric current between the tip and the specimen. Electrons "tunnel" through short distances, so an STM need not contact the specimen as does an AFM.

The value of the SPM is not only in sensing characteristics of the sample at a single point, but in building up an image of a large area of the sample. The tip is moved back and forth in a raster pattern through a horizontal plane (Figure 2.2), while the controller varies the vertical position of the specimen in order to keep the measured value constant. [4] This gives a three-dimensional isovalue map over the surface - for an AFM, isoforce.

Besides the primary signal – conductivity for an STM, force for an AFM– used to determine the height of the specimen, several additional signals may be monitored at the same time. With an AFM, one of the most important is the lateral (horizontal) force that twists the tip when it encounters obstacles while it images the sample by moving in a raster pattern.

2.2 The nanoManipulator

The nanoManipulator is an interface to SPMs that includes three-dimensional graphics and haptic force feedback. It was developed to help scientists increase their understanding of the data from these microscopes mid-experiment, enabling them to steer the experiment to respond to data as they gather it (Taylor 1994). For example, scientists using the nanoManipulator observed that carbon nanotubes that seemed to be moving randomly when pushed around on a graphite substrate were actually preferring certain regular orientations regardless of their position on the substrate. The nanoManipulator allowed them to change their experiment plan mid-stride and discover a useful nanoscale phenomenon: predictable gear-like interlock between the nanotubes and the regularly-spaced carbon atoms of the graphite substrate (Falvo

detail, consult Howland and Benatar (1997).

[4]Here, the output of the control system is the measured interaction between tip and specimen, and the error signal is the difference between the output signal and the desired output (a constant specified by the scientist, taking into account the material properties of the specimen). If the error signal is positive, the specimen is lowered (voltage on the scanner in the Z dimension is reduced) until the desired output is reached; if the error signal is negative, the specimen is raised. This control loop needs to execute tens of thousands of times per second. If the measured interaction is force, as in an AFM, the tip in effect measures the height of the specimen.

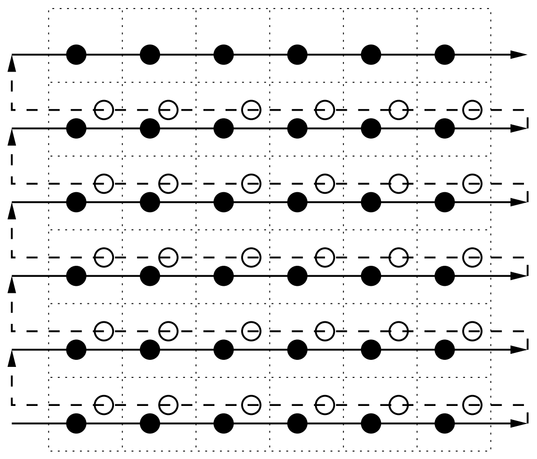

Figure 2.2: Illustration of the scanning pattern of Atomic Force Microscope. Each solid dot represents a sample taken during a left-to-right movement of the tip (solid lines); each hollow dot a sample from the right-to-left pass (dashed lines). The regular grid of dotted lines is a reference to show the intended correspondence between the two raster patterns. Nonlinearities in the response of the stage to voltage applied by the control circuitry cause samples taken while the tip is moving in different directions to not line up.

et al. 2000). The nanoManipulator also provides scientists with the ability to exercise interactive control of the microscope tip during experiments, putting a human "into the loop" and enabling new types of experiments. For example, using the nanoManipulator, scientists were able to precisely manipulate virus particles in liquid more accurately by hand than was possible using pre-programmed positioning of the microscope (Falvo et al. 1997). The scientist controlling the force feedback interface "feels" the surface scaled up a million times, and makes use of their proprioception and their reflexes in guiding the tip.

The nanoManipulator application records all the data received from the microscope during an experiment. The nanoManipulator can replay this recorded data as if it were coming from the microscope again, effectively simulating the AFM and allowing the scientist to "replay" the experiment. This feature allows the complete history of an experiment to be shown to scientists who were not present when the data was initially taken, and allows reanalysis of old data using new techniques. Replay has resulted in the discovery of new facts from old datasets, such as the discovery of periodic graphite fault planes projecting from an apparently-noisy surface (Taylor et al. 1993). All of these capabilities – leveraging proprioception and reflexes, mid-experiment understanding and changes of plans, replay, and reanalysis – have helped suggest follow-up experiments and shape the path of research (Taylor et al. 1997).

The nanoManipulator engages scientists using two senses: sight and touch. It uses interactive 3-D graphics to show users the shape of the surface being imaged by the microscope. Scientists can look at the surface from different directions, exaggerate the Z scale, change coloring, superimpose regular grids, make measurements, and apply numerous other analysis techniques while the experiment is in progress.

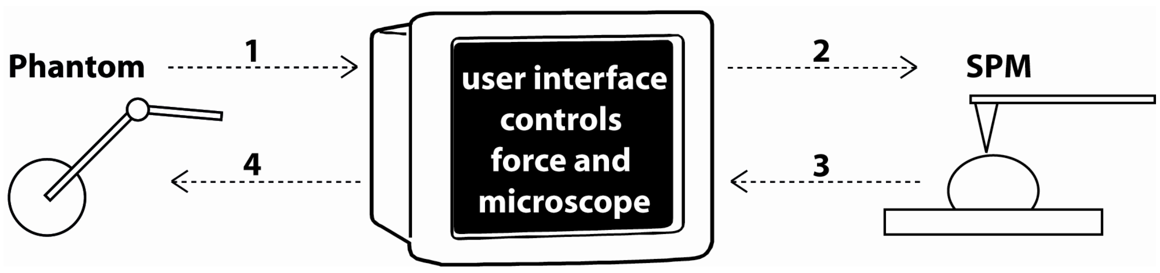

Figure 2.3: Conceptual feedback loop between a force-feedback haptic device and a Scanning Probe Microscope. (1) position of user's hand is measured, (2) movement commands are sent to the microscope and it moves, (3) height of sample is measured at the new location, (4) force is applied to the user.

Touch is provided by a SensAble(TM) Technologies PHANTOM(TM) (Salisbury et al. 1995), which lets the user direct the tip of the microscope and feel the shape of the surface. A PHANTOM is a robotic arm with a pen-like stylus on the end; the arm can sense the position and orientation of the stylus.[5] A PHANTOM can also be made to exert force in three dimensions to change the position of the pen.

A scientist using the nanoManipulator works with this stylus in his hand. The computer can sense where he positions the stylus. The tip of the microscope is coupled to the PHANTOM by the software, so that every movement of the scientist's hand is imitated by the microscope's tip on a scale roughly one million times smaller.[6]

[5]In the lexicon of virtual reality, this is a six degree-of-freedom, or 6DOF, tracker, with three dimensions of force display.

[6]The tip normally remains "in feedback," using the microscope's circuitry to exert a constant force on the specimen, mirroring the scientist's hand motions in the horizontal plane but moving

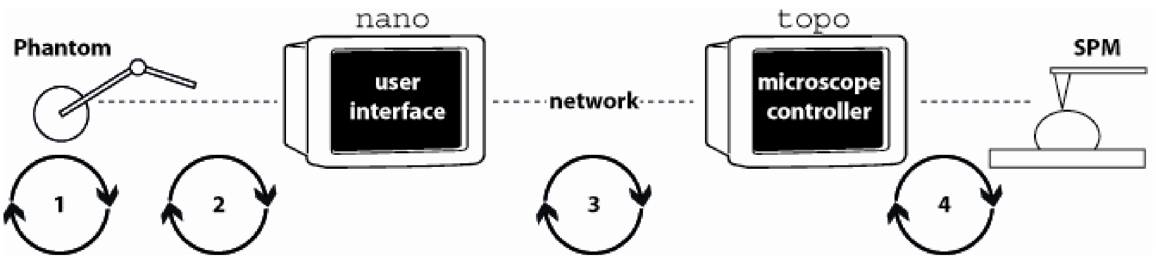

Figure 2.4: Series of linked feedback loops between a haptic device and a Scanning Probe Microscope. (1) Hardware inside the Phantom measures the position of the end effector and updates the output force at 1000 Hz. (2) The Phantom sends the current position to `nano` at 60 Hz; `nano` sends new information about the sample at frame rate. (3) `nano` sends microscope positioning commands to `topo` at frame rate; `topo` sends microscope position information to `nano` as movements complete (20-200 Hz). (4) `topo` interacts with the hardware on the SPM.

The shape of the surface measured by the tip is turned into forces that are exerted by the PHANTOM against the scientist's hand, so that he feels as if the probe in his hand is really in contact with the surface scaled up by a like order of magnitude (Figure 2.3).[7] Implemented directly, this would **position-control force feedback**: the user controls the position of a sensor, and the system response is expressed as forces on the user.[8] It is possible to build a force feedback device using the dual, equivalent formulation of **force-control**: the manipulator measures force exerted by the user, and the system response is expressed by moving the manipulator to a resultant position. The nanoManipulator's PHANTOM provides position control through an intermediate representation, the plane approximation, such that the force exerted on the user is not a direct function of the force measured by the microscope (section 4.4.2). Although Figure 2.3 showed a single conceptual feedback loop, there

independently vertically and transmitting representative forces back to the scientist's hand. Direct 3-dimensional control of the microscope tip by the PHANTOM is occasionally used for precise manipulations of the specimen. This 3D control was not used in the work described in this dissertation.

[7]Scaling depends on the degree of zoom the scientist chooses, which varies with the specimen and with his intent, typically from ten thousand to one million times: magnifying a region of the specimen from 100 to 10,000 nm on a side to an apparent size of 10 cm square.

[8]In the lexicon of control theory, position is the controlled quantity and force is the feedback signal.

are actually several loops, all running at different rates, as shown in Figure 2.4.

2.2.1 Issues of Control in Scanning Probe Microscopes

There are many sources of noise and error in an SPM. Three that are significant for the nanoManipulator are nonlinearity, hysteresis, and creep. The motion of the scanner is inherently nonlinear: the distance it moves is not a linear function of the voltage applied, and the deviation can be as much as 25% from linear. The scanner's motion also displays hysteresis, a nonlinearity proportional to the sign of the derivative of the voltage: the position of a scanner with x volts applied when the voltage is increasing may differ from the position of that scanner with x volts applied when the voltage is decreasing by as much as 20% of the total distance the scanner can move.[9] Finally, the scanner's response to a constant voltage is not constant: after rapidly assuming an initial position (within 1 millisecond of a voltage being applied), over the next 10 to 100 seconds a scanner may creep as much as 20% of the distance moved in that first millisecond (Leckenby 2000), exponentially approaching some rest state.

Due to the nonlinear response of the piezoelectric crystals used in SPM stages, adding human control and force-feedback gives the scientist more precise control of manipulations of the specimen than are achievable by conventional programmed control (Taylor and Superfine 1999). The behavior of piezoelectric stages is not consistent between microscopes' scanning modes (moving back and forth at relatively high, constant speeds) and the slower, unstructured movements used for manipulation. This results in unavoidable error. What the user "sees" in the three-dimensional visualization of the microscope's previous scan data does not match what the microscope tip will encounter when it makes a modification: the map is not the territory. A modification based on the "map" – the topography of the surface reported by even the most recent microscope scan – will not execute where intended, due to hysteresis and creep. However, as a user prepares to make a modification, he can feel where the tip is on the surface and compensate for those nonlinearities. When an SPM is to be used for manipulation, the visualization of the scan lies: the haptics display the truth.

Direct human control of the microscope tip has consistently enabled more complex

[9]Although the manufacturer reports this as the worst-case hysteresis, we have never observed a hysteresis magnitude of more than 20% of the size of the current scan region, which is typically much less than the possible scan size.

experiments to be performed than are possible under automatic control. For example, Finch et al. (1995) were able to use the nanoManipulator to push a gold colloid across the specimen substrate until it filled a gap in a wire. The first programmed robotic control of a similar task was carried out five years later by Resch et al. (2000) and Meltzer et al. (2001).

2.3 The Distributed nanoManipulator

The aim of the distributed nanoManipulator system is to help scientists carry out their tasks without regard to the distance between scientist and microscope. One early goal was to provide remote access to scientific equipment: SPMs are expensive and uncommon, and the three-dimensional graphics display of the nanoManipulator requires computational power that until recently was not found in most workstations. The distributed nanoManipulator helps scientists use SPMs and graphics hardware not present in their labs. Even more important than access to remote equipment is access to remote expertise: planning an SPM experiment, preparing a specimen for the microscope, operating the microscope, and interpreting the data all require expertise and intellectual understanding of a physical regime that is not part of the background of most scientists. Typically, a domain expert – perhaps a physicist or a biochemist – who wants to do SPM microscopy on a new type of specimen will require many months of experience to reliably find the correct specimen preparation method, microscope settings and imaging mode, and other techniques necessary to get good data. Giving these domain experts access to an SPM technologist greatly eases their learning curve.

After an initial implementation involving the Chemistry Department of the University of California at Los Angeles, the nanoManipulator was further developed by the Department of Computer Science and the Department of Physics and Astronomy at UNC. The two departments are located in adjacent buildings; collaboration was straightforward, and a dedicated computer network to serve the project was created.

By the late 1990s, the audience for the nanoManipulator had grown significantly. It was used for research by UNC's Departments of Chemistry and Gene Therapy and in ongoing collaborations with other labs ten to twenty miles from campus. nanoManipulators were installed in the Chemistry Department of the Catholic University of Leuven (Leuven, Belgium) and the Computer Science Department of Arizona State University (Phoenix, AZ). The National Institutes of Health (NIH) National Center for Research Resources (NCRR) supported a postdoctoral researcher at UNC to bring in scientists from across the United States to learn to apply atomic force microscopy and the nanoManipulator to problems in biology. The UNC Department of Education had taken the nanoManipulator into local high schools to enrich science education. Demand for copies of the device was high enough that a company was formed to transform the nanoManipulator into a commercial product.

This increased use of the nanoManipulator revealed two problems. First, those sites that installed their own copy of the system needed support from UNC Computer Science to set it up and to add the features they required. After the installation at Leuven, one or two graduate students per summer went to Belgium to work on Leuven's copy of the code. As other sites requested nanoManipulators of their own, the project realized that it would soon run out of graduate students! Commercialization of the system was intended to offload this support onto an external organization.

Second, collaborators from across town and the scientists brought in by the NIH were not accomplishing as much as we, or they, hoped. Bringing in an academic scientist from a remote site is difficult. Schedules must be aligned so that they can visit at a time assistants and instruments are available. Airfare and accommodation must be arranged. During their visit, they typically must understand how our AFM works, discover how to prepare their specimens for imaging under an AFM, carry out a preliminary experiment, analyze that experiment, and then use their preliminary results to plan and carry out a follow-on experiment. The press of commitments at visitors' home institutions limits their ability to stay, typically to one week or less. We repeatedly found that in a single week we could press the process through a preliminary experiment, but could not do a significant portion of the analysis or plan any follow-on experiments. Despite the promising nature of the early results, our visitors usually could not arrange funding for a second visit, or had to wait many months to come again.

This kind of difficulty – experienced on a much smaller scale by our local collaborators – produced what we termed "peak and valley" collaborations, with small spurts of productivity separated by long periods of low activity (Sonnenwald et al. 2001). This process was frustrating for its participants, and did not produce nearly as many results as hoped. Results "in limbo" waiting until another visit for completion can become outdated, lose their relevance, or simply be abandoned in favor of non-collaborative projects on which steady progress is possible.

2.3.1 Goals

For all of our visitors and collaborators, the principal difficulties were scheduling and travel. By late 1997, we had decided to try to overcome these problems using the Internet. If we could use the public network to give remote users access to our instrument and our microscopy expertise, they could reduce their need to travel to our site and would be able to schedule their collaborations more flexibly.

Our first goal was to give remote users access to the microscope and to the nanoManipulator's key feature, the ability to have a human control specimen modifications using force-feedback cues. This is the interaction mode of the system most degraded by high network latency. It has been known since at least 1966 that force feedback suffers when transmissions between operator and machinery are delayed (Ferrell 1966), but the nanoManipulator is sufficiently different from previous work in teleoperation to present new problems and afford new solutions.

When we began work on the distributed nanoManipulator, the 3-dimensional visualization software used required a powerful SGI workstation. Graphics capabilities have increased since then; today, the newest PC-class computers and graphics hardware are capable of running the software (the nVidia Quadro2 Pro card could update a 300x300 scan of the specimen at 30 fps (Helser 2001); more recent graphics hardware has even higher performance). Some visualization techniques that our scientists want to adopt require significantly more computation to render each frame than the nanoManipulator's original modes (Weigle et al. 2000; Bokinsky 2003), so there is still a need to enable access to more graphics power than is present on any signle workstation.

Our rendering requirements encouraged us to give users access to a remote rendering engine – originally an SGI Infinite Reality, PixelFlow (Molnar et al. 1992), or similar special-purpose machine at a supercomputer center. Today a high-end PC or a PC cluster across a Local Area Network (LAN) can be used as the remote rendering engine. This process is another distributed feedback loop: the application's current state is sent to the rendering engine, which draws the current view and sends the image back across the network. Every time the application state changes, the user must wait to see a visual indication of the new state until the updated parameters have reached the server, a new image has been rendered, and the image has been received and displayed by the user's machine. When the application state is changing in response to user input, significant delay in this loop can make the application unusable (chapters 7 and 8). Compression techniques like MPEG (Mitchell et al. 1996) are commonly used to "stream" remote video and graphics data across networks, but these focus on reducing bandwidth requirements rather than latency. Depending on the network bandwidth available, compression can even increase latency due to encoding/decoding overhead.

By 1999, we had an additional goal: to enable two scientists to use the distributed nanoManipulator at one time. The NIH NCRR funded us to add this capability, to deploy software with the collaborative features to our collaborators, and to conduct both an ethnographic and an experimental study of the process of scientific collaboration and the utility of our tool (Sonnenwald et al. 2001; Sonnenwald et al. 2001). The NIH supports the development or purchase of scientific instruments in research labs across the US. The NIH NCRR funds scientists to travel to those labs to increase the utilization of these instruments, spread knowledge of their use, and encourage scientific collaboration. We based our design for the collaborative features on an ethnographic study conducted for the NIH NCRR, with hours of interviews with and observations of scientists leading to a prototype, usability reviews, and a formal evaluation experiment (Hudson et al. 2000). The study will continue into 2004 to evaluate the distributed nanoManipulator as it is used by scientists to do their work.

In summary, to make remote resources as useful as local resources, our original goals were to give a user access to:

- a remote microscope, with correct, complete data, a complete interface, and the ability to plan and execute manipulations of the specimen;
- a remote rendering engine, with a clear, accurate, fully-detailed visualization of the surface that responds to user input and updates at interactive rates;
- and a remote collaborator, both users having full access to the nanoManipulator functionality, the ability to work together or in parallel, awareness of one another's actions, and the ability to share data analysis and other external applications.

2.3.2 A System Architecture

To discuss the ideas I introduce in this dissertation, I make reference throughout to the architecture of the distributed nanoManipulator. I give a brief description of the architecture here, with more details, deployment diagrams, and implementation notes in Appendix B. The core components are shown in Figure 2.4.

There are two central processes: `nano`, the user interface, and `topo`, the microscope controller. These two processes run on machines connected by a network. There is also a process controlling the PHANTOM force feedback device; in our deployments, this runs on a second processor of the dual-processor computer hosting `nano`, although it could be hosted on another computer connected to `nano` by a LAN. Finally, there may be an additional process, `render`, responsible for rendering or pre-rendering the graphics to be displayed at `nano`; this may run in another thread on `nano`'s host, or on a graphics workstation or supercomputer across the LAN or Wide-Area Network (WAN).

2.4 Interactivity and Latency

This dissertation is about feedback-critical user interfaces. What is the "immediacy of response to user input" required by an application with this type of interface? Being feedback-critical is a particularly strong form of the common term "interactive"; in this section I consider the broad range of definitions of interactive already in the literature.

One conventional model of interaction used by researchers in human-computer interaction entails a series of four activities: thought and action by a human, then the resulting computation and response by the computer (Figure 2.5) (Bhola and Ahamad 1999). **Response time** is the time between the user's action and the user's perception of the response – the time elapsed during the computation and response stages.

Working with the nanoManipulator, we have observed an important distinction within user interfaces between continuous and discrete input. **Continuous input tasks** are tasks executed by a mouse, a Phantom, or other tracking device. Familiar examples include using a mouse to hilight several words in a text editor, move position clip art on the slide in presentation software, or select a file in a conventional windowing system and drag it into a new folder. **Discrete input tasks** entail typing short sequences on a keyboard or pressing buttons using a mouse. Feedback is much less important for discrete tasks than it is for continuous. This same distinction has been reported in the literature, e.g. by Shneiderman (1998).

Under the four-stage model of human-computer interaction, an application is "interactive" if the application's response time does not dominate the user's thought and action time. In continuous tasks, the user is continuously providing input, so

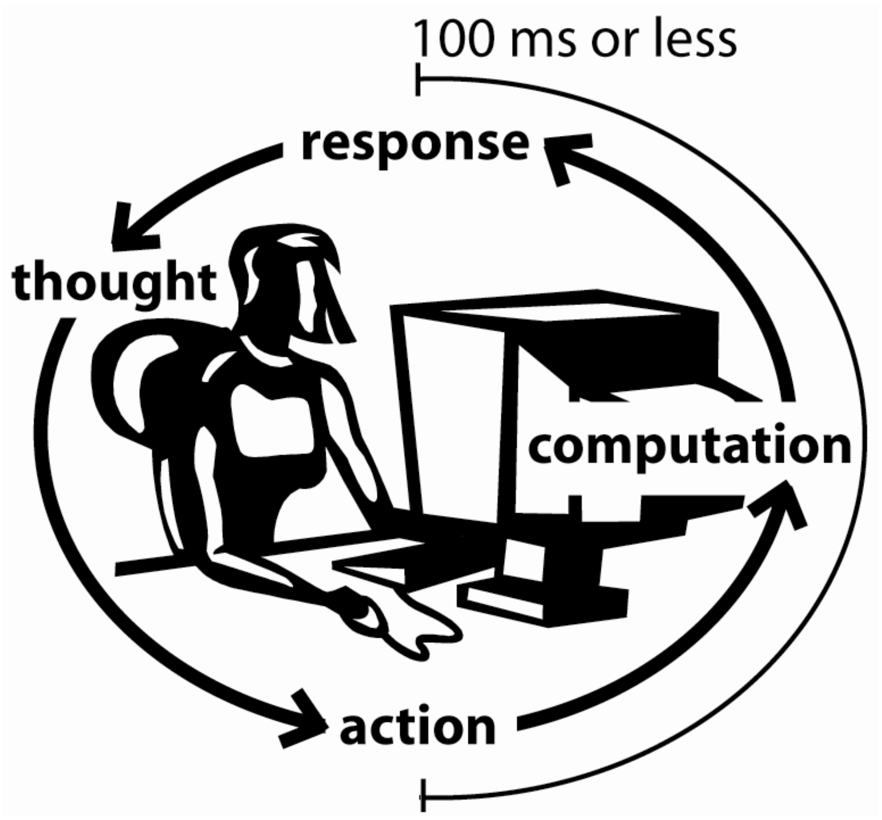

Figure 2.5: A simple model of a user interacting with a computer in a closed feedback loop: the user thinks, then takes an action that is interpreted by the system; the system makes some computation, then displays a response to be interpreted by the user. In a teleoperator like the nanoManipulator, the last stage also includes physical action.

the application must respond as fast as the user can perceive. Shneiderman (1998) synthesizes many narrow studies from the literature to give 100 ms as a rough upper bound for response time to continuous tasks in interactive applications.

Researchers in the virtual reality and computer graphics communities frequently use "interactive" to describe applications, but rarely carefully define or quantify the term. The general qualitative meaning is that a user sees the system respond to his input "fast enough." Writers claiming that a system is interactive often cite its frame rate – the number of times per second it displays updated imagery – as quantitative evidence. Any application that updates its display at a rate of 10 frames per second (fps) or better may casually be called interactive. However, this is only a lower bound on response time. Most 10 fps virtual reality applications have a response time much longer than their 100 ms frame time (Wloka 1995). A good frame rate is necessary, but not sufficient, to guarantee a good response time from a graphicsintensive application.

Many graphics applications increase frame rate by using pipelining – each of several processors executes a small portion of the work necessary to render a frame then hands off that frame to the next processor. Increasing the degree of pipelining improves frame rate, but not response time: frame time is equal to the length of the longest stage of the pipeline, while response time is equal to the total length of the pipeline. Only for non-distributed, non-pipelined graphics applications does frame time approximate response time.

> In this dissertation, I use the term **interactive applications** to refer to applications where closed-loop feedback driven by continuous input makes low response time critical to productive use.

Precise characterizations of interactivity are application-specific and depend on the interface modality. There is no task-independent metric or user study reported in the literature. In this dissertation, I use response time as the measure of interactivity.

I have measured the response time of the nanoManipulator system for three distinct types of interface: haptic force feedback (chapter 4), awareness and synchronization information in collaboration (chapter 7), and graphics driven by a remote rendering engine (chapter 8). All three of these interfaces involve continuous tasks, for which Shneiderman's (1998) 100 ms upper bound on response time is approximately valid. [10] Chapters 4, 7, and 8 survey prior work, giving specific information on consequences of and bounds on response time as found in the literature.

2.5 Studying Latency

Response time is the sum of various latencies – delays – in a system. Careful study of latency is not common in the Virtual Environment (VE) community. One recent study of VE latency is the work of Taylor et al. (1996), who have instrumented a number of networked VE systems to measure the response time (which they call "end-to-end lag," a term typically used in networking research) and built a standard

[10]With the use of an intermediate representation, discussed in chapter 4, haptics becomes a set of linked feedback loops, and the "outer" loop that includes the user can sustain a longer response time; in our experiments, depending on the intermediate representation used, this can be 200 ms or more. The inner loop requires a response time on the order of 1-2 ms.

model of system latency. The system they base their model on is a scientific simulation connected by a network to a virtual environment interface. They identify six basic sources of latency in such a system:

1. **Tracking lag** is the time that input devices require to measure and report a change in the user's position (measured by a tracker); more generally, the time to report a button press or release, or any other user input event.

2. **Simulation lag** is the time required by the simulation to compute a new set of outputs when given a new set of inputs.

3. **Rendering lag** is frame time: the time between the data being computed for a new view and that view being rendered into the frame buffer by the graphics hardware.

4. **Synchronization lag** is the time between any one process producing its output and the next process in a pipeline being ready to consume it.

5. **Frame rate induced lag** is the time between successive updates of the display device. (This latency is also known as scan-out time.)

6. **Network lag** is the time that messages spend travelling through the network between processes in the system.

As Taylor et al. (1995) emphasize, there are actually two different critical response times in the system:

View latency is the time between the user changing the parameters of their view (such as the position of their head) and seeing the view updated to reflect the new parameters.

Simulation latency is the time between the user changing the controls of the simulation and seeing the view updated to reflect the new control settings.

When two users share a collaborative application across a network, Bhola et al. (1998) similarly divide latency into two round-trip times:

Response Time is the time between a user's input and that user seeing the results of their input.

User-User Time is the time between a user's input and some other collaborating user seeing the results of their input.

This model of system latency is further discussed in chapter 7, where I analyze the nanoManipulator's collaboration subsystem.

The distributed nanoManipulator is a very similar system to those studied by Taylor et al. (1995): a virtual reality interface connected by a network to a microscope. However, working on the distributed nanoManipulator we have found that there is another important distinction to be made within the view latency:

Continuous command latency is the view latency in response to continuous input, such as trackers or mice.

Discrete command latency is the view latency in response to discrete input, such as button presses.

Continuous command latency would ideally be limited to 100 ms or less to get adequate immersion in a three-dimensional interface; current research points to 50 ms as a worthwhile target for virtual environments (Meehan et al. 2003). Discrete commands can often tolerate significantly higher end-to-end latency. Many of the approaches discussed in chapters 7 and 8 result from carefully tuning algorithms to each of these three types of end-to-end latency: continuous command, discrete command, and user-to-user.

Chapter 3

Related Work

In this chapter I introduce the Open Systems Interconnection (OSI) model of a computer network, which I use to classify and contrast the approaches to distributing feedback-critical user interfaces presented in this dissertation. I then examine a critical divide in approaches to building distributed applications. On one hand, we can consider changing the network to provide "better" service to the application; on the other hand, we can change applications to operate with the service that the network currently provides. Finally, I look at how other collaborative science systems – co-laboratories – have approached this divide.

The two high-level approaches laid out in chapter 1 – changing either the application or the network – were explored by researchers in the early 1990s investigating audio- and video-conferencing. Although the latency requirements of conferencing are less stringent than those of interactive applications, conferencing presents tougher bandwidth and jitter requirements, and none of these requirements are satisfied by the best-effort Internet. Many researchers believed that the network should offer guaranteed performance to enable reliable support for high-performance and real-time applications; this desire to "change the network" lead to today's body of work on QoS.

Other researchers felt that applications need to adapt to a best-effort network; this approach – "exploiting the inherent flexibility in the application" – led to work like the Two-Dimensional Scaling of Talley and Jeffay (1994). Adaptive techniques like these have become increasingly important with the rise of wireless networks, which have increased network heterogeneity and consequently reduced the feasibility of providing guarantees. However, neither network QoS nor application adaptation provides a simple solution to the problem.

QoS technology is not widely deployed on public networks, so an application intended for use over the public networks cannot rely on it. Further, although QoS methods can reduce the effects of wide-area distribution that interfere with an interactive application, they cannot eliminate these problems. Most methods of reducing loss and jitter also increase latency; they cannot support an application that requires low-latency message transmission as well as low jitter and low loss. This indicates that effort needs to be put into extending latency-tolerant and loss-tolerant application design techniques to apply to a wider range of problems.

3.1 The "OSI Stack" Model of the Network

It is common to both model and implement network software as a stack of independent layers. Today's Internet is conventionally described using a five-layer stack, with Physical, Link, Network, Transport, and Application layers (Kurose and Ross 2002). During an earlier era of networking research, the OSI proposed a seven-layer model, dividing the Internet's Application layer into Session, Presentation, and Application (Day and Zimmermann 1983; Tanenbaum 1989). The additional layers of the older model make distinctions that are useful in this dissertation (figure 3.1).

According to the OSI model, the Session layer handles end-to-end interactions between communicating processes, including directory services, load sharing, and access rights (Bertsekas and Gallager 1992). In this dissertation's model of adaptive applications, I place in the Session layer algorithms that coordinate behavior in the communicating processes to change the way they use the transport and network layers. For example, the Session layer contains the code that controls the choice of transport protocol, e.g. whether a communication uses TCP, UDP, or UDP with Forward Error Correction (FEC) (see section 6.2). I also place in the Session layer the use of either synchronous or asynchronous communication.

The OSI model groups data encryption, data compression, and code conversion into the Presentation layer. In the Presentation layer I place algorithms that encode intermediate representations, or choose to vary which intermediate representations are used (for example, see section 5.2.1). This can include algorithms which control how computations are distributed among processes – for example, choosing whether to render graphics locally or in a remote process (see chapter 8).

Finally, the Application layer contains the "leftovers." In this dissertation, those are the algorithms that choose to vary the user interface in response to network conditions.

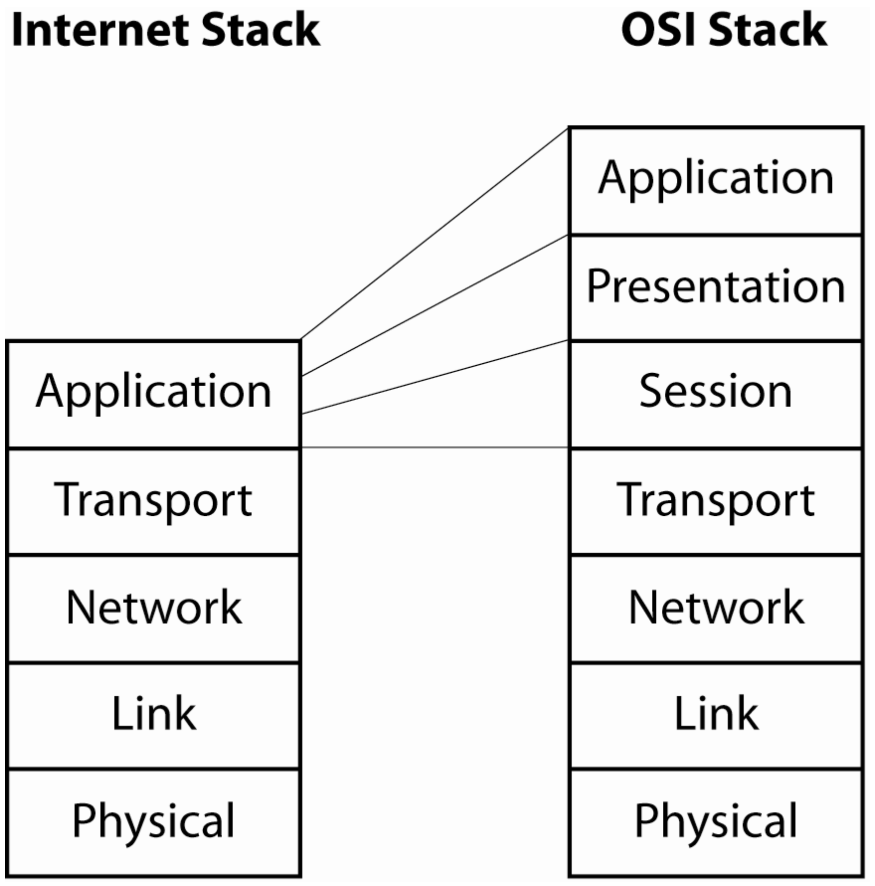

Figure 3.1: The Internet and OSI network stacks. Although current convention is to use a five-layer stack, this dissertation proposes roles for the Session and Presentation layer in implementing an adaptive application.

3.2 Changing the Network

For a LAN, network latencies range from 1 to 10 ms, well below Shneiderman's 100 ms bound. However, a WAN imposes much higher latencies on message transmission. Packet-switched WANs like the Internet also suffer from loss and jitter.

Today, all Internet traffic receives the same "best-effort" service. The network attempts to deliver all packets transmitted, but does not guarantee that any of them will be received. Latency-critical packets from an interactive scientific or medical application may be delayed by email, file transfer, or other less urgent data. QoS services attempt to remedy this problem by identifying packets that need better service and protecting those packets from interference by lower-priority traffic. In the extreme, each flow of packets receiving guaranteed QoS would be isolated from all other packets on the Internet, traversing through its own virtual private network. Two major approaches have been proposed for QoS in the Internet: Integrated Services (IntServ) and Differentiated Services (DiffServ).

Going beyond best-effort, Integrated Services (Braden et al. 1994) provides two premium levels of service, known as Guaranteed and Controlled-Load. Both services attempt to control latency for every packet in a flow that has requested QoS: Controlled-Load service insures that almost all packets will meet the bounds, while Guaranteed service gives an absolute promise that every single packet meets the latency bounds (unless some part of the network fails completely). IntServ provides this bound by reserving network capacity before any data packets are transmitted. The endpoints of a connection specify the service they need as part of a call setup procedure. Conceptually, this "Flow Specification" states the peak bandwidth an application will consume and maximum latency it can tolerate (Wroclawski 1997a; Shenker et al. 1997). Each router along the network path between the communicating computers executes an admission control algorithm to determine whether or not it can meet the guarantees requested by this new traffic source. Once all routers have agreed that a connection can be supported, data can begin to flow (Braden et al. 1997; Wroclawski 1997b).

IntServ makes a strong guarantee of the performance that will be provided to packets. However, IntServ is a radical departure from the architecture of the existing Internet. IntServ requires that routers maintain status for each flow of information that transits them and participate in end-to-end signalling, roles explicitly forbidden in the initial design of the Internet Protocol. Although IntServ has been specified and sample implementations are available, it has not been and is not likely to be deployed. Implementing IntServ adds a large computational and storage overhead to routers. Furthermore, IntServ can only make guarantees if it is deployed in every router along a network path.

A simpler but weaker approach to QoS is Differentiated Services. Nichols et al. (1999) proposed a strawman design for DiffServ, the "Two-Bit Architecture." Like IntServ, Two-Bit DiffServ provides two new levels of service to network flows, which it calls Premium and Assured. The architecture's name derives from the fact that two bits in the Internet Protocol (IP) header are used to specify (or "differentiate") the level of service each packet should receive.

A packet marked for Premium (high-priority) service will be forwarded through a router before any non-Premium packet waiting in that router's queues is forwarded. In return, the application sending Premium packets guarantees that its bandwidth demands will not exceed given peak and average rates; any packets that do exceed the limit will be dropped in the network. This contract between the application and the network is known as a service level agreement. The limited peak (low burst size) enables the network to minimize the amount of buffering used for Premium flows and helps reduce latency.

A packet marked for Assured service (with its "A-bit" set) does not have the absolute preference of Premium packets but instead receives a statistical advantage over unmarked best-effort traffic. Best-effort and Assured packets are mixed together in router queues, but when queues begin to overflow best-effort packets are dropped first.

One of the great strengths of Two-Bit DiffServ is that it should be simple to implement in routers. A router in the interior of a network need only check the Premium bit in the header of each incoming packet to determine which queue to place it into. Premium packets are put into a high-priority queue; all packets from the high-priority queue are sent before any packet in the low-priority queue is sent. Assured packets are mixed with best-effort packets in the lower-priority queue which uses the RED/RIO[1] queue management scheme to control packet dropping; best-effort packets are preferentially dropped. Following the original Internet design, the complexities of QoS management are pushed to the edges of the network; no state needs to be maintained at the routers. Unlike IntServ, DiffServ can provide partial QoS even if only some routers in the network support it.

Current development of both IntServ and DiffServ has shown promising results on the small segments of the network over which they have been deployed. However, above and beyond the challenge of deploying these new services across the Internet – a one-time problem – is the ongoing problem of administering and maintaining QoS. One of the reasons IntServ appears to not be succeeding in the marketplace is its complexity – it includes support for flexible, dynamic allocation of QoS to applications

[1]Random Early Drop / RED with In and Out (Clark and Wroclawski 1997), a congestion control scheme for routers. DiffServ requires that some method of congestion control be adopted at routers, and RED is the current favorite.

that request it on the spur of the moment. DiffServ's strength – its simplicity – is also a critical weakness: without automated support for administration and management, DiffServ requires either expensive, long-term contracts for a given level of QoS, or a large amount of work by technicians and administrators to set up shorter-term connections. For an interactive application like the nanoManipulator, meant to be used intermittently and informally, neither significant administrative overhead (to get guarantees on short notice) nor high cost of service (to pay for continuous QoS that will only occasionally be used) is acceptable.

With a private network – one not open to all comers like the Internet – some widely-distributed interactive applications may be possible. In September 2001, transcontinental telesurgery was performed: doctors in New York did a laparoscopic gall bladder removal on a patient in Strasbourg, France (Marescaux et al. 2002). They had a direct network connection, high-speed fiber-optics from end to end with no cross traffic. Round-trip latency was 80 ms, high enough to prevent force feedback from being useful. The straight-line distance between these two points is roughly 7000 km, leading to a round-trip propagation delay of at least 47 ms. Since surgeons typically use force-feedback as much as vision in performing laparoscopic surgery, and only video was available (no force feedback was provided), the surgeons worked slowly, much like the "move-and-wait" use of teleoperators discussed in section 4.4.1. Other components of the system added at least 75 ms of latency, bringing total system latency over 155 ms.

One significant contributor to latency in packet-switching is the time each packet spends sitting in router queues waiting to be routed or serviced. Kessler and Hodges (1996) proposed the Updatable Queue mechanism to reduce this latency: for certain types of data, if one message has been sitting in a queue and a newer one arrives, the old message can be thrown out since it is made obsolete by the new one. In the nanoManipulator, this could be a report from `nano` telling the AFM where to move, or an update from `topo` telling the Phantom what local plane approximation to use. The nanoManipulator runs a mechanism derived from Kessler and Hodges's to reduce queueing latency at `nano`.[2]However, queueing latency can also accumulate at every router in a network. In a dedicated network we could consider running an updatable

[2]A more significant contribution of the Updatable Queue mechanism to the nanoManipulator is its use in matching rates: `nano` generates commands as fast as it can, and `topo` processes them at its own best speed (significantly slower); the Updatable Queue guarantees that excess commands from `nano` are thrown away instead of contributing to an ever-increasing backlog.

queue at every router, significantly decreasing network latency. This is not feasible in today's Internet, although it might be enabled by recent proposals calling for "Active Networks," in which each connection can specify complex operations to be carried out on its messages at mid-stream routers (Tennenhouse et al. 1997).

3.3 Changing the Application

Without guaranteed performance from the network, the application must sense what performance the network is giving it and choose an appropriate algorithm or adaptation scheme to achieve its goals.

In this dissertation, I discuss three classes of adaptive algorithms. Session-layer adaptive algorithms change the network behavior of the application. Presentation-layer adaptive algorithms change the way function is distributed between the system's processes to change the network behavior required by the application. Application-layer adaptive algorithms change the application's user interface to change the application behavior required by the user.

3.3.1 Session-Layer Adaptation

Audio and video applications have been the major drivers for QoS networking. Audio and video hardware that connects to personal computers has become widely available, supporting teleconferencing. Web sites "stream" audio and video across the Internet. Consumers are also using Internet telephony. All of this demand for networked audio and video has created a demand for research: these media require high bandwidth, low jitter, and low latency, none of which are provided by the Internet. However, there are many possible tradeoffs, sacrificing some component of media quality in return for some reduction of network requirements. Thus, networked audio and video have also driven research into adaptive applications. [3]

The network requirements of different video streams vary, depending on their content, the coding and compression schemes used, and the resolution and frame rate desired, which in turn depend on the application for which the video is being used. Early video telephony was aimed at providing acceptable pictures of a person's

[3]Audio and video are often called streaming media, and a transmission of A/V data called a stream.

head using 64 - 128 kilobits per second over standard telephone lines (Turletti 1993). General-purpose video requires a higher bandwidth: the H.261 standard can use bandwidth ranging from from 40 kilobits to 2 Megabits per second (Mbps); MPEG1 aims at rates between 1 and 1.5 Mbps (Mitchell et al. 1996). Although today's networking technology can provide aggregate bandwidth far in excess of these levels, congestion and the "last mile" problem mean that we can not rely on having this bandwidth available.[4]

Audio has much more stringent loss and latency requirements than video. For human speech to be understood, audio data must be delivered in a continuous stream with few gaps – little jitter. Studies of teleconferencing show that comparable amounts of jitter and loss make an audio stream incomprehensible long before they begin to interfere with video. Recent authors have argued that latency must be kept to less than 200 ms to provide acceptable audio quality (Ferrari 1990).

TCP corrects for loss by detecting when loss occurs and retransmitting the lost packet. Often, loss is only detected when a timer expires. This unavoidably adds latency, since loss is equivalent to unexpectedly high delay. FEC has been used to ameliorate loss in audio and video (Bolot and Vega-Garcia 1996; Bolot and Turletti 1996). FEC anticipates loss and takes action to recover from this loss in advance, rather than reacting to it. FEC introduces redundancy into the stream; if a packet is lost, one of the redundant copies of that packet will probably not be. This approach to controlling loss makes a tradeoff, significantly increasing the bandwidth required for a stream in exchange for greatly reducing the expected latency required to correct loss. Bolot and Vega-Garcia (1996) is fully adaptive, monitoring the distribution of losses in the network to determine the necessary degree of redundancy.

Many audio and video transport systems have been designed to operate in a multicast environment, where one computer is transmitting data to many receivers. Different receivers may want differently-parameterized data, or be connected by networks of differing bandwidths. Systems such as the Heidelberg Transport System (HeiTS)

[4]In 2004, the existence of the "last mile" problem is debatable. In the mid- to late 1990s, the network backbone had been upgraded, but many individual PCs were still sharing 10 Mbps connections with large numbers of other computers. This changed faster than expected. For example, many university campuses are now wired at 10 or 100 Mbps to every desktop. For Research I institutions, such as the University of North Carolina at Chapel Hill, bandwidth may no longer be in short supply. For smaller universities, such as the University of North Carolina at Wilmington, this moves the bottleneck from the LAN to their uplink to the WAN, where traffic from many desktop and laptop computers may be aggregated to a link of 10 Mbps or less. A high school is still lucky to be connected to the Internet at 1.5 Mbps.

(Delgrossi et al. 1993), the Multimedia Multicast Channel (Pasquale et al. 1992), and Receiver-Driven Layered Multicast (McCanne et al. 1996) are designed with this heterogeneity in mind. These systems decompose the audio and video streams into several substreams, allowing receivers to reconstruct high-quality picture and sound if they receive all the streams or lower-quality picture and sound if they only receive a subset of the streams. HeiTS also allows the parameters of an individual substream to be modified.

Adaptive application research proposes novel techniques for distribution which are well-suited to network pathologies. These methods can be combined into a parameterized system driven by closed-loop feedback: a system which evaluates the current performance of the network and dynamically changes its data transfer requirements. When these changes alter the quality of the output, the system is said to be performing **media scaling**.

HeiTS performs closed-loop feedback, dropping substreams when multicasting to less capable receivers or reducing fidelity when unicasting. Talley and Jeffay (1994) expanded this media scaling from one-dimensional, adapting to meet only a bandwidth constraint, to two-dimensional: a network's feasible packet rate is partially independent of its feasible bit rate. Additional possible dimensions for adaptation are added when supporting wireless communication. For example, the Fugue system adapts transmitter power, bit rate, quantization (lossyness of compression), and frame rate (Corner et al. 2001).

3.3.2 Presentation-Layer Adaptation

Time-critical algorithms adapt to meet deadlines: if an exact computation is too expensive (will take too long and cause the system to miss a deadline), an approximate computation is performed instead. They are used in situations where a timely, approximate answer is more useful than an answer that is correct but late. Time-critical systems have been the focus of a significant body of prior work on adaptive, non-distributed interactive applications. Like many networked adaptive algorithms, they cannot directly measure or compute their parameters, and must instead estimate costs based on recent history. If the previous iteration of a computation missed its deadline, the next iteration uses a cheaper approximation. Applied to graphics, this technique has yielded a number of systems that render an approximately-correct or low-quality image when they do not have enough time to render a high-quality image (Holloway 1992; Gobbetti and Bouvier 1999; Klosowski and Silva 1999). Some of the earliest applications of this idea were systems that displayed a wireframe image of a scene while the user was moving, and rendered in solid objects when the user held still. There are also time-critical simulations, which approximate the physics of a scenario when the fully simulated computations are too expensive to complete in their allotted time. For example, Hubbard's (1995) dissertation covered time-critical collision detection, giving an algorithm that smoothly varied from approximately to exactly detecting collision or interpenetration of complex polytopes.

A number of authors have used multiprocessing to attempt to minimize latency inside virtual environment systems (Shaw et al. 1992). Wloka (1995) looked at combining ideas from time-critical computing research with these multiprocessor latency-minimization techniques. He presented a general multiprocessor architecture for virtual environments and a synchronization scheme to minimize lag (latency) due to synchronization stalls among the processes. Synchronization stalls occur when one process produces data before another process needs to use it; the subsequent process can either use the stale data, which has been sitting in buffers since it was produced, or wait until the producer has new data for it. Either option adds to the effective latency of the system – either the amount of time that the data was sitting in buffers or the amount of time that the consumer process waits for fresh data.

One example of a time-critical system is a Distributed Virtual Environment (DVE): a virtual environment designed to be distributed across a network, almost always a wide-area network. Many DVE applications focus on their bandwidth requirement as the chief obstacle to be overcome (Sandin et al. 1997; Capps 2000). Naive transmission schemes require large bandwidths to distribute the model or model updates. In some systems, the "world model" – model of the environment – is transmitted to the user incrementally. In others, various area-of-interest or level-of-detail schemes are used to reduce the amount of model data or number of updates that must be sent to each user (Macedonia et al. 1995; Hesina and Schmalstieg 1998).

Area-of-interest approaches have also been used to control dissemination of audio and video streams (Benford et al. 1994). Here, again, the dissemination was controlled to minimize bandwidth consumption by the virtual environments system.

When Capps (2000) writes about latency, it is primarily viewed as a side-effect of limited bandwidth and large numbers of users: "...limitations on network bandwidth usually affect latency more than round-trip communication times ..." In Capps's QUICK system, end-users use computers connected over modems (28,800 baud) to view a huge dataset or virtual space. QUICK assumes that users must dynamically download the relevant portions of this dataset, and that bandwidth used is constrained by their 28,800 baud connection. The nanoManipulator has the opposite situation, where propagation and queueing times are much greater than transmission time, i.e. distance and congestion contribute more to latency than does bandwidth. This makes many of the techniques, and even the conclusions, of prior research inapplicable to our system.

3.3.3 Application-Layer Adaptation

The third approach I explore is to assume that network latency is beyond the application's control or ability to hide from the user. Instead of trying to influence the network performance, or change the implementation of the application and its demands of the network, we can change the application's user interface. Human beings are highly adaptable; it has long been established that operators can teleoperate with delays of tens of seconds if they do not use force feedback. Instead depending on vision, users adopt a "stop-and-wait" approach: they make an incremental change to the system, then wait for that change to propagate across the delay and for its effects to be observable (Ferrell 1966). Stop-and-wait is quite slow, and with high delays even stop-and-wait teleoperation is error-prone (Ferrell 1966). Modern research is trying to find ways to augment the user interface so that humans can operate in high-latency environments without having to discard their reflexes.

Conner and Holden (1997) devised a consistent set of visual cues for users in a DVE with latency problems. In their system, users attempted to manipulate objects, but only one user could manipulate each object at a time. When both tried to manipulate the same object, race conditions could ensue, and network latency interfered with user actions. In their system, when a user attempts an operation, the object being manipulated is rendered transparently until the success or failure of the operation is known. When a user moves an object, motion blur is used to visually smooth jittery updates from the network. When an expected message does not arrive from the network, the object from which that message was expected is defocused to indicate uncertainty. These three cues – transparency, blur, and defocus – are meant to work together, to have similar visual impact while approximately informing the user of several different network states.

Fraser et al. (2000) developed a set of visual cues to address latency and other characteristic problems of DVEs. A user of their collaborative system was shown latency in two ways. First, every other user's avatar is surrounded by a sphere that represented the uncertainty in the user's position, which is the product of network latency and the user's potential velocity. Second, every other user's avatar could also be annotated with an icon that showed the network latency that user was subject to.

Vaghi et al. (1999) studied the problem of inconsistency in DVEs with high network latency. In their system, two players competitively played a game of Pong. The authors simulated a variable network latency between the two players, studying how their performance changed as a result of several different levels of latency (50 - 999 ms). Vaghi et al. (1999) listed a lengthy set of general approaches to designing user interfaces that alert the user to latency and help her adapt to it, as well as a set of specific mechanisms that are suitable to their application. Some of their approaches are closely related to the techniques I used in chapter 7, and are discussed in detail there.

The one major project that addresses the effects of network latency on user interaction is Distributed Interactive Simulation (DIS), the United States Army's massively multi-user training VE. DIS trains soldiers in simulated combat, on foot, driving, or flying aircraft. Because the trainees are at military bases spread across the continent, latency is unavoidable. The simulation can also require high bandwidth, which can lead to congestion and additional latency. The DIS literature is voluminous; Singhal and Zyda (1999) provides a summary of the lessons of DIS and of NPSNET (Macedonia et al. 1994), a research system from the Naval Postgraduate School that improved on DIS's performance.

DIS is best known for its use of "dead reckoning" to keep track of players' locations. In addition to broadcasting her position, a player A broadcasts her velocity and acceleration. The code driving other players' displays stores A's velocity and acceleration and applies it to its knowledge of her location until players receive a new update from A. This enables A to send far fewer location updates than would be necessary otherwise: A models the position that all other players simulate for her and only sends an update when error between that position estimate and her true position exceeds an a priori error bound. The system can vary the error bound, adjusting the rate of A's updates to adapt to network conditions.

DIS's designers carefully decided which algorithms would be distributed and which would be centralized, basing the design on the underlying semantics of the simulation. Using dead reckoning to estimate positions permits bounded divergence between the worldviews of different players, minimizing the latency they experience. For data that players need to agree on – for example, whether or not one player is hit when shot at

– DIS maintains consistency by determining "the truth" at a single location. Users at other locations have their perception of the result delayed by network latency, but the system ensures that the results they perceive are consistent.

3.4 Collaboratories

Collaboratories ("co - laboratories") are computer systems that enable cooperation among scientists. The term is also used loosely to refer to systems that provide remote access to instrumentation. The distributed nanoManipulator is one of several collaboratories developed near the turn of the millenium. Researchers at the Beckman Institute have created a number of programs that allow simultaneous remote access to a Scanning Electron Microscope (SEM) by many users, including Bugscope (Potter et al. 2000). Users have a web-based interface that shows them frames of video captured by the SEM. Buttons on this interface cause the SEM to pan, zoom, or otherwise modify its parameters in discrete steps. In the Beckman systems, either each remote user is either independent, completely unaware of the simultaneous investigations being carried out by other remote users on the same sample, or passive, merely observing another remote user's investigation without being able to communicate with them (Carragher and Potter 1999). Bugscope does provide the ability to exchange text messages with a technologist operating the microscope during these sessions.

The Upper Atmospheric Research Collaboratory provides shared access to data from a number of scientific instruments: incoherent scatter radar and other instruments, mostly based on the ground, that observe Earth's ionosphere (Olson et al. 1998). The software does not provide direct control of the instrumentation: scientists who want to vary parameters must send text messages to operators located at the instruments' sites.

Neither of these examples is an interactive application within the scope of this dissertation. Both user interfaces are discrete, not continuous; human latency tolerance is significantly higher in this case.

Arizona State University has a distributed, non-collaborative version of the nanoManipulator reminiscent of the Bugscope. That is, it provides a web-based interface to the nanoManipulator's three-dimensional renderings of a live AFM, and lets users manipulate the visualization and the AFM from their web browser, but does not support manipulation or interaction between multiple sites simultaneously viewing the data (Razdan et al. 2000). Control of the instrument is discrete, not continuous, giving this collaboratory a higher latency tolerance than the nanoManipulator but less manipulative potential.

3.5 Summary

Even applications using QoS-enabled networks may not get the network performance they need. Thus, it is necessary to build applications that can adapt to the performance that the network provides. It is possible to adapt at the application, session, or presentation layers. This dissertation presents a remote teleoperator, remote renderer, and collaboratory using session- and presentation-layer adaptations, which are not application-specific, and are thus a good starting point for future applications.

Chapter 4

Haptics and Latency

haptics: active exploration by touch.

The nanoManipulator's force feedback (or "haptic") interface – the PHANTOM robot arm that is coupled to the microscope's tip – is the component of the system whose performance is most degraded by distribution across a network. After defining haptics and considering the problem of measuring haptic quality, I survey prior attempts to provide force feedback across a network. One of the foci of this body of research is determining a latency bound: "How much latency is too much?" The bounds previously discovered do not directly apply to the nanoManipulator, due to our different implementation techniques;[1] I show how to use Fourier analysis to extend these bounds, deriving bounds for the techniques used in the nanoManipulator, and give both anecdotal reports by scientists of the nanoManipulator's usability and experimental results that appear to validate them. Having latency bounds for the plane approximation will let implementors know whether or not it will work for their application before they implement it.

4.1 Haptic User Interfaces

Psychologists divide the sense of touch into two subtypes: the kinesthetic sense, through which we sense movement or force in muscles and joints; and the tactile sense, through which we sense shapes and textures (Hannaford and Venema 1995). When you carry out an action, you feel force exerted on your body by the environment.

[1]Prior experiments seeking bounds have used point-sampled position control (section 4.4.1); we used the plane approximation (section 4.4.2).

I use the terms "haptic interface" and "force-feedback interface" to mean a user interface that communicates information to a human user via the kinesthetic sense. Adachi et al. (1995) define a haptic interface as "a device which generates mechanical impedance," where the user feels a stiff surface when there is "a sudden transition from very low to very high impedance."

Haptics is often used in teleoperation, the control of machinery at one location by a person at another location. The user transmits commands to the machine over a network or radio link and receives data about the progress of the operation. Most teleoperation has involved manipulators – jointed arms with a gripper used to grasp objects at the remote site. The operator usually uses his own arm and hand to carry out an action naturally; the manipulator arm mimics his movements. This gives the operator a kinesthetic and proprioceptive sense of his action. The mapping from operator input to manipulator action is direct, instead of abstracted. With appropriately low latency and visual feedback, the operator can use his own reflexes to carry out the task.

Many manipulators use force feedback: the remote manipulator measures the force exerted by the environment, and this force is exerted on the operator through the same device that is reading his position as input. When used in teleoperation of remote manipulators, force feedback usually improves task performance (Hannaford et al. 1991). Even when the end-effector is simple – in the nanoManipulator's case, just a sharp tip that the user moves horizontally while the computer controls its vertical position (Figure 4.1) – force feedback can be very useful. See section 2.2 and Taylor et al. (1997) for the nanoManipulator, and Mitsuishi et al. (1995) for another example.

The standard implementation of teleoperation is a simple feedback loop: a user moves an input device (the "master"), the new position of the device is measured and transmitted over the network to the effector (the "slave"), the slave moves to a corresponding position, the force exerted on the slave by the environment is measured and transmitted back to the master, where it is exerted on the user (possibly scaled). Since the user controls the position of the slave by moving the master, this is known as **position control**. Since the measured forces are being fed back to the user, it is also known as **Direct Force Feedback (DFF)** – direct in the sense that no intermediate representation is used.

The nanoManipulator uses position control, but not DFF: it uses an intermediate representation to model the environment and to compute an appropriate force to exert on the user, rather than exerting the force from the environment on the user. In this dissertation I use the phrase "position control" to refer to any device where the user controls the position, and DFF only for devices where the force fed back is directly measured.

Force feedback has also been used in VE interfaces for applications other than teleoperation, usually applications where the virtual "remote environment" manipulated is a computer-driven simulation instead of actual physical objects. For example, the GROPE project used haptics to help scientists understand the interactions of proteins (Brooks et al. 1990). InTouch lets users "paint" the surface of a computerized model while feeling as if they were applying a real paintbrush to a real surface (Gregory et al. 2000).

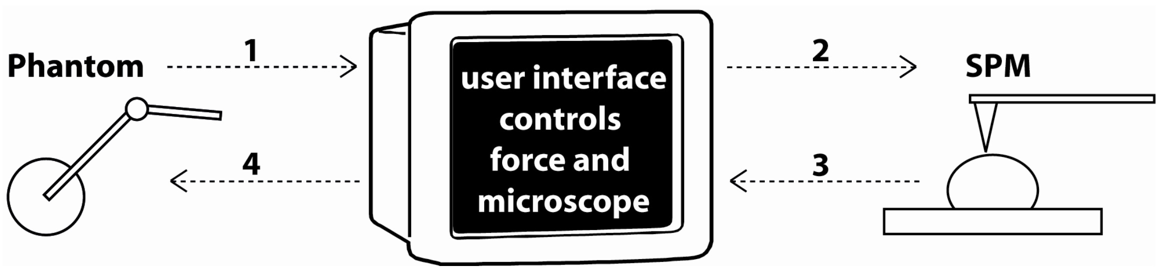

Figure 4.1: Conceptual feedback loop between a haptic device and a Scanning Probe Microscope, repeated from figure 2.3. (1) position of user's hand is measured, (2) movement commands are sent to the microscope and it moves, (3) height of sample is measured at the new location, (4) force is applied to the user.

4.2 Evaluating Haptic Interfaces

There are three general requirements for the user interface of any force-feedback system: stiffness, correctness, and stability. However, no task-independent, perceptually-based measurement of the quality of a haptic system in terms of those requirements has been established. Investigators have compared the effectiveness of force feedback manipulators by measuring test subjects' completion times for simple tasks; they have also tracked error rate, peak force exerted by the effector, variance in force exerted, and sum of squared forces exerted.

In assessing users' perception of remote environments, the first measure of the quality of a haptic system is the maximum stiffness of the surface that it can present while maintaining stability. When the user causes the effector to move into contact with the surface of a specimen, he feels a **virtual surface** through his input device. Is the shape he feels **stiff** – crisp and well-defined? Or is it "mushy" and indistinct.

It is also important that correct surfaces are displayed. If the operator feels a surface contact through the input device when the effector hasn't actually made contact, or if the surface geometry felt is misaligned or incorrect, he will make mistakes.

Unstable systems are systems that can go out of control. When sufficiently perturbed, an unstable system can begin oscillating; the increasing magnitude of the oscillations can cause the system to fail. Smaller perturbations can cause oscillations that do not lead to failure but interfere with normal operation. There is a branch of mathematics, control theory, which deals with this phenomenon. Particularly in the face of latency, human input can be a cause of instability in electro-mechanical feedback control systems. To be safe and correct, every force-feedback system needs to be stable. Unfortunately, high stiffness contributes to instability, so stability concerns limit the stiffness displayed by force-feedback devices. The stiffer the surface displayed by the system, the stronger and more sudden the force that will be exerted on the user when he pushes up against it. He is pushed away from the surface by the sudden force and attempts to contact it again, beginning an oscillatory cycle. Stiff surfaces will miscue his reflexes, causing inadvertent over-reactions that lead to instability.

We need to consider the perceptual capabilities of human users to gauge exactly how much stiffness and correctness is necessary. These two essential qualities depend on a number of parameters, many of them determined by the mechanical design of the input device and the effector. In this dissertation, I look at two of the parameters that affect stability and correctness: update rate and latency.

4.2.1 Haptic Device Update Rate

Human touch responds to different types of stimuli at different rates. Although a human being normally cannot discriminate between consecutive forces occurring at frequencies above 300 Hz, during skillful manipulation tasks we may be sensitive to vibrations at frequencies as high as 5,000 to 10,000 Hz (Burdea 1996). Haptic systems must run a control loop at a comparable rate to successfully present stiff force feedback: the force they exert on their human user is updated roughly 500 to 1000 times per second. The force output update rate is typically matched by the rate at which haptic devices sample their position sensors, so that this is also often an approximate measure of how frequently the device measures the position of the user's hand.

4.2.2 Latency and Latency Bounds

Early work assessing the effects of delay on teleoperators found that 100 ms of delay in haptic feedback was enough to destabilize position-control teleoperators, causing them to oscillate to failure (Ferrell 1966). Even smaller amounts of delay would make the system effectively unusable. Ferrell also found that a stop-and-wait strategy (operators made a small movement, then waited to discern the results of that movement) avoided instability, but made force feedback *unclear* and significantly degraded performance. Few studies of the human factors of time-delayed teleoperation were made from that date until the 1990s, when new theoretical approaches to latency-tolerant force-feedback needed to be evaluated. Vertut et al. (1981) tried to use delayed teleoperators without the move-and-wait strategy. To successfully complete their tasks users had to reduce velocity to 10 cm/s, greatly increasing completion time. Anderson and Spong (1989) found instability with as little as 40 ms of network delay; similar figures are reported in the avionics literature (Wickens 1986). In Anderson and Spong's experiments, the instability occurred with tasks much less complicated than those required in the nanoManipulator: linear motion into contact with a hard surface, or sinusoidal motion without contacting the environment.

All of these studies used position-controlled force-feedback manipulators, so their results are not directly applicable to the nanoManipulator (section 2.2). In section 4.6 I show how to use Fourier analysis to derive latency bounds for the plane approximation given these approximate 40-100 ms latency bounds for position control.

4.3 Constraints on Device Update Rate and on Latency

Both high update rate and low latency are necessary for stability.

The first constraint on update rate is provided by the hardware that displays forces to the user. SensAble(TM) Technologies' 6 Degree of Freedom (DOF) Phantom(TM) arm updates its output force at 15,000 Hz, but most haptic devices run at lower speeds. The 3-DOF Desktop Phantom used in the nanoManipulator is controlled at 1,000 Hz. Burdea's (1996) survey lists several devices, with update rates as low as 10 Hz and none higher than 3,000 Hz.

Most traditional teleoperators have control software that sends signals across the network – control signals from controller to manipulator and responses from manipulator back to controller – and updates the force to be displayed at the same frequency as the hardware's internal display loop (section 4.4.1). However, most VE applications use approaches that allow both the VE software and network transmissions to run at lower rates (section 4.4.2), as do non-force-feedback teleoperators designed for high-latency situations (section 4.5).

Update rate determines a lower bound for latency – if the software only communicates with the hardware at 10 Hz, then the hardware will display forces that are as stale as 100 ms, and the software will use similarly out-of-date measurements of the user's input in its calculations. The nanoManipulator typically only receives position reports from the Phantom every 16.7 ms (at 60 Hz), even though the Phantom can provide samples every 1 ms (at 1000 Hz). Thus, the nanoManipulator system design, as implemented, adds over 15 ms of latency to the haptic control loop.

Once update rates are sufficiently high, latency is principally influenced by the transmission delay. This chapter addresses the problem of latency in haptics: it surveys previous approaches to operating in the presence of transmission delay, proposes new methods, and evaluates these new methods.

4.4 Force-Feedback Teleoperation

In the next two sections I survey previous work in teleoperation. Direct force feedback, as described above, is an excellent interface for low-latency remote manipulation. However, force-feedback teleoperation is so impaired by latency that many researchers in long-distance remote manipulation have abandoned it for open-loop approaches, which are discussed in section 4.5.

4.4.1 Direct Force Feedback

DFF position-controlled teleoperators have used a dedicated network or bus to connect the human-manipulated "master" to the mechanical "slave." Although the bandwidth requirements are small (perhaps 9600 bits per degree of freedom per second), 500 to 1000 tiny messages have to be transmitted from slave to master (with position commands) and back (with force responses) every second; this is only possible over a tightly synchronous, dedicated, short-haul network with minimal protocol overhead.

To avoid instability, many teleoperation researchers have reformulated the control loop to use “passive control” (Anderson and Spong 1989). This exploits an analogy in control-theoretic analysis between a teleoperator and an electrical transmission line. Because transmission lines are **passive devices** – devices that never add energy to the system – they are provably stable under all time delays. However, a passive manipulator gives decreased surface stiffness – a spongy or mushy feeling – when delay is long, and increases tasks' completion times at all levels of network delay (Lawn and Hannaford 1993). These make passive control a less-than-optimum solution for distributing the nanoManipulator.

Most work in passive control assumes a known constant delay. The Internet suffers from both high latency and high jitter. Variable delays are so difficult to design control systems around that most researchers do not consider the Internet a feasible medium for teleoperation. Niemeyer and Slotine (1991) developed a new mathematical approach to expressing passivity. Their more recent work shows that a simulated passive teleoperator allows stable teleoperation over the Internet in the face of unknown, varying latency (Niemeyer and Slotine 1998). Like earlier passive approaches, their system “feels soft when the instantaneous delay is large,” although it automatically improves the stiffness displayed as delay reduces. Niemeyer and Slotine (1998) provide a novel method to compute some of the system parameters that avoids the numerical errors normally suffered by digital force-control algorithms,[2] but in their simulations the approach still suffers from incremental divergence between master and slave. Deployed in the nanoManipulator, this technique would mean an increasing error between the positions of the Phantom and the AFM tip. Thus divergence is another argument against the use of passive control for manipulation in the AFM.

4.4.2 Intermediate Representations

The high update rate required by traditional approaches to force-feedback teleoperation is a significant obstacle to combining haptics with programs that have other

[2]In a system where only position measurements are available, velocity must be estimated by taking differences. Unfortunately, this approach to differentiation is known to be highly noisy.

computational demands. The most common solution used in VE applications is an intermediate representation, particularly the plane approximation (Adachi et al. 1995; Mark et al. 1996).

In chapter 1, I defined an intermediate representation as "... a representation of some data chosen for its suitability for a distributed application, ... not the "native" form in which the data was generated or will be displayed." If a force feedback system measures and displays positions and forces, then a force feedback system using an intermediate representation is one that does not transmit positions and forces over the network. The approaches discussed in this dissertation transmit a position command from the Phantom to the AFM, but the response from the AFM is not the measured force exerted on the AFM tip by the sample, even though that force is what the user would expect to feel. *Not* transmitting the force allows us to increase the latency tolerance of the feedback loop.

Direct force feedback measures force exerted at the effector and feeds it back to the user. Instead, Adachi et al. (1995) specify the force fed back indirectly, as the tangent plane of the contact between the effector and the environment. The process that determines the tangent plane can specify the plane at a low rate, say 20 Hz; a process dedicated to controlling the force feedback device can sample the device position at the 500-1000 Hz required and compute the force that should be felt at that position given the plane. When a new plane is specified, the device controller can interpolate between the old and new planes for 10-20 ms to hide the transition; this helps present a smooth surface (Mark et al. 1996). Returning to figure 2.4, although sections (1) and (4) of the teleoperation system – hardwired, direct links from devices to controllers – may represent the feedback as force, sections (2) and (3) represent the feedback as a plane.

This plane approximation is a first-order approximation to the shape of the surface, which is valid in a small region around the point at which it is specified; it is a close physical analog to using the first two terms of the Taylor series of a function as an approximation to that function at a point. Interpolation between adjacent samples also makes the plane approximation moderately tolerant of messages being lost in the network. It is most applicable to environments that only have a single surface, as it does not do a good job of displaying a sharp crease where two surfaces come together or a narrow channel defined by two parallel surfaces.[3]

[3]Adachi et al.'s (1995) original test case was a cylindrical surface without features – this kind of uninterrupted constant curvature is an excellent but simple case for the plane approximation.

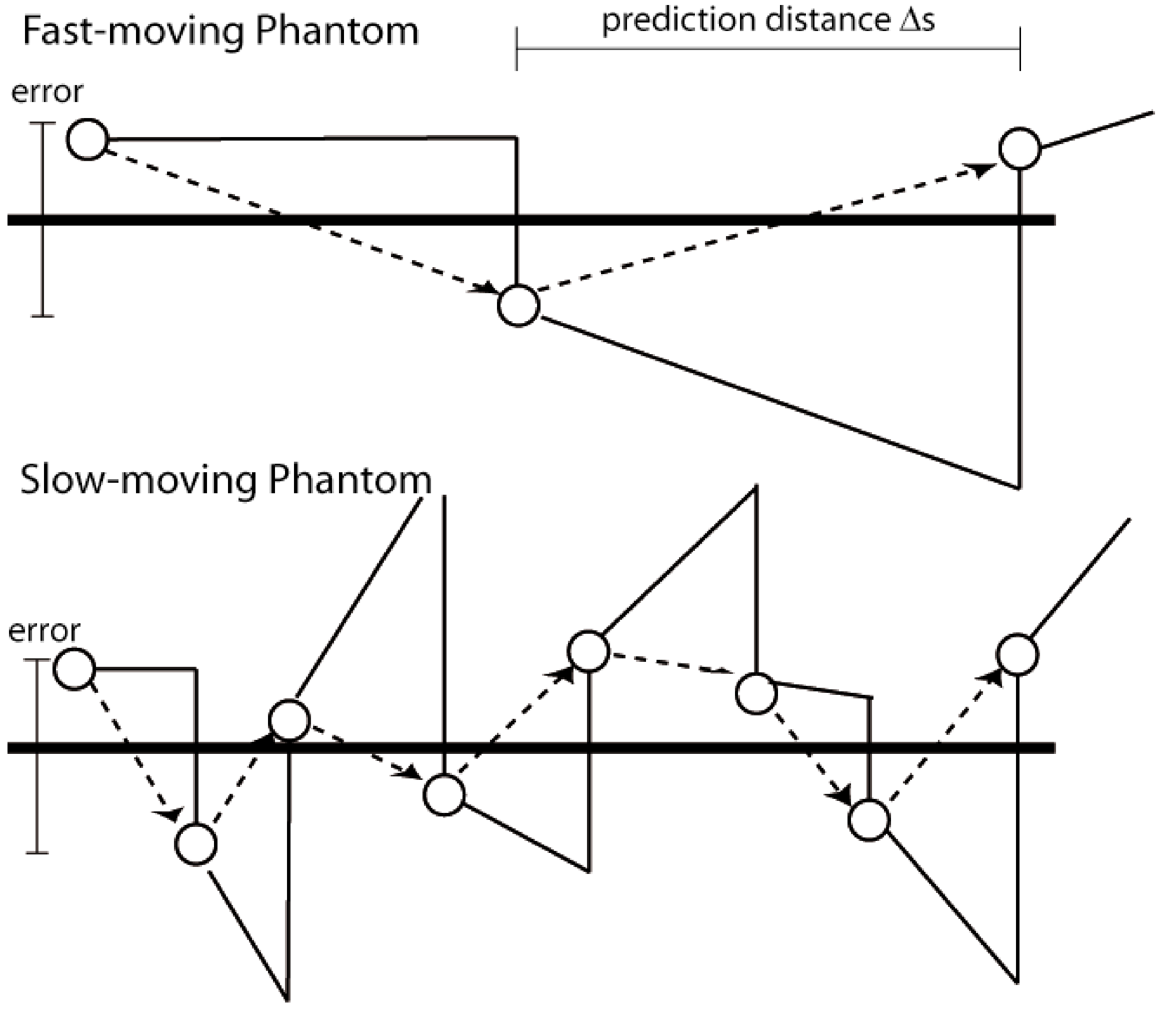

Figure 4.2: Effect of measurement error on perceived surface shape. The dotted line is the sequence of planes that will be felt by the user. The signal-to-noise ratio is effectively much worse when the user is moving slowly, because the same magnitude of noise creates a much higher variance in the slope of the felt surface.

The plane approximation is usually indirect in a second way: instead of measuring the surface stiffness, it assumes a (constant) stiffness for the felt surfaces.[4] For malleable surfaces, subsequent updates to the position of the approximating plane will move the plane, producing some approximation of the actual surface stiffness.

[4]The nanoManipulator is also capable of setting the stiffness displayed for a surface region to be equal to the value of some other data set measured in that region, which can include measured surface stiffness.

Finding the Tangent Plane to the Sample

Determining the tangent plane for contact between the effector and the environment is not particularly difficult in virtual environment applications (Adachi et al. 1995; Ruspini et al. 1997). However, it is much more difficult to apply this technique to remote manipulators, since the geometry of the environment is generally not known. The nanoManipulator uses a simple method of estimating a tangent plane using two successive samples and an assumed "up" direction. Since the AFM perceives the sample as a height field, this assumption is valid; the method gives good results in practice and is presented in Taylor (1994). This method is particularly sensitive to measurement noise; when investigating some materials, force feedback is nearly unusable due to the noise in the microscope imaging that substance (Figure 4.2). The tangent plane estimation method also derives some of its utility from the fact that samples being examined in an SPM can be treated as height fields: there are no truly vertical regions of the surface or overhangs reported by the SPM. (A height field is an array of values each of which represents height above some zero plane. Height fields have been widely used to approximately represent terrain, but can exactly represent the data returned by the SPM.)

4.4.3 Total Environment Sampling

Using the plane approximation, the nanoManipulator runs a tight control loop similar to that of the classical DFF teleoperator: move to one point in the environment, measure the impedance,[5]report that impedance to the user as a force, and allow them to specify the next destination, "close the loop," by moving to that location.

The general approach that I call total environment sampling begins with a manipulator automatically building a model of the environment as a pre-process. The model can be used to drive force feedback, or it can be used as the basis for a plan developed off-line and sent to the manipulator to be automatically executed. This solves some of the problems faced by automation and hierarchical control (see section 4.5). As implemented in the nanoManipulator, this is known as "feel-from-grid mode." The microscope first makes an imaging pass over the specimen, to determine the complete topography of an area of the specimen. Measurements are taken on a regular grid,

[5]The nanoManipulator normally directly measures location, not impedance. However, the force exerted on the AFM tip is known, so the impedance can be computed, and the user typically perceives himself to be feeling the impedance in the sense of Funaya and Takanasi (1993).

typically 300 x 300 points. Once these measurements have been reported to `nano`, `topo` is not involved in the force feedback – when the user moves the PHANTOM, the topography of the corresponding location in the model of the environment is used to compute the plane approximation.

There are two significant problems with total environment sampling applied to the nanoManipulator. First, because of the nonlinearities and drift in the SPM scanner (section 2.1), the environment model is displaced or distorted; it is not correctly registered with the true environment. Manipulations planned with reference to this model will be incorrectly executed by the manipulator.[6] Second, some classes of samples – particularly biological substances – will be inadvertently modified by the imaging pass, which makes the environment model invalid even before it is fully constructed.

4.5 Teleoperation without Force Feedback

Because excessive latency in the feedback loop can cause instability (Ferrell 1966), most researchers trying to do long-distance teleoperation have abandoned force feedback. By "opening" the control loop, destructive instability is avoided; instead, latency causes time-dependent position errors. Many open-loop teleoperators attempt to compensate for the position errors caused by latency, as well as to compensate for the fact that, without force feedback, operators cannot bring their reflexes and kinesthetic sense to bear on the problem nearly as effectively.

For example, Goktas et al. (1997) provide one example of an open-loop teleoperator designed to minimize latency-induced errors. Even though open-loop teleoperation should be significantly more stable than operation with force feedback, the random latency of the Internet still gives them a great deal of trouble – delayed commands mean that manipulators move jerkily. To avoid random latency, Goktas et al. use a network that provides QoS guarantees, allowing them to limit the worst-case delay. Once the delay is bounded, they can design a controller that assumes worst-case delay, is stable, and minimizes error under that delay condition. They conjecture that this approach might also be applicable to force-feedback teleoperators, but would

[6]The locations measured in the imaging pass correspond to particular voltages applied to the scanner. The nonlinearities discussed in section 2.1 mean that applying the same absolute voltage to the scanner with a different first derivative will produce a different position.

require that surface stiffness be kept inversely proportional to latency. This is the same limitation that caused us to look for solutions other than passive control for the nanoManipulator's force feedback interface.

4.5.1 Automation

If one removes the human completely from the real-time control loop, teleoperators become robots, executing programs devised offline. This is a good solution for automating tasks that are well understood and need to be repeated many times.

A group at the University of Southern California (USC) has done automatic nanomanipulation of a gold particle using an SPM (Resch et al. 2000; Meltzer et al. 2001). The USC experiments are similar to manual manipulations carried out five years previously using the nanoManipulator (Finch et al. 1995). The nanoManipulator group at UNC has considered using automation for construction of microelectromechanical systems and other repetitive tasks (Robinett 1998), but finds it ill-suited to experimental science. Automation relies on an understanding of the environment in which the effector operates; nanoscience is interesting precisely because the physics is not well understood. The original impetus for the nanoManipulator was that automation was not serving scientists well. The intent of the nanoManipulator is to give the scientist a more central role in the control loop, to enable enough analysis and understanding as the experiment was carried out that the scientist could replan mid-stream (Taylor et al. 1997).

4.5.2 Hierarchical Control

Automation requires too much foreknowledge for many tasks in the real world. To automate a teleoperator, the programmer needs both an understanding of the execution environment and a complete plan for the task. If any part of the task is unknown or uncertain, automation is not completely feasible. Developers of satellite repair teleoperators, for example, face this kind of task under very high latency; hierarchical control is one method they use to combine human supervision with as much automation as feasible (Brunner et al. 1995).

Latency destabilizes teleoperation when the human operator is controlling every tiny movement of the manipulator, commanding positions and sensing forces. In hierarchical control, or teleprogramming, the operator sends more abstract commands – "move to this position and orientation," "pick up this bolt" – which are executed by a fast control loop local to the manipulator. This approach has two principal requirements: a rich set of sensors whose output can be processed automatically to drive a feedback loop at the manipulator (Lumelsky 1991) and a well-understood operating environment that can be explicitly modelled. There has been some success at abstracting environments so that implicit models can be constructed and programming-by-example can be used (Brunner et al. 1995).

4.5.3 Environment Modeling

Some force-feedback systems use Total Environment Sampling; similarly, a number of non-force-feedback systems attempt to build a geometric model of the unknown environment they are operating in. In systems like Milgram et al.'s (1995) ARGOS, the hope is that as the model is built it can be used to incrementally create programs: that after an initial period of exploration, experimentation, and teaching-by-example the operator can find an automated solution to his problem and take up a supervisory role (Rastogi et al. 1996). ARGOS uses stereo cameras to captures pictures of the environment, then applies image-processing algorithms to build a geometric model of it. Once the user has captured and verified the model, he can use it to plan his manipulations and execute them under program control without force-feedback.

In the nanoManipulator, the manipulator – the microscope tip – is the only sensor we have to explore the environment. For well-known environments, this may be sufficient to build a model. For example, some work has been done using image processing to automatically recognize carbon nanotubes from their SPM scans and build geometric models of them. A nanoManipulator that uses a combined AFM/SEM (Scanning Electron Microscope) has also been constructed; the SEM gives us a video-camera-like image of the surface at magnification sufficient to display the SPM's working volume. An ARGOS-like system for the AFM/SEM nanoManipulator is currently under development, using video images to monitor the sample while the microscope tip is being used to manipulate (Taylor et al. 2003).

4.6 Relating Direct Force Feedback to the Plane Approximation

To date, all prior studies on latency bounds in force-feedback systems have been conducted with position-controlled DFF systems. The plane approximation was introduced to provide *indirect* force feedback, less sensitive to latency. Thus, the results from position control studies do not directly tell us anything about the latency tolerance of applications that use the plane approximation.

Azuma and Bishop (1995) used Fourier analysis to compare the theoretical error performance of various predictors of the motion of a head tracker. A Phantom is also a tracker – a device that measures and reports the position and orientation of some object. Fourier techniques can be used to compare the theoretical error performance of position control and the plane approximation. First, we need to specify an error metric.

4.6.1 Haptic Error Metrics

When Azuma and Bishop (1995) studied the error performance of algorithms to predict the motion of user's heads, the error metric was simple: error was the distance between the predicted position and the actual, measured position. However, measuring error in human perceptual processes is not straightforward.

Adachi et al. (1995) briefly introduced an error metric $d()$. When the user is feeling a plane approximating the surface, the error is the distance perpendicularly from that plane to the surface (Figure 4.3). This is a psychologically plausible measurement. The user can be assumed to be exerting force approximately perpendicularly to the plane; if the plane is updated without any smoothing, the user's hand will fall through the distance $d()$ before reaching the new plane. Adachi et al. reported the Just-Noticeable Difference (JND) for $d()$ – the minimum error that was noticed by a human user – as 0.8 millimeters, given a surface stiffness of 10000 N/m. Establishing the JND is an important part of making an error measurement useful.

The Adachi et al.'s definition of $d()$ was with respect to the plane approximation, but it could reasonably be generalized to point-sampled position control. Instead of using the normal to the approximating plane, use the direction in which force is exerted.

For this dissertation I have developed an error metric I call Vertical Error, $v()$. We choose a "vertical" vector for all measurements, and measure error along it. This vector is approximately perpendicular to the surface.[7] Although $v()$ is not as psycho-

[7]This condition is most reasonable when dealing with force feedback over a height field. For the nanoManipulator, this vector can be either the "up" vector internally chosen by the microscope, or the perpendicular to the surface estimated by the nanoManipulator's "flattening" operation (Taylor 1994), which skews the surface to compensate for apparent errors in the microscope's chosen up

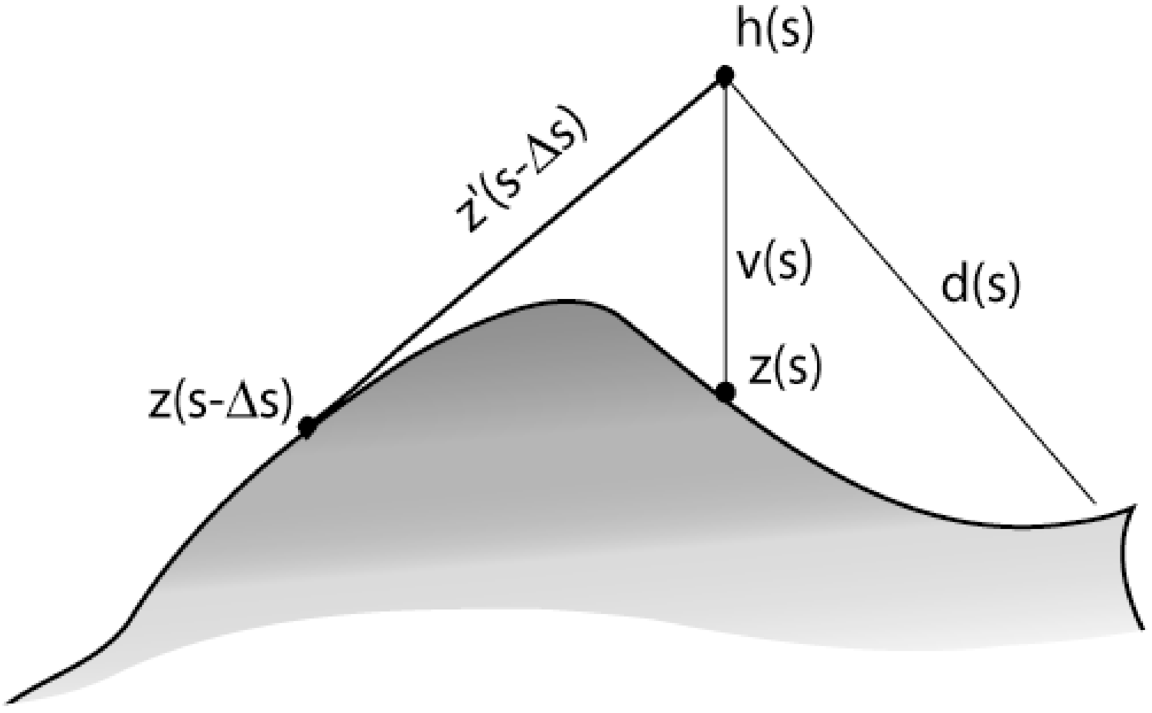

Figure 4.3: Adachi's error metric, $d(s)$, and Vertical Error, $v(s)$. The figure shows a nanoManipulator user travelling along a path parameterized by s; the previously sampled value was $z(s - \triangle s)$ with approximating plane slope $z'(s - \triangle s)$. When a new sample is returned at $z(s)$ with approximating plane slope $z'(s)$, the user's hand is at position $h(s)$.

logically motivated as $d()$, it is much more tractable under Fourier analysis.

4.7 A Frequency-Space Analysis of Force-Feedback Error

Fourier analysis decomposes a signal into a sum of an infinite number of sinusoidal waves, where the waves are described by their frequency in cycles per second and their magnitude in appropriate units of power. I treat a path over a surface in Fourier terms as follows: First, parameterize the path with respect to path length s. The sequence of points that make up the path can be treated as three independent functions:

vector.

$x(s)$, $y(s)$, and $z(s)$. Any one of these functions is the sum of an infinite number of sinusoidal waves with a "spatial" frequency σ measured in cycles per meter and a magnitude in squared centimeters. If most of the energy in the power spectrum of a path is at low spatial frequencies, the path is relatively smooth; if it has substantial energy at high frequencies, the path is relatively rough.

For the nanoManipulator, the x and y specified by the user are an input, and the z at that point is measured by the microscope. Thus, in the remainder of this chapter, as well as when we return to the question of analyzing haptics in chapter 6, I analyze z.

Control theory models systems as transforming an input function to an output function. The system's **transfer function** is the quotient of output and input. If the Fourier transform of the height measured by the microscope of points along the path as $Z(\sigma)$, but the transform of the sequence of heights displayed to the user is $H(\sigma)$, then the transfer function of the force feedback is $\frac{H(\sigma)}{Z(\sigma)}$. The **magnitude ratio** of the transfer function is $\frac{\|H(\sigma)\|}{\|Z(\sigma)\|}$. The **phase shift** of the system is another descriptor of the difference between the output and the input, denoted $\tan \alpha - \phi$.

In this section, I compute the magnitude ratios and phase shifts of the transfer functions of DFF and of the plane approximation. Magnitude ratio and phase shift are both functions of the quantity $\sigma \triangle s$, which is the product of the prediction distance ($\triangle s$) and the spatial frequency of the path (σ). The prediction distance is the product of the response time of the force feedback system and the speed at which the user is moving. Given an input signal – the shape of the path – the magnitude ratios let us compute the shape of the power spectra of the output and error signals, which are indicators of the relative smoothness of the path the user will actually feel. Since the inputs are sampled, finite signals, we only have estimates of their power spectra, and thus are only estimating the shape of the output spectra; section 4.8 shows how these estimates are consistent with our experience running the nanoManipulator in a variety of distributed configurations.

4.7.1 Fourier Analysis of Direct Force Feedback

Teleoperation is most often implemented as position control: the user moves the force feedback device to a position, the computer measures the position, and the device responds with some force. Since the device's output is a force, it is necessary to measure errors in position control force feedback in units of force.

However, there is also force control teleoperation: the user exerts a force on the device, the computer measures the force, and responds by moving the device to some position; the user feels the device at this new location. We can then measure error in distance: the distance between the position the user feels and the position the user should feel in a zero-latency system. Although force control and position control are dual (opposite) implementation techniques, in use they have equivalent performance (Anderson and Spong 1989). Thus, although I analyze the performance of force control teleoperation, the results apply equally to position control.

In Azuma and Bishop's (1995) language, a DFF force control interface with latency $\triangle t$ is a zeroth-order predictor: the felt position at a time t, $h(t)$, is equal to the position that was measured some time earlier, $z(t-\triangle t)$. Vertical error is the difference between felt position and measured position: $v(t) = h(t) - z(t) = z(t - \triangle t) - z(t)$.

Instead of parameterizing with respect to time, consider a parametrization with respect to path length s. During latency $\triangle t$ the user moves a distance $\triangle s$. Assume that $\triangle s$ is constant.[8] After this reparameterization, we still have three functions: $z(s)$, $h(s)$, and $v(s)$ – measured, felt, and error signals (Figure 4.3). Now use Fourier techniques to transform into the frequency domain, such that $z(s) \Rightarrow Z(\sigma)$.[9] In the next section I perform the same calculations for the plane approximation, then compare the results.

measured height	$z(s)$
felt height	$h(s)$
vertical error	$v(s)$
transfer function magnitude	$\frac{\|H(\sigma)\|}{\|Z(\sigma)\|}$
error magnitude ratio	$\frac{\|V(\sigma)\|}{\|Z(\sigma)\|}$
phase of input $Z(\sigma)$	ϕ
phase of felt height $H(\sigma)$	α
phase shift	$\tan(\alpha - \phi)$

$$\begin{aligned} h(s) &= z(s - \triangle s) \\ v(s) &= z(s - \triangle s) - z(s) \end{aligned}$$

[8]This requires that the user moves at a constant speed and that $\triangle t$ is constant.

[9]Appendix C has the details of these calculations.

$$\begin{aligned}
H(\sigma) &= Z(\sigma)e^{-j\sigma\triangle s} \\
V(\sigma) &= Z(\sigma)(e^{-j\sigma\triangle s} - 1) \\
\frac{\|H(\sigma)\|}{\|Z(\sigma)\|} &= 1 \\
\frac{\|V(\sigma)\|}{\|Z(\sigma)\|} &= \sqrt{2 - 2\cos(\sigma\triangle s)} \\
\tan(\alpha - \phi) &= -\sigma\triangle s
\end{aligned}$$

4.7.2 Fourier Analysis of the Plane Approximation

Parameterizing with respect to path length s, let $z(s)$ be the height in meters of the surface measured by the SPM, transformed into the space in which the user is feeling; let $h(s)$ be the height of the surface felt by the user. $v(s)$ be the error in the force feedback, $v(s) = h(s) - z(s)$.

If the user has moved $\triangle s$ along the path since the last sample, they have moved a distance $\triangle x$ in space, where $\triangle x \leq \triangle s$. Using the plane approximation, we can say $h(s) = z(s - \triangle s) + \triangle x\ z'(s - \triangle s)$; since $\triangle x \leq \triangle s$, we can consider the worst case by assuming that the path is a straight line segment and writing $h(s) = z(s - \triangle s) + \triangle s\ z'(s - \triangle s)$.[10]

Transformed into the frequency domain, these become $H(\sigma)$ and $V(\sigma)$:

$$\begin{aligned}
h(s) &= z(s - \triangle s) + \triangle s\ z'(s - \triangle s) \\
v(s) &= z(s - \triangle s) + \triangle s\ z'(s - \triangle s) - z(s) \\
H(\sigma) &= Z(\sigma)e^{-j\sigma\triangle s}(1 + j\sigma\triangle s) \\
V(\sigma) &= Z(\sigma)((\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s) - 1) \\
&\qquad + j(\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s)))
\end{aligned}$$

These equations have the same general form as Azuma and Bishop's (1995), and the expected properties that $\lim_{\triangle s\to 0} H(s) = Z(s)$ and $\lim_{\triangle s\to 0} V(\sigma) = 0$. Azuma and

[10]The nanoManipulator does not directly measure z', but approximates it based on the last two samples $x(s)$ and $x(s - \triangle s)$. First compute the approximate surface normal at s, $\tilde{n}(s)$:

$$\tilde{n}(s) = ((x(s) - x(s - \triangle s)) \times (0, 0, 1)) \times (x(s) - x(s - \triangle s)).$$

$z'(s)$ is the z-component of $\tilde{n}(s)$. As mentioned in section 4.4.2, this works well in practice, and so in the development of this fourier analysis I assume that it is a reasonably accurate approximation.

Bishop (1995) looks at the magnitude ratio and phase shift of the transfer functions to understand their frequency-dependent behavior. For the plane approximation, these are:

$$
\begin{aligned}
\frac{\|H(\sigma)\|}{\|Z(\sigma)\|} &= \sqrt{1 + (\sigma \triangle s)^2} \\
\frac{\|V(\sigma)\|}{\|Z(\sigma)\|} &= \sqrt{2 + (\sigma \triangle s)^2 - 2\cos(\sigma \triangle s) - 2\sigma \triangle s \sin(\sigma \triangle s)} \\
\tan(\alpha - \phi) &= \arctan\left(\frac{\sigma \triangle s \cos(\sigma \triangle s) - \sin(\sigma \triangle s)}{\cos(\sigma \triangle s) + \sigma \triangle s \sin(\sigma \triangle s)}\right)
\end{aligned}
$$

For any given $\triangle s$, the ratio will be larger for large σ than for small. The units of σ are meters^{-1}; large σ means small features on the surface. Thus, any system using the plane approximation magnifies the effects of small details of surface shape. The error to signal ratio is larger when the surface has more small features. We can draw the same qualitative conclusion as Azuma and Bishop (1995): "it is important to keep the prediction interval small and avoid high-frequency signals." $\triangle s$ is our prediction interval, σ the signal frequency.

Much like Azuma and Bishop (1995), we see a recipe for a constant performance level, but one with an additional degree of freedom: all of the above expressions depend on $\sigma \triangle s$. $\triangle s$ is the product of the speed at which the user moves and the response time of the system; when response time is high, the user can move more slowly and get the same performance. The bandwidth of σ expresses the roughness of the surface: for rough surfaces, i.e. surfaces with small features, $\triangle s$ must be reduced – response time, speed of movement, or both – to maintain performance.

Unfortunately, the AFM signal includes high-frequency noise. Regardless of the true bandwidth of the path, the mechanical noise in the AFM will add additional power to the high frequencies of the spectrum. Furthermore, this analysis only confirms that these systems are inherently not stable. Control theory tells us that a system is stable if its gain is < 1 when the phase shift is $\pm 180°$, but neither DFF nor the plane approximation ever have gain less than one (that is, $\frac{\|H(\sigma)\|}{\|Z(\sigma)\|} \geq 1 \forall \sigma$). Even if they behave stably for most inputs, there exists some input which can destabilize them.

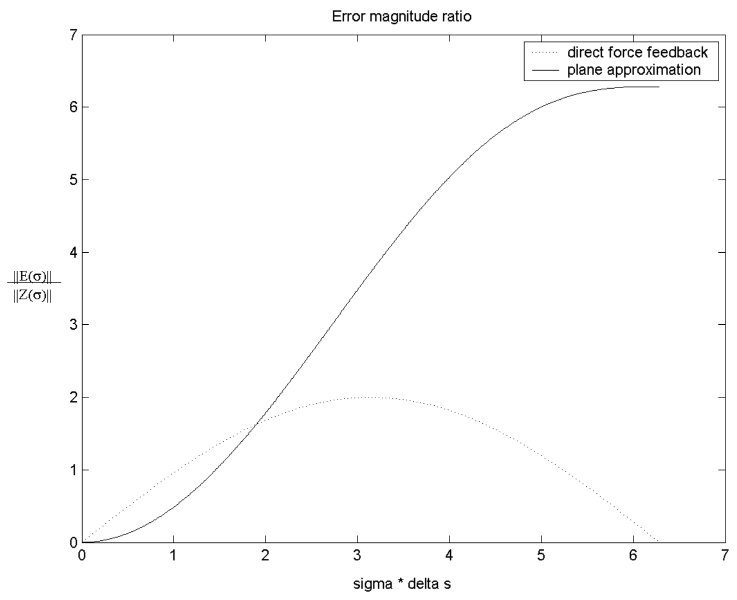

Figure 4.4: Error-magnitude ratio of Direct Force Feedback and the Plane Approximation. So long as the energy in the power spectrum of the path is concentrated below the crossing point of the graphs, the plane approximation will have less error than a DFF implementation. The horizontal axis is in terms of σ, the frequency component of the path, and $\triangle s$, the sample interval, which is the product of path speed and $\triangle t$, the system latency.

4.8 Verifying the Theory

Consider the shapes of these two error magnitude ratios (Figure 4.4). The horizontal axis is $\sigma \triangle s$, which is dimensionless. The spectrum of the output error is the product of the spectrum of the input (the path spectrum, scaled by response time and user speed) and the error magnitude ratio.

The plane approximation has a smaller error magnitude ratio than does force control below approximately $\sigma \triangle s = 1.9$. If the power in the spectrum of the path is concentrated below $1.9/\triangle s$ cycles/meter, we expect to have less error from the plane approximation than we would from DFF.

As shown in Figure 4.5, six paths over widely varied surfaces were all 100-fold below peak power by 70 cycles/meter. If 70 cycles per meter is the expected bandwidth for surfaces felt by the nanoManipulator, then the power in the input signal will be concentrated in the left-hand region of figure 4.4, where the plane approximation outperforms DFF, so long as the product of user speed and latency is no greater than 0.027 m.

In these same six paths, mean user speed ranges from 3 to 8 cm/s, with peak speeds between 20 and 30 cm/s. At 8 cm/s, the plane approximation outperforms DFF up to 337 ms of latency; at 30 cm/s, only up to 90 ms.

A nanoManipulator deployed on a local network has a haptic response time of approximately 125 ms. In our experience, this make DFF unusable, but is satisfactory with the plane approximation. At 125 ms latency, so long as speed does not exceed 22 cm/s the plane approximation should outperform DFF. This is reasonably consistent with the observed mean below 8 cm/s; this should be equivalent to 45 ms latency in DFF, which is well inside most stability bounds reported above. The observed peak speeds are on the verge of instability, but should only cause rare glitches.

When we ran the nanoManipulator over the Internet2 between Chapel Hill and OSU or Washington, DC (section 1.1), average response time was 160 ms, and the system remained usable. However, when we attempted to control a microscope in Chapel Hill from a workstation in Seattle, with average response time 195 ms, the system was unusable. At 195 ms of latency, with a mean speed below 8 cm/s, we would expect performance equivalent to DFF with 71 ms latency - not good, if perhaps still stable. However, the observed peak speeds of 20-30 cm/s are equivalent to 170 ms or more of latency in DFF, which would trigger the observed instability.

4.9 Summary

Haptics is an essential interface for planning and carrying out experiments using the nanoManipulator. The effective latency bounds for haptics are low, somewhere between 40 and 100 ms for DFF. However, the plane approximation has always provided stable and stiff operation at latencies in those range; Fourier analysis methods let us understand why and extrapolate DFF latency bounds to the plane approximation. Our observations bear out the Fourier analysis: the plane approximation allows teleoperation reliably at 120 ms latency, adequately at 160 ms, but not at 195 ms. This analysis supports my decision to develop new intermediate representations for haptic force feedback, reported in the next chapter.

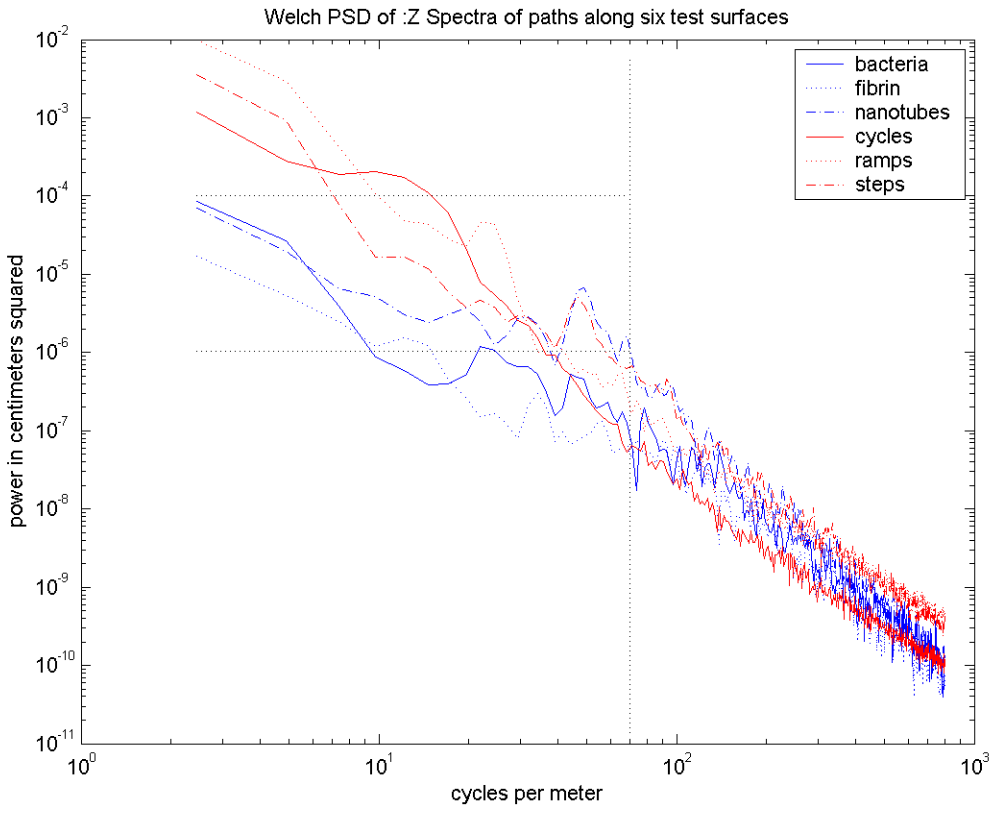

Figure 4.5: Estimated power spectra of measured component six paths along test surfaces, showing a falloff in energy with increasing frequency. All six surfaces were 100-fold below peak power by 70 cycles/meter.

Chapter 5

Haptics Beyond the Plane Approximation

In chapter 4, I built an analytical framework for determining how much more latency the plane approximation can tolerate than can position control. However, the plane approximation is not a complete solution to the problem of latency in haptics. In transcontinental or intercontinental use, the system response time can exceed 200 ms. In our experience, this high latency prevents the system from working acceptably.

In this chapter I build a taxonomy from the force feedback approaches previously discussed, and identify a productive, previously unpopulated region of the taxonomy. I propose two new force feedback modes in this region that enable effective operation even in the presence of high latency. Although these techniques are not well-suited to remote manipulation, they are useful for remote haptic sensing. I discuss some additional approaches to force feedback under high latency suggested by the taxonomy that were not deployed in the nanoManipulator but may be useful in similar systems. Finally, I discuss an experiment conducted to evaluate these new force feedback modes, including its design, its results, and its shortcomings.

5.1 Framework

Milgram et al. (1995) created a three-dimensional taxonomy of teleoperation modes: the degree to which a manipulator is autonomous, the degree to which a remote environment is known in advance ("modelled"), and the degree to which the properties and geometry of the constituent parts of or objects in the remote environment are known in advance ("structured"). For the distributed nanoManipulator, the design purpose is investigation of novel physical systems, and so this discussion assumes an unmodelled, unstructured environment, and considers how much autonomy is possible in such environments (Table 5.1).

Table 5.1 reveals that the degree of automation in teleoperation closely parallels the complexity of of the network representation used to transmit feedback to the haptic device. At one extreme are traditional teleoperation approaches with direct human control, suitable for unmodelled and unstructured environments, which have a low tolerance of latency and use instantaneous measurements of force and position transmitted at high frequency as the network representation. At the other extreme lies robotics, with no human in the control loop; this approach has a high tolerance of latency, but requires a completely modelled and structured environment; the network representation is abstract commands and status reports transmitted at low frequency.

The techniques discussed in chapter 4 and summarized in Table 5.1 use a number of different types of intermediate representation (in the "Network Representations" column). The two new intermediate representations presented in this chapter occupy the (previously unfilled) middle of Table 5.1's continuum, enabling *human* control of the teleoperator in *high-latency* (O(1 second)) situations.

Network latency is the chief obstacle to remote use of the nanoManipulator's force-feedback capabilities. In order to combat the effects of latency, we can afford to trade off other system parameters, such as network bandwidth or measuring capability at the microscope, so long as they are kept low enough to not become the system bottleneck. This is an instance of the classic tradeoff between communication and computation. Table 5.2 lists a number of adaptations which are discussed in this chapter and the next. The adaptations are classified by their place in the OSI network model, discussed in section 3.1.

5.2 Novel Intermediate Representations

The two new intermediate representations – local environment sampling and the warped plane approximation – were explicitly designed to make the tradeoff between computation and communication. Local environment sampling requires increased computation and measurement time at the server, as well as increased message size, but reduces the rate at which messages need to be transmitted and their sensitivity to latency. The warped plane approximation requires that the client monitor network latency to construct explicit adjustments in the force feedback. Both techniques are well-suited to haptic sensing, where the user's goal is to understand the shape of the surface, but not to remote manipulation, where the user's goal is to change the shape of the surface.

5.2.1 Local Environment Sampling

Local environment sampling is an acceleration of total environment sampling, blending interactive and supervisory teleoperation (section 4.5.3). Local environment sampling attempts to make interactive control feasible at latencies that previously required supervisory teleoperation. Instead of using sensors or the effector to determine the shape of the *entire* environment, as does total environment sampling, local environment sampling samples only the area immediately around the teleoperator's effector. This yields a burst of up-to-date data. In its implementation, we choose to keep the sampled area very small, requiring little network bandwidth to transmit and enabling the planning of the immediately subsequent steps of the manipulation. With local environment sampling, the cycles of sense-transmit-plan-transmit-manipulate are much shorter and faster than those in total environment sampling. Increasing the sampled area permits construction of longer plans at lower frequencies, providing a parameterized continuum from interactive to supervisory manipulation.

For the nanoManipulator, the sensor is the end-effector: the tip of the AFM. The local environment sampling cycle can be run several times per second,[1] yielding data much less stale and much less distorted by nonlinearities than that from total environment sampling and thus a more accurate view of the sample (in a small area). Each of these cycles involves the user's Phantom position being sampled, a command sent to `topo` to examine the corresponding area of the surface, `topo` taking several measurements of surface height in that area, and these samples being transmitted back to `nano`, where they are displayed by the Phantom as a small "patch" of surface (figure 5.1).

In distributed computing, carefully choosing the balance between computation and communication is a classic problem. Local environment sampling is yet another instance: by increasing "computation" (measurement of the environment by the AFM),

[1]How often depends on characteristics of the sample, the microscope being used, and the scale at which and resolution with which we are creating this local model. For carbon nanotubes, a "typical" sample for our materials science collaborators, the microscope can scan a 25 nm square region at 5×5 resolution in approximately 175 ms.

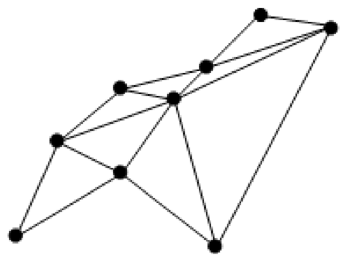

Figure 5.1: Samples taken and planar facets generated by local environment sampling. The central point is the position commanded by the user; the microscope autonomously samples the eight adjacent points and sends them back to the user with the central point. The user then feels the set of planar facets formed from those nine points until the next round-trip is complete.

we reduce the frequency of communication necessary to provide distributed haptics.

In the distributed nanoManipulator implementation for this dissertation, local environment sampling was used to feel the surface, not to make modifications. If the user leaves the small surface patch most currently measured by the microscope, force feedback cuts off until data for a new surface patch that contains her current location is received from the microscope. This forces users to move slowly "in the large," but they can make rapid movements across a small distance – an important distinction which permits users to recognize surface features on samples in the microscope.

We informally call local environment sampling "feelahead" mode, after the image of a blindfolded pool player feeling very gingerly with the cue to find out where the balls are.[2]

Future Work

The implementation of local environment sampling used in my dissertation was fixed and non-adaptive - adaptation was in choosing to use this technique instead of the plane approximation or the warped plane (section 5.2.2). However, local environment sampling could benefit from additional adaptation. Assume that we have a default $25nm \times 25nm$ local sample region. When the user is moving the Phantom slowly, or

[2]The general task of manipulating nano-scale objects has been likened to feeling among thin plastic bags full of jello using a screwdriver in the dark (Finch et al. 1995).

studying to some detail, they are not likely to leave this region for several seconds. When they are moving the Phantom rapidly, to change which portion of the sample they are working in or because they perceive no relevant features in the current area, they will may leave a local sample region faster than it can be measured and sent from the AFM. Thus, for a slow-moving user we could iteratively refine our model of the area, in a manner similar to Funaya and Takanasi (1993), while for a fast-moving user we ought to predict her movements so that we can scan a local sample region of relevance, or scan a large, low-resolution local sample area.

5.2.2 The Warped Plane Approximation

Transport-level adaptations can reduce the latency seen by a force-feedback teleoperator, allowing the plane approximation to work well over a wider range of networks. The plane approximation still assumes zero latency, so if latency is high it fails, presenting an incorrect surface shape (as shown in section 4.7). I introduce a variant, the warped plane approximation, which explicitly measures and compensates for positioning error due to latency. This yields a haptic representation that preserves the shape of the sample, making features recognizable, but does not preserve the relative location of features when latency varies.

When a new approximating plane arrives from the microscope, the plane approximation displays it at the position it was measured. It can be thought of as a Taylor series expansion of the shape of the surface at this point – valid within a small neighborhood, but with increasing error as distance from that point increases, roughly proportional to the second derivative of the (real) surface shape. When the plane arrives, the Phantom is typically offset some small amount from the point at which the approximating plane was measured due to the user's movement and the latency of the AFM's measurement. However, increased network latency increases the offset, and thus increases the expected error (figure 5.2c).

With the warped plane approximation, the nanoManipulator maintains both a record of the recent path of the Phantom and an estimate of the current network round-trip time. New approximating planes arriving from the microscope are not displayed at the position they were measured, but are "warped" along user's the path a distance proportional to the fraction of the total latency due to network delay. Thus, the offset between the current Phantom position and the location at which the plane is displayed is only the offset expected due to the user's Phantom speed and microscope latency, independent of the network latency, and a "reasonable" feature

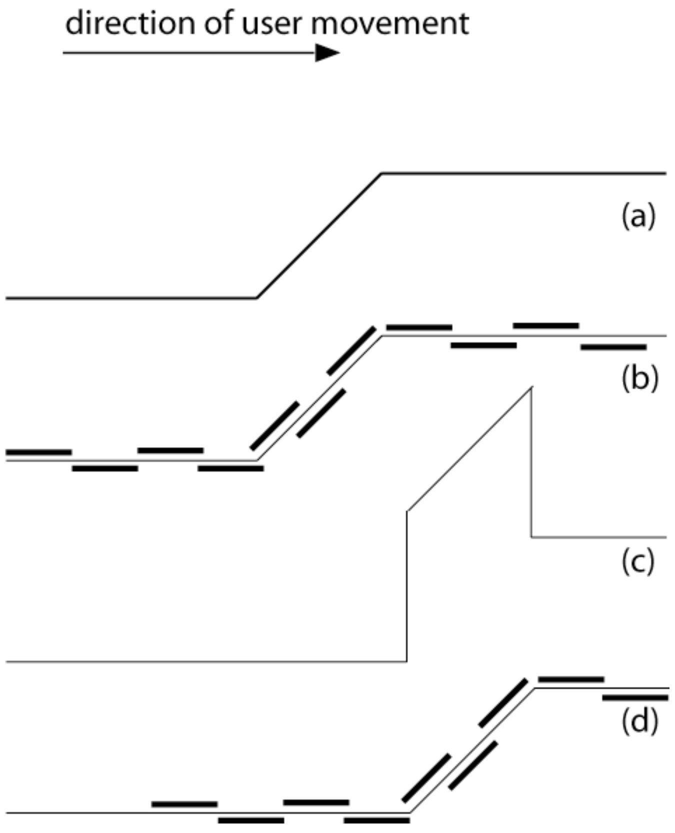

Figure 5.2: The warped plane approximation. For a surface with shape (a), the microscope generates a series of planes (b). The plane approximation displays these at the position they were sampled; when latency is high, this produces an incorrect surface shape (c). The warped plane approximation compensates for latency, maintaining shape while losing position (d).

shape is displayed (figure 5.2d). Sequential planes have equivalent offsets so long as user speed and network latency remain constant. If the user stops moving the Phantom on some feature of interest, the feature seems to "slide away" under their hand, and they have to backtrack to find it.

5.3 Adaptations in the User Interface

A number of user interface adaptations used in traditional teleoperation applications can be applied to force-feedback for the distributed nanoManipulator, as can latency-compensation techniques developed for other kinds of user interfaces. After preliminary experimentation, we chose not to deploy these methods, but they may be of use for other force feedback interfaces coping with high response time.

- Reduced stiffness: A force-feedback control system must have high gain and high bandwidth[3] to display stiff surfaces. However, gain and bandwidth are both inversely related to stability. To keep a system stable under high latency, we can reduce the system gain, reducing the stiffness of the surfaces felt, or reduce the system bandwidth, which reduces the manipulator's velocity (Vertut et al. 1981). This leads to mushy, indistinct haptics; it has proved adequate for some teleoperation tasks, but has not satisfied users during typical nanoManipulator experiments. The lower the stiffness of the force feedback, the harder it is for the scientists to find features on the surface and use them to guide modifications.
- Predictive displays: Avionics has used measured or computed derivatives – higher-order information – to display to the pilot the projected state of the aircraft at some future point in time (Wickens 1986); Distributed Interactive Simulation uses similar prediction to explicitly compensate for network latency (section 3.3.3). Teleoperators using satellite-mounted robot arms have guided their efforts based on a display of where the arm is measured to be as well as where the operator has commanded it to move. Applying this to an AFM, we can display to the user both the position to which she has commanded the microscope tip to move and the most recently known location of the tip. This gives the user a sense of the total latency in the system – from AFM dynamics and from network delay – and lets her change her behavior to compensate. nanoManipulator users have not found predictive displays useful; in our experience, they appear to be well-suited to planning applications, but not to sensing applications. As discussed in chapter 2, the nonlinearity, hysteresis, and creep of the AFM makes planning of little use, and require a human being using their senses to actively control manipulations. One of the goals of the scientists using the nanoManipulator is to leverage their kinesthetic and proprioceptive senses to understand the shape of a specimen, which is not aided by a predictive display.

[3]Unfortunately, the control theory or signal analysis and networking communities use the term "bandwidth" differently, and neither has an accepted alternate term. When discussing stability, bandwidth should be taken in a signal analysis sense, to mean the range of frequencies of signal that can be transmitted. Generally, transmitting a higher signal bandwidth requires more network bandwidth, but the former is measured in cycles per second (or, in section 4.7, in cycles per meter) and the latter in bits per second.

5.4 Comparing Intermediate Representations

Chapter 4 discussed metrics for evaluating the fidelity of force feedback systems. Neither Adachi et al.'s $d()$ nor my $e()$ (Figure 4.3) is applicable to the warped plane approximation, because it attempts to preserve local surface shape rather than surface position.

After simulating an "ideal" force feedback system, I ran a series of trials of the plane approximation, the warped plane approximation, and local environment sampling under a variety of different network loads and end-to-end latencies.

The evidence of usefulness for these techniques rests on the testimony of expert nanoManipulator users, who found the new representations to present a clearer sense of the surface shape than did the plane approximation. Although I summarize the trials I conducted, our numerical evaluations of them were inconclusive, serving more to underscore the limitations of the previously-established metrics than the potential of the new representations.

5.4.1 Experimental design

To create an "ideal case" force feedback, I used total environment sampling to create a canonical representation of the surface. Force feedback using the total-surface representation stored at `nano` eliminates network delay, instrument delay, and microscope noise from the haptics. I recorded a path taken by a user feeling on this total environment sampling surface; the set of planes displayed by `nano` along this path is the "correct" force feedback against which the various network representations were compared.

In looking at the vertical error between the sensed surface and the ideal case, I considered Root Mean Square (RMS) error, the distribution of RMS error, and the rate of errors above two thresholds (chosen to approximate minimal perceptible error and error at which the PHANTOM would detect apparent instability and cut off). I also considered errors in orientation of the surfaces displayed. None of these analytic approaches were adequate.

The experiment included six fixed surfaces. Three were mathematical constructs based on standard waveforms from control theory, intended to informally determine the response of the haptic system. The other three were data from real experiments, to determine the performance of the system in actual use. A separate user path was recorded for each surface, lasting between 60 and 90 seconds. Each surface has a short name by which it will be referred to in the experimental results:

Steps A series of small, nearly-vertical steps separated by flat regions (10 nm high at 50 nm horizontal intervals). The surface was 300 nm square, sampled at 1 nm intervals. Taller steps were not used because they caused instability in the plane approximation at high network latencies.

Ramps A series of 45° ramps separated by flat regions (50 nm high at 50 nm horizontal intervals). The surface was 300 nm square, sampled at 1 nm intervals.

Circles A circular, concentric sinusoidal pattern (20 nm high with period of 40π nm), centered on the surface. The surface was 300 nm square, sampled at 1 nm intervals.

Viruses A scanned surface covered with a random scattering of icosahedral virus capsids (Falvo et al. 1997).

Fibrin A scanned surface with a few strands of fibrin, a biological substance that forms long, winding tubes (Guthold et al. 2000). Clumps of fibrin adhere red blood cells to make blood clots.

Nanotubes A scanned surface with a few carbon nanotubes, short stiff cylinders of interest to materials scientists (Falvo et al. 1997).

Regardless of their actual size, all of these surfaces were scaled to the nanoManipulator's standard working volume, yielding a human scale of about one square foot of surface.

For these experiments, I set up a test LAN: two 100 Mb networks connected through a 10 Mb bottleneck. The nanoManipulator software was deployed on one side of the bottleneck; a microscope simulator (Appendix B.1) was deployed on the other. This general scheme abstracts the Internet (figure 5.3).

A large number of simulated web browsers ran on the same side of the bottleneck as `nano`; web servers ran on the other side of the bottleneck. By varying the number of browsers, we could vary the load on the bottleneck (Christiansen et al. 2001). Prior experiments with this apparatus had found that when large numbers of browsers were run, performance was irregular for roughly fifteen minutes of each run. Once it stabliized, 2700 browsers connecting to 3 servers produced a 90% load across the bottleneck link with 2% loss, and 4500 browsers connecting to 5 servers produced a 98% load with 10% loss. The actual hardware used is shown in figure 5.4.

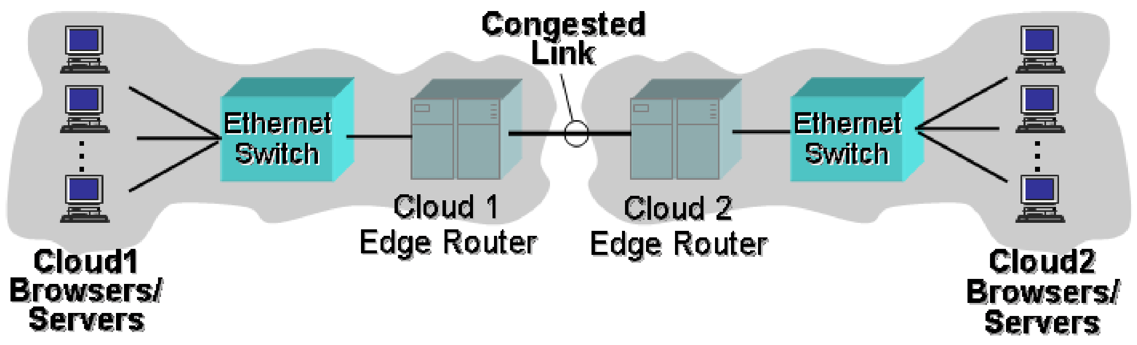

Figure 5.3: Conceptual diagram of an experimental network. A local campus (Cloud 1) connects to the Internet, where there is a congested "bottleneck" link. Clients in Cloud 1 are attempting to communicate with servers beyond the bottleneck in Cloud 2.

Six different sets of experimental network conditions were used, intended to simulate different situations in which the distributed nanoManipulator might be deployed. Each was characterized by a constant delay (added to the LAN by the dummynet package (Rizzo 1997)) and some additional loss (caused by cross-traffic from the web browsers). In each case, the traffic generators (web servers and simulated web browsers) were allowed to run for 30 minutes before the distributed nanoManipulator test began, producing the expected stable network conditions. The six conditions were:

1. An unloaded LAN (the nanoManipulator's original environment): 0 ms of latency, 0% loss.
2. A congested campus or metropolitan-area network: 0 ms of latency, 10% loss.
3. A three-state network with QoS guarantees: 30 ms of latency, 0% loss.
4. A congested, three-state segment of the Internet: 30 ms of latency, 10% loss.
5. A cross-country network with QoS guarantees: 90 ms of latency, 0% loss.
6. A congested, cross-country segment of the Internet: 90 ms of latency, 10% loss.

Under these six different conditions, I replayed each surface's recorded path for three different intermediate representations – plane approximation, warped plane approximation, and local environment sampling– under the six network conditions. The

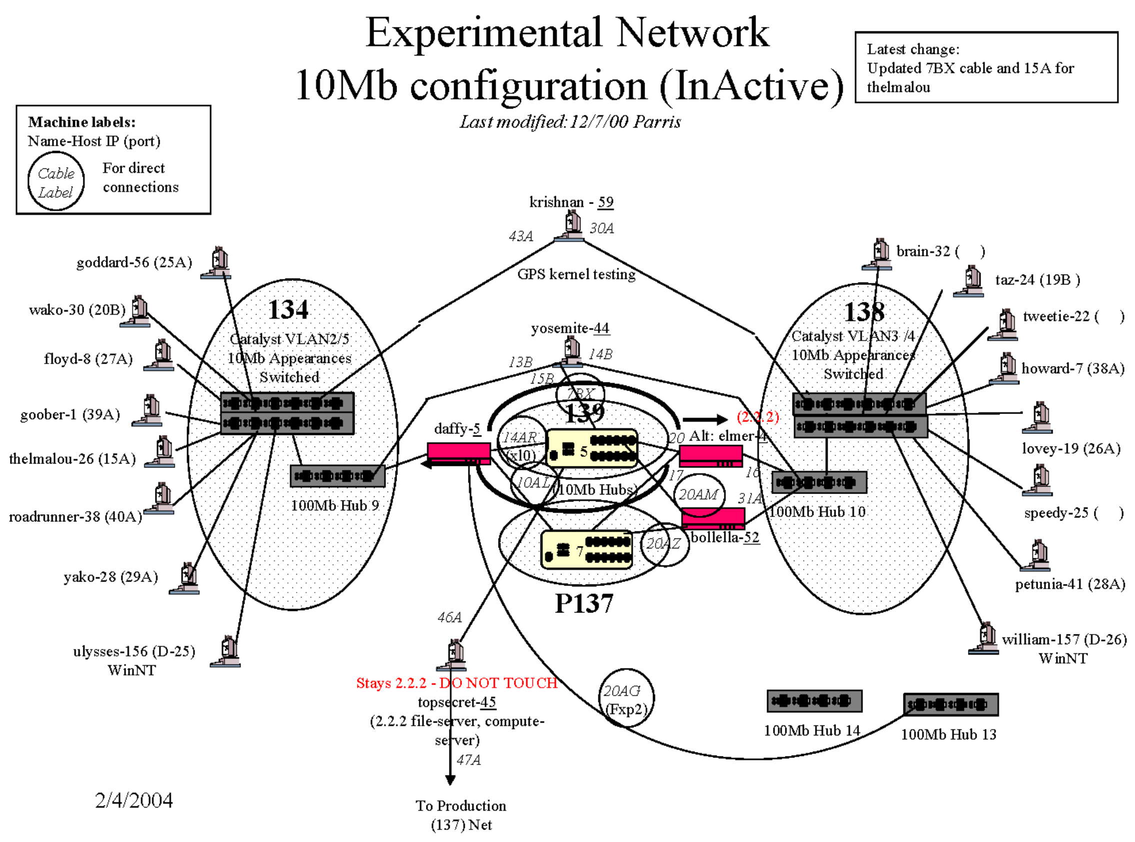

Figure 5.4: Implementation map of an experimental network. The “134” and “138” switched subnets are separated by two routers (red) and two hubs (yellow) configured to act as a full-duplex 10 Mbps bottleneck.

commands that the `nano` process sent to the PHANTOM to create force feedback were logged; this log was then compared to the ideal case.

This experimental approach – replaying one set of user inputs in a number of different environments – is controllable and repeatable, but suffers from the flaws discussed by Bhola and Ahamad (1999). Real users would, consciously or not, adjust their behavior to adapt to perceived changes in latency. A study that uses automated traces of behavior ignores this fact and is not recreating the actual conditions of use; a study that includes this phenomenon requires a high degree of repetition across many subjects to attain repeatability. A user study would be more appropriate if we had a precise task to perform and a task-relevant metric, but where instead we want to quantify the response of the various representations to the same consistent input, automated analyses should suffice.

The inconsistency of user behavior with recorded traces noted by Bhola and Ahamad (1999) may help to account for the discrepancy between positive user reports and inconclusive numerical analyses. In Chapter 6 I report on the success of session-level adaptations in changing the network performance of force feedback in ways that should be perceptible to the user.

5.4.2 Experimental Results

The analysis in section 4.7 led me to expect that the vertical error for the plane approximation would be linear in $\sigma \triangle s$. I analogously tried treating vertical error as a linear function of the latency $\triangle t$, when there is no loss, the first-order estimates are shown in table 5.3.

These estimates, combined with figure 4.5, help verify the analytic framework discussed in section 4.7. They show no apparent difference between TCP and UDP in the absence of loss; since section 4.7 did not model loss, I did not attempt to make a fit that I could not justify. They show an increase in $v()$ with increased prediction distance, which is the product of latency and the speed with which the user moved. The nanotubes sample has an estimated 70 cycle per meter bandwidth, while both fibrin and bacteria have bandwidths below 30 cycles per meter. During the nanotubes experiment, the user moved roughly twice as fast as during the fibrin and bacteria experiments (8 mm/sec vs. 4 mm/sec). For all six experiments, the RMS error in the linear estimation of $v(t)$ was approximately one order of magnitude smaller than the smallest coefficient in the fit.

5.4.3 User Response

Because I had planned an experimental, quantitative evaluation of the new force feedback modes, user responses were only gathered informally. Scientists experienced with the nanoManipulator reported that both local environment sampling and the warped plane approximation let them feel the surface.

5.4.4 Experimental Considerations

I planned the experiment without any explicit replications. The infrastructure had been previously shown to produce stable, apparently reasonable latency and loss

(Christiansen et al. 2001), the six trials at each network condition were expected to be consistent and could thus be used to verify one another's performance, and each path had on the order of two thousand data points, so there was redundancy inside each trial.

I also planned the experiment without sufficient consideration of my intended analysis methods. This had two repercussions. First, $v()$ does not make sense applied to the warped plane approximation; neither I nor my committee have yet devised or discovered a quantitative metric that I can use to evaluate the idea of warping.

Second, even for the the modes that $v()$ was a reasonable metric, I did not verify that the originally-intended approaches to analysis would work. They did not; it was more than a year between the data gathering and the final determination of an acceptable analysis method. Only at this time did I realize that there was so much missing or bad data in the local environment sampling data that no conclusions could be drawn; the experimental network had been disconnected, preventing a repeat of the experiment to gain more data for local environment sampling.

5.5 Summary

This chapter introduced two new intermediate representations for force feedback, local environment sampling and the warped plane approximation. User evaluations of these new representations were positive, but experiments comparing them with the plane approximation were inconclusive.

Modality	Human Involvement	Required Understanding of Environment	Latency Tolerance	Network Representation
Traditional Teleoperation (Ferrell 1966)		none	none – O(1 ms)	instantaneous measurements of force and position
Passive Teleoperation (Anderson and Spong 1989; Niemeyer and Slotine 1998)	direct	none	low – O(100 ms)	instantaneous measurements and derivatives of force and position
Plane Approximation (Adachi et al. 1995)		minimal	low	first-order approximation of environment geometry
Warped-Plane Approximation		**minimal**	**moderate – O(200 ms)**	**first-order approximation of environment geometry, instantaneous measurements of network state**
Local Environment Sampling		**minimal**	**medium – O(1 second)**	**low-level commands; higher-order approximation of environment geometry**
Total Environment Sampling	indirect	moderate	high – several seconds	high-level commands; description of entire environment geometry
Hierarchical Control (Brunner et al. 1995)		high	high	high-level commands; description of entire environment
Automation / Robotics (Robinett 1998; Meltzer et al. 2001)	supervisory	very high	very high	programs (sequences of high-level commands); description of entire environment

Table 5.1: Implementations along a continuum of teleoperation modes, showing decreasing human control, increasing required foreknowledge of the environment the teleoperator is manipulating, and increasing tolerance of latency. The further down the list a mode lies, the more low-level control is removed from the human and automated by a computer located at the effector. Novel methods described in this dissertation are in boldface.

Network Layer	Technique
Application	Reduced stiffness or bandwidth (5.3)
	Predictive displays (5.3)
Presentation	**Warped plane approximation** (5.2.2)
	Local environment sampling (5.2.1)
Transport	**UDP** as a replacement for TCP (6.2)
	FEC (6.2)
	Queue Monitoring (5.2.2)

Table 5.2: Proposed adaptations for force feedback systems. Those in boldface were used in the experiments discussed in this dissertation.

Transport Protocol	Sample	$v(t)$ Estimate	RMS Error
TCP	Fibrin	$0.0015 + .00028\triangle t$	2.1×10^{-5}
	Bacteria	$0.0036 + .00067\triangle t$	4.4×10^{-5}
	Nanotubes	$0.0086 + .0022\triangle t$	3.0×10^{-4}
UDP	Fibrin	$0.0017 + .00029\triangle t$	6.3×10^{-5}
	Bacteria	$0.0032 + .00069\triangle t$	1.5×10^{-5}
	Nanotubes	$0.013 + .0021\triangle t$	3.6×10^{-4}

Table 5.3: Estimated $v(t)$ as a function of latency $\triangle t$ for TCP and UDP over the real surfaces. Fourth column is root-mean-square error of the fit.

Chapter 6

Network Adaptations for Haptics

Each of the force feedback modes discussed in chapter 5 – plane approximation, local environment sampling, and the warped plane approximation – is sensitive to the network's operating conditions. All three require bounded latency; local environment sampling is relatively sensitive to loss, while the warped plane approximation is relatively sensitive to jitter. In this chapter I explore how known techniques for adaptive audio and video applications can be applied to the haptic data streams produced by these three modes of operation. Some of this chapter's results were previously published in Hudson et al. (2001).

6.1 Characterizing the nanoManipulator's Haptic Data Stream

A standard teleoperator requires a low-bandwidth connection for control: for example, Kim et al. (1992) requires only a full-duplex 8-kilobyte per second stream. However, this stream consists of 1000 8-byte packets per second in each direction, each packet having significant computational overhead, and requires minimal latency.

When using the plane approximation (section 4.4.2), `topo` sends 20 60-byte packets per second reporting a position and a normal vector; hardware at the force-feedback device interpolates this series of planes to the 1000 Hz required to give convincing force feedback. The warped plane approximation (section 5.2.2) has a similar data rate, changing only how the data is processed.

Using local environment sampling, or "feelahead" mode (section 5.2.1), `topo` only transmits a few packets per second , each in excess of 300 bytes .

The data streams associated with these techniques are summarized in Table 6.1. Both the warped plane approximation and local environment sampling change the requirements of the `topo` ⇒`nano` link, but not the `nano` ⇒`topo` link. These techniques both improve the quality of the force feedback felt by the scientist using the tool, but neither directly addresses the problem of latency and jitter interfering with the control messages sent from the scientist to the tool. The networking experiments discussed in section 6.3 address this problem.

Data Stream	Endpoints	Bandwidth (bps)	Packet Rate (per second)	Latency Tolerance (ms)	Jitter Tolerance (ms)
Microscope Control	`nano` ⇒`topo`	10,880	20	$< O(100)$	O(100)
Plane-Approximation Point Data	`topo` ⇒`nano`	15,360	20	$< O(100)$	O(100)
Warped-Plane Point Data	`topo` ⇒`nano`	15,360	20	$> O(100)$	O(30)
Feel-Ahead Point Data	`topo` ⇒`nano`	15,120	5	$> O(100)$	$> O(100)$
Traditional Teleoperation	`topo` ⇔`nano`	*8,000*	*1000*	$< O(40)$	*O(0)*

Table 6.1: Peak bandwidth, packet rate, and latency and jitter requirements for haptic data streams in the distributed nanoManipulator for various latency-tolerance and adaptive networking techniques. All are "Manual" data streams in the sense of Table B.1, caused by the user taking direct, interactive control of the microscope tip during a manipulation. Network characteristics of a notional stereotypical teleoperator are given for comparison. Latency and Jitter tolerance are approximate values based on our experiences and the literature.

6.2 Network Transport Adaptations

The plane approximation is a simple improvement on traditional teleoperation approaches that works well with update rates around 20 Hz and latency under 100 ms (Table 6.1) (Adachi et al. 1995). When latency increases, instead of changing the representation or algorithm used, we can first look at changing the way the data is transported across the network. I used three approaches: replacing TCP with UDP to reduce latency (section 6.3.1), extending UDP with Forward Error Control (FEC) to reduce loss (section 6.3.2), and using Queue Monitoring to reduce jitter (section 6.3.3).

Replacing TCP with UDP significantly reduces both latency and jitter, but introduces loss. Since UDP does not guarantee delivery of packets, I was then faced with the problem of data loss. The nanoManipulator originally depended on the fact that no packet would be lost to log events at the client that occurred on the server.[1] I refactored the application so that logging could occur at either client or server; when server-side logging was used I could safely lose packets without compromising our records of the experiment.

In our experience with the nanoManipulator, force feedback using the plane approximation performs well even under moderate (e.g. 10%) loss. Like audio and video, 3d tracked sensors send data frequently but tracking applications are tolerant of losing the occasional update. However, local environment sampling, with its much lower message rate, is not nearly so tolerant: for our notional 25 nm square nanotube region, a single lost region message means three to four times as long without valid force feedback as does a single lost plane approximation update message. To control loss, I added FEC. Instead of waiting to detect loss and then responding to it, as Automatic Repeat Request (ARQ) approaches do, FEC attempts to prevent loss by sending extra copies of messages (Keshav 1997), trading bandwidth for latency.

The plane approximation also performs well under fairly high jitter: in running TCP it is not uncommon to see jitter (variance in latency) roughly equal to latency – see Table 6.2 for an example – without the users noticing. However, the warped plane approximation does not tolerate jitter well at all. UDP shows less jitter than does TCP, but not little enough to serve for the warped plane approximation. To run widely distributed experiments with the warped plane approximation requires a playout queue to smooth jitter. Unfortunately, a playout queue adds latency to the stream; using one requires striking a precarious balance between latency and jitter.

To determine the length of the queue, and thus the amount of latency it added to the stream, I use the Queue Monitoring algorithm described by Stone and Jeffay (1993). This is a control method with two parameters that, by varying the parameters used, can emulate standard single-parameter methods from the literature (Naylor 1992), as well as provide superior performance for regulating jitter on audio flows. The queue monitoring experiment is described in section 6.3.3.

[1]Logging, or recording, events received from the server allows us to “replay” the experiment at a later time, as discussed in section 2.2.

6.3 Experiments With Network Transport

A series of experiments (using the same apparatus described in section 5.4.1) proved the utility of the transport adaptations described above, and determined good parameter settings to use for each (Hudson et al. 2001).

6.3.1 Adapting Transport to Reduce Latency

The first set of experiments measured the effect of transport protocol on latency. TCP guarantees lossless, ordered delivery of messages. TCP's guaranteed, ordered delivery has the unwelcome side effect that a significant fraction of the lost message's delay is also suffered by every message sent between the transmission of the lost packet and its eventual correct receipt. The behavior of the TCP protocol that increases throughput in the face of congestion also increases latency. Thus, I altered the nanoManipulator to allow a choice between TCP and UDP as transport protocols for the haptic data stream, comparing loss, latency, and jitter over a 90% saturated bottleneck.[2]

Protocol	Mean Bandwidth (bps)	Application-level loss (packets per second)	Application-level loss (fractional)	Mean latency (ms)	Jitter (ms)
TCP	23,474	0	0	145	124
UDP	20,229	0.34	0.018	97	33

Table 6.2: nanoManipulator latency as a function of transport protocol with 90% bottleneck utilization.

Changing from TCP to UDP reduced latency by 33%, from 145 ms to 97 ms, and jitter by 75% (Table 6.2). Since 100 ms is a known breakpoint for stability in traditional teleoperation and an approximate breakpoint for usability in teleoperation with complex intermediate representations, this was an encouraging result. The price of this improvement, however, was loss, reordering, and potential duplication of messages.

Adding a sequence number to each message and dropping messages whose numbers are too low turns duplication and reordering into loss. The plane approximation was

[2]The experiments reported in this chapter vary conditions from 90% saturation to 98% saturation of the bottleneck. At 90% saturation, we saw enough latency to make TCP problematic. However, after switching to UDP, there was not enough jitter to need correction. Only at 98% saturation, with roughly 10% of packets lost, did UDP suffer enough jitter to be worth amelioration.

still usable at the ensuing 1.8% loss level, whereas it had been unusable under TCP with 145ms latency and no loss.

The nanoManipulator needs to record all data taken from the microscope for later replay or analysis. The original implementation did this recording at `nano`, but by instead logging at `topo` we avoid permanently losing any data to the unreliable network protocol. The scientist may not see some data while the experiment is in progress, but it will be guaranteed to be available for replay.

6.3.2 Adapting Transport for Loss

A lossy connection is adequate for the plane approximation, where the microscope sends data often and temporary lapses are permissable, but does not work well for local environment sampling.

FEC was added to the system to reduce loss without the latency penalty of reactive reliability protocols. Instead of assuming that errors are rare and invoking expensive recovery mechanisms when an error occurs, FEC assumes errors are common and provides a constant-overhead mechanism to statistically reduce the error rate. FEC sends multiple copies of every packet; if packet losses are independent[3] and one packet has a 10% chance of being lost, two packets will be lost only one time in a hundred.

Redundancy	Mean Bandwidth (bps)	Application-level loss (packets per second)	Application-level loss (fractional)	Mean latency (ms)	Jitter (ms)
0	20,229	0.34	0.018	97	33
1	39,509	0.04	0.0021	94	33
3	71,974	0.004	0.0002	95	33

Table 6.3: nanoManipulator performance as a function of error-correction protocol with 90% bottleneck utilization. Transport protocol is UDP; error-correction is Forward Error Correction (FEC) with the number of redundant copies of every packet specified in the table.

FEC produced an order-of-magnitude decrease in loss when sending two copies of every packet (doubling the bandwidth), and a two order-of-magnitude decrease when sending four copies (Table 6.3). Because this is considerably slower than an

[3]A common assumption of Internet protocols; not strictly true, but practical.

inverse-exponential decrease in loss rate, it would appear that packet losses on our test network were not actually independent.

This implementation sent duplicates of packet n when it sent packets n+1, n+2, ... Unfortunately, this means that to guarantee in-order playout of packets would require adding latency equal to the inter-packet time (typically 33 ms) times the level of redundancy. This is excessive for the purpose of lossless playout for force feedback, although it would allow a designer to combine a loss-tolerant haptic interface mode with logging at `nano`, which is perhaps more convenient for the scientists.

A more sophisticated implementation could send the duplicate packets at 1 ms intervals.[4]A triply redundant implementation, enough in this experiment to reduce loss one-hundredfold, would then only add 3 ms of latency to guarantee low-loss, in-order playout for the force feedback. This would, however, more likely run afoul of any temporal correlation of packet losses.

6.3.3 Adapting Transport for Jitter

The warped plane approximation performs well with 150 ms of response time. However, it breaks down when jitter exceeds 100 ms. When a network is designed to provide Quality of Service guarantees, one of the services commonly offered is bounded jitter (See section 3.2). Unfortunately, solutions to the problem of jitter add latency. Jitter is fundamental to best-effort networks; in order to mask the network's jitter it is necessary to add latency. I implemented Queue Monitoring, an adaptive jitter-control algorithm that can trade off between loss and jitter (Stone and Jeffay 1993).

Although jitter is a useful summary measure of delay variation, it does not capture the distribution of delays observed. I use two values that more completely describe the problems caused by delay-jitter in any process that tries to play back continuous media: gap rate and drop rate. When messages are buffered at a receiver for playout, two problems can occur. Either the queue can underflow, causing a gap in the playout, or the queue can overflow, requiring that the receiver throw out (drop) a sample.

For any application which is "playing out samples," the raw amount of jitter caused by the network is only a secondary measure which indicates the possible presence of gaps and drops. If one message is delayed, but is delayed less than the time it would have waited in buffers before playout, the delay is meaningless – no

[4]These packets are small enough – less than 100 bytes each, including all headers – that this is feasible over high-bandwidth networks.

gap occurs. Thus, even though the message may have been subject to network-level jitter, the application-level measurement of delay variation for that message is 0.

For this experiment, I loaded the bottleneck routers to 98% (4500 simulated web browsers, 5 servers). The nanoManipulator used UDP for transport, since we had already shown that it gave lower latency and jitter than TCP on congested networks (section 6.3.1).

Buffer Management Scheme	Application-level loss (per second)	Drop rate (per second)	Gap rate (per second)	Mean latency (ms)
none	2.8 (9.7%)	3.4 (11.7%)	6.2 (21.5%)	89
QM (30, 2)	2.8 (10%)	0.18 (0.6%)	3.0 (10.6%)	94
QM (150, 2)	2.8 (9.7%)	0.06 (0.02%)	2.8 (9.7%)	96
QM (3600, 2)	2.8 (9.5%)	0.003 (0.001%)	2.8 (9.5%)	91

Table 6.4: nanoManipulator performance as a function of buffer management, using UDP for transport, with 98% bottleneck utilization. QM(x, y) is Queue Monitoring with threshold x at queue length y. Drop rate is messages lost due to buffer overflow caused by jitter; gap rate is messages lost due to buffer underflow caused by jitter or loss.

At 98% utilization and 10% loss, jitter remained in the neighborhood of 30 ms. However, the gap rate was high; Queue Monitoring (at a variety of parameter settings) essentially eliminated drops with only a modest additional cost in latency (Table 6.4). As the experiment with Queue Monitoring shows, the distribution of delays caused bad performance (high drop and gap rates). In a followup test, the dummynet package (Rizzo 1997) was used to add additional fixed latency at the router to simulate larger networks. With 40 ms of added delay from dummynet, transmission using UDP and no buffer management saw 90 ms of jitter, enough to begin interfering with force feedback using the plane approximation. Adding Queue Monitoring reduced jitter to the level seen on the LAN (30 ms).

Queue monitoring helps the warped plane approximation, but it would contribute to marginally worse force feedback with the plane approximation. Queue monitoring smooths the playout of messages received from the microscope; the warped plane approximation explicitly corrects for (and relies on) this, but the plane approximation does not, and even this small increase in latency would weaken the spatial correspondence between hand position, sample position, and approximating plane position. For the simpler representation, it is better to always discard late force feedback data from the microscope: surface features may be missed, but those that are felt will not be so distorted.

6.4 Summary

The feedback control loop for the nanoManipulator's haptic interface requires less than 20 kbps of bandwidth and no more than 20 packets per second. This is easily supported by modern networks, but the control loop's latency constraints are not.

This chapter discussed experiments that verified that using UDP, FEC, and Queue Monitoring (QM) work for the nanoManipulator as well as for traditional audio and video streams, extending the useful range of haptic feedback. UDP reduced latency by 35% (50 ms) and jitter by 75% (90 ms) compared to TCP. FEC on top of UDP reduced loss tenfold (to 0.2%) by doubling bandwidth to 40 kbps, or one-hundred-fold (to 0.02%) by quadrupling bandwidth to 80 kbps. QM on top of UDP reduced the drop rate asymptotically close to zero with less than a 10% (8 ms) increase in mean latency. All three of these techniques should be considered by future implementors of networked force-feedback systems.

Chapter 7

Collaboration

The distributed nanoManipulator was conceived to enable scientific research across distances: to let scientists work with equipment they did not have nearby, whether microscopes or supercomputers. In practice, access to remote expertise (people) can be even more important than access to remote equipment. We extended the distributed nanoManipulator to support synchronous collaboration: sharing the application so that two scientists, separated by distance, can participate in the same experiment at the same time, consulting with one another and even trading control of the microscope. This set of extensions is referred to as the **Collaborative nanoManipulator**. We used a custom implementation of application sharing that was built to be flexible in its choice of algorithms, particularly algorithms for concurrency control. This chapter documents the design choices we made and the experiments I carried out to evaluate those design choices.

The Collaborative nanoManipulator focuses on sharing and extending the capabilities already present in the stand-alone system. To work together effectively, our users also want video and audio conferencing, as well as application sharing. For these secondary purposes, we used commercial off-the-shelf hardware and software running on an extra computer at each site. The entire system strives to be media-agnostic, rather than VE-centric, allowing the collaborators to use whatever combination of media is most appropriate for their current tasks (Robinson et al. 2001); this dissertation addresses only the VE component. Portions of this chapter were previously published as "Managing Collaboration in the nanoManipulator," by Thomas C. Hudson, Aron Helser, Diane H. Sonnenwald, and Mary C. Whitton, in *Presence: Teleoperators and Virtual Environments*, volume 13, number 2.

7.1 Motivation

Dewan (1997) lists three motivations for synchronous collaboration: distributed collaboration, sharing computer state, and going "beyond being there." The Collaborative nanoManipulator addresses each of these.

Distributed Collaboration: Synchronous collaboration enables people who are remote from one another to work together. This is often enabled by audio and video communication links, although for some tasks text "chat" is sufficient. The Collaborative nanoManipulator provides this capability through auxiliary programs; the issues that arise in enabling this communication are external to this dissertation.

Sharing Computer State: Synchronous collaboration allows people to share the state of one computer application while working from multiple computers. In the Collaborative nanoManipulator, the collaborators share access to the microscope. They can use a "shared workspace," so that they see identical views of the experiment.

Beyond Being There: At its best, synchronous collaboration gives people capabilities that they would not have if they were working side-by-side. Users of the Collaborative nanoManipulator each have their own force-feedback device and their own workspace, allowing them to simultaneously explore the same recorded dataset at different points in time or examine the same sample from different viewpoints or with different visualization parameters.

As discussed in chapter 1, the combination of these capabilities yields several desirable benefits. The system provides access to remote instrumentation and expertise, replacing travel. This, in turn, is expected to improve scientists' ability to successfully carry out lengthy, productive collaborations.

7.2 Goals and Requirements

The requirements for the Collaborative nanoManipulator grew out of an ethnographic study of scientists using the nanoManipulator (Sonnenwald et al. 2001). Based on the study, we decided to focus on the cognitive tasks – to support the work of understanding and experimentation that scientists normally carry out with the nanoManipulator. Support for scientists' social behaviors and communication was of secondary importance.

The primary goal of the Collaborative nanoManipulator is to allow scientists to share control of an experiment in progress. To do this, they must each have full command of all the features of the nanoManipulator, including control of the microscope. They must be able to work on the experiment together, with a common frame of reference, or work independently. They must be able to easily shift between these **coupled** and **uncoupled** modes of work. They must be able to take turns controlling the microscope, and must not be able to interfere with one another's work, but must not have their work hindered by the user interface that manages all of these features.

Scientists also review the results of past experiments as part of their collaborative work. The nanoManipulator has long been used for asynchronous collaboration, where one scientist carries out an experiment and another reviews it hours or days later. Scientists also wanted to be able to review previously captured data synchronously, so that two widely-separated people could look at the same recorded datasets at the same time and consult on analysis techniques or follow-up experiments.

These goals resulted in three requirements that drove the architecture of the shared nanoManipulator application. First, the application had to be highly interactive, with both response time and user-to-user time under 300 ms. Second, it had to support the ability for a collaborator to interleave periods of coupled and uncoupled – shared and private – work, with the corollary that there be an easy mechanism for copying application state data back and forth between the independent (private) and collaborative (shared) work modes. Three, it had to have all the features and functionality of the single-user system.

7.3 Human Factors

Latency is a significant determinant of the utility of a user interface; collaborative applications have not one but two significant latencies that must be controlled: response time and user-to-user time (Bhola et al. 1998).

Response Time is the time between a user's input and that user seeing the results of her input.

User-to-User Time is the time between a user's input and some other collaborating user seeing the results of that input.

Bryson and Johan (1996) reports 100 ms as an approximate upper bound for direct manipulation tasks, and Macedonia et al. (1994) provided the same upper limit for wargames. Bhola et al. (1998), in the context of a collaborative application, state the limit varies between 50 and 100 ms; for their user tests of mouse-controlled drawing packages, 50 ms of response time noticeably interfered with usability and 80 ms made the application unusable. When force feedback is being used, or the input is driving a control loop, response time becomes even more critical; 50 ms of latency in flight simulators reduces performance, and only a little more leads to user-induced oscillations and instability (Wickens and Baker 1995).

When users are attempting to coordinate their actions, or understand how their actions affect one another, user-to-user time is important. Leigh et al. (1998) and Vaghi et al. (1999) both report 200 ms of delay interfering with the completion of closely-coordinated tasks in collaborative virtual environments. (Johnson and Leigh (2001) report that collaborators are better able to adapt to latency than they are to jitter, and that collaboration can remain successful with 150 to 200 ms of latency.)

7.4 Design

There are a large number of design decisions to be made in creating a multi-user collaborative application. Drawing from Dewan's (1997) list, the following issues must be considered in the design:

Coupling: Coupling is the relationship between different users' views of the application state. To allow scientists to use the same working patterns we observed in our ethnographic study (Sonnenwald et al. 2001), we support two modes of coupling. Each scientist has a private space, where they can work independently. There is also a shared space, where the two user's views are fully synchronized: "What You See Is What I See." The application state can be copied from private to shared space, to allow both scientists to look at work done independently, or from shared to private space, to allow one scientist to take joint work and further develop it on her own.

Collaboration focuses on sharing the scientists' views of the data. In the private state, each user's view is not influenced by the collaborator. In the shared state, all system parameters are controllable by either user at any time. Collaboration requires mutual intent – both scientists must explicitly choose to collaborate.

Access Control: Control of the physical devices in the nanoManipulator system – the microscope and any subsidiary devices like an ohmeter – cannot be safely shared if we are to preserve user intent. A single mutex guards access to these devices; the user who is currently controlling the device must explicitly yield before her collaborator can take over.

Awareness: Awareness is the extent to which each scientist is given data by the program about their collaborator's actions. During a collaborative experiment, both scientists are connected to the same microscope. They see each other's changes to the state of the device, whether they are in private or shared space. When working together in the shared space, they also see one another's changes to the visualization parameters and one another's pointers (annotated to show the partner's current mode of interaction with the application).

Concurrency Control: Concurrency control attempts to guarantee that all users see a consistent and expected application state. The most critical decision for the performance and usability of our Collaborative nanoManipulator was the choice of concurrency control methods; they are discussed in sections 7.7 and 7.8.

The Collaborative nanoManipulator uses Dourish's (1995) notion of **divergence** in its architecture, but not in its implementation. At a high level, supporting both shared and private spaces allows for a **multi-synchronous** approach to collaboration. In a document-based system, synchronous collaborators look at the same document at the same time, while asynchronous collaborators look at the same document at different times. Multi-synchronous collaborators work independently in parallel – simultaneously creating different versions of the same document, then integrating them. The several spaces of the Collaborative nanoManipulator give us the same advantages as a more generalized support for divergence: they allow smooth transitions between interactive collaboration and asynchronous or independent work, directly supporting patterns of activity we have observed in collaborating scientists (Sonnenwald et al. 2001). We do not support reintegration, bringing multi-synchronous versions together, but only the selection of one or the other version as a starting point for further synchronous work. [1] As an implementation technique, divergence

[1]Our collaborative framework was designed to enable mixing of the shared spaces for reintegration, but users have been satisfied without it. It also appears that our users can much more easily

provides scalable synchronization: it allows variation from tight coupling (frequent synchronization) to loose coupling (occasional synchronization). The nanoManipulator's shared space requires tight coupling, without any variation in synchronization, so we used implementation techniques that maintain close coupling without paying the overhead of divergence-based approaches.

Network Layer	Technique
Presentation	**Optimistic concurrency control** (7.6.2)
	Asynchronous remote procedure call (7.7.2)

Table 7.1: Proposed adaptations for collaborative systems. Those in boldface were used in the experiments discussed in this chapter.

7.5 Concurrency Control

A concurrency-control protocol ensures that incorrect behavior cannot occur as a result of concurrent access [to one object] by multiple clients (Herlihy 1987).

Concurrency control attempts to guarantee that all users see a consistent application state: that no errors occur in synchronizing replicas. For example, collaborators editing a document together should see the same text. The details of a consistency guarantee are highly application-dependent (Munson and Dewan 1996). Concurrency control also attempts to guarantee that users see an expected application state, sometimes expressed as the "Principle of No Surprises" or the "Isolation Property." Users should never be surprised by the application's response to their input or have their input cause unintended effects due to another user's simultaneous actions (Weihl 1993b).

The literature on collaborative software has a large set of specialized definitions. When two scientists are collaborating, the software running on each of their computers is referred to as a **peer** or **client** in the collaboration. The collaborative application lets the collaborators share a number of **objects**; each peer has a **replica** of every object. **Synchronization** is the process of ensuring all peers agree on the state of the objects: keeping replicas **consistent**. Any occurrence at either peer is an **event**; the

deal with the work model of distinct spaces – private and shared – than a model where they may choose to have some attributes of their view defined by one "space" and some by the other.

most significant events are an **operation** (a change to the value of a replica), a **send** (the transmission of a message from a peer), and a **receive** (the receipt of a message by a peer). Operations conducted by two peers that would lead to inconsistent state **conflict**. **Serialization** is the process of ensuring that all peers agree on the order in which events occur. While users are executing operations or replica-update messages are in transit, the collaboration is **active**; when no state changes are pending, the collaboration is **quiescent**.

Messages must be exchanged between peers to keep replicas consistent. This sets up a critical problem that limits the performance of distributed systems, stated by Singhal's Consistency-Throughput Tradeoff: "It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state." (Singhal and Zyda 1999)

7.6 Concurrency Control Strategies

The Collaborative nanoManipulator was written within a framework that enabled concurrency control methods to easily be added and switched among. This permitted experiments that compared the performance of different algorithms – measuring their effect on response time and user-user time – in the same setting. Many algorithms have been devised for concurrency control; the wide range of possibilities is surveyed below. These range from pessimistic to optimistic concurrency control, serialization techniques needed to support optimism, and even those applications that operate without concurrency control.

These algorithms all exhibit a tension between several goals of concurrency control, among them:

- minimizing response time
- maximizing flexibility and the ability for multiple users to work in parallel
- minimizing the overhead collaboration adds to a user interface
- minimizing the amount of work lost when two collaborators attempt inconsistent operations.

For this dissertation, I evaluated concurrency control strategies until I found one that adequately supported the Collaborative nanoManipulator. After surveying the possibilities, I discuss the architecture that gave the Collaborative nanoManipulator the ability to use multiple concurrency control methods, and the experiments that revealed the methods' suitability for the Collaborative nanoManipulator.

7.6.1 Pessimistic Concurrency Control

Pessimistic approaches to concurrency control try to prevent users from performing any operations that could lead to inconsistent state. The simplest way to do this is with a lock.

Locking in Databases: The first widespread applications that needed to worry about concurrency control were distributed databases. The typical approach taken by databases is to lock records being modified so that only one user can operate on a record at a time. A lock is secured on the records needed by the operation, the operation is performed, and then the lock is released. This sequence of steps introduces latency due to the overhead of acquiring and releasing the lock; each of these operations generally requires at least one network round-trip (Weihl 1993a). Database locks are typically procured automatically, without user intervention. Locking in databases yields risk of deadlock on operations that span multiple records or tables (Weihl 1993a). Weihl (1989) has a taxonomy of other approaches used in databases. Databases are straightforward environments for concurrency control because the semantics are regular and uniform; the variety found in other applications is described below.

Floor Control: The simplest form of concurrency control seen in non-database applications is "floor control" (Lantz 1986; Malpani and Rowe 1997). Floor control uses a single lock to guard access to the entire application or document being shared. Only one user at a time can actively manipulate the application. Inactive users press a button to request control of the application. As in a well-run meeting, only one speaker at a time "has the floor" and can address others or take action, while others watch and listen. A widely disseminated system that uses floor control is Microsoft(TM)'s NetMeeting(TM). NetMeeting allows users of Windows workstations to share standard Windows applications. Only one user has control of the on-screen mouse pointer at any time; others watch. If an inactive user clicks her mouse, she takes control – after waiting for network traffic to deactivate the previously active user's mouse. In our studies of the Collaborative nanoManipulator deployed with

NetMeeting, users have found this explicit floor control both confusing and slow, interfering with their collaboration (Sonnenwald 2002).

Many collaborative systems require the current active user to explicitly relinquish control before another user can take control, which reduces surprises but adds to the overhead that managing collaboration imposes on task performance. Some systems also support queueing requests for control by inactive users, while others go even further by designating one collaborator as a moderator who can reorder the queue or revoke the control of the currently active user, much like a moderator under Robert's Rules of Order.

Floor control is often a reasonable method of concurrency control for formal meetings held via teleconferencing, just as it is in real life. However, it is not appropriate for many other modes of work or applications. If used to control access to a large document, floor control allows only one user to work at a time, removing any possibility of parallel activity. Some collaborations proceed well in this mode, while others proceed poorly.

Fine-Grained Locks: To allow multiple users to be active at once, locks can be made fine-grained. In a text document, each paragraph could have its own lock. In a virtual world, each object could have a lock. Managing fine-grained locks can be difficult for users, since they may become as complex as the document itself.

This discussion assumes that for each lock one client is the designated server, arbitrating which client receives the lock when several clients concurrently request it. A machine other than the clients can also arbitrate lock ownership, but this does not improve performance. There also exist distributed approaches to arbitrating lock requests, where no one machine has responsibility; these pay overhead for serialization similar to that discussed in section 7.6.3.

Implicit Locks: One common strategy that attempts to preserve the flexibility of fine-grained locks without burdening users with their management is to make them implicit: the user takes no action outside the normal flow of work to request the lock. As soon as the user undertakes some manipulation that would require the lock, her local interface blocks her progress until the lock is obtained. Once she has the lock, she is allowed to continue with her operation. If she can not obtain the lock because another user is holding it, her local application may unblock and display a failure indicator, or continue to block until the other user is finished. When the operation is complete, the lock is released.

Implicit lock management algorithms do not know when to release a lock. Consider a user in a 2D shape editor who moves an object, pauses to consider the location briefly, and then decides to move it further. If she releases the mouse between the two movements, the concurrency control system may release her lock. She must then wait for the network round-trip needed to regain the lock, and must also contend with other users who may be trying to carry out different operations on the same object. Correct behavior by the application relies on understanding the user's intent; thus, the decision between implicit and explicit lock management can be reduced to the decision between greater overhead on every operation and occasional unexpected overhead on some operations. In some sense this is equivalent to the question of whether users prefer high mean latency with low jitter or low mean latency with high jitter.

Migrating Locks: This time penalty can be partially ameliorated by "migrating" a lock to the workstation of the last user who requested it. That is, when a client obtains a lock it is also assigned responsibility for determining who next receives the lock. If user A uses an object, releases it, then uses it again, no network traffic is needed to enable the second use, since the responsibility of managing the lock was moved to her workstation during her first use of the object. If user B wants to use the object, she must send a message to A's workstation rather than to a central lock server. The lock migration protocol needs to be careful to avoid thrashing – if A and B both try to use the object frequently, and the lock migrates back and forth between them, migration adds additional overhead.

Speculative Execution: Whether or not they use migration, even fine-grained, implicit locks add latency to the user interface. The interface is blocked – unresponsive – from the time the user begins an operation until the time that the concurrency control system obtains the required lock(s) and allows the operation to continue.

To avoid this latency, some collaborative systems carry out speculative execution: the user interface is never blocked, and the user is allowed to begin the operation without obtaining a lock. However, the system still tries to lock the objects being manipulated; if a lock can not be obtained, the system has to abort or undo the operation. Although this avoids any latency overhead, in the case of a conflict this causes surprise and lost work (Conner and Holden 1997).

Optimistic approaches carry speculative locking one step farther: they allow the user to begin working immediately, and take some step less extreme than aborting if they detect that the user's input conflicts with another user's action.

7.6.2 Optimistic Concurrency Control

Optimistic concurrency control algorithms guarantee consistency with low latency. Following Amdahl's Law, informally *"Make the Common Case Fast"*: systems using optimistic concurrency control perform an operation, then check to see whether any other peer executed a conflicting operation simultaneously. If conflicts occur, an application-dependent procedure resolves them (Herlihy 1990). In most applications, most operations do not conflict, and so can be executed rapidly, without the overhead of locking. Optimistic concurrency control algorithms do not guarantee that all replicas at all clients are always consistent, but that they will be consistent when quiescent,[2] after any necessary conflict resolution has occurred.

Typically, asynchronous collaboration applications can use optimistic concurrency control with manual human intervention for conflict resolution. This is because operations are carried out on local replicas of the application state, and local replicas are only occasionally synchronized with peers; when synchronization occurs with low frequency on large amounts of work, manual merges are acceptable. Synchronous collaboration requires awareness, which leads to frequent synchronization of small changes. In this case, users generally require automatic conflict resolution.

Optimistic concurrency control avoids the time overhead of locking. Each operation is applied at a user's workstation as soon as she requests it. Her operation is then sent to her collaborators' workstations, which are responsible for executing it in such a way as to guarantee consistency. Most optimistic algorithms require that an ordering be established over all operations – a **global order** – but do not require that operations be delivered in order.

The simplest case of automatic conflict resolution for optimism occurs when all operations are commutative or idempotent and need not be serialized. Operations on an object arriving that predate (in the global order) the most-recently-executed

[2]Like silence in speech, quiescence is common in streams of user actions. However, the frequency of both of these "stillnesses" varies widely between applications and circumstances. Most optimistic concurrency control algorithms reach global agreement on the result of any one operation relatively quickly, but in the worst case the effects of some operations may not be everywhere consistently reflected until quiescence.

operation on that object can be discarded. For more complex operation semantics, optimistic systems either undo the later operation, execute the early operation, and then redo the later operation (**undo-redo** schemes), or transform the earlier operation into the operation it "would have been" if it had been executed later in the global order (**transformational** schemes).

7.6.3 Serialization

If operations are broadcast from the replica at which they occur to every other replica, the order in which they occur is unknown. Some additional information is necessary to put them in a consistent order, or **serialize** them. For most optimistic concurrency control methods, peers in a collaborative application must come to a global agreement on the order in which events occurred. The goal of optimistic concurrency control is low-latency performance; latency depends on the serialization technique used. Creating a global ordering of distributed events has historically been an expensive operation (Birman and Joseph 1987).

Centralized Serialization

The simplest way to serialize is to centralize: all message traffic passes through a single server, which is responsible for giving it a consistent ordering. Every replica sends operations requested by its user to the serialization server, which then broadcasts them to all replicas, including the originator, in the order in which it received them; the replicas execute the operation when it is broadcast to them. If operation A arrives at the serialization server before operation B, then every replica in the system executes A before executing B, regardless of the time at which they were requested by users. Centralized serialization defines a total order over all operations in the system, and makes sure that they are delivered to all replicas in the same order.

The drawback to centralized serialization is that every operation coming from a site other than the server experiences one full network round-trip of latency. Instant response is impossible; a user cannot see the results of her action until it has been to the serialization server and back. Over a wide area network, this delay can be hundreds of miliseconds, enough to interfere with the application's usability (Prakash 1999). The performance of centralized serialization is similar to that of centralized locking.

The latency penalty can be statistically reduced by *distributing* the responsibility for serialization. Operations on a single object may only need to have a consistent ordering with respect to other operations on that object. In this case, serialization for operations on that object may be migrated to the system that most uses the object. By moving serialization out of a central server and into the peers, each object can have one user who sees no latency to operate on it; as the use pattern of an object changes, serialization for that object can move to whichever peer uses it most. Migrating, fine-grained serialization has a performance similar to migrating, fine-grained locking schemes, which have been widely proposed as a mechanism for concurrency control in shared virtual environments. However, optimistic concurrency control protocols should be able to do better.

Distributed Serialization

Centralized approaches to serialization suffer because all clients (except, perhaps, one client which also serves as a serialization server) pay a latency penalty. When a user's input commands an operation, that operation does not execute until a message has been sent to the serialization server and a response received – one full network round-trip. This means that the performance of the local user interface becomes dependent on the network latency.

Serialization attempts to order all events that occur in a system. We already have support for ordering of events built into every computer: the on-board clock. However, clocks are not well synchronized; out-of-sync clocks will yield incorrectly-serialized messages. Traditional algorithms for distributed serialization attempt to provide a global ordering without relying on a "real-time" or "wall" clock. The "global real-time clock" is regarded as a practical impossibility but a necessary starting point for theorizing. [3]

The **logical clock** is a distributed algorithm that provides a consistent ordering of events in a real-time system (Lamport 1978; Babaoglu and Marzullo 1993). Using the logical clock, we can get rid of the need to have any serializing server. Although a logical clock does not require a network round-trip to a server in order to serialize an operation, it is still heavily dependent on network performance. Once a user requests an operation, the logical clock cannot place it into a total order and allow it to be executed until a network message has been received from every other peer

[3]The theory of distributed clocks makes a number of careful distinctions, such as that between causal and total ordering, which are not necessary in this dissertation.

in the system (Babaoglu and Marzullo 1993). Thus, delay from the logical clock is controlled by how much bandwidth we are willing to consume with otherwise-meaningless “clocking” messages, and the bandwidth grows with the square of the number of peers. See appendix D for the details of the logical clock algorithm; other uses of the logical clock are discussed in Raynal (1992).

The **vector clock** is a construct related to the logical clock that can guarantee consistent, causal delivery of messages without the added delay of waiting for messages from every other process in the system (Babaoglu and Marzullo 1993). I give the vector clock algorithm in Appendix E. Adding a few constraints to a vector clock can turn its causal ordering into a global ordering.

Both the logical clock and the vector clock add significant implementation complexity to a system, and increase the network bandwidth consumed (in proportion to the number of peers in a collaboration). Although latency is bounded, within that bound latency also increases as the number of peers rises. Today, freely-available software uses the Network Time Protocol (NTP) (Mills 1996) to attempt to synchronize one computer’s clock with another’s, taking into account network latency and all the other details that have traditionally interfered with accurate synchronization of clocks. Computers running NTP have a very good approximation of the mythical global real-time clock. Thus, by time-stamping messages with the clock of the sending peer we can get approximate serialization with a small constant network overhead and negligible added latency.

I call this approach **wall clock serialization**: using the peers’ on-board clocks, synchronized with NTP, to assign timestamps to every operation, guaranteeing that a consistent, total global order is agreed upon by all peers. Errors in synchronization – **clock skew** – lead not to inconsistency or a breakdown of serialization but to a mismatch between the global order agreed upon by the peers and the order observed in reality (a *noncausal* ordering). NTP makes the synchronization error small enough to be both negligible and imperceptible when the events serialized are part of the control of the user interface.

Schneider (1993) used wall-clock serialization plus an extra delay (based on an upper bound of possible clock skew) to provide pessimistic concurrency control. Wall-clock synchronization does not guarantee in-order delivery of messages without this extra delay, but the optimistic concurrency control methods discussed above do not require in-order message delivery. However, using NTP the clock skew bound is so low that Schneider’s method may be competitive with optimistic approaches.

7.6.4 Social Protocol

Students of the human side of collaboration use the term "social protocol" to refer to the communication patterns people use while collaborating. Some collaboration systems, rather than implementing any of the above concurrency control protocols to preserve intent, rely on the social protocols performed by their users. People who are collaborating closely and have audio or text chat channels with which to communicate tend to stay aware of one another's work and talk about what they are doing well enough to avoid the collisions and inconsistencies that the above methods attempt to mechanically prevent. Leigh et al. (1999) advocate this approach, with optimistic methods coming in second-best in the applications they survey and pessimistic methods worst.

7.7 Implementing the Collaborative nanoManipulator

There are several frameworks and toolkits for building multi-user distributed virtual environments; many are surveyed in Meehan (1999). Were we building the Collaborative nanoManipulator from scratch, we would have liked to have adopted a framework like Suite (Dewan and Shen 1998). Working with a large existing application, we found it easier to write our own support software for concurrency control (10,000 lines of source) than to rewrite the entire application (125,000 lines of source) to fit within an external framework. This gave us considerable freedom to experiment with various concurrency control methods and to tune them to suit the application semantics. However, we also agree with Leigh et al. (1999) that it is much better to build an application for collaboration from the ground up than to attempt to retrofit it.

7.7.1 A Model-View-Controller Decomposition

The normal use of the Model-View-Controller paradigm (see Appendix F) is to factor an entire application into one Model, one View, and one Controller. However, the Collaborative nanoManipulator handles two types of commands with very different semantics: commands that modify the state of the microscope and commands that modify the the user's view of the data returned by the microscope. We found it useful to follow this division, designing the application around two Models, each with its

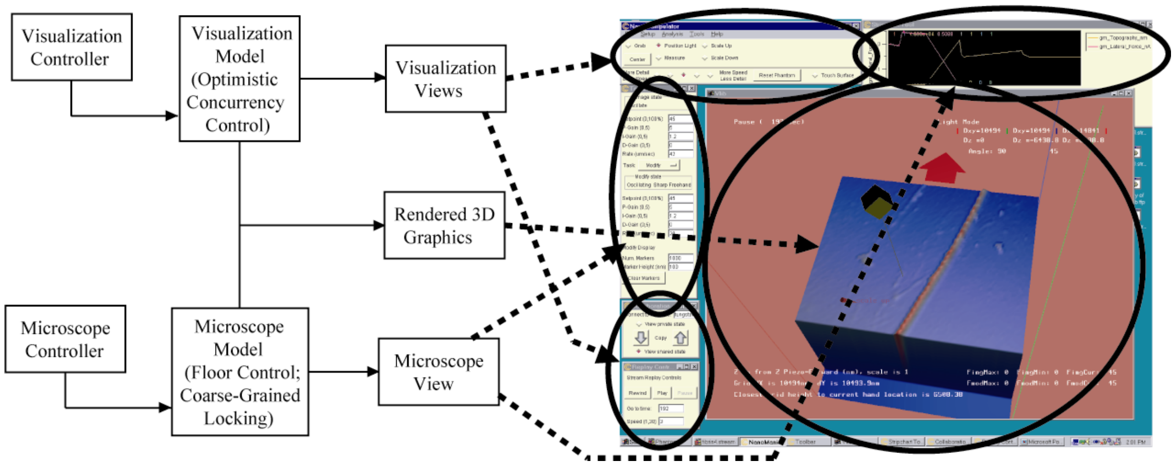
Visualization Controller
Visualization Model (Optimistic Concurrency Control)
Visualization Views
Rendered 3D Graphics
Microscope Controller
Microscope Model (Floor Control; Coarse-Grained Locking)
Microscope View

Figure 7.1: The nanoManipulator as two models, two controllers, and three views. Separating the two models gives us a clean separation of concurrency control methods.

own View and Controller. This gave us a clear separation of concerns in the code and made it easy to use a different concurrency control method for each Model.

Figure 7.1 shows our decomposition of the Collaborative nanoManipulator into two controllers, two models, and three views. The two Controllers are each integrated with one View as a traditional GUI (implemented using Tcl/Tk) with buttons, sliders, drop-down menus, and other standard widgets to allow users to view and request changes to the current state of the microscope and the visualization methods applied to that state. The third View renders data from the microscope Model using the methods and parameters specified by the visualization Model.

For example, in the Collaborative nanoManipulator system, there is a replicated Model of the microscope. This Model contains such information as, “What is the current location of the microscope tip? What is the current force exerted by the microscope on the sample? What is the topography of the surface recently scanned?” This is the current position and geometry of objects in the world data sensed by a device external to the virtual world, rather than directly controlled by the user and the parameters that control that device. The mechanics of Atomic Force Microscopy are discussed in Section 3. A locking mechanism is required for this model because two users trying to direct the tip of the microscope at the same time while it interacts with a sample could cause damage to both microscope and sample and would guarantee that neither users intention would be satisfied.

The way the data in the Model of the microscope is visualized (View) is controlled by a complex set of parameters, which form a second, subsidiary Model that we call the visualization Model. The data in the visualization Model answer such questions as, “From what position (in a virtual space) is the user viewing the sample? What color map is applied to the sample? How is it illuminated? Are isocontour lines displayed?” This is information about the rendering modes used by the program and about the position and geometry of purely virtual objects.

Because of the continuous nature of its inputs, control of the visualization Model is very sensitive to latency. The operations in the visualization Control are easily commutable and non-critical. The latency requirement and low probability of causing an inappropriate application response suggest that the visualization Model can be treated with optimistic concurrency control.

The data in the visualization-Model defines the difference between shared and private work: if two collaborators have consistent visualization-Models, they see the microscope data in the same way; if not, they work independently. Separating a set of requirements into distinct Models this way, along the borders between feasible concurrency control schemes, helps us determine the architecture of the overall program and helps us build in flexibility.

The “Microscope-Model” had a single consistent master state that was maintained on a computer physically collocated with the microscope; state changes were sent from the microscope to local replicas at each copy of the nanoManipulator. The Microscope-Model required coarse-grained, pessimistic concurrency control to guarantee that user intent was preserved across a series of operations; failure to do so could damage the microscope or the sample and ruin the experiment.

We chose to use a single lock for all of the microscope’s controls to guarantee that the intention of the user is preserved. If we had used fine-grained locking for the microscope, we could guarantee that individual operations were atomic – that changes to microscope variables were not corrupted – but could not preserve intent. If one user engages the microscope tip with the surface and begins to feel along a path, while the other user unknowingly increases the force exerted by the microscope tenfold, the sample or the tip can be damaged, something neither user intended. This is similar to the concept of transactions in a database, which recognize that fine-grained locking is not enough when one operation in a database depends on another.

State for the visualization-Model, or “View-Model”, was fully replicated between the peers. I implemented two forms of concurrency control for this replicated state.

Both were based on a simple optimistic approach, but varied in the form of serialization they used. At one extreme was server-based serialization, which was expected to perform equivalently to server-based locking. At the opposite extreme was wall-clock serialization. In addition to NTP, additional clock synchronization code running in a user-level library helped bound clock skew and ensure that the ordering of events was causally consistent. As did NPSNET and AVIARY, we found that wall-clock serialization works extremely well for shared VE.

For server-based serialization, one of the collaborating peers was chosen as the server. This caused the application's frame time to be added to network round-trip time in determining the system's response time and user-user time. Because of the single-process architecture of the nanoManipulator, frame time appears multiple times in many control loops; all of my measurements of collaborative performance are strongly dependent on frame time (as well as network delay). Repeating these experiments with faster graphics hardware reduces the performance advantage of optimistic, wall-clock serialized approaches, but they should always provide a faster response time than the other algorithms.

7.7.2 Synchronizing Replicated Models

We used the Virtual Reality Peripheral Network protocol (Taylor et al. 2001) as the Session layer for the Collaborative nanoManipulator, giving us the ability to make asynchronous remote procedure calls. With asynchronous Remote Procedure Call (RPC), when a process requests an operation from another process, the requesting process does not wait for values to be sent back by the remote server, but instead resumes execution immediately. Receipt of the response from the remote server will trigger a callback in the requesting process. This non-blocking communication helps to decouple the application from network latency, allowing an interactive application to reduce response time for operations that are being carried out across the network. In this way, asynchronous RPC is more useful for interactive applications than is traditional synchronous RPC. Asynchronous RPC typically leads to a complex programming style centered on callbacks; however, this same style is also required by modern graphical user interfaces, and was adopted for most of the Collaborative nanoManipulator.

7.8 Experiments with Collaboration

Our infrastructure enabled several different implementations of concurrency control for the View-Model. I conducted a series of experiments to measure the relative utility of two.

I placed two dual-processor 500 MHz Windows NT workstations running the Collaborative nanoManipulator on the department network, a switched 10/100 Mb Ethernet LAN. For the following experiments, instead of connecting to a live microscope, I replayed a streamfile – the recorded data that came from a live microscope during a past experiment. This increases repeatability while maintaining the timing of data arriving from an actual microscope. During these experiments, the computers were equipped with Intense3D$^{(TM)}$ Wildcat$^{(TM)}$ 4000 graphics hardware.

7.8.1 Timing Measurements

My baseline measurements determined that with a single scientist using the nanoManipulator (non-collaborative), average response time was 214 ms. Thus, even though the frame rate was 15 fps or better, which in an ideal implementation would lead to a response time under 66 ms, 214 ms lapsed between user input to the application (changes to the View-Model) and the corresponding update of the screen.

I measured the performance of the concurrency control implementations by approximately repeating the same series of operations with the Collaborative nanoManipulator in different concurrency control modes. The operations used were a series of movements of the surface, variations of surface scale, and movements of the measure lines. See table 7.2 for a summary of results.

A client hosting the serialization server sees no perceptible degradation in performance. However, any other collaborating client experiences a dramatic difference, with response time increasing in my experiments to 533 ms. The average cost of network communication and serialization was 319 ms, more than doubling the response time. Remote collaborators strongly objected to the system's high latency, refusing to use the collaborative system. An improved implementation, or a dedicated serialization server, might reduce this by the frame time of the collaborating peer, but that would still leave more than 105 ms of overhead for centralized serialization due to network latency, O/S latency, and the cost of concurrency control algorithms.

Another option is to use implicit fine-grained migrating serialization, so that the latency penalty only needs to be paid once for each extended series of manipulations.

When a user tries to manipulate one particular object or variable, responsibility for serializing operations on that users object migrates to the users host. The initial operation incurs network latency while waiting for serialization to migrate, but once the responsibility has been assigned to the users host computer further operations do not see any network latency (until another user attempts to manipulate that object and the responsibility is migrated to the other users computer). However, this is a complex approach that does not completely eliminate serialization latency and does not degrade gracefully. Two users attempting to manipulate the same facet of the application state can experience both surprise (as they repeatedly undo one anothers work) and significantly heightened latency (if the responsibility for serialization ping-pongs back and forth between their clients, they pay the serialization latency penalty many times over; less pathological failure modes still leave one user experiencing network latency in their attempts to carry out an operation.)

Exploring alternatives, we used the "wall clock" (the workstations' built-in clocks) to serialize messages for our concurrency control protocol. Dimension4 software synchronized the clocks. Our peak measured error between a clock and the reference source while running Dimension4 was less than 10 ms; a conservative bound on error between workstation clocks would then be 40 ms, below users' threshold of perception. Even clock mis-synchronization on the order of 100ms does not lead to a visibly inconsistent ordering of events.

Mode	Mean Response Time (ms)	Mean Response Time due to Concurrency Control (ms)
Single-user (baseline)	214	–
Server-based serialization	533	319
Wall-Clock serialization	262	48

Table 7.2: Latency cost of concurrency control methods. Response time is time between receipt of user input and display of corresponding output.

Tests on a private network (also a switched 10/100 Mb Ethernet LAN, but without the background traffic of our departmental network) showed similar results.

In practice, the only weakness we have found in the wall-clock approach is that our users do not realize the necessity of Dimension4's NTP client and shut it down; PC clocks are sufficiently inaccurate that the clocks on two workstations can become out of synch by an hour or more within a few days, leading to obvious errors when users try to collaborate.

7.8.2 User Perception

Members of the nanoManipulator team conducted a controlled experiment to evaluate users perceptions of the system and their success using it. In the experiment, 20 pairs of upper-level undergraduate science majors participated in two lab sessions during which they used the replay feature of the Collaborative nanoManipulator to explore and analyze AFM data collected earlier. During one lab session the study participants worked face-to-face using the nanoManipulator in single-user mode and during the other lab session they worked remotely (in two different locations) using the Collaborative nanoManipulator. Order of the two conditions, face-to-face and remote, was counterbalanced, i.e., ten pairs worked remotely first and ten pairs worked face-to-face first. A detailed description and discussion of the experiment can be found in Sonnenwald et al. (2002). In this section I focus only on interview data related to concurrency control and, specifically, user comments comparing the explicit, coarse-grained floor control of the analysis and productivity applications and the implicit, optimistic concurrency technique employed for shared visualization-view parameters.

After each lab session, the experimenters interviewed participants asking for their perceptions of the system. In particular, they were asked what they found most satisfying and dissatisfying about using the technology. The questions were open-ended and participants were free to discuss any aspect of the technology, including visualization functionality, scientific data analysis tools, haptic feedback, audio and video conference capabilities as well as concurrency control. Eight participants (out of 40) specifically discussed the relationship of concurrency control and usability of the system in their responses. The responses indicate that participants preferred the ability to work simultaneously in the shared nanoManipulator application, which uses optimistic concurrency control, over the explicit floor control required in the off-the-shelf shared application software, Microsoft NetMeeting.

Some participants also preferred the Collaborative nanoManipulator system over working face-to-face because they each had full access to a system, they could go into the private work mode and make progress independent of their lab partner, they could cooperate to accomplish a task, and/or they could work in parallel to accomplish the entire task more quickly.

In summary, collaborating study participants reported that the optimistic concurrency control as implemented in the shared nanoManipulator application provided advantages over the explicit floor control found in both the shared applications and in face-to-face cooperation. Verbal communication was used to help resolve conflicts that arose when control had to be explicitly passed. In comparison, participants did not report extra verbal communication was needed to mediate sharing in the shared nanoManipulator application.

7.9 Summary

Building the Collaborative nanoManipulator, I found that the Model-View-Controller paradigm was a useful way to analyze Collaborative Virtual Environment (CVE)s. Extending Model-View-Controller to include multiple parallel or hierarchical Models highlights the differing concurrency control requirements of different subsets of an applications state data and helps us design the applications architecture. To support the interactivity requirements of our CVE, we used asynchronous RPC and optimistic concurrency control. Optimism is a valuable technique particularly well suited for minimizing latency of continuous, easily reversed inputs to a VE, such as a users viewpoint position and orientation. It is less amenable to supporting transactions and complex assemblies of primitive operations, but can be used to do so with a moderate software engineering effort.

Chapter 8

Remote Rendering

When we began this research, rendering the nanoManipulator's three-dimensional view of the sample at a sufficiently high frame rate was too demanding a task for personal computers. High-end Silicon Graphics, Incorporated (SGI) workstations were capable, but were not found in the labs and offices of our collaborators. Thus, the third interface I investigated for this dissertation is remote rendering: graphics being drawn by a workstation or supercomputer and transmitted over the network to be displayed at the user's PC.

In 2004, PCs are fast enough to provide all the graphics power that the nanoManipulator currently requires. However, there are still calls for remote rendering. Models or simulations that drive graphics applications may be too big to run on desktop machines (Aliaga et al. 1999; Stone et al. 2001). Volume visualization and advanced shading techniques require hardware support to run in real-time (Bokinsky 2003). All of these new applications can be rendered on capable machines and transmitted across the network for display.

8.1 Human Factors

When describing the human factors of an interactive graphics system, we should recall chapter 2's distinction between continuous and discrete modes of input.

Input with a tracked device – a PHANTOM, or a mouse – is *continuous* input: the user needs to see the rapidly updated graphics that show the intermediate positions through which they move the device. He is controlling his movements with a reflexive feedback loop; if the update is delayed, he will overshoot his target.

Input via the keyboard or "buttons" on two-dimensional graphical user interfaces is *discrete* input: the user is executing an action for which they do not need immediate feedback.[1]

Users can tolerate latency of as high as one second for discrete input. However, continuous input has a bound below 100 ms, for some tasks below 50 ms (chapter 2). Image-based remote rendering takes advantage of this distinction. Most continuous inputs change only the position or orientation of the user's viewpoint, and Image-based rendering (IBR) can re-render these changes at the client without waiting for an update from the server. Most inputs that significantly change the image to be displayed, invalidating the current image-based representation, are discrete, and so users more tolerant of the wait for the server to recompute and retransmit a representation.

8.2 Approaches to Remote Rendering

8.2.1 Video-Mode Rendering

The simplest approach to remote rendering is to have the graphics engine render a scene from the client's viewpoint, capture this image, and transmit the image to the client for display. This approach has been used for remote scientific and medical visualization, particularly when there are no continuous inputs being modified, or where the continuous inputs are modified through user interfaces implemented at the client and then final values are transmitted to the server for rendering (State et al. 1994).

With this "video-mode," where every image seen at the client is an image rendered at and transmitted from the server, significant bandwidth is consumed (tables 8.2 and 8.5) and significant delay is unavoidable – every image is out-of-date by at least one round-trip-time. This requires the distributed control loop to operate at the same frequency as the user's screen is to update.

Transmitting raw, uncompressed video over the network has primarily been used by remote medical imaging applications where perfect image quality is necessary (Poulton 1991). The nanoManipulator does not have such stringent requirements, and compression techniques like the Motion Picture Experts Group (MPEG) standard

[1]When the user is moving a mouse cursor to point to a button in a GUI, they are making a continuous input, but clicking on the mouse to "press" that button is a discrete input.

could improve on the results reported in this experiment.

8.2.2 Image-Based Rendering

MPEG gains extra compression by transmitting an image plus some "dead reckoning" information about how groups of pixels in that image are moving. At future times, the dead reckoning information can be used to create an updated version of the image. This means the transmitted information is valid from one viewpoint at a number of different times – since consecutive frames of video are related to one another, MPEG can exploit temporal coherence.

Image-based rendering can be thought of as a form of compression that exploits spatial coherence. IBR transmits an image, plus information about the distance from the viewpoint to groups of pixels in the image. From other nearby viewpoints, the depth information can be used to create a "warped" version of the image. This means that the transmitted information is valid at one time from a number of different viewpoints – IBR exploits spatial coherence.

Depth Meshes

There are two implementations of IBR used in the nanoManipulator. The first begins with Mark et al.'s (1997) method of obtaining images with depth information: an image is rendered at the server, and both the pixel buffer and the depth buffer are read out of the hardware. These two arrays are then transmitted to the client to be warped to the user's current viewpoint and displayed.

For the nanoManipulator, further optimizations are possible. Because the AFM reports a height field, we can guarantee that there will not be any occlusion in the captured image by rendering in an orthographic projection from directly over the center of the sample. This special-case lets us avoid one of the most chronic problems of IBR, "holes" due to occlusion. The quality of the output image does not depend on how far the image is warped from its original viewpoint. The nanoManipulator also takes advantage of its fixed viewpoint at the server to minimize network bandwidth, sending to the client only the portion of the image that has changed since the previous frame. During regular steady-state scanning of the sample by the AFM, this is only the pixels corresponding to the most recently updated measurements received from the AFM. When the user makes a command that changes the appearance of the entire surface, such as changing the color map applied or turning on contour lines, the entire image must be retransmitted. This causes a huge disparity in the amount of data sent over the network – 150x to 300x in typical operation. Since the nanoManipulator's commands that change the surface appearance are discrete, this drives the difference in response time between continuous and discrete commands that appears in the tables below.

8.2.3 Textured Meshes

Since transmitting the geometry measured by the microscope requires very little bandwidth, we tested another specialized variant of IBR in the nanoManipulator. In "texture mesh" mode, the rendering server captures the pixel buffer, and transmits the changed portion to the client. The raw geometry is sent from the microscope to the client, and is textured with the image received from the rendering server. This mode was intended for cases where the client system was incapable of running the desired pixel-shading algorithm fast enough.

8.2.4 Limitations of Image-Based Rendering

Simple approaches to image-based rendering do not correctly handle specular lighting. Unfortunately, specular lighting provides significant cues about the shape of surfaces. Understanding surface shape is the goal of the nanoManipulator's rendering, and so IBR seems to be less than the ideal choice. This dependence on specular lighting is particularly true for a number of the specialized pixel-shading algorithms that we would like to support (Bokinsky 2003).

8.3 Experiments

A first round of experiments was run between two SGI machines. The server was a 300 MHz R12000 processor with InfiniteReality2E graphics. The client was a 250 MHz R4400 processor with InfiniteReality graphics. The two machines were directly connected by 100 Mbps Ethernet to the same Cabletron switch. A similar series of experiments was run between the server SGI and a client PC, a 550 MHz Pentium 3 machine running Windows NT 4 with Intense3D Wildcat 4000 graphics.

All experiments were performed using the same input data, a recorded nanoManipulator experiment manipulating a fiber of fibrin laid down on a micra substrate. The input surface was scanned at a 300 × 300 resolution.

Similar, but non-identical, sequences of user input were used for each experiment, consisting of changes of position and orientation of the surface, changes of which dataset determines the geometry or coloring of the surface, and changes of the mapping between dataset values and colors.

The data from the first experiment are given in tables 8.1, 8.2, and 8.3. The subsequent three tables give the data from the second experiment.

"Local" rendering is a reference value: the client rendering all images with no network transmission or support from the server. "Output Image Resolution" is the size of the window in which images were displayed at `nano`. "Surface Detail Resolution" is the resolution at which colormaps were displayed. "Geometry Resolution" is the number of vertices that were rendered, and thus the detail at which height information was conveyed. Although the sample was measured at a 300×300 resolution by the microscope, some of the rendering modes subsampled this set of measurements, while others effectively interpolated new points of surface detail (due to the texture hardware on the client machine being optimized for texture sizes that are powers of two).

Response time is strongly bimodal: continuous inputs tend to cause viewpoint updates, which can be immediately warped at the client, while discrete updates tend to cause regeneration of the entire surface, which requires a large transmission from the server. Recall from chapter 2 that human tolerance of latency differs significantly between the two kinds of input. Thus, in tables 8.1 and **??** I list these two submeans – Unblocked and Blocked – as well as overall mean response time.

The entries for video bandwidth in table 8.5 are surprising, until they are cross-referenced to table 8.6. In this set of experiments, bandwidth saturated between 5 and 6 Mbps because the server's rendering time was increasing linearly with increasing resolution. Table 8.2 shows a similar but less marked phenomenon.

8.4 Summary

With local rendering, the SGI maintained a response time of 125 ms, the PC 332 ms. The SGI client had mean response time below 100 ms for roughly equivalent-quality modes of both depth and texture rendering. However, the texture mesh implementation took more than twice as long to handle blocked requests than did the depth mesh implementation.

On the PC client, both depth and texture meshes had worse mean performance

Rendering Mode	Output Image Resolution	Surface Detail Resolution	Geometry Resolution	Mean Response Time (ms)	Mean Unblocked Response Time (ms)	Mean Blocked Response Time (ms)
Local	1024×768	300×300	300×300	125	125	n/a
Depth	512×512	100×100	100×100	27	20	2721
Depth	512×512	200×200	200×200	47	30	3639
Depth	512×512	300×300	300×300	93	47	5119
Depth	512×512	512×512	512×512	325	167	11567
Texture	512×512	128×128	300×300	80	60	2072
Texture	512×512	256×256	300×300	95	70	2284
Texture	512×512	512×512	300×300	134	79	2558
Texture	512×512	512×512	100×100	30	21	2384
Texture	512×512	512×512	200×200	45	28	2428
Texture	512×512	512×512	300×300	134	79	2558
Video	128×128	300×300	300×300	187	n/a	187
Video	256×256	300×300	300×300	368	n/a	369
Video	512×512	300×300	300×300	963	n/a	993
Video	1024×768	300×300	300×300	2297	n/a	2343

Table 8.1: Comparison of response time of various remote rendering techniques and resolutions running on SGI hardware.

than local rendering. These mean times were strongly bimodal, however, and response time to unblocked inputs (changes of viewpoint) was significantly improved.

Both IBR modes were faster than and consumed less bandwidth than the video mode. The response time of video mode was so poor that its specular cues were worthless, obviating one of the expected disadvantages of IBR.

Rendering Mode	Output Image Resolution	Surface Detail Resolution	Geometry Resolution	WAN Bandwidth Required (kbps)
Local	1024×768	300×300	300×300	n/a
Depth	512×512	100×100	100×100	43.2
Depth	512×512	200×200	200×200	187
Depth	512×512	300×300	300×300	337
Depth	512×512	512×512	512×512	685
Texture	512×512	128×128	300×300	37.8
Texture	512×512	256×256	300×300	115
Texture	512×512	512×512	300×300	600
Texture	512×512	512×512	100×100	526
Texture	512×512	512×512	200×200	560
Texture	512×512	512×512	300×300	600
Video	128×128	300×300	300×300	3,740
Video	256×256	300×300	300×300	8,340
Video	512×512	300×300	300×300	11,600
Video	1024×768	300×300	300×300	14,000

Table 8.2: Comparison of bandwidth requirements of various remote rendering techniques and resolutions running on SGI hardware.

Rendering Mode	Output Image Resolution	Surface Detail Resolution	Geometry Resolution	Server Mean Rendering Time (ms)	Client Mean Rendering Time (ms)
Local	1024×768	300×300	300×300	n/a	121
Depth	512×512	100×100	100×100	109	19
Depth	512×512	200×200	200×200	94	28
Depth	512×512	300×300	300×300	96	53
Depth	512×512	512×512	512×512	129	208
Texture	512×512	128×128	300×300	90	56
Texture	512×512	256×256	300×300	94	60
Texture	512×512	512×512	300×300	74	66
Texture	512×512	512×512	100×100	108	20
Texture	512×512	512×512	150×150	118	24
Texture	512×512	512×512	300×300	74	66
Video	128×128	300×300	n/a	116	4
Video	256×256	300×300	n/a	208	9
Video	512×512	300×300	n/a	571	37
Video	1024×768	300×300	n/a	1370	112

Table 8.3: Comparison of rendering performance of various remote rendering techniques and resolutions running on SGI hardware.

Rendering Mode	Output Image Resolution	Surface Detail Resolution	Geometry Resolution	Mean Response Time (ms)	Mean Unblocked Response Time (ms)	Mean Blocked Response Time (ms)
Local	1024×768	300×300	300×300	332	322	n/a
Depth	512×512	100×100	100×100	61	19	2986
Depth	512×512	200×200	200×200	179	46	3707
Depth	512×512	300×300	300×300	383	123	6828
Depth	512×512	512×512	512×512	789	357	12458
Texture	512×512	128×128	300×300	296	121	4343
Texture	512×512	256×256	300×300	452	147	3893
Texture	512×512	512×512	300×300	390	146	4342
Texture	512×512	512×512	100×100	72	30	4238
Texture	512×512	512×512	200×200	115	40	4898
Texture	512×512	512×512	300×300	390	146	4342
Video	128×128	300×300	300×300	237	n/a	237
Video	256×256	300×300	300×300	592	n/a	592
Video	512×512	300×300	300×300	1660	n/a	1663
Video	1024×768	300×300	300×300	4125	n/a	4130

Table 8.4: Comparison of response time of various remote rendering techniques and resolutions running on PC hardware.

Rendering Mode	Image Resolution	Geometry Resolution	WAN Bandwidth Required (kbps)
Local	n/a	300 × 300	n/a
Depth	100 × 100	100 × 100	40.5
Depth	200 × 200	200 × 200	339
Depth	300 × 300	300 × 300	549
Depth	512 × 512	512 × 512	859
Texture	128 × 128	300 × 300	58.9
Texture	256 × 256	300 × 300	219
Texture	512 × 512	300 × 300	662
Texture	512 × 512	100 × 100	768
Texture	512 × 512	150 × 150	709
Texture	512 × 512	300 × 300	662
Video	128 × 128	n/a	3,470
Video	256 × 256	n/a	5,150
Video	512 × 512	n/a	5,230
Video	1024 × 768	n/a	5,940

Table 8.5: Bandwidth requirements of various remote rendering techniques and resolutions running on PC hardware.

Rendering Mode	Output Image Resolution	Surface Detail Resolution	Geometry Resolution	Server Mean Rendering Time (ms)	Client Mean Rendering Time (ms)
Local	1024 × 768	300 × 300	300 × 300	n/a	178
Depth	512 × 512	100 × 100	100 × 100	84	16
Depth	512 × 512	200 × 200	200 × 200	110	58
Depth	512 × 512	300 × 300	300 × 300	132	133
Depth	512 × 512	512 × 512	512 × 512	141	385
Texture	512 × 512	128 × 128	300 × 300	112	141
Texture	512 × 512	256 × 256	300 × 300	103	169
Texture	512 × 512	512 × 512	300 × 300	104	175
Texture	512 × 512	512 × 512	100 × 100	137	49
Texture	512 × 512	512 × 512	150 × 150	94	56
Texture	512 × 512	512 × 512	300 × 300	104	175
Video	128 × 128	300 × 300	n/a	125	17
Video	256 × 256	300 × 300	n/a	325	7
Video	512 × 512	300 × 300	n/a	1163	49
Video	1024 × 768	300 × 300	n/a	4348	256

Table 8.6: Comparison of various remote rendering techniques and resolutions running on PC hardware.

Network Layer	Technique
Presentation	**Video-mode remote rendering** (8.2.1)
	Image+Depth remote rendering (8.2.2)

Table 8.7: Proposed adaptations for remote graphics systems. Those in boldface were used in the experiments discussed in this chapter.

Chapter 9

Conclusion

This dissertation evaluated presentation- and session-level adaptations that let the nanoManipulator, a local-area network application, function effectively when distributed across a wide-area network. In engineering the distributed nanoManipulator for the Internet, the chief difficulty was latency. Latency interferes with using a remote microscope or a remote rendering engine and makes it difficult to work synchronously with distant collaborators. From the set of latency-compensation techniques that were found to be effective, we can identify three prevalent themes that emerged in designing and implementing an interactive collaborative distributed teleoperator: avoid blocking, choose an appropriate intermediate representation, and minimize the frequency with which distributed feedback loops must run.

Distributing an interactive application across a wide-area network can be a very difficult proposition. If the original implementation was not done with distribution in mind, a complete rewrite may be necessary. Not only does the code need to be refactored into a distributed system, every algorithm needs to be considered for appropriateness. For the distributed nanoManipulator, the three critical questions to ask of every algorithm were:

- Does it introduce unnecessary blocking?
- Does it use an appropriate intermediate representation?
- Does it distribute a feedback loop across a network connection?

No one technique is suitable for all interfaces or all distributed configurations. For example, FEC controlled packet loss without the latency penalty of TCP's reactive reliability, giving us another network protocol option for haptics. FEC assumes errors are common and provides a constant-overhead mechanism to statistically reduce the error rate. Contrast this with collaboration, which under identical network conditions benefited from optimistic concurrency control. Optimistic approaches assume errors are rare and reduces the overhead of normal, correct operation by relying on expensive recovery mechanisms. Thus, we can not simply endorse optimistic or pessimistic approaches – each class of interface must be considered independently.

Portions of this work have been published in (Hudson et al. 2000), (Hudson et al. 2001), (Jeffay et al. 2001), (Hudson et al. 2003), and (Hudson et al. 2004).

9.1 Results

My thesis in this dissertation is that using appropriate intermediate representations allows feedback-critical applications to be distributed across wide areas. I have shown five results that support this thesis:

Analysis of Force Feedback: Chapter 4 showed how we could apply Azuma and Bishop's analytic methods to force feedback, relating previous studies of the latency tolerance of position-controlled teleoperation to the latency tolerance of the plane approximation. It explains the dependence of system stability on surface roughness, speed of motion, and the latency in the teleoperator. It proposes a new metric, $v()$, and discusses its strengths and weaknesses compared to Adachi et al.'s $d()$. This technique allows us to mathematically evaluate the latency tolerance of a wider range of intermediate representations than previously tractable.

Intermediate Representations for Force Feedback: Chapter 5 introduced two new intermediate representations for teleoperation, discussing their motivation and relating them to previous work. These representations perform well under operating conditions where force-feedback teleoperation was previously infeasible due to latency.

- Local environment sampling, or "feelahead" mode, which increases the number of measurements taken by the remote effector for every position commanded in order to decrease the frequency with which measurements of the environment must be transmitted across the network.
- The warped plane approximation, which emphasizes correctly conveying the shape of individual features on the specimen over conveying the absolute position of features.

Network Adaptations for Force Feedback: Chapter 6 discussed experiments that verified that using UDP, FEC, and QM work as well for the force feedback data of the nanoManipulator as for traditional audio and video streams in changing network behavior, even though the semantics of the force feedback data are quite different from those of traditional streaming media. UDP reduced latency by 35% and jitter by 75% compared to TCP. FEC on top of UDP reduced loss tenfold by doubling bandwidth, or one-hundred-fold by quadrupling bandwidth. QM on top of UDP reduced the drop rate asymptotically close to zero with less than a 10% increase in mean latency. The chapter also summarizes the results of a study comparing both $e(s)$ and traditional application-level audio quality metrics for TCP, UDP, and FEC.

Optimistic Concurrency Control for Collaboration: Chapter 7 compared optimistic and pessimistic approaches to concurrency control for the distributed nanoManipulator. Optimism reduced the latency penalty from concurrency control by 85%, which will allow for more network latency to be tolerated. The chapter also carefully distinguishes the portion of the application state that can be treated optimistically from that which can not, and discusses using multiple instances of a Model View Controller (MVC) architecture to take advantage of the distinction.

Image-Based Rendering for Remote Graphics: Chapter 8 showed that texture- and depth mesh-based image based rendering techniques could provide a better response time than video-based remote rendering, while consuming less bandwidth. For similar-quality renderings, the SGI hardware produced one tenth the mean response time when used in an IBR mode as it did in a video mode.

Using these five methods reduced response time for remote rendering and collaboration, reduced haptic display error, and increased satisfaction of both haptic and collaborative users. The combination of techniques allows the nanoManipulator to be successfully distributed across a wide-area network.

9.2 Future Work

The frequency-mode analysis discussed in chapter 4 requires information about the spectrum of motion along a path. This data was not reported in previous studies of position-control force feedback; obtaining it would allow the relationship between position control and the plane approximation or other intermediate representations to be more precisely characterized. Frequency analysis can be applied to more complex intermediate representations, although the mathematics become correspondingly more complex. Neither metric, $d(s)$ or $e(s)$, is completely general or satisfactory.

The distributed nanoManipulator was a relatively easy target for optimistic concurrency control. Systems where collaborating users can interact in more complex ways will provide more challenges, requiring more elaborate algorithms for resolving conflicts than the nanoManipulator used, but should still be well-suited to it. (Significantly more complex systems have used optimistic concurrency control, although most have not been feedback-critical or taken continuous input.) Some of this work is already occurring in the games industry, but has not been studied or documented in the academic literature.

Not long after this work began, the computer science community began denigrating remote rendering as unnecessary because of the wide availability of high-performance graphics hardware for PCs. However, recent emphasis on Grid computing underscores the fact that massive datasets and complex simulations are not widely distributable (Karonis et al. 2003). For the Grid, we may want to do the rendering at a server and send it over the network not because the client computer is incapable of doing the rendering, but because it is incapable of doing the computation that drives the rendering or managing the data that drives the computation (Reed 2003). As the capabilities of hardware and the demands of problems evolve, the balance of power between remote and local rendering will continue to shift. Volume rendering is currently an excellent target for the application of these techniques (Norton and Rockwood 2003).

Finally, the haptic adaptation in this dissertation was manual: when the user realized that force feedback was problematic, she could shift to a different intermediate representation, or change the network algorithms used. Automatic adaptation has been implemented for the simpler network adaptations in multimedia, and would make users of the distributed nanoManipulator even less aware of network latency.

9.3 Summary

In this dissertation I introduced new intermediate representations for haptics that tolerate more latency than the previous work, showed how adaptations in the transport layer of the network stack could increase the feasibility of distributed haptics by reducing latency, jitter, and loss, and showed how optimistic concurrency control and image-based rendering provide better response time than traditional approaches to distributing collaboration and rendering. They combine to make the distributed nanoManipulator feasible; shared live microscopy has been conducted over hundreds of miles, and shared review of prior experiments over thousands of miles. This increase in our ability to carry out distributed collaborative science is due to the use of latency-tolerant operation semantics and intermediate representations.

Appendix A

Network Time Budgets

Even dedicated or QoS-enabled networks to all of our collaborators would not be enough to provide the performance our application requires. As we saw in our Internet2 trial to Microsoft Research (section 1.1), a high-speed network without significant congestion can still have a significant, unavoidable delay. In a store-and-forward network, four components contribute to latency: routing, queueing, transmission, and propagation. Each router requires some time to determine where next to send a packet, even if they are advertised as routing "at wire speeds." If the network is congested, packets may spend time waiting in queues before they can be sent out over their outgoing link. Transmitting each bit on the network takes a small amount of time; [1] signals "on the wire" then must physically propagate to the next router. An increase in bandwidth can directly reduce transmission time, indirectly reducing queueing time, and improved hardware can reduce routing time, but propagation time is limited by the speed of light. Consider a best-case scenario today: assume a direct one gigabit network connection between UNC in North Carolina and Microsoft Research in Washington state, a wire distance of 2800 miles, passing through ten routers. At each router a packet queues for one transmission delay; the router then spends 10 ns computing its route before beginning to transmit it. Sending 60 byte packets (including all headers) over this network requires 100 ns for routing ($\frac{\text{10 ns}}{\text{route computation}} \times 10$ routers), 480 ns for transmission ($\frac{\text{1 ns}}{\text{bit}} \times \frac{\text{480 bits}}{\text{packet}}$), 12 ms for propagation ($\frac{\text{300,000 kilometers}}{\text{second}} \times 3600$ kilometers), and 4.8 μs for queueing ($\frac{\text{480 ns}}{\text{packet}} \times \frac{\text{1 packet}}{\text{router queue}} \times 10$ routers). This network, better than today's configuration, is already limited by the speed of light to no more than 41 round trips per second and 24 ms of network-originated latency. Actual speed of signals through a network is slower than the speed of light in a vaccuum; electrical signals propagate slower in copper, while optical fiber requires repeaters and amplifiers that convert from light

[1] The time is proportional to the link speed, so on a low-bandwidth network this may actually be a significant amount of time: a fast modem (56kb) or ISDN line (64kb) requires about 16μs per bit.

to electrical signals and back. With a congested network, queueing delay can increase fiftyfold; most of today's networks will be ten to one hundred times slower to transmit, thus multiplying queueing delay another ten to one hundred times. A ten megabit network link with similar routers would instead require 48μs for packet transmission ($\frac{100\text{ ns}}{\text{bit}} \times \frac{480\text{ bits}}{\text{packet}}$); with an average of twenty packets in each router queue another 19.2 ms would be added to response time ($\frac{48\ \mu\text{s}}{\text{packet}} \times \frac{20\text{ packets}}{\text{router queue}} \times 10$ routers).

The current state of the Internet is significantly worse than this ideal 24 to 43 ms of delay per round-trip. Niemeyer and Slotine (1998) measured a mean round-trip time greater than 100 ms between MIT and a site in California; the delay was nowhere near constant, with variance of 50%. There was also a strong 10 Hz component to the delay variation.

Appendix B

Design and Implementation of the Distributed nanoManipulator

Design Given the functions desired in a distributed application and a set of computers to host the system, we can view design as determining how functions should be divided between processes and processes divided between hosts. This **partitioning**, and the choice of which algorithms are used to implement the desired functions, determines the application's network requirements. The problem of balancing communication with computation seen here is the same as that found in parallel systems. Placing all function into one process removes all network requirements but may place too large a computational load on the host processor, while splitting function up into many small processes may cause the network requirements to grow too large. Designers of interactive applications need to be especially careful with communication costs, because network latency impedes interactivity.

In designing the distributed nanoManipulator we have little flexibility to choose a partition. The application's goal is to give access to remote resources. We need to assign at least one process to each site, and then determine what demands those processes place on the network.

The distributed nanoManipulator centers around the user interface process, `nano`. If the nanoManipulator is being used to review old experiments, no additional processes are necessary; if it is being used to run an experiment, `nano` closely interacts with `topo`, the microscope control program. When multiple scientists are collaborating, one `nano` process serves each scientist. Remote rendering also requires a `render` process for each `nano`. Figure B.1 shows the distributed nanoManipulator deployed across a wide-area network to conduct a collaborative experiment and use remote rendering, which requires one `topo`, two `nano`, and two `render` processes.

For each `nano`, an additional minor process controls the Phantom force-feedback device, usually spawned as an extra thread within `nano` or on a machine connected to `nano` by a LAN. Other minor processes may control additional devices, such as an

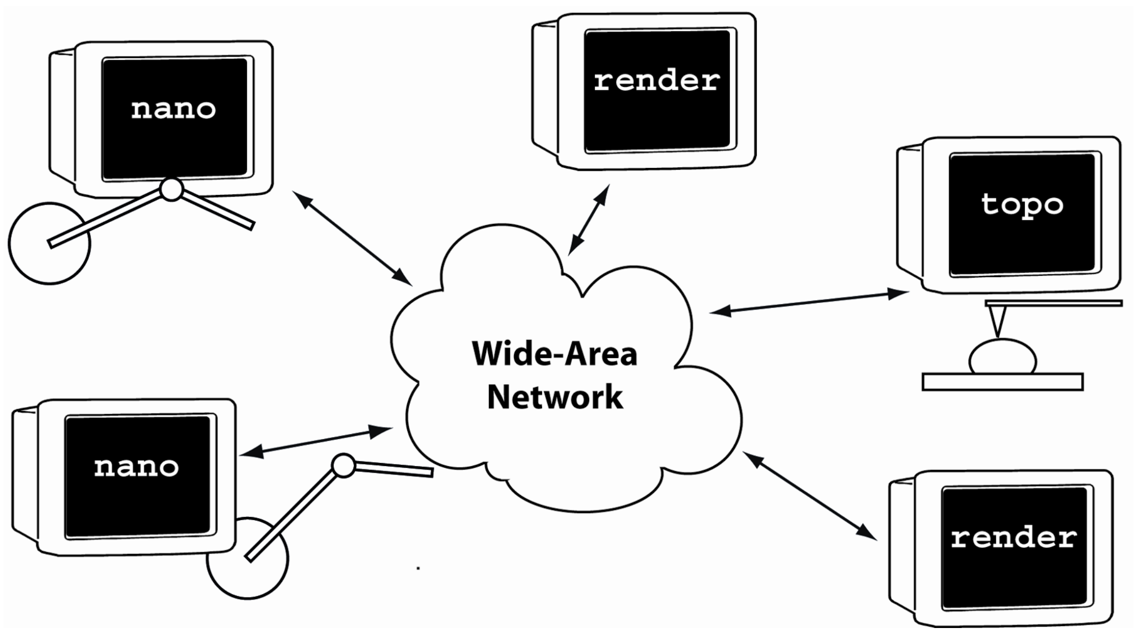

Figure B.1: Deployment diagram for the distributed nanoManipulator. This diagram shows two scientists engaged in an experiment, both using remote rendering servers. One `topo`, two `nano`, and two `render` processes are spread across a wide-area network.

ohmmeter, or parallelize complicated tasks; these processes are typically colocated with the microscope (and thus connected to `topo` by a LAN). Although some of these processes have data streams not easily distributed interactively across a WAN, they are not discussed in this dissertation.

The basic data flows required by the nanoManipulator are given in Figure B.2 and Table B.1. During an experiment, each `nano` requires a medium-bandwidth stream to the `topo` for imaging a sample and automatic manipulations, but a low-bandwidth low-latency stream for human-guided manipulations of the sample. To collaborate, the two `nano` processes must be connected by another low-bandwidth low-latency stream. Remote graphics makes more stringent demands, since a `nano` and its `render` require a high-bandwidth low-latency stream. Since we have little flexibility in choosing how we partition functions across the network, this dissertation concentrates on ways to change the algorithms we use in the distributed nanoManipulator to reduce the demands the system makes on the network.

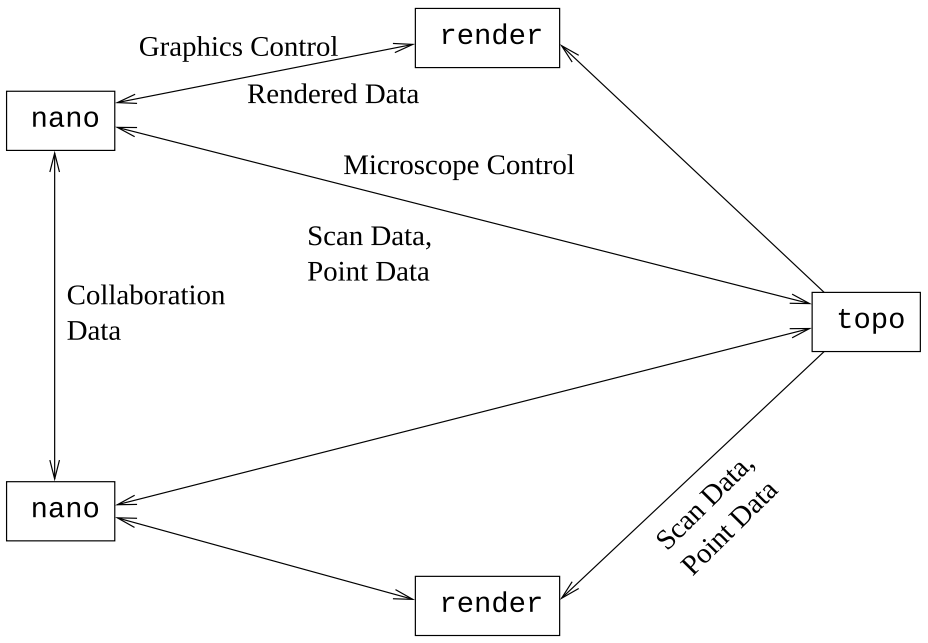

Figure B.2: Data flow diagram for the distributed nanoManipulator. Each stream's requirements are given in Table B.1.

Implementation Often, maintaining a program is more difficult and consumes more resources than the original design. Programs tend to become significantly more complex as additional features are added that were not planned for in the original design. This was already a problem with the nanoManipulator, so I began with a regimen of "preventative maintenance" on the `nano` process before attempting to build the distributed nanoManipulator.

To document the software system, I refer to a number of software patterns. A software pattern is a named solution to a common software problem. Identifying the fact that a certain approach has been widely and succesfully used across numerous systems to solve the same problem is a significant contribution to the art of programming; giving it a standardized name allows practitioners to discuss it meaningfully and makes it easier to teach. The current standard reference work for software patterns is Gamma et al. (1995), the "Gang of Four" book; all patterns which do not explicitly cite some other source are taken from that book. (Gamma et al.) also popularized a standard form for describing patterns; the essential components are a name, a description of the problem solved, a description of the solution, and a description of the consequences, especially the trade-offs. The pattern community hopes that software patterns can be assembled into a language, similar to the architectural

Data Stream	Endpoints	Bandwidth (bps)	Message Rate (per second)	Approximate Latency Tolerance
Microscope Control	`nano` ⇒`topo`	6,080	10	1 second
Manual Microscope Control	`nano` ⇒`topo`	10,880	20	50 ms
Scan Data	`topo` ⇒`nano`, `topo` ⇒ `render`	20,672	1	1 second
Point Data	`topo` ⇒`nano`	192,000	250	1 second
Manual Point Data	`topo` ⇒`nano`	15,360	20	50 ms
Collaboration Data	`nano` ⇔`nano`	7,680	10	250 ms
Graphics Control	`nano` ⇒`render`	19,840	20	50 ms
Rendered Data	`render` ⇒`nano`	18,284,544	2,048	50 ms

Table B.1: Peak bandwidth, packet rate, and latency requirement for major data streams in the distributed nanoManipulator before applying latency-tolerance techniques and adding adaptive networking. "Manual" data streams occur when the user interactively guides surface modifications using the PHANTOM. Bandwidth figures include overhead of TCP/UDP and IP headers assuming no fragmentation occurs.

pattern language of Alexander (1977).

At the beginning of my work, the nanoManipulator was a large program, primarily written in C, with many global variables, low cohesion, and high coupling. Cohesion is unity of purpose within components: class A will be simpler to understand and maintain if it tries to do one thing, not ten things. Coupling is dependence between components: for example, when file A requires file B, otherwise entirely unrelated, in order to compile, perhaps for access to a declaration of a small struct, the two are unnecessarily coupled. A can not be tested without using B, and changes in B will require changes (or at least recompilation) of A.

I used refactoring techniques similar to those advocated by Fowler (1999), incrementally moving code around the program without trying to add new functions. Bit-by-bit this changed the nanoManipulator into a C++ program. Following the precepts of Lakos (1996), I undertook to fix the physical as well as the logical design, avoiding unnecessary dependencies between components through careful levelization[1] and insulation.[2] Bodies of C code with high cohesion were grouped together into encapsulating C++ classes, or "Wrapper Facades" (Schmidt et al. 2000). (A Wrapper

[1] "A subsystem is *levelizable* if it compiles and the graph implied by the include directives of the individual components (including the `.c` files) is acyclic" (Lakos 1996).

[2] "Insulation is the process of avoiding or removing unnecessary compile-time coupling," the physical analog of logical encapsulation (Lakos 1996).

Facade is a specialization of the general Facade pattern introduced by the Gang of Four.) The two largest subsystems wrapped in this manner were the graphics code and the interface to the microscope. I reduced coupling between these two subsystems, making them independent of one another by removing all calls from either to the other and moving all shared information onto lower-level data structures that were simple to maintain in shared memory. Throughout the nanoManipulator I applied naming conventions similar to Lakos' to further increase maintainability.

Having made the code more modular and having separated independent concerns, I began reworking these wrapped subsystems into components that could be distributed. The public interface of the Wrapper Facade was moved into a Protocol, or Abstract Base Class[3] (Meyers 1992). Another specialization of this base class was written as a "Proxy," a class that could serve to relay function calls to an object, whether it was located in another process or on another machine across the network.[4] Code to marshal and unmarshal the function call arguments was added to the base class (making it no longer abstract), and the Wrapper Facade extended to receive calls relayed across the network by the Proxy.

This tripartite division into base class (abstract Protocol with network handlers), "Remote" class (Proxy), and implementation is the standard approach of the Virtual Reality Peripheral Network (Taylor et al. 2001). I used VRPN as the underlying Session layer (See Figure B.3) for the distributed nanoManipulator, giving us asynchronous remote procedure call (RPC) capabilities. With asynchronous RPC, when a process calls a stub, the stub does not wait for values to be sent back by the remote server, but instead returns control to the calling process immediately. This decouples the application from network latency, allowing an interactive application to reduce response time for operations that are being carried out across the network. In this way, Asynchronous RPC is more useful for interactive applications than is traditional

[3]An abstract base class is a common C++ technique where a class specifies no implementation, only an interface. Subclasses inherit the interface and must supply a complete implementation. This helps lead to a clean use of inheritance, which is notoriously problematic in C++. It also increases insulation (Lakos 1996).

[4]The classical precursor of the Proxy pattern is Remote Procedure Call (RPC) (Birrell 1984). Instead of making a function call *f(a)*, we call a "stub" function *fstub(a)*. *fstub(a)* converts the parameter *a* to a "neutral" representation and sends it over the network to a server. The server executes *f(a)* and sends back its result, which *fstub()* returns to the calling process. Birrell et al. (1993) introduced Network Objects, an object-oriented version of RPC similar to the design used by VRPN.

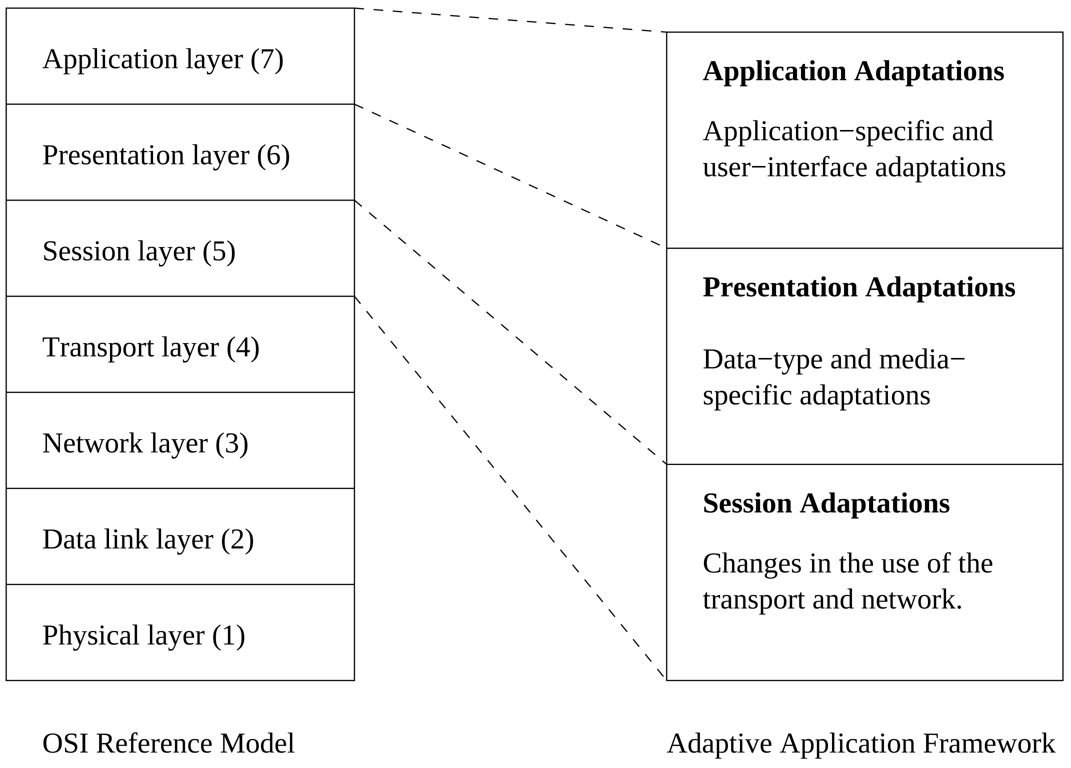

Figure B.3: The OSI Reference Model and the framework for adaptive applications I develop in this dissertation.

synchronous RPC. The one drawback to asynchronous RPC is that it typically leads to a more complex programming style centered on callbacks; however, this same style is also required by modern graphical user interfaces, and was adopted for most of the distributed nanoManipulator.

The original nanoManipulator used an unorganized collection of global variables for its user interface. When we added collaboration to the system (Chapter 7), user interface management code became necessary. I created a flexible hierarchy that allow executing operations on a portion or the entirety of the user interface by calling functions on the hierarchy elements. This approach is related to the Composite pattern, but less extreme.

The distributed nanoManipulator uses an architecture in line with Wloka's (1995) recommendations (Chapter 3): one thread dedicated to each high-rate input and output device (the PHANTOM and the AFM), one thread dedicated to graphics, and the application and low-rate input and output services in a final independent thread. Our normal deployment does not have enough processors to map each thread to its own CPU, so the graphics and application threads are merged, but Chapter 8 discusses the possibilities when they are independent processes, whether on one computer or separated by the network.

B.1 The Microscope Simulator

To increase repeatability and reduce risk, instead of connecting to a live AFM with a real sample, the quantitative experiments discussed in this dissertation connected to a simulator program known as `afmsim`. This program shares its high-level C++ classes with `topo`. However, instead of a device controller, the underlying implementation is software that reads in a data file from a previous experiment and emulates an instance of `topo` reporting the same data. `afmsim` repeatedly "scans" the surface, sending back reports using inter-report timing sampled from `topo`. It accepts commands from `nano`, although most are silently ignored. The value of `afmsim` is that having once scanned a sample in the AFM, we can be sure that experiments evaluating `nano`'s treatment of that sample will always get the same data, without the risk of a tip breaking or a sample being contaminated that would happen were we to run the experiments with an AFM but with more flexibility to change the events of the experiment than we would have replaying the original data file. Thus, we can apply local environment sampling to a dataset that was gathered using standard point sampling and the plane approximation, something not possible using the nanoManipulator's standard replay capability.

Appendix C

Fourier Analysis: Detailed Calculations

Let the phase of $Z(\sigma)$ be ϕ, and the phase of $H(\sigma)$, α. Then phase shift is $\tan(\alpha - \phi)$. Axioms, definitions, and basic algebraic manipulations:

$$\begin{aligned}
e^{-j\sigma\triangle s} &= \cos(\sigma\triangle s) - j\sin(\sigma\triangle s) \\
\|x_1x_2\| &= \|x_1\|\,\|x_2\| \\
\|x\| &= \sqrt{\Re(x)^2 + \Im(x)^2} \\
\mathcal{F}(z(s-\triangle s)) &= e^{-j\sigma\triangle s}Z(\sigma) \\
\mathcal{F}(z'(s-\triangle s)) &= j\sigma e^{-j\sigma\triangle s}Z(\sigma) \\
Z(\sigma) &= x + jy \\
\phi &= \arctan\frac{y}{x} \\
\alpha &= \arctan\frac{\Im(H(\sigma))}{\Re(H(\sigma))} \\
\tan(\alpha-\phi) &= \frac{\tan\alpha - \tan\phi}{1+\tan\alpha\tan\phi} \\
&= \frac{\frac{\Im(H(\sigma))}{\Re(H(\sigma))} - \frac{y}{x}}{1+\frac{y}{x}\frac{\Im(H(\sigma))}{\Re(H(\sigma))}} \\
&= \frac{x\Im(H(\sigma)) - y\Re(H(\sigma))}{x\Re(H(\sigma)) + y\Im(H(\sigma))}
\end{aligned}$$

Direct Force Feedback:

$$\begin{aligned}
h(s) &= z(s-\triangle s) \\
H(\sigma) &= Z(\sigma)e^{-j\sigma\triangle s} \\
\|H(\sigma)\| &= \|Z(\sigma)\|\,\|e^{-j\sigma\triangle s}\| \\
&= \|Z(\sigma)\|\sqrt{(\cos(\sigma\triangle s))^2 + (-\sin(\sigma\triangle s))^2}
\end{aligned}$$

$$\begin{aligned}
&= \|Z(\sigma)\|\sqrt{\cos^2(\sigma\triangle s) + \sin^2(\sigma\triangle s)} \\
&= \|Z(\sigma)\| \\
v(s) &= z(s - \triangle s) - z(s) \\
V(\sigma) &= Z(\sigma)e^{-j\sigma\triangle s} - Z(\sigma) \\
&= Z(\sigma)(e^{-j\sigma\triangle s} - 1) \\
\|V(\sigma)\| &= \|Z(\sigma)\|\ \|(e^{-j\sigma\triangle s} - 1)\| \\
&= \|Z(\sigma)\|\sqrt{(\cos(\sigma\triangle s) - 1)^2 + (-\sin(\sigma\triangle s))^2} \\
&= \|Z(\sigma)\|\sqrt{(\cos^2(\sigma\triangle s) - 2\cos(\sigma\triangle s) + 1 + \sin^2(\sigma\triangle s))} \\
&= \|Z(\sigma)\|\sqrt{2 - 2\cos(\sigma\triangle s)} \\
\tan(\alpha - \phi) &= -\sigma\triangle s
\end{aligned}$$

Plane Approximation:

$$\begin{aligned}
h(s) &= z(s - \triangle s) + \triangle s\ z'(s - \triangle s) \\
H(\sigma) &= e^{-j\sigma\triangle s}Z(\sigma) + j\sigma\triangle s e^{-j\sigma\triangle s}Z(\sigma) \\
&= Z(\sigma)e^{-j\sigma\triangle s}(1 + j\sigma\triangle s) \\
&= (\cos(\sigma\triangle s) - j\sin(\sigma\triangle s))Z(\sigma) \\
&\quad + j\sigma\triangle s(\cos(\sigma\triangle s) - j\sin(\sigma\triangle s))Z(\sigma) \\
&= Z(\sigma)(\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s) \\
&\qquad + j(\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s))) \\
\|H(\sigma)\| &= \sqrt{\Re(H(\sigma))^2 + \Im(H(\sigma))^2} \\
\Re(H(\sigma)) &= Z(\sigma)(\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s)) \\
\Im(H(\sigma)) &= Z(\sigma)(\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s)) \\
\|H(\sigma)\| &= \|Z(\sigma)\|\sqrt{(\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s))^2 + (\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s))^2} \\
&= \|Z(\sigma)\|(\cos^2(\sigma\triangle s) + (\sigma\triangle s)^2\sin^2(\sigma\triangle s) + 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s) \\
&\qquad + (\sigma\triangle s)^2\cos^2(\sigma\triangle s) + \sin^2(\sigma\triangle s) - 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s))^{1/2} \\
&= \|Z(\sigma)\|\sqrt{1 + (\sigma\triangle s)^2} \\
v(s) &= h(s) - z(s) \\
V(\sigma) &= H(\sigma) - Z(\sigma) \\
&= Z(\sigma)(e^{-j\sigma\triangle s}(1 + j\sigma\triangle s) - 1) \\
&= (\cos(\sigma\triangle s) - j\sin(\sigma\triangle s))Z(\sigma)
\end{aligned}$$

$$\begin{aligned}
& +j\sigma\triangle s(\cos(\sigma\triangle s) - j\sin(\sigma\triangle s))Z(\sigma) - Z(\sigma) \\
= \; & Z(\sigma)((\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s) - 1) \\
& \quad + j(\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s))) \\
\|V(\sigma)\| = \; & \sqrt{\Re(V(\sigma))^2 + \Im(V(\sigma))^2} \\
= \; & \|Z(\sigma)\|\sqrt{(\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s) - 1)^2 + (\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s))^2} \\
= \; & \|Z(\sigma)\|(\cos^2(\sigma\triangle s) + 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s) \\
& \quad - 2\cos(\sigma\triangle s) + (\sigma\triangle s)^2\sin^2(\sigma\triangle s) - 2\sigma\triangle s\sin(\sigma\triangle s) \\
& \quad + 1 + (\sigma\triangle s)^2\cos^2(\sigma\triangle s) - 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s) + \sin^2(\sigma\triangle s))^{1/2} \\
= \; & \|Z(\sigma)\|(\cos^2(\sigma\triangle s) + \sin^2(\sigma\triangle s) + 1 \\
& \quad + 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s) - 2\sigma\triangle s\cos(\sigma\triangle s)\sin(\sigma\triangle s) \\
& \quad + (\sigma\triangle s)^2\cos^2(\sigma\triangle s) + (\sigma\triangle s)^2\sin^2(\sigma\triangle s) - 2\cos(\sigma\triangle s) \\
& \quad - 2\sigma\triangle s\sin(\sigma\triangle s))^{1/2} \\
= \; & \|Z(\sigma)\|\sqrt{2 + (\sigma\triangle s)^2 - 2\cos(\sigma\triangle s) - 2\sigma\triangle s\sin(\sigma\triangle s)} \\
\tan(\alpha - \phi) = \; & \frac{\sigma\triangle s\cos(\sigma\triangle s) - \sin(\sigma\triangle s)}{\cos(\sigma\triangle s) + \sigma\triangle s\sin(\sigma\triangle s)}
\end{aligned}$$

Appendix D

The Logical Clock Algorithm

A logical clock can be used to create a total ordering of messages in a distributed system (Lamport 1978; Babaoglu and Marzullo 1993). Processes are event-driven, with three types of events: sends of messages to other processes, receives of messages from other processes, and internal events that do not directly involve communication but may cause messages to be sent or be caused by received messages. Every process maintains a local variable LC. Every message sent is given a timestamp, $TS(m)$, equal to $LC(e)$, the value of LC when m is sent. When an event e occurs, LC is updated:

$$LC = \begin{cases} LC + 1 & \text{if } e \text{ is a send or internal event} \\ max(LC, TS(m)) + 1 & \text{if } e \text{ is a receive event} \end{cases}$$

Events are consistently ordered by their timestamps, which increase monotonically. To obtain a causal ordering of events (if e^1 causes e^2 in some process, then all processes receive all messages sent by e^1 before any sent by e^2), messages received must not be executed immediately. Instead, once a process has received messages with timestamps greater than $TS(m)$ from every other process except the sender of m, message m is "stable" and can be executed.

Appendix E

The Vector Clock Algorithm

The vector clock algorithm is a more complex approach to generating a total order than the logical clock algorithm, but it also provides more efficient causal execution, reducing the need to wait for messages from other processes (Babaoglu and Marzullo 1993).

Every process maintains a local array variable VC. Every message sent is given a timestamp, $TS(m)$, equal to $VC(e)$, the value of VC when m is sent. When an event e occurs at process i, VC is updated:

$$\begin{array}{ll} VC[i] = VC[i] + 1 & \text{if } e \text{ is a send or internal event} \\ & \\ VC = \max(VC, TS(m)) & \text{if } e = \textit{receive}(m) \\ VC[i] = VC[i] + 1 & \end{array}$$

Causal ordering is guaranteed by only executing at process i a message from process j after executing all events that had been executed at process j before the message was sent. The information needed to determine whether or not this is true is encoded in VC_i, the virtual clock at process i, $TS(e_j^x)$, the message's timestamp, and knowledge of which process originated the timestamp. First, if the previous event from j, e_j^{x-1}, has already been executed, then $VC_i[j] = TS(e_j^x)[j] - 1 = x - 1$. Second, if all other events that were executed at j before e_j^x have been executed at i, then $VC_i[k] \geq TS(e_j^x)[k]$ for all $k \neq j$.

Appendix F

The Model-View-Controller Paradigm

The Model-View-Controller (MVC) paradigm (Krasner and Pope 1988) was first introduced as a standard structure for applications in the Smalltalk programming language, and has since been widely used to build many types of interactive applications, including collaborative tools or "groupware" (Graham et al. 1996). It specifies two facets of an architecture: the division of function among modules and the pattern of communication between modules. Application data and logic are grouped together in the Model, input-handling functions into the Controller, and output into the View. The Model notifies both View and Controller if any of its data changes, and they are responsible for determining whether or not the change is relevant to them (figure F.1).

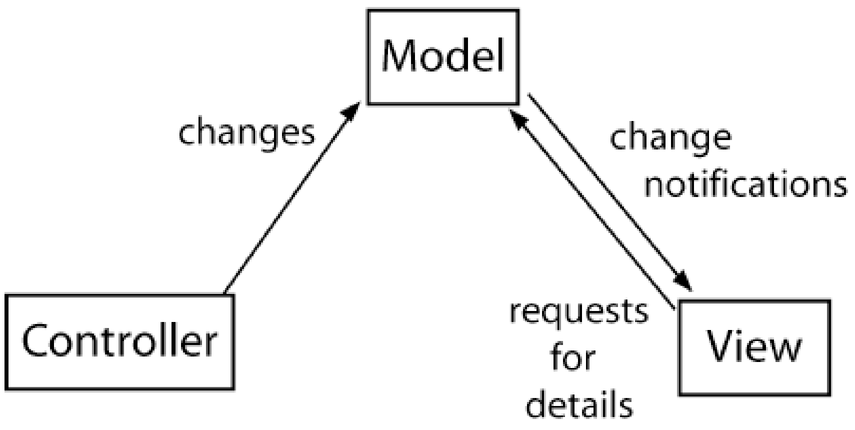

Figure F.1: Basic interactions between components in the Model-View-Controller paradigm. Controllers send changes to Models; Models informs Views of changes, and interested Views request details.

There are several reasons for this decomposition. This formalized architecture provides a clear separation of concerns between application logic and application interface, while allowing the interface (both View and Controller) to be specialized to the Model as necessary. Where specialization is not necessary, it promotes reuse. For a modern example, Java Swing user interface components are based on a modified MVC pattern to increase their breadth of applicability.

The communication pattern of Model-View-Controller is more suited to implementation on a single machine than it is to being distributed across a network. Every update from a model to a remote view or controller has at least three messages cross the network: a notification from the model, a query from the remote machine, and then the actual data from the model. Several groups have made extensions or optimizations that improve its performance (Schuckmann et al. 1999). The most common approach is to allow the model to push data preemptively to the remote modules along with the change notification. Either (1) all changes are broadcast to all dependents, or (2) dependents register in advance their interest in specific types of data with the Model, which then sends them only the updates they are interested in. The first option increases the bandwidth used by the system, while the second increases its complexity (figure F.2).

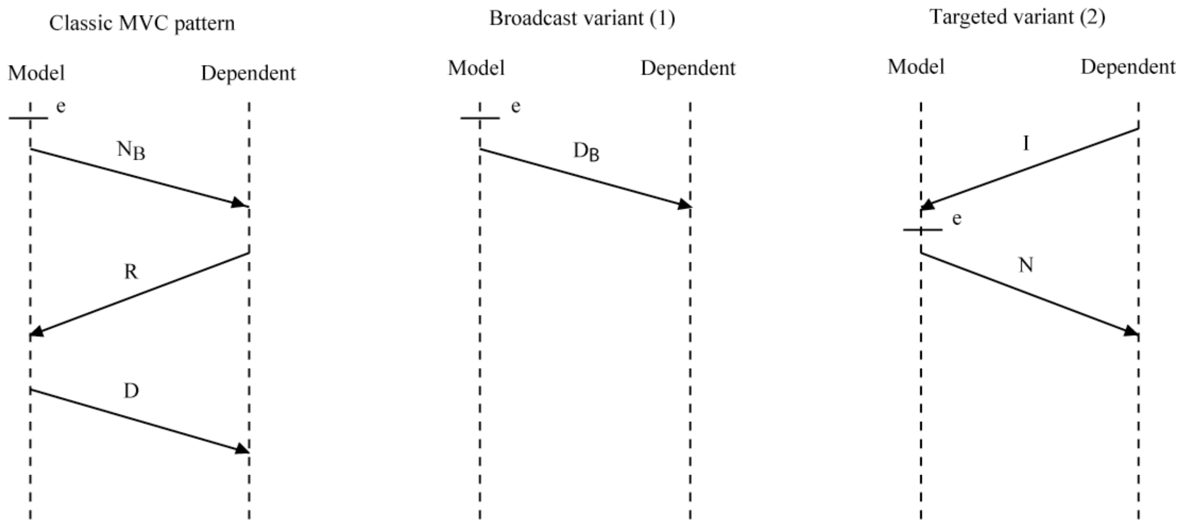

Figure F.2: Three implementations of the Model-View-Controller paradigm. The classic design requires moderate bandwidth and has high latency; typical improvements increase either bandwidth or implementation complexity to reduce latency.

The separation into Model, View, and Controller has also been used to design the division of function in a system without adhering to communication pattern specified by Krasner and Pope (1988). It is in this sense that we speak of an MVC decompo-

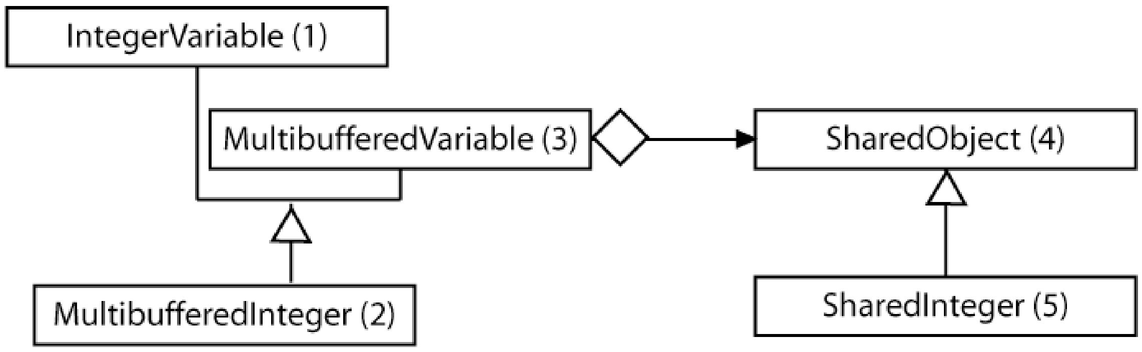

Figure F.3: A description of the multi-buffered variables in UML. Inheritance was used to extend the preexisting IntegerVariable class to have multiple states, each implemented as an instance of a SharedInteger from the VRPN library.

sition of the Collaborative nanoManipulator.

F.1 Implementing Model-View-Controller in the nanoManipulator

For its visualization-Model, the nanoManipulator uses a fine-grained object-oriented design. For each primitive value (integer, float, or string) in a Model, there is one heavyweight object, which we will call a Variable (subclassed into IntegerVariable, FloatVariable, and StringVariable). The Variable object knows how to display its value in the two-dimensional graphical user interface and accept updates to its value from the user interface. It also implements a callback system, which propagate changes of the Variables value to all interested entities. (In the language of Gamma et al. (1995), this callback system is the push variant of the OBSERVER pattern.)

We subclassed Variable, adding two buffers: one for each mode (private or shared). All concurrency control and networking concerns were handled by these buffers, so that we only needed to add to the interface of the primitive value objects the ability to select which mode was active for that object. This separation of concerns was very valuable in debugging and maintenance. When the user changes their collaboration mode, the value in an objects buffer for that mode becomes the objects value, triggering the objects associated callbacks. (These object buffers are similar to the MEMENTO pattern.)

An example is shown in figure ref fig:multibufferUML. Instances of the preexisting IntegerVariable class (1) were replaced with MultibufferedInteger (3), which inherited both from IntegerVariable and a new utility class named MultibufferedVariable (2). MultibufferedVariable abstracted out the type-independent requirements of managing multiple buffers and switching between them. The buffers were subclasses of SharedObject (4); for a MultibufferedInteger, these were instantiated as SharedIntegers (5). SharedObject and its descendants are classes that know how to keep one value synchronized over a network connection; which value is actually used at any point in time is controlled by the Multibuffered containers.

As defined, this architecture can support arbitrary numbers of users, each with their own private state and all having access to one shared state. If this system were to support additional shared states, each MultibufferedVariable would keep track of more than two SharedObjects. Variables that have special widgets in the GUI are supported by classes inheriting from MultibufferedVariable. To support datatypes other than the standard integer, float, and string, one would add additional pairs of classes descending from MultibufferedVariable and from SharedObject. (The networked SharedObject implementation uses the push, aspect variant of the OBSERVER pattern.)

The original nanoManipulator kept Variables as an unstructured "sea" of globals. We converted this into a hierarchy. Using the hierarchy to group related interface parameters together increased the maintainability of the code. It also let us present meaningful chunks of the user interface coherently to the users. For example, all variables related to the color used to render the microscope data the name of the dataset to map to color, the colormap to apply are grouped into a "color" submodel. This is grouped with a contour map submodel, lighting submodel, and others into the "viewing parameter" submodel, which, together with other high-level application state, forms the visualization Model (figure F.4).

We have experimented with using this hierarchy to allow the user to mix together their shared and private states, so that they can choose to have some parameters of their visualization View matching a collaborator while keeping others independent. For example, if one user normally labels their data with a colormap in shades of red and green, and another user is red-green colorblind, the colorblind user could apply their own colormap while otherwise seeing everything as the first user specified. In practice, we have found little demand among our users for this kind of mixed View.

The fine-grained approach to maintaining the nanoManipulators state allowed extensive use of a few base classes and was an excellent design for the original single-user application. When extended to a distributed application, it showed several shortcom-

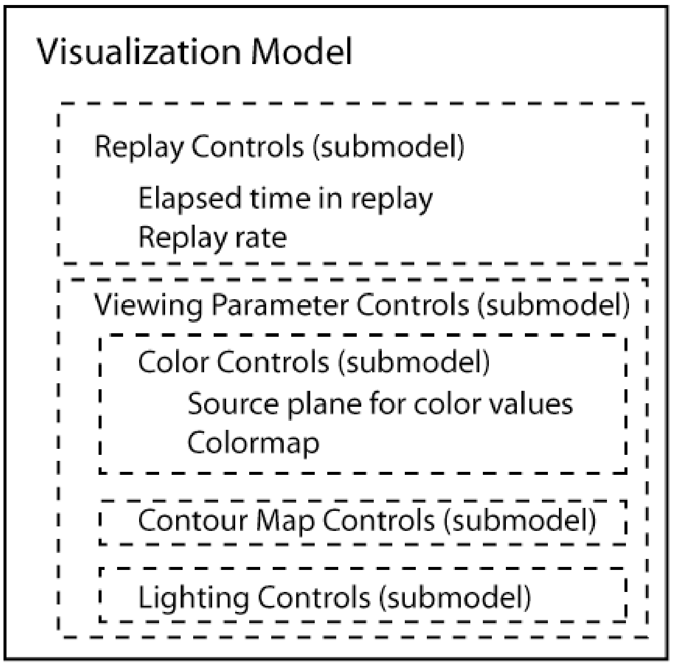
Visualization Model
Replay Controls (submodel)
Elapsed time in replay
Replay rate
Viewing Parameter Controls (submodel)
Color Controls (submodel)
Source plane for color values
Colormap
Contour Map Controls (submodel)
Lighting Controls (submodel)

Figure F.4: Sample nested sub-Models within the visualization Model. The visualization Model is divided into replay control and viewing parameter Models; the latter is further divided into smaller Models which contain parameters relevant to one particular aspect of the view, such as contour maps or lighting.

ings. There are a number of complex objects composed from these primitive value objects; for example, the rendered orientation of the data received from the microscope is a quaternion but is implemented as four independent floating-point numbers. Changes to several of the values comprising a single complex object happened in a coordinated manner within a single process, but when transmitted across the network occurred as distinct changes to the remote copy of the complex object. This caused both unnecessary network traffic and a good deal of difficulty with the design of the callbacks controlling these complex objects. For example, quaternions could transiently take on non-normalized values, which should not be displayed to the user.

A system that would support composition of primitive value objects into complex objects would provide cleaner sharing. This coarser-grained system would not invalidate any of our results or proposed architecture. However, it could violate the separation of concerns (the layering) that placed basic synchronization in the SharedObject and selection in the MultibufferedVariable, since the network synchronization would need to know about the embedding of the SharedObject into the coarse-grained MultibufferedVariable to synchronize each coarse-grained variable atomically.

Programmers have also resisted a coarse-grained system because they perceive it to entail a loss of flexibility. Instead of declaring ad-hoc variables as each is required, programmers using a coarse-grained system must explicitly write code for these clusters of variables to be recognized and treated as a single unit by the networking layer. Sun RPC and Java Remote Method Invocation (RMI) have solutions to this: data structures or function calls can be analyzed by a preprocess that automatically constructs the code for their network transmission. Similarly, we have built a set of Perl scripts that parses a message-parameter-definition file and automatically writes C++ functions to pack and unpack those data structures (convert them from machine-native representation as several variables to network representation as a string of bytes, and vice-versa).

Bibliography

Adachi, Y., T. Kumano, and K. Ogino (1995). Intermediate Representation for Stiff Virtual Objects. In *Virtual Reality Annual International Symposium*, Research Triangle Park, NC, pp. 203 – 210. IEEE.

Alexander, C. (1977). *A pattern language: towns, buildings, construction.* Oxford University Press.

Aliaga, D., J. D. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. E. Hoff, III, T. C. Hudson, W. Sturzlinger, R. Bastos, M. C. Whitton, F. P. Brooks, Jr., and D. Manocha (1999). MMR: An Integrated Massive Model Rendering System Using Geometric and Image-Based Acceleration. In *Symposium on Interactive 3D Graphics*. ACM.

Anderson, R. J. and M. W. Spong (1989). Bilateral Control of Teleoperators with Time Delay. *IEEE Transactions on Automatic Control 34*(5), 494 – 501.

Azuma, R. and G. Bishop (1995). A Frequency-Domain Analysis of Head-Motion Prediction. In *Proceedings of ACM SIGGRAPH*, Los Angeles, CA, pp. 401 – 408. ACM.

Babaoglu, O. and K. Marzullo (1993). Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In S. Mullender (Ed.), *Distributed Systems* (Second ed.)., pp. 55–96. ACM Press.

Benford, S., J. Bowers, L. E. Fahlen, J. Mariani, and T. Rodden (1994). Supporting Cooperative Work in Virtual Environments. *The Computer Journal 37*(8), 653 – 668.

Bertsekas, D. and R. Gallager (1992). *Data Networks* (Second ed.). Englewood Cliffs, NJ: Prentice Hall.

Bhola, S. and M. Ahamad (1999). Workload Modeling for Highly Interactive Applications. Technical Report GIT-CC-99-2, Georgia Institute of Technology.

Bhola, S., G. Banavar, and M. Ahamad (1998). Responsiveness and Consistency Tradeoffs in Interactive Groupware. In *ACM Conference on Computer-Supported Cooperative Work*, pp. 79 – 88.

Birman, K. P. and T. A. Joseph (1987). Exploiting Virtual Synchrony in Distributed Systems. *ACM Operating Systems Review 21*(5), 123 – 138.

Birrell, A. D. (1984). Implementing Remote Procedure Calls. *Transactions on Computer Systems 2*(1), 39 – 59.

Birrell, A. D., G. Nelson, S. Owicki, and E. Wobber (1993). Network Objects. *Operating Systems Review 27*(5), 217 – 230.

Bokinsky, A. A. (2003). *Multivariate Data Visualization with Data-Driven Spots.* Ph. D. thesis, Department of Computer Science, University of North Carolina at Chapel Hill.

Bolot, J.-C. and T. Turletti (1996). Adaptive Error Control for Packet Video in the Internet. In *International Conference on Internet Protocols*, Lausanne.

Bolot, J.-C. and A. Vega-Garcia (1996). Control mechanisms for packet audio in the Internet. In *IEEE Infocom*, San Francisco, CA, pp. 232 – 239.

Braden, R., D. Clark, and S. Shenker (1994). Integrated Services in the Internet Architecture: an Overview. Technical Report RFC 1633, IETF Network Working Group.

Braden, R., L. Zhang, S. Berson, S. Herzog, and S. Jamin (1997, September). Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Technical Report RFC 2205, IETF.

Brooks, Jr., F. P., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick (1990). Project GROPE - Haptic Displays for Scientific Visualization. In *Proceedings of ACM SIGGRAPH*, Volume 24, pp. 177 – 185. ACM.

Brunner, B., K. Landzettel, B.-M. Steinmetz, and G. Hirzinger (1995). Tele-sensor-programming: A task-directed programming approach for sensor-based space robots. In *Advanced Robotics*, Catalonia, Spain.

Bryson, S. and S. Johan (1996). Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments. In *Proceedings of IEEE Visualization*, pp. 255 – 263.

Burdea, G. C. (1996). *Force and Touch Feedback for Virtual Reality.* New York, NY: John Wiley and Sons.

Capps, M. (2000). *Fidelity Optimization in Distributed Virtual Environments.* Ph. D. thesis, Department of Computer Science, Naval Postgraduate School.

Carragher, B. and C. S. Potter (1999). The World Wide Laboratory: Remote and Automated Access to Imaging Instrumentation. In *Impact of Advances in Computing and Communications on Chemical Science and Technology*, pp. 141 – 153. Washington, DC: National Academy Prss.

Cheshire, S. (1996). It's the Latency, Stupid.

Christiansen, K. Jeffay, D. Ott, and F. D. Smith (2001). Tuning RED for Web Traffic. *Transactions on Networking 9*(3), 249 – 264.

Clark, D. and J. Wroclawski (1997, July 1997). An Approach to Service Allocation in the Internet. Internet Draft, MIT Laboratory for Computer Science.

Conner, B. and L. Holden (1997). Providing A Low Latency User Experience In A High Latency Application. In *Symposium on Interactive 3D Graphics*, Providence, RI, pp. 45 – 48, 184.

Corner, M. D., B. D. Noble, and K. M. Wasserman (2001). Fugue: Time Scales of Adaptation in Mobile Video. In W.-c. Feng and M. G. Kienzle (Eds.), *Multimedia Computing and Networking*, Volume 4312, San Jose, CA, pp. 75 – 87. SPIE.

Day, J. D. and H. Zimmermann (1983). The OSI Reference Model. In *IEEE*, Volume 71, pp. 1334 – 1340. IEEE.

Delgrossi, L., C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt (1993). Media Scaling for Audiovisual Communication with the Heidelberg Transport System. In *Multimedia*, pp. 99 – 104. ACM.

Dewan, P. (1997). Course notes for Seminar on Collaborative System.

Dewan, P. and H. Shen (1998). Controlling access in multiuser interfaces. *Transactions on Computer-Human Interaction 5*(1), 34 – 62.

Dourish, P. (1995). The Parting of Ways: Divergence, Data Management and Collaborative Work. In H. Marmolin, Y. Sundblad, and K. Schmidt (Eds.), *ECSCW*, Stockholm, Sweden, pp. 215 – 230. Kluwer Academic Publishers.

Falvo, M., J. Steele, R. M. Taylor II, and R. Superfine (2000). Gearlike rolling motion mediated by commensurate contact: Carbon nanotubes on HOPG. *Physics Review B 62*, 10665 – 10667.

Falvo, M., S. Washburn, R. Superfine, M. Finch, F. P. Brooks, Jr., V. Chi, and R. M. Taylor II (1997). Manipulation of Individual Viruses: Friction and Mechanical Properties. *Biophysical Journal 72*(3), 1396 – 1403.

Falvo, M. R., G. J. Clary, R. M. Taylor II, V. Chi, F. P. Brooks Jr, S. Washburn, and R. Superfine (1997). Bending and buckling of carbon nanotubes under large strain. *Nature 389*, 582 – 584.

Ferrari, D. (1990). Client Requirements for Real-Time Communication Services. *IEEE Communications*, 65 – 72.

Ferrell, W. R. (1966). Delayed force feedback. *IEEE Transactions on Human Factor Electron 8*, 449 – 455.

Finch, M., V. Chi, R. M. Taylor II, M. Falvo, S. Washburn, and R. Superfine (1995). Surface Modification Tools in a Virtual Environment Interface to a Scanning Probe Microscope. In *Symposium on Interactive 3D Graphics*, Monterey, CA, pp. 13 – 18. ACM.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. The Addison-Wesley Object Technology Series. Reading, MA: Addison-Wesley.

Fraser, M., T. Glover, I. Vaghi, S. Benford, C. Greenhalgh, J. Hindmarsh, and C. Heath (2000). Revealing the Realities of Collaborative Virtual Reality. In *Collaborative Virtual Environments*, San Francisco, CA, pp. 29 – 37. ACM.

Funaya, K. and N. Takanasi (1993). Predictive Bilateral Master-Slave Manipulation with Statistical Environment Model. In *International Conference on Robotics and Automation*, Volume 3, Atlanta, pp. 755 – 760. IEEE.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Reading, MA: Addison-Wesley.

Gobbetti, E. and E. Bouvier (1999). Time-critical Multiresolution Scene Rendering. In D. Ebert, M. Gross, and B. Hamann (Eds.), *Proceedings of IEEE Visualization*, San Francisco, CA, pp. 123 – 130. IEEE.

Goktas, F., J. M. Smith, and R. Bajcsy (1997). Telerobotics Over Communication Networks: Control and Networking Issues. In *Decision and Control*. IEEE.

Graham, T. C. N., T. Urnes, and R. Nejabi (1996). Efficient Distributed Implementation of Semi-Replicated Synchronous Groupware. In *UIST*, Seattle, WA. ACM.

Greenhalgh, C., S. Benford, and G. Reynard (1999). A QoS Architecture for Collaborative Virtual Environments. In *ACM Multimedia*, Orlando, FL, pp. 121 – 130. ACM.

Gregory, A. D., S. A. Ehmann, and M. C. Lin (2000). inTouch: Interactive Multiresolution Modeling and 3D Painting with a Haptic Interface. In *IEEE Virtual Reality Conference*.

Guthold, M., M. Falvo, W. G. Matthews, S. Paulson, A. Negishi, S. Washburn, R. Superfine, F. P. Brooks Jr, and R. M. Taylor II (2000). Controlled Manipulation of Molecular Samples with the nanoManipulator. *Transactions on Mechatronics 5*, 189 – 198.

Hannaford, B. and S. Venema (1995). Kinesthetic Displays for Remote and Virtual Environments. In W. Barfield and T. A. Furness, III (Eds.), *Virtual Environments and Advanced Interface Design*, pp. 415 – 436. New York: Oxford University Press.

Hannaford, B., L. Wood, D. A. McAffee, and H. Zak (1991). Performance Evaluation of a Six-Axis Generalized Force-Reflecting Teleoperator. *IEEE Transactions on Systems, Man, and Cybernetics 21*(3), 620 – 633.

Helser, A. (2001, 25 May 2001). personal communication.

Herlihy, M. (1987). Concurrency versus Availability: Atomicity Mechanisms for Replicated Data. *Transactions on Computer Systems 5*(3), 249 – 274.

Herlihy, M. (1990). Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types. *Transactions on Database Systems 15*(1), 96 – 124.

Hesina, G. and D. Schmalstieg (1998). A Network Architecture for Remote Rendering. In *Second International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pp. 88 – 91.

Holloway, R. (1992). Viper: A Quasi-Real-Time Virtual-Environment Application. Technical Report TR92-004, University of North Carolina at Chapel Hill.

Howland, R. and L. Benatar (1997). A Practical Guide to Scanning Probe Microscopy. Technical Report, Park Scientific Instruments.

Hubbard, P. M. (1995). *Collision Detection for Interactive Graphics Applications.* Ph. D. thesis, Computer Science, Brown University.

Hudson, T. C., M. Clark Weigle, K. Jeffay, and R. M. Taylor, II (2001). Experiments in Best-Effort Multimedia Networking for a Distributed Virtual Environment. In W.-c. Feng and M. G. Kienzle (Eds.), *Multimedia Computing and Networking*, Volume 4312, San Jose, pp. 88 – 98. SPIE.

Hudson, T. C., A. Helser, D. H. Sonnenwald, and M. C. Whitton (2003). Managing Collaboration in the Distributed nanoManipulator. In *IEEE Virtual Reality*, Los Angeles, pp. 180 – 190. IEEE.

Hudson, T. C., A. Helser, D. H. Sonnenwald, and M. C. Whitton (2004). Managing Collaboration in the nanoManipulator. *Presence 13*(2).

Hudson, T. C., D. H. Sonnenwald, K. L. Maglaughlin, M. C. Whitton, and R. E. Bergquist (2000, December 2000). Enabling Distributed Collaborative Science. Video Proceedings of ACM Conference on Computer-Supported Cooperative Work.

Huitema, C. (1996). *IPv6: The New Internet Protocol.* Upper Saddle River, NJ: Prentice Hall PTR.

Jeffay, K., T. C. Hudson, and M. Parris (2001). Beyond Audio andVideo: Multimedia Networking Support for a Distributed Virtual Environment. In *Euromicro 2001.*

Johnson, A. E. and J. Leigh (2001). Tele-Immersive Collaboration in the CAVE Research Network. In E. F. Churchill, D. N. Snowdon, and A. J. Munro (Eds.), *Collaborative Virtual Environments*, CSCW, pp. 225 – 243. London: Springer-Verlag.

Karonis, N. T., M. E. Papka, J. Binns, J. Bresnahan, J. A. Insley, D. Jones, and L. J. M (2003). High-resolution remote rendering of large datasets in a collaborative environment. *Future Generation Computer Systems 19*(6), 909 – 917.

Keshav, S. (1997). *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network.* Addison-Wesley Professional Computing Series. Reading, MA: Addison-Wesley.

Kessler, G. D. and L. F. Hodges (1996). A Network Communication Protocol for Distributed Virtual Environment Systems. In S. Stansfield and M. J. Zyda (Eds.), *Virtual Reality Annual International Symposium*, Santa Clara, CA, pp. 214 – 221. IEEE.

Kim, W. S., B. Hannaford, and A. K. Bejczy (1992). Force-Reflection and Shared Compliant Control in Operating Telemanipulators with Time Delay. *IEEE Transactions on Robotics and Automation 8*(2), 176 – 185.

Klosowski, J. and C. Silva (1999). Rendering on a Budget: a Framework for Time-Critical Rendering. In D. Ebert, M. Gross, and B. Hamann (Eds.), *Visualization*, San Francisco, CA, pp. 115 – 122. IEEE.

Krasner, G. E. and S. T. Pope (1988). A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming 1*(3), 26 – 49.

Kurose, J. F. and K. W. Ross (2002). *Computer networking: a top-down approach featuring the Internet* (Second ed.). Boston: Pearson Education.

Lakos, J. (1996). *Large-Scale C++ Software Design*. Addison-Wesley Professional Computing. Reading, MA: Addison Wesley Longman.

Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Commmunications of the ACM 21*(7), 558 – 565.

Lantz, K. A. (1986). An Experiment in Integrated Multimedia Conferencing. In *Computer-Supported Cooperative Work*, pp. 267–275. ACM.

Lawn, C. A. and B. Hannaford (1993). Performace Testing of Passive Communication and Control in Teleoperation with Time Delay. pp. 776 – 781.

Leckenby, J. (2000). *A Practical Guide to Scanning Probe Microscopy* (2 ed.). ThermoMicroscopes.

Leigh, J., A. E. Johnson, T. A. DeFanti, M. Brown, M. Dastagir Ali, S. Bailey, A. Banerjee, P. Banerjee, J. Chen, K. Curry, J. Curtis, F. Dech, F. Dodds, S. Fraser, K. Ganeshan, D. Glen, R. Grossman, R. Heiland, J. Hicks, A. D. Hudson, T. Imai, M. Ali Khan, A. Kapoor, R. V. Kenyon, J. Kelso, R. Kriz, C. Lascara, X. Liu, Y. Lin, T. Mason, A. Millman, N. Kukimoto, K. Park, B. Parod, P. J. Rajlich, M. Rasmussen, M. Rawlings, D. H. Robertson, S. Thongrong, R. J. Stein, K. Swartz, S. Tuecke, H. Wallach, H.-Y. Wong, and G. H. Wheless (1999). A Review of Tele-Immersive Applications in the CAVE Research Network. In *IEEE Virtual Reality*, Houston, TX, pp. 180 – 189. IEEE.

Leigh, J., K. Park, R. V. Kenyon, A. E. Johnson, T. A. DeFanti, and H.-Y. Wong (1998). Preliminary STAR TAP Tele-Immersion Experiments between Chicago and Singapore. In *HPCAsia*.

Lumelsky, V. (1991). On Human Performance in Telerobotics. *IEEE Transactions on Systems, Man, and Cybernetics 21*(5), 971 –.

Macedonia, M. R., D. P. Brutzman, M. J. Zyda, D. R. Pratt, P. T. Barham, J. Falby, and J. Locke (1995). NPSNET: A multi-player 3D virtual environment over the internet. In P. Hanrahan and J. Winget (Eds.), *Symposium on Interactive 3D Graphics*, pp. 93 – 94. ACM.

Macedonia, M. R., M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz (1994). NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. *Presence 3*(4), 265 – 287.

Malpani, R. and L. A. Rowe (1997). Floor Control for Large-Scale MBone Seminars. In *ACM Multimedia*, Seattle, WA, pp. 155 – 163. ACM.

Marescaux, J., J. Leroy, F. Rubino, M. Smith, M. Vix, M. Simone, and D. Mutter (2002). Transcontinental Robot-Assisted Remote Telesurgery: Feasibility and Potential Applications. *Annals of Surgery 235*(4), 487 – 492.

Mark, W. R., L. McMillan, and G. Bishop (1997). Post-Rendering 3D Warping. In *Symposium on Interactive 3D Graphics*, Providence, RI, pp. 7 – 16. ACM.

Mark, W. R., S. Randolph, M. Finch, J. Van Verth, and R. M. Taylor II (1996). Adding Force Fedback to Graphics Systems: Issues and Solutions. In H. Rushmier (Ed.), *Proceedings of ACM SIGGRAPH*, New Orleans, LA, pp. 447 – 452. ACM Siggraph.

McCanne, S., V. Jacobson, and M. Vetterli (1996). Receiver-Driven Layered Multicast. *Computer Communication Review 26*(4).

Meehan, M. (1999). Survey of Multi-user Distributed Virtual Environments. In *course notes: Developing Shared Virtual Environments*. Los Angeles: ACM SIGGRAPH.

Meehan, M., S. Razzaque, M. C. Whitton, and F. P. Brooks Jr (2003). Effect of Latency on Presence in Stressful Virtual Environments. In *Virtual Reality*, Los Angeles, CA, pp. 141 – 148. IEEE.

Meltzer, S., R. Resch, B. E. Koel, M. E. Thompson, A. Madhukar, A. A. G. Requicha, and P. Will (2001). Fabrication of Nanostructures by Hydroxylamine Seeding of Gold Nanoparticle Templates. *Langmuir 17*, 1713 – 1718.

Meyers, S. (1992). *Effective C++*. Addison-Wesley Professional Computing Series. Reading, MA: Addison-Wesley.

Milgram, P., A. Rastogi, and J. J. Grodski (1995). Telerobotic Control Using Augmented Reality. In *Robot and Human Communication*, Tokyo, Japan. IEEE.

Mills, D. (1996). Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6, and OSI. Technical Report RFC 2030, IETF.

Mitchell, J. L., W. B. Pennebaker, C. E. Fogg, and D. J. LeGall (1996). *MPEG Video Compression Standard*. Digital Multimedia Standards. Chapman and Hall.

Mitsuishi, M., T. Hori, and T. Nagao (1995). Predictive, Augmented and Transformed Information Display for Time Delay Compensation in Tele-Handling/Machining. In *IEEE International Conference on Robotics and Automation*, pp. 45 – 52.

Molnar, S., J. Eyles, and J. Poulton (1992). PixelFlow: High-Speed Rendering Using Image Composition. *Computer Graphics 26*(2), 231 – 240.

Munson, J. P. and P. Dewan (1996). A Concurrency Control Framework for Collaborative Systems. In *Computer Supported Cooperative Work*, pp. 278 – 287. ACM.

Naylor, B. F. (1992). Partitioning Tree Image Representation and Generation from 3D Geometric Models. In *Graphics Interface*, pp. 201 – 211.

Nichols, K., V. Jacobson, and L. Zhang (1999, July). A Two-bit Differentiated Services Architecture for the Internet. Technical Report RFC2638, IETF Network Working Group.

Niemeyer and Slotine (1991). Stable Adaptive Teleoperation. *IEEE Journal of Oceanic Engineering 16*(1), 152 – 162.

Niemeyer, G. and J.-J. E. Slotine (1998). Towards force-reflecting teleoperation over the Internet. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Volume 3, Leuven, Belgium, pp. 1909 – 1915. IEEE.

Nise, N. S. (1995). *Control Systems Engineering* (Second ed.). Menlo Park, CA: Addison-Wesley.

Norton, A. and A. Rockwood (2003). Enabling View-Dependent Progressive Volume Visualization on the Grid. *IEEE Computer Graphics and Applications 23*(2), 22 – 31.

Olson, G. M., D. E. Atkins, R. Clauer, T. A. Finholt, F. Jahanian, T. L. Killeen, A. Prakash, and T. Weymouth (1998). The Upper Atmospheric Research Collaboratory. *interactions 5*(3), 48 – 54.

Pasquale, J. C., G. C. Polyzos, E. W. Anderson, and V. P. Kompella (1992). The Multimedia Multicast Channel. In *Network and Operating System Support for Digital Audio and Video*, San Diego, CA, pp. 197 – 208.

Potter, C. S., B. Carragher, L. Carroll, C. Conway, B. Grosser, J. Hanlon, N. Kisseberth, S. Robinson, U. Thakkar, and D. Weber (2000, November). Bugscope: A Practical Approach to Providing Remote Microscopy for Science Education Outreach. Technical Report 00-012, Imaging Technology Group, Beckman Institute, UIUC.

Poulton, J. (1991). Building Microelectronic Systems in a University Environment. In *Advanced Research in VLSI*, Santa Cruz, CA.

Prakash, A. (1999). Group Editors. In M. Beaudouin-Lafon (Ed.), *Computer Supported Co-operative Work*, Trends in Software, pp. 103 – 134. Chichester, West Sussex: John Wiley and Sons.

Rastogi, A., P. Milgram, D. Drascic, and J. J. Grodski (1996). Telerobotic Control with Stereoscopic Augmented Reality. In M. T. Bolas, S. S. Fisher, and J. O. Merritt (Eds.), *SPIE*, San Jose, CA, pp. 115 – 122.

Raynal, M. (1992). About logical clocks for distributed systems. *ACM Operating System Review 26*(1), 41 – 48.

Razdan, A., A. Amresh, N. K. Bade, S. H. Pandya, J. Sun, E. W. Ong, B. L. Ramakrishna, V. B. Pizziconi, and W. S. Glaunsinger (2000). Remote Operation of Scanning Probe Microscopes Over the Internet. *Proceedings of Royal Microscopy Society 35*(4), 297 – 305.

Reed, D. A. (2003, January). Grids, the TeraGrid, and Beyond. *IEEE Computer 36*(1), 62 – 68.

Resch, R., D. Lewis, S. Meltzer, N. Montoya, B. E. Koel, A. Madhukar, A. A. G. Requicha, and P. Will (2000). Manipulation of gold nanoparticles in liquid environments using scanning force microscopy. *Ultramicroscopy 82*, 135 – 139.

Rizzo, L. (1997). Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review 27*(1), 31 – 41.

Robinett, W. (1998). Switching Among the Four Modes of a Teleoperator System: Teleoperation, Simulation, Replay and Robot. In *International Conference on Artificial Reality and Tele-existance*, Tokyo.

Robinson, M., S. Pekkola, J. Korhonen, S. Hujala, T. Toivonen, and M.-J. O. Saarinen (2001). Extending the Limits of Collaborative Virtual Environments. In E. F. Churchill, D. N. Snowdon, and A. J. Munro (Eds.), *Collaborative Virtual Environments*, CSCW, pp. 21 – 42. London: Springer-Verlag.

Ruspini, D. C., K. Kolarov, and O. Khatib (1997). The Haptic Display of Complex Graphical Environments. In T. Whitted (Ed.), *Siggraph*, pp. 345 – 352. ACM.

Salisbury, K., D. Brock, T. Massie, N. Swarup, and C. Zilles (1995). Haptic Rendering: Programming Touch Interaction with Virtual Objects. In *Symposium on Interactive 3D Graphics*, Monterey, CA, pp. 123 – 130. ACM.

Sandin, D., G. Olson, and M. R. Macedonia (1997). Distributed, interactive collaboration - where is it? In *Interactive 3D Graphics*, Providence, RI.

Schmidt, D. C., M. Stal, H. Rohnert, and F. Buschmann (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Volume 2 of *Wiley Series in Software Design Patterns.* Wiley and Sons.

Schneider, F. B. (1993). Replication Management using the State-Machine Approach. In S. Mullender (Ed.), *Distributed Systems* (Second ed.)., pp. 169–197. ACM Press.

Schuckmann, C., J. Schummer, and P. Seitz (1999). Modeling Collaboration using Shared Objects. In *Groupware*, Phoenix, AZ. ACM.

Shaw, C., J. Liang, M. Green, and Y. Sun (1992). The decoupled simulation model for VR systems. In *Human Factors in Computer Systems (CHI)*, Monterey, CA, pp. 321 – 328. ACM.

Shenker, S., C. Partridge, and R. Guerin (1997, September). Specification of Guaranteed Quality of Service. Technical Report RFC 2212, Internet Engineering Task Force.

Sheridan, T. B. (1993). Space Teleoperation Through Time Delay: Review and Prognosis. *IEEE Transactions on Robotics and Automation 9*(5), 592 – 606.

Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Third ed.). Reading, MA: Addison-Wesley.

Singhal, S. and M. J. Zyda (1999). *Networked Virtual Environments: design and implementation.* ACM Press.

Sonnenwald, D. H. (2002). personal communication.

Sonnenwald, D. H., R. E. Bergquist, K. L. Maglaughlin, E. Kupstas Soo, and M. C. Whitton (2001). Designing to Support Collaborative Scientific Research Accross Distances: The nanoManipulator Environment. In E. F. Churchill, D. N. Snowdon, and A. J. Munro (Eds.), *Collaborative Virtual Environments*, CSCW, pp. 202 – 224. London: Springer-Verlag.

Sonnenwald, D. H., K. L. Maglaughlin, and M. C. Whitton (2001). Using Innovation Diffusion Theory to Guide Collaboration Technology Evaluation: Work in Progress. WETICE Conference.

Sonnenwald, D. H., M. C. Whitton, and K. L. Maglaughlin (2002). Evaluating a scientific collaboratory: Results of a controlled experiment. *ACM Transactions on Computer Human Interaction 10*(2), 151–176.

State, A., J. Rosenman, H. Fuchs, T. J. Cullip, and J. Symon (1994). VISTAnet: Radiation therapy treatment planning through rapid dose calculation and interactive 3D volume visualization. In *Visualization in Biomedical Computing*, Rochester, MN, pp. 484 – 492.

Stone, D. L. and K. Jeffay (1993). Queue Monitoring: A Delay Jitter Management Policy. In *Network and Operating System Support for Digital Audio and Video*, pp. 149 – 160.

Stone, J. E., J. Gullingsrud, and K. Schulten (2001). A System for Interactive Molecular Dynamics Simulation. In *Symposium on Interactive 3D Graphics*, Chapel Hill, NC, pp. 191 – 194. ACM.

Talley, T. and K. Jeffay (1994). Two-Dimensional Scaling Techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams. In *ACM Multimedia*, San Francisco, pp. 247 – 254. ACM.

Tanenbaum, A. S. (1989). *Computer Networks* (Second ed.). Englewood Cliffs, NJ: Prentice Hall.

Taylor, II, R. M. (1994). *The Nanomanipulator: A Virtual-Reality Interface to a Scanning Tunneling Microscope.* Ph. D. thesis, Department of Computer Science, University of North Carolina at Chapel Hill.

Taylor, II, R. M., D. Borland, F. P. Brooks Jr, M. Falvo, M. Guthold, T. C. Hudson, K. Jeffay, G. Jones, D. Marshburn, S. J. Papadakis, L. C. Qin, A. Seeger, F. D. Smith, D. H. Sonnenwald, R. Superfine, S. Washburn, C. Weigle, M. C. Whitton, P. Williams, L. Vicci, and W. Robinett (2003). Visualization and Natural Control Systems for Microscopy. In C. J. Hansen and C. Hansen (Eds.), *Visualization Handbook*.

Taylor, II, R. M., J. Chen, S. Okimoto, N. Llopis-Artime, V. Chi, F. P. Brooks, Jr., M. Falvo, S. Paulson, P. Thiansathaporn, D. Glick, S. Washburn, and R. Superfine (1997). Pearls Found on the Way to the Ideal Interface for Scanned Probe Microscopes. In *IEEE Visualization 97*, pp. 467–470. IEEE.

Taylor, II, R. M., T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser (2001). VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *ACM Symposium on Virtual Reality Software and Technology*, Banff Centre, Canada.

Taylor, II, R. M., W. Robinett, V. Chi, F. P. Brooks, Jr., W. V. Wright, R. S. Williams, and E. J. Snyder (1993). The Nanomanipulator: A Virtual-Reality Interface for a Scanning Tunneling Microscope. In *Proceedings of ACM SIGGRAPH*, pp. 127 – 134. ACM.

Taylor, II, R. M. and R. Superfine (1999). Advanced Interfaces to Scanning Probe Microscopes. In H. S. Nalwa (Ed.), *Handbook of Nanostructured Materials and Nanotechnology*, pp. 271 – 308. New York: Academic Press.

Taylor, V. E., J. Chen, T. L. Disz, M. E. Papka, and R. Stevens (1996). Immersive Visualization of Supercomputer Applications: A Survey of Lag Models. *Computational Science and Engineering 3*(4), 46 – 54.

Taylor, V. E., R. Stevens, and T. Canfield (1995). Performance Models of Interactive, Immersive Visualization for Scientific Applications. In *High Performance Computing for Computer Graphics and Visualization.*

Teitelbaum, B., S. Hares, L. Dunn, R. Neilson, R. Narayan, and F. Reichmeyer (1999). Internet2 qbone: Building a Testbed for Differentiated Services. *IEEE Network 13*(5), 8 – 16.

Tennenhouse, D. L., J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden (1997, January). A Survey of Active Network Research. *IEEE Communications Magazine 35*(1), 80 – 86.

Turletti, T. (1993, January). H.261 Software Codec for Videoconferencing Over the Internet. Technical Report 1834, INRIA.

Vaghi, I., C. Greenhalgh, and S. Benford (1999). Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments. In *Virtual Reality Software Tools 99*, London, UK, pp. 42 – 49. ACM.

Vertut, J., A. Micaelli, P. Marchal, and J. Guittet (1981). Short transmission delay on a force reflective bilateral manipulator. In *Rom-An-Sy*, Warsaw, Poland, pp.

267 – 280.

Weigle, C., W. Emigh, G. Liu, R. M. Taylor, II, J. T. Enns, and C. G. Healey (2000). Oriented Sliver Textures: A Technique for Local Value Estimation of Multiple Scalar Fields. In *Graphics Interface*, Montreal, Canada.

Weihl, W. E. (1989). Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types. *ACM Transactions on Programming Languages and Systems 11*(2), 249 – 282.

Weihl, W. E. (1993a). Specifications of Concurrent and Distributed Systems. In S. Mullender (Ed.), *Distributed Systems* (Second ed.)., pp. 27 – 54. ACM Press.

Weihl, W. E. (1993b). Transaction-Processing Techniques. In S. Mullender (Ed.), *Distributed Systems* (Second ed.)., pp. 329–352. ACM Press.

Wickens, C. D. (1986). The effects of control dynamics on performance. In K. Boff (Ed.), *Handbook of Perception and Performance*, Volume 2, pp. 39–1 – 39–60. New York: Wiley.

Wickens, C. D. and P. Baker (1995). Cognitive Issues in Virtual Reality. In W. Barfield and T. A. Furness, III (Eds.), *Virtual Environments and Advanced Interface Design*, pp. 514 – 541. New York: Oxford University Press.

Wloka, M. M. (1995). Lag in Multiprocessor Virtual Reality. *Presence 4*(1), 50 – 63.

Wroclawski, J. (1997a). Specification of the Controlled-Load Network Element Service. Technical Report RFC 2211, Internet Engineering Task Force.

Wroclawski, J. (1997b). The Use of RSVP with IETF Integrated Services. Technical Report RFC 2210, Internet Engineering Task Force.