Iowa State University
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

1-1-2002

Cluster Juggler - PC cluster virtual reality

Eric Charles Olson
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
Olson, Eric Charles, "Cluster Juggler - PC cluster virtual reality" (2002). *Retrospective Theses and Dissertations*. 20188.
https://lib.dr.iastate.edu/rtd/20188

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

Cluster Juggler – PC cluster virtual reality

by

Eric Charles Olson

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Carolina Cruz-Neira, Major Professor
James Bernard
Govindarasu Manimaran
Adrian Sannier

Iowa State University

Ames, Iowa

2002

Copyright © Eric Charles Olson, 2002. All rights reserved.

Graduate College
Iowa State University

This is to certify that the master's thesis of

Eric Charles Olson

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

Table of Contents

List of Figures

Acknowledgements

I would like to thank Dr. Carolina Cruz-Neira for her time and effort involved in helping me with my Master's thesis. I would also like to thank all those who have contributed to this research, especially the developers and contributors involved with VR Juggler. I would also like to thank Dr. James Bernard, Dr. Manimaran Govindarasu, and Dr. Adrian Sannier for being on my committee. Finally, I would like to thank my family for supporting me throughout my education.

Chapter 1: Introduction

Interactive computer graphics are being used as a routine tool in many disciplines, and there is a growing demand to move these interactive tools into immersive environments as technology advances. However, immersive environments (or virtual reality) still require highly specialized equipment and skilled technical people to develop the applications and operate the systems. These requirements prevent the widespread acceptance of virtual reality in research and industrial communities. There is a strong need in these communities to bring virtual reality to a level that allows groups with basic technical computer skills and limited resources to use this technology.

The research in this thesis is motivated by that need. The goal of this work is to take advantage of recent advances in commodity hardware and low-end graphics systems to create a development framework for virtual reality applications. To achieve this goal, we have designed a software system that enables the clustering of PCs or low-end workstations to drive a complex virtual reality environment.

The design presented in this thesis aims to make VR systems more flexible, to simplify the skill level requirements for VR, and to provide a scalable architecture to support a wide range of VR applications. The goal is to design and develop capabilities that allow us to build VR systems that are powered by a set of common personal computers and provide readily available software to manage this distributed environment. This project removes most of the specialized requirements for the computing system required for VR systems to make it easier to obtain and utilize a working VR system.

By using multiple smaller computers instead of a single high-performance computer, this research also addresses the problem of scalability in current VR systems. The goal is to improve scalability by increasing the modularity of VR systems with a concept familiar to cluster computing: adding more “nodes” or low-end computers increases a VR system’s capabilities. For example, a research institution can gradually develop a VR system based on funding phases. The institution can start with one or two PCs driving a single projection screen and add more PCs as funding becomes available to control more display surfaces. With the traditional high-end systems, upgrades may need to replace a single large computer with another specialized computer or may require significant funding to extend the computing and graphics capabilities of the existing one.

Scope of Research

To achieve our goals of using off-the-shelf low-end computing systems and to enable scalability for VR systems, we need to focus on the design of the software to integrate the components in a seamless manner. With distributed VR software on PC environments, VR systems can be tailored in smaller increments without worrying about node limits set by software. This provides VR with a scalability that is lacking in current VR systems.

To create a PC-based VR environment, the research in this thesis has been structured to meet the following objectives:

1. Design a distributed computing extension to current VR software (VR Juggler) to enable a virtual reality system to be powered by a cluster of PCs instead of a single specialized shared memory computer.
2. Allow for VR customization in cluster systems. The nodes in a cluster do not need to be identical nor do they need to perform similar functions. Also, the number of cluster nodes will not be limited by software.
3. Manage the distribution and synchronization of a cluster VR system through our distributed software extension to VR Juggler.
4. Avoid placing extra burden on the application developer when using a PC cluster by hiding the complexities of a cluster system.
5. Maintain application compatibility with our distributed software: existing high-end VR Juggler applications must run in our cluster design with little modifications. The same VR application will run unmodified on a either a shared memory system or cluster system.
6. Implement this thesis' design as an extension to openly available software (VR Juggler). This software extension to VR Juggler will be publicly available to be

used or improved by others. A sample VR system will utilize this software with commodity hardware.

7. Support as many platforms as possible in the implementation by supporting the major operating systems supported by VR Juggler such as Irix, Windows, Linux, etc.
8. With proper PC hardware, support the display of frame-locked and gen-locked active stereo displays.
9. Test a selected group of existing VR applications with the software created by this research.
10. Compare the performance of a sample PC cluster VR system with the performance of a specialized shared memory VR system.

Chapter 2: Background

Virtual Reality

The popular term “virtual reality” (VR) has been used to describe a range of computer-generated technologies. It is associated with everything from a 3D-image on a computer screen to a future technology that can portray a fabricated environment as real as the world itself. Although definitions of virtual reality vary greatly, this thesis is concerned with the research field of virtual reality and its real-world applications. Some of the minimum characteristics for the definition of virtual reality in this thesis include “response to user actions, real-time 3D graphics, and a sense of immersion [Pimentel95].”

When an environment creates a sense of immersion, “the display creates the impression that you’re inside the environment produced by the computer [Stuart96].” An interactive, immersive environment allows the users stop thinking about the virtualness of the environment, and lets them react to it the same way they would react to a real environment. This feeling can be described as a “suspension of disbelief” and is most apparent when VR users physically react to stimuli from the virtual environment as if they were real. Examples of this include users reaching out to (attempt to) touch virtual objects, ducking away from objects, or buckling their knees after falling from a high (virtual) ledge. The objects in these VR environments (or virtual environments) do not have to simulate the real world, but simply engross users so that they interact as if the objects really existed.

A more specific description of virtual reality is provided by Dr. Cruz-Neira: “Virtual reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional, computer-generated environments and the combination of technologies required to build these environments [Cruz93].” In order to generate an experience that meets these criteria, certain technology must be present in any VR system, including our PC cluster VR system.

VR System

In accordance with the above definitions of virtual reality, a VR system should be able to: track the user, be interactive, update in real-time, and drive stereoscopic (3D) displays. These features allow users to feel and act as they would in the real world. If one of the system requirements is missing, the sense of immersiveness may not be achieved.

These criteria exclude some other perceptions of virtual reality such as web-based interactive 3D environments. In the past, virtual reality has required very specialized expensive hardware and software and therefore has continually driven users to find less specialized, more efficient solutions. This is one of the primary motivations for this thesis – finding an alternative to the high-end specialized computer.

Although specific components in virtual reality systems vary, the basic components include a computer (system), tracking and other input devices, display device(s), and software. Descriptions of these different components will follow because knowledge of their operation and purpose is important when incorporating them into the design of our PC cluster.

Tracking and Other Input Devices

One of the basic requirements of virtual reality is that the computer-generated environment reacts to the user and adjusts the perspective based on the position and direction of the user's head. The position and orientation of a person's head, hands, or body parts can be determined with the help of tracking devices. In order to track a person's head, a tracking sensor or emitter is placed (on 3D glasses or a helmet) on the head, and changes are detected and reported back to the computer. The most common methods of tracking use magnetic, visual, or auditory detection methods.

Just to avoid degrading a user's ability to perform tasks, it is generally accepted that computer input must be processed at least ten times per second [Held91]. However, studies have shown that a higher resolution of input is needed to make the virtual environment entirely believable and the users comfortable. Even in non-VR first-person walkthroughs/games, it seems that at 40 Hz (Win98) or 60 Hz (WinNT), the standard mouse sampling rates have been inadequate and have motivated people to use utilities to increase the sample rate (to 120 Hz) [Mouse01].

This thesis needs to take into consideration the performance issues involved in processing and responding to users' input. Maintaining acceptable interaction performance is one of the main requirements for this thesis.

To those unfamiliar with VR or its input devices, it is reasonable to question why one computer in a cluster couldn't handle the entire input load since one PC can handle multiple input devices in applications like flight simulators with two pedals and two joysticks? There are a few reasons: the type of application, the type/quality of input, and the different hardware requirements.

First of all, relatively slow input is acceptable in recreational flight simulators because planes make predictable turns and never quickly turn 180 degrees the way a standing person could. Secondly, delayed updates to the visual system may not be a problem on a simulation using a monitor, since a person's senses are not fully engaged; but when a person is completely surrounded by an environment, a consistently delayed update of the world (position-tracking lag) is suspected of causing simulator sickness [Kolasinksi95]. Lastly, special input devices such as potentiometers can require additional I/O cards to be inserted into a PC. Like any input device, this can consume computing resources, but in addition, a single PC box is hardware limited to the number of extra cards that can be inserted.

Therefore, the combination of processing time constraints for inputs and the number of input devices needed in a virtual environment places a severe performance requirement on a VR system. To meet the performance requirements, it may be necessary to spread the handling of input devices across the available CPUs in the cluster for a better load balancing of the system.

Display Devices

Common VR display devices include standard monitors, head mounted displays (HMDs), and immersive projection displays. Each has its own advantages and disadvantages.

Standard Monitor

The most basic display device is the standard desktop monitor. Although it lacks the features of other display devices, it is the most readily available display, and with the help of head tracking, it functions as a low-end VR system. Its largest disadvantage is a limited field of view, which hinders immersion.

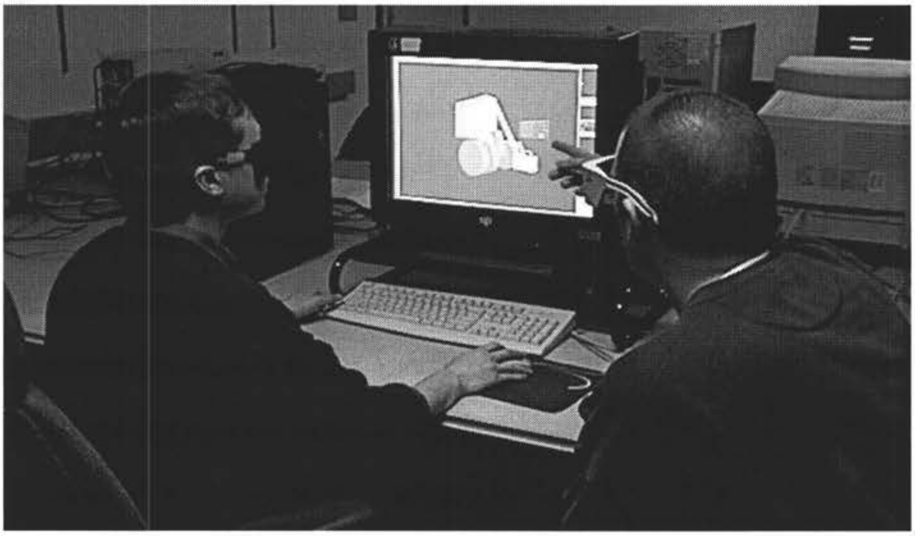


Figure 1 Standard Monitor VR Display

Head Mounted Display

A head mounted display (HMD) is a single-user VR display device. With the help of a tracking device, an HMD has an unrestricted field of view and therefore can completely surround you with a 3D virtual environment. It avoids the distraction of screen edges that can be found on a single planar display.

Wearing the head mounted device is similar to using goggles or a helmet to hold a small monitor in front of each eye. The two small displays are rendered separately by the computer. The images contain perspectives that are slightly shifted apart from each other in order produce the effect of a single stereoscopic image. The tracking device is used to detect the position of the user's head, and adjust the view according the direction the user is facing. Although an HMD has the advantage of unlimited look-around capabilities in a virtual environment, it eliminates the view of the real world. This may be undesirable as it is difficult to interact with physical props or sensors, and also limits interactions with other people.

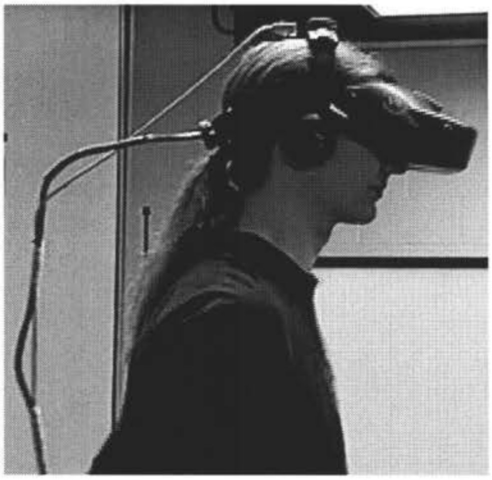


Figure 2 Head Mounted Display (HMD)

Some HMDs have translucent displays that allow users to blend virtual images with real worlds. This is helpful because it can remove the disorientation and isolation feelings of regular HMDs, but seeing other distractions such as a real world office may also detract from immersion. In general, these see-through HMDs are used in augmented reality applications, which are outside the scope of this thesis.

Unfortunately HMDs in the past have been more like helmets than goggles, and have been uncomfortable, cumbersome, and tiring for a user to wear. HMDs are getting smaller all the time, but they are still intended for a single person, making group collaboration difficult. One of the reasons multiple HMDs are not often used for collaboration is because of the high cost to add another specialized computer to drive the displays. The cost would be less of a factor if instead of buying another specialized computer, only a single PC needed to be added to a PC cluster. With the low cost of PCs, it may be reasonable to add, for example, three more HMDs (if they are affordable) instead of none.

The cost of specialized computer hardware may have been one obstacle to multiple HMDs, but the lack of collaborative software would have made it difficult to combine the views into a collaborative program. With a PC cluster system and robust software, the displays should be available to be rendered on any of the nodes, without requiring the application developer to develop additional collaborative software.

Immersive Projection Displays

Immersive projection displays are an alternative to HMDs. Like HMDs, projection displays can produce stereoscopic images but are less isolating than a head mounted device. Because of this, projection displays have been more suited toward group collaboration than HMDs. When multiple people view the same stereo display at once, they can talk and interact with real people in a virtual environment. Of course, with groups of people, current hardware limitations prevent us from displaying tracked images for very many people. This is one of the trade-offs that must be considered. For example, how an application is used will help decide if the application is better suited for five collaborative HMDs or a single immersive projection system with five users.

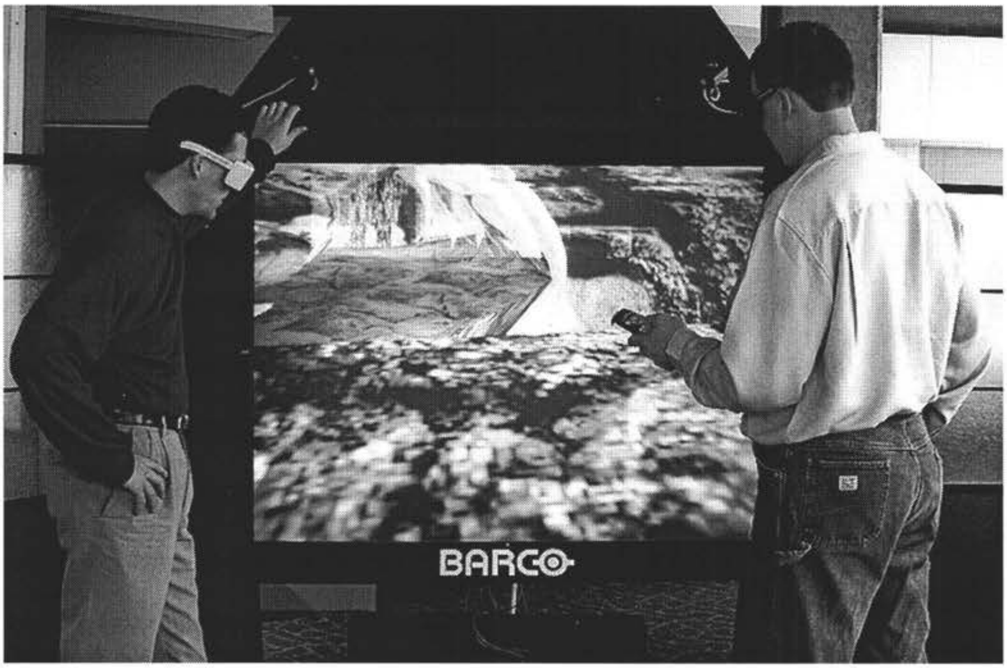


Figure 3 Single Immersive Projection Display

Not only do projection displays allow multiple users to collaborate more easily, but they also allow a single user to operate a VR application more comfortably. Users of projection systems are able to see themselves and any objects or people nearby, reducing disorientation and increasing comfort.

Although a single flat projection display can be used, using multiple projection displays increases the field of view and resolution. Completely surrounding a user with projection displays provides an unlimited field of view and an increased sense of immersiveness. The conveniences of joining multiple stereo displays come at the cost of extra hardware and software synchronization considerations, which are only starting to be addressed on PC clusters.

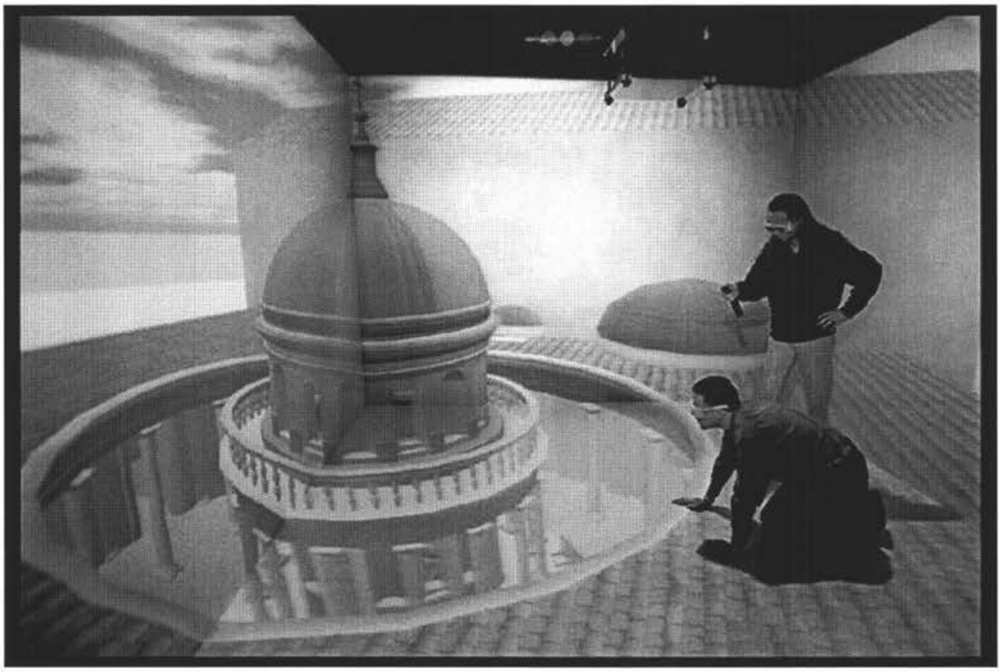


Figure 4 Multiple Immersive Projection Systems (CAVE-like systems)

Special Display Requirements

The type of display used can largely affect the computer system requirements. For example, using multiple stereoscopic displays makes more demands on a computer system than a single stereoscopic monitor or HMD would.

In virtual reality, active stereo shutter glasses are typically used with projection displays. Active stereo means that each consecutive refresh of a display alternates between the left and right eye. So, when the display for the left eye is visible, the right eye's view is blocked (due to the active stereo glasses), and vice versa. The graphics generators need to provide a signal that allows high-speed synchronizations to take place between multiple graphics displays and glasses. In addition, active stereo requires specialized projectors with very high display bandwidth to support the fast alternating high-resolution images.

The first difficulty with active stereo synchronization is the need to synchronize the times when displays refresh their screens. Multiple displays can synchronize this event with the help of a gen-lock signal.

A second requirement when using active stereo is the need to synchronize the displays with users' glasses. A stereo synchronization signal is transmitted to a person's shutter glasses, allowing the glasses to block out the view for each eye at the correct time.

These additional signals are necessary when using active stereo with projection displays. Until recently, PC graphics hardware was not sophisticated enough to provide this high-speed synchronization.

An alternative to active stereo is passive stereo. It requires two projectors (although possibly less expensive) instead of one for each display wall. It also generally requires two image generators, one for each projector. The projectors' images are projected onto the same screen. With the help of an additional lens on each projector, the two projections are polarized differently, using linear or circular polarization methods. This allows inexpensive 3D glasses to correctly allow only one of the projectors' displays to reach each eye.

The passive stereo solution introduces the need for doubling the number of projectors and image generators on each display, but because it eliminates the need for gen-locking and stereo synchronization signals, it has fewer hardware and software synchronization requirements. Passive stereo also reduces the display bandwidth requirements, which enables the use of standard low-end projectors like the ones typically found in conference rooms. This also helps with the overall cost of the VR system, because projectors can be one of the most expensive parts of a projection-based VR system.

Although active stereo is supported by the software design in this thesis, until PC graphics hardware and drivers become more evolved, passive stereo will be a simpler option for PC clusters, therefore we also support it in our design. For example, at the time of this writing, graphics hardware from Intergraph has gen-locking capabilities in Windows, but not for other operating systems such as Linux. The cards are also more expensive than standard commodity hardware.[1]

With the help of a software gen-lock from the Université d'Orléans [SoftGen01], graphics cards from Nvidia can produce active stereo, but this solution is not yet simple to setup in software or hardware and is not a hundred percent stable. It currently requires modifications to the external cabling since gen-locking and sync signals are not provided by the card. Also, the current initial version is only available for Nvidia cards on the Linux operating system and requires a real-time operating system such as RT-Linux. This is another active stereo option to consider while graphics cards supporting gen-locking are still being developed. Until then, passive stereo remains one of the simpler stereo solutions.

Computer System

The computer system has many responsibilities in a VR system. It provides graphics for displays, processes input from devices, and processes and controls applications. Current computer systems available that meet the needs of virtual reality consist of a limited set of shared memory systems or powerful workstations.

The ability to support multiple displays is often necessary when generating the graphics for a VR system, specifically when multiple projection systems are involved. These multi-processor shared memory systems distribute the workload of a VR application and synchronize data between processors and displays seamlessly. By internally handling the synchronization of data between processors and displays, shared-memory systems provide

[1] Video cards from Diamond Multimedia and possibly other companies are also starting to support active stereo on PCs. The costs are still higher than the cost of commodity graphics cards.

the features and power needed by VR without requiring the synchronization to be performed by application developers.

The performance of PCs and low-end workstations has increased enough to consider clustering a set of them as an alternative to high-end systems. Unfortunately, the technology required to integrate a cluster of small computers into a driving engine for a VR system is not as seamless as it is for shared-memory systems. Ethernet hardware is necessary to join the computers and is slower than the connections between multiple processors on a shared-memory system. Also, the communication between computers needs to be written by application developers instead of the machine developers as it is shared-memory systems.

Currently there are only limited tools for cluster-based virtual reality. There is a growing need for a general tool that handles VR rendering, display, and interaction in a clustered system. This research addresses that need by designing a VR Juggler framework for distributed systems.

VR Software Available

Virtual reality is still a developing technology, but while common VR hardware devices have become accepted and supported, there is still no standard for VR software. One of the goals of this thesis is to make the software additions publicly available so they can be used and improved by the VR community. Another critical goal is that the code be cross-platform so it is available to more users. In choosing which VR development software to extend, we need to look for software that is open source, extendable, cross-platform, and supported.

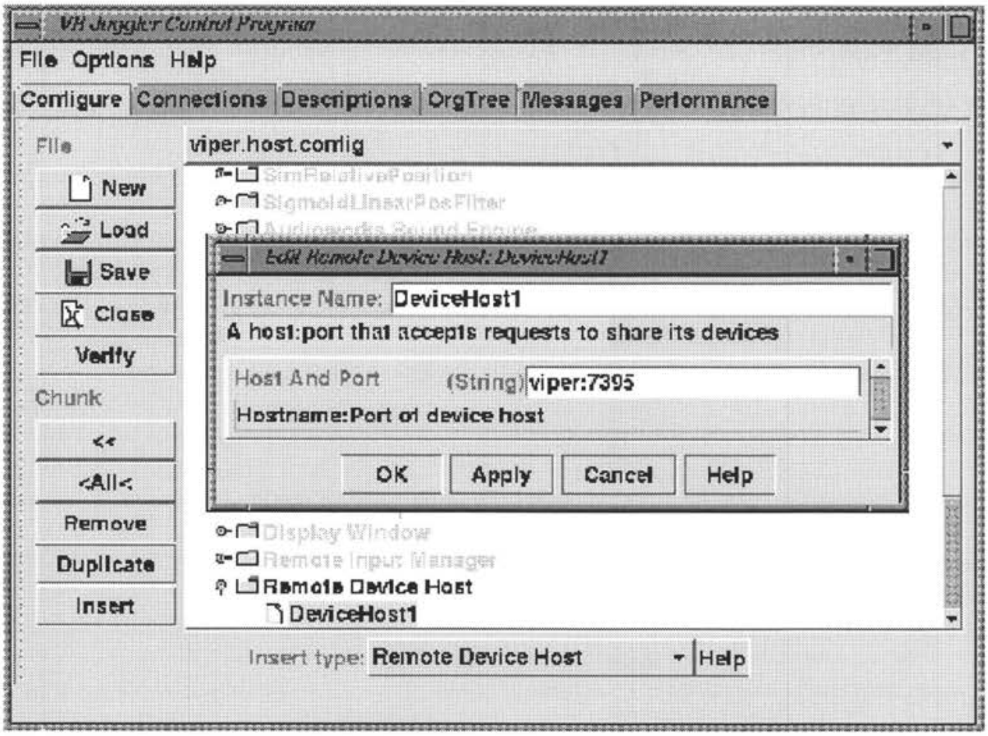
Bierbaum analyzes seven VR development packages in his Master's thesis [Bierbaum00] describing the development platform VR Juggler. World Took Kit and the Cave Library are commercial products. Therefore, using the tools would restrict some people from benefiting from this research. Alice, Performer, Avango, Lightning, and MR

Toolkit are not as cross-platform focused as VR Juggler, and therefore would also limit the applications of this research. FreeVR is another existing development platform, but it has not been developed as much as VR Juggler and does not support as wide a range of platforms. VR Juggler is our development package of choice for this software extension because it is available to everyone, easily extendable, and is supported by its developers. It places the least restrictions on the availability of this thesis work and also is developed enough to provide the features required. For more details about the differences between VR software and the reasons for using VR Juggler, please refer to Bierbaum's thesis [Bierbaum00].

Using VR Juggler

VR Juggler is a virtual reality development environment. It is designed as a virtual platform, allowing applications that are written with VR Juggler to run on any of the platforms it supports. It provides this functionality as transparently as possible so application developers needn't be concerned with the differences between systems. It also applies this concept of abstraction to the use of hardware devices. Hardware such as displays and similar input devices can be changed without the need to modify the application. The ability to configure this functionality is provided by a tool called *VjControl.*

VjControl is the main configuration utility for VR Juggler. It is a graphical interface that allows us to specify a configuration for our virtual reality system. Different configurations are stored in configuration files and allow us to choose a specific configuration for our VR system when we start our application. VjControl will help us configure the additions we make to VR Juggler in this thesis. The graphical interface to VjControl is presented in Figure 5.



File: viper.host.config

Instance Name: DeviceHost1

A host:port that accepts requests to share its devices

Host And Port (String) viper:7395

Hostname:Port of device host

Insert type: Remote Device Host

Help

Figure 5 VjControl Graphical Interface

VR in Research and Industry

Research

Virtual reality is an emerging research area with a considerable amount of research potential in the applications arena. Unfortunately, it is not easy to get started in virtual reality research because of the amount of equipment and technical expertise it requires. The goal of this thesis is to change this by providing the ability to use commodity VR systems and hiding the extra technical complications of a cluster system.

It is also not always easy to advance the field of VR because sometimes researchers are required to reproduce other research before advancing it. Closed research yields an abundance of custom solutions and forces people into reinventing the wheel. These custom solutions and a lack of established standards have slowed new VR research developments.

Fortunately, research developments are becoming more open and reusable. Researchers are beginning to gravitate towards established VR software instead of reimplementing low level VR technology. This allows them to focus on new VR technology.

Industry

Large industries and corporations are the companies most likely to lead in the application of virtual reality to their business. These large companies can afford to make an investment into VR in hopes of large returns. The current state of VR is generally not advanced enough for most corporate users to accept. Industry users want turnkey solutions that don't yet exist in such a young field. The possible benefits of VR are becoming more visible to industry users, but with the current state of virtual reality, the amount of development required is not always apparent or reasonable.

Although limited tools are appearing that allow quick design of VR applications, the ability to design VR applications still often requires backgrounds in virtual reality and computer programming. This is experience that typical IT or R&D employees do not often have and makes it difficult for many companies to be able to apply VR in their working processes without adding new personnel to their groups.

Since companies need to justify their expenditures, they also can't easily spend tremendous resources improving the state of VR. This is where open research such as this thesis helps the advancement and availability of the technology. One company cannot design all the VR tools that it needs, but supporting open VR research is a way to improve the state of VR and allow those tools to be created. Of course companies will promote and ask for features or enhancements based on their own interests, but this is understandable considering they want to stay in business.

An example of the usefulness of open software is the development of Net Juggler [NetJglr01] from the Université d'Orléans. The researchers there did not waste time rewriting an entire VR software library, but they were able to focus on new research by adding capabilities to existing VR Juggler software and then releasing their research results. Both institutions and anyone else who is interested benefit from the software results.

By supporting and participating in VR research, companies can see their contributions improve their business with the help of VR. In VR's current development state, industry users of VR are starting to apply virtual reality to their problems, but so far it is forcing them to work hard for their results.

Although VR is being used more often in both research and industry, people still need a considerable amount of technical expertise and other resources in order to do so. The development of established standards and open software is making it easier for everyone to advance the state of VR. Likewise, this project attempts to improve the state of VR by making current VR systems more flexible and reducing the entry requirements for new VR researchers.

Chapter 3: Basis for Design

Traditional VR: Shared Memory Computer

Traditional virtual reality systems are powered by high-end specialized computers which utilize shared memory architecture. This means a single computer can have multiple processors, all with direct access to a common memory pool. The management of memory access, such as locking while writing, is handled by the operating system. Having more than one processing component allows a computer to efficiently process different parts of an application simultaneously or efficiently process many applications at once. This functionality requires a considerable amount of extra hardware design when compared to the design of a single processor computer.

Although the design of shared memory systems is complicated, for developers and users it simplifies the development and efficient execution of complex multi-processed applications. This is most obvious when running multiple applications. Even if only running operating system tasks, computers are always running multiple applications at once. The performance of multiple applications on shared memory systems is naturally faster than on a computer with a single processor. In a shared memory system, multiple processors allow applications or processes to be separated, reducing or eliminating competition for CPU time. Because of this design, the execution of multiple applications should be faster on a shared memory system than on a single processor computer. Next, we'll examine if the performance of single applications is also improved by using shared memory.

The existence of multiple processors should not exceptionally improve the execution of a completely single-threaded application, except for removing competition for resources with operating system tasks. However, if a program is designed to use multiple threads, its execution can be sped up considerably. Care needs to be taken to lock data when it could be used by multiple threads at once, but with otherwise few inconveniences, shared memory systems can considerably improve performance of applications by using multiple threads.

It should be obvious that a multi-processor shared memory computer performs better than a single processor computer, but can a cluster of single processor computers perform as well as a shared memory computer? This is debatable, but there are advantages and disadvantages to each type of system. This thesis deals with providing an alternative to shared memory systems with a cluster of personal computers for VR applications, so it is important to know the differences between the two types of systems.

Shared Memory Computer vs. PC Cluster

The main difference between the two types of systems is that a cluster's hardware and software were not originally designed for closely synchronized multiprocessing of applications. In a shared memory system, data is automatically coordinated in the operating system and quickly updated by being passed across a memory bus. This process of sharing data is not as inherent in a PC's design as it is in a shared memory system's design. Because of the original design goals, the networking hardware for PCs is not designed specially for cluster computing. Data must be passed across a network where network latency is naturally slower than it is on hardware inside a computer. Not only is cluster networking hardware naturally slower, but extra coordinating software must be added on top of the operating system to coordinate the sharing of data. Usually, developers of cluster applications need to be aware of the network configuration and handle a significant part of the data sharing and synchronization.

The software and hardware additions to a PC can potentially degrade multi-processing performance, but they also add considerable inconvenience to the user. The extra data coordination software is usually not recoded for each individual application, but it still complicates application development by adding restrictions and the need for extra programming knowledge. Even after the initial code for a program is finished, extra inconveniences for the programmer involve configuring the cluster and tuning the application to work well with the communication software.

PC clusters seem to have their share of disadvantages, but they do have substantial benefits as well. While limitations to configuring a shared memory computer are specified by the designer, PC clusters need not have rigid constraints unless they are imposed in software. Shared memory systems often have limits on the number of configurations, maximum number of nodes, or a maximum number of certain components (i.e. graphics nodes). With well designed software, PC clusters can have unlimited nodes, custom configurations, and even customized individual nodes to serve special purposes. Of course identical cluster nodes running a single application in parallel would cease to yield performance gains after a certain number of nodes, but the ability to customize a PC cluster would allow its users to make each node useful. The use of PC clusters in this thesis will show how useful it is to be able to customize individual cluster nodes.

The advantages of a shared memory system over a PC cluster are its performance due to dedicated hardware and software, and its ability to allow a programmer to remain unaware of process sharing between processors. However, these features are costly to develop, and this leads to one obvious motivation behind PC cluster computing: the idea that with fewer resources, PC clusters can generate a computer comparable to a shared memory system. But there are additional advantages to PC clusters such as the removal of design limitations, the ability to customize individual nodes, and the ability use smaller increments to increase or decrease the size of a system.

Traditional Cluster Computing

The idea of clustering personal computers first rose from the need to create extremely powerful scalable computers. This need was created by computationally intense applications that took a long time to run or did not run at all on smaller computers. These applications often involve simulations from various areas such as physics, chemistry, or the military. These and other applications are considered well suited for cluster use because of their need for raw computing power.

Software is available that allows the execution of applications to be distributed between nodes. MPI [MPI] and PVM [PVM] are the most commonly used cluster communication software. In order to utilize the power available in these clusters, application programmers need to customize their programs for these protocols and also learn how to write useful parallel code. Also, these protocols have a fairly static method of execution. The application code is split up at the start of the program, loaded in the available nodes, and in most cases not modified during execution. The cluster and application are also very self-contained so there is little (or no) communication in or out of the cluster from the computers. Cluster computing methods and software exist and are available, but traditional software and methods are not suited to drive a virtual reality cluster system without additions and modifications.

Virtual Reality on a PC Cluster

Virtual reality applications have many different needs than traditional high-performance cluster applications. Virtual reality by nature is highly interactive, and therefore has different hardware and software requirements than conventional computing applications. For example, a virtual reality system needs to be able to receive input, process application data, and produce output (often for multiple displays) at least ten times per second (preferably 30) to keep from degrading human performance [Held91]. Current cluster software is not optimized for communicating with computers or devices outside the cluster and is instead only designed for communicating quickly within the cluster. This external communication needs to be taken into consideration in the design of virtual reality computer clusters.

Another common characteristic of traditional clustering is the practice of making every node identical. This method has worked well because there is generally one computationally heavy task applied to very large dataset that needs to be split up evenly to balance the computational load. Virtual reality is different, since it requires many different tasks to be performed, and in most applications, the data is relatively small compared with the high-performance computing datasets. With tasks such as receiving inputs, processing the application, managing output sounds, and producing output displays, there are many tasks whose performance will be lowered if their data needs to be synchronized between many cluster nodes. Also, a single computer may not be able to handle even part of a few tasks at once. For these reasons a VR cluster cannot be restricted to having identical nodes; some nodes will have to be individually specified to receive input, generate displays, or perform other tasks.

Not only are traditional cluster nodes usually identical, but they are also usually not modifiable once a program starts. This is a problem for virtual reality systems as there are times when changes in an application may cause changes in the demand on a system. Having a dynamic cluster system where nodes can be added, modified, or removed will allow these demands to be met.

One of the most important considerations is to avoid presenting additional difficulties to the application developer. There are enough things to consider when developing a VR application; forcing programmers to customize applications for a cluster is undesirable. The design of traditional shared memory VR systems is ideal for programmers: they don't have to worry about multiple processors, but they can take advantage of them if necessary. This idea needs to be maintained to more easily allow virtual reality to be useful to people with different technical backgrounds. Avoiding additional cluster programming complexities allows as many people as possible to benefit from VR. One of the goals of this thesis is to allow VR applications that run on a shared memory VR system to run on a PC cluster VR system without modifications.

Existing Cluster VR

There has already been a need to use multiple computers to drive a VR system, but past developments have not included our design goals mentioned above. Even the first virtual reality software developed, CaveLib [Cruz95], was extended to support multiple computers. The resulting usability of the system was extremely different from a normal single-computer VR system. The application programmers were required to handle the inter-computer communication. Message passing and synchronization function calls needed to be inserted throughout the code by the programmers. The initial network software was also not intended for PC cluster use as it only ran on SGI (shared-memory) systems.

Since then there have been many other developments, but the most recent and advanced cluster software is called "Net Juggler" from Laboratoire d'Informatique d'Orléans in the Université d'Orléans [NetJglr01]. It is a layer written on top of VR Juggler software that manages communication between cluster PCs in a VR system. Net Juggler uses MPI for its distributed communication. MPI is a protocol developed for traditional high-performance computing applications. Its design expects a cluster to run in identical nodes as in IBM's SP architecture or a Beowulf cluster. This makes it more difficult to make dynamic modifications to the cluster structure or a running application's configuration. Also, using MPI introduces another tool that application developers may need to learn in order to develop a VR application.

One of the motivations for this thesis is to simplify the use of low-end computing systems for VR applications, while providing flexibility and dynamic utilization of the available resources. This requires that we investigate a design beyond high-performance clustering. The following chapters describe our design approach and a discussion of our results.

Chapter 4: Design of Cluster Juggler

Overview

The virtual reality cluster design in this thesis is intended to make PC clusters a viable alternative to virtual reality on shared memory systems. As mentioned previously, it cannot simply utilize the design of traditional clusters because virtual reality requires special functionality not present in conventional cluster applications. It also should not put the burden on the VR application developer to perform communication between cluster nodes. To prevent burdening the developer, the design retains some of the features that shared memory systems provide for virtual reality by hiding the complexities of a cluster. It is beyond the scope of this thesis to design and implement a fully distributed shared memory system for VR. Our goal is to design a distributed shared memory system for VR application I/O (input/output) data. As a result of distributed I/O, application development and execution can transparently move between shared memory VR systems and PC cluster VR systems. This means the same VR applications will run on both systems, with no or very minimal changes.

Not only is this design intended to make a useful virtual reality cluster, it intends to make this technology easily available to anyone who desires to use it. To allow this, the key components to this cluster design are commodity hardware and open software. This thesis makes extensions to the already open source virtual reality project VR Juggler. The commodity computer hardware needed to power the cluster is personal computers with Ethernet cards, but there is still no commodity replacement for some specialized components such as tracking devices and quality projection systems (although tradeoffs can be made with technology such as passive stereo). The specific nodes and components in an individual cluster will vary and can be customized to the needs of each cluster.

Cluster Functions

The design for this thesis presents four types of cluster functionality. The purpose of distinguishing between these functions is to allow them to be separated so that their tasks can be performed on different cluster nodes if desired. Any node can perform one, more than one, or even all of these functions. The four types include:

- Input Functions: receive and interpret data from input devices
- Application Functions: run application specific code
- Rendering Functions: produce data for displays
- Specialized Functions: cover all VR support functions

The following sections will describe the how each type works in the system.

Cluster Input

Virtual reality systems have input devices to allow them to be interactive. The input functions in the PC cluster involve handling the data from these devices. A node that handles input does not need to run the code from the application (although it can), but it processes the input and then sends the resulting data to the nodes processing the application if they need it. It is not necessary to dedicate an entire node to processing input, but this can help if the burden of running other tasks simultaneously is too great and there are extra nodes available. Also, if a large number of input devices are to be used, our design allows the devices to be spread across the cluster nodes. This prevents problems such as multiple input devices creating a large workload on a single computer, or all input hardware not being able to attach to a single PC.

In order to distribute input from one computer to other nodes in the cluster, this thesis makes additions VR Juggler software [Bierbaum00]. VR Juggler already has an input manager that handles local input data on a single computer. As seen in Figure 6, the input manager is part of VR Juggler's specialized microkernel design that uses managers to handle different tasks.

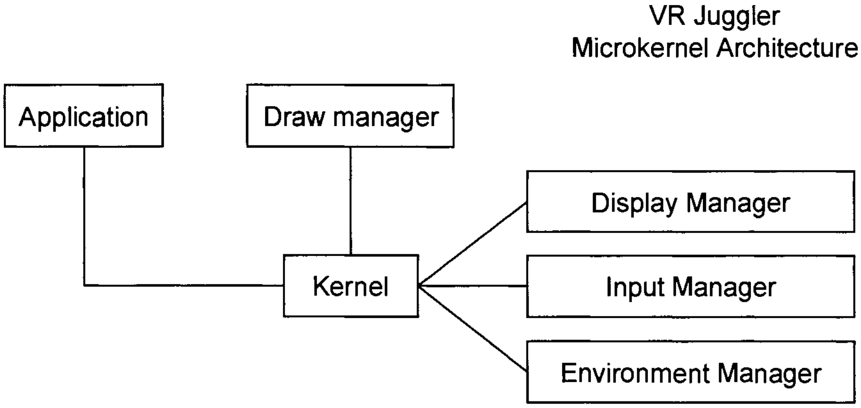


Figure 6 VR Juggler Microkernel Architecture

This thesis extends VR Juggler by adding a remote input manager to the existing input manager. This addition handles the communication of input data between cluster nodes. In communicating the data, the remote input manager also synchronizes the nodes each frame. If there is more than one node executing application code and these nodes need to maintain the same application state, the remote input manager performs the task of synchronization.

There are other benefits to the remote input manager besides synchronization. Depending on the type of input device and the number of devices, the process of receiving input can be burdensome. The remote input manager can alleviate some of the burden on nodes by processing input data before distributing it. This frees up processing time on the other nodes and can also reduce network traffic. This is illustrated in Figure 7.

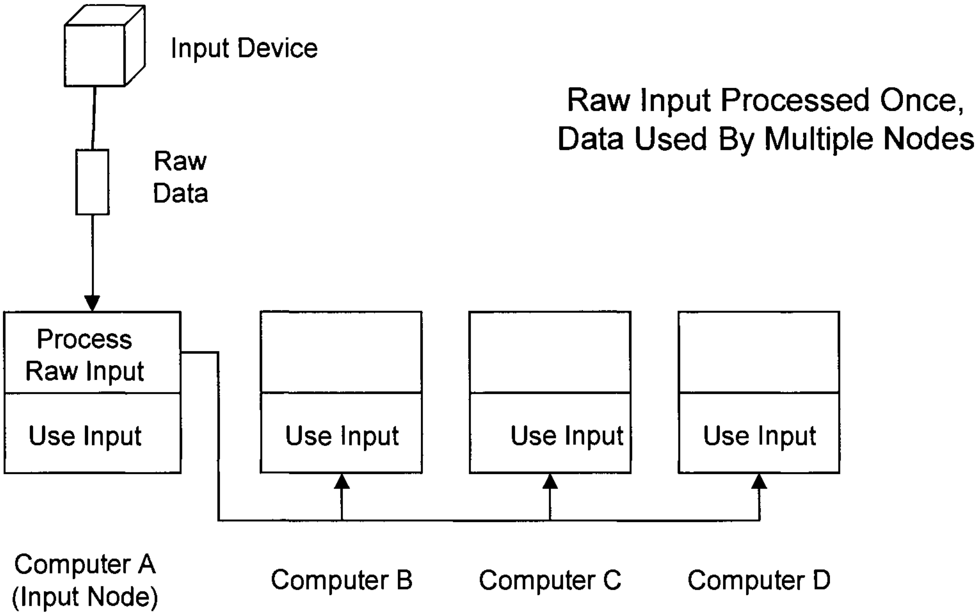


Figure 7 Processing and Distribution of Input

By distributing the input to cluster nodes across the network, we can remove the typical cluster computing constraint of having identical computers at each node. To allow different platforms in our nodes, our network communications design needs to account for cross-platform issues. For example, in Figure 7 *Computer A* can be a Linux based machine; *Computer B* can be a Windows based machine, and *Computer C* can be an SGI computer.

Not only does the remote input manager have useful functionality, but it also intends to avoid forcing the application programmer to worry about the location of devices. Once a cluster is set up, the programmer and application can act as if every input device is connected to every cluster node. The remote input manager avoids putting extra burden on application developers and hides the complications of the cluster from them. Figure 8 illustrates this concept.

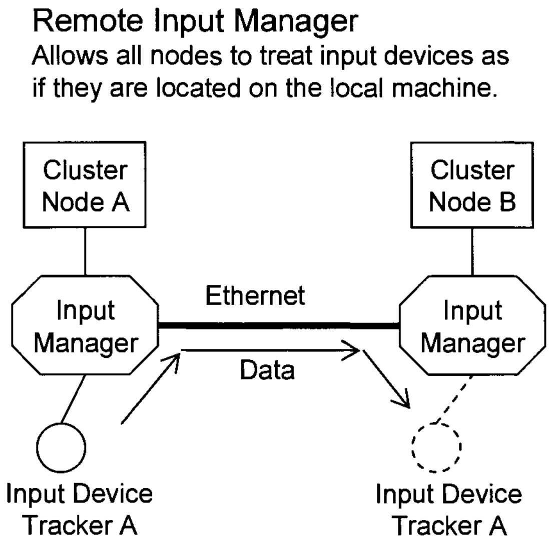


Figure 8 Transparency of Remote Device Access

In the relatively new and evolving field of virtual reality, the remote input manager can help virtual reality keep up with new technology even if the software is not completely available. Although VR Juggler supports many platforms, new devices require their own drivers for each platform. This can restrict the use of new devices on VR systems because writing device drivers is time consuming. With the help of the remote input manager, devices whose drivers have been integrated into VR Juggler on one platform can be utilized on other platforms as well. By connecting to another computer through the remote input manager, other platforms can retrieve device data across the network until drivers for the specific platforms are finally integrated.

Besides avoiding cross-platform issues, there are other reasons for not having the VR application talking directly to the input device. For example, a simple device host can be created using VR Juggler and the remote input manager. By starting up VR Juggler and loading the device, the host is ready to listen for connections from VR Juggler applications. It is just as simple to setup applications to connect to the host. Normal VR Juggler configurations are used, with the addition of filling in the location of an input device with its host and listening port.

Having a host listening for connections to an input device would be useful for a variety of reasons. For example, it has been accepted in the past that only one application has access to an (often expensive) input device at a time. There is no reason for this except for software restrictions. In the past, if two people were modifying and testing applications, they needed to wait for each other to finish before using the input device. It would be convenient if two or more people could test their programs simultaneously. It should be noted that the frame rate of the two applications would be locked due to the normal need for display synchronization, although modifications could be made to our extension to decouple them. This ability to share hardware between devices could be especially useful if the input device is large enough for two people to use it at once. For example, a tracking system with four or more sensors could be split up and used with two applications at the same time. It is also convenient (especially with older tracking systems) if the input node in a cluster system does not need to be reinitialized each time an application is restarted.

New developments in virtual reality can also benefit from this technology. Multiple stereo views in a CAVE-like system are becoming possible [Blom01]. In the past, if the two views were running as separate applications, they weren't both able to connect to the input hardware directly. With a VR Juggler device host, two separate applications can connect to the same input device simply by filling in the location of the device to point to the listening host and port. If running on a PC cluster, nodes can just as easily be used to run two or more separate VR applications instead of one.

Since devices on different computers in a local area network can be used together, couldn't devices on a remote network be used just as easily? If so, a possible extension to VR Juggler's distributed input capabilities would be remotely collaborative applications. Although it is as easy to connect to remote devices from another location's networks as it is to connect to devices on a local network, large network latency would likely reduce system update rate to an undesirable level. The design in this thesis is optimized to provide the best possible performance on a local cluster, not on wide area networks. The high latency caused by passing through additional routers and across large distances would prevent rapid synchronizations.

Although this design may not help with remotely collaborative systems, it would be reasonable for locally collaborative systems. There is no reason why two CAVE-like systems or HMDs couldn't monitor each other's input devices and run a synchronized application as a single collaborative system. Of course the more data and computers that are added, the longer synchronization takes. Tests would need to be done to determine if the update rate on such systems would be adequate.

Cluster Displays

Synchronizing multiple displays is one of the biggest challenges in creating immersive VR on PCs. This is because a single PC is not enough to generate the graphics for multiple displays in VR systems such as a CAVE. Now that graphics hardware is becoming available to synchronize multiple active stereo PCs or show passive stereo, the design in this thesis must provide a way for a running application to be distributed across multiple PCs and graphic hardware.

The first method involves using the additions to VR Juggler from this thesis to run a copy of an application on each PC, while always providing the same input to each PC. The copies of the application stay synchronized with the help of the remote input manager.

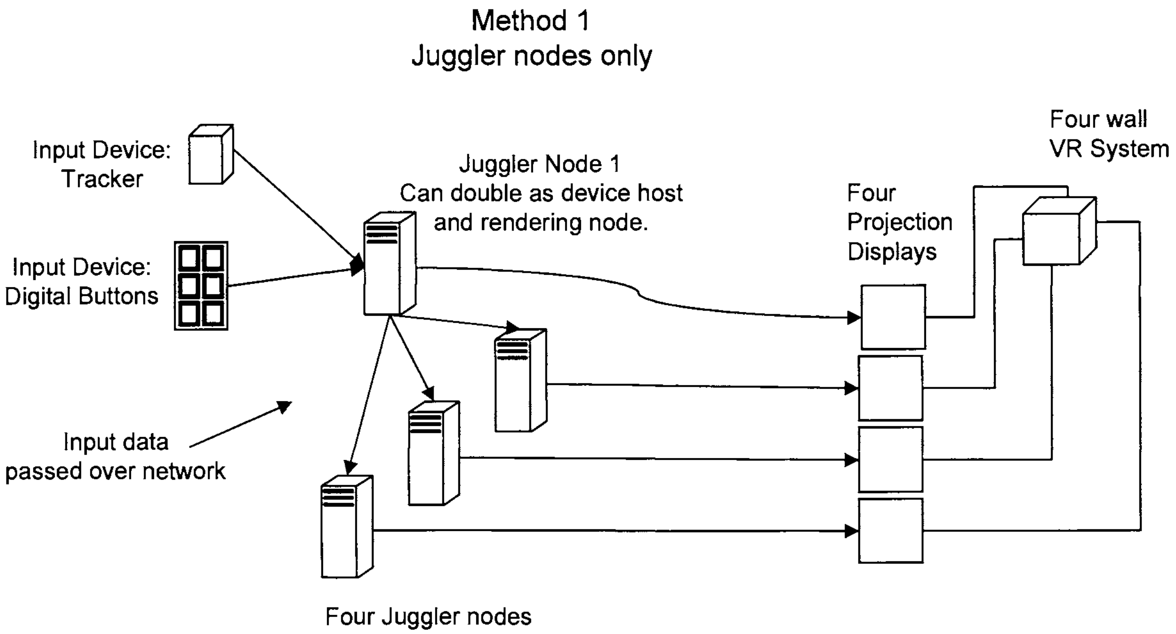


Figure 9 VR Juggler Cluster

The second method involves using the tile display library WireGL from Stanford to distribute the graphics [Wiregl01]. With WireGL, instead of application code being executed on more than one computer, OpenGL graphics commands are sent across the network to each PC. With this method there is effectively one VR Juggler application running. The first method uses only VR Juggler to distribute application data, while the second method uses VR Juggler in conjunction with WireGL to distribute graphics primitives. The second method is shown in Figure 10.

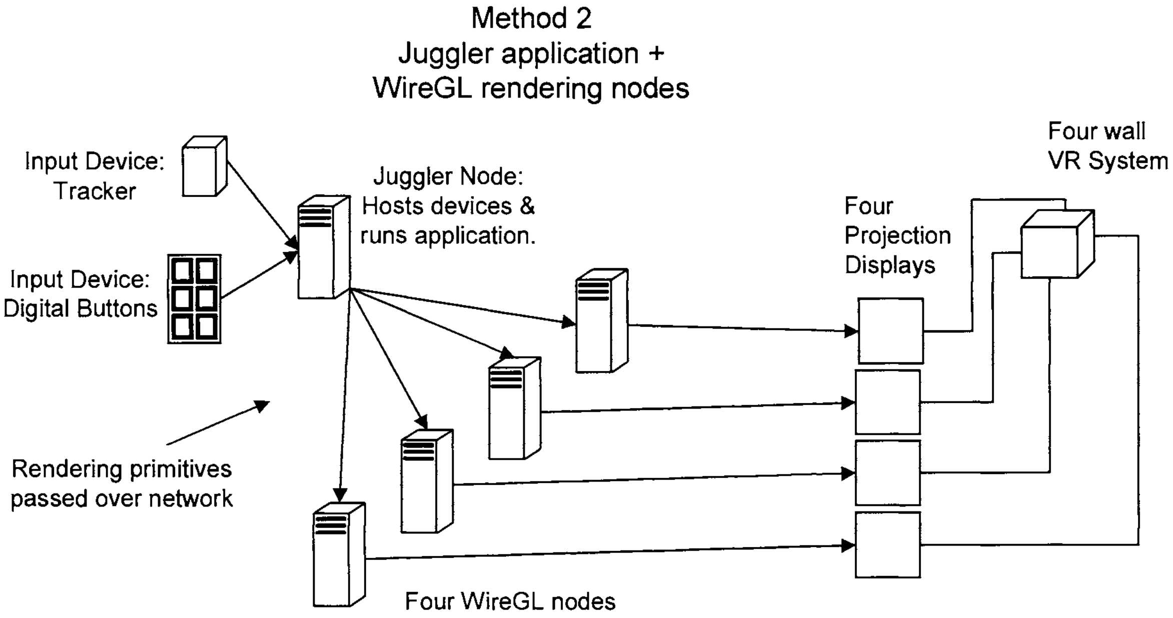


Figure 10 VR Juggler Application, Graphics Distribution with WireGL

Which method performs better may depend on what type of application is running. If the program uses many graphics function calls, the first method using only VR Juggler should be faster. This is because WireGL will be passing large numbers of graphics primitives over the network. However, if the number of graphics calls is small, using WireGL in addition to VR Juggler could speed up VR applications on the cluster. For large systems, combinations of the two designs may result in the best performance. An example of such a hybrid system is shown in Figure 10.

It should be noted that in the past, limitations of WireGL have not allowed the correct rendering of more than simple VR applications. Such limitations are outside the scope of this thesis, but may be addressed in future revisions or releases from WireGL's authors [WireGL01].

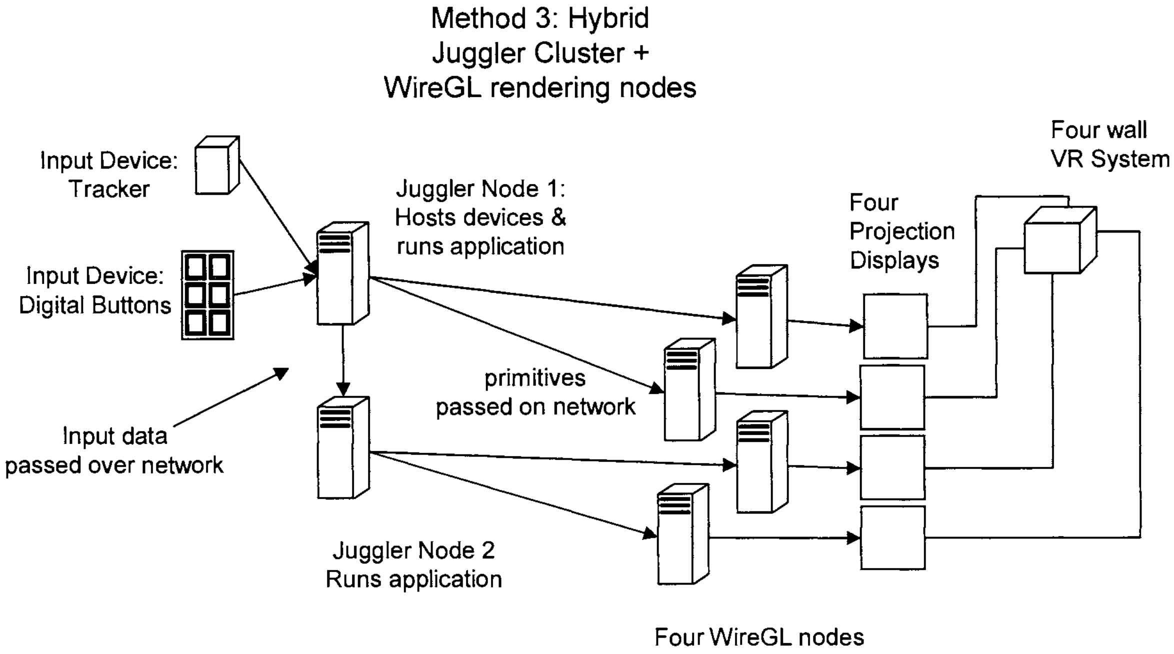

Method 3: Hybrid
Juggler Cluster +
WireGL rendering nodes
Input Device: Tracker
Input Device: Digital Buttons
Input data passed over network
Juggler Node 1: Hosts devices & runs application
primitives passed on network
Juggler Node 2 Runs application
Four WireGL nodes
Four Projection Displays
Four wall VR System

Figure 11 VR Juggler Cluster, Graphics Distribution with both VR Juggler and WireGL

Cluster Application Processing

The VR Juggler-only (first) method of graphics distribution requires synchronized application processing across nodes. Each node running application code runs an entire copy of the application and is configured to receive the same input data from specific cluster nodes. Only one node is required to run the application code, but if multiple immersive displays are desired (and WireGL is not used), then multiple nodes executing application code are necessary. Of course all application nodes do not need to be synchronized, but for running standard applications on multiple displays, it is important.

In order for application synchronization to work, the execution of an application needs to start on all application nodes at the same time. To do this, a barrier can be set up that pauses the start of an application until all application nodes are connected and ready to start. This is not done automatically yet, so the delay in the start of an application's execution must be performed by the application programmer at this time.[2] Once an application has started, the input to each application is kept identical at all times in order for these nodes to maintain the same state. This input data is updated at a point of synchronization, when the application code is not being processed.

Cluster Computing Power

Although massive parallel processing of a traditional PC cluster is not the focus of this design, it can still be used in conjunction with it. An immediate solution for programmers with MPI experience would be to use "Net Juggler" [NetJglr01] since it uses the common parallel programming tool MPI. Another option would be to connect the input and output of a traditional cluster into VR Juggler input and outputs. Although it would require some development to send output to the traditional cluster, receiving input from the cluster would simply involve treating the other cluster as an input device. This technique would be useful if only parts of large applications need to be displayed and updated every frame. The rest of the application could be continually processed in the background by the high-performance computing cluster. Examples where this would be useful include computationally intensive applications that are difficult to process in real-time including molecular simulations, computational fluid dynamics, ecosystem simulations, warfare simulations, etc.

Although MPI and high-performance computing clusters are useful, most virtual reality application developers are not experienced with MPI or other parallel programming tools. Therefore, we focus on developing a clustering method that does not require additional tools or knowledge from those typically found on VR and interactive graphics programmers. Our goal in this issue is to allow programmers to develop applications as if they were using a simple single computer and a basic programming environment with C++ and OpenGL. More

[2] To allow for manual initial synchronization if necessary, a programmer can include the ability to start or reset his/her application and activate this with a simple interaction, such as a key or button press.

sophisticated tools and techniques are supported by our design and the VR Juggler structure if needed by particular users. Examples of such tools include OpenGL Performer or The Visualization Toolkit (VTK).

Other Node Functions

So far cluster input, displays, and application processing have been discussed. The rest of the functions can be grouped together as VR support. This involves things such as audio, performance or application monitors, or any other function that isn't covered in the other main functions.

The most noticeable function in this group is audio. Although not currently implemented, the design for distributed audio would simply label the sound engine with a host location. As the audio engine develops, it may be desirable to specify multiple sound engines on different cluster nodes, allowing 3D sound from different points to be generated by different computers. Even with current technology, this should not be necessary for normal applications since a single computer and soundcard is able to generate high quality three-dimensional sounds.

The primary design goal for this thesis is to provide VR capabilities on a cluster system. In reaching this goal, we have created by-products of our design that we can also benefit from. One of these benefits includes the ability to use heterogeneous computers in our VR cluster. This could especially benefit groups with limited resources by allowing them to extend the use of their existing computing resources to power a VR system.

For example, when a group of people in a company attend a meeting, they will likely leave their computers unoccupied. If they would like to use a VR system in their meeting, their idle computers could be used to power the VR system. This eliminates the need for many dedicated VR systems, reuses idle computing hardware, and makes more VR systems available more often. The systems could be standard Windows, Linux, Unix, or Macintosh

PCs. A graphics card with minimum 3D capabilities would need to be present in the machines, but this is not a costly requirement.

Design Summary

From the above research on the design of a cluster system, the design in this thesis will focus on elements that make a PC cluster useable as a VR system and easy to use as well. The important elements incorporated into this design include:

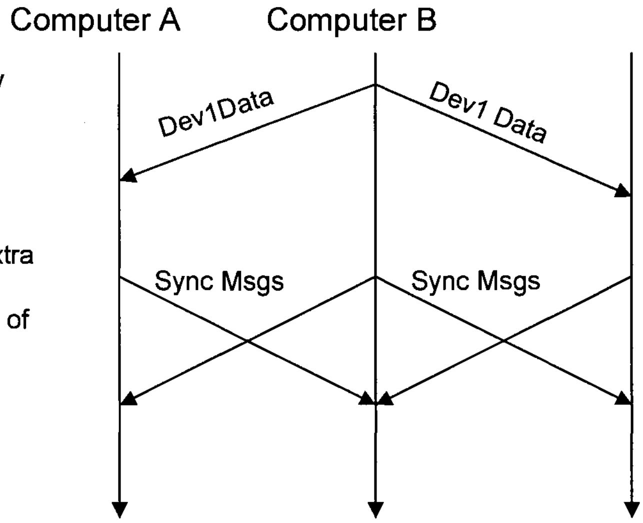
- separating the functionality of a system so it can be spread across cluster nodes,
- hiding the complexity of the cluster from the common application developer,
- focusing on input management to synchronize nodes when necessary, and
- allowing other developments such as Net Juggler to handle traditional high-performance computing clustering techniques for those who desire it and are willing to try it or are experienced in using MPI.

Chapter 5: Implementation Details

Cluster Frame Synchronization

The remote input manager not only distributes input but also synchronizes the entire cluster system. Even though some nodes' remote input managers aren't transmitting device data, they all send an end of frame message once per frame to confirm they have no more device data to send. This assures no node can start processing the next frame until it is has received an "End of Input" message from all other connected hosts. Since all main nodes are connected in some way to each other (at least through the devices host nodes), VR Juggler nodes synchronize by waiting for each other once per frame. This design avoids deadlocks as long as all nodes are active or legally disconnected. Future work may add detection of disconnects through time outs.Figure 12 gives a general idea of how input transmission data and synchronization messages work in our system.

B is connected to both A and C.
B is sending data from device Dev1 to both A and C.



1. Nodes send data if they have any to send.

2. Each node sends an extra message to allow synchronization at the end of each iteration.

Figure 12 Remote Device Update

Figure 13 shows a more realistic timeline of the data exchange for a single frame.

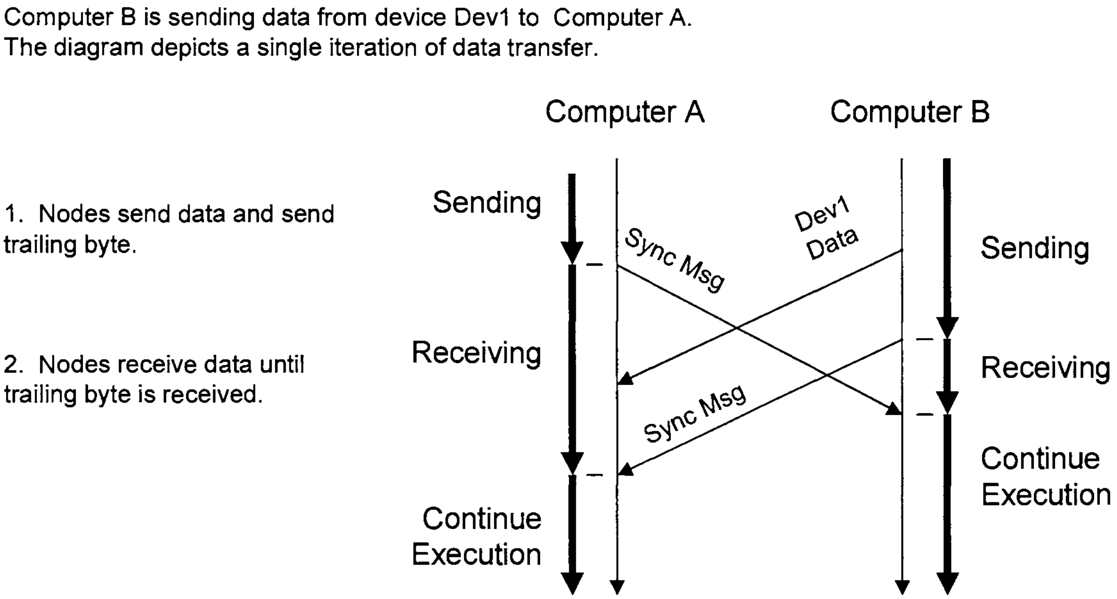

Computer B is sending data from device Dev1 to Computer A.
The diagram depicts a single iteration of data transfer.
Computer A
Computer B
1. Nodes send data and send trailing byte.
2. Nodes receive data until trailing byte is received.
Sending
Receiving
Continue Execution
Sync Msg
Dev1 Data
Sync Msg
Sending
Receiving
Continue Execution

Figure 13 Remote Input Manager Data Timeline

The process shown in the Figure 13 requires the configuration of devices to remain the same while data updates are being made. If a connection is added in the middle of the process, a node may never get sent the end of input message and wait for it indefinitely. Because our software utilizes multiple threads, the software needs to worry about thread safety issues like this. Mutex locks are used to assure certain sections of code are not executed at the same time. This is done by requesting control of the mutex before a section of code and releasing it afterwards. In our case, a mutex is requested and released around any code that modifies or depends on configuration data. This includes the process of updating remote input data and also the process making configuration changes such as adding new connections.

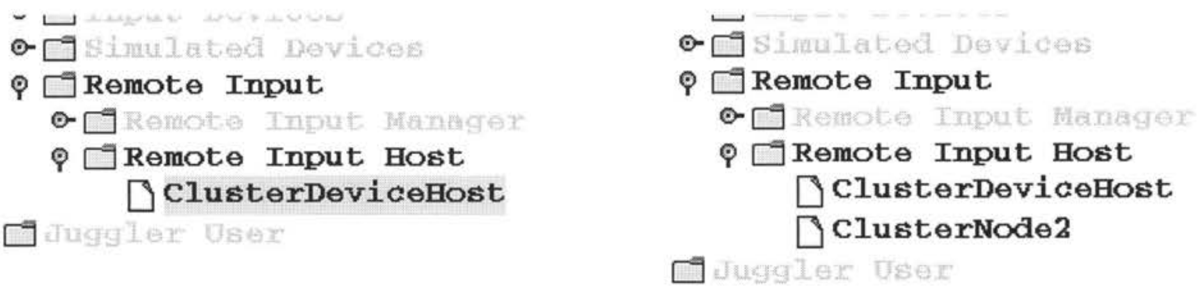
The implementation of the remote input manager allows it to perform its role in our cluster design: to provide a cluster of VR Juggler PCs with the ability to synchronize with each other once per frame (even if no devices are being shared), and to distribute input device data between nodes. When necessary, this allows nodes in a cluster to run the same application simultaneously with the same input and synchronized output.

Chapter 6: Practical Details to Configure a System

This chapter describes the practical details for setting up and using Cluster Juggler. As presented in the previous chapters, Cluster Juggler can be considered an extension to VR Juggler. Therefore we have taken advantage of the configuration capabilities of VR Juggler to use Cluster Juggler. The information provided in this chapter assumes that the reader has basic knowledge of VR Juggler.

VR Juggler has an extensive configuration system called *VjControl* that we introduced in Chapter Two. Different VR Juggler configurations are created with VjControl and stored in configuration files. The basic element of a VR Juggler configuration is a "chunk." These configuration chunks are used to setup pieces of a VR system including displays, input, output, or anything else that can be configured in a VR system. Cluster Juggler extends the chunks related to input devices to support a set of nodes connected by a network.

To use an input device with your application, a device chunk is required in the configuration. If a device has a VR Juggler driver there will be a device chunk available to configure it. Another type of chunk needed to access the input device is a "Proxy". Applications can access the device through a proxy, which simply points to the specific input device. The extension from this thesis adds an additional configuration item for the location of each proxy. This lets you specify which computer in the cluster your device resides on. For simplicity, the location is specified from a drop down list of computers in your cluster. Before you can select this location, you must first specify which computers are connected to each node in your cluster as shown in Figure 14.



```
Simulated Devices
Remote Input
   Remote Input Manager
   Remote Input Host
      ClusterDeviceHost
Juggler User
```

```
Simulated Devices
Remote Input
   Remote Input Manager
   Remote Input Host
      ClusterDeviceHost
      ClusterNode2
Juggler User
```

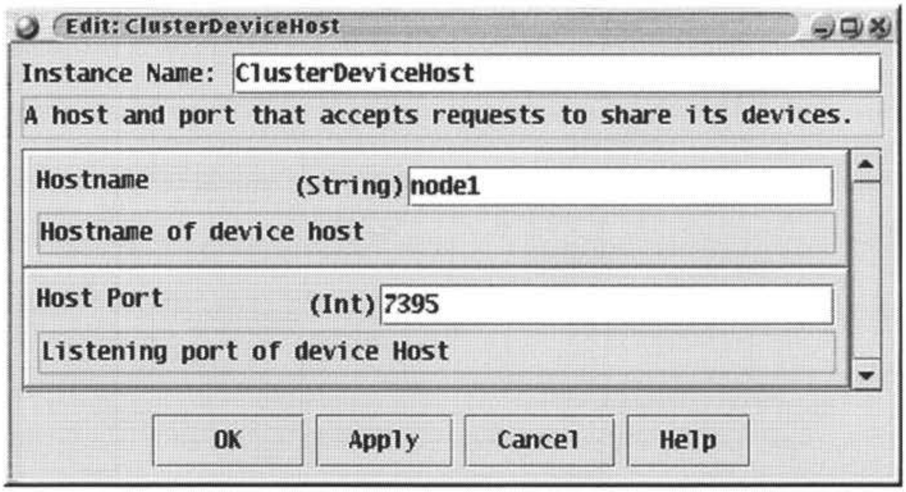
Figure 14 Sample Configurations

All the nodes read a list of hosts to determine which hosts they should initiate connections with. When starting an application, the nodes in the cluster process configuration files that contain this information. Each host can recognize its own hostname in the list and listen on the port specified there. If any other nodes are listed, it will attempt to connect to them.

To specify the computers or nodes in your cluster, "Remote Device Host" chunks are needed. As mentioned previously, we are using VjControl to create a VR Juggler configuration. We will create the two types configuration files shown in Figure 14: one for a node that will share its input devices and another for other nodes. It is possible to create a single configuration file containing the information for all cluster nodes, but at this time this method is not as efficient because it creates more connections than necessary.[3]

To create the first configuration file, create a new configuration file and insert a "Remote Device Host" chunk. Fill in the hostname and a listening port for the computer that has input devices(s) attached to it (any node if simulated devices are being used) and save it in a configuration file such as "cluster.devicehost.config". Figure 15 demonstrates how to do this. This file will tell the first node, our device host, how to configure itself.

[3] In the future, this will be the primary method of configuration as the system will be smart enough to determine which connections are unnecessary.


Edit: ClusterDeviceHost

Instance Name: ClusterDeviceHost

A host and port that accepts requests to share its devices.

Hostname (String) node1

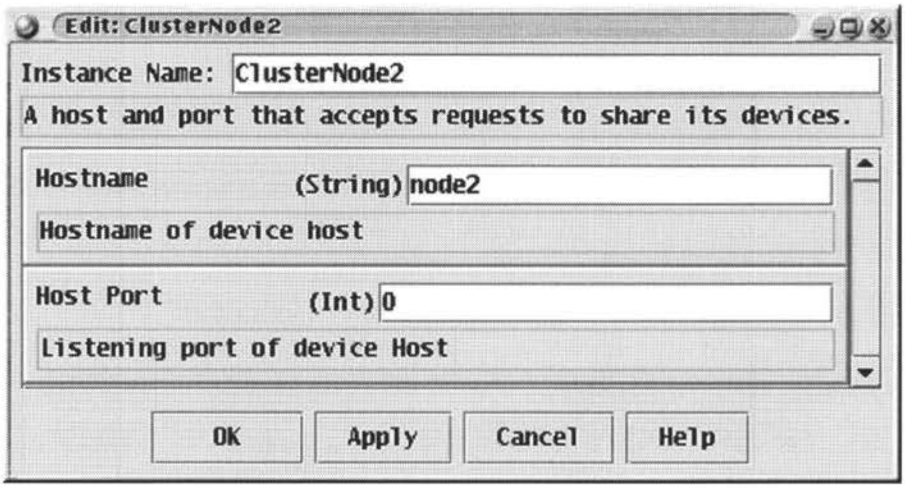
Hostname of device host

Host Port (Int) 7395

Listening port of device Host

Figure 15 Device Sharing Node

Next, insert another “Remote Device Host” chunk and name it “ClusterNode2.” As shown in Figure 16, fill in the hostname of the second computer, and set its listening port to zero since it doesn’t need to share any devices. Save this in a file such as “cluster.node2.config”. This file will tell the second computer how to configure itself and how to connect to our first node.


Edit: ClusterNode2

Instance Name: ClusterNode2

A host and port that accepts requests to share its devices.

Hostname (String) node2

Hostname of device host

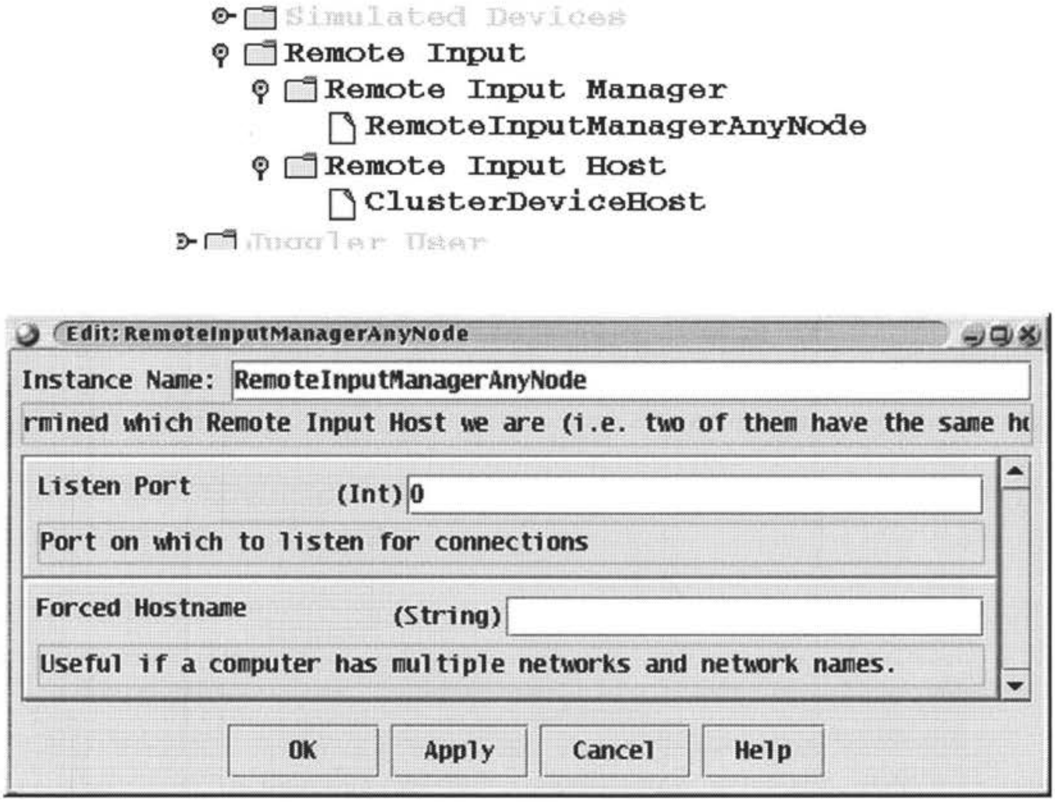
Host Port (Int) 0

Listening port of device Host

Figure 16 Cluster Node 2

Repeat the setup procedure for node2 to setup the rest of the cluster nodes. In each configuration, replace the hostname “node2” with the appropriate hostname and save to an appropriate file such as "cluster.node3.config". The configuration files we created only initiate a connection to the "ClusterDeviceHost" node. Only if another one of the nodes was sharing input devices would we need to add an additional connection per node.

A "Remote Input Manager" chunk is not needed in this example, but it can be useful in other situations such as distinguishing between nodes when testing Cluster Juggler on a single computer. Also, until the previously mentioned single configuration file[3] works efficiently, it can also be useful to create a common node configuration file without the need to specify a hostname. Figure 17 shows how a "cluster.anynode.config" can currently be used to replace "cluster.node2.config", "cluster.node3.config", etc.



Simulated Devices
Remote Input
Remote Input Manager
RemoteInputManagerAnyNode
Remote Input Host
ClusterDeviceHost
Juggler User

Edit: RemoteInputManagerAnyNode

Instance Name: RemoteInputManagerAnyNode

rmined which Remote Input Host we are (i.e. two of them have the same h

Listen Port (Int) 0

Port on which to listen for connections

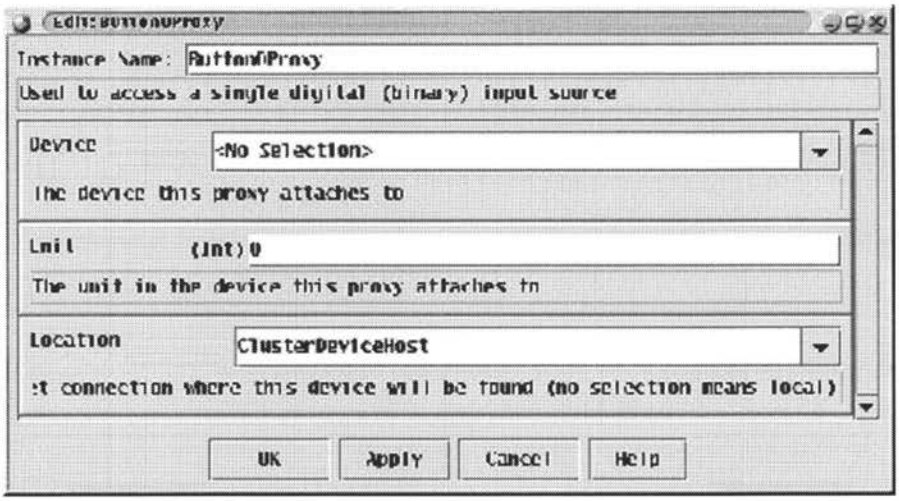
Forced Hostname (String)

Useful if a computer has multiple networks and network names.

Figure 17 Alternative Cluster Node N

Configuring Remote Devices

Now that we have specified the nodes in our cluster we can continue to specify the devices. The configuration of devices has not changed much from previous versions of VR Juggler, except there is an additional "Location" entry for each proxy as shown in Figure 18.



Edit: Button0Proxy

Instance Name: Button0Proxy

Used to access a single digital (binary) input source

Device <No Selection>

The device this proxy attaches to

Unit (Int) 0

The unit in the device this proxy attaches to

Location ClusterDeviceHost

:t connection where this device will be found (no selection means local)

Figure 18 Device Proxy

The options available from the Location drop-down menu are "No Selection", "ClusterDeviceHost", "ClusterNode2", etc. By picking "ClusterDeviceHost", we specify that the data for this device will be requested from the network host "ClusterDeviceHost". In our configuration so far, this refers to host and port *node1:7395*. The VR Juggler application running on "ClusterDeviceHost" will recognize that the device is located locally. It will configure it like a normal device instead of requesting it remotely.

A location should be specified for the proxies of all devices shared by the device host and used in the VR application. For simple applications, this applies to the Wand Proxy, Head Proxy, and all the Button Proxies. After changing the location portion of the device proxies, save the changes into new configuration files.

With your newly modified configuration files you should be ready to start your application. Currently you need to start the nodes one at a time, but this will be an automated process in the future. Start your application the same way you started the sample VR Juggler application: executable name first, followed by a list of configuration files. Don't forget to include the appropriate cluster node configuration file you created for each node.

Chapter 7: Results

The main goal of this thesis is to design a software environment that produces a working VR Cluster. As we designed the software, our main concern was over the performance of the resulting distributed system, especially in comparison to the shared memory systems used in most VR environments. We took a series of performance measurements to determine if the presence of a physical network and the need for synchronization would cause noticeable penalties in the overall performance of a PC cluster.

Test Variables

To evaluate the performance of the cluster and the shared memory system, we varied the number of displays surfaces (corresponding to number of nodes in the cluster) and the application type/load. Our current cluster can support a four-wall monoscopic display room[4] so our performance measurements are based on this system. The reason we measured performance on a single node was to evaluate the capabilities of the PC vs. the workstation. The single node tests did not use the work from this thesis, and neither did the shared memory system tests.

Test Systems

The PC cluster consists of four dual processor Dell Pentium III 600 MHz computers with 256 MB of memory each. The first two computers contain Nvidia GE Force 2 GTS graphics cards with 32 Mb of memory. The second two contain Nvidia Quaddro DCC graphics cards with 64 Mb of memory. The connecting network is dedicated Gigabit Ethernet to decrease latency, although standard Fast Ethernet could also be used. The gigabit

[4] After completing the performance testing and most of this work, additional computer systems have arrived to allow us to display a four-walled projection system in passive stereo with eight computers.

switch used is the Summit 5i switch from Extreme Networks. Because it is openly available, the operating system used for these tests is Linux (Redhat 7.2).

The shared memory system is a multi-rack 24 processor SGI Onyx2 with twelve gigabytes of memory and six IR3 image generators.

Test Applications

In our performance tests we wanted to select a group of representative VR applications that would cover the spectrum of graphics-intensive, computationally-intensive, and interactive-intensive workloads of VR applications. We selected three applications based on these characteristics.

The first application, Triangle, is extremely simple. It is designed to remove any computation or graphics load in order to provide us with a "pure" measurement of network performance. The second application, Vegas, contains a considerable amount of computation, graphics, and interaction. The third application, Pantheon, is a walkthrough of a model of the Pantheon. It is graphically intensive as it contains large amounts of polygons and textures. All the applications use OpenGL as an average programmer would use it. We did not use any graphics optimization or advanced tools such as Iris Performer to speed up the rendering.

Testing Conditions

The hardware is not entirely homogenous across the cluster. We aimed to make a fair performance analysis; therefore we established the following conditions for our experiments:

- Graphics boards:

  Two computers contain high performance Quaddro DCC graphics cards, and the other two computers contain GeForce2 GTS cards. The need to synchronize frames forces the frame rate of a cluster system to drop to the slowest graphics card and/or network connection, therefore, when testing with one or two nodes we measure performance on the computers with the slower GeForce2 graphics cards.

- Ensure same graphics load at each node:

  The difference in video cards also forces us to display the same view on each of the displays in our tests. Otherwise the speed of the system could vary depending on which node (and graphics card) is rendering the more complex part of the scene.

- Number of input devices present in a typical VR setup:

  Since input device data is passed across the network each frame, it is likely that the number of input devices slightly affects the performance. Therefore the tests use eight input devices (two positional, six digital) to ensure that a reasonable amount of data is used in all tests.

- Ensure identical input in tests on each platform

  In order for all tests to be consistent, the same input data must be used in tests on both platforms. To ensure the input data is identical, simulated input devices are used instead of input hardware such as tracking systems.

- Data collection

  The test applications were modified to display recent frame rate and average overall frame rate. When the overall frame rate stabilized at a constant value and the recent frame rate matched closely, we record the overall frame rate as the performance measurement. The initial setup frames of an application take considerably longer and therefore can either be discarded from the overall frame rate, or included, which simply requires more time for the overall rate to settle.

Results

Triangle Application

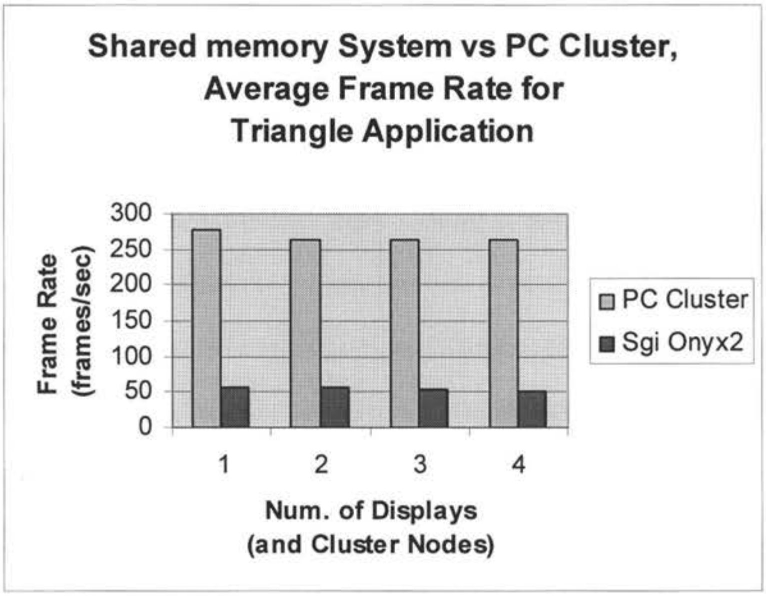


Figure 19 Performance Results for Triangle Application (Low Load)

In the Triangle test, both platforms produce more than acceptable results. The shared memory system maximum frame rate is limited by the video refresh rate which was set 60 Hz. Due to a difference in architecture, the frame rates in the PC systems were not limited by their video refresh rates so the results may seem slightly misleading. By removing computation and graphics load, this test isolates the network communications and synchronization in the cluster. The results show that a cluster can synchronize at very high rates while only introducing a slight performance penalty. The slight initial impact of the inter-node communications appears as expected in a two node scenario and remains fairly constant for subsequently added nodes.

Vegas Application

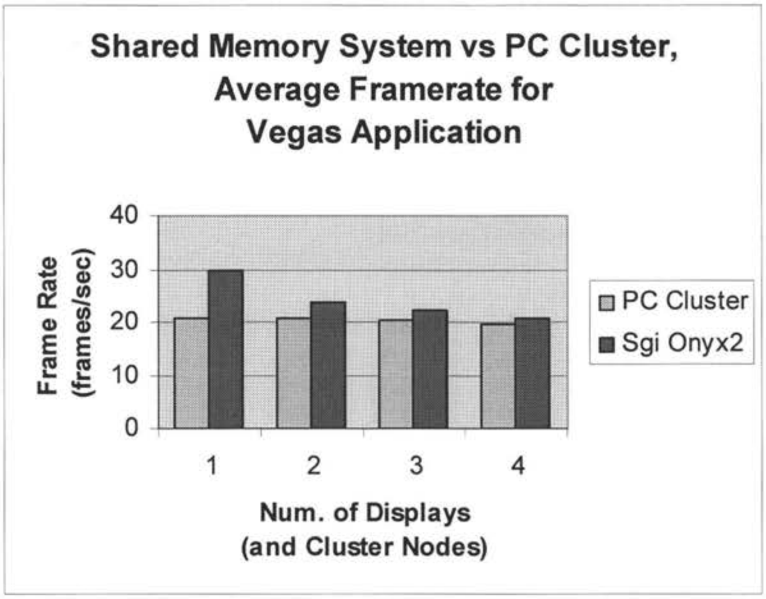


Figure 20 Performance Results for Vegas Application (Average Load)

The Vegas application shows similar performance measurements for both types of systems. In performance for CAVE-like systems, it is most important to compare the results at the largest number of displays, which are almost identical in this case. It is also interesting to note that performance decreases on the shared memory system as the number of displays increases. Although investigating specific graphics hardware performance issues are outside the scope of this thesis, the performance drop could be caused by conflicts in interleaved memory accesses, the duplication of data between displays resulting in high memory usage, or problems with the synchronization method used by the shared memory system.

Prior to testing, the addition of displays/nodes was expected to noticeably decrease the performance of a PC cluster due to its external network, but it seems that the synchronization of data in the cluster takes less time per frame than expected.

Pantheon Application

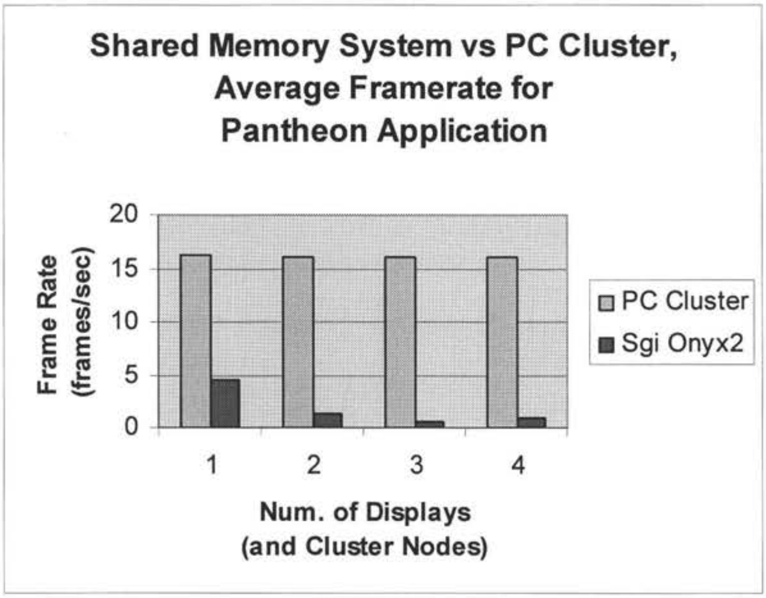


Figure 21 Performance Results for Pantheon Application (High Graphics Load)

The Pantheon application demonstrates the strengths of current PC systems and graphics cards. The application mainly consists of rendering graphics primitives with and without textures. The sheer number of polygons, the fill rate, and the number of (graphics API) function calls likely slow the performance on the shared memory system. Again the number of nodes does not appear to severely impact cluster performance.

Summary of Results

The purpose of the tests is to show that the performance of VR on a PC cluster does not prevent it from being a viable alternative to VR on a shared memory system. The results show that a PC cluster can perform as well as a shared memory system and in some cases better. The tests also show that adding additional nodes up to the amount needed for a CAVE-like system does not considerably impact performance.

Chapter 8: Conclusions and Future Work

Conclusions

With Cluster Juggler we provide the ability to run VR applications on a cluster of PCs instead of a large shared memory system. Hardware improvements in PCs have made a PC cluster a viable and cost effective alternative to shared memory systems, but additional synchronization and network software is needed to allow VR applications to take advantage of such a cluster. In providing software that allows us to use PC clusters for VR, we attempt to remove the complexity of clusters and allow application developers to develop applications instead of worrying about communication between computers. Our goal is to provide this software while making as few demands on the developers and users as possible. There are enough complications involved in virtual reality that it is important to remove low-level complexities where possible.

This thesis attempts to make VR more accessible and affordable. There are several fundamental obstacles to achieving this goal, but the ability to use mass produced computer hardware in VR is a major step towards it. Our work contributes to this goal by simplifying the network and synchronization programming necessary to create distributed VR applications. This will enable developers with basic programming skills to build complex cluster-based VR applications. In addition, it also enables the use of mixed resources to drive these complex applications.

Future Work

- Hiding All Cluster Complexities

This work for this thesis is primarily concerned with allowing VR to work on a PC cluster. Incorporated into the design is the attempt to completely hide the complexity of the cluster from the user. This is not entirely possible and there is still room for improvement in this area. For example, users still need to worry about synchronizing random data between nodes.

- Simplifying Configurations (Configuration Wizard)

  Another component that could be even more simplified is the configuration. There is not a great deal of complexity in specifying the location (host) of a device, but the addition of a simplified configuration or wizard could simplify the setup for new users. This convenience can be added when such a wizard system is incorporated into VR Juggler.

- Complete Data Synchronization

  The design of this thesis does not intend to synchronize all of an application's data across the cluster nodes. It synchronizes the input to the nodes which allows the output to also be synchronized. A complete synchronization of all data on all nodes would require a more complex fully distributed system.

- Larger Number of Nodes

  Further investigation is necessary to test our design's scalability to a larger number of nodes. We believe that this will not be a major concern in our design because VR systems to not require a massive number of nodes as conventional computational clusters do. A further verification of scalability to 12 or 20 nodes may be appropriate.

- Integration with a Computational Cluster

  In the Cluster Computing Power section of Chapter 4, we provide some initial discussion on how the remote input manager could aid in communicating between a VR cluster and a computational computing cluster. This is an unexplored area of research that may open interesting opportunities.

- Hierarchical Framework

There may be uses for this software that can benefit from organizing cluster nodes into a hierarchy. Our design goals do not require this functionality, but this capability could be useful if a massive number of nodes or displays is ever used. The increase in time and computing resources needed to synchronize such a large number of nodes with each other could by reduced if the nodes were organized into such a hierarchy.

Bibliography

[Bierbaum00] Allen Bierbaum, VR Juggler: "A Virtual Platform for Virtual Reality Application Development", MS Thesis, Iowa State University, 2000.

[Blom01] Kris Blom, "Multiple viewers in projection-based multi-screen immersive environments", Iowa State University, 2001.

[Cruz93] Carolina Cruz-Neira, "Applied Virtual Reality," Course #23, ACM Siggraph '93 Conference, Anaheim, CA, July 1993.

[Cruz95] Carolina Cruz-Neira "Projection-based Virtual Reality: The CAVE and its Applications to Computational Science", Ph.D. Thesis, University of Illinois at Chicago, 1995.

[Held91] R. Held & N. Durlach. "Telepresence, time delay and adaptation". in S. Ellis (ed.) *Pictorial Communication in Virtual and Real Environments*, Taylor and Francis, 1991, 232-246.

[Kolasinksi95] Eugenia M. Kolasinski, "Simulator Sickness in Virtual Environments", May 1995. U.S. Army Project Number 2O262785A791, Education and Training Technology. http://www.cyberedge.com/4a7a.html, date accessed: April 24, 2002.

[Mouse01] http://www.tweak3d.net/tweak/mouse/2.shtml, http://www.bluesnews.com/mouse.shtml, http://www.blog.com/ps2.htm, date accessed: April 24, 2002.

[MPI] The Message Passing Interface (MPI) Standard, http://www.mcs.anl.gov/mpi, date accessed: April 24, 2002.

[NetJglr01] Net Juggler, Scalable High Performance Virtual Reality Project, Laboratoire d'Informatique d'Orléans, Université d'Orléans, http://netjuggler.sourceforge.net/NetJuggler.php, date accessed: April 24, 2002.

[Pimentel95] Ken Pimentel and Kevin Teixeira, "Virtual Reality, Through the Looking Glass," 2nd Edition, 1995

[PVM] PVM: Parallel Virtual Machine, official site, http://www.epm.ornl.gov/pvm, date accessed: April 24, 2002.

[SoftGen01] SoftGenLock, Scalable High Performance Virtual Reality Project, Laboratoire d'Informatique d'Orléans, Université d'Orléans. http://netjuggler.sourceforge.net/SoftGenLock.php, date accessed: April 24, 2002.

[Stuart96] Rory Stuart "The Design of Virtual Environments," 1996.

[Wiregl01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett and Pat Hanrahan, WireGL, "A Scalable Graphics System for Clusters", *Proceedings of SIGGRAPH 2001*, http://graphics.stanford.edu/papers/wiregl, http://graphics.stanford.edu/software/wiregl, date accessed: April 24, 2002.