BUILDING A HIGH-RESOLUTION SCALABLE VISUALIZATION WALL

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information

_____

Zhenni Li

Certificate of Approval:

_____

Kai H. Chang
Professor
Computer Science and Software Engineering

_____

W. Homer Carlisle, Chair
Associate Professor
Computer Science and Software Engineering

_____

Cheryl D. Seals
Assistant Professor
Computer Science and Software Engineering

_____

Stephen L. McFarland
Acting Dean
Graduate School

BUILDING A HIGH-RESOLUTION SCALABLE VISUALIZATION WALL

Zhenni Li

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements of the

Degree of

Master of Science

Auburn, Alabama
December 15, 2006

BUILDING A HIGH-RESOLUTION SCALABLE VISUALIZATION WALL

Zhenni Li

_____
Signature of Author

_____
Date of Graduation

VITA

Zhenni Li, daughter of Shiqi Li and Huiqin Du, was born in September 1981 in Chongqing, the People's Republic of China. She entered Chongqing University in China in 1999 and received a Bachelor of Science degree in Automation Engineering in July 2003. Ms. Li entered Graduate School at Auburn University in January, 2004.

THESIS ABSTRACT

BUILDING A HIGH-RESOLUTION SCALABLE VISUALIZATION WALL

Zhenni Li

Master of Science, December 15, 2006
(B.S. Chongqing University, 2003)

71 Typed Pages

Directed by W. Homer Carlisle

High-resolution and scalable displays are increasingly being used for interactive 3D graphics applications, including large-scale data visualization, immersive virtual environments, and scalable 3D rendering subsystems. A display wall is a unique environment by combining a high-resolution display surface and a parallel computing cluster. Most display wall systems today are built with high-end graphics machines. We are investigating how systems that use only inexpensive commodity components of a PC cluster are build.

Rocks is an excellent commodity-based cluster management toolkit, which centers around a Linux distribution based on Red Hat. Additionally, Rocks allows end-users to add their own software via a mechanism called Rolls. Viz is one of the Rolls provided in Rocks to build tiled display visualization clusters.

In this thesis, we describe the architecture, major components and render algorithms of this Viz visualization cluster in detail. We also show how to build, configure, and

implement visualization programs with several approaches. Our experiment validates the framework and principles of the cluster as well as performance improvement in terms of service availability, load balancing brought about using a sort algorithm.

ACKNOWLEDGEMENTS

I would like to express sincerely appreciation to Dr. Homer Carlisle for his guidance, insight, and encouragement throughout the research to help me succeed.

I would also like to thank the rest of my thesis committee members, Dr. Kai Chang and Dr. Cheryl Seals for their valuable suggestions and comments.

Last but not least, I would like to thank my family and friends for their understanding, motivation and support.

Style manual or journal used: Journal of SAMPE

Computer software used: Microsoft Word

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

Very often applications need more computing power than a sequential computer can provide. One way of overcoming this limitation is to improve the operating speed of processors and other components so that they can offer the power required by computationally intensive applications. Even though this is currently possible to a certain extent, future improvements are constrained by the speed of light, the high financial costs for processor fabrication and etc. A viable and cost-efficient alternative solution is to connect multiple processors together and coordinate their computational efforts. The resulting systems are popularly known as parallel computers, and they allow the sharing of a computational task among multiple processors [1].

A cluster is a type of parallel processing system which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource. Clusters can be used to prototype, debug, and run parallel applications. It is becoming an increasingly popular alternative to using specialized, typically expensive, parallel computing platform. An important factor that has made the usage of clusters a practical proposition is the standardization of many of the tools and utilities used by parallel applications.

Traditionally, in science and industry, a workstation referred to a UNIX platform and the dominant function of PC-based machines was for administrative work and word processing. The coverage of kernel-level functionality of UNIX workstations in PC-based machines led to an increased level of interest in utilizing PC-based systems as a cost-effective computational resource of parallel computing [1].

Nowadays, though there is a rapid increase in commodity-based cluster management toolkit, the complexity of cluster management (e.g., determining if all nodes have a consistent set of software) still overwhelms part-time cluster administrators. When this occurs, machine state is forced to either of two extremes: the cluster is not stable due to configuration problems, or software becomes stale, security holes abound, and known software bugs remain unpatched [2].

While earlier clustering toolkits expend a great deal of effort (i.e., software) to compare configurations of nodes, Rocks makes a complete cluster installation on nodes. It uses a graph–based framework to describe the configuration of all node types (termed appliances) that make up the cluster. This toolkit centers around a Linux distribution based on Red Hat. Additionally, Rocks allows end-users to add their own software via a mechanism called Rolls. Rolls are a collection of packages and their configuration details that modularly plug into the base Rocks distribution. These Rolls are used prevalently in many areas including databases, distributed (grid-based) systems, visualization, and storage [3].

This thesis will focus on a Roll called Viz Roll. This Roll is used to build tiled display visualization clusters. It creates a new Rocks appliance type called a “Tile”. Just as the “Compute” appliance is used to build compute clusters, this new “Tile” appliance

is used to build visualization clusters. The core software that drives the tiled display is called DMX [4]. DMX creates a single unified X11 desktop from multiple graphics cards, monitors, and computers. Visualization clusters are built to create larger displays (thousands by thousands of pixels), and are primarily of interest to OpenGL applications [5]. DMX is also being used to provide for scientific visualization on the TeraGrid [6].

1.2 Motivation & Research Objective

The scale and resolution of a display device defines how much information a user can view at a time. Large data visualization problems generally demand higher resolutions than are available on typical desktop displays.

Consider the case of a building covered by "digital wallpaper." It may have over 100 million square inches of surface area on the walls, ceilings, and floors, which could accommodate around 500 gigapixels of display imagery (at 72 dpi). If each image is updated 30 times per second, then the display system must be capable of updating 15 terapixels per second, or 500,000 times more pixels per second than current display devices. Fortunately, in most reasonable scenarios, not all the displays need to be updated at full resolution, or at video frame rates, all at once. Instead, a few displays showing feedback associated with user interaction require fast refresh rates, while most other displays can be updated less frequently or at lower resolution. Meanwhile, the displays not visible to any user can be left blank. The challenge is to design a rendering system powerful enough and flexible enough to drive a large number of pixels spread over multiple displays in a dynamic environment [7].

The most natural strategy is visualization clusters, which can tile multiple projection devices over a single display surface. The products using this method are usually called Display Walls [8].

A typical display wall uses special piece of hardware, called a video processor, to scale lower-resolution video content, such as in NTSC, VGA, SVGA, and HDTV formats to fit a large display surface. It does not provide digital applications with higher intrinsic display resolution [8]. In order to provide higher intrinsic resolution, some research prototypes and commercial products use a high-end graphics machine with multiple tightly coupled graphics pipelines to render images and drive the tiled display devices [9]. Such systems are very expensive, often costing millions of dollars.

Not all virtual reality applications today require the power or expense of single large visualization “super-computers”. The Scalable Display Wall is much more low-cost. It can use commodity components such as PCs, PC graphics accelerators, off–the-shelf network, consumer video and sound equipment. The better price and performance of these components often improves the quality faster than special-purpose hardware [10].

Our goal is to research current technology with a being to find a system using low cost commodity components to build a display wall in the new computer science building. The challenging obstacle exists in developing scalable algorithms to partition and distribute rendering tasks effectively under the bandwidth, processing, and storage constraints of a distributed system. Rocks Cluster is a tool that can be installed easily, abstract out many of the hardware differences and can benefit from the robust and rich support from commercial Linux distributions.

1.3 Contribution of This Research

The primary set of problems that this research effort addresses is;

- The definition, architecture, and implementation of a visualization cluster using Viz Roll
- The investigation of applications that the visualization cluster can be applied
- To understand ways in which this system might be improved.

The thesis is organized as follows. Chapter 1 introduces the background of visualization cluster and raised the objectives of this research. Chapter 2 reviews the related work and former methods that are used to support visualization clusters. The related technologies are also discussed. Following this review, we present the software components, architecture and implementation of the visualization cluster in Chapter3 and Chapter4. In the Chapter 3, we also include an algorithm applied in the cluster. Finally, conclusions are given in Chapter 5 discussing the benefits and limitations of the tool as well as future work that would be done in this research area.

2. LITERATURE REVIEW

As there are a number of separate areas of research that have influenced this work, we will discuss related technologies in the following areas respectively:

2.1 Former Methods

The common way to run digital applications on a video wall is to use the video content scaling hardware (video processor) to drive an application's output on a tiled display system. This approach does not allow application to use the intrinsic resolution of the tiled displays [8].

- Power Wall

The Power Wall [11] developed at the University of Minnesota aims to visualize and display very high resolution data from large scientific simulations performed on supercomputers or from high resolution imaging applications. In addition to this high resolution, a single 6 foot by 8 foot screen illuminated from the rear by a 2 by 2 matrix of Electrohome video projectors can facilitate collaborations of small groups of researchers using the same data. All the collaborators can see the display clearly without obstruction, and the rear-projection technology makes it possible to walk up to the display and point to features of interest, just as one would do while discussing work at a blackboard [11].

These projectors are driven by 4 RealityEngine2 graphics engines. Each projector provides a resolution of 1600x1200 pixels (~2 MegaPixels), making the entire

PowerWall resolution 3200 x 2400 pixels (~8 MegaPixels). The Ciprico disk arrays supply the RealityEngines with more than 300 MegaBytes per second of data in order to display smooth motion animation across the entire viewing area. The Power Wall does not consist solely of a high resolution display system; it is in itself a supercomputing system. In the configuration set up at Supercomputing '94, the Power Wall is an integrated visualization system connected by a HiPPI network to the POWER CHALLENGE Array distributed parallel processing system which includes large and extremely fast disk storage systems for raw or image data and many powerful Silicon Graphics MIPS R8000 processors [11].

Figure 1 Power wall (from ref [11]).

The Power Wall can be used as a Virtual Reality (VR) system as well by utilizing specialized software for navigating through data sets. These data sets could come from computer simulations and can be accessed by applications running on the Silicon Graphics systems that drive the Power Wall. As the user explores the data sets, the Power Wall also becomes a window onto the virtual world of the simulation [11].

The Main drawback of this approach is that it is very expensive, sometimes costing millions of dollars.

- Infinite Wall

Infinite Wall [11] at the University of Illinois at Chicago is a VR version of the Power Wall. Virtual Reality (VR) can be defined as interactive computer graphics that provides viewer-centered perspective, large field of view and stereo. This projection-based VR system is often used as a collaborative VR tool to run NCSA and EVL programs such as NICE and CALVIN. In these situations, the I-Wall becomes a networked VR environment based on the CAVE [12].

The Infinity Wall is designed for presentations to large groups, as in a classroom setting. It comprises a single 9x12 foot (or larger) screen, four projectors which tile the display, one or two SGI Onyxes to drive the display, and a specified tracking system.

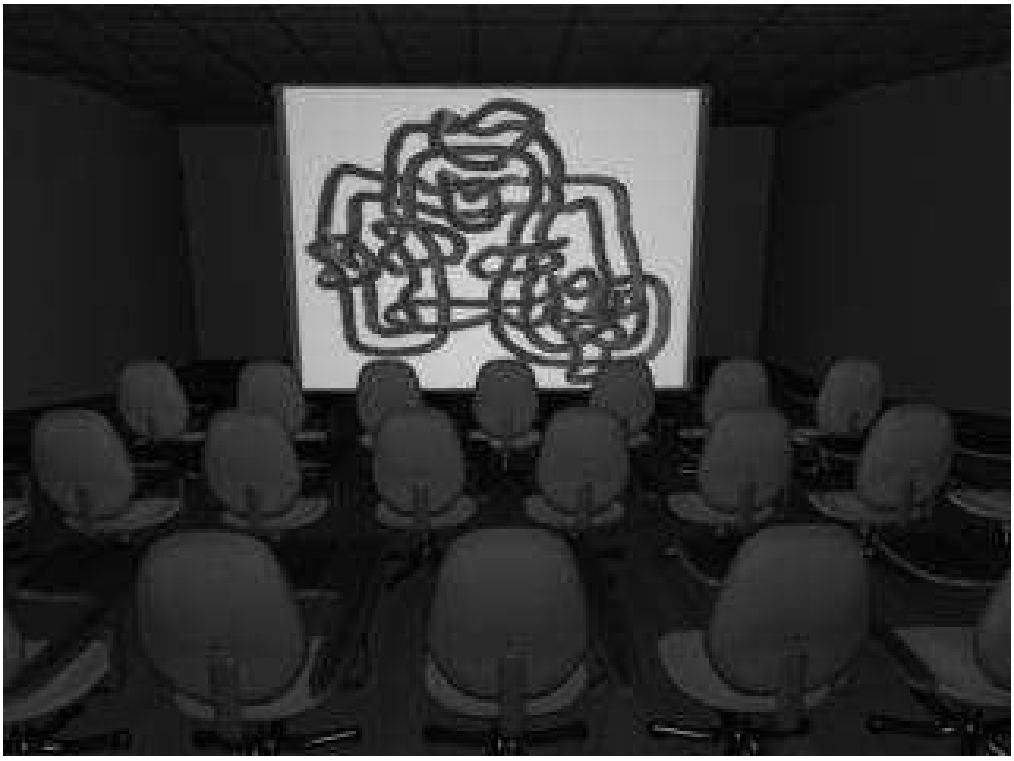

Figure 2 The infinite wall (From ref [12]).

Though the CAVE [13] achieves the goals of producing a large angle of view, creating high-resolution full-color images, allowing a multi-person presentation format, it is big and expensive. The graphics engines for the rendering costs about $100,000.

Examples of display wall also include the Information Mural at University of Stanford [14], and Office of the Future of UNC [16]. Each in these cases is driven by an SGI Onyx2 or equivalent high-end machine that has multiple graphics pipelines. Since these systems are not using a PC-cluster architecture, they do not address the issue of software support for running sequential, off-the-shelf applications on a scalable resolution display wall.

The parallel graphics interface designed at Stanford [16] proposes a parallel API that allows parallel traversal of an explicitly ordered scene via a set of predefined

synchronization primitives. The parallel API was not designed to execute multiple instances of a sequential program on a scalable display system built with a PC cluster.

- Interactive Mural

Stanford's "Interactive Mural" is comprised of several individual displays arranged in an array and used as one large logical display. It use an overlapping four by two array of projectors that back-project onto a diffuse screen to form a 6' by 2' display area with a resolution of over 60 dpi. The system is designed to be scalable, supporting variable numbers of computers, graphics cards, and video output ports. Although the system is for a distributed shared memory machine with two graphics pipes, it also can be implemented using a PC cluster configuration.

The graphics system is designed as a distributed server to support multiple remote clients. Each client is a different application and is typically running on a different machine. Unlike the X window system, which typically has a single server per display, it is distributed since the graphics system may be partitioned across multiple machines. The system was also designed to support simultaneous rendering from multiple streams [14]. Interactive Mural allows multiple applications to share the display space without overloading any one system component. To achieve this, it was necessary to separate the Mural into a client /server architecture, and to define a wire protocol to transport the entire API and callback mechanism between the Mural server and the Mural applications.

This system works as follow procedures:

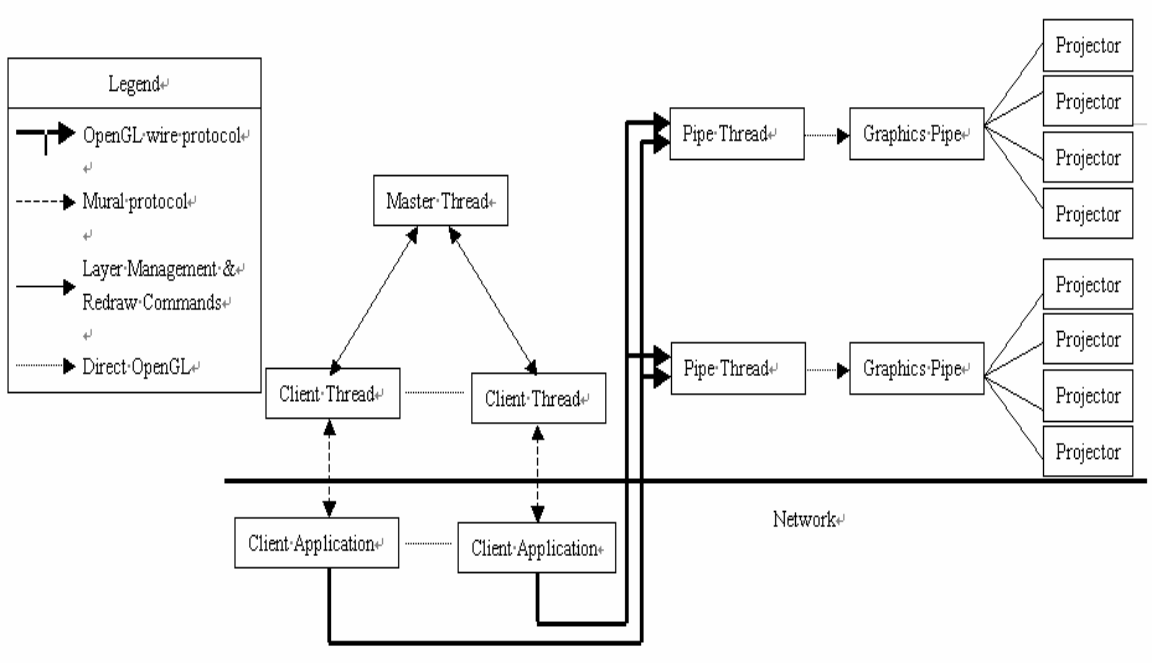

Figure 3 Block diagram of interactive mural system architecture.

Each application may be running on a different remote site. Notice that graphics commands do not go directly to the graphics pipes, but rather through the pipe thread "proxies", to avoid context switching the graphics pipe. The pipe thread will cache these graphics commands for use when a redraw is required [16].

The system can be scaled to larger displays by a network of workstations. One workstation is for each projector. In this configuration, the "pipe threads" in figure 3 become separate "pipe servers" that can communicate with the "master thread" over a network. The new version of the system uses eight PC's driving eight projectors and a ninth computer running the Mural server, all tied together on a dedicated high speed network. Because of the scalability of broadcast networking on a local area network, such a system should be cost-effective and scale well. To address the remote rendering overhead, a plug-in architecture is designed for the Mural server, so that speed critical

rendering routines can be built into the server and render directly to the screen. On the SGI version of the system, these plug-ins can keep their scenes and other data consistent using a simple shared memory model, but this is a much more challenging task on the PC version. The group is still investigating methods for simplifying data management and consistency in distributed remote rendering. Speed critical applications (such as applications using real time head tracking) may be needed to manage a large scene database and properly handle updates to that database in a consistent, fast, and scalable way [16].

2.2 Related Work

There are many ways to virtualize a tiled display system in software. MacOS and, more recently, Microsoft Windows have support for extending the desktop onto multiple monitors connected to a single computer system. In these cases, the device driver takes care of managing the screen real estate and the distributed framebuffer memory. Similarly, the latest release of the X Window system contains the XINERAMA extension, which allows multiple displays to be combined into one large virtual desktop. The Silicon Graphics InfiniteReality [17] system allows a single framebuffer to drive multiple displays through one large logical X server.

A more general approach is taken by DEXON Systems' DXVirtualWall [18], which provides extremely large tiled displays that run either X Windows or Microsoft Windows. Clients connect to a display proxy that broadcasts the display protocol to multiple display servers, each of which offsets the coordinates to display its own small portion of the larger desktop. Another use for protocol proxies of this nature is to duplicate a single

display across several remote displays, as in Brown University's XmX [4] project. These proxy-based systems do not currently support any high performance 3D graphics API such as OpenGL or Direct3D, and do not handle overlapping displays. Additionally, as the number of displays gets very large, the number of concurrent redraws exerts increasing pressure on the network, causing performance to suffer.

In the area of remote graphics, the X Window System has provided a remote graphics abstraction for many years. This system has a heavily optimized network usage model. GLX is the dominant API and protocol used for rendering OpenGL remotely over a network. GLX provides a seamless way to display 3D graphics on a remote workstation [6]. However, neither GLX or X window is designed to operate efficiently over very high-speed networks. WireGL [19] in Stanford University, introduces new techniques for data management and distribution to make effective use of high-speed networks.

WireGL uses a cluster of off-the-shelf PCs connected with a high-speed network. It allows an unmodified existing application to achieve scalable output resolution on such a display.

WireGL support an immediate-mode API like OpenGL to users. It is implemented as a driver that stands in for the system's OpenGL driver so applications can render to a tiled display without modification. The rendering severs form a cluster of PCs, each with its own graphics accelerator. The output of each server is connected to a projector which projects onto a common screen. Then these projectors are configured into a tiled array so that their outputs produce a single large image [19].

Actually, WireGL built a 36-node cluster named "Chromium". Chromium consists of 32 rendering workstations contains dual Pentium III 800MHz processors and an

NVIDIA GeForce2 GTS graphics accelerator. The cluster is connected with a Myrinet network which provides a point-to-point observed bandwidth of 100MB/sec. Each workstation outputs a 1024*768 resolution video signal, which can be connected to a large tiled display [19].

We have reviewed the Power Wall, which uses the output of multiple graphics supercomputer to drive multiple outputs, and create a large tiled display for high resolution video playback and immersive applications. The Infinite Wall, which extend this system to support stereo display and user tracking. Both of these systems are designed to facilitate a single full-screen application, which is often an immersive virtual reality system. Also they use very expensive hardwares.

Then we reviewed the Interactive Mural, which is designed as a distributed server to support multiple remote clients and also designed to support simultaneous rendering from multiple streams. The Interactive Mural is quite simple to implement, but suffers 2 major drawbacks. First, hardware accelerated off-screen buffers are an extremely scarce resource. Second, switching the contexts for the pipes between the applications and the Mural server is very expensive operation.

The X Window system that contains the XINERAMA extension, provides an easy way to allow multiple displays to be combined into one large virtual desktop. The WireGL uses a rending system just like X Window system. Also, it uses a sorting-first algorithm which works with immediate-mode rendering.

With all the former methods and related work, we have a roughly impression of display wall. In the following chapters, we will introduce a way to build a visualization cluster using workstations.

3. THE ARCHITECTURE DESIGN

This chapter describes the design methodology, introduces the components of the system and system architecture for large scale visulization.

3.1 The Design Methodology

Our design goal is to build a scalable visualization cluster that is based on inexpensive workstations and connected with a high-speed network. The cluster has a distributed server to support multiple remote clients. Each client is a different application and is typically running on a different machine. In order to display the graphics on the cluster's PCs, this system need to provide OpenGL. a graphics API for specifying polygonal scenes and imagery to be rendered on a display.

Our approach is to use a software cluster distribution called Rocks [2]. With the attribute of the Rocks cluster, it is easy to build a visualization cluster. After Rock visualization (Viz) configuration, OpenGL programs can be rendered on the displays and the position or size can be changed by users.

The detail architecture and principle of this system will be described in the following.

3.2 System Components

3.2.1 Rocks Cluster

High-performance clusters have become the computing tool of choice for a wide range of scientific disciplines. However, straightforward software installation, management, and

monitoring for large-scale clusters have been a consistent and nagging problem for non-cluster experts [2]. The free Rocks Clustering Toolkit has been in development for 5 years driven by the goal to make clusters easy to install and manage.

The Rocks toolkit is based on the Red Hat Linux operating system. Unlike a user's desktop, the OS on a cluster node is considered to be soft state that can be changed and reinstalled rapidly. At first glance, it seems wrong to reinstall the OS when a configuration parameter needs to be changed. Indeed, for a single node this might seem too severe. However, this approach scales exceptionally well, making it a preferred mode for even a modest-sized cluster. Because the OS can be installed from scratch in a short period of time, different (and perhaps incompatible) application-specific configurations can easily be added on nodes. In addition, this structure insures any upgrade will not interfere with actively running jobs [26]. This is clearly more heavyweights than the philosophy of configuration management tools that perform exhaustive examination and parity checking of an installed OS [3].

Figure 4 shows the cluster software stack. We can see Rocks mainly acts the middleware that resides between the operating system and user-level environment. Another part is HPC (High Performance Computing) driver which is used to glue together operating systems on all nodes to offer unified access to system resources.

The Roll, like a “package” for car, breaks apart Rocks as in Figure 5. Effectively, Rolls are first-class extensions to the base system. They provide optional configuration and software to give us more flexibility.

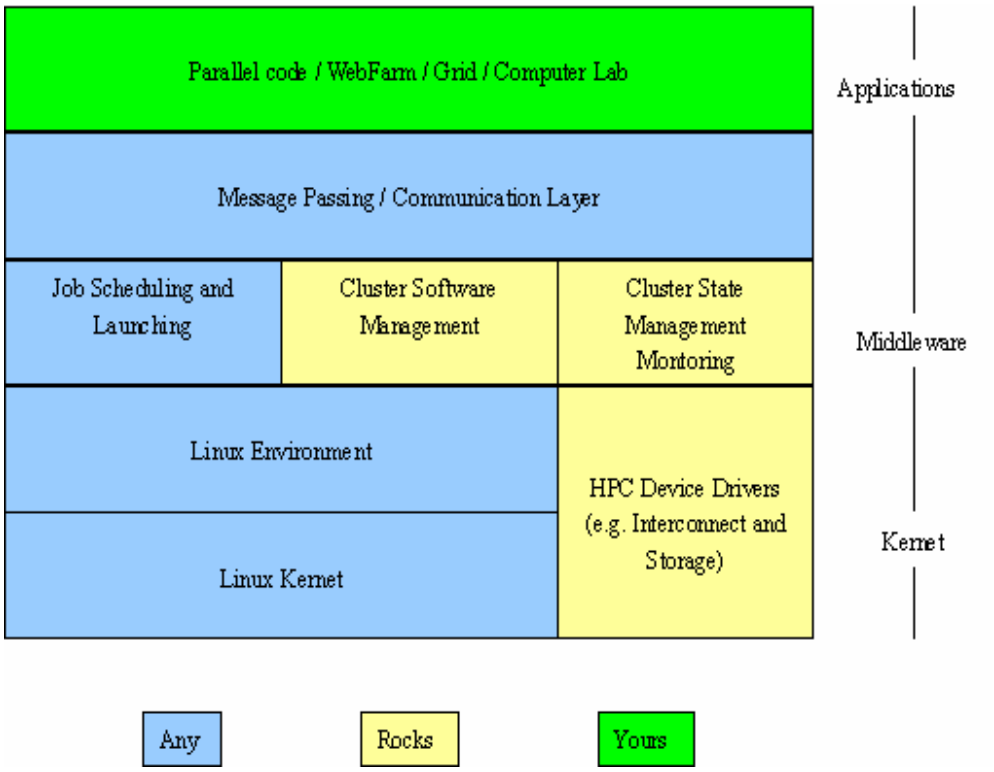

Figure 4 Cluster software stack

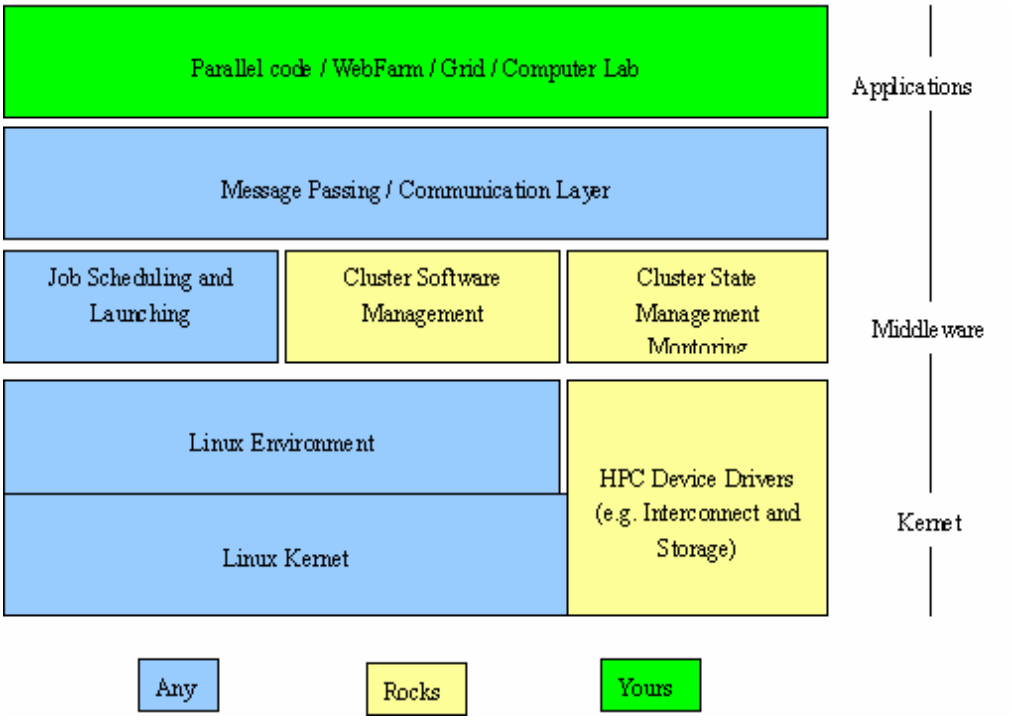

Figure 5 Rolls break the Rocks

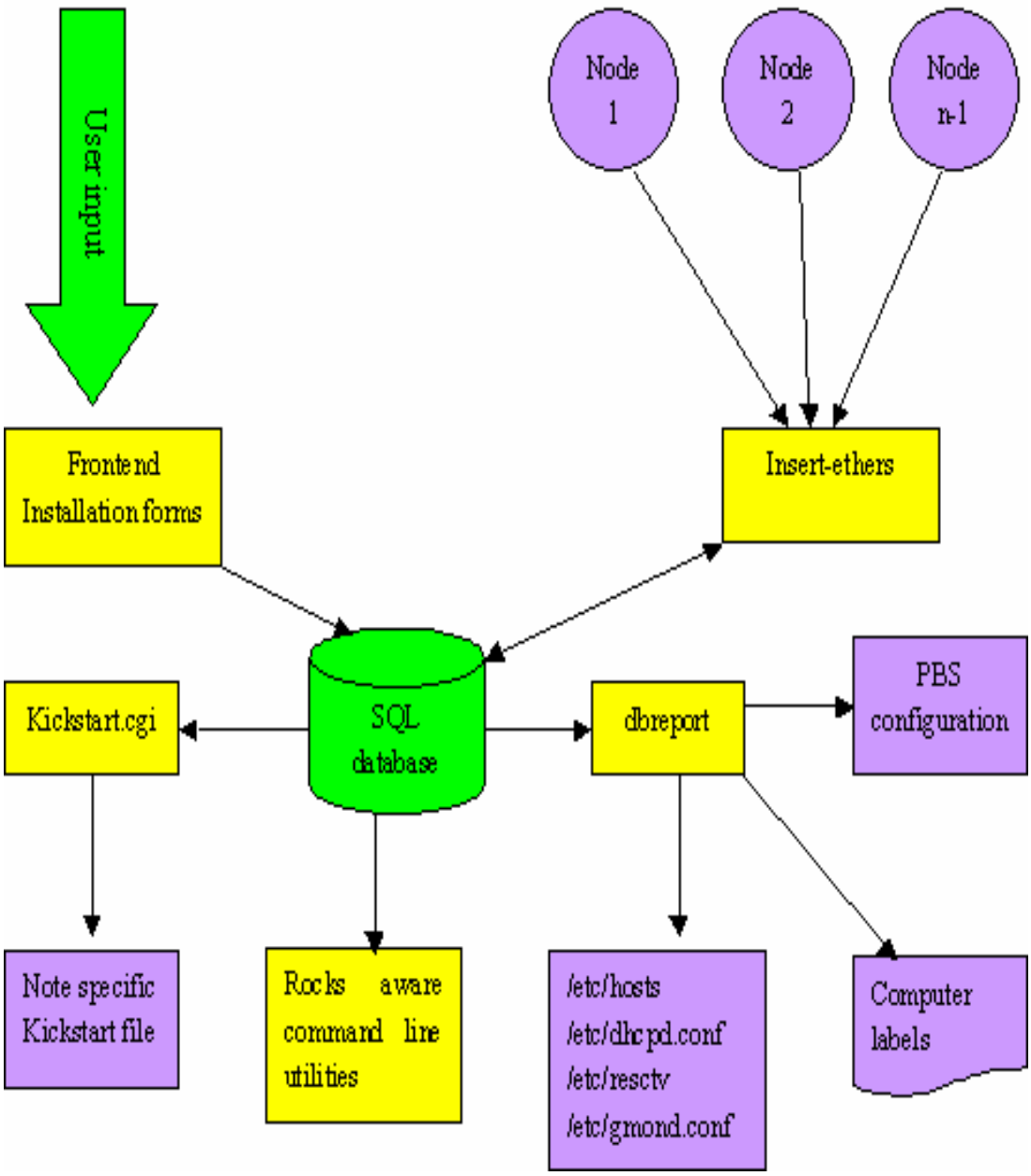

Figure 6 Cluster database structure

Rocks uses automatic methods to determine configuration differences. Yet, because clusters are unified machines, there are a few services that require ``global" knowledge of the machine -- e.g., a listing of all compute nodes for the hosts database and queuing system. Rocks uses an SQL database to store the definitions of these global configurations and then generates database reports to create service-specific

configuration files (e.g., DHCP configuration file, /etc/hosts, and nodes file)[2]. The clustering database construction is illustrated in Figure 6.

Once both the software packages and software configuration are installed on a machine, we refer to the machine as an appliance. Rocks clusters often contain Frontend, Compute, and NFS appliances. A simple configuration graph (expressed in XML) allows cluster architects to define new appliance types and take full advantage of code re-use for software installation and configuration.

3.2.2 Viz Roll

Viz roll is used to build tiled display visualization clusters. This Roll creates a new Rocks appliance type called a “Tile”. Just as the “Compute” appliance is used to build computer clusters, this new “Tile” appliance is used to build visualization clusters. The core software that drives the tiled display is called DMX, which creates a single unified X11 desktop from multiple graphics cards, monitors, and computers. Visualization clusters are built to create larger displays (thousands by thousands of pixels), and are primarily of interest to OpenGL applications [5].

We will talk about how to build a scalable visualization cluster by using Viz Roll in detail in Chapter 4.

3.2.3 X window System

X window System is widely used in the UNIX community. In X, a base window system provides high-performance graphics to a hierarchy of resizable windows. Rather than mandate a particular user interface, X provides primitives to support several policies and

styles. Unlike most window systems, the base system in X is defined by a network protocol so the system can be implemented on a variety of displays. The stream-based interprocess communication replaces the traditional procedure call or kernel call interface. An application can utilize window on any display in a network in a device-independent, network-transparent fashion. Interposing a network connection greatly enhances the utility of the window system, without significantly affecting performance. The performance of existing X implementations is comparable to that of contemporary window systems and in general, is limited by display hardware rather than network communication [22].

Figure 7 illustrates the X window system structure in a network environment. The X window system is based on a client-server model. For each physical display, there is a controlling server. A client application and a server communicate over a reliable duplex byte stream. A simple block- stream protocol is layered on top of the byte stream. If the client and server are on the same machine, the stream is typically based on a local interprocess communication (IPC) mechanism; otherwise a network connection is established between the pair. Requiring nothing more than a reliable duplex byte stream (without urgent data) for communication makes X usable in many environments.

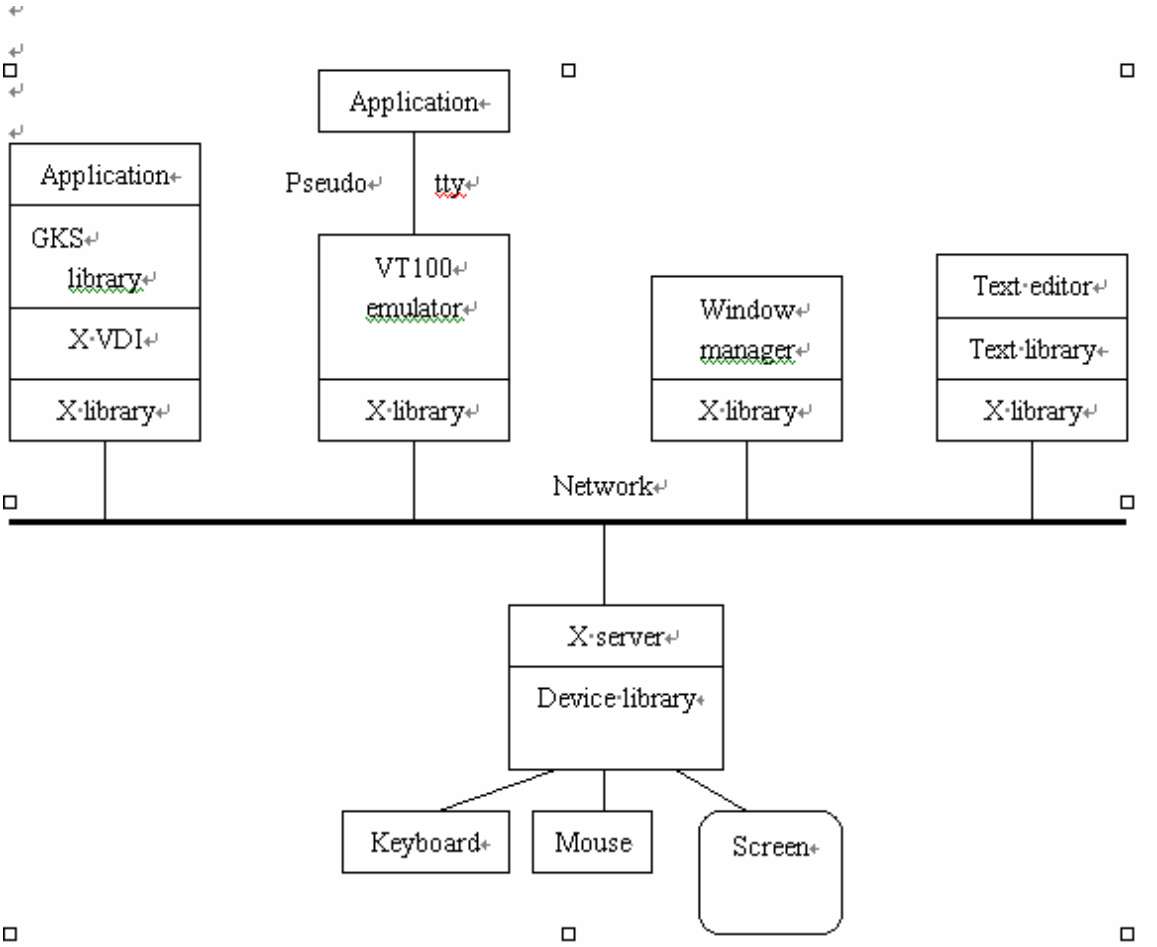

Figure 7 X window system structure

Multiple clients can have connections open to a server simultaneously, and a client can have connections open to multiple servers simultaneously. The essential tasks of the server are to multiplex requests from clients to the display, and demultiplex keyboard and mouse input back to the appropriate clients. Typically, the server is implemented as a single sequential process, using round-robin scheduling among the clients, and this centralized control trivially solves many synchronization problems; however, a multiprocess server has been implemented. Although one might place the server in the kernel of the operating system in an attempt to increase performance, a user-level server

process is vastly easier to debug and maintain, and performance under UNIX in fact does not seem to suffer [22].

The server encapsulates the base window system. It provides the fundamental resources and mechanisms, and the hooks required to implement various user interfaces. All device dependencies are encapsulated by the server; the communication protocol between clients and server is device independent. By placing all device dependencies on one end of a network connection, applications are truly device independent. The addition of a new display type simply requires the addition of a new server implementation; no application changes are required [22].

3.2.4 GNOME Window System

The GNOME window system interface is also used in a Rocks cluster. The GNOME window system provides two things: the GNOME desktop environment, an intuitive and attractive desktop for users, and the GNOME development platform, an extensive framework for building applications that integrate into the rest of the desktop. Like Microsoft window system, GNOME is user friendly compared to Xterm. For example, when you start a desktop session for the first time, you should see a default startup screen, with panels, windows, and various icons [23].

3.3 System Architecture

3.3.1 Cluster Architecture Requirements

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource.

A computer node can be a single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system. A cluster generally refers to two or more computers (nodes) connected together. The nodes can exist in a single cabinet or be physically separated and connected via a LAN. An interconnected (LAN-based) cluster of computers can appear as a single system to users and applications. Such a system can provide a cost-effective way to gain features and benefits (fast and reliable services) that have historically been found only on more expensive proprietary shared memory systems. The typical architecture of a cluster is shown in Figure 8.

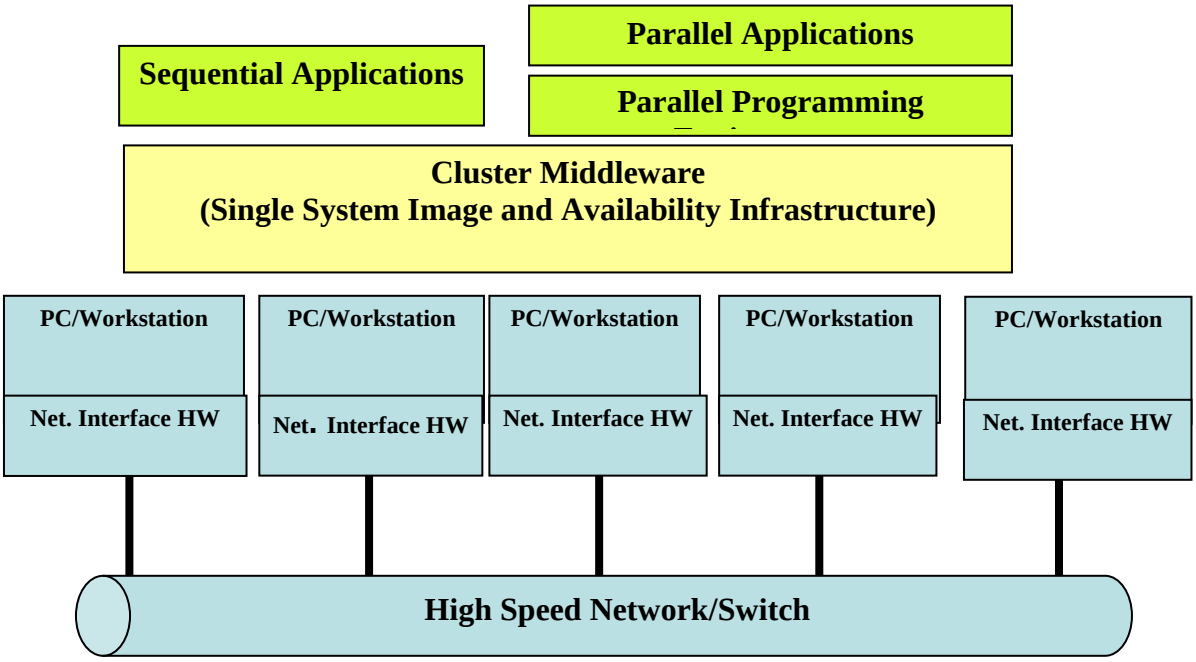

Figure 8 Cluster computer architecture

The main components of cluster computers are as following:

- Multiple High Performance Computers (PCs, Workstations, or SMPs)
- State-of-the-art Operating Systems (Layered or Micro-Kernel based)
- High Performance Networks/Switches (such as Gigabit Ethernet and Myrinet)

- Network Interface Cards (NICs)
- Fast Communication Protocols and Services (such as Active and Fast Messages)
- Cluster Middleware (Single System Image (SSI) and System Availability Infrastructure
  - Hardware (such as Digital (DEC) Memory Channel, hardware DSM, and SMP techniques)
  - Operating System Kernel or Gluing Layer (such as Solaris MC and GLUnix)
  - Applications and Subsystems
    - Applications (such as system management tools and electronic forms)
    - Runtime Systems (such as software DSM and parallel file system)
    - Resource Management and Scheduling software (such as LSF (Load Sharing Facility) and CODINE (Computing in Distributed Networked Environments))
- Parallel Programming Environments and Tools (such as compilers, PVM Parallel Virtual Machine), and MPI (Message Passing Interface))
- Application

The network interface hardware acts as a communication processor and is responsible for transmitting and receiving packets of data between cluster nodes via a network/switch.

Communication software offers a means of fast and reliable data communication among cluster nodes and to the outside world. Often, clusters with a special network /switch like Myrinet use communication protocols such as active messages for fast

communication among its nodes [1]. They potentially bypass the operating system and thus remove the critical communication overheads providing direct user-level access to the network interface.

The cluster nodes can work collectively, as an integrated computing resource, or they can operate as individual computers. The cluster middleware is responsible for offering an illusion of a unified system image (single system image) and availability out of a collection on independent but interconnected computers.

Programming environments should offer portable, efficient, and easy-to-use tools for development of applications. They include message passing libraries, debuggers, and profilers. It should not be forgotten that clusters could be used for the execution of sequential or parallel applications [1].

3.3.2 Actual Architecture

Figure 9 shows a traditional architecture used for high performance computing clusters. This design was pioneered by the Network of Workstations [24], and popularized by the Beowulf project [25]. In this method the cluster is composed of standard high-volume servers, an Ethernet network and an optional off-the-shelf performance interconnect (e.g., Gigabit Ethernet or Myrinet). The Rocks cluster architecture favors high volume components that lend themselves to reliable systems by making failed hardware easy and inexpensive to replace.

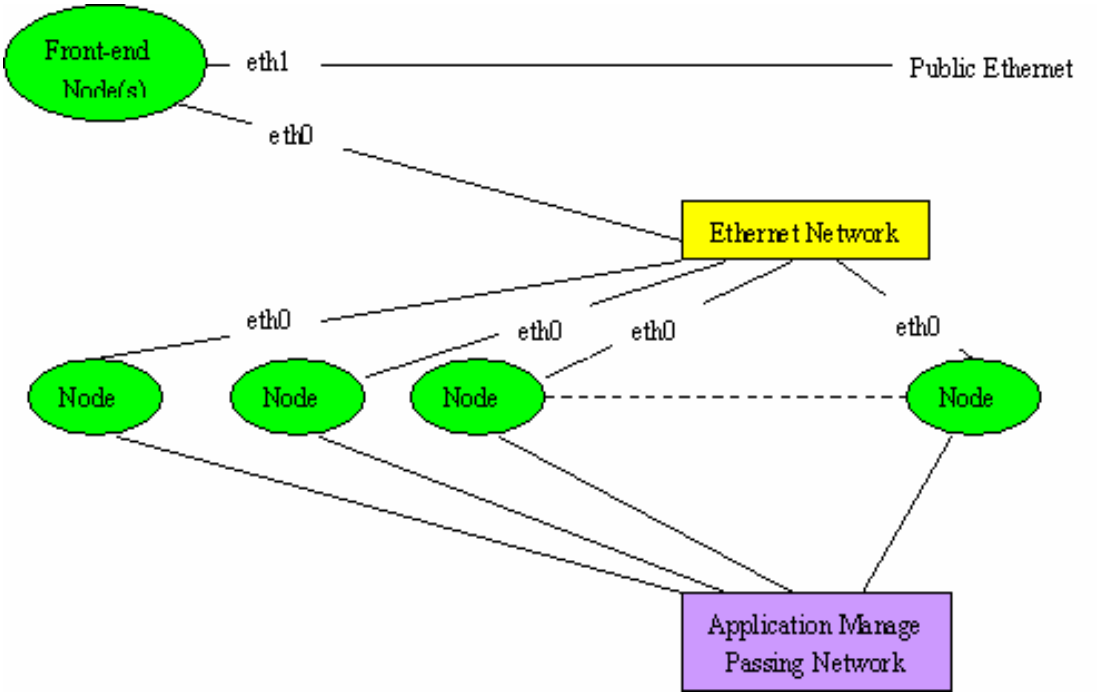

Figure 9 Rocks cluster hardware architecture.

The frontend node acts like a firewall and gateway between the private internal networks and the public internet.

Rocks frontend nodes are installed with the base distribution and any desired rolls. Frontend nodes serve as login and compile hosts for users. Compute nodes typically comprise the rest of the cluster and function as execution nodes. Compute nodes and other cluster appliances receive their software footprint from the frontend. Installation is a strong suit of Rocks; a single frontend on modern hardware can install over 100 compute nodes in parallel, a process taking only several minutes. Typically cluster nodes install automatically, using PXE to obtain a boot kernel from the frontend [26].

3. 4 Viz Components

The major software components included in the Viz Roll are: Chromium, CMake, Doom Legacy, DMX, GLUT, NCSA Pixel Blaster, SDL, wxGTK.

Xdmx is proxy X server that provides multi-head support for multiple displays attached to different machines (each of which is running a typical X server). When Xinerama is used with Xdmx, the multiple displays on multiple machines are presented to the user as a single unified screen.

Chromium is a system for interactive rendering on clusters of graphic workstations. Various parallel rendering techniques such as sort-first and sort-last may be implemented with Chromium. Furthermore, Chromium allows filtering and manipulation of OpenGL command streams for non-invasive rendering algorithms.

3.4.1 XDMX

The most important software component is XDMX, which together with xinerama is responsible for sorting and transporting the information and rendering on the nodes. We will discuss XDMX and xinerama in detail.

Current Open Source multihead solutions are limited to a single physical machine. A single X server controls multiple display devices, which can be arranged as independent heads or unified into a single desktop. These solutions are limited to the number of physical devices that can co-exist in a single machine. Thus, large tiled displays are not currently possible. This limitation will be solved by eliminating the requirement that the display devices reside in the same physical machine. This will be accomplished by developing a front-end proxy X server that will control multiple back-end X servers that make up the large display. These X servers can either run on the same or separate machines [4].

Typical X servers provide multi-head support for multiple displays attached to the same machine. When Xinerama is in use, these multiple displays are presented to the user as a single unified screen.

A simple application for Xdmx would be to provide multi-head support using two desktop machines, each of which has a single display device attached to it. A complex application for Xdmx would be to unify a 4 by 4 grid of 1280x1024 displays (each attached to one of 16 computers) into a unified 5120x4096 display.

Xdmx was developed and run under Linux (ia32 and x86_64) and has been tested with SGI Irix. The overall structure of the distributed multihead X (DMX) project is as follows: A single front-end X server will act as a proxy to a set of back-end X servers, which handle all of the visible rendering. X clients will connect to the front-end server just as they normally would to a regular X server. To the client, everything appears as if they have a single large display; however, using an extension they can request the arrangement of the back-end server screens as well as other information that might be useful to them (e.g., for placement of pop-up windows, window alignments by the window manager, etc.). A configuration tool is proposed that will use this extension and will aid setup and arrangement of the DMX system [4].

- Xinerama

Xinerama, supported by Xdmx, is an X extension that allows multiple physical screens controlled by a single X server to appear as a single screen. Although the extension allows clients to find the physical screen layout via extension requests, it is completely transparent to clients at the core X11 protocol level [4].

The current implementation of Xinerama is based primarily in the DIX (device independent) and MI (machine independent) layers of the X server. With few exceptions the DDX (Device dependent) layers do not need any changes to support Xinerama. X server extensions often do need modifications to provide full Xinerama functionality [4].

3.4.2 Chromium

Chromium is a system for interactive rendering on clusters of graphics workstations. Various parallel rendering techniques such as sort-first and sort-last may be implemented with Chromium. Furthermore, Chromium allows filtering and manipulation of OpenGL command streams for non-invasive rendering algorithms [29].

The main Chromium's features are:

1). Sort-first (tiled) rendering - the frame buffer is subdivided into rectangular tiles which may be rendered in parallel by the hosts of a rendering cluster.

2). Sort-last (Z-compositing) rendering - the 3D dataset is broken into N parts which are rendered in parallel by N processors. The resulting images are composited together according to their Z buffers to form the final image.

3). Hybrid parallel rendering - sort-first and sort-last rendering may be combined into a hybrid configuration.

4). OpenGL command stream filtering - OpenGL command streams may be intercepted and modified by a stream processing unit (SPU) to implement non-photorealistic rendering (NPR) effects, etc.

5). Many OpenGL programs can be used with Chromium without modification. One can write Chromium-specific applications which perform parallel rendering with the aid of special synchronization primitives.

6). Chromium runs on Linux, IRIX, AIX, SunOS and Windows-based systems.

7). Chromium is an open-source project.

3.5 Algorithm

Graphics usually happens in a pipeline that can perform operations in parallel. The task is performed in two major stages: transformation and rasterization. The former converts the model coordinates of each object primitive into screen coordinates, while the later converts the resulting geometry information for each primitive into a set of shaded pixels.

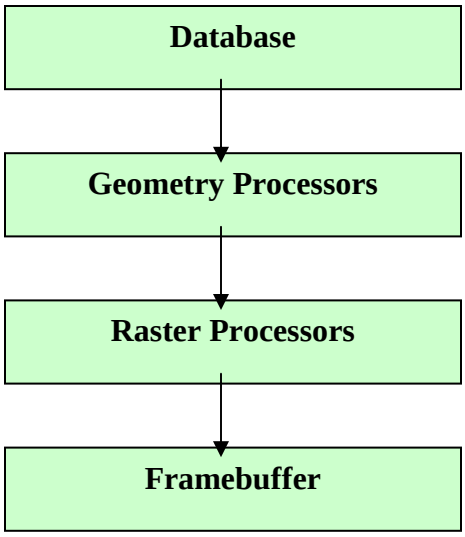

Figure 10 Rendering pipeline.

So, how do we allocate a scene graph to the graphics processors? The graphics primitives need to be assigned to GPU (Graphic Processing Units); so where in the pipeline above dose this occur?

The choices for how to partition and recombine the parallelized work at the different pipeline stages lead to 3 basic approaches: sort-first, sort-middle, and sort-last [30].

- Sort-First

The rendering system is integrated in Xdmx that can deliver high polygon performance with high-resolution displays in an efficient manner. This algorithm is called “sort-first”.

In sort-first, each processor is assigned a portion of the screen to render. First, the processors examine their primitives and classify them according to their positions on the screen. This is an initial transformation step to decide to which processors the primitives actually belong, typically based upon which regions a primitive’s bounding box overlaps. During classification, the processors redistribute the primitives such that they all receive all of the primitives that fall in their respective portions of the screen. The results of this redistribution form the initial distribution for the next frame [31].

Following classification, each processor performs the remaining transformation and rasterization steps for all of its resulting primitives. Finished pixels are sent to one or more frame buffers to be displayed.

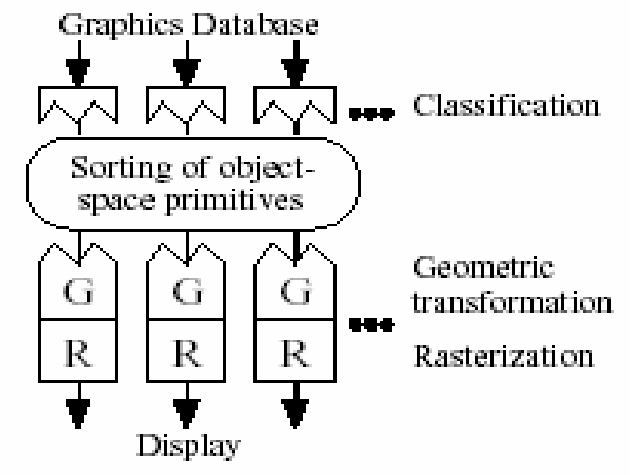

Figure 11 Sort-first pipeline.

In sort-first, once a processor has the correct set of primitives to render, only that processor is responsible for computing the final image for its portion of the screen. This

allows great flexibility in terms of the rendering algorithms which may be used. All the speed-ups which have been developed over time for serial renderers may be applied here. Since only finished pixels need to be sent to the frame-buffer, sort-first can easily handle very-high-resolution displays. Also sort-first is the only architecture of the three that is ready to handle large databases and large displays [30].

However, sort-first is not without its share of problems. Load balancing is perhaps one of the biggest concerns: because the on-screen distribution of primitives may be highly variable, some thought must go into how the processors are assigned screen regions. Also, managing a set of migrating primitives is a complex task [31].

- Sort-Last

Chromium, another major software component, uses sort-last to render the graphics. For sort-last, each processor has a complete rendering pipeline and produces an incomplete full-area image by transforming and rasterizing its fraction of the primitives. These partial images are composited together, typically by depth sorting each pixel, in order to yield a complete image for the frame buffer. The composition step requires that pixel information (at least color and depth values) from each processor be sent across a network and sorted along the way to the frame buffer.

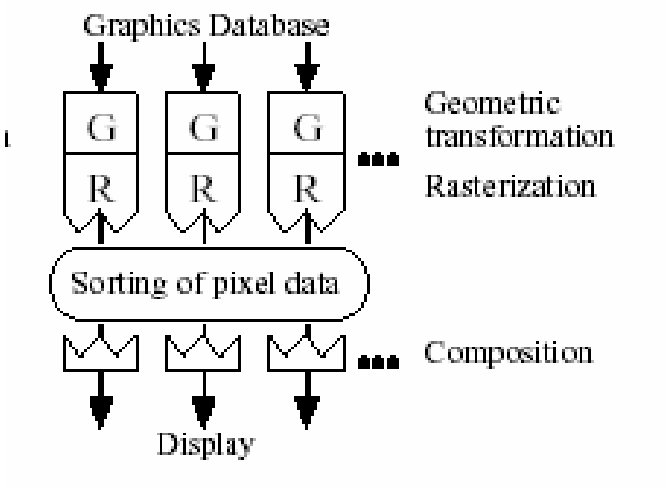

Figure 12 Sort-last pipeline

Sort-last offers excellent scalability in terms of the number of primitives it can handle. However, its pixel budget is limited by the bandwidth available at the composition stage. Using a specialized composition network can help to overcome this problem [32].

4. SYSTEM IMPLEMENTATION

This chapter describes the Rocks-Viz implementation where includes the system hardware/software installation, configuration and example program.

4.1 Installation and Setup

The prototype cluster of processors includes a frontend and two backends. The architecture is as in the Figure 13.

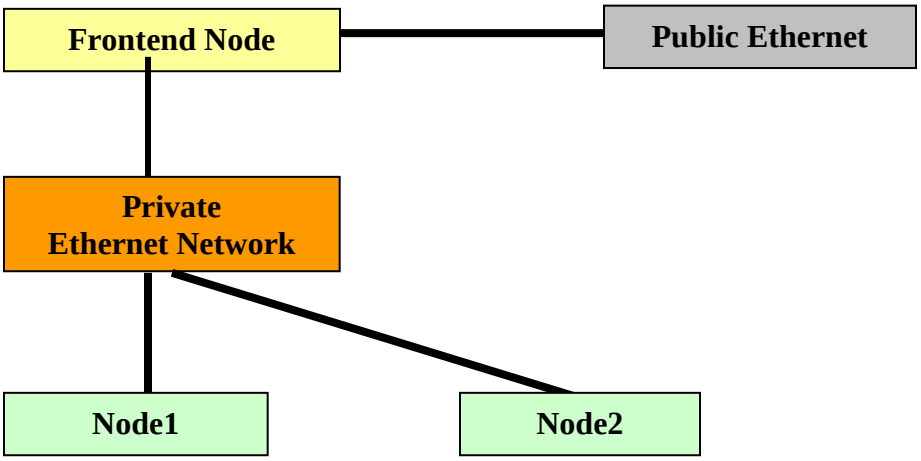

Figure 13 Architecture of Viz cluster

These three PCs have the same type which is dual Pentium 4 server running Rocks 4.1 software with the Red Hat Linux operating system. The frontend node can function as the central server, while the two tiles work as the slaves. The frontend and backends are located on the same local area network (LAN) via copper-based Gigabit Ethernet capable of 100mbps. The frontend requires another network card if it is to connect to a WAN in the same organizational domain.

The command –ifconfig can be used to check if ethernet connections are OK. The result should show two ethernet cards, eth0 and eth1.

After the cluster is physically assembled, we must define a complex software stack for all nodes and then physically install them. Commonly, the frontend should be installed first. Compute and other types of nodes are then installed using instructions or images housed on the frontend.

To install Rocks and the Viz Roll, the steps are as follows:

1). Turn on the frontend PC, insert the Boot CD into the CDROM, type in "frontend" after Figure 14 is displayed.

Figure 14 Boot Roll

2). Anaconda starts and asks for rolls to insert. Click "yes" after Figure 15 is displayed. Insert 4 operating system CDs and one Viz Roll CD. These CDs includes base, hpc, kernel, etc.

Figure 15 The Rocks installation ask for Rolls

3). Fill in the Cluster information such as cluster host name, SSL/HTTP for certificates, etc.

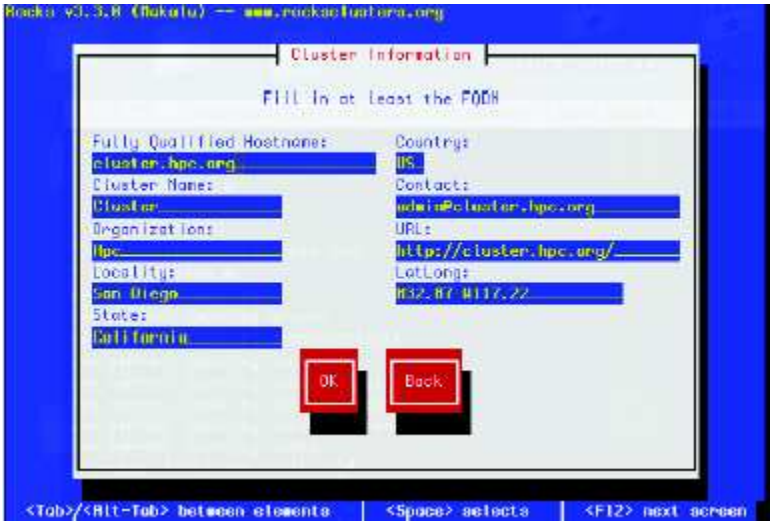

Figure 16 Fill in the cluster information

4). Disk partitioning. This task can be done by both automatic partition and manual partition.

5). Network Configuration. There should be two ether cards inside. Ethe0 is for cluster-side only, while the ethe1 for the Internet/LAN.

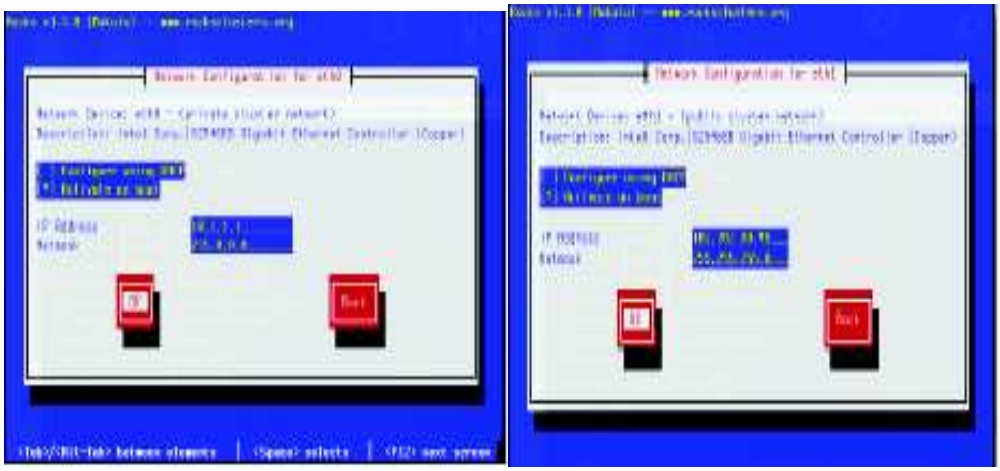

Figure 17 Network configuration during installation

6). Fill in the Gateway/DNS (Domain Name Server) information. Note that all traffic for compute nodes is NATed through the frontend. DNS is only for the frontend, compute nodes use the frontend as their DNS.

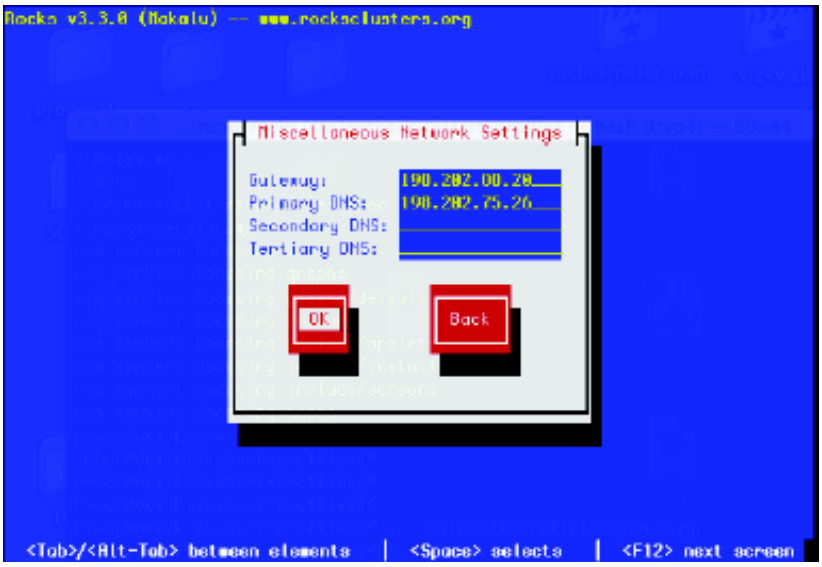

Figure 18 Filling in the Gateway/DNS information

7). After configure the network time protocol, we need to type in the Root password. All information goes into the cluster database as illustrated in Figure 19.

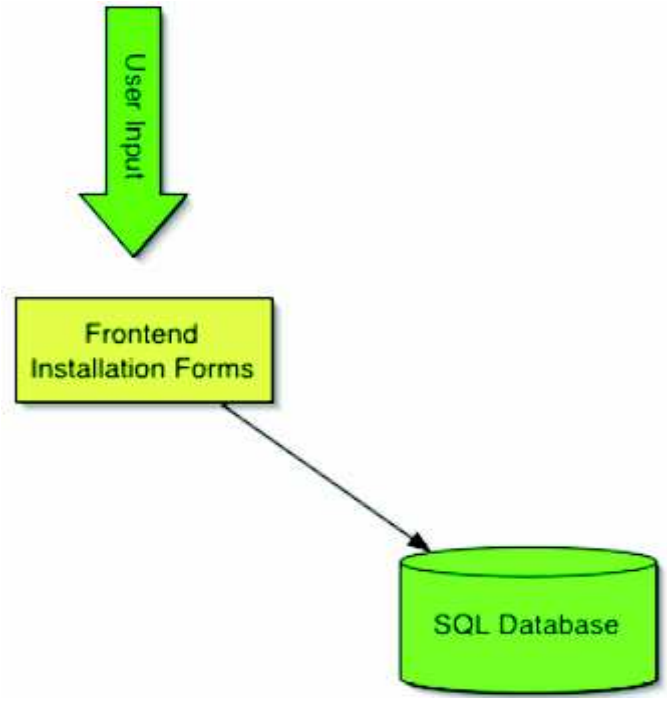

Figure 19 The information are saved in the cluster database

8). The automatic installation begins, and the packages of Rocks will be merged together.

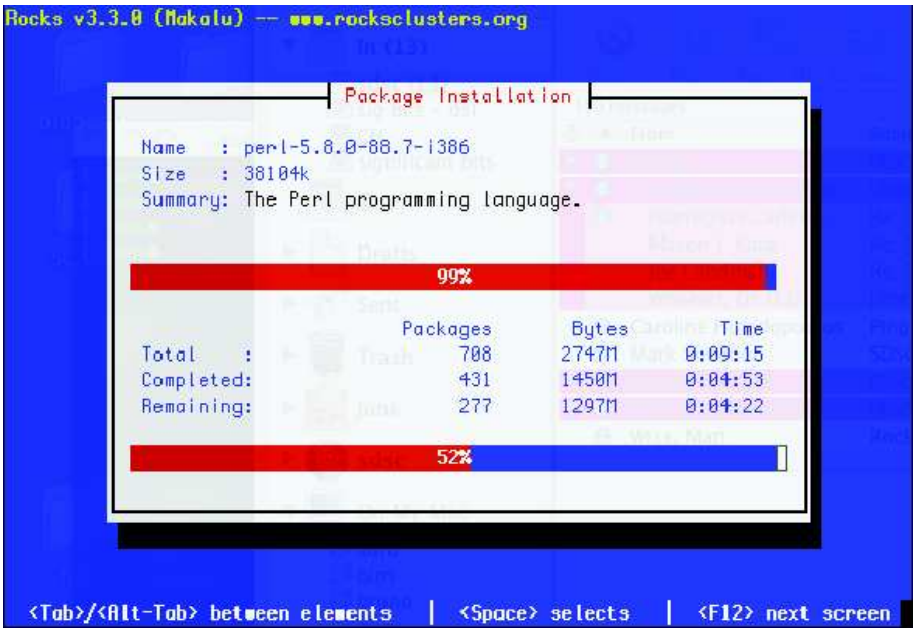

Figure 20 Rocks automatic installation.

9). Once the frontend installation has completed, the node will reboot and X11 will start in the default 800 x 600 screen resolution. Now we need to configure the video card, first open a new Xterm window (right mouse click menu) and the shell prompt type: system-config-display. Then this screen will prompt you to configure your video card, and we will see the following screen:

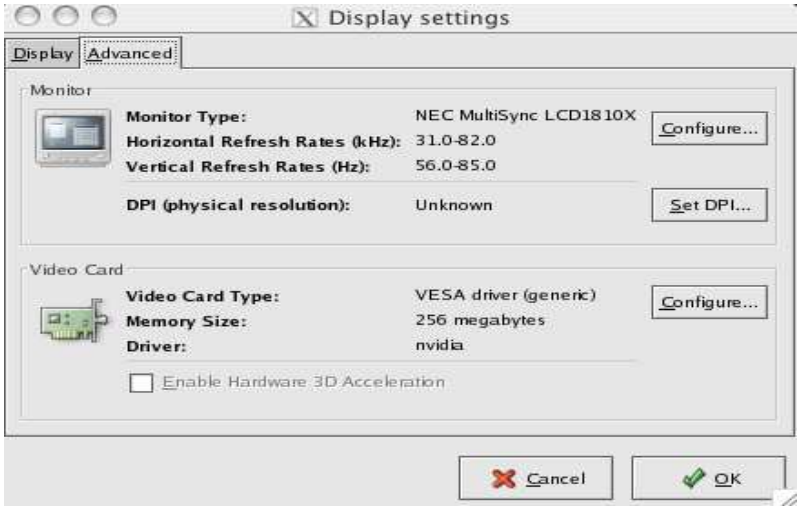

Figure 21 Node rebooting and X11 installation

Then click on the "Configure" button for the Monitor section to select the appropriate LCD (tile) model number.

10). Rocks numbers freshly installed nodes in the form <name>-<rack>-<rank>, where "name" is the appliance name (e.g. "compute" or "tile") and "rack" and "rank" correspond to the physical location of the machine in its rack.

The Viz Roll interprets this notion of rack and rank differently. Instead of referring to the location of the host, rack and rank specify the location of the individual display

attached to the nodes. For example, to build a small 3x3 display wall, insert-ethers will need to be run three times, each time discovering 3 hosts.

To start the software installation process, on the frontend, execute: #insert-ethers

This brings up a screen that looks like Figure 22.

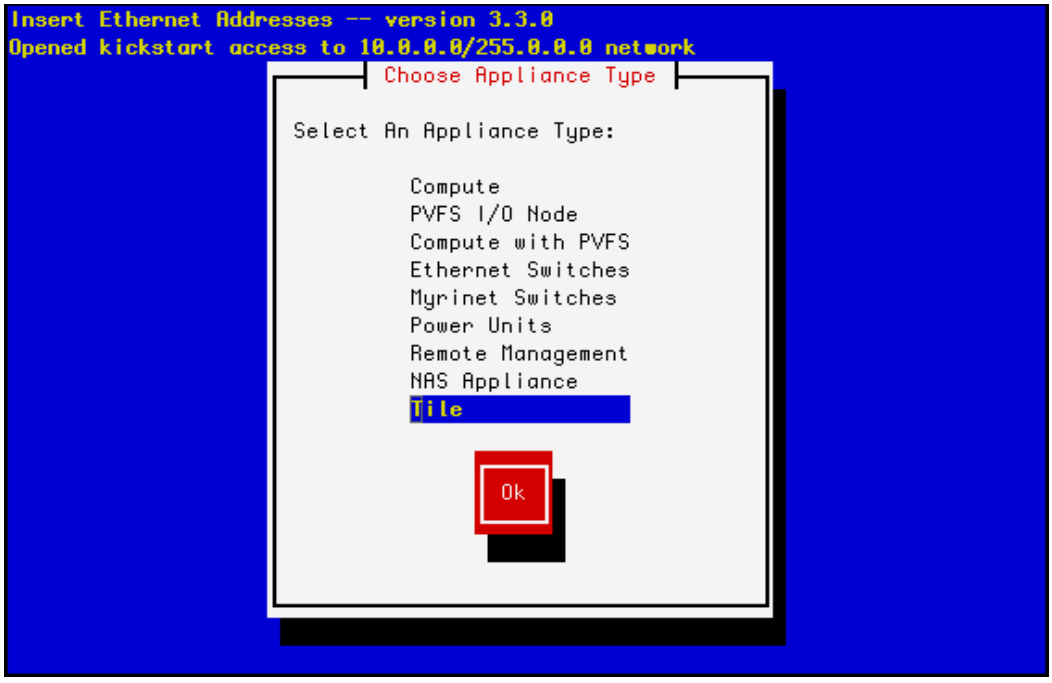

Figure 22 #Insert-ethers execution

Select "Tile" as the appliance type and hit "Ok". Insert-ethers is now waiting for the first tile node to ask for its software.

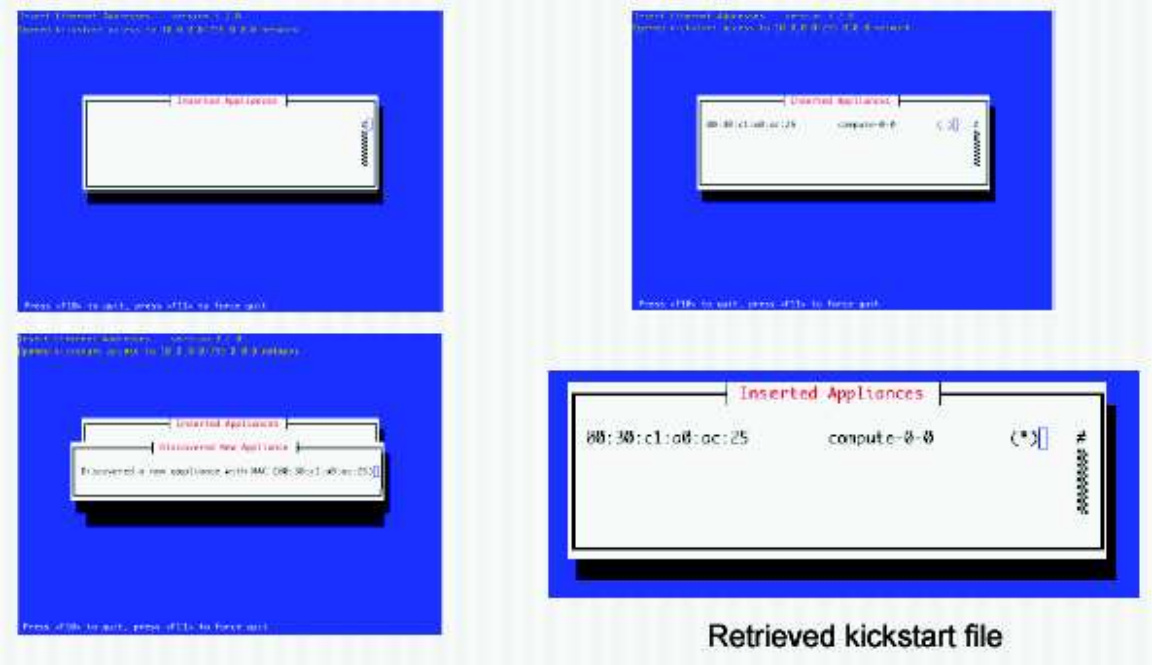

Figure 23 The frontend is discovering the tile-0-0.

We have only two tiles in this experiment, so we need the second tile to be in the same row/column as the first one. Tile-0-1 means the second node is as the same column as the tile-0-0, while the tile-1-0 means the second node is as the same row as the tile-0-0.

In our experiment, we choose the horizontal fashion which means the two tiles are in different columns. To do this, first quit insert-ethers by hitting "F10". Then restart insert-ethers, but this time tell it label the tile nodes with cabinet id "1": # insert-ethers –carbinet=1

Figure 24 The two installed tiles.

As shown in Figure 24, the two tiles are installed. After the tile node installs its software, it will reboot and display its hostname in the top left corner. The tile-0-0 and tile-1-0 are in one row and can render the image as one display.

4.2 System Configuration

Before running the application, we need to do some configuration for the system.

1). Enter in frontend, click the right button of the mouse, and choose to open a Xterm.

2). Change to the directory: cd /etc/selinux. Then find the configuration file of selinux.

3). Edit the selinux: remove the “#” of the sentence: # SELINUX = disabled.

The purpose of doing this is to confirm the connection between the frontend and two backends. Selinux is the secure mechanism of Linux operating system, which can improve the secure level to the B level that equals to national secure level. In the cluster, too high level may effect the connections between information transmissions.

Now please use command –ping tile-0-0 or –ping tile-1-0 on the frontend. If result shows there is transmission between the frontend and backend, then we can ignore the following part. Otherwise, it means the frontend can only recognize the tile-0-0 and tile-1-0’s IP addresses other than their names. The DNS configuration can be done by following steps:

1). Go to directory: cd /etc/hosts

2). Edit the configuration file:

Eg. 10.255.255.253 tile-0-0

10.255.255.254 tile-1-0

Notice that the IP address should be written before the tile’s name.

With this modification, the tile name and its IP are associated and be easily recognize by the frontend.

3). Now check if it works: -ping tile-0-0

Another Configuration is for the screen bezels. The display should be calibrated to adjust the desktop image. That is, the desktop looks like you are looking though a window pane. For most applications this is the preferred setup, however it is not the default. To calibrate a display wall, we need to know the width and height (in pixels) of the PC's bezels. Then we need to edit the .Xclients file in the home directory and uncomment the last line:

#BEZELARGS = "125 100"

This will give us a reasonable starting point (and might be perfect) for the monitors. To test these settings, just logout by exiting the window manager on the the frontend display by using right mouse button. Now login again on the frontend and repeat this and modify the BEZELARGS until we are satisfied with the result.

4.3 Implementation

4.3.1 Start the Viz Clutster by XDMX

To start using your visualization cluster, login with your user account at the X11 graphical console on the frontend. XDMX will automatically start on all of the tiled displays. This gives you a single large desktop with the Motif Window Manager (mwm). To open an Xterm, use the right mouse button and select New Window.

The following demos are provided to demonstrate that the system is working correctly:

1) To run the demo OpenGL program on the frontend, change the demo directory with the command: cd /opt/viz/demos/rollercoaster.

2) Then run it with command: ./roller

3) Now render this image on the tiles. Using command:

./roller Xdmx :1 –display tile-0-0:0 -display tile-1-0:0 input :0

The –display option specifies the name(s) of the back-end X server display(s) to connect to. This option may be specified multiple times to connect to more than one back-end display. The first is used as screen 0, the second as screen 1, etc. If this option is omitted, the $DISPLAY environment variable is used as the single back-end X server display.

The -input option specifies the source to use for the core input devices.

4) We can see the demo image is only rendered on one of these tiles. The reason is this graphic has too low resolution. So in the next step, we will try to render a high-resolution image on both tiles.

4.3.2 Rendering High-Resolution Image on Both Tiles

1) Create a file and open it with VI, then write an OpenGL program in it. For example we name it Teapots.

2) We need to compile and run this program on the frontend first to make sure this program is right.

Compile command is: cc Teapots –o Teapots1 –lgut

Then run it: /Teapots1.

Notice that the Teapots1 is the compiled version.

3) Now it is time to run this OpenGL program on both tiles. The command is:

#Xdmx :1 +xinerama –display tile-1-0:0 -display tile-0-0:0 -input localhost:0 –ac –br –ingorebadfontpaths –norender

As we discussed before, the Xdmx functions as a front-end X server that acts as a proxy to a set of back-end X servers. All of the visible rendering is passed to the back-end X servers. Clients connect to the Xdmx front-end, and everything appears as it would in a regular multi-head configuration. If Xinerama is enabled (e.g., with +xinerama on the command line), the clients act as a single large screen.

The syntax and semantics of the +xinerama are as follows: Assume you are running Xdmx on a machine called x0 and you have two other machines available, x1 and x2, both of which have X servers running on them (i.e., you have logged into the console on x1 and x2 and are running an X session). To start Xdmx, use the following command:

Xdmx :1 +xinerama -display x1:0 -display x2:0

There should now be an X server running on x0 with the DISPLAY name ":1" (or, "x0:1"). The displays on x1 and x2 provide a unified input display, and the mouse on x1 should be able to move the cursor from x1 to x2 and back again. You can now set up applications to run on ":1", with the output being displayed on the x1 and x2 displays.

The font path used by the Xdmx front-end server will be propagated to each back-end server,which requires that each back-end server have access to the exact same font paths as the front-end server. This can be most easily handled by either using a font server (e.g., xfs) or by remotely mounting the font paths on each back-end server, and then setting the Xdmx server's default font path with the -I "-fontpath" command line option described above.

For example, if you specify a font path with the following command line:

Xdmx :1 -display d0:0 -fontpath /usr/fonts/75dpi/ -fontpath /usr/fonts/Type1/ +xinerama Then, /usr/fonts/75dpi/ and /usr/fonts/Type1/ must be valid font paths on the Xdmx server and all back-end servers.

The –ignorebadfontpaths option ignores font paths that are not available on all back-end servers by removing the bad font path(s) from the default font path list. If no valid font paths are left after removing the bad paths, an error to that effect is printed in the log.

The –norender option disables the RENDER extension.

As shown in Figure 25, the Xinerama is the flag of the command line and creates a single window in the frontend to simulate the two tiles' screens. And the two tile's screen becomes one black display. The cursor can move freely on both 2 tiles. The cursor is controlled by the frontend's mouse.

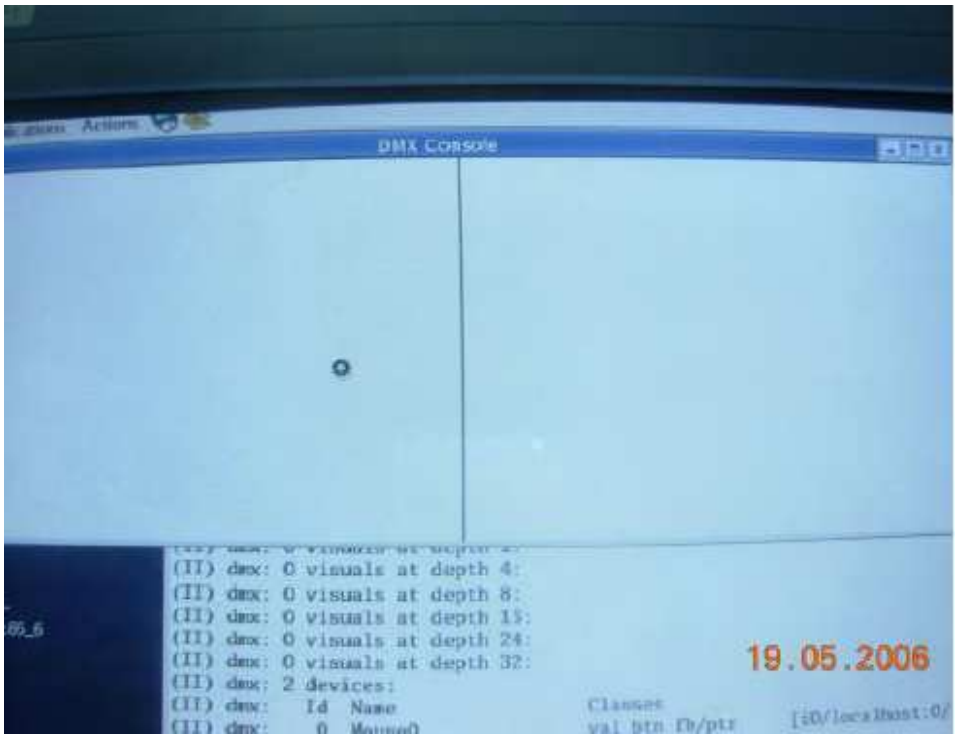

Figure 25 A single window in the frontend to simulate the two tiles' screens.

Figure 26 The combined nodes' screens to a large black screen.

Figure 26 shows that the nodes' screens are combined to a large black screen. Using the frontend's mouse, one can move the cursor from one screen to another. Also one can open a Xterm window in one of this screens by clicking the right button of the mouse.

4). Now open another Xterm in the frontend, and set the environment viarable as shown in Figure 27. This is for avoiding the bugs in Ganglia. Ganglia is a toolkit running on each cluster node and gathers values for various metrics such as CPU load, free memory, disk usage, network I/O, operating system version, etc. When a number of heartbeats from any node are missed, this web page will declare it "dead". These dead nodes often have problems which require additional attention, and are marked with the Skull-and-Crossbones icon, or a red background.

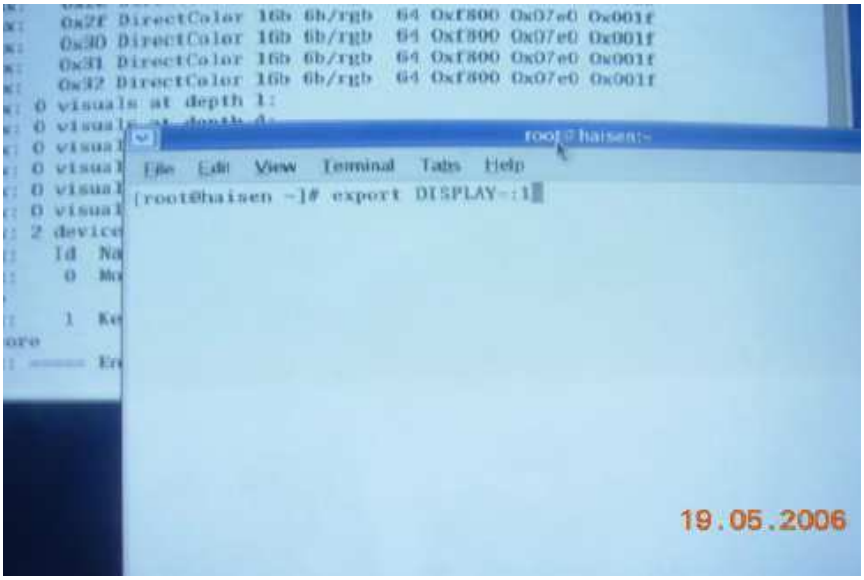

Figure 27 Setting the environment variables in the frontend.

5). Now compile and run “Teapot” again in this Xterm, the image should be rendered on both tiles. The result is showed in Figure 28.

Figure 28 The OpenGL program is rendered on both screens.

4.3.3 Running Another OpenGL Program On the Same Display

We will run another OpenGL program and see if these graphics can overlaps. Open one Xterm in one node by clicking the mouse in the control window on the frontend. Open a Xterm Window in one of the nodes as shown in Figure 28.

Figure 29 A Xterm Window opened in one of the nodes.

Then run the demo program roller. Two programs are overlapped on both tiles and can be moved by the mouse of frontend. Both graphics are moving as shown in Figure 29.

Figure 30 Moving graphics in both tiles

4.3.4 Accelerating OpenGL Applications (Chromium)

For small display walls, the XDMX should meet all of our OpenGL application needs. But for large walls, Chromium can be used to speed up the graphics. Chromium is a replacement OpenGL library that is used to optimize OpenGL code for tiled displays.

To start any OpenGL application with Chromium, just follow these steps below:

1) First set the fontpath:

export LD_LIBRARY_PATH = /opt/chromium/lib/Linux

#export LD_LIBRARY_PATH = /opt/wx/lib

#export PYTHONPATH = usr/lib/python2.2/site-packages

2) Running the OpenGL program as same as in Xdmx.

5. CONCLUSIONS

In order to build a high-performance scalable visualization clusters to support more realistic and flexible applications, we introduced the Viz Roll which is based on Rocks cluster toolkit. Unlike most display wall systems today, which are built with high-end graphics machines and high-end projectors, this system is built with low-cost commodity components: a cluster of PCs, PC graphics accelerators, and a network. To achieve the effect of immersive sound, we also can add the consumer video and sound equipment with the PCs.

The advantages of this approach are low cost and technology tracking, as high-volume commodity components typically have better price and performance ratios and improve at faster rates than special-purpose hardware. The challenge is to use commodity components to construct a high-quality collaborative environment that delivers display, rendering, and input to compete with the custom-designed, high-end graphics machine approach.

The Rocks toolkit can be used to build a scalable display computer system. It also can be maintained and configured easily. It uses a graph-based framework to describe the configuration of all node type (appliances) that make up a complete cluster. The appliance types except for the initial frontend can be modified and extended by the Rolls. Roll is a software pack that provides both the architecture and mechanisms that enable the end-user to incrementally and programmatically modify the graph description for all

appliance types. The functionality of Rolls is added or overwritten by inserting the desired Roll CD at installation time. Rolls are optional, automatically configured, cluster-aware software systems. Current Rolls include: Grid (based on NSF Middleware Initiative), Database Support (DB2), Viz, scheduling systems (SGE, PBS), Condor, Integrity Checking (Tripwire) and the Intel Complier.

The Viz Roll was used to build a scalable tiled display visualization cluster. Visualization clusters are built to create larger displays, and are primarily of interest to OpenGL applications.

The X window system was used as the user interface in this visualization cluster.

Xdmx was used as the rendering proxy in this cluster. The single forntend X server acts as a proxy to a set of backends, which handle all of the visible rendering. X clients connect to the frontend server just as they normally would to a regular X server. The clients appear as a single large display. The Xinerama extension combines multiple displays into one large virtual desktop. Some configurations is needed to satisfy different Linux System.

Sort-first and sort-last parallel rendering methods are used to minimize communication requirements and balance the rendering load across a cluster of PCs. Sort-first can send only a subset of primitives to each tile, and redistribute only when the primitive crosses a boundary to another tile, while Sort-last generate an image without regard to where it appears on the screen in each geometry processor/rasterizer. In this cluster, Sort-first is adapted by Xdmx. We also can use Chromium, another software component of Viz, to render the graphics by Sort-last algorithm.

We also addressed the architecture of the cluster and its implementation. Future improvements of distributed composition service have already been proposed in the following directions:

- Sound, should be integrated with the images
- The clustering algorithm needs to be simulated in a more practical environment. We need to design experiments to evaluate more aspects of our clustering mechanism
- Solve the problem that low-resolution image can only be rendered in one of the backends

REFERENCES

[1] Buyya, R. (1999). *High Performance Cluster Computing. Volume 1: Architecture and Systems.* Prentice Hall PTR, NJ, USA.

[2] Rocks Cluster Distribution: Users Guide. (2006). http://www.rocksclusters.org/ rocks-documentation/4.1/ (accessed on Jun. 14, 2006)

[3] Bruno, G., Katz, M. J., Sacerdoti, F. D., and Papadopoulos, P. M. (2004). "Rolls: Modifying a Standard System Installer to Support User-customizable Cluster Frontend Appliances." *Cluster Computing, 2004 IEEE International Conference on,* pp. 421-430.

[4] XmX - An X Protocol Multiplexor. http://www.cs.brown.edu/software/xmx (accessed on Jun. 2nd, 2006)

[5] Viz Roll: Users Guide (2006). http://www.rocksclusters.org/roll-documentation/viz/4.1/ roll-viz-usersguide.Pdf (accessed on Jun. 14, 2006)

[6] TeraGrid http:// www.teragrid.org (accessed on Jun 10th, 2006)

[7] Chen, H., Chen, Y., Finkelstein, A., Funkhouser, T., Li, K., Liu, Z., Samanta, R., and Wallace, G.. (2001). "Data Distribution Strategies for High-resolution Displays" *Computers & Graphics,* 25(5), pp. 811-818.

[8] Chen, Y., Chen, H., Clark, D. W., Liu, Z., Wallace, G., and Li, K. (2001). "Software Environments for Cluster-based Display Systems" In *proceeding of 1st International Symposium on Cluster Computing and the Grid.* pp. 202.

[9] Mayer, T. (1997). “New Options and Considerations for Creating Enhanced Viewing Experiences.” *ACM SIGGRAPH Computer Graphics, 31(2),* pp. 32-34.

[10] Maxwell, D. B., Bryden, A., Schmidt, G.. S., Roth, I., and Swan, E. II. (2002). “Integration of a Commodity Cluster into an Existing 4-Wall Display System.” In *Proceedings of Workshop on Commodity-Based Visualization Clusters ,IEEE Visualization*.

[11] Power Wall. (2006) http://www.lcse.umn.edu/research/powerwall/powerwall.html (accessed on May. 30, 2006).

[12] infinite Wall: http://access.ncsa.uiuc.edu/Stories/97Stories/IWall.html (accessed on May 30, 2006)

[13] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. (1993). “Surround-screen Projection-based Virtual Reality: the Design and Implementation of the CAVE.” In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques.* pp. 135-142.

[14] Humphreys, G., and Hanrahan, P. (1999). “A Distributed Graphics System for Large Tiled Displays.” In *Proceedings of the conference on Visualization '99: celebrating ten years.* pp. 215-223.

[15] Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., and Fuchs, H. (1998). “The Office of the Future: a Unified Approach to Image-based Modeling and Spatially Immersive Displays.” In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* pp. 179-188.

[16] Igehy, H., Stoll, G., and Hanrahan, P. (1998). "The Design of a Parallel Graphics Interface." In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques***.** pp. 141-150**.**

[17] Montrym, J. S., Baum, D. R., Dignam, D. L., and Migdal, C. J. (1997). "InfiniteReality: a Real-time Graphics System." In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. pp. 293-302.

[18] DEXON Systems Ltd. http://www.dexonsystems.com/jdxvir.html. (accessed on Jun. 2nd, 2006)

[19] Humphreys, G., Buck, I., Eldridge, M., and Hanrahan, P. (2000). "Distributed rendering for scalable displays" In *Proceedings of Supercomputing*. pp. 30.

[20] Overview of Rocks http://www.rocksclusters.org/rocksapalooza/2006/tutorial-session2. pdf (accessed on Jun.2nd, 2006)

[21] Katz, M. J., Papadopoulos, P. M., and Bruno, G. (2002). ""Leveraging standard core technologies to programmatically build Linux cluster appliances." *Cluster Computing. Proceedings. 2002 IEEE International Conference on.* pp. 47-53

[22] Scheifler, R. W., and Gettys, J. (1986). "The X Window System". *ACM Transactions on Graphics (TOG)*, 5(2), pp. 79-109.

[23] GNOME. http://www.gnome.org/ (accessed on Jun. 2nd, 2006)

[24] Anderson, T., Culler, D., and Patterson, D. (1995). "A case for NOW (Networks of Workstations)." *IEEE Micro.* 15(1), pp. 54-64.

[25] Sterling, T., and Savarese, D. (1995). "BEOWULF: A parallel workstation for scientific computation." In *24th International conference on parallel processing.* Oconomowoc, WI.

[26] Sacerdoti, F. D., Chandra, S., and Bhatia, K. (2004). “Grid systems deployment & management using rocks.” *IEEE Cluster*. pp.337-345.

[27] OpenGL Architecture Review Board. (1993). *OpenGL Reference Manual: the Official Reference Document for OpenGL, Release 1*. Addison-Wesley.

[28] Distributed Multihead X Project. http://dmx.sourceforge.net/(accessed on Jun. 2nd, 2006)

[29] Chromium http://chromium.sourceforge.net/ (accessed on Jun 4th, 2006)

[30] Cox, M., Molnar, S., Ellsworth, D., and Fuchs, H. (1994). “A sorting classification of parallel rendening.” *IEEE Computer Graphics and Algorithms*. pp. 23-32.

[31] Mueller, C. (1995). “The sort-first rendering architecture for high-performance graphics”. In Proceedings of the 1995 symposium on Interactive 3D graphics. pp. 75-85.

[32] Samanta, R., Funkhouser, T., Li, K., and Singh, J. P. (2000) “Hybrid sort-first and sort-last parallel rendering with a cluster of PCs” *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware.* pp. 97-108.