Petteri Kontio

Cluster Rendering with Commodity Hardware in the Experimental Virtual Environment

Teknillinen korkeakoulu
Tietotekniikan osasto

Helsinki University of Technology
Department of Computer Science
and Engineering

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Petteri Kontio	
Name of the thesis:	Cluster Rendering with Commodity Hardware in the Experimental Virtual Environment	
Date:	February 10, 2005	**Number of pages:** 85
Department:	Department of Computer Science and Engineering	**Professorship:** T-111
Supervisor:	Professor Lauri Savioja	
Instructor:	M.Sc. (Tech.) Seppo Äyräväinen	

The purpose of this thesis was to examine the different alternatives for replacing the rendering equipment in the Experimental Virtual Environment (EVE) at Helsinki University of Technology (HUT). The work was divided into three parts, starting with the analysis of different hardware alternatives, then building the chosen system and finally choosing and evaluating the most suitable software architecture for the system. A cluster of commodity hardware PCs using software-genlocked active stereo was chosen as the basis of the new system.

In consequence, the work concentrated on setting up a cluster of commodity hardware PCs and installing that system to the virtual environment. The video signals from the rendering PCs were synchronized using Open Source software (SoftGenLock) and the polygon data was transferred through a Gigabit Ethernet network with a particular software library that was developed for this purpose, namely Broadcast GL (BGL).

Keywords: Cluster rendering, Virtual Reality, Commodity hardware, Active Stereo

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Petteri Kontio	
Työn nimi:	PC-pohjaisten klusterirenderöintitekniikoiden soveltaminen EVE-virtuaalihuoneessa	
Päivämäärä:	10.2.2005	**Sivuja:** 85
Osasto:	Tietotekniikan osasto	**Professuuri:** T-111
Työn valvoja:	Professori Lauri Savioja	
Työn ohjaaja:	DI Seppo Äyräväinen	

Tämän diplomityön tavoitteena oli selvittää erilaisten ratkaisujen toimivuutta uusittaessa Teknillisen korkeakoulun Experimental Virtual Environment (EVE) -virtuaalihuoneen renderöintilaitteistoa. Työn alkuvaiheessa tutkittiin ja vertailtiin eri laitteistoalustoja ja lopulta soveltuvimmaksi alustaksi osoittautui PC-pohjainen ohjelmallisesti synkronoitu klusteriratkaisu, joka tulisi hyödyntämään EVE:ssä valmiina olevaa aktiivistereoympäristöä.

Työ keskittyi PC-pohjaisen renderöintiklusteriympäristön rakentamiseen ja testaamiseen sekä lopulta asentamiseen virtuaalihuoneeseen. Genlock-synkronointi hoidettiin tarkoitukseen sopivalla SoftGenLock-ohjelmistolla ja koneet yhdistettiin Gigabit Ethernet -verkkoa käyttäen. Koneiden klusterointia varten kehitettiin tarpeeseen sopiva ohjelmisto (Broadcast GL), jonka tehtävänä on jakaa keinotodellisuussovellusten käyttämä polygonidata renderöiville koneille.

Avainsanat: Klusterirenderöinti, Keinotodellisuus, Aktiivistereo

Acknowledgements

This Master's thesis has been done for the Telecommunications Software and Multimedia Laboratory at Helsinki University of Technology.

I wish to thank professors Lauri Savioja and Tapio Takala for continuous guidance with the thesis.

My gratitude also goes to Janne Kontkanen, who gratefully introduced me to the project.

I want to thank all the people who contributed into the new hardware and software construction, especially Markku Reunanen and Tommi Ilmonen for the codework and Seppo Äyräväinen for help with EVE and PC hardware.

I would also like to thank system administrator Mika Metsola for professional support with the Linux environment.

Finally, I would like to thank my family and friends for a delightful interest in my work and constant support along the way. Especially all of you that participated in proofreading the text and helping with the LaTeX typesetting.

Otaniemi, February 10, 2005

Petteri Kontio

Contents

List of Figures

List of Tables

Abbreviations

3D	Three Dimensional
AGP	Accelerated Graphics Port
AMD	Advanced Micro Devices Inc.
API	Application Programming Interface
AR	Augmented Reality
ATI	Array Technology Inc.
BGL	Broadcast GL
CAD	Computer-aided Design
CAVE	Cave Automatic Virtual Environment
CPU	Central Processing Unit
CRT	Cathode Ray Tube
DLP	Digital Light Processing
EPIC	Extendable Phased Interaction Controller
EVE	The Experimental Virtual Environment at HUT
FC1	RedHat Linux Distribution Fedora Core 1
FLUID	Flexible User Input Design
FOV	Field of View
FPS	Frames per Second
Gb	Gigabit (1,000,000,000 bits)
GB	Gigabyte (8,589,934,592 bits)
GLX	OpenGL with X Window System
GPU	Graphics Processing Unit
HMD	Head Mounted Display
HP	Hewlett-Packard
HUT	Helsinki University of Technology
I/O	Input/Output
IP	Internet Protocol
IR	Infrared
IR2	SGI InfiniteReality2 Graphics Subsystem
IRIX	SGI UNIX-like Operating System
IRQ	Interrupt Request
LAN	Local Area Network
LCD	Liquid Crystal Display
LOM	LAN on Motherboard

Mb	Megabit (1,000,000 bits)
MB	Megabyte (8,388,608 bits)
MIPS	Microprosessor without Interlocked Pipeline Stages
MPI	Message Passing Interface
NIC	Network Interface Card
NPV	Net Present Value
OpenGL	Open Graphics Library
OpenSG	Open Source Scenegraph
OS	Operating System
PCI	Peripheral Component Interconnect
RAM	Random Access Memory
RGB	Red-Green-Blue Component Video Signal
RISC	Reduced Instruction Set Computing
RPM	Red Hat Package Manager
RTAI	Realtime Application Interface for Linux
SGI	Silicon Graphics, Inc.
SIG	Special Interest Group
SLI	Scalable Link Interface
SMP	Symmetric Multi-Processor
SMT	Simultanous Multithreading
SSH	Secure Shell
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TML	Telecommunications Software and Multimedia Laboratory (HUT)
UDP	User Datagram Protocol
VGA	Video Graphics Array
VR	Virtual Reality
VRAC	Virtual Reality Applications Center (Iowa State University)
x86	Intel 80x86 Microprocessor Architecture

Chapter 1

Introduction

The concept of Virtual Reality has been heavily dominated by the presence of proprietary rendering hardware. In the past few years the increasing performance and low price of commodity graphics hardware has brought up new potential in the design of virtual reality environments.

To meet the rendering performance requirements, the commodity graphics hardware is built up into a cluster, a flexible configuration of multiple computers. Although the low-cost hardware solution might initially seem optimal for several applications, there are plenty of issues that need to be tackled.

1.1 Motivation

At the current rate of increase in the requirements for rendering hardware, constant upgrading of a high-end rendering platform has become too expensive for the purposes of the Experimental Virtual Environment (EVE) (see Figure 2.1) at Helsinki University of Technology (HUT). This study was set out to find out if the same functionality could be achieved with a low maintenance cost solution.

1.2 Objectives

The first objective of this study was to find out the hardware solution that is most suitable for the EVE. The following criteria were given to outline the new system:

1. To ensure that the current active stereo (see Section 2.2.2) environment could be efficiently utilized.
2. To ensure that the system will not cause unnecessary burden for the application developer.
3. To ensure that most of the currently used applications can be used in the new system without an excessive workload in porting the software. In other words, there should be a defined way to transfer the applications to the new system.
4. To enable short and preferably low-cost upgrade cycles.
5. To offer competitive performance compared to the current solution with lower operating costs.

The final objective was to analyze the chosen hardware platform and design the standard software architecture to be used on it. The chosen software was expected to meet the following requirements:

1. To offer a compact environment for commonly needed functions, such as calculation of the wall projections and navigation support.
2. To offer full utilization of the rendering capabilities of the hardware.
3. To offer a user-friendly Application Programming Interface (API) for graphics calls and external libraries.

1.3 Research Problem

The main issues in the construction of the rendering environment were performance, upgradeability and software support. In respect to performance, the new system should exceed the rendering speed of the current Silicon Graphics (SGI) Onyx2 system. Measuring this performance is not trivial, as it depends on several factors, like structure of the rendered scene, network usage and user interaction. Furthermore, both the software and the hardware in the new system should be easy to upgrade when needed.

Software support was also taken into consideration. Given that the old system uses the IRIX operating system, all software needs to be reconfigured if this environment changes.

1.4 Organization of the Thesis

The goal of the theoretical part of this project was to study the different hardware alternatives for real-time rendering in virtual environments and find out the most suitable option for the HUT EVE. The first two chapters present the theoretical background and the third chapter compares the different hardware solutions and presents the chosen solution.

In the practical part, we constructed, documented and benchmarked a cluster rendering environment using specialized software and commodity hardware. The emphasis here is on the fourth chapter, which presents the evaluation of different software frameworks and the reasoning behind the chosen software platform. The last two chapters present the outcome of the project and the possible future development of the system.

Chapter 2

Background

This chapter presents some of the Virtual Reality (VR) -related concepts that are needed to understand the later parts of the book. The chapter begins with the relevant historical data and later introduces the different hardware and software systems that apply to building clustered rendering environments.

2.1 History of CAVE-like Environments

The Cave Automatic Virtual Environment (CAVE) (Cruz-Neira, Sandin & DeFanti 1993) is a cubical construction of four to six projection surfaces. These surfaces are usually back-projected, so that the image comes from behind, but the surface can still be viewed from the front. Often due to space limitations in the facilities, the images from the projectors have to be reflected through mirrors. As an example of a CAVE-like environment, the EVE setup showing the projectors, mirrors, tracker and audio system is presented in Figure 2.1.

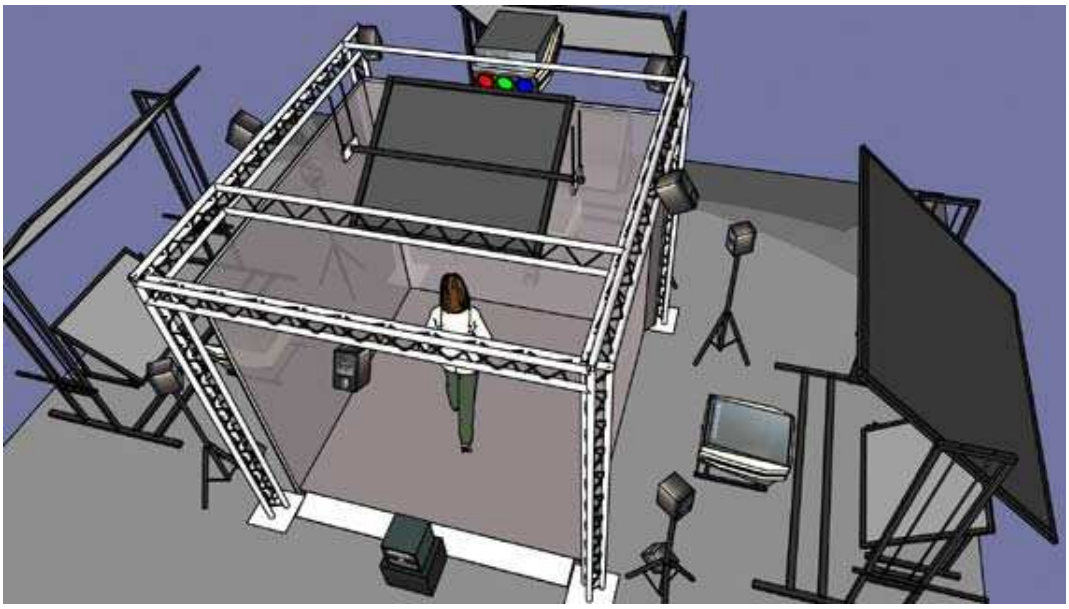

Figure 2.1: Model of the Experimental Virtual Environment at HUT (Image courtesy of Seppo Äyräväinen)

The history of the CAVE dates back to the beginning of 1990s, when a research group at University of Illinois, led by Dr. Carolina Cruz-Neira, started developing an immersive projection-based virtual room. The system was built to offer a competitive alternative for workstation-based scientific visualization and an intriguing showcase for the SIGGRAPH conference in 1992. (Cruz-Neira et al. 1993)

Furthermore, the CAVE took advantage of a construction of walls, where the projection planes needed not to be perpendicular to the viewer. This allowed the projections on the cube to be constructed into a spherical approximation, thus dissipating the cubical room from the viewer. (Cruz-Neira et al. 1993)

After the design of CAVE was presented, it quickly gained wider popularity in different research projects around the world. Currently there are dozens of CAVE-like installations, some of which are delivered as commercial products and some that are built by the users. CAVE is now a registered trademark of Fakespace Systems Inc., which is the reason why the virtual environment at HUT is called EVE rather than CAVE.

2.2 Virtual Reality

Virtual Reality is usually considered as a computer-generated medium for presenting artificially created sensations to the user. Sometimes the concept of VR is extended with the concept of *Augmented Reality* (AR) that mixes the physical world with the computer generated virtual information (Sherman & Craig 2004). This section presents the concepts required to understand VR in the way it is commonly used.

The term virtual reality has several different definitions. One of the more comprehensive of them is the one presented by Sherman & Craig (2004), where VR is divided into four key elements: *virtual world*, *immersion*, *sensory feedback* and *interactivity*. Other similar definitions are available, for example, in Kalawsky (1993) and Burdea & Coiffet (2003).

A virtual world is a simulated, usually 3D, environment that presents the virtual objects and their interdependency to the user. In a wider sense, almost any imaginary space can be considered a virtual world, although when associated with virtual reality, we usually want to define it as a collection of virtual objects and their relationships.

Immersion describes the user's more or less strong feeling that they are present in the virtual world. In respect to VR, immersion is usually achieved with a stereographic image over a large field or view (FOV), but this is not necessary by definition. Even an everyday experience like reading a book can be considered immersive, but the level of immersion is usually the stronger the more senses are being stimulated. Furthermore, immersion can be divided into mental and physical immersion, where the mental immersion is used to describe the suspension of disbelief and the physical immersion to describe the stimulus of the physical senses, which is most often achieved through sensory feedback.

Sensory feedback is the most crucial element of virtual reality. All equipment that create sensations to any human sense are considered sensory feedback devices. A typical VR environment produces at least visual and auditory feedback, but also motion-based (producing stimuli for sense of balance), haptic (producing touch stimuli) and even gustatory (producing taste stimuli) feedback solutions have been successfully utilized. The visual feedback is usually given as a stereographic image, so that the view between the eyes differs slightly. The auditory feedback is usually three-dimensional, so that the user can sense the direction and the distance of the sound source.

To provide the interactivity, a VR system is usually equipped with *motion tracking* to receive the movements of the user as an input to the rendering system. Tracking the position and orientation of the user's head is used to calculate and render the correct wall projections from the supposed viewpoint. Tracking at least one of the user's hands is used to provide an input interface. Although much more comprehensive motion tracking systems are available, these are usually minimum requirements for a usable VR installation.

2.2.1 Stereoscopic Vision

Most humans have the ability of seeing the world in three dimensions through the use of stereoscopic vision, also known as *stereopsis*. The 3D vision is created in the brain from the two separate images that are seen by the eyes. The Figure 2.2 shows how the two separate flat images are combined into a single three-dimensional stereoscopic image. The stereopsis, however, is not the only depth cue that affects the stereoscopic vision. For example, the relative positioning and movement of the objects and the focus of the vision can be used to increase the stereoscopic perception.

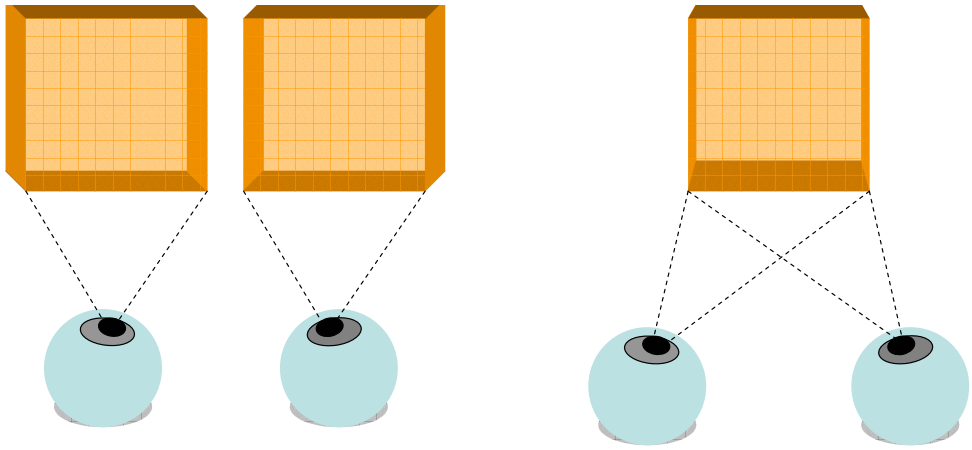

Figure 2.2: Human Stereoscopic Vision. On the left side of the figure, both the eyes separately see a slightly different image of the target. The right side of the figure shows, how the target is perceived with the help of stereoscopic vision.

2.2.2 Stereographic Displays

The stereographic projection of images is one of the most crucial elements in VR installations. This section describes different types of displays for creating three-dimensional illusions for the user.

Sherman & Craig (2004) categorize the stereographic displays under three different paradigms: *stationary displays*, *head-based displays* and *hand-based displays*. The first category is further divided into *fishtank VR* and *projection VR* and the second category into *occlusive HMDs* (Head Mounted Displays) and *nonocclusive HMDs*. Our interest here is directed towards traditional VR setups, not so much towards nonocclusive HMDs or hand-based displays, which are usually associated with augmented reality.

Fishtank VR stands for a stationary, usually *autostereoscopic* display. This means that the users do not need to wear any additional equipment to sense the stereographic image. These systems are usually well suited for workstation-based tasks, such as Computer-aided Design (CAD). Such installations are relatively inexpen-

sive, but often also limited only to provide the image for a single user and offer a FOV limited by the edges of the monitor (Sherman & Craig 2004).

A HMD is a special helmet or glasses that the users need to wear to see the virtual environment. The stereoscopic vision is created by installing two small displays inside the helmet. HMDs work best in solutions where the FOV must be unlimited. However, HMDs usually have fairly low resolution and can only support one user at a time.

The projection display systems are typically large VR installations, such as the CAVE. To clarify the problem of calculating the correct projections to the projections surfaces, the Figure 2.3 shows an example of a four-sided CAVE-like environment with correct projections. The projections for each wall are presented in the left side of the figure and the actual setup is shown in the right. In this example, the position of the viewer is represented by the colored squares in the middle of the cube and the projections are calculated using that viewpoint. Note that the figure only shows the correct projections for a single eye. In a real life application, a four-wall CAVE would require eight different projections to be calculated.

The two basic methods for creating a stereoscopic projection is with active or with passive equipment. In active stereo rendering, the projected image is rapidly changed between the images for the left and right eyes. This switching rate of the images is then synchronized with shutter glasses that only show one of the two images at a time. The largest limitation in this system is the problem of producing high quality images at a sufficient rate. The minimum vertical refresh rate for a flicker-free image is around 50 Hz but a rate of 60 Hz is usually the preferred minimum. In an active-stereo environment, the projectors have to be capable of displaying consequent images at double the speed of a single-eye refresh rate, resulting in the desired minimum vertical refresh rate of around 120 Hz.

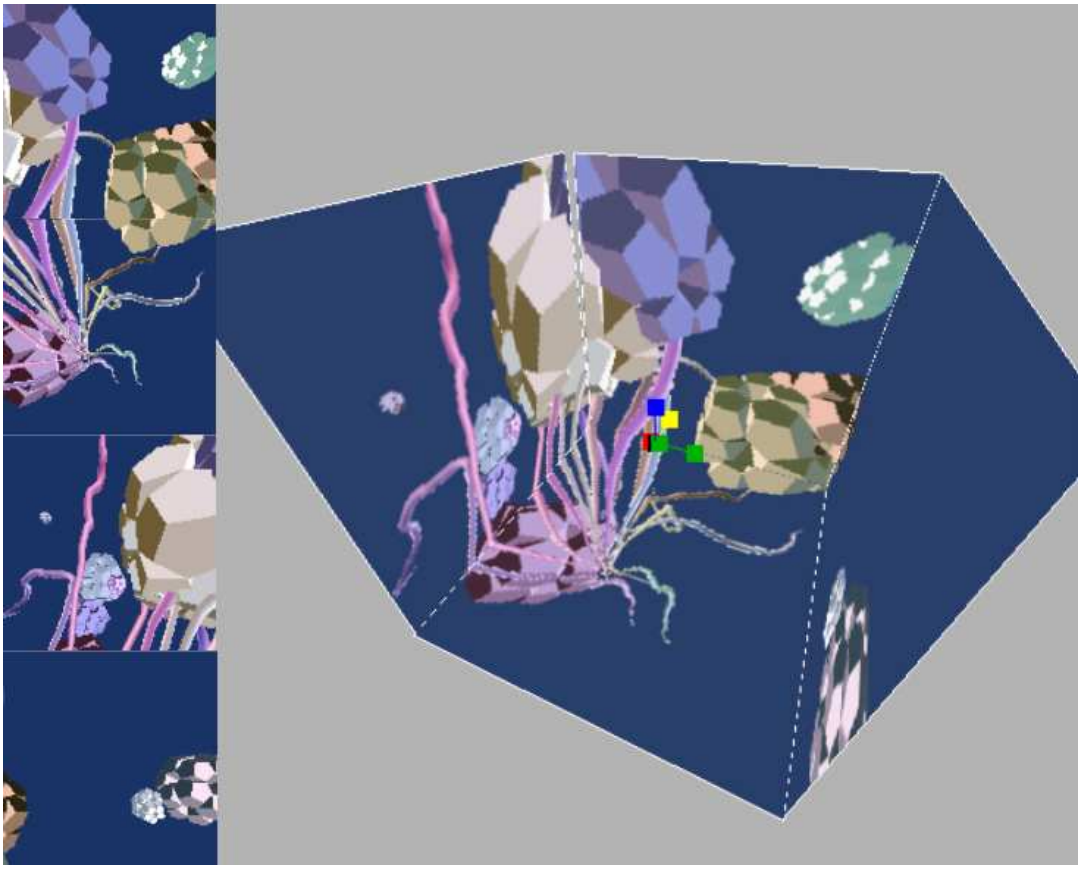

Figure 2.3: Surface Projections on a Four-wall CAVE (Image courtesy of Samuli Laine)

The projector technology used in active stereo rendering has traditionally been Cathode Ray Tube-based (CRT). Only projectors with fast enough phosphorus can be used with active stereo. This means that the projected image must not remain in the display for longer than the single frame delay, as too slow image decomposition will cause unwanted artifacts in the image. Liquid Crystal Display (LCD) projector technology is natively too slow to reach the high refresh rates required by active stereo. On the other hand, in Digital Light Processing (DLP) projectors, the desired vertical refresh rate is only supported in the high-end models.

Passive stereo means that the separate images for the left and right eyes are displayed simultaneously. When projected to the display, the passive stereo images must be separated for each eye. Most commonly this is done by installing linear (perpendicular) polarization filters to the projectors and by using glasses with matching linear polarizations. This method allows one projector to display vertically polarized and the other one to display horizontally polarized image on the same screen. As these projectors need not reach high vertical refresh rates, there is much more freedom in the selection of projectors. Therefore passive stereo can be done even with inexpensive LCD projectors (Woods 2001). In this case, however, the projection walls need maintain the polarity of the light.

The linear polarization method discussed above has a slight drawback. When the viewers tilt their head so that the two polarizations are no longer parallel, the stereo effect disappears. This problem has been solved with *circular polarization*. This method filters the incoming light in a spiral pattern, where the left and the right eye have a different directions on the orbicular path. With this technology, tilting the head has no effect to the stereo.

2.3 Experimental Virtual Environment at HUT

The establishing of the Telecommunications Software and Multimedia Laboratory (TML) in 1995 and an explicit demand from Finnish industry in 1997 were the driving forces behind the creation of the Experimental Virtual Environment (EVE, formerly known as HUTCAVE) project at Helsinki University of Technology. Originally the system consisted of a single wall, but after the system was moved to new premises in 1999, it was extended to a four-wall setup. (Jalkanen 2000)

2.3.1 Current EVE Hardware

The rendering hardware used in the EVE was built in 1997 upon SGI Onyx2 platform. This system is highly scalable, in theory allowing up to 128 CPUs and up to 16 InfiniteReality2 (IR2) graphics subsystems, also known as pipes. Initially the system, later named *Hermit*, was equipped with one IR2 and four CPUs running at 195 MHz. Three years later, in late 2000, Hermit was extended to carry two IR2s and 8 CPUs running at 250 MHz. Currently this is the maximum that can be installed into a single Onyx2 rack, making further upgrading of Hermit require additional racks, a more spacious server room and a new cooling system. (Jalkanen 2000)

2.3.2 Current EVE Software Architecture

The software used in Hermit can be roughly divided into three categories, that is, commercial software packages and plug-ins, third party open source software and software written in the EVE research group. The Figure 2.4 depicts the main components in the current software architecture and their main fuctions. The figure also depicts the two tracks for building software in the EVE. The components involved in the FLUID track are marked with white background and the components using the Performer track are marked with gray background.

The architecture is shared into four levels that are marked with yellow boxes on the left side of the figure. The first level is reserved for the applications that are executed in the system. The second level is for special controller software, in this case EPIC (see Section 2.3.2), that add certain useful high-level functions to the application. The third level is for input or output (I/O) software that control the user interaction and graphic and sound output. The fourth level represents the hardware parts, i.e. the actual input and output devices, such as displays and sound hardware.

The following sections contain a description of the software products that are commonly used in the Onyx2-based system.

FLUID

FLexible User Input Design (FLUID) (Ilmonen & Kontkanen 2002), is an open source software developed at the EVE research group. The software is constructed as a lightweight library, designed to process data from a variety of input devices. FLUID has become the de facto method for handling of input device events in the EVE, so it is bound to be used regardless of the underlying rendering hardware.

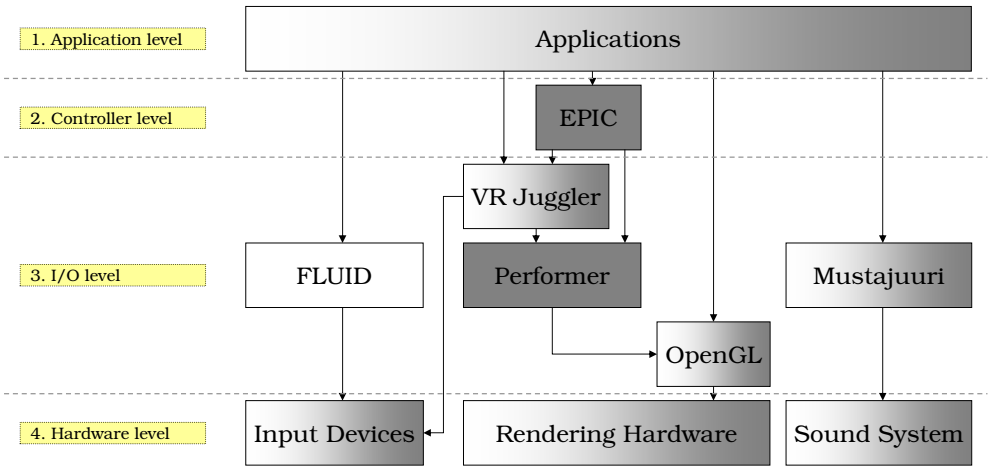

Figure 2.4: Current EVE Software Architecture. The two software tracks are marked with white and gray backgrounds.

FLUID was primarily designed for Linux, but was ported for IRIX for the purposes of the EVE.

In Figure 2.4 the components on the FLUID track are marked with white background. Generally the applications that use FLUID, do not depend on the components marked with gray background, although they may optionally use the ones marked with both colors.

Mustajuuri

The audio signal processing application and toolkit Mustajuuri (Ilmonen 2001), is also an open source product, designed for creating three-dimensional sound environments. Mustajuuri has been designed with portability in mind, so the software is available for a variety of platforms, including SGI IRIX and Linux. In the EVE, Mustajuuri is running in a separate PC system for controlling the entire audio signal processing, so it has no interaction with the other software components. Hence, in Figure 2.4 Mustajuuri is presented as a separate entity controlling the sound system.

VR Juggler

VR Juggler is a complete software toolkit for handling almost all aspects related to designing VR software (Cruz-Neira, Bierbaum, Hartling, Just & Meinert 2002). In the EVE, however, VR Juggler has been used mainly for calculating the correct projections to each of the projection surfaces. VR Juggler is further described in Section 2.7.5. Since VR Juggler can be used with both FLUID and EPIC/Performer, in Figure 2.4 it is presented in shaded gray. However, the ability of VR Juggler to control the input devices is mutually exclusive with FLUID. Another common solution to control the input devices and projections in SGI-based VR installations is the CAVELib (Pape 1997), but it is not used in the EVE.

OpenGL Performer

SGI OpenGL Performer (Eckel, Jones & Domeier 2004), later referred as Performer, is an application toolkit for designing OpenGL applications. In the EVE, Performer has been used for creating and executing *scene graphs* for several interactive applications. Scene graphs are data structures that hold more or less complex sets of polygon structures and their properties but also instructions or code for polygon and object interaction. Essentially, the scene graph implementation in Performer supports polygon setups, light sources, textures and *engines* that can be used to perform more complex actions.

Currently the most used feature in the software is the ability to load polygon models in several formats. Many of the most commonly used applications in the EVE have had their 3D models stored in either *OpenFlight* or *OpenInventor* format, both of which are supported by Performer.

SGI offers Performer also for Linux and Windows platforms, but the software has a policy that does not allow transferring the license from one operating system to the other. The currently used IRIX version of Performer would therefore be useless in a commodity cluster-based environment. The software is also available as a free demo version, but that has some unwanted features that make it unusable in production use. Recently, most of the features offered by Performer could also be provided by other software, such as OpenSG, but in the EVE, Performer was the solution of choice.

EPIC

Extendable Phased Interaction Controller (EPIC) is a toolkit for creating interaction and navigation in a virtual environment. EPIC was designed in the EVE research group by Matti Gröhn, Iikka Olli, Markku Mantere, Jukka Rönkkö and Seppo Äyräväinen. In addition to the basic navigation, EPIC was designed to meet

a variety of user interaction methods, like controls and menus. EPIC was designed to act as a front end for the functions offered by Performer, such as changing textures and materials. Unlike FLUID, EPIC is built upon VR Juggler and Performer and cannot be used separately (see Figure 2.4). Nevertheless, EPIC was designed to be portable and it should be easy to port on top of a working VR Juggler/Performer installation in some other than an IRIX environment.

2.4 Hardware Status

Traditionally, visualization of large datasets for scientific purposes has required special high-end hardware based on *shared memory*. Shared memory is a block of random access memory (RAM) that can be accessed by several central processing units (CPUs) simultaneously. VR solutions usually require substantial bandwidth to render the data from the memory, which tends to favor a genuinely shared memory system over a clustered system using the *distributed memory* concept. This section presents both the high- and low-end hardware as they are currently available in the market.

In 1965 Gordon Moore made his famous discovery, now known as the Moore's Law, which states that the number of components in an integrated circuit doubles every 12 months (Moore 1965). Later, Moore's Law was adopted into a form that the processing powers of computers will double every 18 months. This development has continued to the present day and will especially well take place within commodity hardware. According to Intel (Intel Inc. 2005) the trend will continue in the foreseeable future, at least through the end of this decade. In such an environment it is economically feasible to support short upgrade cycles for inexpensive hardware rather than long cycles for expensive equipment.

2.4.1 SMP Hardware Status

Symmetric Multiprocessing (SMP) is a shared memory architecture where all the CPUs share an access bus to a single chunk of RAM. The next two sections evaluate the current hardware status from the perspective of this project.

Traditionally the rendering supercomputers, such as the SGI Onyx2, have been equipped with CPUs built with the RISC (Reduced Instruction Set Computing) design philosophy according to the MIPS (Microprocessor without Interlocked Pipeline Stages) architecture. These systems were natively built for SMP and could easily outrun the x86 processors in computing performance. Currently SMP computing has become a consumer market and the two major players, Intel Corporation and AMD (Advanced Micro Devices Inc.) both have their own line of CPUs designed for SMP. Both Intel and AMD are also pursuing to the supercomputing market with their highly scalable Itanium 2 and Opteron product lines.

The low-end CPUs, such as the Intel Pentium 4, are equipped with Simultaneous Multithreading (SMT) (Tullsen, Eggers & Levy 1995), called *Hyper-Threading*. The technology is actually nothing more than a system for better utilizing the functional units of the CPU when running threaded applications. In practice, the operating system sees the Hyper-Threading implementation as a two-processor SMP system, although the optimal process scheduling between SMP and SMT differs considerably.

Due to the fact that it has become increasingly difficult to increase the clock speeds of the CPUs, the next, more sophisticated method of bringing SMP to the low-end CPUs is to add a second core to the processor. Such a dual-core CPU would be natively SMP, although both the cores could be further equipped with an SMT solution.

2.4.2 PC Hardware Status

The Accelerated Graphics Port (AGP) bus has been the most common interconnection method between graphics adapters and the system bus. Essentially, the architecture only supports a single AGP card per system, making clustering the only effective way to increase performance. In 2003, the Peripheral Component Interconnect Special Interest Group (PCI-SIG) introduced a new bus system called the PCI Express. PCI Express is a serial local bus implementation that is expected to replace both PCI and AGP buses in the near future. At the time of writing, major graphics card manufacturers have already adopted PCI Express into their service offering, but currently there is no substantial performance increase between AGP and PCI Express. Nevertheless, the PCI Express architecture allows architecture of several high-speed buses for graphics cards in a single PC system, which might be a useful feature when building parallel rendering systems. In January of 2005 there were already Scalable Link Interface (SLI) mainboards available, which allow a setup of two parallel graphics cards connected to the PCI Express bus.

2.4.3 PC Networking

Transferring polygon data over a cluster of PCs generally requires a high-speed, low-latency local area network. Such network solutions at an affordable price range are not common, leaving only a few viable alternatives.

The IEEE 802.3z Gigabit Ethernet standard is slowly replacing its predecessor, the 100 Mbps Ethernet that has become the de facto Local Area Network (LAN) standard all over the world. In September of 2004, the Gigabit Ethernet Network Interface Cards (NIC) were available in retail for as low as 15 euros per card, and low-end 8-port switches starting from 70 euros.

The upcoming IEEE 802.3ae 10 Gigabit Ethernet standard is bound to become an appealing alternative for the current gigabit networks, but the price point is currently

too high for a commodity hardware solution. In March of 2004, the technology was still priced beyond consumer market, to about 10,000 euros per port (Reardon 2004), but the prices are decreasing and in January of 2005 10 Gigabit NICs were available for around 3,000 euros.

In scientific visualization and computation clusters, the Myrinet (Myricom 2004) network has become a viable alternative for Ethernet networks. Compared to the Ethernet solution, the Myrinet offers an extremely low-latency connection with transmission speeds up to 7 Gbps (Boden, Cohen, Felderman, Kulawik, Seitz, Seizovic & Su 1995). In September of 2004 the starting price for the NICs was at 600 euros per card and 4,000 euros for an 8-port switch.

An important issue with the network performance is the switch's backplane data transfer rate. Under heavy traffic, the bandwidth of low-end Gigabit Ethernet switches can become the whole system's bottleneck. The backplane bandwidth of gigabit-class switches varies from a couple of gigabits per second to dozens of gigabits per second.

2.4.4 PC Graphics Hardware

The commodity graphics hardware has undergone a great increase in performance during the last decade. Nowadays, the PC graphics hardware can be divided into two rough categories: the low-end hardware designed especially for gaming, and the high-end hardware designed for advanced visualization solutions.

In the highest end of graphics cards, NVIDIA is offering Quadro FX cards with hardware-generated genlock. In September of 2004 they had two models especially designed for rendering clusters, namely Quadro FX 3000G and Quadro FX 4400G. The web-based list price for the 3000G is currently around 2,000 euros (Monarch Computer.com 2004), whereas the 4400G is a slightly more advanced model and because it was only recently released, it is still lacking online price infor-

mation. Also 3Dlabs has a genlock-enabled adapter, the Wildcat II 5110-G, in the same price group. In October of 2004 NVIDIAs strongest competitor, ATI (Array Technology Inc.) had no product with hardware genlock capability.

Within the low-end cards, the selection is broader and the prices vary from under 100 to nearly 1,000 euros. For the last few years, top-notch cards with a reasonable price-performance ratio have been priced around 500 euros. When choosing between commodity graphics cards, the key issue is the pure rendering performance. This is usually proportional to the price of the card, although the newest models tend to have a slightly worse price-performance ratio.

During the year 2004, the newly introduced mainboards and PCI Express graphics cards have become a viable alternative for the traditional AGP solutions. NVIDIA states that their Linux drivers have included a support for PCI Express since version 1.0-6106 from June 30, 2004 (NVIDIA.com 2004*b*). Since the hardware and the driver are at a relatively early stage of development, it can be assumed that there will be compatibility issues for months to come.

2.5 Polygon Data Transfer Levels

Before examining the different software used for polygon data transfer within a cluster, it is worthwhile to notice that the different methods function on different levels of the software architecture. This review is especially useful for understanding the events that take place when rendering in a cluster environment. The four basic levels are presented here in detail.

2.5.1 Input Event Level

Data transfer on input event level means that each rendering computer (later referred as a node) simultaneously runs the same application. In other words, each user

input event that affects the model, must be transferred to each node. This method becomes most useful when the network bandwidth is an issue, while the amount of transferred data is usually low. This method is in use e.g. with Cluster Juggler (Olson 2002) and Net Juggler (Allard, Gouranton, Lecointre, Melin & Raffin 2002*b*).

2.5.2 Scene Graph Level

Data transfer on this level is usually strongly accompanied with the so called *retained rendering mode*. In this mode, the nodes hold an instance of the rendered scene and the transferred data only includes changes in it. While the system must maintain several instances of the same data, the data between the nodes need to be synchronized (see Section 2.6.3). OpenSG (Reiners, Voß& Behr 2002) is an example of software using this method.

2.5.3 Graphics Primitive Level

Data transfer on graphics primitive level means that each node receives the set of polygons to be rendered. This method can be further divided into two common methods. The *sort-first rendering* is the most common and functions by culling unwanted polygons from the polygon sets that are transferred to the rendering nodes. In the *sort-last rendering* each of the nodes is given a different set of polygons to render and after the images are rendered, they are composed together. Chromium (Humpreys, Houston, Ng, Frank, Ahern, Kirchner & Klosowski 2002) is likely the best-known solution that can utilize either one of these methods of data transfer.

2.5.4 Pixel Level

In pixel level transfer, the data contains image bitmaps that are already rendered. In this model, the actual drawing of the image is extremely straightforward, thus giving

a performance advantage over the other methods. On the other hand, pixel level transfer takes hardly any use of the graphics rendering capabilities on displaying nodes, so the trouble of rendering the image is transferred to the source node. This level can be considered more like a reference level when comparing the network usage of different rendering methods.

2.6 Synchronization Requirements

In a cluster-based environment, the data stream between nodes has to be strictly controlled and synchronized. This task is characterized by three required lock steps, namely genlock, swaplock and datalock. (Maxwell, Bryden, Schmidt, Roth & Swan 2002)

2.6.1 GenLock

The term *GenLock* is used to describe the method of synchronizing the video frames. This means that the separate video projectors controlled by the different nodes in the cluster must produce their images in the same phase. In other words, all the projectors start to draw a new image according to the genlock signal and the vertical refresh rate that it defines (video retrace synchronization). (NVIDIA.com 2004*a*)

Genlock can be achieved either with a specific hardware or software solution. Some specially made high-end graphics cards, such as the ones presented in Section 2.4.4, natively have a method of sharing the genlock signal. Due to the high price of these cards, there are also some software solutions to synchronize the nodes.

2.6.2 SwapLock

SwapLock is required to adjust the buffer swap of the nodes so that none of them is able to advance further than the others, even though it could render the data more quickly. SwapLock can be achieved with special graphics hardware that supports it, such as the genlock-enabled cards described in Section 2.4.4, or with software that handles the synchronization, such as Net Juggler (Allard et al. 2002*b*). If the nodes are not swaplocked, there is a risk of considerable lag appearing on the screen that takes most time to render.

2.6.3 DataLock

DataLock becomes an issue especially when using the aforementioned retained rendering mode. While each node has locally stored information about the data to be rendered, there is a risk that this will eventually produce inconsistent images (Maxwell et al. 2002). The locally saved data must also contain all the information required to render the view, for example the user position and head orientation. DataLock cannot be achieved with hardware, so the clustering software must support it.

2.7 Software

There are several pieces of software on the market that contribute to enabling a clustered rendering environment for commodity hardware. This chapter introduces some useful software packages that have gained a share of the market.

2.7.1 Linux Kernels

The Linux software architecture is built upon the kernel. The kernel is the core of the operating system and it controls low-level operations, such as load-balancing and multi-tasking. Of the software presented in this chapter, SoftGenLock is the most demanding, requiring a specific real-time patched kernel.

At the moment, the most commonly used kernel families are the 2.4 series, introduced in 2001, and the 2.6 series from 2003. A 2.4 series kernel should be suitable for SoftGenLock, but the developers of the software are expecting a real-time patched 2.6 series kernel to offer better scheduler accuracy.

2.7.2 SoftGenLock

SoftGenLock is a Linux software package that provides genlock and stereo within a cluster of PCs (Allard, Gouranton, Lecointre, Melin & Raffin 2002*a*). The stereo provided by SoftGenLock is *page-flipped*, which means that the XFree86 windowing system is given a virtual desktop, double the size of the rendered screen. This system provides a canvas that enables the application to draw separate images for the left and the right eye on the same desktop. SoftGenLock, as genlock in general, is only needed when using active stereo.

The operation of SoftGenLock depends on timing the VGA (Video Graphics Array) signal. The VGA signal consists of a set of horizontal pixel strips, called *scanlines*. The hidden area in the display, which allows the cathode-ray beam to move from a scanline to the next is called the *overscan* (Vorozcovs 2002). The overscan makes it possible to slow down or speed up the drawing of the image by adding or removing pixels located in this area. The synchronization between nodes is achieved through a parallel port interface. This system works according to the master-slave architecture, where the master sends a synchronization signal at the end of a frame. The slave nodes then measure the difference between the incoming signal and the end

of their own frame and proceed by adding or removing pixels accordingly (Allard, Gouranton, Lamarque, Melin & Raffin 2003). This method of using VGA registers is later referred as the timer-based solution.

The alternative method for synchronizing the signal is using pixel clock access. According to Allard et al. (2003), pixel clock is the hardware component used to activate the output of each pixel in the video signal. It is possible to slow down or speed up the image by altering the speed of this clock. This is done during the period of vertical blanking, that is, between the frames. However, the pixel clock implementation is usually hardware specific, so a different method must be used for different graphics hardware. The current version of SoftGenLock only supports NVIDIA hardware. This method of using the pixel clock is later referred as the IRQ-based solution. This is due to the fact, that the pixel clock is activated using the Interrupt Request (IRQ) that is given for the graphics card.

2.7.3 GLX

The X Window System, commonly known a X, is the standard graphical interface in Unix-like systems, such as Linux. The most commonly used implementation of X is the *XFree86*, which is available as an Open Source project.

GLX is the solution for delivering OpenGL instructions to be rendered to an X window. The normal X windowing system controls the screen output as an array of pixels in two dimensions, whereas using OpenGL and GLX makes it possible to use sophisticated methods for handling the capabilities of the frame buffer. (Kilgard 1994)

Using a built-in wire protocol of GLX, the OpenGL instructions can be transformed into a stream of commands. This stream is considered distinct from the stream of normal requests to the X window system, allowing direct access to the graphics pipeline (Womack & Leech 1998). This process, called *direct rendering*, considerably increases the polygon throughput performance.

2.7.4 Chromium

Chromium is an Open Source, graphics primitive level cluster rendering framework, based on the *WireGL* project at Stanford University. Chromium supports an extensive subset of OpenGL instructions and both sort-first and sort-last rendering methods. In addition to transferring the polygon data, Chromium has a variety of options for handling the data. The most distinctive feature in the Chromium architecture is the use of Stream Processing Units (SPUs) that are pieces of software designed for a specific purpose in handling the stream of rendering instructions, thus providing particular effects, such as turning the stream into grayscale or inverting its colors.

Chromium works according to client-server architecture. Each of the rendering nodes has a special Chromium server running to handle the incoming instructions. In addition, there is a separate client, called the *mothership*, which connects to each of the servers and generates the rendering commands. The normal OpenGL application is executed through the mothership so that it uses the Chromium's replacement OpenGL library instead of the one supplied by the system.

The Chromium system was originally designed to support tiled display walls, which differ from CAVE-like systems mainly in that the projection is essentially the same for all the tiles. This design also causes a flaw that only allows Chromium to handle a single projection matrix per configuration (Humpreys 2004), which makes it impossible to find a configuration that would calculate the correct projections in all applications.

2.7.5 VR Juggler

VR Juggler is an Open Source VR platform, originated from Virtual Reality Applications Center (VRAC) at Iowa State University. The project was largely based on the fact that most application libraries for VR applications were available as commercial products (Cruz-Neira et al. 2002). Nowadays VR Juggler is supposedly one of the most popular comprehensive VR toolkits mainly because it has been adopted by the growing Open Source community.

VR Juggler functions as a software layer between the application and hardware in a VR system. The layer is built upon generic interfaces that abstract the underlying Input-Output (I/O) devices, which makes the application unaware of any possible changes in the configuration of the hardware. Furthermore, the architecture of VR Juggler is based on a microkernel that mediates the instructions between so-called *managers*. The two most important managers are the `vjDrawManager` and the `vjInputManager`, of which the former handles the API calls for example with OpenGL, OpenSG or Performer, and the latter controls the input devices, such as motion tracker, data gloves and joysticks. (Cruz-Neira et al. 2002)

Although VR Juggler is often considered as an all-inclusive package, it can co-operate with third party software. A good example of this is the VR Juggler cluster support, which can be set up in various ways. Depending on the application, VR Juggler can be configured to work with nearly all data transfer methods described in Section 2.5. VR Juggler is known to work at least with OpenSG and Chromium.

2.7.6 Cluster Juggler

Cluster Juggler is an input event level cluster extension to VR Juggler, developed at VRAC. The system is designed especially to facilitate the migration from a high-end VR Juggler solution to a commodity cluster environment. The system architecture is designed to be so flexible that the nodes in the cluster do not even need to have a similar operating system. (Olson 2002)

2.7.7 Net Juggler

Net Juggler is an input event level cluster support addition for VR Juggler, developed by the SoftGenLock team. Contrary to Cluster Juggler, Net Juggler is based on a specific protocol called the Message Passing Interface (MPI) (Allard et al. 2002*b*). MPI (Forum MPI 1994) has become a popular communications protocol in distributed computing and Net Juggler supports both of the most common implementations, *MPICH* and *LAM-MPI*.

Chapter 3

Hardware Implementation

This chapter presents the different options that were considered to replace the current rendering hardware in the EVE. The options are compared based on the available information, but some rough estimation had to be made when the information was unavailable.

3.1 New Rendering Hardware Considerations

The two fundamental ways to construct the rendering environment for the EVE were either with a high-end Symmetric Multi-Processor (SMP) server or with a cluster of multiple low-end computers. This section presents the options that we considered, the criteria to effectively compare them, and the results of our comparison.

3.1.1 SMP Solutions

The traditional way to construct a visualization system is using high-end Symmetric Multi-Processor (SMP) server hardware with a sophisticated graphics subsystem. Such solutions are widely available mainly for scientific visualization purposes. This section presents a couple of solutions utilizing such hardware.

SGI

While the SGI Onyx2 has served well in its use in the EVE, it would be natural to update it using a more current version of the same hardware. From the software point of view, an IRIX-based option would be the most straightforward, as all or most of the existing software could be directly used in the new system.

The straight descendant of the Onyx2 would be the Onyx4 UltimateVision platform. This system is designed for high-end scientific visualization and it is available from the smallest form factor with 2 CPUs and 2 graphics pipes to a system of 64 CPUs and 32 graphics pipes (Silicon Graphics Inc. 2003). The Onyx4 is an IRIX-based system and is based largely upon SGI hardware, although it uses MIPS CPUs and ATI Graphics Processing Units (GPU). The list prices for Onyx4 configurations start at approximately 50,000 euros.

On the other hand, SGI seems to be geared even more strongly towards third party hardware in its most recent rendering solutions. The SGI Prism is a visualization system utilizing the Altix supercomputer platform that is based on the Linux OS, Intel Itanium 2 CPUs and ATI FireGL GPUs (Silicon Graphics Inc. 2004). The list prices for a Prism configuration start at approximately 25,000 euros.

The price difference between the Onyx4 and the Prism can largely be explained with the difference of technology. The Prism consists mainly of off-the-shelf hardware, while the Onyx4 is mostly equipped with proprietary SGI hardware. In addition, the starting price for the Prism is set relatively low to better compete with other similar setups on the market.

Other SMP Solutions

Also other major hardware providers have SMP-based solutions that could serve as a rendering platform. For example, such solution could be the Sun Microsystems Fire 880 server with two or more graphics pipes. Nevertheless, since the system would not run IRIX, but Solaris, the applications would still need to be ported. The task would not, however, be as demanding as porting for Microsoft Windows. The list price for such a system was estimated to start at around 40,000 euros.

3.1.2 Cluster Solutions

Some major hardware providers are offering preconfigured solutions for visualization clusters that could also be used for equipping the EVE. For example Hewlett-Packard (HP) offers a cluster solution based on their PCs and hardware genlocked NVIDIA FX 3000G graphics cards. The HP system includes some proprietary clustering software that is built on the Microsoft Windows Operating System. This was generally considered a drawback since all of the EVE applications were designed for a Unix-like OS. The list price for such a system was estimated to start at around 40,000 euros

Hardware Synched Cluster

The obvious solution for commodity hardware active stereo would be to equip the cluster with genlock-enabled graphics adapters. As described in Section 2.4.4, there is a variety of products available. The price for such a system is estimated to start at around 20,000 euros, roughly half of which is contributed by the rendering hardware.

Software Synched Cluster

Recently some research groups have studied software-generated genlock. The best-known open source solution for doing this is called SoftGenLock, which was better described in Section 2.7.2.

While there is an obvious requirement for specific synchronization software, the hardware can be selected more freely. The current version of SoftGenLock runs only on Linux and has a more comprehensive support only for NVIDIA GeForce graphics adapters. In addition to that, there are no strict limitations to the hardware.

As described in Section 2.5, there are various ways to handle the distribution of the polygon data. Especially, some of the methods have a larger inter-node bandwidth requirement, which increases the need for high-speed network infrastructure.

3.2 EVE Hardware Decision

Although commercial solutions seem to offer clear advantages over building a cluster from scratch, after the initial tests we had high hopes for our cluster. Therefore we decided to go for the software synched PC cluster and try to overcome the remaining problems as the project proceeds further.

While virtually free of operating expenses, the software-genlocked solution lacks the reliability associated with the more sophisticated systems that are usually bundled with a service contract. Nevertheless, the current hardware upgrade cycle of seven years is clearly too long to keep the environment up to date for newest VR applications. With off-the-shelf hardware it is possible to upgrade either the rendering hardware, or the whole system at relatively low cost – every two years, for example.

To fully understand the cost of a PC cluster, we constructed a model for the total cost of ownership (TCO). The largest issue here is the number of man-hours needed to get all the required applications running tolerably on a cluster. This can only be roughly estimated, but the cost will be considerable. A cluster can also be considered to cause some overhead for system administration, as there are multiple separate systems to be maintained. Furthermore, there are some hidden expenses in cluster environments that might incur in the future. For example, license policies in some software products might require a separate license for each node, or hardware upgrades might turn out to be more difficult and therefore more costly than in a commercial solution.

The following cursory model estimates the TCO incurred by different hardware solutions. This model only takes into account the following:

a. Initial investment.

b. Approximation of workload. The estimate is in full man-months of 4,000 € each.

c. Cost of a minor upgrade. This is an estimate of the replacement of the rendering hardware in the system.

d. Cost of a major upgrade. This cost is estimated to be equivalent to the initial investment.

e. Maintenance cost. This cost is estimated both by the work needed for system support, and by the need for spare parts. The value for HP and SGI systems includes the price of a service contract. Furthermore, according to conversations with the maintanence staff, the price of the upkeep workload was estimated to be around 50 euros per node per month in the cluster systems and 150 euros for other than Linux-based systems.

f. Discounting the net present value (NPV) of the investments to year 2016 on an interest rate of 5 percent.

The five different solutions that were compared according to the TCO model are listed in the Table 3.1. The given values are mostly estimates, including the investment and upgrade costs. The upgrade interval is estimated so that the hardware would constantly offer enough rendering power to exceed the performance requirements. The upgrade costs are estimates of the price of the rendering hardware and the whole hardware system. For high-end solutions, the upgrade interval is 50 percent longer than for the low-end clusters. The estimate on the workload was based on several estimates on the amount of time needed to get both the needed hardware and the software running smoothly on the new platform. The workload was only considered for the initial configuration, setup of hardware and porting of software, while further costs were included in the maintenance cost, as described in Table 3.2.

Solution	*Investment Cost*	*Workload Cost*	*Cost of a Minor Upgrade*	*Upgrade Interval (Minor / Major)*
PC Software Cluster	10,000 €	16,000 €	2,000 €	2/4 years
PC Hardware Cluster	21,000 €	4,000 €	8,000 €	2/4 years
HP Cluster	40,000 €	4,000 €	8,000 €	2/4 years
SGI Prism	25,000 €	4,000 €	8,000 €	3/6 years
SGI Onyx4	50,000 €	0 €	20,000 €	3/6 years

Table 3.1: Annual Investment and Upgrade Costs for Different Hardware Options

A graph visualizing the data of the TCO model is presented in Figure 3.1. The same data is available in form of yearly expenses in Table 3.3. Note that the expenses for the years 2004–2016 are given without discounting, while the total is a sum of all

Solution	*Number of Nodes*	*Service Expense*	*System Admin. Expense / Node*	*Software Support*	*Total Maintenance*
PC Software Cluster	5	1,200 €	600 €	Linux	4,200 €
PC Hardware Cluster	5	1,200 €	600 €	Linux	4,200 €
HP Cluster	5	3,000 €	1,200 €	Linux/Windows	9,000 €
SGI Prism	1	6,000 €	600 €	Linux	6,600 €
SGI Onyx4	1	12,000 €	1,800 €	IRIX	13,800 €

Table 3.2: Annual Maintenance Costs for Different Hardware Options

the expenses, including the yearly interest rate of 5 percent. This time scale was chosen because the development of the industry would be impossible to predict on a longer scale. On the other hand, the twelve-year period is long enough to give an impression of the trends related to the expenses.

Solution	*2004*	*2005*	*2006*	*2007*	*2008*	*2009*	*2010*
PC Software Cluster	27,000 €	4,200 €	6,200 €	4,200 €	19,200 €	4,200 €	6,200 €
PC Hardware Cluster	25,000 €	4,200 €	12,200 €	4,200 €	25,200 €	4,200 €	12,200 €
HP Cluster	44,000 €	9,000 €	17,000 €	9,000 €	49,000 €	9,000 €	17,000 €
SGI Prism	29,000 €	6,600 €	6,600 €	14,600 €	6,600 €	6,600 €	31,600 €
SGI Onyx4	50,000 €	13,800 €	13,800 €	33,800 €	13,800 €	13,800 €	63,800 €

	2011	*2012*	*2013*	*2014*	*2015*	*2016*	*Total (NPV)*
PC Software Cluster	4,200 €	19,200 €	4,200 €	6,200 €	4,200 €	19,200 €	178,877 €
PC Hardware Cluster	4,200 €	25,200 €	4,200 €	12,200 €	4,200 €	25,200 €	221,873 €
HP Cluster	9,000 €	49,000 €	9,000 €	17,000 €	9,000 €	49,000 €	402,562 €
SGI Prism	6,600 €	6,600 €	14,600 €	6,600 €	6,600 €	31,600 €	250,338 €
SGI Onyx4	13,800 €	13,800 €	33,800 €	13,800 €	13,800 €	63,800 €	513,211 €

Table 3.3: Annual Expenses for Different Hardware Options

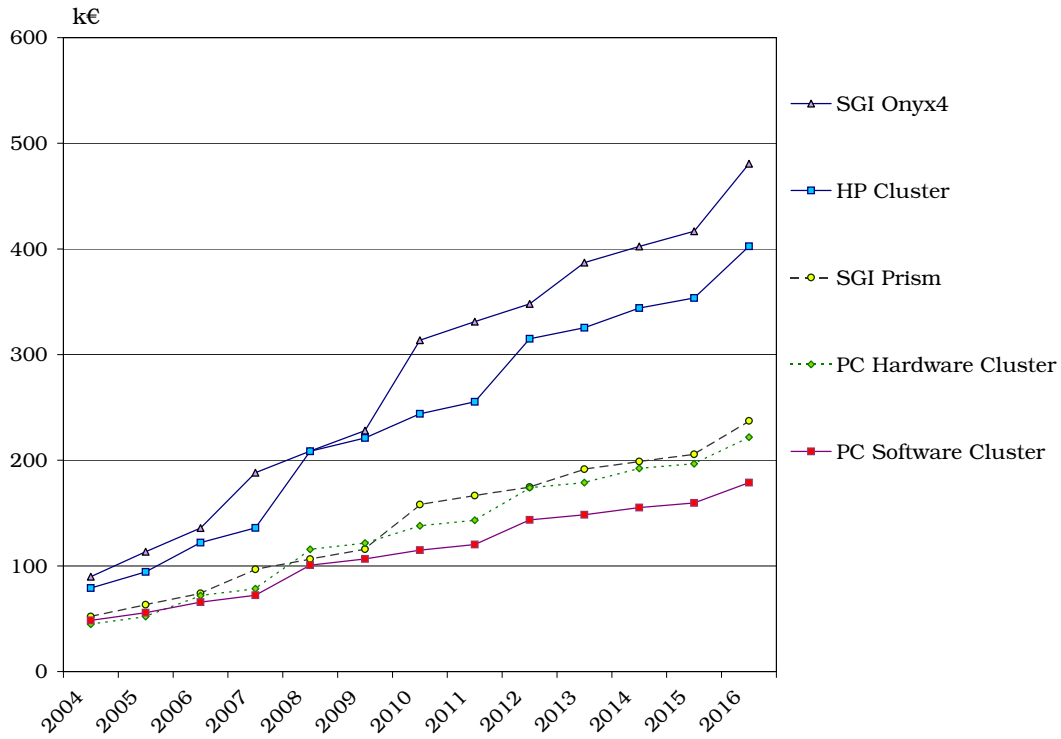

Figure 3.1: Total Cost of Ownership Model for the Different Hardware Solutions

The chosen solution meets relatively well the five hardware-related objectives presented in Section 1.2. From these objectives we derived the following criteria that were the most significant when we decided to go for the software-genlocked commodity hardware cluster.

1. High performance with low investment cost. According to the TCO model the chosen solution was clearly the most affordable. The amount of effort put into the work for porting the software can only be estimated, but that will be a major part of the investment. This corresponds to the fifth objective presented in Section 1.2.

2. Short and low-cost hardware upgrade cycles. Again according to the TCO model, the chosen solution seems the most affordable in the long run. This corresponds to the fourth objective in Section 1.2.

3. Ease of utilization of the existing software. Since some of the most important software packages used in the EVE, such as FLUID, were already available for the Linux platform, they could be easily moved to the new system. This corresponds to the second objective in Section 1.2.

4. Good support for Open Source software. Since the Open Source software supply for Linux can be considered better than for Irix and Microsoft Windows, choosing a Linux-based system was seen more as an investment for the future. Furthermore, although migrating into Linux platform was bound to create compatibility issues with the existing software, it was seen as an important move in the long run. Although the work needed to port the old software to the new system can be considerable, also the third object presented in Section 1.2 can be considered at least partially achieved.

5. Possibility to better research the cluster environments. As an academic investment, a lot of value was given for the fact that the investment itself could be considered as a stepping stone for new fields of research.

In light of these criteria and the result of the TCO model, the software-genlocked PC cluster seemed superior. Nevertheless, the most important aspect was the low price, which would enable us to avoid the pitfall of getting locked-in to a single high-end solution without the possibility of an inexpensive upgrade. The solution was also considered safe, in that it could be easily upgraded to the hardware-genlocked solution if necessary.

3.3 Hardware Buildup

The original three-PC cluster hardware was acquired in July of 2003 with a purpose of expanding it into a complete five-PC configuration, when the system was fully tested and operational. One of the original PCs was designed to function as the application node, and was therefore equipped with a bit different hardware setup. This section describes the constructed cluster in detail. An overview of the architecture and the connections is available in Figure 3.2.

The basic setup for each of the nodes was a Gigabyte GA-8KNXP (i875P) Socket 478 motherboard with an Intel Pentium4 2.8 GHz HT (800 FSB) CPU and 2 x 512 MB Kingston HyperX PC-3200 CL2 memory, and an integrated Gigabit Ethernet NIC. The graphics hardware for the rendering nodes was Creative GeForce FX 5900 Ultra with 256 MB of memory. In July of 2003, they were the highest performance commodity cards available off-the-shelf.

Since the application node will not participate in the rendering process, it was equipped with a low-end PNY Verto GeForce FX 5200 graphics adapter, and with an additional Gigabit Ethernet NIC to connect the cluster to an outside network. The application node is denoted as *A*, and the rendering nodes as *1*, *2*, *3* and *4* in Figure 3.2.

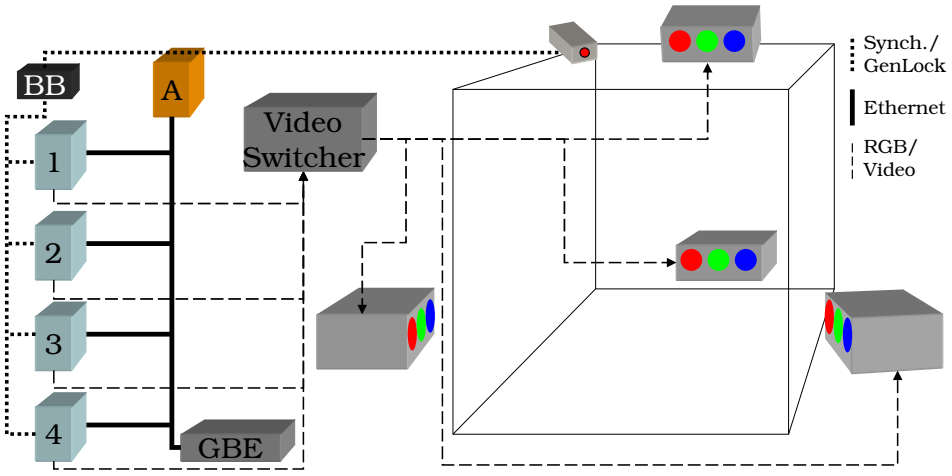

Figure 3.2: Architectural Overview of the EVE PC Cluster

The cluster was networked with an Edimax ES-5800R+ Gigabit Smart Switch (denoted as *GBE* in Figure 3.2). Although this switch is a low-end model, it was considered to be fast enough for most of the applications.

In addition, the Figure 3.2 shows the synchronization network connection box (denoted as *BB*), which is further described in Section 4.1.3, and the parallel port genlock signal connections denoted as a dotted line. Furthermore, the RGB component video signal is denoted as a dash line. The image also depicts the *Video Switcher* that controls the video inputs and outputs. With the help of the switcher, we can easily have both the PC cluster and the Onyx2 system connected as the inputs and choose which input we want to send to the projectors. Some photos of the system are available in Appendix D.

Chapter 4

Implementation of the Software

This chapter presents the steps taken in designing and implementing the software architecture for our rendering cluster. The chapter begins by presenting the process of building the system and later presents the used software and benchmarks the different solutions.

4.1 Cluster Setup

This section describes the manual process needed for installing a working cluster. The basic configuration of our cluster consists of five PCs: one application PC and one rendering PC for each of the four display surfaces of the EVE.

4.1.1 Linux Install

The installation of the cluster started with choosing a Linux distribution. We chose Fedora Core 1 (FC1) as most of the required binaries were available in Red Hat Package Manager (RPM) format, and at the time, FC1 was the most current version of that distribution. RedHat was also the distribution used in some previous

projects utilizing SoftGenLock, for example by Maxwell et al. (2002) and by Vorozcovs (2002), so it was a safe choice. It was also important that the author had most experience working with that distribution.

4.1.2 Real Time Linux Kernel Setup

SoftGenLock version 1.0 was designed to support both RTLinux and Realtime Application Interface for Linux (RTAI) real time kernels. However, the latest version `SoftGenLock-2.0-alpha3` only supports RTAI, so we started out by testing the RTAI patch. Furthermore, RTLinux is available also as a commercial version, which implies that the free version appears merely as an unsupported demo for the actual product.

The RTAI version we chose to install was `2.4.25-adeos`, which was the most current 2.4 kernel at the time. For the compilation of the custom kernel, we tried to include only the modules that were needed by the setup. SoftGenLock installs itself as a kernel module into `/dev`, so support for *devfs* was needed. Since our Pentium 4 processors supported the Hyper-Threading technology, we also compiled the SMP support into the kernel.

4.1.3 Parallel Synchronization Network Setup

To test the feasibility of software-genlocked stereo, a synchronization network utilizing parallel port connections was built. The SoftGenLock team basically offers two options for such a network: a ParaCable network and a TTL_PAPERS network (Raffin 2002). The ParaCable network was chosen because it is slightly more straightforward to construct. Also, the increase in scalability and the suggested 30 percent decrease in synchronization delay using the TTL_PAPERS network were not considered significant for the purposes of this project.

The ParaCable connection box was built according to the instructions by Raffin, except that the Mini Din 3 connector to the glasses was replaced by a DB-9 connector, used by our StereoGraphics emitters. The 12 V DC required by the emitters was taken directly from the PC power supply. The combined cost of the system, including the connection box and the cables, was under 50 euros.

4.1.4 SoftGenLock Install

Basically, installing SoftGenLock to a real-time patched kernel should be very straightforward. Nevertheless, we encountered several problems when setting it up. At the time of writing, the latest version of the software was 2.0-alpha3, so our tests focused on that.

SoftGenLock as a Kernel Module

The installation of SoftGenLock as a kernel module consists of two separate modules, namely `sgl_kernel.o` and `sgl_rtai.o`. The former is a kernel module to run the software, and the purpose of the latter is to enable the real-time mode.

The two basic methods for setting up the module are IRQ-based and timer-based solutions, as described in Section 2.7.2. We were first trying to get the timer-based solution to work but it turned out eventually to drop frames and lose synchronization, regardless of the functioning of the 3D application. The almost entirely undocumented IRQ-based solution was stable but useless, as the process consumed all available CPU cycles. This problem is further explained in section 5.1.2.

SoftGenLock as a User Process

In addition to the kernel module, the software contains a separate user-level program to enable genlocking. This program, called `sgl_user`, can be forced with a command

line parameter to function in real-time mode. At any rate, we noticed that the program is badly designed and, for example, contains a counter overflow that causes it to stop functioning after about an hour from the execution. The original code and the fix are presented below. However, according to the authors, this program is not even meant to be used for extended periods of time, but rather merely to quickly test that the system works.

```
static __inline__ sgl_time sgl_get_current_time_rdtsc()
{
  sgl_time st;
  rdtscll(st);
  return div6432((st-sgl_system_user_t0)<<8,sgl_system_user_tdiv);
}
```

This code creates an overflow because the quotient of the division (`div6432`) will not fit into the 32-bit target. For increased precision the numerator is switched 8 bits to the left which makes it unnecessarily large even for the 64-bit space. The following listing presents a quick hack to overcome the problem.

```
static __inline__ sgl_time sgl_get_current_time_rdtsc()
{
  sgl_time st;
  rdtscll(st);
  sgl_time st2 = st - sgl_system_user_t0;
  st = div6432((st2) << 6, sgl_system_user_tdiv);
  return st<<2;
}
```

Instead of switching 8 bits to the left we chose to switch only 6 bits with a small cost of the precision. Nevertheless, this had no visible effect to the functioning of the program, but allowed the program to run four times as long (approximately 6 hours) without interruption.

This user mode SoftGenLock became the best-working solution for our system. However, this is far from optimal, since the process consumes around 50 percent of the available CPU resources according to the *top* utility. As mentioned by Vorozcovs

(2002), SoftGenLock is designed so that the resource consumption should be minimal. Therefore, there is still work to be done to make the system run as it was intended.

4.2 Polygon Data Transfer Methods

To benchmark the performance of the rendering cluster, we tested a variety of methods for transferring the polygon data between the nodes. Some of the described methods turned out to lack the performance required to run interactive software, but they are still presented for completeness.

4.2.1 Serialized GLX Method

Starting from the most trivial construction, our first attempt to transfer the polygon data was to use a serialized program sending the GLX data. This program had a single loop that successively sent the data to each of the rendering nodes. The basic structure of the program is presented below.

```
#include <stdlib.h>
#include "glx_wrapper.h"
#include <GL/gl.h>

/* Number of nodes */
#define N 4

int main(void)
{
  int quit=0,n;
  char *displays[]={"10.0.0.1:0","10.0.0.2:0","10.0.0.3:0","10.0.0.4:0"};
  GLX window[N];

  for(n=0;n<N;n++)
    GLX_Open(&window[n],displays[n],0,0,1024,768,GLX_LOCAL|GLX_NODECOR);
```

```
  while(!quit)
  {
    for(n=0;n<N;n++)
    {
      GLX_SetActive(&window[n]);

      /* GL content here ... */

    }
    for(n=0;n<N;n++)
      quit+=GLX_Swap(&window[n]);
  }

  for(n=0;n<N;n++)
    GLX_Close(&window[n]);
  return(0);
}
```

The drawback of this solution turned out to be bad sequencing of commands. While information was fed to a single node, the other two nodes were idle. This caused unnecessary lag throughout the system, and flicker in video output.

4.2.2 Threaded GLX Method

After the poor performance of the serialized method, the GLX software was improved by adding a separate thread for each of the streams. This method turned out to be superior to the previous implementation and could be used to successfully render complex scenes in the cluster. The outline of the implementation is presented in Appendix C.

4.2.3 Broadcast GL Method

For the best possible network performance, a UDP/IP (User Datagram Protocol/Internet Protocol) broadcast method was constructed to transfer the polygon data. The system, later named Broadcast GL (BGL), encapsulates OpenGL instruc-

tions into UDP chunks and delivers the same set for each of the rendering nodes. While UDP is a *connectionless* protocol, it can be configured as a broadcast setup. Furthermore, compared to a solution using *connection-oriented* TCP (Transmission Control Protocol), the UDP packet parameters can be relatively freely adjusted, and the overhead from the headers is relatively small.

The system supports a UDP/IP multicast or broadcast to send a binary encoded stream of OpenGL API calls to the rendering nodes, which then have a return channel over a TCP/IP connection to acknowledge the instructions. Since the system must encapsulate the information into a specific non-standard form, only a subset of the OpenGL API can be supported. Especially functions that would return information from the rendering node to the application can not be efficiently implemented. (Ilmonen, Reunanen & Kontio 2005)

In respect to network load, it is useful to support UDP to transmit the OpenGL calls. While UDP is unreliable, it will not require similar packet acknowledgement as TCP does, which is especially useful when a large amount of data is transferred. With no acknowledgements whatsoever, the BGL would quickly drift into a chaotic state, so the occasional connection-oriented TCP acknowledgements are required to keep the system stable.

4.2.4 Chromium

For the purposes of this project, Chromium offers a good deal of functionality. Compared with the GLX methods, it has the sort-first rendering algorithm that is able to cut down the transferred data according to the view rendered by the node. This culling can considerably decrease the polygon data transfer requirements.

4.2.5 VR Juggler with GLX

The most straightforward way to get VR Juggler (see Section 2.7.5) software running on a cluster, is to forward the XFree86 displays to the rendering nodes over GLX. This method is especially useful when a VR Juggler-based application must be quickly ported to a PC environment. However, this method has an explicit performance drawback.

4.2.6 Cluster Juggler

The newest version of VR Juggler is equipped with specific software architecture for rendering clusters, called Cluster Juggler (Olson 2002). At the time of writing, however, this version (2.0-alpha4) was in alpha development state and did not give a good impression of the quality of the software. Cluster Juggler was tested, but using the alpha version was considered too difficult and incomplete for building the cluster upon it.

4.2.7 NetJuggler

NetJuggler (Allard et al. 2002*b*) was also tested to determine its suitability for our cluster setup. The software was successfully compiled, but we were not able to get the current version (1.0.3) running on the cluster. Since this system was developed before there was a competing solution included in the VR Juggler, it is likely that the solution will not be supported for long.

4.2.8 VR Smuggler

VR Smuggler is a small piece of software to handle the OpenGL display and context management previously done with VR Juggler (Ilmonen & Reunanen 2004). This

software is included in the FLUID package and allows us to run software that was using VR Juggler for context management, without having VR Juggler installed.

4.3 Rendering Benchmark Tests

The different rendering tests that were run in the cluster are presented here in detail. Generally, the test setups follow the tests conducted by Ilmonen et al. (2005) where applicable. This chapter presents these tests and their results.

4.3.1 Test Setups

To measure the performance of the PC cluster we constructed three test cases. The cases were chosen so that they would correspond to the actual software that would be executed on the cluster.

1. The first test setup includes a static 100,000 polygon scene. The 3D scene consists of an architectural model of a new auditorium at HUT, called *Mellin Hall*. The whole scene is loaded into display lists before rendering. A screenshot of a part of the scene is depicted in Figure 4.1

2. The second test uses the same model as the first tests, but without display lists. This test gives an example of how the system will function under a highly dynamic, constantly deforming, dataset.

3. The third test is designed to examine the functioning of the system under a heavy texture stream. The model consists of a single wall that has a constantly changing texture. A screenshot of a part of the scene is depicted in Figure 4.2

Figure 4.1: Screenshot of the 3D Scene Used in Tests 1 and 2

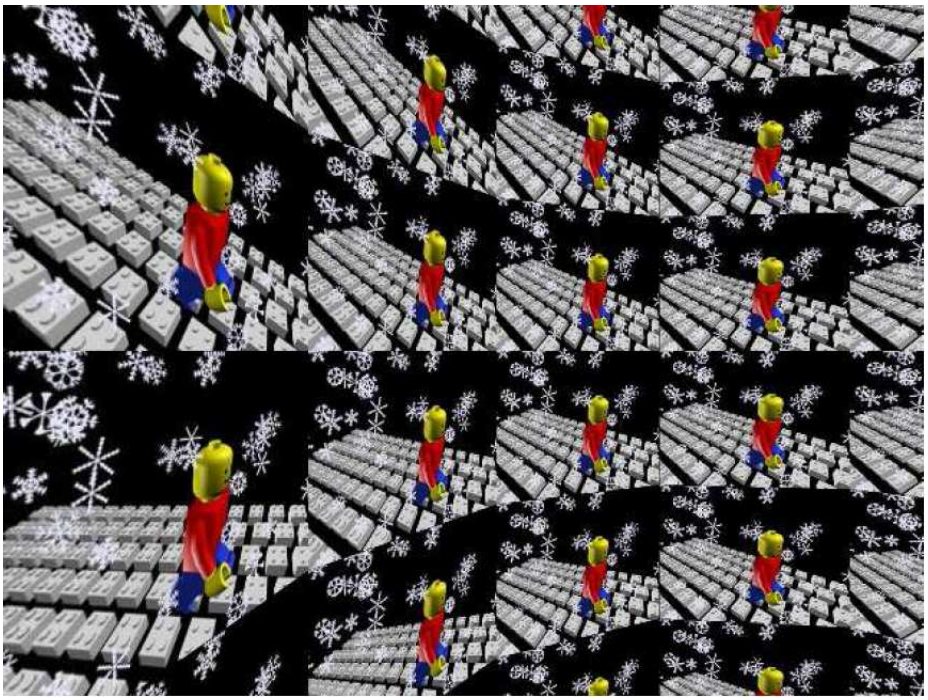

Figure 4.2: Screenshot of the 3D Scene Used in Test 3

4.3.2 Test Platforms

Each of the three benchmarks was tested with five different platforms. The first three platforms were different software setups on the PC cluster, and the last two were reference setups on the PC and SGI Onyx2 platforms. Furthermore, the three cluster tests were conducted both using a 100 Mb and a 1 Gb Ethernet networks to better evaluate their functioning in a saturated network environment. The platforms were the following:

1. Threaded GLX renderer, as presented in Section 4.2.2.
2. BGL renderer, as presented in Section 4.2.3.
3. Chromium renderer, as presented in Section 4.2.4.
4. Local mode rendering with a single node of the PC cluster.
5. Local mode rendering with the SGI Onyx2 platform presented in Section 2.3.1.

4.3.3 Test Metrics

The following metrics were monitored for each of the three test cases. In addition to these three, the picture quality was estimated visually, but there were no notable differences between the test setups.

1. Frame rate in frames per second (FPS). Frame rate was measured using a frame counter in the launcher script. However, in the case of Chromium, we used a frame counter provided with the software. The value is the average frame rate throughout the time the test was running, loading time excluded.
2. Network throughput. The load on the network was monitored by running a *GKrellM* network monitor on the application node. The value is the average network throughput measured in megabytes per second. Although the pure

network load will not give us very specific information about the functioning of the software, we can effectively compare the solutions and also calculate the data rate requirement on a per-frame basis.

3. CPU load. The load on the CPU was measured only on the application node, since all the tests were most CPU intensive on that node. Since the computer was based on a Hyper-Threading Pentium 4 CPU, it is shown as a dual-CPU SMP machine for the operating system. Even so, we decided to use the Linux utility *top* to measure the load, by monitoring the average percentage of idle CPU resources. The load was calculated by deducting this value from 100 percent.

4.4 Test Results

This section presents the results from the rendering tests covered above. The results from each of the tests are presented in a separate table.

Method	*Frame Rate (FPS)*	*CPU Load (%)*	*Network Load MB/s*
Threaded GLX (0.1 Gb)	1.7	6	0.38
Threaded GLX (1.0 Gb)	2.8	44	25
Broadcast GLX (0.1 Gb)	15	2	0.47
Broadcast GLX (1.0 Gb)	19	2	0.48
Chromium (0.1 Gb)	46	20	5.3
Chromium (1.0 Gb)	37	20	4.0
Local, PC	16	-	-
Local, SGI	1.3	-	-

Table 4.1: Comparison of the Data Transfer Methods (Test 1: Static Dataset)

4.4.1 Analysis of the Results

First of all, it is noteworthy that the PC cluster is clearly superior to the SGI Onyx2 system. In fact this is not surprising, since the Onyx2 hardware is outdated and can

Method	*Frame Rate (FPS)*	*CPU Load (%)*	*Network Load MB/s*
Threaded GLX (0.1 Gb)	0.10	10	12
Threaded GLX (1.0 Gb)	0.87	70	95
Broadcast GLX (0.1 Gb)	0.82	2	12
Broadcast GLX (1.0 Gb)	4.2	35	55
Chromium (0.1 Gb)	0.32	10	12
Chromium (1.0 Gb)	2.4	45	87
Local, PC	24	-	-
Local, SGI	1.8	-	-

Table 4.2: Comparison of the Data Transfer Methods (Test 2: Dynamic Dataset)

Method	*Frame Rate (FPS)*	*CPU Load (%)*	*Network Load MB/s*
Threaded GLX (0.1 Gb)	3.1	6	6.4
Threaded GLX (1.0 Gb)	7.5	6	7.5
Broadcast GLX (0.1 Gb)	24	5	9.0
Broadcast GLX (1.0 Gb)	105	18	29
Chromium (0.1 Gb)	1.8	8	11
Chromium (1.0 Gb)	10	24	46
Local, PC	150	-	-
Local, SGI	20	-	-

Table 4.3: Comparison of the Data Transfer Methods (Test 3: Texture Stream)

hardly be compared with the modern-day rendering hardware. However, the used PC cluster did not have SoftGenLock running during the tests. While SoftGenLock currently consumes more CPU cycles than would be necessary, it was considered best to leave it out of the tests. If it would work correctly, having it running would only marginally affect the results.

In the first test (see Table 4.1) with a static polygon set, BGL delivered a high frame rate with the lowest network and CPU load. Chromium differs from the two other platforms in that it has a special system for culling the unneeded polygons from the data sent to the nodes. This turned out to give Chromium an advantage especially in the first test, where the measured FPS even exceeded that of the locally rendered benchmark on a single PC.

In the second test (see Table 4.2) without display lists, BGL reached the best frame rates with lowest network and CPU load. The culling ability used in Chromium lowered the network load only marginally so both GLX and Chromium reached close to the saturation level of even the 1 Gb Ethernet.

In the third test (see Table 4.3), BGL performed best as expected. The fact that both GLX and Chromium had to send the texture data twice to all of the four rendering nodes slowed them down considerably. The performance of the GLX solution was surprisingly low, especially in this test. There was still unused bandwidth in the network, but that obviously was not the bottleneck.

4.5 EVE Software Architecture Decision

According to the results of the tests, the BGL-based solution is highly competitive compared with any one of the tested platforms. Figure 4.3 depicts the main components of the chosen software architecture and their main fuctions. This model should be especially compared with the current Onyx2-based architecture presented in Section 2.3.2.

If we think of the software-related objectives presented in Section 1.2, we can state that the objectives have been fulfilled reasonably well. The first objective was to have a compact software environment. The new architecture has only one possible path for programming the applications, so there is no confusion between choosing the most suitable way. Also, since the functions provided by VR Juggler were mostly unused in the old system, it simplifies the software environment in that VR Juggler is left out of the design. Furthermore, the fact that there is no longer a separate controller level, makes the architecture more compact and easier to understand.

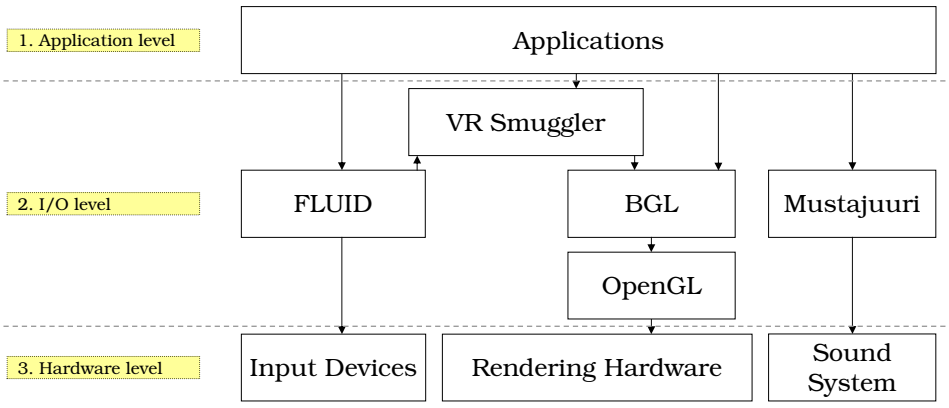

Figure 4.3: EVE PC Cluster Software Architecture

The second objective was to fully utilize the rendering hardware. According to the results of the tests, the BGL-based solution looks very promising and appears to be an optimal solution in the consumption of the network resources.

The third objective was to create a user-friendly API for graphics calls and external libraries. By choosing FLUID as the base library to control the input devices, we allow the users to work on a platform that most of them are already familiar with. In addition, using the VR Smuggler library with FLUID, allows the users to utilize the methods of OpenGL context management that were used in VR Juggler. Mustajuuri uses the sound system in the same way as in the old architecture, so the users are already familiar with the setup.

Chapter 5

Results and Discussion

This chapter presents the challenges we encountered in the project, and gives a glimpse to the future and the status of the project in general.

5.1 Problems

This section lists some issues we had when constructing the PC cluster, and also offers a solution that was implemented to overcome the problem.

5.1.1 Kernel Instability

After first installing the real-time kernel, we quickly noticed that it was not very stable under a CPU-intensive stress test. The same problem occurred with all the tested RTAI kernel versions. In a sense this is understandable, as the kernel allocates the CPU time where it is needed, but it should still be possible to successfully run a stress test, at least when there are no real-time processes running.

This problem remains unsolved. However, in its current use in the EVE, there have been no software crashes caused by this instability.

5.1.2 SoftGenLock as an IRQ-based Kernel Module

SoftGenLock's IRQ-based solution was, according to the developers, the most stable way to run the software. For some reason we were unable to get the system running in the desired manner. The consumption of CPU cycles is likely a result of a deficiency in the program. This causes the kernel module to constantly send interrupt requests, rather than only when needed. Until this problem is fixed, we are using the `sgl_user` program to run SoftGenLock.

As SoftGenLock is chosen as the method for maintaining the active stereo, getting this mode to work properly is necessary to guarantee the future use of the system. At least two methods should still be tried to do this. The first is to migrate back to the SoftGenLock version 1.0. This is basically the same system that is known to work in implementations by Vorozcovs (2002) or Maxwell et al. (2002). The second method is to try to migrate into newer real-time kernels. Especially the timer-based kernel module should be more stable on a 2.6 version Linux kernel. This solution would require someone to port the whole SoftGenLock into the new environment, which is not a trivial task, since the software works as a kernel module and uses version 2.4 specific operations.

5.1.3 NvAGP

NvAGP is the part of the NVIDIA Linux driver that supports the access through the Accelerated Graphics Port. The use of NvAGP is mutually exclusive to the use of *agpgart*, the Linux kernel module designed for the same purpose.

We were under the impression that agpgart would work the same as NvAGP or even better, but it was discovered that running XFree86 with NvAGP option enabled, gave a 30 percent performance increase compared to the agpgart option.

5.1.4 Projector Instability

After the SoftGenLock was first tested in the EVE with the projectors running at 120 Hz, we encountered a problem with the refresh synchronization, where the top of the image became overly distorted. At first, we assumed that the problem might be caused by faulty video cables, or the video switcher. However this was not the case, since the problem occurs regardless of these components. The conclusion is that this defect appears due to the software-generated genlock system not working as intended.

5.1.5 Synchronization Signal Strength

When the PC cluster was connected to the StereoGraphics emitter system in the EVE, we noticed that the signal from the parallel port was too weak to run the whole system properly. The infrared (IR) -operated emitters could send the signal to the glasses only when a single one of the 12 emitters was connected. Two possible solutions were suggested: to build an amplifier between the parallel port and the emitters, or to acquire a separate emitter system that could utilize the weak signal. Since a separate emitter was available for a potentially low cost, the second alternative was chosen, although such system was still not in use in January of 2005.

5.1.6 XFree86 ModeLines

A ModeLine is a monitor-specific configuration option in the XFree86 windowing system. Configuration describe the physical qualities of the monitor and, for exam-

ple, the highest available vertical refresh rate for a given resolution. When setting up a genlocked cluster, it is important that all the monitors are running with the same vertical refresh rate. In a heterogenic environment, all the systems must be set according to the lowest refresh rate. This became a problem in out test cluster, where the maximum refresh rate on one of the standard CRT displays was 85 Hz. This refresh rate also results in an uncomfortable single-eye frequency of 42.5 Hz, which prevents the system to be used for extended periods of time.

In other words, it is highly recommended that all the displays are identical. Having several kinds of displays only causes confusion and the better displays, with a wider range of refresh rates or resolutions, will not be fully utilized.

5.1.7 3COM Gigabit Ethernet Linux Support

At a certain time in building our test cluster we had one node equipped with ASUS P4P800 Deluxe main board with a 3COM Gigabit Ethernet 3C940 LAN on Motherboard (LOM). This NIC worked well with the *sk98lin* Linux driver until it was tested with a BGL cluster. In this environment, the node affected the performance of the whole cluster by constantly stalling the rendering for duration of up to half a second. We were unable to find a solution to this problem and therefore our cluster now uses Intel Gigabit Ethernet Adapters utilizing the *e1000* Linux driver.

5.2 Further Work

The rapid development of commodity hardware is likely to continue also in the future. One of the objectives of this project was to create a hardware and software platform that is easy and affordable to upgrade when required. With the current price level of genlock-enabled display adapters, the option of migrating into using them is still quite far from affordable. Even so, it is worthwhile to constantly follow

the development of rendering hardware. If the price difference between the software- and hardware-genlocked systems becomes small enough, the migration should be taken into consideration.

Furthermore, this study was almost entirely focused on building a replacement for the active stereo environment at the EVE. This was also a natural choice, since the projectors were already installed and running, but it effectively ruled out the possibility of constructing a passive stereo environment. Nevertheless, most of the problems in this work were related to the active stereo synchronization, not to mention the bulky shutter glasses and the loss of CPU cycles caused by SoftGen-Lock. When the time comes, the option of migrating into a passive stereo should be strongly considered. That would, however, require replacing the projection surfaces with ones that preserve the polarization of light. The investment and upkeep required for new projection surfaces, a set of eight medium quality projectors with circular polarized filters and similar glasses is expected to be lower than the cost of four 120 Hz, fast phosphorus CRT projectors and fragile shutter glasses.

Chapter 6

Conclusions

As discussed earlier, the new system either fully or partially meets all of the objectives presented in Section 1.2. This does not mean that there would not be open ended questions related to the installation. The existing SGI Onyx2 system has served well for nearly a decade and it will be a difficult task for the users to migrate into the new system. The test results presented in Section 4.4 look promising, and give us hope that the system will meet the constantly growing performance requirements of VR applications.

Although our new PC cluster system can be considered a working implementation, there are hardly any well-documented reference implementations in the world. This is mainly due to the following facts:

1. The research groups studying VR generally have a substantial hardware budget. The price of the high-end hardware has also come down, and the production systems need to offer high-performance and a high rate of availability.

2. The cost of the hardware is only a small part of a high-end VR installation. Therefore, it is easy to budget high-end rendering hardware to the same investment. Especially active stereo environments, such as CAVE-like systems, can only be built and kept up by large companies and institutions.

3. Porting all the software from one platform to another is a considerable economic burden. Therefore, it is usually preferable to purchase a backwards-compatible system.

Particularly the active stereo environment becomes a burden when designing a system with commodity hardware. For example Olson (2002) considers the SoftGenLock implementation unstable and incomplete for production use. On the other hand, Maxwell et al. (2002) consider the system most promising and reports that the tests with it have been successful, although until today there have been no further reports on the system or its usage.

Bibliography

Allard, J., Gouranton, V., Lamarque, G., Melin, E. & Raffin, B. (2003), SoftGenLock: active stereo and genlock for PC cluster, *in* 'EGVE '03: Proceedings of the workshop on Virtual environments 2003', ACM Press, pp. 255–260.

Allard, J., Gouranton, V., Lecointre, L., Melin, E. & Raffin, B. (2002*a*), *Getting Started with Net Juggler and SoftGenLock*, Laboratoire d'Informatique Fondamentale d'Orléans (LIFO), Orleans, France.

Allard, J., Gouranton, V., Lecointre, L., Melin, E. & Raffin, B. (2002*b*), Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Component Cluster, *in* 'Proceedings of the IEEE VR', Orlando, Florida.

Boden, N. J., Cohen, D., Felderman, R. E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N. & Su, W.-K. (1995), 'Myrinet: A gigabit-per-second local area network', *IEEE Micro* **15**(1), 29–36.
*http://citeseer.ist.psu.edu/boden95myrinet.html

Burdea, G. C. & Coiffet, P. (2003), *Virtual Reality Technology*, Wiley-Interscience.

Cruz-Neira, C., Bierbaum, A., Hartling, P., Just, C. & Meinert, K. (2002), VR Juggler - An Open Source Platform for Virtual Reality Applications, *in* 'Proceedings of the 40th AIAA Aerospace Sciences Meeting and Exhibit', Reno, Nevada.

Cruz-Neira, C., Sandin, D. J. & DeFanti, T. A. (1993), Surround-Screen Projection-Based Virtual Reality: The design and Implementation of the CAVE, *in* J. T. Kajiya, ed., 'Proceedings of ACM SIGGRAPH 1993', Vol. 27, ACM SIGGRAPH, ACM Press, Anaheim, California, pp. 135–142.

Eckel, G., Jones, K. & Domeier, T. (2004), 'OpenGL Performer Getting Started Guide', Web reference.

Forum MPI (1994), MPI: A Message-Passing Interface, Technical report.

Humpreys, G. (2004), *Chromium Documentation Version 1.7*, Stanford University. Retrieved November 25, 2004.
*http://chromium.sourceforge.net/doc/index.html

Humpreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D. & Klosowski, J. T. (2002), Chromium: A Stream Processing Framework for Interactive Rendering on Clusters, *in* 'Proceedings of ACM SIGGRAPH 2002', ACM SIGGRAPH, ACM Press.

Ilmonen, T. (2001), Mustajuuri - An Application and Toolkit for Interactive Audio Processing, *in* 'Proceedings of the 7th International Conference on Auditory Displays', Helsinki, Finland, pp. 284–285.

Ilmonen, T. & Kontkanen, J. (2002), Software Architecture for Multimodal User Input – FLUID, *in* 'The Proceedings of the 7th ERCIM Workshop', Chantilly, France, pp. 223–242.

Ilmonen, T. & Reunanen, M. (2004), *HELMA Technical Overview*. Retrieved January 26, 2005.
*http://www.tml.hut.fi/Research/HELMA/download/helma-tech.pdf

Ilmonen, T., Reunanen, M. & Kontio, P. (2005), Broadcast GL: An Alternative Method for Distributing OpenGL API Calls to Multiple Rendering Slaves, *in* 'The Journal of WSCG'2005', Vol. 13, Plzen, Czech Republic.

Intel Inc. (2005), 'Moore's Law: Overview', Web reference. Retrieved January 14, 2004.
*http://www.intel.com/research/silicon/mooreslaw.htm

Jalkanen, J. (2000), 'Building a spatially immersive display - HUTCAVE'.

Kalawsky, R. S. (1993), *The Science of Virtual Reality and Virtual Environments*, Addison Wesley 1993.

Kilgard, M. J. (1994), 'OpenGL and X', Web reference. Retrieved September 2, 2004.
*http://www.sgi.com/products/software/opengl/glandx/intro/intro.html

Maxwell, D. B., Bryden, A., Schmidt, G. S., Roth, I. & Swan, J. E. (2002), Integration of Commodity Cluster into an Existing 4-Wall Display System, *in* 'Proceedings of Workshop on Commodity-Based Visualization Clusters, IEEE Visualization 2002', Boston, Massachusetts.

Monarch Computer.com (2004), 'PNY Quadro FX 3000G Genlock 256MB 8X4X AGP', Web reference. Retrieved September 23, 2004.
*http://www.monarchcomputer.com/Merchant2/merchant.mv?Screen=PROD &Product_Code=190634

Moore, G. E. (1965), Cramming More Components onto Integrated Circuits, *in* 'Electronics', pp. 114–117.

Myricom (2004), 'Myrinet Product Information', Web reference. Retrieved September 9, 2004.
*http://www.myri.com/myrinet/

NVIDIA.com (2004*a*), 'Genlock & Frame Lock Solutions', Web reference. Retrieved September 23, 2004.
*http://www.nvidia.com/page/quadrofx_3000g.html

NVIDIA.com (2004*b*), 'Linux Display Driver - IA32', Web reference. Retrieved September 23, 2004.
*http://www.nvidia.com/object/linux_display_ia32_1.0-6106.html

Olson, E. C. (2002), Cluster Juggler - PC cluster virtual reality, Master's thesis, Iowa State University.

Pape, D. (1997), *pfCAVE CAVE/Performer Library Manual*. Retrieved January 31, 2005.
*http://www.evl.uic.edu/pape/CAVE/prog/pfCAVE.manual.html

Raffin, B. (2002), *SoftGenLock Manual: Software Active Stereo and Genlock for PC Cluster*, Laboratoire Informatique et Distribution, Grenoble, France.

Reardon, M. (2004), '10 Gigabit Ethernet comes alive', Web reference. Retrieved January 25, 2005.
*http://news.com.com/10-Gigabit+Ethernet+comes+alive/2100-1035_3-5173226.html

Reiners, D., Voß, G. & Behr, J. (2002), OpenSG: Basic Concepts, *in* 'OpenSG Symposium'. Retrieved December 15, 2004.
*http://www.opensg.com/OpenSGPLUS/symposium/Papers2002/Reiners_Basics.pdf

Sherman, W. R. & Craig, A. B. (2004), *Understanding Virtual Reality - Interface, Application and Design*, Morgan Kaufmann.

Silicon Graphics Inc. (2003), 'Silicon Graphics Onyx4 UltimateVision Family Datasheet', Web reference. Retrieved December 15, 2004.
*http://www.sgi.com/pdfs/3501.pdf

Silicon Graphics Inc. (2004), 'Silicon Graphics Prism Product Overview', Web reference. Retrieved December 15, 2004.
*http://www.sgi.com/products/visualization/prism/overview.html

Tullsen, D. M., Eggers, S. J. & Levy, H. M. (1995), Simultaneous Multithreading: Maximizing On-Chip Parallelism, *in* '22nd Annual International Symposium on Computer Architecture', pp. 392–403.

Vorozcovs, A. (2002), 'Synchronized video output from multiple computers with RTLinux', Web reference. Retrieved January 25, 2005.
*http://www.cs.yorku.ca/ av/Genlock.htm

Womack, P. & Leech, J. (1998), 'Opengl graphics with the x window system'.

Woods, A. J. (2001), Optimal usage of LCD projectors for polarized stereoscopic projection, *in* 'Proc. SPIE Vol. 4297, p. 5-7, Stereoscopic Displays and Virtual Reality Systems VIII, Andrew J. Woods; Mark T. Bolas; John O. Merritt; Stephen A. Benton; Eds.', pp. 5–7.

Appendix A

Startup Scripts

This chapter presents the scripts that were constructed to effectively run the cluster in the EVE. Generally, all the scripts are started from the application PC and they use Secure Shell (SSH) to call the necessary items on the remote nodes.

sgl_user_start.sh

This script starts the `sgl_user` SoftGenLock process on the master node and on the three slave nodes.

```
#!/bin/sh
echo "Starting sgl_user on master node..."
ssh render1 'pkill sgl_user; /usr/src/softgenlock2.0a3/bin/sgl_user -fifo
    < /usr/src/softgenlock2.0a3/scripts/eve-master-old
    >> /var/log/sgl.log 2>&1' &

for i in `seq 2 4`; do
echo "Starting sgl_user on slave node $i..."
ssh render$i 'pkill sgl_user; /usr/src/softgenlock2.0a3/bin/sgl_user -fifo
    < /usr/src/softgenlock2.0a3/scripts/eve-slave-old
    >> /var/log/sgl.log 2>&1' &
done
```

sgl_user_stop.sh

This script stops the `sgl_user` SoftGenLock process on all the nodes.

```
#!/bin/sh
for i in ‘seq 1 4‘; do
echo "Stopping sgl_user on node $i..."
ssh -nq render$i ’killall sgl_user’
done
```

bgl_start.sh

This script starts the BGL decoders on each of the four rendering nodes.

```
#!/bin/sh
SENTINEL=7
while [ -n "$1" ];
do
  SENTINEL="$1"
  shift
done
for i in ‘seq 0 $SENTINEL‘; do
  echo "Starting bgl on render node $i..."
  j=$[i/2+1]
  ssh render$j "cd /usr/src/bgl; source env.sh; export DISPLAY=:0;
                hostname; ./bgl_decode --id $i 10001" &
done
```

bgl_kill.sh

This script stops the BGL decoders on each of the four rendering nodes.

```
#!/bin/sh
for i in ‘seq 1 4‘; do
echo "Killing bgl on render node $i..."
  ssh render$i ’cd /usr/src/bgl; make kill;’
done
```

Appendix B

EVE Applications

Applications and Demos

The following table (Table B.1) contains most of the applications that are used in the current system. In addition, there is a suggested action to determine how this software should be ported to the new system. The priority (Pr.) is given on a scale of 1 to 5, value 1 being the most important to get quickly working also in the new system.

Name	*Used Software*	*Pr.*	*Suggested Action*
Model viewer	FLUID	1	Since the 3D models (see Table B.2) cannot be viewed with EPIC or VR Juggler, a new viewer must be constructed.
HELMA	VR Juggler + FLUID	1	Conversion to Smuggler.
ALMA	VR Juggler + FLUID	1	Conversion to Smuggler.
Lumisota	VR Juggler + FLUID	2	Conversion to Smuggler.
Akvaario	VR Juggler + FLUID	3	Conversion to Smuggler.
Animaland	VR Juggler + FLUID	3	Conversion to Smuggler.
VR student works	VR Juggler + FLUID + Performer	4	The ones using Performer will not be ported. The best of the others can be converted to use Smuggler.
Cave Quake III	VR Juggler	5	No conversion.

Table B.1: List of Different Applications in the EVE

3D Models

The following table (Table B.2) contains most of the models that are used in EVE applications. The table also shows the source format and the suggested action. The priority (Pr.) is given on a scale of 1 to 5, value 1 being the most important models to get converted into a format that is usable in the new system.

Name	*Format*	*Pr.*	*Suggested Action*
Sali600	OpenFlight 14.2	1	Conversion to Wavefront.
TUAS	OpenFlight 14.2	1	Conversion to Wavefront.
Scandic	OpenFlight 14.2	1	Conversion to Wavefront.
Toimisto	OpenFlight 14.2	2	Conversion to Wavefront.
Avia Forum	OpenFlight 14.2	3	Conversion to Wavefront.
Pfizer	OpenFlight 14.2	3	Conversion to Wavefront.
NY Paviljonki	OpenFlight 14.2	3	Conversion to Wavefront.
Molekyyli	OpenInventor	4	No methods available to convert.
Julkisivu	OpenInventor	5	No methods available to convert.
Town	Performer (unknown)	5	No conversion.

Table B.2: List of Different 3D Models Used in the EVE

Appendix C

Details of Threaded GLX Implementation

```
#include <stdlib.h>
#include "glx_wrapper.h"
#include <GL/gl.h>
#include <pthread.h>

/* number of windows */
#define N 8

volatile int quit=0,rendering,restart;

pthread_mutex_t mutex,mutex2;
pthread_cond_t  cond,cond2;

/* method for drawing the glx context */
void *draw(void *yes);

int main(void)
{
  int n;
  GLX square[N];
  char *names[]={"10.0.0.1:0","10.0.0.1:0","10.0.0.2:0","10.0.0.2:0",
 "10.0.0.3:0","10.0.0.3:0","10.0.0.4:0","10.0.0.4:0"};
  int  flags[]={GLX_REMOTE,GLX_REMOTE,GLX_REMOTE,GLX_REMOTE,
GLX_REMOTE,GLX_REMOTE,GLX_REMOTE,GLX_REMOTE};
  /* coordinate of the left edge of the X windows */
  int origins_x[]={0, 1024, 0, 1024, 0, 1024, 0, 1024};

  pthread_t       thread[N];
  pthread_attr_t  attr[N];
```

```
  pthread_mutex_init(&mutex,NULL);
  pthread_mutex_init(&mutex2,NULL);
  pthread_cond_init(&cond,NULL);
  pthread_cond_init(&cond2,NULL);

  rendering=N;
  restart=N;
  for(n=0;n<N;n++)
  {
    GLX_Open(&square[n],names[n],origins_x[n],0,1024,768,flags[n]|GLX_NODECOR);

    pthread_attr_init(&attr[n]);
    pthread_create(&thread[n],&attr[n],draw,&square[n]);
  }

  while(!quit)
  {
    pthread_mutex_lock(&mutex);
    pthread_cond_wait(&cond,&mutex);
    pthread_mutex_unlock(&mutex);

    pthread_mutex_lock(&mutex2);
    restart=N;
    pthread_cond_broadcast(&cond2);
    pthread_mutex_unlock(&mutex2);
  }

  pthread_mutex_lock(&mutex);
  pthread_cond_wait(&cond,&mutex);
  pthread_mutex_unlock(&mutex);

  sleep(1);

  pthread_mutex_lock(&mutex2);
  restart=N*3;
  pthread_cond_broadcast(&cond2);
  pthread_mutex_unlock(&mutex2);

  for(n=0;n<N;n++)
  {
    pthread_join(thread[n],NULL);
    GLX_Close(&square[n]);
  }
  return(0);
}
```

Appendix D

Photos from the Project

The Figure D.1 shows the PC cluster installation in the EVE. The application PC is in the white casing and the four rendering PCs in the black casings. The connection box for the parallel synchronization network is mounted on top of the three renderers.

Figure D.1: PC Cluster Installation in the EVE

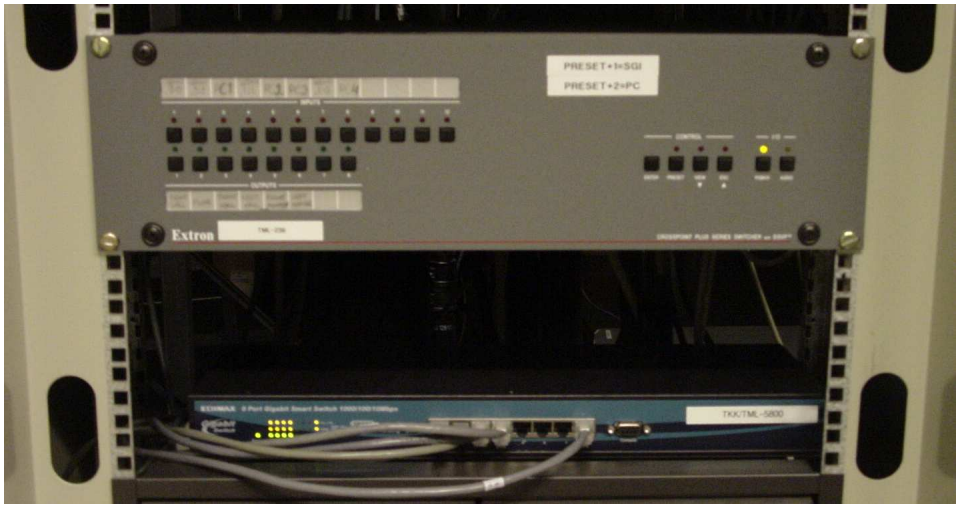

Figure D.2: Video Switcher and Gigabit Ethernet Switch Mounted to a Rack in the EVE

The Figure D.2 shows the connections inside the rack that is also visible in the previous photo. All the video outputs are connected to the video switcher (the device on the top). The device on the bottom of the photo is the Gigabit Ethernet Switch Edimax ES-5800R+.