Bachelor Thesis	2009

CAVE Rendering Framework

Thesis Report

Division	CPVR
Authors	Brigitte Hulliger
	Jonas Walti
	Stefan Broder
Supervisors	Prof. U. Künzler
	Michael Luggen
Expert	H. van der Kleij
Version	1.0

Date	Version	Name	Comment
05.06.2009	0.8	all	First draft for expert H. van der Kleij
09.06.2009	0.9	all	Draft for Supervisors
12.06.2009	1.0	all	Last corrections for delivery

Table 0.1: History

Contents

List of Figures

List of Tables

Abstract

Virtual Reality (VR) installations are often limited in rendering performance or flexibility. A common approach to avoid the first problem is the combination of multiple workstations to a cluster. There are several existing software solutions to make usage of such clusters. The technologies applied in the Cave Automatic Virtual Environment (CAVE) of the Computer Perception & Virtual Reality (CPVR) research group are Chromium which is very network intense or a VRML/X3D based solution which does not allow the programmer to take full advantage of VR possibilities.

The goal of this project was to find a solution that enables both the flexible parallel rendering of an application and the unlimited development of OpenGL applications.
A valuable parallel rendering API could be found in *Equalizer* and OpenSceneGraph (OSG) is a high performance 3D graphics toolkit which allows the development of feature-rich OpenGL applications.

The CAVE Rendering Framework (CRF) provides a library to render OSG applications with Equalizer without further knowledge about parallel rendering or Equalizer. It is compatible with most Equalizer configurations and supports most of the OSG features.

1 Introduction

Over the past sixteen weeks we worked on a project within the scope of our bachelor thesis. The thesis report shows the reader what we made when we began and how we organised ourselves. Furthermore, we bring up what preparation work needed to be done, how we advanced during the project and how we tested our results. Finally, we point out unachieved goals and summarise an outlook on what could be done in the future.

1.1 Initial System Setup

1.2 CAVE Hardware

The CPVR research group of the Berner Fachhochschule - Technik und Informatik (BFH-TI) runs a CAVE. A CAVE is an immersive virtual reality environment. Currently, there are three screens arranged in an open cube geometry where each screen measures about 2x2 meters. Figure **??** shows the mentioned CAVE.

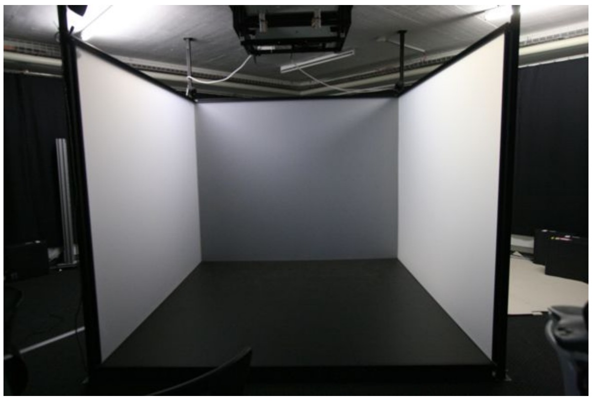


Figure 1.1: CAVE BFH-TI

Two projectors are installed on each side of the cube pointing to the particular screen via a mirror. In front of the lens of each projector is one polarization filter fixed, one for left circular polarisation and one for right circular polarisation for each projector pair on each side. The projections on the screen have to match exactly in order to get a proper alignment. A projector pair is shown in Figure **??**. One such pair is installed on each side.

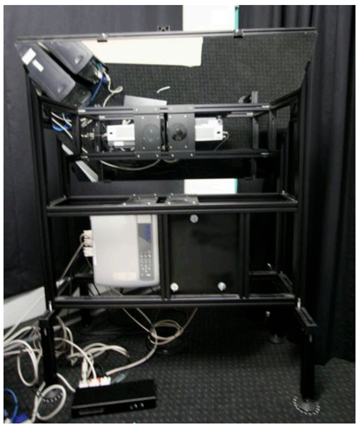


Figure 1.2: CAVE projectors

There are two additional projectors fixed at the ceiling. They are planned to project to the floor, which is going to increase the immersion of the whole installation as soon as the calibration is done.

The CAVE operates with six render clients and one server. Each client is connected to one of the projectors. The server is responsible to coordinate the different clients. A diagram of this setup is shown in Figure **??**.

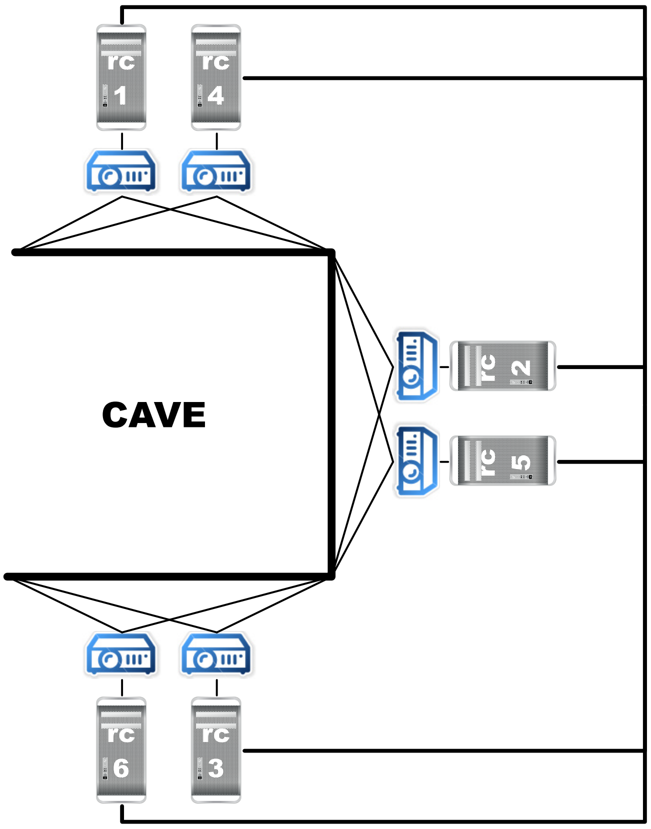


Figure 1.3: CAVE Hardware Setup

For rendering a three dimensional scene in the CAVE, the two clients on each side have to render the same part of the scene so one has the same part of the scene twice on each screen. The two projections on each screen have to be rendered with a slightly different position of the camera in order to get a stereoscopic image. The spatial difference has to configured in the software rather physically and should equal the distance between the right and the left eye of the spectator.

To get the three dimensional feeling in the CAVE, one has to wear polarisation glasses (see Figure **??**). The effect of these glasses is, that one can only see one image per eye on each wall. This leads to a three dimensional stereoscopic effect.

Figure 1.4: Polarisation glasses

1.3 Current CAVE Software Solution

At the moment, there are two given software architectures in use for the CAVE environment. One is using the Chromium Library for network distributed OpenGL rendering. For the other one, a distributed VRML/X3D solution is used. The latter uses only minimal network resources, but is limited to VRML/X3D implemented applications.

1.4 Chromium

One of the currently used frameworks in the BFH-TI CAVE is Chromium**?**. Chromium is a flexible framework for scalable real-time rendering on clusters of workstations, derived from the Stanford WireGL project code base. Details concerning the differences between Chromium and Equalizer can be read in the Technical Documentation. We only describe the setup used in the CAVE of the CPVR research group.

To avoid further terminology misunderstandings we first want to describe the terms client and server in coherence with the CAVE.

client
A client refers to applications which use some servers rendering capabilities.

server
A server refers to services which do the actual rendering.

Considering this terminology the CPVR CAVE consists of seven computers where one of them is the application server and the other six represent the rendering clients. Each of the rendering clients is connected to a projector. There are two projectors for each screen as described in the section above (**??**).

In the setup used by Chromium, the application server takes the role of a client and the rendering clients act as servers. This means there are six servers and one client in the setup. Chromium leads to a high workload of network traffic.

1.5 VRML/X3D

The other setup used in the CPVR CAVE is based on Virtual Reality Modeling Language (VRML) and X3D.

VRML?
VRML is a standard file format for the representation of 3D vector graphics that may be interactive.

X3D?
X3D is the ISO standard XML based file format used to represent 3D computer graphics. It is the successor of VRML. X3D features extensions to VRML.

This setup is much less network demanding but not less problematic. All applications have to be implemented in VRML/X3D. This is a big restriction and allows mostly self proprietary developed applications.

1.6 Requirements for the new solution

In the previous section the two currently running setups were described. Both of them have major disadvantages: Chromium is too network intense and VRML/X3D is restricted to a very limited number of applications. Therefore the goal of a new approach was to get a setup that supports distributed rendering without a high network usage and allows to run off-the-rack applications or at least supports a framework that is more common and less restricted that VRML/X3D.

2 Goals

A detailed goal description can be found in our functional specification document in the appendix. We defined our goals either primary or secondary.

Goal	**Achieved**
Integrate a Scene Graph in Equalizer	✔
Stereoscope Output	✔
Based on OpenGL	✔
Performant Scene Graph Rendering	✔
Extensibility	✔
Configurability	✔

Table 2.1: Primary goals

Goal	**Achieved**
Volume Rendering	✖
Haptics	✖
Shader	✔
Physics	(✔)
Tracking	✖

Table 2.2: Secondary goals

2.1 Achieved Goals

We integrated OSG successfully in Equalizer. OSG applications run flawless on top of Equalizer. Stereoscopic Output can be enabled adjusting the Equalizer configuration. OSG is a Open Graphics Library (OpenGL) based library. At least on the approach with separate rendering clients, we reached a performant scene graph rendering. More on that in the testing chapter (**??**). Furthermore, we provide different Equalizer configurations which can be used for desired topology. The user can chose between a predefined one node configuration with multiple graphic cards and a configuration with different separate rendering clients. Then, the GL Shading Language (GLSL) is fully supported by OSG. As well, there are some basic physics like collision detection that can be done in OSG.

2.2 Unachieved Goals

Unfortunately, there was not enough time to make efforts concerning Volume Rendering, Haptics or Tracking.

3 Prerequisits

3.1 Eurographics 2009

After reading a lot of papers about Parallel Rendering, Equalizer, OSG, Object-Oriented Graphics Rendering Engine (OGRE), etc. we got in contact with some people in the Equalizer community. The fact that the developer of Equalizer, Stefan Eilemann, was already known to the CPVR research group guided some ways. We knew there was already some work in progress concerning the combination of a scene graph with Equalizer, meaning OSG as well as OGRE. We thought, it was not a bad idea to ask him about his opinion, since the required features for our work were fulfilled in both of the frameworks. The response we got was better than we could have hoped: He named us a group of students from the University of Siegen, Germany, who were working on an integration of OSG in Equalizer, and he declared himself ready to comply with introducing us to them. Better yet, he suggested we should come to the upcoming conference *Eurographics 2009* taking place in Munich only three weeks later. He would be there having a Birds-Of-a-Feather meeting. The people from the University of Siegen would be there, too. After some discussions, we decided to go there for at least two days. The meeting was on Tuesday with the option of another one on Wednesday if there should be more to discuss. Besides the meeting itself, there were a lot of other interesting events, mostly paper sessions. The Birds-Of-a-Feather (BOF) meeting was a success, there were a lot more people than expected and a lot of them showed their interests in an integration of OSG in Equalizer. There were quite a lot of people from Siegen and we were introduced to the developers and exchanged contact information. Another group, some PHD students from the University of Zurich were present too and it turned out, that they were interested in the project too.

3.2 CAVE Clone

Because of limited CAVE resources and some student overpopulation we needed to setup an environment similar to the CAVE in order to test our framework. To setup a clone of the CAVE, we needed at least one workstation with enough graphics cards so that we could connect four displays to simulate each CAVE wall. This was enough for a first setup, but we had to keep in mind, that we would have to connect eight projectors afterwards. Therefore, it seemed to be a good start, if we tried it with four graphics cards. Since we were going to render quite a lot of graphics stuff, it could not harm to have a few gigabites of memory and a Quadcore Central Processing Unit (CPU). Our framework should run cross-platform at least under Windows XP and a Linux distribution, therefore we needed a harddisk with enough free space for multiple operating systems. By then, we had a table with the following components on it:

- Mainboard
- CPU
- 16 GB Ram
- 500 GB Harddisk
- 4 x NVidia Quadro FX 1700
- 4 Displays

At the beginning, we wanted to reuse an old computer case which we got from the BFH-TI IT Support. But soon, we had to realise, that the power supply unit was too weak for the brand-new mainboard. We ordered a new one which consequently did not fit into the old computer case. Hence, we had to buy a new computer case as well. The result of our CAVE clone can be seen in Figure **??**.

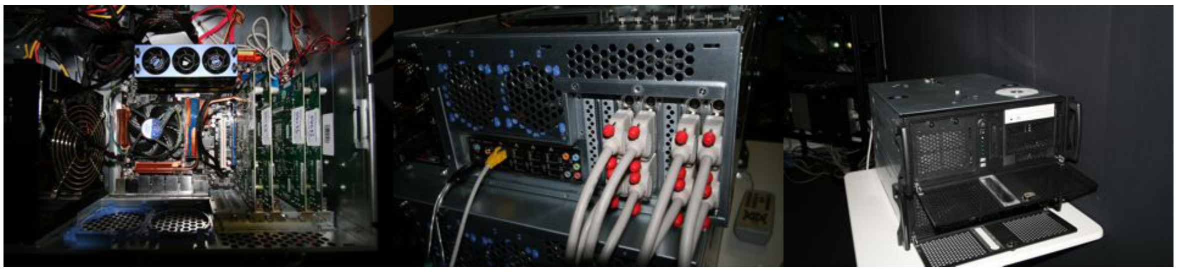


Figure 3.1: Cave Clone

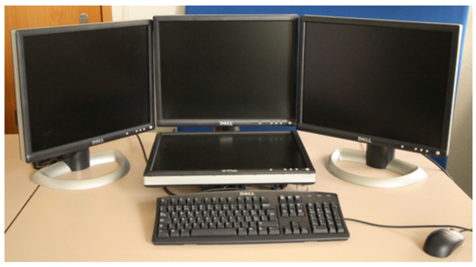


Figure 3.2: Display setup for the CAVE clone

The 4 displays were arranged in a small cube to simulate the CAVE environment as shown in figure **??**.
After screwing it together, we decided to install a dualboot with Windows Vista and Ubuntu 8.10 on it.

3.2.1 Linux Setup

Detecting the four displays did work right away. We were surprised by that, only a few years earlier, it was quite a challenge to setup more than one display in a linux. The biggest problem was to arrange the four displays in the right order with the right resolution. It pointed out that the NVIDIA driver somhow mixed up the xorg settings of the resolution with some intern stuff. It seemed to be a bug.
Setting up OSG on Linux was a challenge at first. There are quite a lot of dependencies you have to look after. And soon we realized, that the website of OSG contains a lot of information, but not really organised, so finding the right dependencies on the list was not that easy. Furthermore, the OSG version in the aptitude tree of Ubuntu was really old and therefore not usable for our purpose. Finally, we got it running.
To install Equalizer was not that difficult. After about an hour, we could test the first Equalizer application with the default configuration.
Since Equalizer comes with a few sample configurations - amongst other some for multiple

pipes - we tried to setup a configuration with four pipes[1], for each display one pipe. We could even say which pipe to use for which display. So, it seemed we had a running linux setup.

3.2.2 Windows Vista Setup

Detecting multiple displays in a Windows system is done fast. Therefore, we could straight go on to install OSG and Equalizer. After we got rid of the Windows Vista security settings which ask you for everything before doing it and consequently skips installing something in the Windows system directories, we could install OSG and Equalizer without any further problems. The test application of Equalizer ran without any problems. But as soon as we tried our new configuration file which specified four different pipes, the operating system crashed. We figured out that we could run an application on all four displays, as long as we did not specify any pipes. But it seemed as if all the rendering was done on the CPU instead of the GPUs and therefore the application ran on a terrible framerate. After a few hours, we tried to use only one pipe for all four displays. It appeared that this worked and the rendering was no longer done on the CPU but only on one GPU (what was expected, since we specified this in the configuration file). Therefore, we had a partial success. So we tried to simply use another graphics card. This did not work any more, the result was quite strange, the application started, but it hat about 10 times longer than with the first GPU and the windows on the displays were all empty. Using the third GPU did bring on another problem again - here the displays were all black. We could not and still can not explain this surprising behaviour. We figured out, that Windows Vista is not really designed to use multiple graphics cards. And Equalizer had not been tested on Windows Vista yet, so there is no guarantee that it works at all under this system. After loosing a lot of hours trying to get a working Windows Vista Setup, we gave up and decided to install a Windows XP instead. Since we are working on a 64-bit system, we do not even have problems with the limited memory support under XP.

Finally, we got the testing computer running and could start with our first tests.

3.2.3 Moving to the CAVE

After our test applications worked like a charm on our system, it was about time to move in the *real* CAVE. We expected some more trouble because we had to deal with eight projectors now and four walls, instead of four displays which means, we have to make sure, that screens do not overlap. Again, Linux did not make any trouble to setup the (now) eight X Servers. The only problem now was that the resolution did not seem to be correct. Despite the fact that we configured our X11 to use a resolution of $1280x768$ on each screen the resolution was set to $1366x768$ - strange. We figured out what the problem was. The NVIDIA driver found out, that our projectors were able to use a $1366x768$ resolution (which it thought must be what we wanted), and therefore stretched the display. So all we had to do was to tell each GPU strictly not to stretch the display no matter what it thinks was best for us. After all, it worked (most of the time). We found a bug in the NVIDIA settings tool: After starting the X Server, the resolution is set to the (unwanted) $1366x768$ resolution. To fix that, you simply have to start the NVIDIA settings tool and close it again. You do not even have to change anything. Hopefully, this bug gets fixed in the next release.

[1]A pipe is an Equalizer abstraction for a Graphics Processing Unit (GPU). Consult the *Technical Report* for further information

3.3 CAVE Rendering Clients Setup

Since *our* CAVE clone worked like a charm, we could move on to the next challenge: Using multiple render clients. One of our tasks of the thesis is to compare different setups in performance. There already are six render clients in the CAVE. It turned out that they were already setup with different operating systems, one of them an Ubuntu. We hoped that we could use them unchanged - well, that was somehow a bit too credulous. Some of the render clients had got new harddisks since the linux was booted the last time, therefore the bootloader configuration did not work anymore. This problem could relatively easily be solved by booting from a live CD and fixing the bootloader. First problem solved, next following... We started the Ubuntu and after trying a few access trials, we finally managed to login and get a root shell. But ...
... the Ubuntu distributions were about five years old and after trying to install the first few needed applications, we gave up. Given the fact that Ubuntu comes with a *distribution upgrade* function, we thought it could not be that hard to upgrade to an up-to-date version. Well again - we thought...

After several hours of installation and upgrading, we finally booted each machine into a up-to-date ubuntu and we could start to install the needed software (Equalizer and OSG). We thought, it would be possible to install the software on one machine, and copy it to all others (since OSG takes about two hours to compile, this did not seem such a bad idea). We already had our problems installing it on the first machine. Well, we found out that the problem were missing dependencies. Luckily we had written down the needed packages the day we installed it on our CAVE clone, so it should not be a problem to get this fixed. It turned out, that there were more packages missing (some that apparently come with the new delivered Ubuntu, but not on the upgrade), therefore again, we had to search for dependencies, writing every one down, so there was no chance we would have to search again. We finally managed to install OSG and Equalizer on the first machine and copied it to the other ones. After trying to start the test application on each machine, it turned out that this did not work after all. Apparently there were more missing packages. Well, we do not want to bore you with another round of searching so lets just say: it was quite a nerve-racking issue until everything was installed as it had to be... To spare anybody else this try-and-error issue, we wrote down some installation instructions in the *User Manual*.

4 Failed Attempts

4.1 Overview

This chapter is dedicated to some interesting but failed approaches. In Order to find a solution for the CRF, some attempts were not as successful as they appeard at first. We want to point them out to prevent that they appear again in further work on that project.

4.2 OsgViewer Setup

When we first built our prototype, we figured out that a lot of the tested OSG functions did not work. After some debugging efforts it turned out that a lot of these tested functions rely on a window definition of the `osgViewer` class. But in our first prototypes the viewer has been initialised wrong and no window definitions have been created. Thus, common OSG features like the `osgViewer::StatsHandler` failed because the handler tries to draw the statistics on a window which has never been created. We fixed this issue by forcing the `osgViewer` to create an virtual OSG window (the real windows are handled by Equalizer). Therefore, we changed the mode of the `osgViewer` to “EmbeddedWindow” by calling `osgViewer::setUpViewerAsEmbeddedInWindow()`. The former mode was “EmbeddedContext” which does not create the desired `osg::window`.

4.3 Facade Implementation

While searching an easy way to pass a common OSG application to the CRF, we first tried to push the pointer of an `osgViewer` of an existing application to the facade. But this lead into a lot of segmentation faults, because multiple Equalizer pipe threads tried to access the same physical scene graph of the passed viewer. We figured out that every pipe needs its completely independent instance of the viewer. Hence, we removed the capability of pushing viewer pointers to the pipe. Instead, we implemented a mechanism to create custom pipe objects and scene graphs by inheriting the pipe class of the framework. The framework gets forced to use the newly introduced pipe by passing a customised factory to the facade class (`crfStarter`). Now, when the framework requests a pipe object, this factory creates a complete and independent instance of the pipe and accordingly its viewer.

4.4 Event Propagation

After the first successful tests with our framework dealing with static scene graphs, we needed a way to handle the synchronisation of multiple dynamic scene graphs, distributed over several nodes (depending on the chosen Equalizer topology). The first idea was to share the event queue of the `osgViewers` with the distributable frame data class. This would easily work on one node configurations without network, but for multinode configurations the frame data object is propagated over the network. Thus, all its members have to be serialised, which is not really easy for the OSG `EventQueue` class. The actual used solution provides a container member in the frame data class which receives the new events at the beginning of every frame from Equalizer and is pushed over the network to the different nodes. The newly introduced `crf::crfPipe` class contains several overridable functions, which convert the Equalizer events to OSG events. These functions are called by the framework. Thereafter, the converted events are submitted to every viewer. Because all these mentioned functions are called simultaneously, all the scene graphs receive the exact same events at the very same time.

5 Procedure

Since this project was a research project that did not promise a working outcome, part of the project was to find out how to find a solution that satisfies the requirements. A few prerequisits were given before the project started, one of them Equalizer. A former project at the BFH-TI did already compare the two scene graph technologies OSG and OGRE. Therefore the usage of one of these technologies was given as a requirement too.

Part of the hardware used for the project was existing. The CAVE was configured with a setup consisting of six render clients. The server that was already setup in the CAVE was taboo since it must always be available for other demos. We did not know if the six render clients were really set up for our project or not.

This was the point of departure in our project. Therefore, the first task of all we had to solve was to bring some system in the upcoming work. Though we all agreed that it does not make sense to set an official milestone each week, we decided to define some kind of internal milestones in weekly steps.

The next organisational decision we made was, that we meet twice a week to work together on the project. On Thursdays all day long and on Fridays for the morning. Therefore it was necessary to have a workplace at school.

After the first few weeks, we decided to devide the project in three parts, one part for each team member.

abbr.	**Name**	**speciality**	**comment**
hullb2	Brigitte Hulliger	Equalizer	Equalizer configuration and setups
waltj3	Jonas Walti	Framework	Implementation of the framework with respect to eqOSG
brods1	Stefan Broder	Hardware Setup	Hardware Setup for CAVE

Table 5.1: Team members and their specialities in the project

Our internal milestones are listed in **??**. At the beginning of the project these milestones were only vaguely worked out since e did not know what we will have to do until the end. But with progressing weeks, this plan got more detailed.

6 Testing

6.1 Framework Tests

6.1.1 Early Tests

Because of the fact that the real CAVE setup had to be configured first, we did the early tests on our local machines on Window and Linux with different Equalizer configurations. Some of these configurations simulated something like a multipipe environment and looked at least technically pretty similar to tests in the real CAVE. Later in the project we managed to setup our CAVE clone rendering client with four GPUs and LCD monitors. This was the time when we first realised that our framework did not work with a lot of early test applications on this multipipe machine. After some debugging and sleepless nights the errors could be tracked down as memory abuse. This happened because we first used a lot of the original OSG demo applications which are often implemented with a lot of static and global objects. This is not a problem on a single pipe machine with multiple simulated pipes, but with the four real pipes in our CAVE clone, these global objects messed up everything and caused a lot of segmentation faults and other weird behaviour. This was a really important finding for our further work and a lot of early ideas to simplify the whole framework had to be thrown over. The fact, that the scene graph has to reside on each pipe completely independently, changed everything and raised the complexity of the whole project.

6.1.2 CAVE Tests

After a lot of technical issues we finally made it to a real and working CAVE environment with the two intended render topologies. Unfortunately at this time, we already hit the second last week of our project time. Anyway, now we were able to do real tests on our framework with two different Equalizer configurations (four pipes on one node and six nodes with one pipe). Some of the achievements had to be discarded again because they failed the "real" tests but thereafter we were able to run all the use cases we defined in our functional specification document.

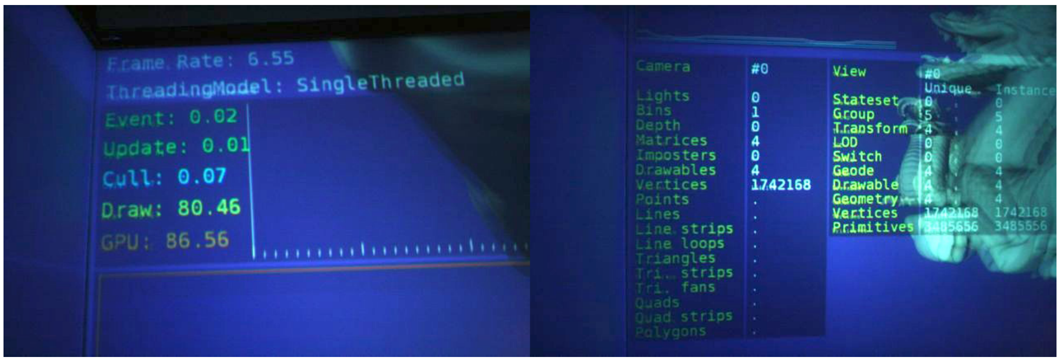
Stress Tests

In order to test the performance of the Cave Rendering Framework we wrote a test application. Purpose of this application is to test the framework under extreme circumstances. We used huge models[1] to exhaust the rendering process. The application draws a scene graph with one model on each wall, or floor, respectively. This way, the application should challenge each pipe equally. Additionally, one can let the models rotate and add some light sources to pressure the CRF even more. It turned out that the CRF does not really care about rotations but illumination almost cut the frame rate in half.

The biggest model we could load was the dragon with about 1.1 million triangles. Loading the next bigger model, another dragon with 7.2 million triangles, failed due to hardware limitations.

To measure the output we implemented a function to display generic OSG statistics as well as Equalizer statistics. Figure **??** shows a sample output of the stress application with several OSG statistics.

[1]We used models from the Stanford 3D Scanning Repository, provided on http://www-graphics.stanford.edu/data/3Dscanrep. The models range from about 700'000 (Stanford Bunny) up to 28'000'000 (Lucy) triangles.



```
Frame Rate: 6.55
ThreadingModel: SingleThreaded
Event: 0.02
Update: 0.01
Cull: 0.07
Draw: 80.46
GPU: 86.56
```

```
Camera        #0          View        #0
                                      Unique   Instance
Lights        0           Stateset    0        0
Bins          1           Group       5        5
Depth         0           Transform   4        4
Matrices      4           LOD         0        0
Imposters     0           Switch      0        0
Drawables     4           Geode       4        4
Vertices      1742168     Drawable    4        4
Points        .           Geometry    4        4
Lines         .           Vertices    1742168  1742168
Line strips   .           Primitives  3485656  3485556
Line loops    .
Triangles     .
Tri. strips   .
Tri. fans     .
Quads         .
Quad strips   .
Polygons      .
```

Figure 6.1: OSG statistics

The most interesting output for our measurements was the framerate per second (fps). As mentioned before, we tested two different Equalizer setups. One with a single node, using four graphics cards. The other, a distributed setup with six rendering clients and one server. On the latter, we achieved a more or less constant framerate of 100 fps with a model of 1'100'000 triangles. This was about what we expected.

The hardware and software setup as well as the detailed results of the tests can be found in the Technical Report.

Figure **??** shows the output of Equalizer measured with the multi-node setup. Equalizer orients itself on the slowest rendering client. The green at the bottom of the statistics represents client number six of the setup. It is the slowest client, meaning, it needs the most time to render a single frame. The five other clients listed above need about 1/5 of the time to render a frame.

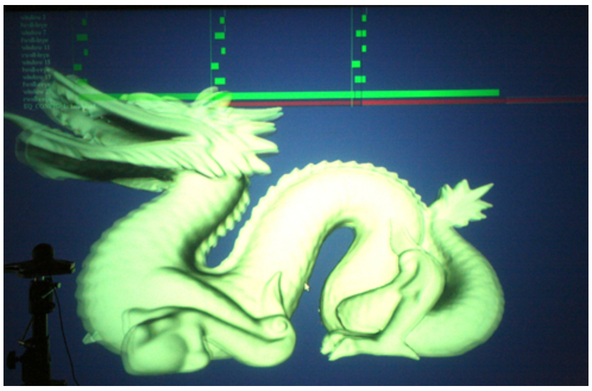


Figure 6.2: Equalizer statistics

The second approach with one computer was less satisfying. With this setup we reached a maximum of 20 fps. Conspiciously, the frame rate decreases linearly with each model, even though, they appear on separate walls and should therefore be rendered in different pipes. This behaviour leads to the conclusion that the entire scene graph is rendered on each graphics card. Concerning this issue, we wrote on the Equalizer mailing list. Stefan Eilemann, the founder and developer of Equalizer pointed out that this might be a problem of the NVIDIA graphics driver which could be solved with a later driver release. Other possible reasons he pointed out were:

- CPU/Memory contention
- Serialisation of the draw calls

- Bad multi-threading driver support
- Lock contention in OSG render traversal.

The last reason seemed unlikely since the OSG community never reported similar problems.

6.2 Equalizer Conclusion

Equalizer follows a new approach in parallel rendering. Other than Chromium, it renders the code in parallel and does not send the output over the network. Therefore, it solves the network traffic problem of Chromium.

But this leads to the disadvantage of Equalizer since it requires that an application has to be installed on *all* nodes that are used for rendering. We wrote a script that synchronised our clients after a new release of the application. Otherwise it is very time consuming to update all clients one by one.

Other than that, we could verify the performance results of Equalizer provided by its developers.

7 Organisation

7.1 Team

The project team consists of three students: Brigitte Hulliger, Jonas Walti and Stefan Broder. All of us study Information Technology at the BFH-TI in Biel. We are in the last semester of our bachelor degree course. Our main topic is Computer Perception and Virtual Reality in which we were educated for the last three semesters.

The bachelor thesis was supervised by Michael Luggen, an assistent and former student of the BFH, and Prof. Künzler, lecturer and researcher in CPVR.

7.2 Wiki

To share and exchange common project information within the thesis group, it turned out to be very helpful to use a wiki. The wiki can be found on https://wiki.brooper.dyndns.org. The wiki page runs on a private server from one of the students. Hence, it is not guaranteed to be accessible while you read this.

7.3 Subversion

To enable development on common sources it was vital to use some kind of version control. Therefore, we decided to use subversion. BFH-TI runs a subversion server which can be used for thesis projects by students. We used version control for our source code as well as for our documentation.

Our repository can be checked out on https://svn.bfh.ch/repos/projects/thesis09_cave. You need to have a valid and authorised BFH-TI account for that matter.

8 Project Management

8.1 Strategy

When we first discussed our process strategy we decided to use the sequential waterfall model, because all team members already worked with this kind of project management. But further discussions with our team assistant and thesis expert lead us to use a more iterative approach. After all, we used a combination of the classic waterfall model and the iterative approach. Because of the lack of knowledge and the complexity of our project, several maxims had to be tried out. Hence, a lot of small prototypes were built and tested to figure out what works and what does not. All these prototypes formed an iterative process model with the components *Analysis and Design*, *Implementation* and *Testing*.

8.2 Milestones

Please consider the functional specification document (*Pflichtenheft*, written in German) attached in the appendix. This section refers to the initially defined milestones in the mentioned document. For each milestone the objectives are listed and if they were achieved. Moreover, it is listed until when they were initially planed and when they were achieved. Delays are marked red while early achievements are marked green.

8.2.1 M1: Functional Specification created

Objective	Achieved	Date expected	Date realised
Functional specification completed	✔	18.03	18.03

Table 8.1: Objectives of milestones 1

8.2.2 M2: Prototype finished

This milestone changed in his manner. Instead of developing one prototype, we developed several to figure out the best way to go. So, this milestone was cycled multiple times. Thus, we needed some extra time to achieve this task.

Objective	Achieved	Date expected	Date realised
Final test cases defined	✔	18.03	29.03
Project set up defined	✔	18.03	18.03
Techniques to use defined	✔	18.03	18.03

Table 8.2: Updated objectives of milestone two

8.2.3 M3: Developing finished

M3.1: Design Concept realised

Based on the gathered experience of the prototypes, we were able to define a final software design for our framework.

Objective	Achieved	Date expected	Date realised
Class diagram created	✔	29.05	24.04
Sequence diagram	partially[1]	29.05	29.05
Techniques to use are well defined	✔	18.03	18.03

Table 8.3: Updated objectives of milestone 3.1

M3.2: Implementation finished

The user guide has been split up in a user manual on how to set up the BFH-TI and a programming guide about our CAVE Rendering Framework.

Objective	Achieved	Date expected	Date realised
Technical documentation completed	✔	29.05	28.05
User guide completed	✔	04.06	06.06
Programming guide completed	✔	04.06	09.06
Primary use cases implemented	✔	29.03	24.05
Primary requirements achieved	partially[2]	29.03	24.05
Runnable demo application	✔	29.03	18.05

Table 8.4: Updated milestones 3.2 criteria

M4: Tests finished

Objective	Achieved	Date expected	Date realised
Test reports available	✔	12.06	09.06
Bugs fixed	✔	29.05	29.05
All demo and test applications run well	✔	12.06	10.06

Table 8.5: Updated objectives of milestone 4

M5: Documentation/Project finished

Objective	Achieved	Date expected	Date realised
Milestones 1-4 achieved	✔	12.06	12.06
Technical documentation finished	✔	29.05	10.06
User guide and programming guide finished	✔	12.06	11.06

Table 8.6: Updated milestones 5 criteria

[1]Too much of the processes use Equalizer. It would have been far to complex to depict sequence diagrams of good quality.
[2]The rendering performance on multipipe configurations is insufficient

8.3 Appointments

Kick-Off meeting with whole project group	
Date	20.02.2009
Location	BFH-TI
Participants	- Prof. U. Künzler - M. Luggen - J. Walti - B. Hulliger - S. Broder
Substance	- content of the functional specification document - project environment - requirements - use cases - contacts - milestones and deadlines

Review of functional specification document	
Date	09.03.2009
Location	BFH-TI
Participants	- Prof. U. Künzler - M. Luggen - J. Walti - B. Hulliger - S. Broder
Substance	We discussed our functional specification document, what adjustments we had to make.

Meeting with expert H. van der Kleij	
Date	20.03.2009
Location	SBB, Bollwerk Berne
Participants	- H. van der Kleij - J. Walti - B. Hulliger - S. Broder
Substance	- project overview - review of the functional specification document - further procedure

eqOSG Meeting at the Eurographics 2009 in Munich	
Date	31.03.2009
Location	Eurographics, University of Munich
Participants	- S. Eilemann & Crew (Equalizer, Eyescale) - M. Luggen, J. Walti, B. Hulliger, S. Broder (BFH-TI) - Thomas McGuire & Team (University of Siegen)
Substance	This meeting was very interesting because a group of students from the University of Siegen already developed a comparable solution in a similar project. We could exchange some technical facts and see what they have done so fare. S. Eilemann was particularly interested in a generic integration of OSG into Equalizer.

Periodic meetings with supervisor M. Luggen	
Date	every two weeks
Location	BFH-TI
Participants	- M. Luggen - J. Walti - B. Hulliger - S. Broder
Substance	These meetings were excellent for experience exchange with M. Luggen, who gave us regularly worthwile inputs. We discussed general project decisions during these meetings and showed Michael Luggen what we already achieved.

9 Personal Summary

The developing of a simple framework for parallel rendering was stated as main goal of our project work. In the first phase of our project, we all tried to get a closer look on the Equalizer library and parallel rendering in general. This effort was expected and we were all quite eager to learn about parallel rendering. Thereafter, we had to put quite a bit time in the hardware setup. There was the CAVE clone to setup and some CAVE maintenance and cleanup that needed to be done. This was more time consuming than we expected. So, the the whole project delayed because the testing of the framework could not be done in a real environment and some ideas for the framework ended in an impasse, which, again, did cost us a lot of time. Nevertheless, some of the early ideas survived all the tests and are now part of the final release.

The idea exchange with the team from the university of Siegen was valuable and helped us at the beginning to get the right direction regarding the integration of OSG into Equalizer. We did reuse some parts of their solution, improved it for our purposes and eliminated some errors. After the first running prototypes we searched a way to simplify the whole thing which was one of our main goals. This was quite difficult because an Equalizer application is not simple and we could deskill a lot, but not all of it. Especially the rendering of a dynamic scene graph is very difficult with Equalizer, because the interaction between the different rendering outputs, what also has to work over the network for example, needs a kind of specialised protocol. We could not find a way to simplify this in general. At least all the mouse and keyboard inputs are committed to all the scene graphs and makes it now possible to change the scene graphs synchronously.

The development of the CRF was accompanied by configuring Equalizer. This, as well, did cost us a lot of time since our setup was pretty exceptional. Therefore, a lot of effort went into internet research and mailing lists. Initially, we tried address multiple graphics cards in Windows Vista what did turn out to be impossible at the moment. Thereafter, we had trouble setting up a network configuration of Equalizer. In fact, it reavealed to be a general problem when using 32-bit and 64-bit CPU architectures combined. After all, Equalizer seems to work out of the box for standard setups, but not so much for rather exotic setups.

To sum up, we all learned a lot about the difficulties of parallel rendering and framework development. The most important part of such a project is testing. To develop in complex surroundings like these different parallel rendering topologies, it is above all necessary to test under real conditions. If we could do such a project once again, we would first focus on the setup of the CAVE environment to ensure the availability of a working setup. This would have boost the development of the framework tremendously and thus, our result would be much more enhanced. The approach to do the CAVE setup and the development concurrently was kind of intuitiv, because of our limited work and time resources. Eventually, it might have been the better choice to only focus on the setups first.

10 Conclusion

After a tough start and some disappointing first test results, we could determine the key points and the difficulties of our project. A disappointing insight was the fact, that OSG provides a lot of features but it was pretty hard to find tutorials which were actual and of good quality. The API documentation, too, is rather poorly documentet. A lot of function descriptons are missing or detailed enough. Furthermore, we tried to find some advanced open source OSG applications to test our framework. Unfortunately, we could find nothing but some old and unspectacular examples.

Finally, with the hard-earned knowledge and our final test environment, we were able to create a fully working and easy-to-use framework for the rendering of OSG scene graphs with Equalizer. This framework is compatible with most of the possible and useful Equalizer topologies (not everything could be tested), renders the OSG scene graph with high performance and provides basic solutions for event handling and dynamic scene graph manipulation. Furthermore, the CRF is able to display a lot of useful statistics and should be easily extendible with new features like tracking, haptics and others.

11 Outlook

11.1 Possible Next Steps

There are several possible follow up projects conceivable. First, a real OSG application with a practical purpose which uses the currently available CRF features would bring a lot of benefits. Such a project could point out some useful enhancements and desires. In this context, it is not unlikely that new bugs appear which we could not discover. Therefore, some maintenance tasks are imaginable.

The creation of a derived version of the CRF could also be an interesting attempt: Actually, it should not be a big deal to integrate another graphic engine instead of OSG into the CRF. Such a trial could proof the flexibility of the CRF and would be worth it. If a different engine like OGRE provides a similar mechanism like the OSG viewer, it would be only necessary to change the classes `eqOsg::Channel`, `eqOsg::Pipe` and `crf::crfPipe` to run the CRF with another graphic engine.

A Worklog

16.02.2009	Project Start

week 1-2	16.02.2009 - 26.02.2009			
who?	**what?**	**output**	**achieved**	**comment**
hullb2	Equalizer knowledge	running Equalizer installation	✓	-
waltj3 / brods1	OSG/OGRE setup	-	✓	-
all	index for requirement specification	-	✓	-

Table A.1: Worklog Week 1-2: 16.02.2009 - 26.02.2009

week 3	27.02.2009 - 05.03.2009			
who?	**what?**	**output**	**achieved**	**comment**
all	Prototypes	-	✓	-
all	eqOSG evaluation	-	✓	-
all	eqOGRE evaluation	-	✓	-
all	first draft of requirements specification	-	✓	-

Table A.2: Worklog Week 3: 27.02.2009 - 05.03.2009

week 4	06.03.2009 - 12.03.2009			
who?	**what?**	**output**	**achieved**	**comment**
all	requirements specification finished	-	✓	-
all	**13.03.09: Appointment with Expert: H. van der Kleij**			

Table A.3: Worklog Week 4: 06.03.2009 - 12.03.2009

week 5	13.03.2009 - 19.03.2009			
who?	**what?**	**output**	**achieved**	**comment**
all	corrections of requirements specification	-	✓	-
M1	**Functional specification**		✓	-
M2	**Prototype finished**		✓	-

Table A.4: Worklog Week 5: 13.03.2009 - 19.03.2009

week 6	**20.03.2009 - 26.03.2009**			
who?	**what?**	**output**	**achieved**	**comment**
all	Evaluate scene graph techniques	-	✓	-

Table A.5: Worklog Week 6: 20.03.2009 - 26.03.2009

week 7	**27.03.2009 - 02.04.2009**			
who?	**what?**	**output**	**achieved**	**comment**
all	Eurographics 2009	networking with other interested people		

Table A.6: Worklog Week 7: 27.03.2009 - 02.04.2009

week 8	**03.04.2009 - 09.04.2009**			
who?	**what?**	**output**	**achieved**	**comment**
all	holidays	-	-	-

Table A.7: Worklog Week 8: 03.04.2009 - 09.04.2009

week 9	**10.04.2009 - 16.04.2009**			
who?	**what?**	**output**	**achieved**	**comment**
waltj3	Integration eqPly und OSG	-	✓	-
hullb2 / brods1	Compilation of CAVE clone	CAVE Clone	✓	-
hullb2 / brods1	Setup Ubuntu & Windows Vista on CAVE Clone	CAVE clone ready	✓	-
hullb2	1 node N pipe config	config files	✓	problems with windows configs
hullb2	config files for Windows	config files	✗	multipipe config for Windows Vista not possible

Table A.8: Worklog Week 9: 10.04.2009 - 16.04.2009

week 10	**17.04.2009 - 23.04.2009**			
who?	**what?**	**output**	**achieved**	**comment**
hullb2	Equalizer Configs for CAVE Server	configuration files	✓	-
waltj3	Integration eqPly and OSG	-	✓	-
brods1	CMake	CMake build scripts	✓	-

Table A.9: Worklog Week 10: 17.04.2009 - 23.04.2009

week 11	**24.04.2009 - 30.04.2009**			
who?	**what?**	**output**	**achieved**	**comment**
hullb2	planning until May	agenda	✓	-
hullb2 / brods1	3D stereo tests with eqOSG	-	✓	-
hullb2 / brods1	network rendering: terminology server/client, tests		✗	terminology OK, network tests successful on private setup, but failed in CAVE
hullb2 / waltj3	CAVE 4-pipe Setup: Hardware config	config file	✓	-
hullb2 / brods1	CAVE 4-pipe Setup: Software config	config file	✓	-
waltj3	Eventhandling with eqOSG	-	✓	-
waltj3	Define CRF Structure	Index	!	partially done
brods1	Templates for documentation	templates	!	partially done

Table A.10: Worklog Week 11: 24.04.2009 - 30.04.2009

week 12	**01.05.2009 - 07.05.2009**			
who?	**what?**	**output**	**achieved**	**comment**
brods1	Concept for Demo App	-	✓	-
all	Concept for all documents	Index	✓	-
brods1	Templates for documentation	templates	✓	-
waltj3	Test protocols	protocols	✗	-
waltj3	Eventhandling	-	✓	-
hullb2	network rendering in CAVE	-	✗	did not work yet. In contact with community
waltj3	class diagram for CRF	diagram	✓	-

Table A.11: Worklog Week 12: 01.05.2009 - 07.05.2009

week 13	**08.05.2009 - 14.05.2009**			
who?	**what?**	**output**	**achieved**	**comment**
hullb2 / brods1	First version of demo app	demo app	✓	-
brods1	Replace Windows Vista with Windows XP	Windows XP setup	✓	Equalizer did not work properly on Windows Vista
waltj3	first working CRF	crf	✓	-
waltj3	eqOSG problems solved	-	✓	-
hullb2	sample document for documentation	-	✓	-

Table A.12: Worklog Week 13: 08.05.2009 - 14.05.2009

week 14	**15.05.2009 - 21.05.2009**			
who?	**what?**	**output**	**achieved**	**comment**
brods1	Concept for user manual	Concept	✓	-
hullb2 / waltj3	80% of Technical Documentation	Tech. Doc.	!	About 50% finished

Table A.13: Worklog Week 14: 15.05.2009 - 21.05.2009

week 15	**22.05.2009 - 28.05.2009**			
who?	**what?**	**output**	**achieved**	**comment**
hullb2 / brods1	Major features integrated in demo app	-	!	development of demo app not forced. Feature-by-Feature app implemented instead.
waltj3 / hullb2	Testcases	Excel sheet with testcases	✓	-
all	Technical Documentation finished	Documentation	!	About 80% done
all	80% of Thesis Documentation	Thesis Documentation	!	About 20% done
waltj3	Framework finished	CRF	!	open test cases to pass
M3	**Development finished**		✓	-

Table A.14: Worklog Week 15: 22.05.2009 - 28.05.2009

week 16	29.05.2009 - 04.06.2009			
who?	**what?**	**output**	**achieved**	**comment**
all	all documents finished	documents	!	first draft finished, sent to expert and supervisors
hullb2 / brods1	Demo App finished	-	✔	running demo app and running feature by feature app
brods1	Stress Test application	working app	✔	-
waltj3 / hullb2	Testcases in CAVE	passed tests	✔	-
brods1	CMake script for delivery	CMake Script	✔	-

Table A.15: Worklog Week 16: 29.05.2009 - 04.06.2009

week 17	05.06.2009 - 11.06.2009			
who?	**what?**	**output**	**achieved**	**comment**
brods1	Bash Scripts for delivery	scripts	✔	-
hullb2 / waltj3	final tests	-	✔	-
all	final review of documentation	-	✔	-
hullb2	A4 page for Bachelor Book	A4 page	✔	-
brods1	CDs with source code for delivery	CDs	✔	-
hullb2	Printing documentation	5 hard-copies	✔	-
all	Delivery of hardcopies	-	✔	-
M4	**Tests finished**		✔	-
16.02.2009		**Project End**		

Table A.16: Worklog Week 17: 05.06.2009 - 11.06.2009