Berner Fachhochschule
Technik und Informatik

Bachelor Thesis 2009

CAVE Rendering Framework

Pflichtenheft

Abteilung	CPVR
Autoren	Brigitte Hulliger
	Jonas Walti
	Stefan Broder
Betreuer	Prof. U. Künzler
	Michael Luggen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Allgemeines

Das Projekt *CAVE Rendering Framework* ist eine Bachelor Thesis von drei Studenten der Berner Fachhochschule für Technik und Informatik (TI-BFH) in Biel. Die Arbeit wird in der Vertiefungsrichtung *Computer Perception & Virtual Reality* (CPVR) erstellt. Ziel ist es, im Zeitraum Februar bis Juni 2009 selbstständig ein Projekt zu erarbeiten, inklusive der benötigten Projektplanung und Dokumentation. Betreut wird das Projekt von einem Dozenten der Forschungsgruppe CPVR - Prof. U. Künzler, und seinem Assistenten Michael Luggen. Nebst der Betreuung ist ein Experte in das Projekt involviert - Herr Han Van der Kleij.

1.1 Zweck des Dokuments

Das vorliegende Pflichtenheft ist Teil der Projektplanung. Es soll die Ziele und Anforderungen an die Bachelor Thesis *CAVE Rendering Framework* wiedergeben. Es soll das Vorgehen und die Form der Arbeit regeln und nach Projektabschluss das Messen der Ziele erlauben.

Einleitend werden die verschiedenen bereits bestehenden Ressourcen, die in der Bachelor Thesis zum Einsatz kommen, näher erklärt. Somit sollte es auch für Informatiker ohne Hintergrundwissen in *Virtual Reality* (VR) möglich sein, das Pflichtenheft zu verstehen. Fachspezifische Begriffe und Abkürzungen werden im Glossar oder in Fussnoten erläutert.

2 Ausgangslage

2.1 CPVR - CAVE

Die Forschungsgruppe *Computer Perception & Virtual Reality* der Berner Fachhochschule für Technik und Informatik, betreibt einen 4-Wand CAVE (CAVE Automatic Virtual Environment) für die Visualisierung und Simulation von Projekten im medizinischen Umfeld.

Ein CAVE bezeichnet einen Raum zur Projektion einer dreidimensionalen Illusionswelt der virtuellen Realität. Dieser Raum kann auf verschiedene Weisen realisiert werden. Im Falle der CPVR Forschungsgruppe handelt es sich um einen offenen Würfel, bestehend aus 4 Wänden: einer Frontwand, zwei Seitenwänden und einem Boden. Bei den drei Wänden handelt es sich um Rückprojektionsleinwände, die in einem Kubus angeordnet sind. Auf jede Rückwand wird mittels zweier Projektoren ein stereoskopisches Bild projeziert, welches durch einen Computer erzeugt wird. Zwischen den Projektionsflächen nimmt der Benutzer mit einer Stereobrille eine dreidimensionale virtuelle Umgebung wahr, in der er sich mit Hilfe verschiedener Eingabegeräte frei bewegen kann. Weiter wird ein Tracker eingesetzt. Falls sich ein Benutzer im virtuellen Raum bewegt, so bewegen sich die Objekte mit. Mit dem eingebauten Tracker wird erfasst wohin ein Benutzer schaut. Mit diesen Angaben berechnet der CAVE-Server anschliessed die Korrektur, damit die virtuellen Objekte scheinbar stationär bleiben.

Die vierte Projektionsfläche - der Boden - ist erst seit kurzem in Betrieb. Anders als bei den Wänden, wird beim Boden nicht von ausserhalb des Würfels projeziert, sondern von der Decke. Mit dem Boden als weitere Projektionsfläche kann eine wesentlich bessere Immersion erreicht werden. Die Projektoren für die Bodenfläche sind an der Decke befestigt. Damit möglichst wenig Schatten auf die Projektionsfläche fallen, wird das Bild etwas geneigt von vorne projiziert. So fallen die Schatten hinter den Betrachter und beeinflussen die Immersion weniger. Dies bedingt jedoch, dass sich immer nur eine Person im CAVE befindet und diese nicht nach hinten schaut.

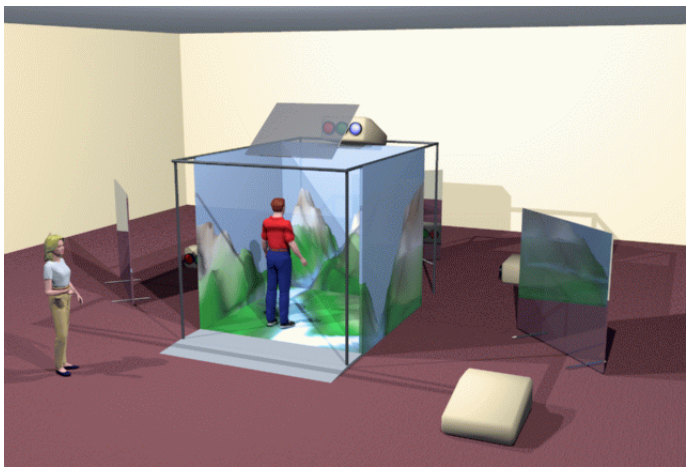

Abbildung 2.1: CAVE Lab (Quelle: [4])

2.1.1 Konfiguration

Der CAVE wird mit einem Server und acht Render Clients betrieben, die über ein Ethernet Netzwerk kommunizieren. Der Server ist verantwortlich für die Koordination der Render Clients. Die Render Clients berechnen die Bilder und stellen diese dar. Über den digitalen Videokanal (DVI)

werden die Projektoren an die Rendering Clients angeschlossen. Die Projektoren lassen sich über eine RS232C Schnittstelle von einem PC aus steuern. An einem Render Client sind per daisy chain von je zwei Projektoren angeschlossen. Abbildung **??** ist eine vereinfachte Darstellung, welche den CAVE mit nur drei Wänden zeigt.

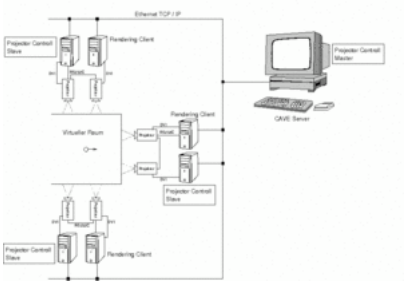

Abbildung 2.2: CAVE Config für drei Wände (Quelle: [4])

2.1.2 Stereoskopie

Um ein stereoskopisches Bild zu erzeugen, müssen zwei Bilder von zwei Projektoren gleichzeitig übereinander auf die gleiche Leinwand projeziert werden. Die Bilder der beiden Projektoren werden exakt parallel auf die Leinwand projeziert. Der mechanische Abstand der beiden Projektionsquellen muss softwareseitig korrigiert werden, d.h. die Bilder werden übereinander geschoben. Die Projektoren verfügen über eine spezielle Firmware Version (901.2202.41), welche die Positionierung des Bildes unterstützt. Dabei wird die maximale Hardwareauflösung von 1400x1050 softwaremässig auf 1280x1024 reduziert.

Der somit entstandene Rand kann für die Positionierung des Bildes gebraucht werden. Die Bilder lassen sich mit dieser Funktion pixelgenau übereinander positionieren.

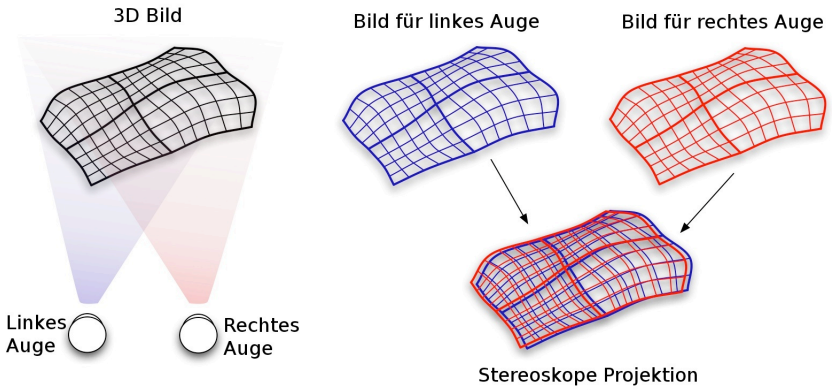

Abbildung 2.3: Stereoskopie

2.2 Paralleles Rendering

Standard OpenGL Applikationen werden nicht parallel gerendert. In einer sogenannten Event-Loop wird die Szene immer wieder neu gezeichnet, die Daten werden verändert und Events verarbeitet. Je nach Output wird das Frame neu gezeichnet.

Eine Applikation, die parallel gerendert wird, sieht vom Aufbau her sehr ähnlich aus. Im Gegensatz zu einer herkömmlichen OpenGL Applikation wird hier aber der Rendering Code von dem Main-Event-Loop getrennt. Der Rendering Code wird dann parallel auf verschiedenen Resourcen ausgeführt.

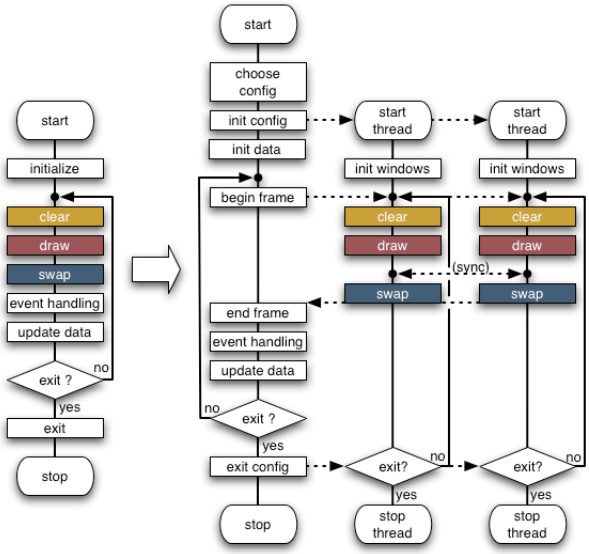

Abbildung 2.4: Paralleles Rendering (Quelle: [5])

2.2.1 Equalizer

Equalizer ist ein Framework zur Erstellung von parallelen, OpenGL-basierten Applikationen. Es ermöglicht, Applikationen auf mehreren Grafikkarten, Prozessoren und Computern zu renderen und somit die Performance zu verbessern. Eine auf Equalizer basierende Applikation kann unverändert aus verschiedenen Visualisierungssystemen (Workstations, Cluster, Virtual Reality Installationen, etc) gestartet werden.

Das Equalizer-Projekt ist richtungsweisend und einzigartig was paralleles Rendering anbelangt. Initiator und Hauptentwickler Stefan Eilemann ist der BFH-TI nicht unbekannt und mit ihm wurden bereits Kontakte geknüpft. Da es nichts vergleichbares gibt und gute Kontakte bestehen, ist die Verwendung von Equalizer vorgegeben. Ausserdem wird Equalizer rege genutzt und stetig weiterentwickelt.

Equalizer im CAVE

Equalizer unterstützt sowohl aktives wie auch passives Stereo Rendering. Dabei gibt es die Möglichkeit, dass jede Stereo View auf einer separaten CPU und Grafikkarte gerenderet werden kann.

2.3 Scene Graph

Ein Scene Graph ist eine objektorientierte Datenstruktur, mit der die logische und räumliche Anordnung von zwei- oder dreidimensionalen Objekten in einer Szene beschrieben werden kann. In der Computergrafik gibt es verschiedene Scene Graph Bibliotheken, die sich etabliert haben. Im Open

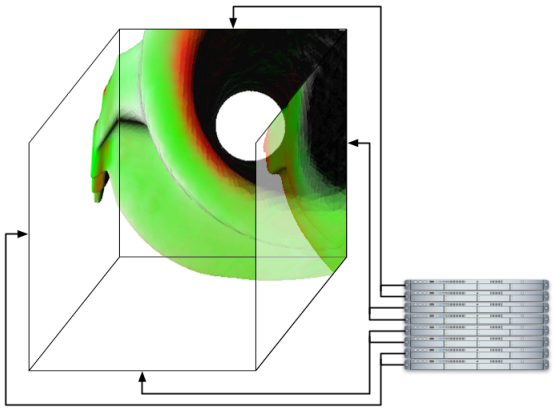

Abbildung 2.5: Equalizer im CAVE (Quelle: [5])

Source Umfeld gibt es zwei potentielle Bibliotheken, welche bereits in mehreren renommierten Projekten eingesetzt werden: OpenSceneGraph und OGRE.

2.3.1 OpenSceneGraph

OpenSceneGraph (www.openscenegraph.org) ist ein auf OpenGL basierendes 3D Grafik Toolkit für (wissenschaftliche) Simulationen, Modellierungen, Games und Virtual Reality. OSG bietet high-level rendering features die nicht in der Standard OpenGL API integriert sind. OSG ist komplett in standard C++ geschrieben und ist somit plattformunabhängig.

2.3.2 OGRE

OGRE (www.ogre3d.org) ist eine Open Source Enginge für 3D Grafikdarstellung mit C++ und OpenGL. OGRE bietet Hilfe bei der Entwicklung neuerer Techniken wie Vertex- oder Pixelshader, Normalmapping oder Verarbeitung von gängigen Model-Dateien.

2.3.3 Evaluation

Im Rahmen eines Vorprojekts an der BFH-TI wurden die beiden Scene Graphen auf verschiedene Kriterien untersucht. Dabei kamen wir zu folgenden Resultaten[1]:

		OpenSceneGraph			**OGRE**		
Kriterien	Gewichtung	Wert	Pkt.	%Pkt.	Wert	Pkt.	%Pkt.
Performance I	25%	160fps	3	75	220fps	7	175
Performance II	25%	260fps	7	175	220fps	3	75
Community	10%	++	4	40	+++	6	60
ähnl. Projekte	20%	eqOSG	7	140	eqOGRE	3	60
Dokumentation	10%	++	4	40	+++	6	60
Erweiterungen	10%	++	4	40	+++	6	60
Total	100%		**510**			**490**	

Tabelle 2.1: Entscheidungsmatrix

Wie man aus der obenstehenden Entscheidungsmatrix entnehmen kann, ist OSG punktemässig

[1] Die Punkteskala reicht von 1 bis 10.

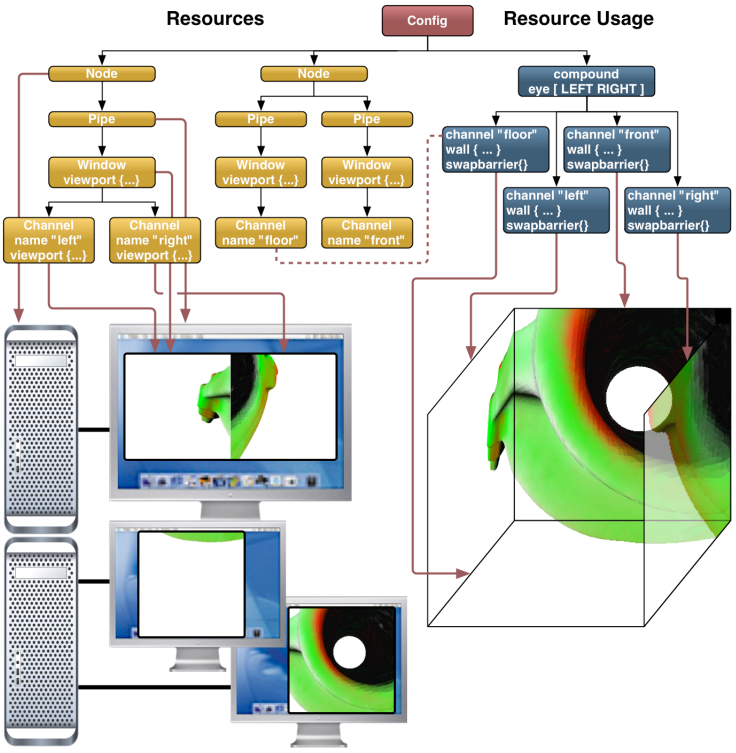

Abbildung 2.6: Equalizer im CAVE - Beispiel Konfiguration (Quelle: [1, Seite 10]

der knappe Sieger. Dies vor allem wegen der aktuellen Ambitionen welche in ein ähnliches Projekt investiert werden und der allgemein grossen Verbreitung in wissenschaftlichen Arbeiten. In der Performance [2] sind beide Engines etwa gleichauf. Im Test I ist OGRE zwar deutlich schneller, doch die Tatsache das OSG im Test II OGRE sogar überholt, zeigt die ausgeklügelte Technik, welche zur Optimierung bei OSG eingesetzt wird.
Klare Pluspunkte gewinnt OGRE in der Kategorie „Community“, da OGRE ein gutes Wiki-Nachschlagewerk und ein Forum mit reger Beteiligung vorweisen kann. OSG bietet ähnliches, jedoch nicht so gepflegt und ausgeprägt.
Bei den Erweiterungen lassen sich in beide Engines diverse thirdparty Plugins installieren, welche grösstenteils frei verfügbar sind. So gibt es für OGRE und OSG beispielsweise Erweiterungen für Physik und Haptik.

2.4 Grafik API

2.4.1 OpenGL

OpenGL ist eine weitverbreitete, (im Gegensatz zu Microsofts windowsbasiertem DirectX) plattformunabhängige und sehr performante Grafik API. Viele GPU unterstützen einen grossen Teil von OpenGL auf Hardwareebene, was OpenGL sehr schnell macht. Da im BFH Cave sowohl Linux, wie auch Windows eingesetzt wird, war bereits zu Beginn klar, dass eine OpenGL basierte Grafik

[2] Demo-Setup: Grosses 3D-Model mit einer knappen Million Triangles in einem leeren Universum. System: Intel Pentium Quadcore, nVidia Krafikkarte mit 768MB dediziertem Grafikspeicher. Performance I: Gesamtes Modell auf dem Screen. Performance II (Gezoomtes Objekt): Nur etwa ein Drittel des Models wird auf dem Bildschirm dargestellt. Der Rest (ausserhalb des Bildschirms) wird abgeschnitten.

Engine verwendet wird.

3 Anforderungen

Die Anforderungen sind unterteilt in einen primären und einen optionalen Teil. Dabei sind die Primären Anforderungen für einen erfolgreichen Abschluss der Bachelor Thesis relevant. Optionale Anforderungen werden in Angriff genommen, wenn die Zeit dafür reicht. Sollte dies nicht der Fall sein, so werden diese allenfalls in einer nachfolgenden Projektarbeit umgesetzt.

3.1 Primäre Anforderungen

- Scenegraph in Equalizer integrieren
- Stereoskope Ausgabe
- OpenGL basiert
- Performantes Scenegraph Rendering
- Erweiterbarkeit
- Konfigurierbarkeit

3.1.1 Scenegraph in Equalizer integrieren

Ein Hauptbestandteil dieser Arbeit ist die Integration einer gängigen C++ Scenegraph Library in Equalizer. Der Benutzer soll eine Applikation in OSG oder OGRE verfassen können, wobei das Rendering von Equalizer übernommen wird.
Es existiert bereits ein Framework (eqOSG) das diese Integration wahrnimmt. Ist es für unsere Aufgabe einsetzbar, so verwenden wir dieses verwenden.

3.1.2 Stereoskope Bildausgabe

Eine weitere Anforderung an unser Framework ist die Möglichkeit zur stereoskopen Bildausgabe. Im CAVE wird diese Technik zur 3D Umgebungsdarstellung verwendet. Daher ein Muss für das *CAVE Rendering Framework*.

3.1.3 OpenGL basiert

Das Framework soll auf OpenGL basieren. OpenGL ist eine weit verbreitete und sehr performante, hardwareunterstützte Grafik API. Des Weiteren hat OpenGL den Vorteil, dass es Plattformunabhängig ist.

3.1.4 Performantes Scenegraph Rendering

Das Rendern des Scenegraphen soll durch die Verwendung von Equalizer so performant wie möglich ablaufen. Ziel ist es, mindestens die gleiche Performanz zu erlangen, die mit den bisherigen Lösungen im CAVE möglich waren. Hier gilt es verschiedene Topologien zu testen.

3.1.5 Erweiterbarkeit

Das Framework soll eine spätere Integration von Plugins wie Haptics, Physics etc. ermöglichen.

3.1.6 Konfigurierbarkeit

Equalizer soll so konfigurierbar sein, dass beispielsweise nur ein Rechner (mit mehreren Grafikkarten) oder auch ein Cluster von mehreren Rechnern das Rendering tätigt.

3.2 Sekundäre Anforderungen

- Volume Rendering, Surface Rendering
- Haptics
- Shader
- Physics
- Tracking

Volume Rendering, Surface Rendering

Volume[1] und Surface Rendering[2] soll unterstützt werden.

Haptics

Integrieren einer Computer Haptics[3] Library in unser Framework. Präferenzen des Auftraggebers wären H3DAPI oder Chai3D. Es handelt sich bei beiden um OpenSource Lösungen. Beide Plattformen basieren auf OpenGL und sind in Form von C++ Libraries verfügbar.

Shader

Die Möglichkeit eigene Shader[4] zu programmieren und einzubinden soll bestehen.

Physics

Das Einbinden einer Physik Engine, die unter Anderem Kollisions Detektion unterstützt, soll vorgenommen werden. Hierzu gibt es viele OpenSource Kandidaten, wie zum Beispiel Open Dynamics Engine (ODE) oder Physics Abstraction Layer (PAL).

Tracking

Die Implementation von einem Tracking System soll es ermöglichen, den Körper oder einzelne Körperteile eines Benutzers zu verfolgen.

[1] Volume Rendering ist eine Technik, die sich mit der 3D Darstellung von (allen) Volumendaten befasst.
[2] Surface Rendering verwendet, im Gegensatz zum Volume Rendering, nur ein Teil der Volumendaten zur Abbildung des Objekts.
[3] Unter Computer Haptics versteht man, dass der Benutzer seine virtuelle Umgebung fühlen kann. Dies wird durch spezielle haptische Interfaces ermöglicht, wie einem Cyber Glove oder einem Cyber Pen.
[4] Durch Shaderprogrammierung lassen sich bestimmte Renderingeffekte auf 3D Computermodelle anwenden.

4 Use Cases

4.1 Primäre Use Cases

4.1.1 Use Case 1: Stereoskopische 3D CAVE Applikation

Der Virtual Reality Forscher Klaus möchte einen virtuellen Operationssaal erstellen, um damit Abläufe für seine Forschungsgruppe zu simulieren. Dieses 3D Modell möchte er im CAVE der TI-BFH stereoskopisch darstellen.
Klaus erstellt dazu sein 3D Modell mit dem bekannten Graphic Framework "OpenSceneGraph". Danach übergibt er den Scene Graphen dem CAVE Rendering Framework und wählt als Renderingkonfiguration das CRF Standard Set-Up aus. Dieses Set-Up konfiguriert Equalizer nun so, dass die 3D Szene im CAVE auf vier Leinwänden mit je zwei überlagerten Bildern pro Leinwand gerendert wird. Mit der Verteilung auf die verschiedenen Rendering Clients und der Konfiguration von Equalizer hat der Forscher aber nichts zu tun. Dies wird durch die im CAVE Rendering Framework zur Verfügung gestellte Konfiguration erledigt.

Nötige Schritte für Klaus

- Erstellen des Scene Graphen mit OpenSceneGraph.
- „Übergabe" des Scene Graphen an das CAVE Rendering Framework.
- Im CRF das Standard CAVE Setup auswählen.
- Starten der Applikation auf dem CAVE Server.

4.1.2 Use Case 2: monoskopische 3D CAVE Applikation

Vorgehen wie oben beschrieben mit dem Unterschied, dass Klaus seine Applikation ohne Stereoeffekt rendern will. Dazu geht Klaus wie bei Use Case 1 vor, mit dem Unterschied, dass er aus dem CRF eine Konfiguration ohne Stereoskopie auswählt.

Nötige Schritte für Klaus

- Erstellen des Scene Graphen mit OpenSceneGraph.
- „Übergabe" des Scene Graphen an das CAVE Rendering Framework.
- Im CRF das non-stereo CAVE Setup auswählen.
- Starten der Applikation auf dem CAVE Server.

4.1.3 Use Case 3: 3D Applikation mit Ausgabe auf zwei Monitoren (Custom Rendering Set-Up)

Klaus erstellt eine 3D Applikation und möchte diese an seinem Arbeitsplatz auf zwei Bildschirmen ausgeben (linke/rechte Hälfte). Die Grafik wird via einer Desktop-Workstation mit zwei GPUs gerendert. Klaus erstellt seine 3D Szene wie bisher mit OSG. Da das CRF eine solche Konfiguration nicht standardmässig zur Verfügung stellt, muss Klaus seine eigene Konfigurationsklasse schreiben. Dazu kann ein, vom CRF zur Verfügung gestelltes, Interface implementiert oder eine CRF Basisklasse abgeleitet werden.

Nötige Schritte für Klaus

- Erstellen des Scene Graphen mit OpenSceneGraph.
- Implementieren/erweitern eines CRF Interfaces/ einer CRF Basisklasse.
- „Übergabe“ des Scene Graphen an das CAVE Rendering Framework.
- Starten der Applikation auf seiner Desktop-Workstation.

4.2 Sekundäre Use Cases

4.2.1 Use Case 4: 3D Applikation mit Ausgabe auf zwei Monitoren (Custom Rendering Set-Up) und Haptik

Klaus möchte seine Applikation von Use Case 3 mit einem haptischen Interface von Senseable^{TM}bedienen. Dazu ergänzt er den Scene Graphen mit den nötigen Informationen für die Haptik (Material, Physik etc.) und startet seine 3D Applikation wie bei den anderen Use Cases mit dem CRF.

Nötige Schritte für Klaus

- Erstellen des Scene Graphen mit OpenSceneGraph und der nötwendigen Haptik.
- Implementieren/erweitern eines CRF Interfaces/einer CRF Basisklasse.
- „Übergabe“ des Scene Graphen an das CAVE Rendering Framework.
- Starten der Applikation auf seiner Desktop-Workstation.

5 Architektur

5.1 Equalizer

5.1.1 Equalizer Prozesse

Equalizer ist auf dem Client-Server Modell aufgebaut. Eine vereinfachte Darstellung der Architektur ist in Abbildung **??** ersichtlich.

Equalizer Server Der Server ist für das Management eines einzelnen Visualisierungssystems zuständig. Er startet die Rendering Clients der Applikation.

Application Die Applikation ist das Herzstück. Beim Starten der Applikation verbindet sich diese zum Equalizer Server, welcher daraufhin eine Konfiguration schickt. Die Applikation verarbeitet ausserdem Events und kontrolliert das Rendering ihrer Clients.

Application Render Client Der Render Client implementiert den Rendering Teil der Applikation. Es gibt keinen Main-Loop in der Ausführung, alle Tasks werden vom Server gesteuert.

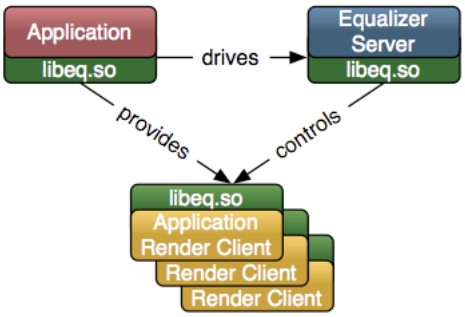

Abbildung 5.1: Equalizer Prozesse, (Quelle: [1, Seite 2])

Abbildung **??** zeigt ein vereinfachtes Modell eines Execution Models. Equalizer kreiert für jede verwendete Grafikkarte einen einzelnen Thread. Die Threads werden asynchron zueinander ausgeführt.

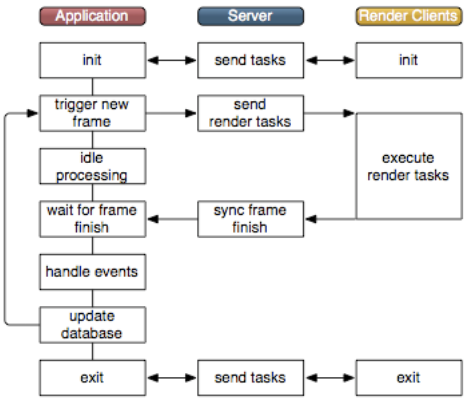

Abbildung 5.2: Equalizer Execution Model, (Quelle [1, Seite 9])

5.1.2 Stereo

Wie bereits in einem früheren Kapitel beschrieben, unterstützt Equalizer Stereo Compounds. Bei Stereo Compounds wird für jedes Auge eine separate Rendering Unit verwendet. Die Bilder werden nach dem Rendern in einen Stereobuffer gespeichert.

5.1.3 Config

Die Konfiguration einer Equalizer Applikation repräsentiert die Session, in der alle Render Clients registriert werden. Sie beinhaltet eine Beschreibung der Rendering Ressourcen und deren Verwendung. Diese werden in einer Baumstruktur aufgebaut, die der physikalischen Hierarchie einer 3D Rendering Umgebung entspricht. Abbildung **??** zeigt eine Beispiel Konfiguration für einen 4-Wand CAVE. Die Applikation wird auf zwei Workstations (Nodes) mit drei Grafikkarten (Pipes) gerechnet. Möchte man dieselbe Applikation auf einer anderen Topologie rendern lassen, so reicht es also, die Konfiguration von Equalizer anzupassen.

5.2 CAVE Rendering Framework

5.2.1 Grobarchitektur

Das Hauptziel unserer Arbeit besteht darin, eine Verbindung zwischen einem SceneGraph und Equalizer herzustellen (Abbildung **??**). Eine Studentengruppe aus Siegen, Deutschland, hat vor einiger Zeit mit einem ähnlichen Projekt angefangen - eqOSG. Sie haben dazu eine Beispiel Applikation entwickelt. Es gilt noch zu prüfen, ob diese Implementation unseren Zwecken genügt, und ob wir darauf aufbauen können. Wichtig für das Framework unsererseits ist es, dass es flexibel ist gegenüber anderen VR Techniken wie zum Beispiel Haptics.

5.2.2 eqOSG

Wie bereits erwähnt, wird an der Universität Siegen, Deutschland, ebenfalls an einem Projekt gearbeitet, OpenSceneGraph in Equalizer zu integrieren. Equalizer stellt ein Interface von Klassen zur Verfügung um Applikationen zu entwickeln. Abbildung **??** zeigt die Grobstruktur einer Implementation. Wie man sieht, gibt es in Equalizer ein Paket eq mit den Hauptklassen.

5.2.3 CAVE Konfigurationen

Wie in Abschnitt **??** beschrieben, soll Equalizer so konfigurierbar sein, dass das Rendering mit verschiedenen Topologien möglich ist. Dabei sollte es möglich sein, dem Entwickler ein Interface zu bieten, bei dem er sich nicht um Konfigurationen kümmern muss, ausser er will eine neue Topologie aufbauen.

Das Design Pattern *Facade* bietet sich dafür an. Bei diesem Pattern handelt es sich um ein Structural Pattern. Es bietet eine einheitliche Schnittstelle zu einer Menge von Schnittstellen eines Subsystems, in unserem Fall die verschiedenen Konfigurationen der VR-Systeme. Ein Subsystem wie es Equalizer darstellt, beinhaltet viele technisch orientierte Klassen, die von aussen selten oder nie gebraucht werden. Ein Entwickler braucht meist keine Informationen über die verschiedenen Nodes und Pipes zu haben. Die Fassade ist eine Klasse mit ausgewählten Methoden, die eine häufig benötigte Untermenge an Funktionalität des Subsystems umfasst. Sie delegiert die Funktionalität an andere Klassen des Subsystems und vereinfacht dadurch den Umgang mit dem System.

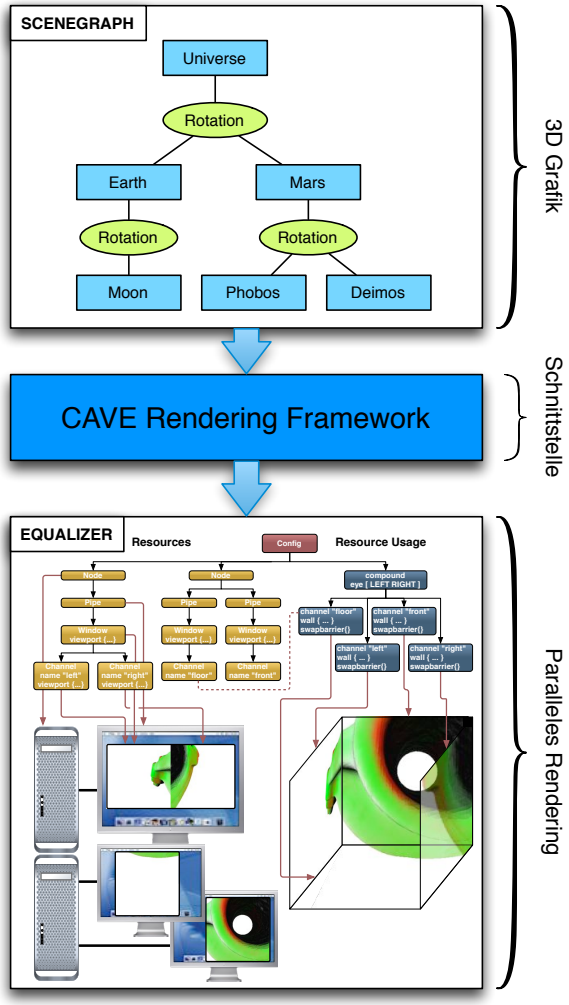

Abbildung 5.3: CAVE Architektur

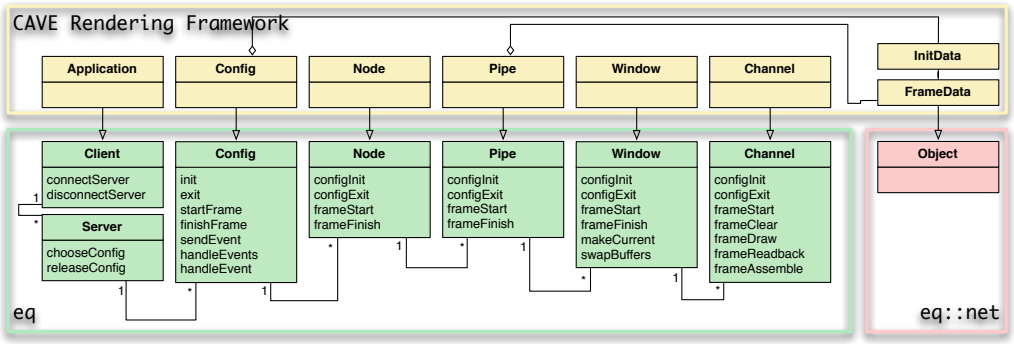

Abbildung 5.4: Package Modul

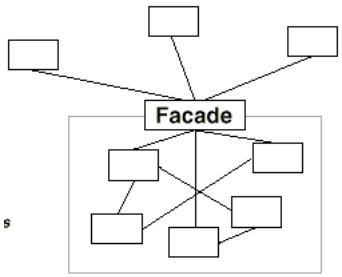

Abbildung 5.5: Facade Pattern (Quelle: [3])

6 Testing

Automatisierte Tests für ein Framework im grafischen Bereich sind meist recht schwierig zu realisieren. Wir gehen davon aus, dass die verwendeten Frameworks Equalizer und SceneGraph (OSG oder OGRE) selbst schon getestet sind. Unsere Tests beschränken sich also darauf, ob die Schnittstellen vom SceneGraph zu dem *CAVE Rendering Framework* und von diesem zu Equalizer wie gewünscht funktionieren.

Nebst der korrekten Funktionsweise des Frameworks interessieren wir uns vor allem auch auf eine gute Performance.

Um unsere Arbeit zu verifizieren, implementieren wir eine Testapplikation, mit deren Hilfe die Use Cases getestet werden können.

6.1 Testapplikationen

Die Testapplikation muss im CAVE der BFH-TI lauffähig sein. Folgende Funktionalitäten des Frameworks müssen mithilfe der Testapplikation überprüft werden können:

Modell Aus der Applikation müssen SG-Modelle geladen werden können.

Stereo Performance Es muss eine Möglichkeit geben, die Performance von Stereobilder zu überprüfen.

Shader Das Framework sollte Shader-tauglich sein (Optionale Anforderung). Es sollten also verschiedene Shader geladen werden können.

Haptics Falls Haptics im Framework bereits integriert wird (Optionale Anforderung), so muss es eine Möglichkeit geben, Haptics ein- und auszuschalten.

Grundsätzlich gilt, dass die Testapplikation eine Feature-by-Feature Applikation sein soll, d.h. für jedes Implementierte Feature im Framework soll es einen Test Case geben der (dynamisch) ein- und ausgeschaltet werden kann.

Falls eine geeignete bestehende Applikation gefunden werden kann, mit deren Hilfe die Features getestet werden können, so kann auch diese verwendet werden.

6.2 Test Cases

Um eine möglichst komplette und dokumentierte Testumgebung zu erlangen, soll zu jedem implementierten Feature des Frameworks ein Test Case formuliert werden. Dieser soll nebst dem Test Szenario auch dessen Pre- und Postconditions enthalten. Wichtig ist, dass das Testing iterativ erfolgt und wo möglich. Tests, die oft ausgeführt werden sollen, sollten wenn möglich automatisiert werden. Grundsätzlich sollte immer ersichtlich sein, was getestet wird, damit zum Schluss eine vollständige Liste von Tests ersichtlich ist.

7 Projektmanagement

7.1 Phasen

Das Projekt erfolgt in verschiedenen Projektphasen. Da es sich um ein Forschungsprojekt handelt, wird ein iteratives Vorgehensmodell einem Wasserfallmodell vorgezogen. Aus diesem Grund werden die im Folgenden beschriebenen Phasen iterativ wiederholt.

7.1.1 Evaluation/Planung

Evaluation

Da für das CRF Projekt viel Evaluation, in der Richtung: „Was ist überhaupt möglich?“, „Welche Techniken sind die geeignetsten?“ (siehe Scene Graph Evaluation) oder „Was gibt es bereits?“, betrieben werden muss, stellt die Evaluation einen grossen Teil des CRF Projektes dar.

Planung

Erst wenn gewisse Erkenntnisse aus der Evaluation gewonnen werden konnten, ist es möglich eine seriöse Planung zu erstellen. Deshalb sind diese Projektabschnitte verknüpft und laufen teilweise parallel.

7.1.2 Design

In der Designphase geht es hauptsächlich darum, das Software Design für das Cave Rendering Framework zu definieren. Zum Output gehören die verschiedenen Softwaredesign-Diagramme wie das Klassen- oder das Sequenzdiagramm nach UML-Standard. Der Detaillevel sollte so hoch wie möglich sein und diese Modelle sollten später nicht mehr allzugross ändern.

7.1.3 Implementation

In der Implementationsphase wird das CRF gemäss dem festgelegten Design implementiert. Ein wichtiger Teil dabei ist der parallel dazu zu entwickelnde Prototyp einer 3D Applikation, welcher das wachsende CRF nutzt und parallel dazu weiterentwickelt wird. Gegen Ende dieser Phase wird eine Demonstrator-Applikation erstellt, mit welcher sämtliche Möglichkeiten des CRF aufgezeigt und im CAVE demonstriert werden können. Diese Applikation soll ausserdem auch für allgemeine CAVE Demonstrationen verwendet werden können.

7.1.4 Testphase

Für die Testphase werden Testcases erstellt, welche sämtliche verwirklichten Anforderungen und Use Cases testen und messen. Dazu gehören insbesondere Funktions- und Performancetests. Fehler werden dokumentiert und korrigiert. Falls möglich, werden Performanceverbesserungen vorgenommen.

7.1.5 Einführung

Die CRF Demoapplikation wird im CAVE Labor so eingerichtet, dass sie von allen eingeführten Personen ohne grossen Aufwand zu Demonstrationszwecken und für Tests verwendet werden kann. Dazu werden die nötigen Personen geschult.

Das CRF ist so abgelegt und Dokumentiert, dass es von allen CPVR Mitarbeiter benutzt werden kann, um ihre eigenen Applikationen für den CAVE zu erstellen.

7.2 Meilensteine

7.2.1 M1: Pflichtenheft liegt vor - bis 18.03.2009

Kriterien

- Pflichtenheft vorhanden mit:
  - Ausgangslage
  - Architektur
  - Ziele
  - Anforderungen
  - Use Cases
  - Projektmanagement

7.2.2 M2: Prototyp entwickelt - bis 10.04.2009

Kriterien

- Test Cases für jedes Feature
- Projektsetup (Grobarchitektur)
- zu verwendende Techniken klar definiert

7.2.3 M3: Entwicklung abgeschlossen - bis 29.05.2009

M3.1: Designkonzept steht

Kriterien

- Klassendiagramme vorhanden
- Sequenzdiagramme vorhanden
- zu verwendende Techniken klar definiert

M3.2: Implementation abgeschlossen

Kriterien

- Technische Dokumentation liegt vor
- Benutzerdokumentation liegt vor
- Primäre Use Cases implementiert
- Primäre Anforderungen erfüllt
- Demoapplikation läuft

M4: Testphase abgeschlossen

Kriterien

- Testprotokolle liegen vor
- Fehler korrigiert
- Demoapplikationen funktionieren fehlerfrei

7.2.4 M4: Dokumentation abgeschlossen - 12.06.2009

Kriterien

- Meilensteine 1-3 erfolgreich abgeschlossen
- Technische Dokumentation
- Benutzerdokumentation
- Projektjournal

7.3 Projektplan

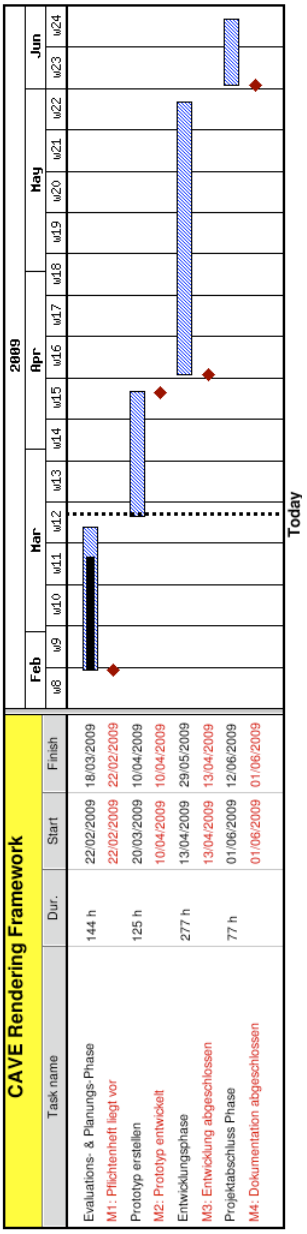

Abbildung 7.1: Projektplanung per 20.03.2009

8 Administratives

8.1 Beteiligte Personen

Name	Funktion	Email-Adresse
Han Van der Kleij	Experte	han.vanderkleij@sbb.ch
Prof. U. Künzler	Betreuer	klu1@bfh.ch
Michael Luggen	Betreuer	lgm5@bfh.ch
Jonas Walti	Student	waltj3@bfh.ch
Stefan Broder	Student	brods1@bfh.ch
Brigitte Hulliger	Student	hullb2@bfh.ch

Tabelle 8.1: Beteiligte Personen

9 Glossar

Begriff	Bedeutung
CRF	CAVE Rendering Framework
CPVR	Computer Perception & Virtual Reality
CAVE	CAVE Automatic Virtual Environment
VRML	Virtual Reality Modeling Language
X3D	ISO standard XML-based file format
OSG	OpenSceneGraph
OGRE	Object-Oriented Graphics Rendering Engine

Tabelle 9.1: Glossar

10 Quellen

	Titel	Ausgabe	Autor	Verlag
[1]	*Equalizer Programming Guide*	Version 1.5.4	Eyescale Software GmbH	
[2]	*Design Patterns CD*	18th printing, ISBN 0-201-63498-8	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides	Addison Wesley Longman, Inc.
[3]	*www.wikipedia.de*			
[4]	*www.cpvr.ti.bfh.ch*			
[5]	*www.equailzergraphics.com*			