



Unity in CAVE

Dokumentation

Julien Villiger, Daniel Inversini
V 3.4, 04.06.2015

Inhaltsverzeichnis

1	Einführung	4
2	Voranalyse Varianten	5
3	Chromium	6
3.1	Architektur	6
3.2	Anwendung	6
3.3	Argumentation	7
3.3.1	Pro	7
3.3.2	Kontra	7
3.4	Zusammenfassung	8
4	Equalizer	9
4.1	Architektur	9
4.2	Anwendung	10
4.3	Argumentation	11
4.3.1	Pro	11
4.3.2	Kontra	11
4.4	Zusammenfassung	11
5	Eigene Lösung	12
5.1	Idee	12
5.2	Typ „Mehrere Instanzen“	12
5.2.1	Architektur	12
5.3	Typ „Eine Instanz“	14
5.3.1	Architektur	14
5.4	Funktionale Anforderungen	14
5.5	Argumentation	15
5.5.1	Pro	15
5.5.2	Kontra	15
5.6	Zusammenfassung	15
6	MiddleVR	16
6.1	Idee	16
6.2	Warum MiddleVR?	16
6.3	Abdeckung MiddleVR	16
6.4	Konzept MiddleVR	17
6.5	Argumentation	18
6.5.1	Pro	18
6.5.2	Kontra	18
6.6	Zusammenfassung	18
7	Gegenüberstellung Varianten	19
8	Entscheid Variante	21
9	Ausgangslage Prototyping	22
9.1	Aktuelle Infrastruktur	22
9.2	Multi-GPU	23
9.3	Clustering	24
9.4	Mosaic	25
9.5	GeForces	27
10	Prototypen	28
10.1	Basic	28
10.2	Basic Extended	28
10.3	AngryBots	29
10.4	Car Tutorial	29
11	Finale Gegenüberstellung	31
11.1	Übersicht	31

11.2 Auswertung	33
11.3 Offene Punkte	33
11.3.1 Mehrere Kameras	33
11.4 GPU Performance	34
12 Finaler Entscheid	36
13 Abbildungsverzeichnis	37
14 Tabellenverzeichnis	37
15 Anhang	38
15.1 Projektorganisation	38
15.1.1 Projektteam	38
15.1.2 Betreuer	38
15.2 Projektplanung, Meilensteine	38
15.3 Qualitätssicherung	39
15.4 Risikoanalyse	39
16 Versionskontrolle	40

1 Einführung

Die vorliegende Arbeit beschäftigt sich mit der Integration von Unity¹ in den CAVE² (Cave Automatic Virtual Environment) der BFH. Die Entwicklung von virtuellen 3D-Welten mit Basis-APIs wie OpenGL³ oder OpenSceneGraph⁴ ist eine zeitraubende und aufwendige Arbeit und erfordert jedes Mal eine Einzelentwicklung.

Dieser Umstand führte zu der Idee, eine highlevel Game Engine⁵ einzusetzen, mit der die Entwicklungszyklen vereinfacht und verkürzt werden können. Unity hat sich in den letzten Jahren in diesem Bereich durchgesetzt und ermöglicht, Spiele gratis zu entwickeln.

Im Rahmen der Projektarbeit 2 wird eine Machbarkeitsstudie durchgeführt und es werden verschiedene Varianten geprüft, wie die Implementierung erfolgen könnte. Nachfolgend zur theoretischen Abklärung werden Prototypen umgesetzt und am CAVE angeschlossen, um eine bessere Entscheidungsbasis zu schaffen.

Ziel dieser Arbeit ist, eine geeignete Vorgehensweise für die Integration von Unity in den CAVE zu finden, um die finale Umsetzung in einer späteren Arbeit (Bachelorthesis) durchzuführen.

¹ <http://unity3d.com/> Unity Game Engine

² http://de.wikipedia.org/wiki/Cave_Automatic_Virtual_Environment CAVE

³ <https://www.opengl.org/> OpenGL

⁴ <http://www.openscenegraph.org/> OpenSceneGraph

⁵ http://en.wikipedia.org/wiki/Game_engine Game Engine

2 Voranalyse Varianten

Basierend auf einer Besprechung mit dem Projektbetreuer wurden vier Möglichkeiten in Erwägung gezogen, wie die Integration von Unity in den CAVE erfolgen kann.

1. Chromium
2. Equalizer
3. Eigene Lösung
4. MiddleVR

Um eine standfeste Entscheidung treffen zu können und die Umsetzung eines Prototyps anzugehen, wurde für jede Variante eine detaillierte Voranalyse vorgenommen. Die Analysen dienen als Gegenüberstellung und sind Basis für den Variantenentscheid.

3 Chromium

3.1 Architektur

Chromium⁶ ist eine OpenGL Implementation. Doch entgegen üblicher Implementationen wird der OpenGL Command nicht in ein Rasterbild umgewandelt, sondern wird manipuliert und an andere OpenGL Implementationen weitergeschickt.

Die Chromium Bibliothek unterstützt eine Server / Client Architektur. Die Verarbeitungskette ist unterteilt in mehrere Stream Processing Units, kurz SPU.

Für jeden OpenGL Command hat eine SPU folgende Möglichkeiten:

- Modifizieren
- Ablehnen
- An eine weitere SPU weiterschicken

Die letzte SPU hat die Wahl, den OpenGL Command an eine lokale OpenGL Implementation zu überreichen um ein Rasterbild zu generieren, oder über ein Netzwerk an einen oder mehrere Chromium Servers zu schicken.

Die Chromium-Instanz läuft auf dem sogenannten „Mothership“ und managed die SPU Kette und Netzwerkverbindungen. Die laufende Applikation setzt die Drawcalls an die Hauptinstanz (Mothership) ab.

3.2 Anwendung

- **Multi-Monitor Displays**

Darstellung der OpenGL Commands auf mehreren Displays. Konzipiert für einen CAVE mit mehreren Leinwänden.

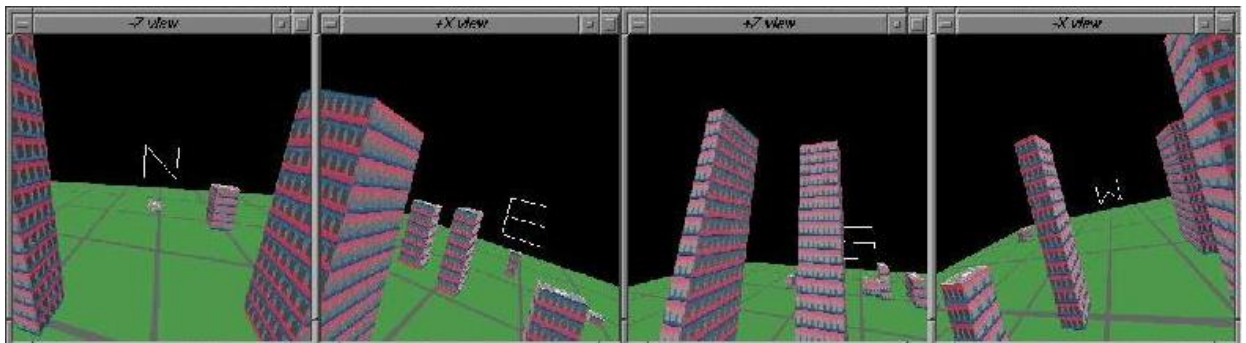


Abbildung 1: Ansicht der Leinwände in vier Himmelsrichtungen, Quelle:

<http://chromium.sourceforge.net/presentations/SantaFe-BrianPaul/siframes.html>

- **Delegation**

Der gesamte OpenGL Stream kann von einem Rechner auf den anderen verschoben werden. Hat eine Maschine keine dedizierte Grafikkarte, kann diese Aufgabe an einen besser ausgerüsteten Rechner delegiert werden.

- **Manipulation**

Polygone eines OpenGL Streams können manipuliert werden. Sogar eigene Rendering Styles können dank der komplett programmierbaren Rendering Pipeline von Chromium implementiert werden.

⁶ <http://chromium.sourceforge.net/> Chromium

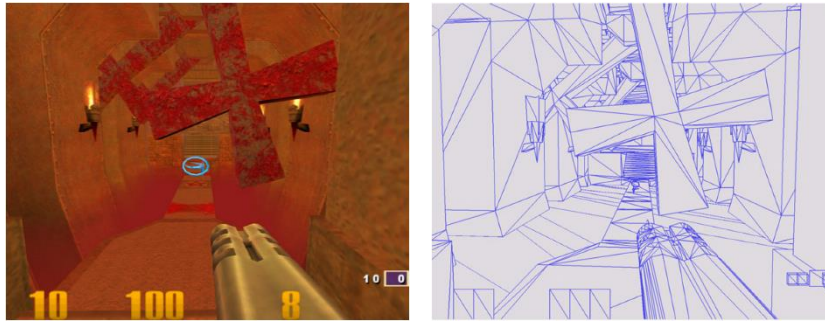


Abbildung 2: Implementation eigener Rendering Styles, Quelle: <http://chromium.sourceforge.net/doc/index.html>

- **Stereoskopie**
Nicht-Stereoskopische Anwendungen können in Stereoskopische umgewandelt werden. Aktive (Shutter Glasses) sowie passive (Polarisierte Lichtprojektion) Stereoskopie werden unterstützt.
- **Command Stream Aufteilung**
Ein OpenGL Command Stream kann aufgeteilt werden, damit versch. Rechner einen Teil des Renderings übernehmen können. Vergleichbar mit Nvidia's SLI⁷.

3.3 Argumentation

3.3.1 Pro

- **Stereoskopie**
Das wichtigste Feature, die Umwandlung in eine stereoskopische Darstellung, wird angeboten.
- **Infrastruktur**
Die benötigte Infrastruktur ist gegeben. Das Rendering kann auf mehrere Rechner verteilt werden.
- **Aufteilung der Monitore**
Jeder Projektor im CAVE hat einen eigenen Viewport und diese Aufteilung wird unterstützt.

3.3.2 Kontra

- **Plattform**
Chromium wurde auf Linux entwickelt und sollte auf diesem System ausgeführt werden. Unter Windows und OSX gibt es bekannte Probleme. Das verlangte Linuxwissen müsste zusätzlich erarbeitet werden.
- **Maintenance**
Seit 2006 gab es kein Update mehr. In einer Präsentation aus dem Jahre 2004 wird als nächste Phase der Support von OpenGL 2.0 angestrebt (Quelle: Slide 27, <http://chromium.sourceforge.net/presentations/SantaFe-BrianPaul/siframes.html>). Bis anhin wurde dieser Task nicht umgesetzt.

Falls Anwender von Chromium auf ein Problem stossen, können Feature Requests auf sourceforge.net abgesetzt werden (<http://sourceforge.net/p/chromium/feature-requests/>). Die letzten Requests wurden im Jahre 2002 bearbeitet und geschlossen. Neu erstellte Einträge

Zitat auf der offiziellen Sourceforge-Seite (<http://sourceforge.net/projects/chromium/>), 22.01.2015: „UPDATE: Chromium is no longer updated or maintained. The project is frozen.“

⁷ <http://www.nvidia.de/object/sli-technology-overview-de.html> Nvidia SLI

- **OpenGL Support**

Die letzte noch unterstützte OpenGL Version war 1.5 mit Chromium Release 1.5 (Dezember 2003). Die aktuelle Version von OpenGL ist 4.5 (Release August 2014).

- **Kompatibilität Unity**

Etliche Features, die über die OpenGL Version 1.5 hinausgehen und von Unity verwendet werden, könnten bei Chromium zu schwerwiegenden Problemen führen.

Der Output von Unity könnte inkompatibel mit den SUPs sein. Eine Modifikation des Outputs müsste in Betracht gezogen werden, wobei der Aufwand sehr schwer abschätzbar und nur bedingt zielführend ist.

- **Netzverkauslastung**

Engpässe könnten entstehen, weil der gesamte OpenGL Stream übers Netz geschickt wird. Bei simplen Anwendungen mit wenigen Primitiven⁸ sollte die Performance ausreichen, in Anbetracht dessen, dass auch komplexe Unity Spiele gerendert werden sollen, würde sicherlich die Netzwerkkapazität nicht ausreichen. Vorgängige Tests müssten durchgeführt werden.

- **Community**

Chromium hat keine aktive Community mehr, die bei Problemen bei der Installation Hilfestellung bieten könnte.

3.4 Zusammenfassung

Basierend auf der Gegenüberstellung der Pro- und Kontra-Argumentation und obwohl unsere geforderten Key-Features von der Chromium Graphics Library abgedeckt werden, sind die Nachteile massiv überwiegend.

Ausschlaggebend sind in erster Linie der eingestellte Support und die fehlende Weiterentwicklung der Bibliothek. Solange Unity und OpenGL sich am Weiterentwickeln sind, müsste Chromium laufend nachziehen und die neu entwickelten Features unterstützen.

Das Ziel der Thesis ist der Einsatz moderner und zukunftsorientierter Technologien. Wird auf ein Relikt gesetzt, ist der Erfolg der Umsetzung fraglich und keinesfalls eine robuste Basis, um zeitgemässe Anwendungen laufen zu lassen.

⁸ <https://www.opengl.org/wiki/Primitive> OpenGL Primitiven

4 Equalizer

Equalizer⁹ ist ein Open Source Framework für skalierbares, paralleles Rendering basierend auf OpenGL, welches ein API zur Verfügung stellt um solche graphischen Applikationen zu entwickeln.

Heute wird das Equalizer Framework im CAVE der BFH bereits verwendet. Das Ziel unseres Projekts ist es, Möglichkeiten aufzuzeigen, wie das Unity im aktuellen CAVE verwendet werden kann. Da unter Umständen Komponenten wiederverwendet werden können, wird dieses API überprüft.

4.1 Architektur

Equalizer verwendet verschiedene Wrapperklassen, um die Systemressourcen abstrahiert darzustellen. Die gesamte Liste der Ressourcenklassen ist hier zu finden:

<http://www.equalizergraphics.com/documents/Developer/API-1.0/internal/annotated.html>

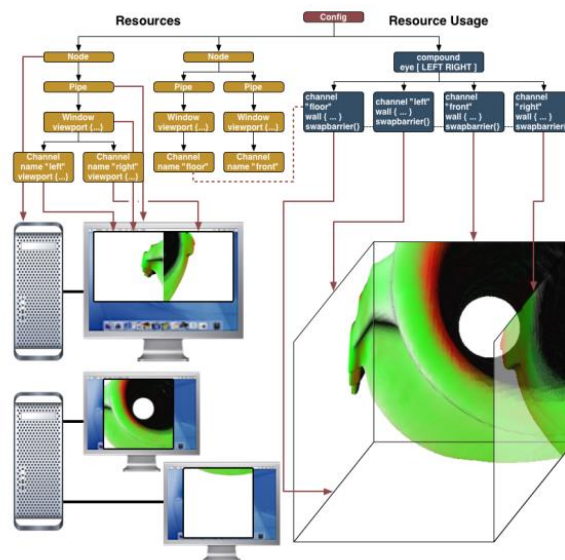


Abbildung 3: Equalizer Beispiel, Quelle: <http://www.equalizergraphics.com/documents/Developer/eqPly.pdf>

Nodes identifizieren einzelne Computer innerhalb des Clusters, wovon jeder mehrere Graphikkarten besitzen kann, Pipe. Dies definiert sich dann weiter zu Window, welche die einzelnen OpenGL Drawables und Contexte verwalten.

Abbildung 1 stellt ein Beispiel mit zwei Computern und drei Graphikkarten dar, welche vier Wände eines CAVEs rendern.

⁹ <http://www.equalizergraphics.com/> Equalizer

Als Überblick noch die wichtigsten Equalizer Klassen:

Equalizer Klasse	Information
eq::Config	Beschreibung der vorhandenen Ressourcen
eq::Node	Ein Client (Computer) im Rendering Cluster
eq::Pipe	Graphikkarte des Knoten (Node)
eq::Window	OpenGL Drawable auf der entsprechenden Pipe
eq::Channel	Viewport im Window
eqNet::Object	Verteiltes Objekt für gemeinsam genutzte Daten

Tabelle 1: Wichtige Equalizer Klassen

4.2 Anwendung

Equalizer kann auf alle Applikationen angewendet werden, welche Quellcode offen sind und auf OpenGL basieren. Vorzugsweise sollte die Applikation in C++ wie Equalizer programmiert sein,

Da Equalizer sehr flexibel ist, sind verschiedene Anwendungen möglich, hier nur Auszüge, welche uns interessieren:

Komplette Liste unter <http://www.equalizergraphics.com/useCases.html>.

- **Multi Displays**

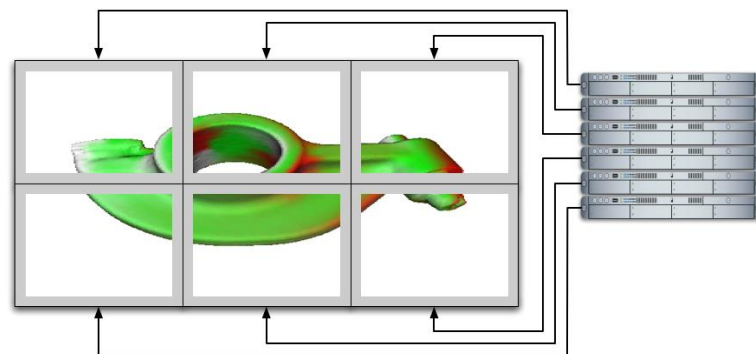


Abbildung 4: Display Wall, Quelle: <http://www.equalizergraphics.com/useCases.html>

Konfigurationen möglich wie Display Wall (oben) und CAVE Anwendungen (unten).

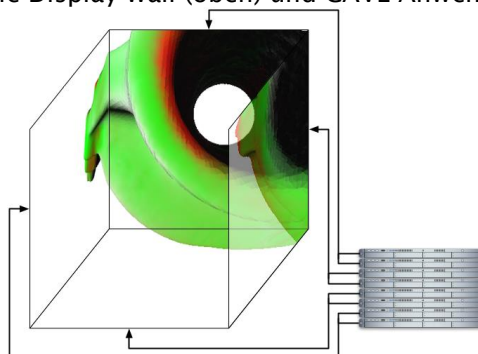


Abbildung 5: Vierseiten VR Installation, Quelle: <http://www.equalizergraphics.com/useCases.html>

4.3 Argumentation

4.3.1 Pro

- **Open Source, Dokumentation**
Equalizer ist Open Source. Es ist eine relativ aktuelle Dokumentation vorhanden.
- **Know-How BFH**
Durch verschiedene Projektarbeiten und eine bereits vorhandene Installation des gesamten Frameworks ist an der Berner Fachhochschule BFH in Biel Know-How vorhanden. Da aber eigenes Knowhow erarbeitet werden soll, ist dieser Punkt sehr tief zu priorisieren.

4.3.2 Kontra

- **Unity Bezug**
Unser Projekt sollte klar auch eine Einarbeitung in Unity sein. Es sollte nicht die Hauptarbeit sein, Implementationen eines anderen Frameworks, Equalizer, anzupassen.
- **Architektur**
Equalizer hat durch seine Wrapperklassen eigentlich eine sehr starre Struktur vorgegeben. Man müsste im Unity sehr tief eingreifen, um diese Klassen einzubauen. Es soll nicht eine komplizierte Version, « einen Hack », hergestellt werden, um dies auf Biegen und Brechen genauso und nur so einzubinden. Weiter ist unklar, ob über die Objektklassen alles andere von Unity auch (KI, Physik, etc) auch abbilden lässt.

4.4 Zusammenfassung

Basierend auf der kurzen Analyse der Dokumentation von Equalizer, unserer Projektanforderungen und den kurzen Pro- und Kontra Argumenten ergeben sich folgende Schlussfolgerungen:

Eine Verwendung des Equalizer Frameworks wäre denkbar, eventuell technisch sogar möglich, aber folgende Contra-Argumente wiegen zu schwer:

1. Wrapperklassen sind möglich im Unity. Da aber Equalizer OpenGL Aufbauend ist, fehlen uns wichtige Element wie für die Physik, KI, etc. So ist das Ziel – ein Unity Projekt/Spiel/Techdemo einfach und bequem im CAVE anzubieten, nicht möglich.
2. Abschweifung von Unity hin zu C++. Es wird angenommen, dass der Grossteil der Arbeit/Prototypen dann direkt auf C++ Ebene durchgeführt werden müsste. Dies ist durchwegs denkbar, aber das Hauptaugenmerk für Unity (mit C#) würde somit verfehlt.

5 Eigene Lösung

5.1 Idee

Eine Variante besteht darin, auf Frameworks von Drittparteien zu verzichten und eine eigene Lösung zu entwickeln. Dieser Ansatz wird nur in Erwägung gezogen, falls sämtliche Möglichkeiten mit einem bestehenden System als unzureichend oder nicht umsetzbar eingestuft wurden.

Eines der Hauptprobleme wird die Synchronisierung des Servers und der verschiedenen Clients sein. Sobald Scripts Zufallskomponenten beinhalten, darf die Berechnung des Verlaufs des Spiels oder der Simulation nur zentral an einem Ort geschehen. Ansonsten sind die Stationen nicht mehr synchron und es kann kein einheitliches Bild mehr im CAVE dargestellt werden.

Weiter sind Methoden zu prüfen, die nur eine einzelne Instanz der Unity Applikation voraussetzen. Die Berechnung geschieht dementsprechend auf einem Client und eine Synchronisierung entfällt hierbei. Die Aufteilung der Bildschirme wird von Windows, bzw. Mosaic¹⁰ übernommen.

Die konkreten Umsetzungsvorschläge werden in zwei Kategorien unterteilt. Einerseits diejenigen, die eine Synchronisierung voraussetzen (parallel laufende Instanzen) und andererseits Lösungen, die mittels einer Single-Instance funktionieren.

5.2 Typ „Mehrere Instanzen“

Zwei Ansätze sind in Betracht zu ziehen.

- **Unity – Clustering**
Unity bietet Module, um Multiplayerspiele zu entwickeln. Das Problem der Synchronisierung tritt vor allem in diesem Bereich auf und ist somit ein zentrales Anliegen der Unity-Entwickler und Anwender. Auf deren Erfahrung und der bereits umgesetzten Module kann zurückgegriffen werden.
- **Eigenes Protokoll**
Falls Unity mit den Standardfunktionen zu wenige Möglichkeiten bietet und auch im Asset Store keine hilfreichen Bibliotheken vorhanden sind, müsste ein eigenes System zum Synchronisieren erarbeitet werden, welches die Kommunikation zwischen den Clients und dem Server unabhängig von Unity betreibt. Zum Einsatz käme das User Datagram Protocol (UDP), welches die nötigen Informationen vom Hauptserver an die Clients schickt.

5.2.1 Architektur

Das geplante System basiert auf einer Client-Server Architektur. Die Hauptinstanz der Unity Anwendung wird auf einem eigenen Server laufen, welche den Rendering-Clients die nötigen Informationen zukommen lässt, um die Synchronisierung zu gewährleisten.

¹⁰ <http://www.nvidia.de/object/nvidia-mosaic-technology-de.html> Nvidia Mosaic

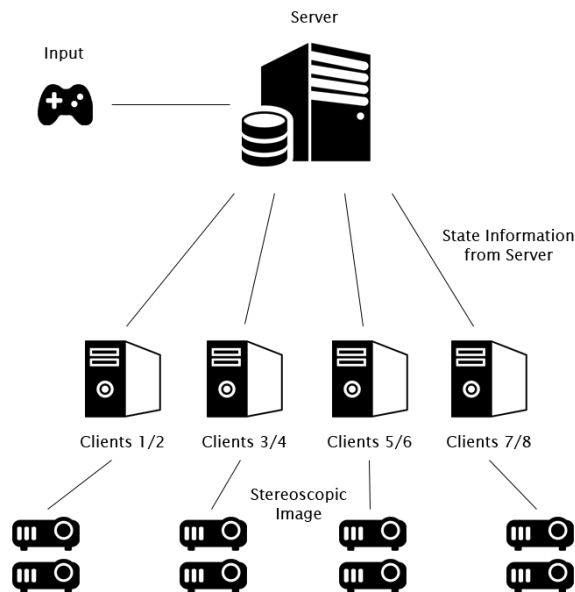


Abbildung 6: Client-Server Architektur, Quelle: Eigendarstellung

- **Aufgabe Server**

Die Hauptinstanz berechnet den Ablauf des Spiels oder der Simulation. User-Inputs werden hier verwaltet und entsprechende Geräte sind deshalb an diesem Rechner angeschlossen.

Nachdem die Anwendung gestartet wurde, ist sie bereit für die regelmässige Auslieferung der Informationen zum Synchronisieren der Clients und wartet auf deren Verbindungsaufbau.

Die Position der Kamera und der aktuelle State des Spiels oder der Simulation werden laufend an die Clients übermittelt, damit die Darstellung im CAVE entsprechend angepasst werden kann.

- **Aufgabe Client**

Sobald der Server gestartet wurde, können sich die Clients beim Server anmelden und erhalten in regelmässigen Abständen Informationen über den aktuellen Status des Spiel-, bzw. Simulationsablaufs. Weil jeder Client für eine Sicht (ein Auge) auf eine Leinwand im CAVE zuständig ist, wird der Bildausschnitt dementsprechend reduziert und die Bildinformationen werden für die stereoskopische Darstellung auf einen Projektor geschickt. Jeder Client rendert jeweils nur eine Sicht der Anwendung, so dass zwei Clients zusammen die 3D Illusion, mit je einem eigenen Projektor, darstellen können.

Der Ablauf des Spiels oder der Simulation wird hier nicht berechnet. Lediglich die Informationen, die nötig sind um die virtuelle Welt darzustellen, werden empfangen und interpretiert.

5.3 Typ „Eine Instanz“

Auch hier sind zwei Ansätze zu prüfen.

- **Nvidia GeForce**
Bloss eine Instanz der Unity Applikation wird ausgeführt. Die Aufteilung der Bildschirme geschieht über die Windows-Systemsteuerung. Die laufende Applikation muss die Auflösung aller kumulierten Bildschirme annehmen.
- **Mosaic**
Auch bei diesem Ansatz wird nur eine Instanz der Unity Applikation ausgeführt. Die Aufteilung der Bildschirme übernimmt jedoch der Mosaic Treiber, damit dem Betriebssystem ein einziger, grosser Bildschirm vorgegaukelt wird. Somit verteilt sich eine Fullscreen-Applikation automatisch auf alle Bildschirme / Beamer.

5.3.1 Architektur

Bei beiden Varianten sind vier GPUs vonnöten (respektive 8 Ausgänge), damit sämtliche 8 Beamer angesprochen werden können. Der grosse Vorteil liegt beim Auslassen der Synchronisierung. Die Applikation kann jede erdenkliche zufällige Wendung nehmen, da nur eine einzige Instanz läuft, in welcher mehrere Kameras gesetzt werden (um alle Leinwände zu bestrahlen und die Stereoskopie zu gewährleisten).

In jede Himmelsrichtung, die auch im CAVE physikalisch vorhanden sind, werden zwei Kameras gesetzt. Zwischen diesen Kameras gibt es jeweils einen leichten Versatz, um die stereoskopische Projektion zu gewährleisten.

Den Viewports der Kameras in der Unity Applikation unterliegen der genau gleichen Aufteilung wie die Bildschirme im Windows, bzw. Mosaic. Somit entspricht eine Kameraansicht genau einem Beamer im CAVE.

5.4 Funktionale Anforderungen

- **Multi-Monitor Displays**
Weil jede Leinwand nur einen Viertel des kompletten Bildes darstellt, liefert jeder einzelne Client selektiv Bildinformationen. Die erarbeitete Lösung sollte also in der Lage sein, basierend auf den Einstellungen des Clients, nur den entsprechenden Bildanteil zu projizieren.
- **Stereoskopie**
Nicht-Stereoskopische Anwendungen sollten in Stereoskopische umgewandelt werden können, wobei passive (Polarisierte Lichtprojektion) Stereoskopie unterstützt wird. Unity unterstützt diese Funktion.
- **Implementierung bestehender Unity Anwendungen**
Eine eigene Unity Anwendung in den CAVE zu implementieren sollte mit wenigen Clicks möglich sein. Die Rendering-Clients sind bereits für die jeweiligen Leinwände vorkonfiguriert. Es gilt lediglich, die erstellte Lösung in eine Unity 4.6 Anwendung mit vorhandenem Sourcecode einzubinden. Das Unity Projekt wird anschliessend für Windows exportiert und auf die Clients / den Server verteilt.
Nachdem die Anwendung auf dem Server gestartet wurde, können die Clients einzeln den Bootvorgang durchlaufen und sich mit dem Server verbinden, um die nötigen Informationen zur Darstellung der Projektion zu erhalten.
- **Mehrere Kameras in der Applikation**
Es soll möglich sein, dass sekundäre Kameras (Rückspiegel bei einem Auto, Minimap, Körperquerschnitt) unabhängig verteilt werden können.
 - Auf dem Frontscreen
 - Auf einer SeiteUnd jeweils an benutzerdefinierter Stelle anzeigen.

5.5 Argumentation

5.5.1 Pro

- **Keine Abhängigkeit**
Weil das gesamte System eine Eigenentwicklung ist, bestehen bis auf die Verbindung zu Unity und der benutzten Infrastruktur keinerlei Abhängigkeiten. Die maximale Flexibilität ist somit gewährleistet. Ausserdem entfällt das Einarbeiten in ein bestehendes Framework.
- **Community**
Unity ist ein sich stetig weiterentwickelndes Framework und geniesst eine immer grösser werdende Community, welche sich aktiv in Foren beteiligt. Sollten Probleme auftauchen, ist das Internet eine grosse Informationsquelle, die täglich an neuen Informationen reicher wird.
- **Netzverkauslastung**
Durch die Client-Server Architektur, bekannt aus Multiplayerspielen, die auch übers Internet gespielt werden, stellt das interne Netzwerk keinen Flaschenhals dar. Die nötigen Informationen für die Clients sind lediglich der Status des aktuellen Hauptspielablaufs.

5.5.2 Kontra

- **From Scratch**
Weil auf keine Bibliothek oder Framework zurückgegriffen werden kann, muss alles von Grund auf selber programmiert werden. Lediglich die Funktionen von Unity können und sollten Verwendung finden.
- **Aufwand**
Der Aufwand und die möglichen Probleme sind schwer abzuschätzen. Das gesamte System der Synchronisierung, der Stereoskopie und des Einpflegens in den CAVE muss geplant, umgesetzt und Debugged werden. Die Verwendung eines fertigen Frameworks, welches von einem Entwicklerteam stammt und sich in der Praxis bewährt hat, kann auf eine Robustheit zurückgreifen, die bei einer eigenen Entwicklung nicht per se gegeben ist.

5.6 Zusammenfassung

Einen eigenen Lösungsansatz zu verfolgen besticht durch seine Flexibilität. Mit der Game-Engine Unity wird auf das richtige Pferd gesetzt und ist zukunftsorientiert. Der Aufwand, ein eigenes Produkt zu erstellen ist im Gegensatz zur Implementierung eines fertigen Frameworks um einiges höher und birgt Gefahren. Deshalb ist die sorgfältige Analyse der bestehenden und geprüften Lösungen entscheidend. Die Implementierung mittels einer eigenen Lösung ist, mit entsprechendem Aufwand, vielversprechend.

6 MiddleVR

6.1 Idee

Da Unity bereits auf einer breiten Community abgestützt ist, und diese auch fördert mit beispielsweise dem Asset Store, kann die Evaluation eines bestehenden Produktes auch in die Lösungsvorschläge fließen.

6.2 Warum MiddleVR?

MiddleVR¹¹ (MiddleVR for Unity) hat uns dank seiner gut lesbaren Dokumentation und dessen Internetauftritt überzeugt, das Produkt konkreter zu beurteilen. Kurz und prägnant stellt es sich wie folgt vor:

MiddleVR erweitert Unity um folgende Funktionalitäten:

- Skalierung einer Visualisierung mit benutzerzentrierter Perspektive
- Support für 3D Interaktionsdevices wie 3D Trackers
- S3D – Aktive und Passive Stereoskopie
- Multiscreen- / Multicomputersynchronisierung für höhere Auflösungen und eindrucksvolle VR Systeme
- 3D Interaktionen: Navigation, Manipulation
- Immersive Menus
- Eigene grafische User Interfaces (in HTML5)
- Anzeige von beliebiger Website in der virtuellen Welt

Kreieren und erleben Sie interaktive & immersive VR Applikationen in wenigen Minuten dank dem simplen und mächtigen MiddleVR Plugin für Unity!

(Quelle: <http://www.middlevr.com/middlevr-for-unity/>)

6.3 Abdeckung MiddleVR

MiddleVR besteht aus zwei Hauptkomponenten. Einer vereinfachten Erstellung und Programmierung von VR Anwendungen und der Möglichkeit, verschiedenste VR Hardware und 3D Anwendungen zu adaptieren.

- **Stereoskopie**

MiddleVR weist darauf hin, wenn aktive Stereoskopie verwendet wird, dass eine Unity Pro – Lizenz vorhanden sein muss, und nur eine Handvoll GPUs unterstützt wird. (<http://www.middlevr.com/doc/current/#stereoscopy---s3d>)

Da der CAVE der BFH aber mit passiver Stereoskopie arbeitet, sollte dies uns hier nicht betreffen.

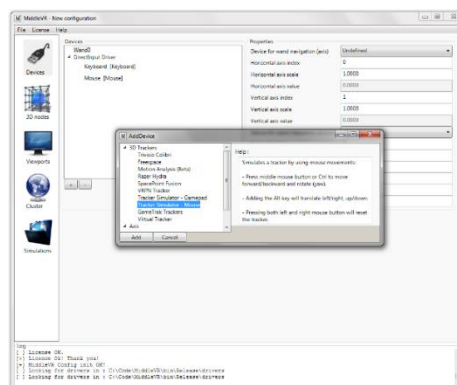


Abbildung 7: MiddleVR Konfigurator

¹¹ <http://www.middlevr.com/> middleVR

- **Funktionalen Anforderungen**
Unsere definierten funktionalen Anforderungen (siehe Pflichtenheft), werden komplett abgedeckt, da es bereits als Unity Asset (Plugin) funktioniert.
- **Nicht funktionalen Anforderungen**
Die nicht funktionalen Anforderungen beinhalten die Ergonomie, die Anwendung des Plugins und die Wiederverwendbarkeit von MiddleVR.

6.4 Konzept MiddleVR

MiddleVR funktioniert wie folgt:

1. Beschreibung des VR Systems
2. Generierung der Konfiguration für das beschriebene VR System
3. Verwendung dieser Konfiguration durch das Unity Plugin

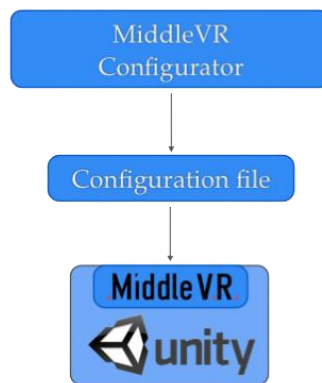


Abbildung 8: Basiskonzept MiddleVR

- **Beschreibung**
Die Beschreibung umfasst folgende Punkte:
 - Die Devices des VR Systems
 - Eine Beschreibung, wie das diese Devices mit der realen Welt interagieren.
(Beispielsweise Tracker A folgt dem Kopf des Benutzers, Tracker B folgt dessen linker Hand)
 - Die Positionen der Screens
 - Welche Kameras was wo rendern müssen
- **Beispiel**
Die Beispielkonfiguration umfasst einen 5-Seiten CAVE mit 5 Clustern.
Anmerkung: das Setting in der BFH wird sich unterscheiden bei den Anzahl Clustern (1 Cluster pro Viewport), sowie der Ausrichtung (hier von Innen nach Aussen), und natürlich den ganzen Parametern.

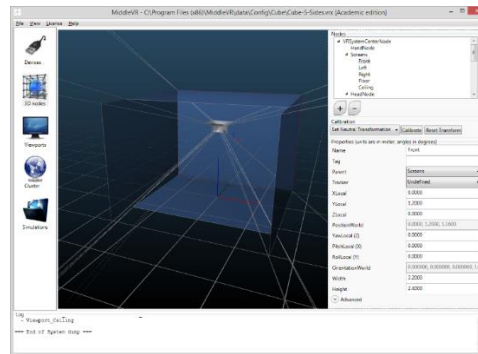


Abbildung 9: Beispielskonfiguration CAVE

6.5 Argumentation

6.5.1 Pro

- **Bereits ausgearbeitete Software**

Da es sich um eine professionelle Software handelt, würde viel oder fast alles an Programmierung ausserhalb von Unity entfallen. Ob Scripts (Kamera oder Assets) angepasst oder selbst geschrieben werden müssen, oder eventuell sogar ein eigener Konfigurator (als Alternative zum MiddleVR Konfigurator) erstellt werden müsste, ist unbekannt.

- **Aufwand**

Da alle nötigen Parameter bereits vorhanden sind für den CAVE zu konfigurieren, fällt ein grosser Teil des Designs weg. Dies aber nur im optimalen Fall, wenn das komplette MiddleVR verwendet werden kann.

- **Support**

MiddleVR ist ein Produkt, welches in mehreren Lizenzmodellen kommt. Somit wäre auch ein aktueller Support vorhanden.

6.5.2 Kontra

- **Abhängigkeit**

Diese Lösung ist stark an MiddleVR gebunden. Falls während der Prototypingphase unüberbrückbare Probleme auftreten, könnte das Projekt nicht realisiert werden.

- **Eigenanteil**

Für den Stolz der CPVR-Studenten ist es auch wichtig, einen signifikanten Eigenanteil beizusteuern.

Im günstigsten Fall funktioniert die Software nach kurzer Konfiguration.

6.6 Zusammenfassung

MiddleVR bietet sich als Lösung an. Es kann jedoch erst beim Prototyping entschieden werden, ob die vorhandenen Features, Möglichkeiten und Dokumentationen ausreichend sind, um den CAVE der BFH abzubilden.

7 Gegenüberstellung Varianten

Basierend auf den Analysen wurden die Pros und Kontras zusammengefasst, um einen Vergleich aufzustellen. Die verschiedenen Kategorien werden kurz erläutert und mit einer Relevanz gewichtet.

		Support	Keyfeatures	Verbindung zu Unity	Architektur
Chromium			<ul style="list-style-type: none"> Stereoskopie Aufteilung der Monitore 		Gegebene Client / Server Struktur im CAVE
		Eingestellt	<ul style="list-style-type: none"> Kein Tracking Mittels Unity keinen grossen Einfluss 	<ul style="list-style-type: none"> Kein direkter Unity Support OpenGL nur bis Version 1.5 	Engpass Netzwerkauslastung
Equalizer		Dank Open Source vorhandene Community	Graphikklassen vorhanden (KI & Physik nicht)		Vorhandene Grafikklassen
		Open Source	Wahrscheinlich grosser Teil nicht vorhanden	<ul style="list-style-type: none"> Kein direkter Unity Support C++ Implementation 	Starre Struktur gegeben durch Wrapperklassen
Eigene Lösung		Viele aktive Unity Entwickler kontaktierbar, jedoch bezüglich CAVE eingeschränkt.	Mit entsprechen-dem Aufwand nahezu alles machbar	Umsetzung direkt in Unity	Gut an CAVE adaptierbar
		Auf eigenes Unitywissen angewiesen	Nur mit entsprechendem Aufwand möglich	Ausschliesslich Möglichkeiten von Unity	Eigene Architektur notwendig
MiddleVR		Professionelle und aktuelle Software	<ul style="list-style-type: none"> Stereoskopie Aufteilung der Monitore Tracking 	Als Package für Unity vorhanden	Auf Unity basierend
		Support über Internet		Geknüpft an Unity Version, Features	Geringe Flexibilität

Tabelle 2: Gegenüberstellung Pro und Kontra

Daraus ergeben sich folgende Diagramme.

(Info: Falls eine Kategorie nicht genau bekannt war, wurde mit der vorhandenen Erfahrung geschätzt)

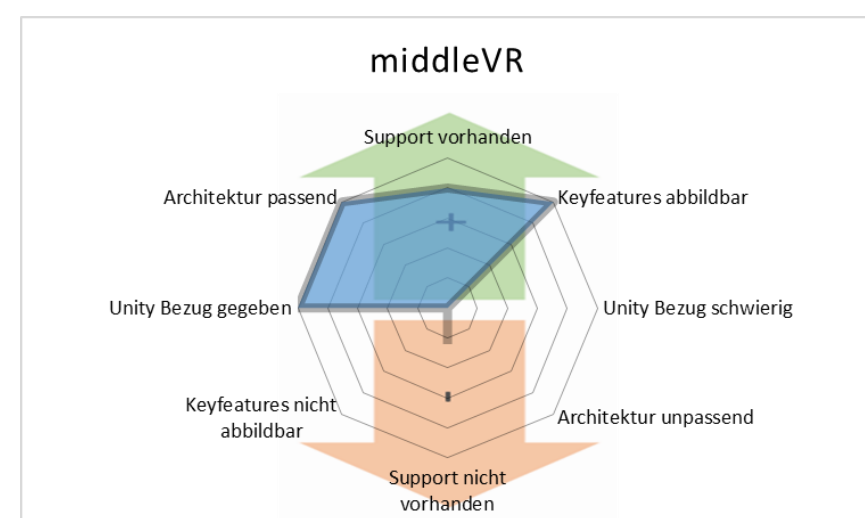
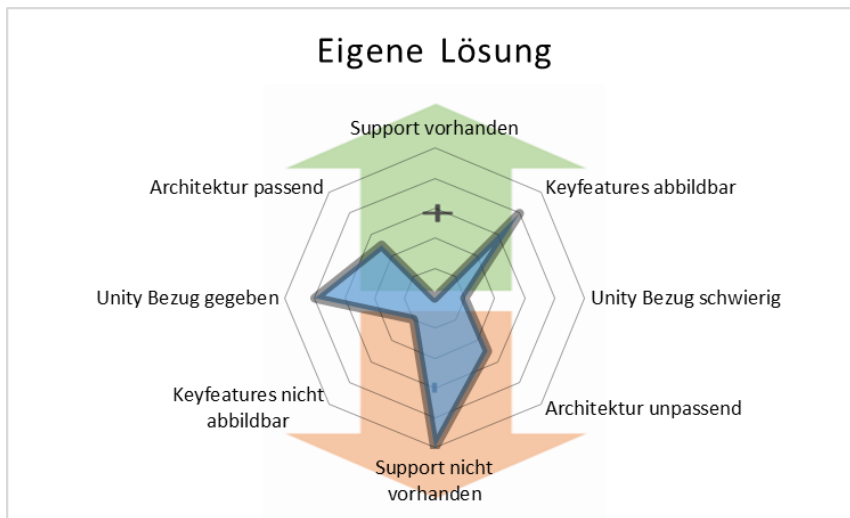
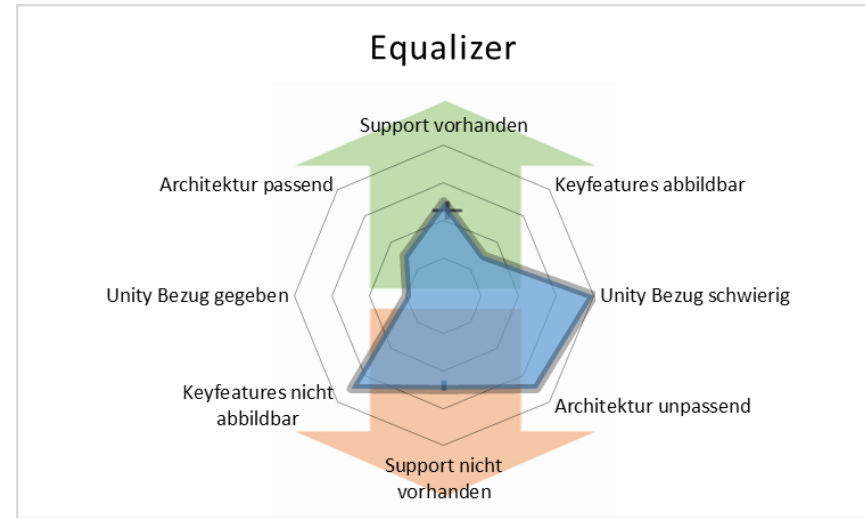
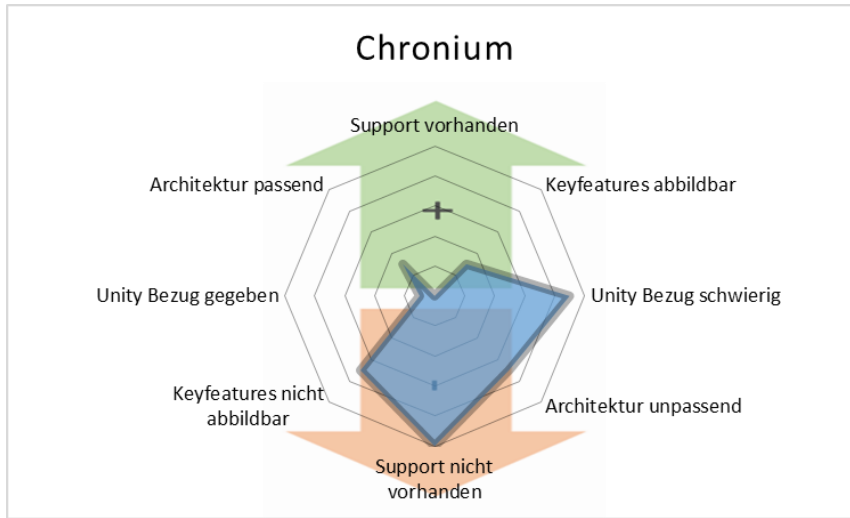


Abbildung 10: Vergleich Varianten

8 Entscheid Variante

Ausgehend von den Pros und Kontras der verschiedenen Möglichkeiten hat sich ein Trend abgezeichnet. **MiddleVR und die eigene Lösung** konnten sich von den übrigen Methoden abheben. MiddleVR bietet eine sichere, geprüfte und supportete Lösung um Unity in den CAVE zu integrieren, gleichzeitig ermöglicht die eigene Lösung eine grösstmögliche Flexibilität.

Dank den Analysen konnten mögliche Probleme weitgehend ausgeschlossen werden, bei der Umsetzung der ersten Prototypen besteht aber nach wie vor die Gefahr, auf ungeahnte und unlösbare Probleme zu stossen. Darum wird an dieser Stelle eine parallele Strategie gefahren, um schnell reagieren zu können.

Mit Absprache unseres Betreuers werden die Varianten MiddleVR und die eigene Lösung verfolgt und in einem nächsten Schritt erfolgt die Umsetzung der Prototypen mit anschliessender Auswahl des final zu implementierenden Systems.

9 Ausgangslage Prototyping

Mit unseren ausgewählten Kandidaten (**MiddleVR und Eigene Lösung**) sind verschiedene Varianten denkbar, um einen CAVE zu implementieren. Jede dieser Methoden hat gewisse Vor- und Nachteile bezüglich Aufwand, Performance, Infrastruktur und Lizenzierungskosten. Nun gilt es in einem nächsten Schritt die Varianten gegenüberzustellen, um eine gute Basis für den anschliessenden Entscheid zu schaffen. Ein wichtiger Faktor sind die entstehenden Kosten. Diese können durch Aufrüstung der Infrastruktur oder Lizenzierung entstehen und es muss abgeschätzt werden, ob die Mehrkosten zu den entstandenen Vorteilen tragbar sind.

Basierend auf der aktuellen Ausgangslage sind sämtliche in Erwägung gezogenen Varianten mittels MiddleVR oder einer eigenen Lösung möglich. Folgend werden die vier Varianten, unabhängig der tatsächlichen Implementierung, detailliert eruiert.

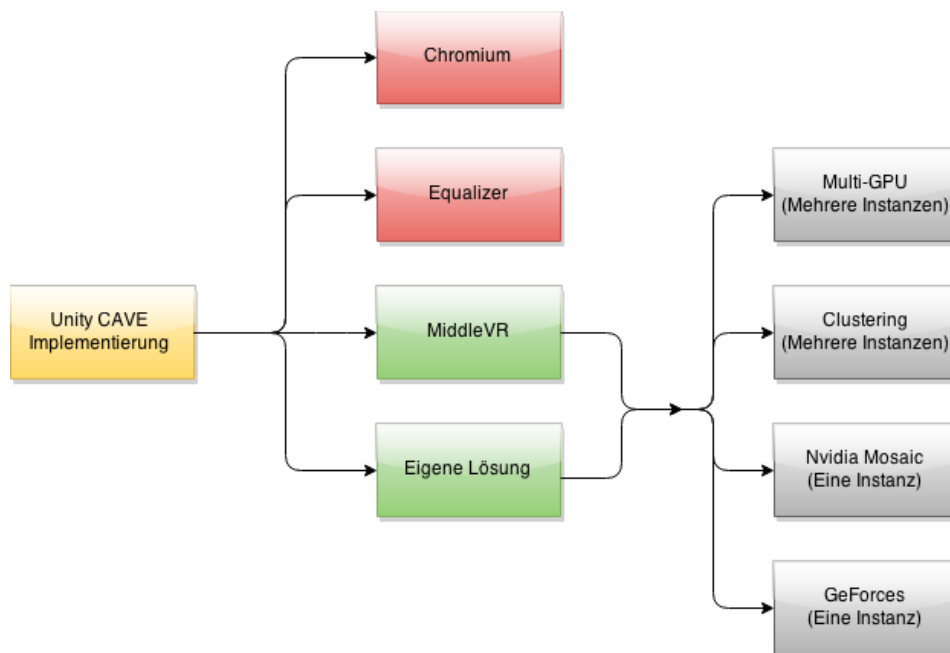


Abbildung 11: Evaluierung Varianten, Quelle: <http://draw.io>

9.1 Aktuelle Infrastruktur

Beim aktuellen Setting des CAVEs sind 8 Clients und 1 Server im Einsatz, je mit einer Grafikkarte mit 2 Outputs. Nur einer dieser Outputs führt zu einem Beamer, der andere dient als Admin-Konsole. So können zur Laufzeit, ohne störende Unterbrüche auf der CAVE-Leinwand, Adaptionen vorgenommen werden.

Das bedeutet, falls diese Infrastruktur übernommen wird, dass auf jedem Client eine Unity-Instanz laufen muss und über das Netzwerk erfolgt die Synchronisierung mittels Clustering. Ein Client ist somit zuständig für einen Teil der stereoskopischen Projektion (Z.B. das Bild für das linke Auge auf der linken Leinwand).

Die nachfolgend diskutierten Ansätze haben einen möglichen Einfluss auf die aktuelle Infrastruktur. Um aber auch ohne Anpassungen des Settings eine Implementierung sicherstellen zu können, wird unabhängig vom schlussendlich gewählten Ansatz ein Prototyp dieser Art umgesetzt. MiddleVR setzt bei der Verwendung eines Cluster-Systems eine Academic oder Pro Lizenz voraus. Damit keine unnötigen Kosten entstehen, wird in einem Zeitfenster von 30 Tagen die Testversion gelöst und ein Prototyp eingebaut.

9.2 Multi-GPU

Eine Möglichkeit wäre, eine Unity-Anwendung mittels mehreren GPUs auf einem Rechner parallel zu rechnen, um eine grösstmögliche Performance zu erreichen. Die Synchronisierung erfolgt demnach nicht über das Netzwerk, sondern die GPUs müssen intern abgeglichen werden. Der mögliche Netzwerk-Flaschenhals wird vermieden. Obwohl alle Berechnungen auf einem Client laufen, gilt es trotzdem die Herausforderung der Synchronisierung zu meistern, analog dem Clustering über das Netzwerk.

„Nevertheless the Multi-GPU option comes at a price because it forces the use of clustering and so to manage its difficulties“ (Zitat MiddleVR Dokumentation, Quelle: <http://www.middlevr.com/doc/current/>).

Jede GPU hat eine eigene Unity-Instanz am Laufen. Erste Prototypen haben zu Tage gebracht, dass die Synchronisierung je nach Feature Probleme verursachen kann und Anpassungen am Quellcode erforderlich werden.

- **Lizenzen**

Pro GPU muss eine MiddleVR Academic-, bzw. Pro-Lizenz gelöst werden. Diese Kosten entfallen, wird die eigene Lösung verfolgt.

Komponente	Anzahl	Preis pro Stück €	Preis gesamt €
MiddleVR Academic Licence	4	3000.-	12'000.-
			12'000.-

Tabelle 3: Geschätzte Kosten MiddleVR Lizenzen Multi-GPU

- **Adaption Infrastruktur**

Ein komplett neuer Rechner mit 4 Grafikkarten muss angeschafft und am CAVE angeschlossen werden.

Komponente	Anzahl	Preis pro Stück CHF	Preis gesamt CHF
Grafikkarte - GeForce GTX 760	4	260.-	1040.-
RAM - 16GB DDR3-1600	1	120.-	120.-
Mainboard - ASUS Z9PE-D8 WS	1	439.-	439.-
CPU - Intel Core i7 4820K	2	347.-	694.-
SSD - Samsung 850 EVO Basic 500 GB	1	214.-	214.-
Server Case	1	200.-	200.-
Netzteil	1	150.-	150.-
Verkabelung			100.-
			2957.-

Tabelle 4: Geschätzte Kosten Infrastruktur Multi-GPU, Quelle: <http://digitec.ch> (19.04.2015)

- **Vorteile**

- Maximale Performance dank Verwendung mehrerer GPUs
- Keine Netzwerk-Einschränkungen
- Ein einzelner Rechner

- **Nachteile**

- Anschaffung neuer Komponenten
- Synchronisierung mehrerer Unity-Instanzen
- Lösen mehrerer Lizenzen
- Inbetriebnahme eines neuen Systems in den CAVE

- Relativ hohe Lizenzkosten
- Möglicher Flaschenhals CPU

- **Fazit**

Das primäre Ziel der Arbeit ist nicht ein hoch performanter CAVE, um sehr rechenintensive Anwendungen eindrucksvoll darstellen zu können, sondern vielmehr ein Showroom, um Spiele, die hauptsächlich am eigenen Computer Zuhause Verwendung finden, stereoskopisch darzustellen. Der dominante Vorteil, nämlich die maximale Performance, wird in unserem Projekt dementsprechend degradiert und hat nicht die stärkste Gewichtung. Denn kann ein Unity-Spiel am eigenen Rechner flüssig gespielt werden, reicht im CAVE eine einzelne Grafikkarte theoretisch auch.

Die zu tragenden Kosten sind verhältnismässig hoch und falls nur ein Rechner zum Einsatz kommt, ist es wünschenswert, die Problematik der Synchronisierung eliminieren zu können, was da leider nicht der Fall ist.

9.3 Clustering

Basierend auf dem aktuellen CAVE Setting ist es möglich ein Clustering zu erstellen, wie es heute auch vom Equalizer-Framework verwendet wird. Die Unity Anwendung wird dann als komplette Netzwerk-anwendung laufen, d.h. es kommt eine ähnliche Synchronisierung zum Zuge, welche auch bei Multi-playerspielen über das Internet verwendet wird.

Unity unterscheidet zwischen zwei verschiedenen Konzepten:

- **Authorative Server:** Verwalten die gesamte Spielszene, Clients nehmen keine Änderungen vor. Sie senden diese Änderungen (Tastatureingaben, etc) an den Server.
- **Nicht authorative Server:** Alle Clients berechnen ihre Spielwelt selbst und bekommen nur neue Positionen für die Objekte von den anderen Clients.

Diese verschiedenen Architekturen müssen bei den Prototypen berücksichtigt werden.

- **Lizenzen**

Falls der MiddleVR-Ansatz verfolgt wird, muss pro Cluster eine MiddleVR Academic-, bzw. Pro-Lizenz gelöst werden. Je nach Serversystem muss noch eine zusätzliche Lizenz gelöst werden. Bei der eigenen Lösung entfallen diese Kosten.

Komponente	Anzahl	Preis pro Stück €	Preis gesamt €
MiddleVR Academic Licence	8	3000.-	24'000.-
MiddleVR Academic Licence	1	3000.-	3000.-
			27'000.-

Tabelle 5: Geschätzte Kosten MiddleVR Lizenzen Cluster 8 Clients

Komponente	Anzahl	Preis pro Stück €	Preis gesamt €
MiddleVR Academic Licence	4	3000.-	12'000.-
MiddleVR Academic Licence	1	3000.-	3000.-
			15'000.-

Tabelle 6: Geschätzte Kosten MiddleVR Lizenzen Cluster 4 Clients

- **Vorteile**

- Aktuelle Infrastruktur muss nicht verändert werden.
- Skalierbar. Beliebige viele Clients können dazu geschaltet werden.

- Keine Limitierung bei CPUs. Jede Unity Instanz hat vollen Zugriff auf den CPU.
- **Nachteile**
 - Kosten
 - Netzwerksynchronisierung (Kann u.U. aufwändig werden, aber prinzipiell möglich)
- **Fazit**

Diese Variante benötigt keinen Umbau. Technisch sicherlich nicht die trivialste Variante, aber Unity und MiddleVR sind für solche Settings ausgelegt. Weiter können sich hier doch noch einige Fehlerquellen offenbaren. Wie ist das Netzwerksetting genau im CAVE? Sind die aktuellen Rechner und Graphikkarten 100% kompatibel? In unseren Augen sind aber die Lizenzkosten der springende Punkt.

9.4 Mosaic

Mosaic ist eine Technologie, um mehrere Graphikkarten zu verlinken und auf einem Desktop darzustellen. Windows wird ein einzelner Output vorgegaukelt, auch wenn physisch mehrere GPUs mit multiplen Ausgängen angeschlossen sind.

„Die NVIDIA® Mosaic™ Mehrbildschirm-Technologie dient zur einfachen Skalierung jeder Anwendung auf mehrere Bildschirme, und das ohne Softwareanpassungen oder Leistungseinbußen. Durch die Mosaic Technologie werden Mehrbildschirm-Konfigurationen vom Betriebssystem als einzelner Bildschirm wahrgenommen.“

(Quelle: <http://www.nvidia.de/object/nvidia-mosaic-technology-de.html>)

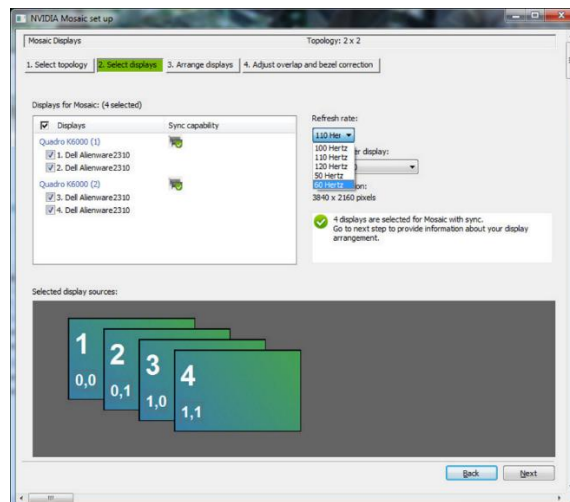


Abbildung 12: Mosaic Einstellungen, Quelle: <http://nvidia.custhelp.com/rnt/rnw/img/enduser/3568-7.png>

- **Limitierungen**

Aus dem Factsheet von Nvidia können folgende Limitierungen entnommen werden:

 1. **Quadros**

Mosaic ist nur mit Quadro Grafikkarten kompatibel. Die günstigeren und weiter verarbeiteten GeForces können nicht eingesetzt werden.
 2. **Auflösung** (Quadro K5000/5000/6000, Quadro Plex 7000)

Maximal 16384*16384 Pixel. (Dies entspricht 4x einer 4K Auflösung horizontal und vertikal)
 3. **Maximale Displayanzahl** (Quadro K5000) Maximal 16 Displays mit vier Quadrokarten.

(Quelle: http://www.nvidia.com/docs/IO/40049/NVMosaic_UC_v05.pdf)

- **Kosten und Lizenzen**

Quadrokomponenten müssten zusätzlich angeschafft werden. Hier ein Beispielsetting:

Komponente	Anzahl	Preis pro Stück CHF	Preis gesamt CHF
Grafikkarte - Nvidia Quadro K5000	4	1797.-	7188.-
RAM – 16GB DDR3-1600	1	120.-	120.-
Mainboard – MSI BIG BANG-MARSHAL (B3)	1	---	---
CPU - Intel Core i7 4820K	2	347.-	694.-
SSD - Samsung 850 EVO Basic 500 GB	1	214.-	214.-
Server Case	1	200.-	200.-
Netzteil	1	150.-	150.-
Verkabelung			100.-
			8666.-

Tabelle 7: Nvidia Quadro K5000 Beispielaufbau

(Der Preis des MSI BIG Bang Marshal – Mainboards ist z.Z. nicht bekannt. Es ist ein spezielles Mainboard mit 8 PCI-Express Slots nötig, damit 4 doppelte Graphikkarten verwendet werden können.)

Komponente	Anzahl	Preis pro Stück €	Preis gesamt €
MiddleVR Academic Licence	1	3000.-	3000.-
			3'000.-

Tabelle 8: Nvidia Quadro K5000 MiddleVR Lizenz

- **Vorteile**

- Fullscreen Funktionalität der Unity Applikationen wird unterstützt

- **Nachteile**

- Neue Hardware benötigt (Nvidia Quadro GPUs)

- **Fazit**

Eine Mosaic Lösung wäre denkbar, man würde sich sogar ganz vom herausfordernden Clustersystem verabschieden. Laut Aussage von MiddleVR sollte diese Variante jedoch möglich sein, aber es ist mit Performanceeinbussen zu rechnen, weil (wiederum laut MiddleVR) angeblich nur eine Grafikkarte die Berechnungen vornimmt und mit Hilfe der „sekundären“ GPUs das Bild schlussendlich dargestellt wird. Diese Aussage deckt sich aber nicht mit der Dokumentation von Nvidia.

9.5 GeForce

Ähnlich der Mosaic Variante ist ein Setting mit GeForce Grafikkarten denkbar. Der Einsatz vom Mosaic Treiber wird zwar verunmöglicht (fehlende Quadro Karten), die Anordnung der Screen erfolgt aber über die Windows-Einstellungen und die Unity Applikation kann in einer Fensteransicht über sämtliche Screens vergrößert werden.

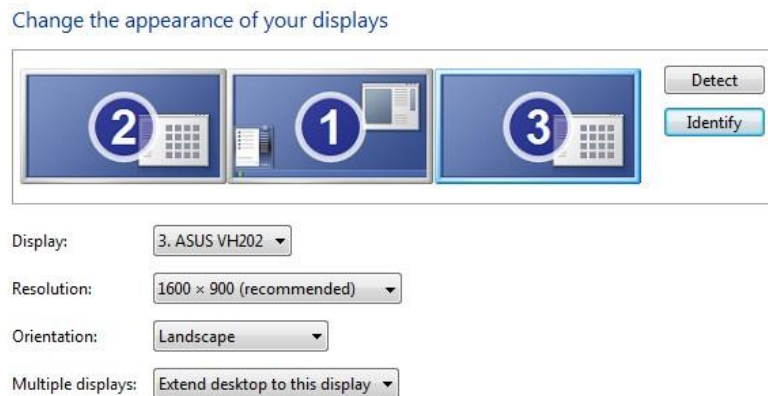


Abbildung 13: Windows Monitor Settings

- **Lizenzen**

Falls nicht eine eigene Implementierung vorgesehen ist, muss eine MiddleVR Academic Licence gekauft werden.

Komponente	Anzahl	Preis pro Stück €	Preis gesamt €
MiddleVR Academic Licence	1	3000.-	3000.-
			3'000.-

- **Adaption Infrastruktur**

Dieselbe Infrastruktur wie bei der Multi-GPU Variante kann Verwendung finden.

- **Vorteile**

- Im Vergleich zu den Quadro-Karten sind die hier angedachten GeForce-Karten kostengünstiger.
- Synchronisierung entfällt

- **Nachteile**

- Möglicherweise können nur bis maximal 3 Screens zur gleichen Zeit angesprochen werden. (Wenn kein Nvidia SLI verwendet wird)
- Performance-Einschränkungen, weil möglicherweise sämtliche Berechnungen nur über 1 GPU laufen.
- Im Vergleich zu Mosaic erfolgt die Aufteilung der Screens nicht auf Treiberebene.
- Die Fullscreen-Option wird immer nur auf einen Screen angewandt.

- **Fazit**

Ein Testing mit einem Prototypen ist an dieser Stelle unerlässlich. Die Anforderungen können nicht rein theoretisch abgeklärt werden und müssen der Praxis standhalten. Aus anwendungstechnischer Sicht ist die Mosaic-Lösung zu bevorzugen, wird jedoch im Verlauf des Testings festgestellt, dass die Performance nicht leidet, könnte diese kostensparende Variante durchaus das Rennen machen.

10 Prototypen

Ein wichtiger Faktor ist, je nach Setting und Ansatz, die Synchronisierung der verschiedenen Clients, bzw. GPUs. Um möglichst viele Fälle zu testen, werden Prototypen mit unterschiedlichen Anforderungen an die Infrastruktur integriert.

Kritische Punkte bezüglich Synchronisierung bei der Darstellung sind unter anderem:

- Die Kamera
- Dynamischen Objekte
- Shader-Effekte
- Grosser Datentransfer über das Netzwerk
- Zufallsscripts

Mit den Prototypen soll eruiert werden, wo möglicherweise Abstriche gemacht werden müssen und welche Features von Unity problemlos integriert werden können. Zudem kann ein direkter Vergleich der Performance erfolgen.

10.1 Basic

Der Basic-Prototyp dient lediglich als Basisanwendung, um die grundlegende Lauffähigkeit der Ansätze testen zu können. Bei der Synchronisierung soll kein grosser Datentransfer entstehen.

In der statischen virtuellen Welt befindet sich nur ein Cube, eine Ebene als Terrain ohne jegliches Material und ein First-Person-Controller, um die Transformationen des Cubes und der rudimentären Landschaft zu visualisieren. Ein Directional Light dient als Lichtquelle.

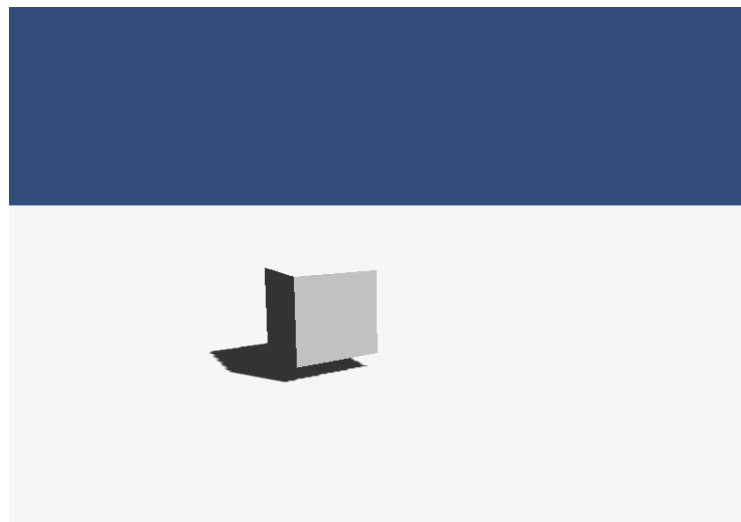


Abbildung 14: Basic Prototyp

10.2 Basic Extended

Dies ist eine leicht erweiterte Version des Basic Prototypen. Es wurden Seitenwände mit einem Schachbrettmuster eingefügt, um die Perspektive und Ausrichtung der Kameras auf den vier Wänden im CAVE zu prüfen.

Weiter sind die Bälle der Physik unterliegend und sollten, falls eine Clustering-Methode getestet wird, zwischen den verschiedenen Clients abgeglichen werden. Per Knopfdruck können die Farben der Bälle

zufällig gesetzt werden. Weil dieser Input nur auf dem Master erfolgt, sollten die restlichen Clients diese Information mitgeteilt kriegen und die eigene Unity-Instanz entsprechend updaten.

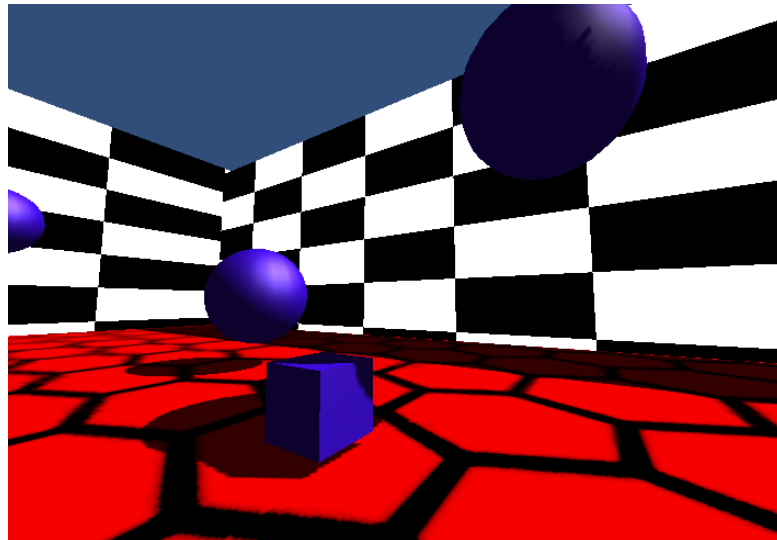


Abbildung 15: Basic-Extended Prototyp

10.3 AngryBots

Mit Unity wurde bis und mit Version 4 das Beispielprojekt AngryBots mitgeliefert, welcher einen geeigneten Showcase der Möglichkeiten von Unity bietet. Etliche Effekte, detaillierte Models, Kampfhandlungen und weitere Keyfeatures von Games sind darin enthalten.

Weil MiddleVR zum aktuellen Zeitpunkt Unity5 nur mit einem Beta-Release unterstützt, wird mit diesem Prototypen noch auf Unity4 gesetzt.



Abbildung 16: AngryBots, Quelle: <http://www.androidmag.de/>

Sämtliche kritischen Punkte, mit Hauptaugenmerk auf visuellen Effekten und der Kamera, können mittels diesem Prototypen eruiert werden und ist gewissermassen ein Härtetest und vertritt optimal die Bedingungen eines realen Games, wie es später eingesetzt werden könnte.

10.4 Car Tutorial

Ein weiteres Beispielprojekt ist ein Autorennspiel, genannt « Car Tutorial ». Es basiert auf Unity3.3. Dieses Tutorial besitzt ein realistisches Autofahrverhalten und eine verbesserte Physik.



Abbildung 17: Car Tutorial, Quelle: <http://blogs.unity3d.com/2010/04/24/car-tutorial/>

Das Hauptaugenmerk liegt auf der Physik, sowie den speziellen Wheelcollidern.

11 Finale Gegenüberstellung

11.1 Übersicht

In der folgenden tabellarischen Übersicht werden sämtliche Möglichkeiten aufgezeigt, die mittels den ausgewählten Prototypen gewertet werden. Bei jeder Variante werden die wichtigsten Pro- und Kontra-Argumente aufgelistet, um eine Basis für den finalen Entscheid zu schaffen.


		Einzelne Unity Instanz		Mehrere Unity Instanzen	
		Mosaic	GeForces	Clustering	Multi-GPU
MiddleVR	👍	<ul style="list-style-type: none"> Keine Synchronisierung nötig Einfache Kamera-Einstellungen Treiber verteilt Rendering auf alle GPUs 	<ul style="list-style-type: none"> Keine Synchronisierung nötig Einfache Kamera-Einstellungen Setting mit günstigen GeForce Karten möglich 	<ul style="list-style-type: none"> Verteiltes Rendering Einfache Kamera-Einstellungen Zentrale Steuerung aller Clients Skalierbarkeit 	<ul style="list-style-type: none"> Verteiltes Rendering Einfache Kamera-Einstellungen Zentrale Steuerung aller Clients Keine Netzwerkeinschränkungen
	👎	<ul style="list-style-type: none"> Max. Anzahl Screens auf physische Outputs beschränkt (abhängig der Hardware, aber voraussichtlich 16 möglich) Kostenfaktor 	<ul style="list-style-type: none"> Rendering nur auf einem GPU Restliche GPUs nur für Anzeige Last wird dementsprechend nicht verteilt Keine Native-Fullscreen-Funktionalität, muss mittels Popuwindow erzwungen werden Max. Anzahl Screens auf physische Outputs beschränkt Kostenfaktor 	<ul style="list-style-type: none"> Nur Position der Objekte / Kamera wird synchronisiert Probleme bei Shadern / Model- und Materialänderungen Spezifische Kamera-Scripts verursachen Probleme Kostenfaktor 	<ul style="list-style-type: none"> Max. Anzahl Screens auf physische Outputs beschränkt Möglicher Flaschenhals CPU Nur Position der Objekte / Kamera wird synchronisiert Probleme bei Shadern / Model- und Materialänderungen Spezifische Kamera-Scripts verursachen Probleme Kostenfaktor
Eigene Lösung	👍	<ul style="list-style-type: none"> Hohe Flexibilität Keine Synchronisierung nötig Treiber verteilt Rendering auf alle GPUs 	<ul style="list-style-type: none"> Hohe Flexibilität Keine Synchronisierung nötig 	<ul style="list-style-type: none"> Hohe Flexibilität Verteiltes Rendering Skalierbarkeit 	<ul style="list-style-type: none"> Hohe Flexibilität Verteiltes Rendering Keine Netzwerkeinschränkungen
	👎	<ul style="list-style-type: none"> Max. Anzahl Screens auf physische Outputs beschränkt (abhängig der Hardware, aber voraussichtlich 16 möglich) Aufwand eigene Implementierung 	<ul style="list-style-type: none"> Rendering nur auf einem GPU Restliche GPUs nur für Anzeige Last wird dementsprechend nicht verteilt Keine Native-Fullscreen-Funktionalität, muss mittels Popuwindow erzwungen werden Max. Anzahl Screens auf physische Outputs beschränkt Aufwand eigene Implementierung 	<ul style="list-style-type: none"> Aufwand eigene Implementierung Implementierung Netzwerksynchronisierung 	<ul style="list-style-type: none"> Aufwand eigene Implementierung Implementierung GPU-Synchronisierung Max. Anzahl Screens auf physische Outputs beschränkt Möglicher Flaschenhals CPU

Tabelle 9: Finale Gegenüberstellung

11.2 Auswertung

Die Prototypen wurden in einem ersten Anlauf allesamt mit MiddleVR integriert. Weil die Ansätze „Multi-GPU“ und „Clustering“ sehr grosse Ähnlichkeiten aufweisen, nämlich Systeme mit mehreren Instanzen sind, bilden diese zusammen eine Auswertungskategorie und Mosaic und GeForces eine Zweite.

- **Typ „Mehrere Unity Instanzen“**

Die Hauptproblematik liegt wie befürchtet bei der Synchronisierung der verschiedenen Instanzen. Das Vertrauen in MiddleVR war gross und es wurde angenommen, dass die kompletten States der unabhängig laufenden Unity-Instanzen synchronisiert werden. Leider musste festgestellt werden, dass grundsätzlich nur die Position der Objekte ausgetauscht wird.

Sobald eine Veränderung vorgenommen wurde, die über die Position hinausgeht (beispielsweise Farbe des Materials oder Objekt komplett entfernen), **fand kein Abgleich mehr statt**. Konkret bei den eingesetzten Prototypen sind das die zufälligen Farben der Kugeln (Basic Extended) und das Vernichten eines Gegners (AngryBots). Grundsätzlich hat die Implementierung mit einfachen Anwendungen, die einen eher statischen Ablauf aufweisen, gut funktioniert.

MiddleVR bietet zwar Wrapperklassen an um diesem Problem entgegenzutreten, das bedeutet aber, dass die Unity Applikationen vorgängig analysiert und in unbestimmten Ausmass angepasst werden müssen. Ausserdem haben die von MiddleVR angegebenen Adaptionen auf Anhieb nicht funktioniert. Selbst für die von Unity mitgelieferten Beispielprojekte muss laut offizieller Doku von MiddleVR eine Liste von Anpassungen abgearbeitet werden, um eine Synchronisierung zu gewährleisten.

Die Zufallselemente jedoch haben bisher keine Probleme verursacht. Bei jeder Instanz wird derselbe Seed gesetzt um einen parallelen Ablauf sicherzustellen.

- **Typ „Einzelne Unity Instanz“**

Die Hauptbefürchtung, dass die Performance der Flaschenhals sein würde wenn die Berechnungen nicht auf verschiedene Clients verteilt werden können, hat sich nicht bestätigt. Gegenteilig, es konnte sogar eine bessere und konstantere Bildrate erreicht werden. Beim direkten Vergleich ist aber Vorsicht geboten, weil die Tests nicht immer auf derselben Hardware durchgeführt wurden.

Nachdem die Inbetriebnahme von MiddleVR mittels Mosaic relativ problemfrei verlief, stellte sich zwangsläufig die Frage, welche Aufgaben MiddleVR ohne Clustering schlussendlich noch übernimmt. Einzig die zusätzlichen Kameras werden in den Unity-Anwendungen gemäss dem definierten CAVE gesetzt.

Basierend auf der Auswertung und den gewonnenen MiddleVR Erkenntnissen, wurde ein eigener rudimentärer Prototyp realisiert und mit dem Mosaic Setting in Betrieb genommen. Die Resultate waren auf Anhieb vielversprechend und machen MiddleVR somit weitgehend obsolet, sofern in einer weiterführenden Arbeit eine eigene Umsetzung geplant ist. Es konnten keine kritischen Punkte eruiert werden, die das relativ teure Tool MiddleVR unabdingbar machen würden.

11.3 Offene Punkte

11.3.1 Mehrere Kameras

Je nach Spiel, Applikation kann es vorkommen, dass neben einer Hauptkamera (deren Viewport an die Wände des CAVEs projiziert werden), noch mehrere sekundäre Kameras vorhanden sind. Dies können eine Minimap, ein Querschnitt eines Körpers, eine Aussenkammer oder auch nur Rückspiegel bei einem Fahrzeug sein.

Mittels folgendem Ansatz könnte diesem Problem entgegengewirkt werden: Dem Benutzer ist es frei zu wählen, wo seine sekundären Kameras platziert werden. An einer Seite des CAVE's, an der Frontseite, etc.

11.4 GPU Performance

Eine wichtiger Punkt beim Typ „Einzelne Unity Instanz“ ist die Auslastung der GPUs. Eine Verteilung der Last ist erstrebenswert, weil die Berechnungen nicht nur auf einem GPU laufen sollten und die restlichen Karten lediglich zum Darstellen der Bildinformationen dienen. Die beiden Systeme „GeForces“ und „Mosaic“ wurden mit dem Monitoring Tool „TechPowerUp GPU-Z“¹² überprüft und ausgewertet.

- **GeForces**

Sehr auffällig ist die intensive Auslastung der ersten Grafikkarte. Unabhängig davon, auf welchem Screen viele Berechnungen anfallen, bleibt dieser GPU Load maximal hoch und die restlichen Karten sind nur bedingt am Berechnen. Der Schluss liegt deshalb nahe, dass Windows die Berechnungen nur durch die primäre Grafikkarte erledigen lässt und die berechneten Informationen an die restlichen Karten weiterverteilt.

So kann also bei weitem nicht die maximale Performance erreicht werden.

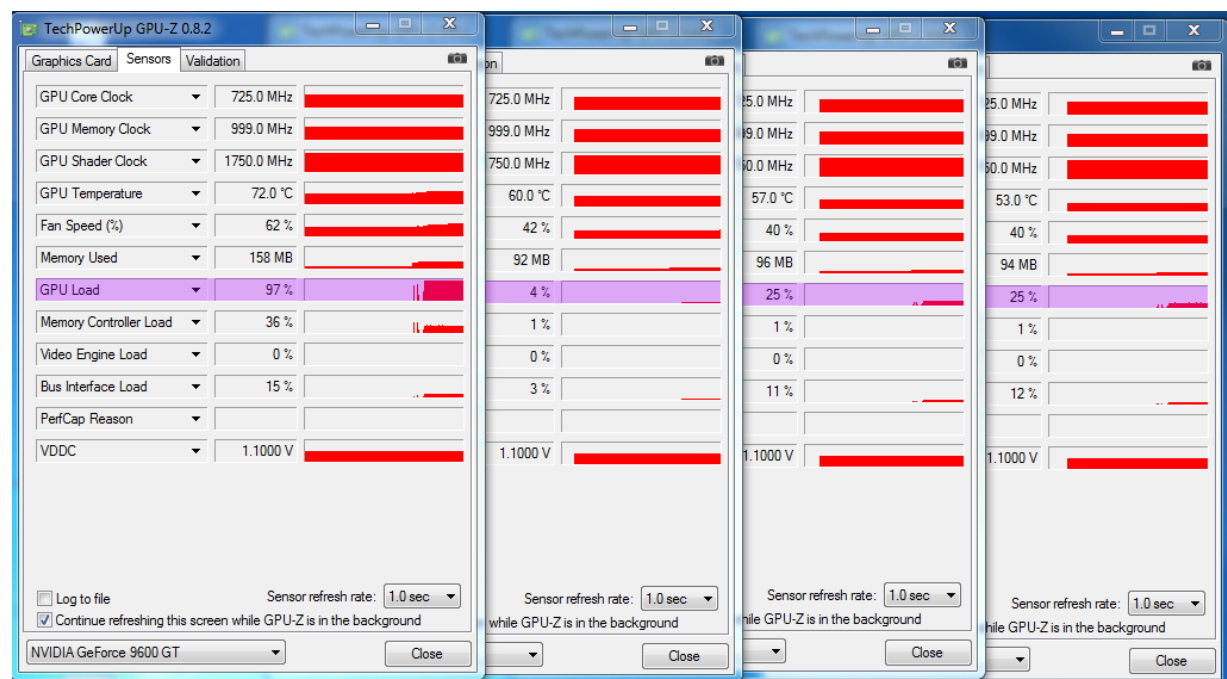


Abbildung 18: Auslastung GPUs GeForces

- **Mosaic**

Diese Variante scheint tatsächlich die vorhandene Hardware zu nutzen und verteilt die Rechenaufgaben auf sämtliche Karten. Auch hier wurde die Testapplikation auf verschiedenen Bildschirmen dargestellt und das Resultat war jedes Mal eine saubere Verteilung der Last auf alle vorhandenen GPUs.

¹² <http://www.techpowerup.com/downloads/SysInfo/GPU-Z/> GPU Monitoring Tool

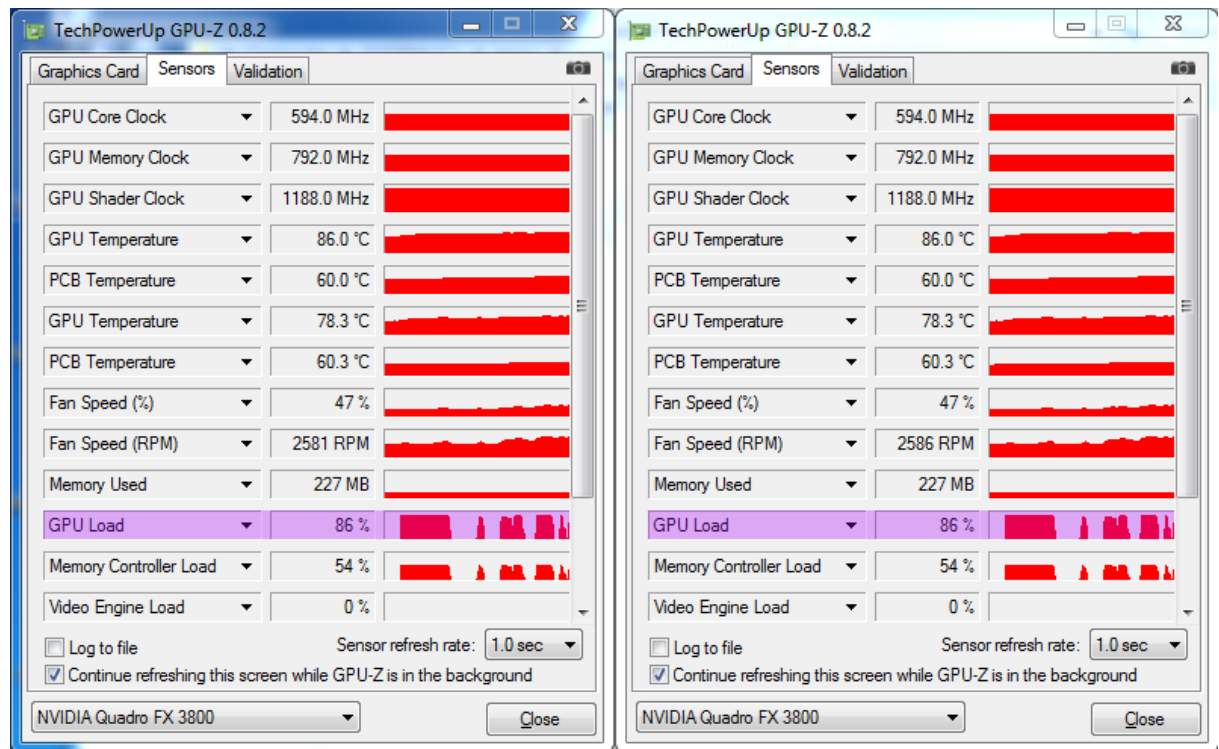


Abbildung 19: Auslastung GPUs Mosaic

12 Finaler Entscheid

Der Ansatz mit einer einzelnen Unity Instanz konnte sich klar hervorheben. Basierend auf der Tatsache, dass jede Unity-Anwendung eine Einzelanfertigung ist und eigene Voraussetzungen mit sich bringt, würden mit MiddleVR vor jeder neuen Inbetriebnahme einer Applikation Adaptionen notwendig sein. Ein allgemeingültiges System wäre nicht denkbar. Weil die Last somit nicht auf mehrere Clients verteilt werden kann, muss die vorhandene, zentrale Hardware bestmöglich ausgelastet werden.

Der definitive Entscheid lautet demnach, eine **eigene Lösung basierend auf Mosaic** umzusetzen.

Auch mit sorgfältiger Voranalyse könnten während der Umsetzung ungeahnte Probleme auftauchen. Deshalb wird als Fallbackplan, auch basierend auf Mosaic, die Variante MiddleVR weiterverfolgt. Sollten sich unüberwindbare Hindernisse bezüglich einer einzelnen Unity Instanz in den Weg stellen, wird auf das Clustering mittels MiddleVR zurückgegriffen.

13 Abbildungsverzeichnis

Abbildung 1: Ansicht der Leinwände in vier Himmelsrichtungen, Quelle: http://chromium.sourceforge.net/presentations/SantaFe-BrianPaul/siframes.html	6
Abbildung 2: Implementation eigener Rendering Styles, Quelle: http://chromium.sourceforge.net/doc/index.html	7
Abbildung 3: Equalizer Beispiel, Quelle: http://www.equalizergraphics.com/documents/Developer/eqPly.pdf	9
Abbildung 4: Display Wall, Quelle: http://www.equalizergraphics.com/useCases.html	10
Abbildung 5: Vierende VR Installation, Quelle: http://www.equalizergraphics.com/useCases.html	10
Abbildung 6: Client-Server Architektur, Quelle: Eigendarstellung	13
Abbildung 7: MiddleVR Konfigurator	16
Abbildung 8: Basiskonzept MiddleVR	17
Abbildung 9: Beispielskonfiguration CAVE	18
Abbildung 10: Vergleich Varianten	20
Abbildung 11: Evaluierung Varianten, Quelle: http://draw.io	22
Abbildung 12: Mosaic Einstellungen, Quelle: http://nvidia.custhelp.com/rnt/rnw/img/enduser/3568-7.png	25
Abbildung 13: Windows Monitor Settings	27
Abbildung 14: Basic Prototyp	28
Abbildung 15: Basic-Extended Prototyp	29
Abbildung 16: AngryBots, Quelle: http://www.androidmag.de/	29
Abbildung 17: Car Tutorial, Quelle: http://blogs.unity3d.com/2010/04/24/car-tutorial/	30
Abbildung 18: Auslastung GPUs GeForces	34
Abbildung 19: Auslastung GPUs Mosaic	35
Abbildung 20: Gantt-Projektplan	38

14 Tabellenverzeichnis

Tabelle 1: Wichtige Equalizer Klassen	10
Tabelle 2: Gegenüberstellung Pro und Kontra	19
Tabelle 3: Geschätzte Kosten MiddleVR Lizenzen Multi-GPU	23
Tabelle 4: Geschätzte Kosten Infrastruktur Multi-GPU, Quelle: http://digitec.ch (19.04.2015)	23
Tabelle 5: Geschätzte Kosten MiddleVR Lizenzen Cluster 8 Clients	24
Tabelle 6: Geschätzte Kosten MiddleVR Lizenzen Cluster 4 Clients	24
Tabelle 7: Nvidia Quadro K5000 Beispielaufbau	26
Tabelle 8: Nvidia Quadro K5000 MiddleVR Lizenz	26
Tabelle 9: Finale Gegenüberstellung	32
Tabelle 10: Meilensteine	38
Tabelle 11: Risikoanalyse	39

15 Anhang

15.1 Projektorganisation

Auf eine stark strukturierte Projektorganisation wird bewusst verzichtet. Die Teammitglieder sind gleichberechtigt. Es kann vorkommen, dass verschiedene Teilprojekte und Verantwortungsbereiche den Teammitgliedern zugewiesen werden. Dies bedeutet aber nicht die alleinige Durchführung dieser Tasks.

Es macht Sinn, die Entwicklung der Prototypen aufzuteilen und erst bei den Testläufen dieser dies zusammen durchzuführen.

15.1.1 Projektteam

Daniel Inversini daniel.inversini@students.bfh.ch
Julien Villiger julien.villiger@students.bfh.ch

15.1.2 Betreuer

Prof. Urs Künzler urs.kuenzler@bfh.ch

15.2 Projektplanung, Meilensteine

Der Gantt¹³-Projektplan ist unter <https://pm.ti.bfh.ch/projects/unity3d-in-cave/issues/gantt> erreichbar.

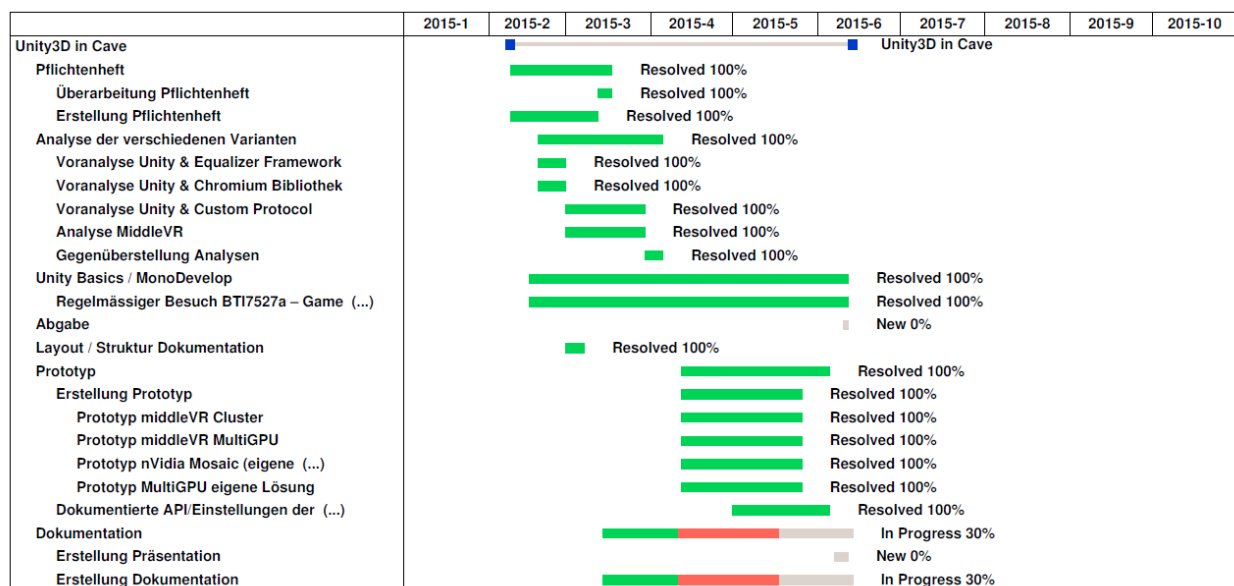


Abbildung 20: Gantt-Projektplan

Folgende Meilensteine wurden in der Voranalyse bereits festgelegt:

Bezeichnung	Fälligkeit
Voranalyse	29.03.2015
Prototyp	31.05.2015
Ergebnis dokumentiert	06.06.2015

Tabelle 10: Meilensteine

¹³ <http://de.wikipedia.org/wiki/Gantt-Diagramm> Gantt

Voranalyse

Analyse und Gegenüberstellung der verschiedenen Varianten, Entscheidung für Prototypen fällen.

Prototyp

Entwicklung mehrere lauffähiger Prototypen.

Ergebnis dokumentiert

Gewählter Lösungsansatz und Prototypen ersichtlich. Soll als Basis für die weiterführende Bachelorarbeit dienen.

15.3 Qualitätssicherung

Als Qualitätssicherung soll regelmässig der aktuelle Ist-Zustand des Projektes mit dem Projektplan verglichen werden. Weiter finden alle zwei Wochen Projektmeetings mit dem Betreuer statt, wo Fortschritte und Probleme diskutiert werden können.

15.4 Risikoanalyse

Sollten erkennbare Risiken auftreten, wird der Auftraggeber vom Projektteam informiert, und es werden bereits Lösungsansätze vorgeschlagen.

A: Auswirkungen

W: Wahrscheinlichkeit

Risiko	Beschreibung	Massnahmen	A	W
Mangelnde Projekterfahrung	Aufgrund der geringen Erfahrung im Planen von grösseren Projektarbeiten für die FH wird der Zeitplan falsch eingeschätzt	Durch die zweiwöchentlichen Sitzungen wird diesem Entgegen gewirkt.	Gross	Niedrig
Hardwareprobleme	Die ältere Infrastruktur des CAVES kann u.U. ein Testszenario nicht komplett wie gewünscht wiedergeben. (Bspw. mehrere Wände mit Stereoskopie)	Lauffähige Hardware modular aus- und einbauen, damit mit den vorhandenen Ressourcen gearbeitet werden kann	Mittel	Mittel
Softwareprobleme	Die vorgesehenen Softwares (MiddleVR, Mosaic) könnten u.U. Probleme bereiten, oder nicht alle gewünschten Features nicht unterstützen.	Frühes Testen mit Prototypen, Spezialfälle eruieren, Alternativen suchen	Mittel	Mittel

Tabelle 11: Risikoanalyse

16 Versionskontrolle

Version	Datum	Beschreibung	Autor
1.00	24.03.2015	Dokument erstellt	Daniel Inversini
1.01	25.03.2015	Dokument überarbeitet	Daniel Inversini
2.00	20.04.2015	Dokument erweitert	Daniel Inversini, Julien Villiger
3.00	17.05.2015	Dokument erweitert	Julien Villiger
3.10	18.05.2015	Dokument erweitert	Daniel Inversini
3.20	24.05.2015	Dokument erweitert, Tabellarische Übersicht Gegenüberstellung und Auswertung.	Julien Villiger
3.30	25.05.2015	Dokument reviewed, Smartspider neu	Daniel Inversini
3.40	04.06.2015	Leichte Umstrukturierung, Korrekturen	Julien Villiger