

# Unity 3D Server for CAVE Rendering

**Bachelorthesis**

Studiengang:	Informatik
Autor:	Julien Villiger, Daniel Inversini
Betreuer:	Prof. Urs Künzler
Auftraggeber:	cpvr Lab BFH
Experten:	Harald Studer
Datum:	19.01.2016

## Management Summary

Die Erstellung einer Applikation für den CAVE<sup>1</sup> der BFH ist ein relativ aufwändiger Prozess und setzt grosses Wissen über die Infrastruktur sowie fundierte C++ Kenntnisse voraus. Nur wenige Studenten realisieren, meist in Form einer Bachelor oder Master Thesis, eine Implementierung. Demzufolge ist das Zielpublikum klein und die Möglichkeiten des CAVEs werden kaum genutzt.

Im Rahmen der Thesis „Unity<sup>2</sup> 3D Server for CAVE Rendering“ wurde erfolgreich eine Integration von Unity in den CAVE vorgenommen. Durch die Einfachheit des Plugins, einer simplen Schritt-für-Schritt Anleitung und Demoapplikationen können auch unerfahrene Anwender von der Infrastruktur Gebrauch machen und die Stärke von Unity mit dem CAVE verbinden. Somit kann der schnell wachsenden Verbreitung von Unity Rechnung getragen werden und der CAVE der BFH ist über eine neue Plattform ansprechbar.

Der Hauptbestandteil der Umsetzung ist die virtuelle Abbildung der Komponenten. Mit Hilfe von Unity wird die Weiterverarbeitung und Interpretation vereinfacht und ist somit Basis für sämtliche Manipulationen der Applikation. Die Erweiterung der Projektion auf mehrere Leinwände erfolgt mittels Generierung virtueller Kameras, die sich einerseits der Logik der Applikation anpassen und andererseits Inputs vom Benutzer über die verschiedenen VR-Eingabegeräte des vorhandenen Trackingsystems von WorldViz<sup>3</sup> entgegennehmen. So kann beispielsweise eine Kopfbewegung eine Änderung der Ansicht im Spiel bewirken. Gleichzeitig wird mit Hilfe dieser erstellten Kameras, Beamern mit Polfilter und polgefilterten Brillen eine stereoskopische<sup>4</sup> Projektion ermöglicht.

Die gesamte Funktionalität lässt sich per Drag & Drop in ein neues oder bereits bestehendes Unity Projekt übernehmen. Einstellungen und der Funktionsumfang eines Spiels werden vom Plugin nicht überschrieben, lediglich erweitert. Um eine generische Lösung anbieten zu können, stehen etliche Einstellungsmöglichkeiten zur Verfügung. So können beispielsweise zusätzliche Kameras individuell platziert oder die Darstellung der GUI-Elemente auf eine CAVE-Leinwand fixiert werden. Die Viewfrustum<sup>5</sup>-Transformation, welche basierend auf der Position des Benutzers im CAVE berechnet wird, gewährleistet eine realistische Perspektive im virtuellen Raum und lässt den Benutzer in die künstliche Welt eintauchen.

Abgerundet wurde die Arbeit mit extra erstellten Demoapplikationen, welche die Möglichkeiten des CAVEs zusammen mit Unity, dem Plugin und dem Trackingsystem demonstrieren. 3D Welten lassen sich innert kurzer Zeit hautnah erleben, was auch für andere Abteilungen der BFH von grossem Nutzen sein kann.

<sup>1</sup> CAVE – [https://de.wikipedia.org/wiki/Cave\\_Automatic\\_Virtual\\_Environment](https://de.wikipedia.org/wiki/Cave_Automatic_Virtual_Environment)

<sup>2</sup> Unity – <http://unity3d.com/>

<sup>3</sup> WorldViz – <http://www.worldviz.com/>

<sup>4</sup> Stereoskopie – <https://de.wikipedia.org/wiki/Stereoskopie>

<sup>5</sup> Frustum – [https://en.wikipedia.org/wiki/Viewing\\_frustum](https://en.wikipedia.org/wiki/Viewing_frustum)

## Inhalt

	Management Summary	2
1	Einführung	5
1.1	Vorarbeiten Projekt 2	5
2	Infrastruktur	8
2.1	Trackingsystem WorldViz	9
2.2	Devices	10
2.3	Unity Server	11
2.4	Audio	12
2.5	Video Matrix Switch	12
3	Architektur der Komponenten	13
3.1	Unterteilung Module	15
3.2	Simplifiziertes Paketdiagramm	15
3.3	Sequenzdiagramm	16
4	Realisation	18
4.1	Stereoskopie	18
4.1.1	Kameraeinstellungen	18
4.1.2	GUI Kameras	19
4.1.3	Cursor Kameras	19
4.1.4	Sekundäre Kameras	19
4.1.5	VR Namespace Unity	20
4.1.6	Mosaic Settings	21
4.2	Immersion	23
4.2.1	Frustum allgemein	23
4.2.2	CAVE XXL Frustum	24
4.2.3	General Projection Matrix Frustum	25
4.2.4	Vergleich CAVE XXL und General Projection Matrix	26
4.3	Devices	28
4.3.1	Wand	28
4.3.2	Eyes	33
4.3.3	Gamepad	33
4.4	VRPN	34
4.4.1	Verwendung mit PPT Studio WorldViz	34
4.4.2	Datenverarbeitung im Unity	35
4.4.3	Datenveredlung im Unity	36
4.5	Unity Plugin	38
4.5.1	Konfiguration	38
4.5.2	Aufgabenverteilung	41
4.5.3	Deployment	48
4.5.4	Troubleshooting	53
4.5.5	Performance Verbesserungen	54
4.6	Warping	57
5	Demo Apps	59
5.1	Shooting Gallery	59
5.2	Model-Viewer	65
5.3	App Drittpartei	66
6	Testing	67
6.1	PPT-Studio und VRPN	67
6.2	Wrapping VRPN in C# und Unity	67
6.3	Unity Plugin Projekt als Debugger	68
6.4	Performance Tests Unity Server	69
7	Projektmanagement	73
7.1	Aufgabenstellung	73

7.2 Zieldefinitionen	73
7.2.1 Funktionale Anforderungen	74
7.3 Projektorganisation	75
7.3.1 Projektteam	75
7.3.2 Betreuer	75
7.3.3 Experte	75
7.4 Projektplanung, Meilensteine	75
7.5 Qualitätssicherung	76
7.6 Risikoanalyse	77
8 Fazit	78
9 Abbildungsverzeichnis	79
10 Tabellenverzeichnis	81
11 Quellcodeverzeichnis	81
12 Glossar	82
13 Literaturverzeichnis	86
14 Anhang	87

# 1 Einführung

## 1.1 Vorarbeiten Projekt 2

Im Rahmen der vorgängigen Projekt 2 Arbeit wurden verschiedene Methoden geprüft, wie eine Integration von Unity in den CAVE erfolgen kann. In einem ersten Schritt erfolgte eine Analyse der bereits verwendeten Frameworks (Equalizer<sup>6</sup>, Chromium<sup>7</sup>) und der Software MiddleVR<sup>8</sup>, die den Einsatz von Unity in einem CAVE deutlich vereinfachen sollte, sowie einem eigenen Plugin. Die Frameworks Equalizer und Chromium konnten sich Aufgrund folgender Punkte nicht durchsetzen:

- Projekt wurde eingestellt (Chromium)
- Kein Unity Support (Chromium und Equalizer)
- Struktur durch Framework gegeben, nicht für Unity adaptierbar (Wrapperklassen von Equalizer)
- Keyfeatures wie KI, Tracking und Physik nicht ansprechbar (jeweils nur OpenGL, Grafik)

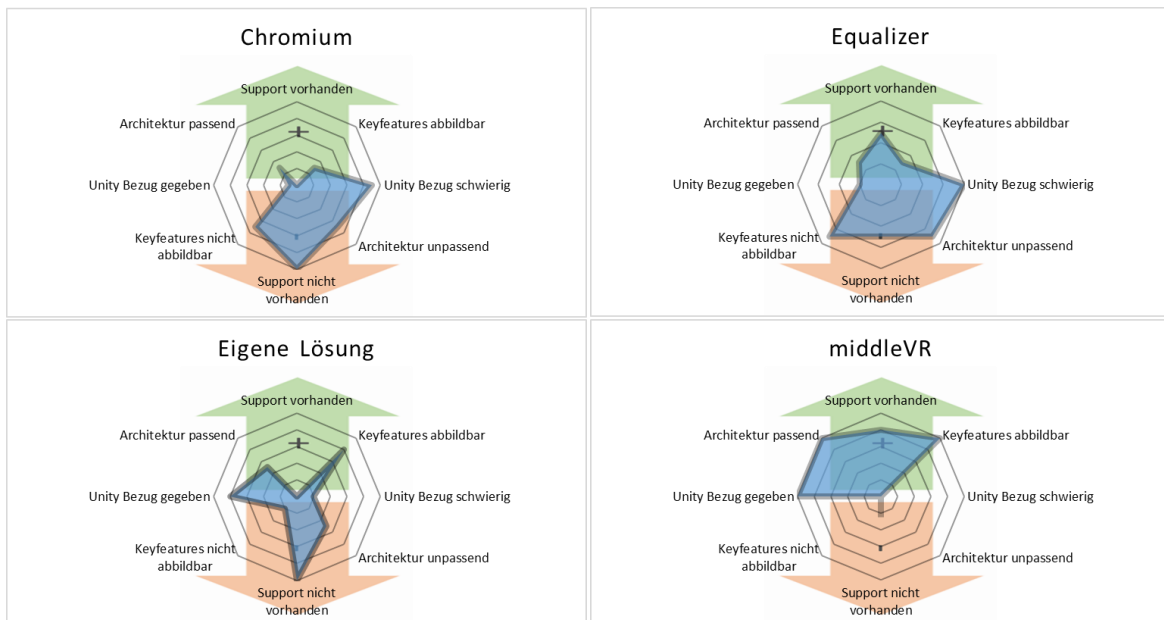


Abbildung 1: Vergleich Varianten Projekt 2

Weiter präsentierte sich die Software MiddleVR als günstiger Kandidat, hatte jedoch nach den vertieften Abklärungen signifikante Schwachstellen:

- Es wird nur das Transform<sup>9</sup> des Unity Objekts über die Cluster (Position, Rotation und Massstab) synchronisiert
- Folglich fehlen Materials<sup>10</sup>, Farben und Partikel
- Die Flexibilität wird durch die proprietäre Software eingeschränkt
- Hohe Lizenzkosten durch Anzahl CPUs und GPUs

<sup>6</sup> Equalizer – <http://www.equalizergraphics.com/>

<sup>7</sup> Chromium – <http://chromium.sourceforge.net/>

<sup>8</sup> MiddleVR – <http://www.middlevr.com/>

<sup>9</sup> Transform Unity – <http://docs.unity3d.com/ScriptReference/Transform.html>

<sup>10</sup> Material Unity – <http://docs.unity3d.com/ScriptReference/Material.html>

Die dadurch entstehende Abhängigkeit der Software, die Anforderungen an die Unity Applikationen (etliche Adaptionen am Code waren jeweils notwendig) und die nicht gebrauchte Fülle an Features waren der Grund, wieso eine eigene Umsetzung eines Unity Plugins erfolgen sollte.

Bezüglich der Infrastruktur wurden folgende Methoden analysiert und bewertet:

1. Mehrere Hosts / Mehrere GPUs / Mehrere Unity Instanzen  
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 Netzwerkanwendung laufen, d.h. es kommt eine ähnliche Synchronisierung zum Zuge, welche auch bei Multiplayerspielen über das Internet verwendet wird.
2. Ein Host / Mehrere GPUs / Mehrere Unity Instanzen  
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 würde auf dem gleichen Client stattfinden. 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.
3. Ein Host / Mehrere GPUs / Eine Unity Instanz mit Mosaic<sup>11</sup> Treiber  
Mosaic ist eine Technologie von Nvidia, um mehrere Quadro<sup>12</sup>-Graphikkarten über den Treiber zu verlinken und auf einem Desktop darzustellen. Windows wird ein einzelner Output vorgegaukelt, auch wenn physisch mehrere GPUs mit multiplen Ausgängen angeschlossen sind.
4. Ein Host / Mehrere GPUs / Eine Unity Instanz  
Ähnlich der Mosaic Variante ist ein Setting mit gängigeren, Nicht-Quadro-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.

Basierend auf Prototypen der verschiedenen Systeme, Prototypen und theoretischer Abklärungen konnte sich Variante 4 (**Ein Host / Mehrere GPUs / Eine Unity Instanz mit Mosaic Treiber**) klar hervorheben. Die dadurch erreichte hohe Flexibilität, die obsolete Synchronisierung und die gute Performance durch Verteilung der Last auf verschiedene GPUs waren ausschlaggebend.

<sup>11</sup> Nvidia Mosaic – <http://www.nvidia.com/object/nvidia-mosaic-technology.html>

<sup>12</sup> Nvidia Quadro – [https://de.wikipedia.org/wiki/Nvidia\\_Quadro](https://de.wikipedia.org/wiki/Nvidia_Quadro)

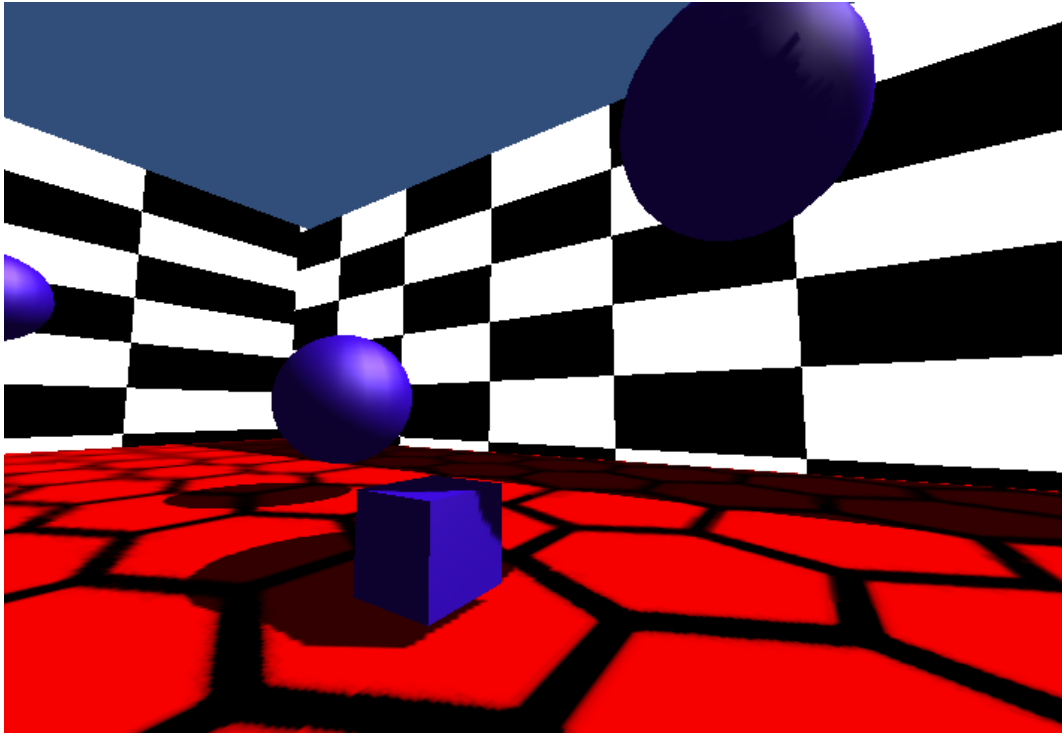


Abbildung 2: Prototyp 1 Projekt 2



Abbildung 3: Prototyp 2 Projekt 2

## 2 Infrastruktur

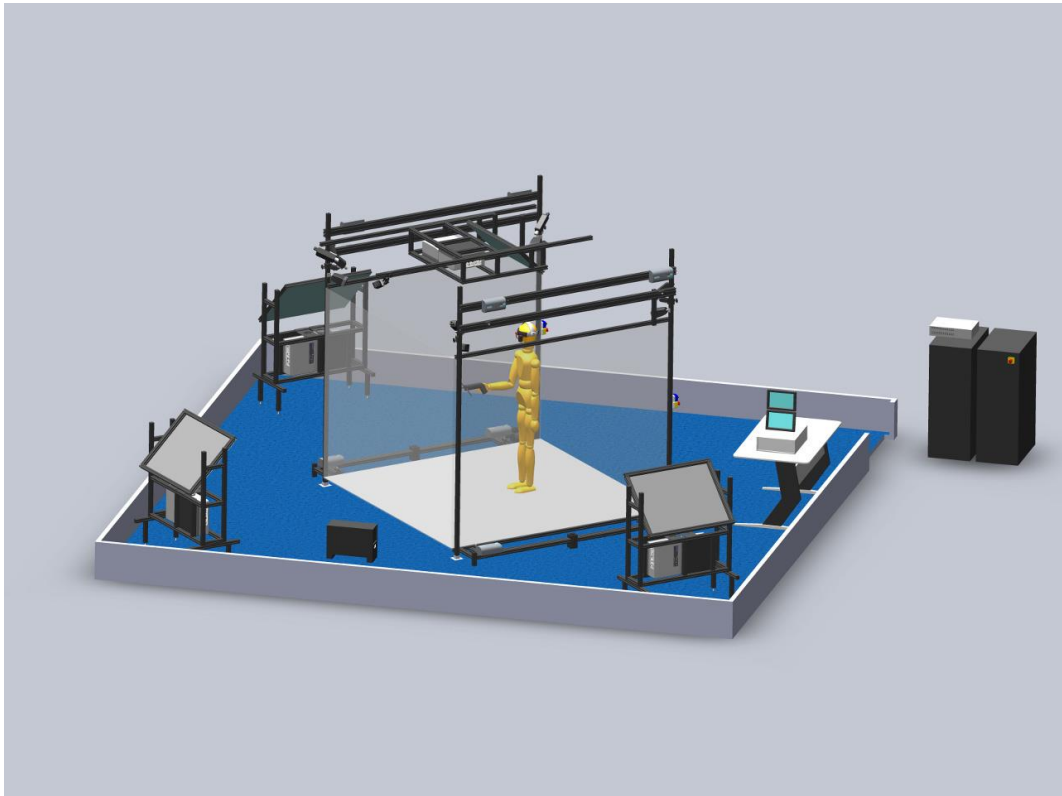


Abbildung 4: CAVE BFH

Der virtual reality haptic<sup>13</sup> CAVE der BFH bietet Entwicklern und Forschern ein mächtiges Instrument, um hochrealistische immersive<sup>14</sup> Applikationen zu bauen. Der CAVE ist eine kubische Konfiguration mit vier Leinwänden (Links, Vorne, Rechts, Boden) die von je 2 Projektoren bestrahlt werden. Um eine realistische 3D Stereoprojektion zu erschaffen, sind bei den Projektoren Polfilter angebracht und die Benutzer tragen entsprechend eine Brille mit Polfilter.

<sup>13</sup> Haptic – <https://en.wikipedia.org/wiki/Haptics>

<sup>14</sup> Immersion – [https://de.wikipedia.org/wiki/Immersion\\_\(virtuelle\\_Realit%C3%A4t\)](https://de.wikipedia.org/wiki/Immersion_(virtuelle_Realit%C3%A4t))



Alle Komponenten des CAVE und dessen Abhängigkeiten werden in der folgenden Grafik dargestellt:

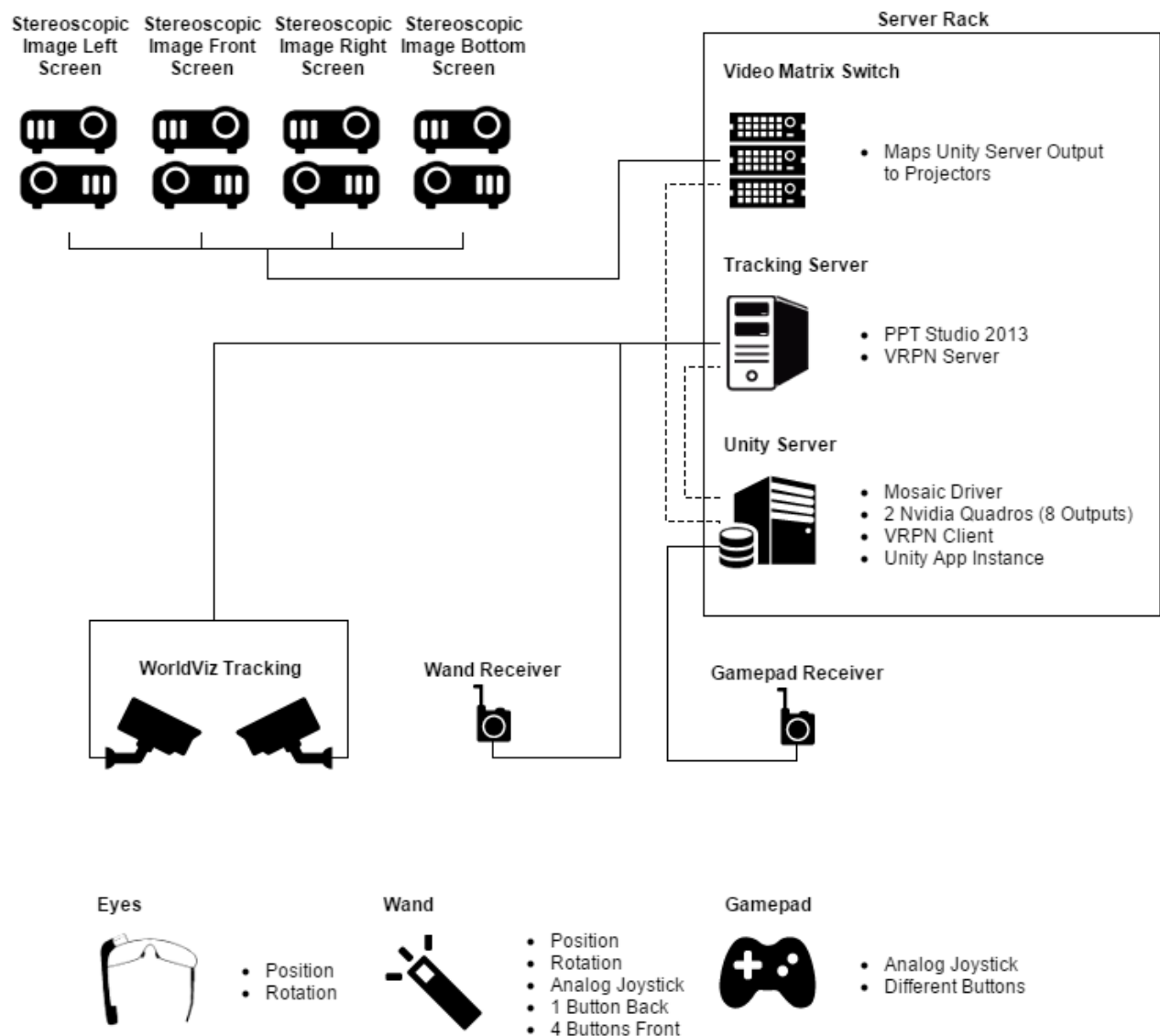


Abbildung 5: Infrastruktur CAVE

## 2.1 Trackingsystem WorldViz

Das Trackingsystem von WorldViz wurde integriert, um Positionen und Rotationen von verschiedenen Devices im CAVE zu erfassen. Die von den 10 WorldViz Kameras übermittelten Daten werden vom PPT Studio 2013<sup>15</sup> zentral auf einem Server interpretiert, d.h. es werden Punkte im Raum und dessen Rotation der Devices berechnet, und können bei Bedarf über das VRPN<sup>16</sup>-Protokoll abgefragt werden.

<sup>15</sup> PPT Studio 2013 – <http://www.worldviz.com/virtual-reality-motion-tracking/>

<sup>16</sup> VRPN – <https://github.com/vrpn/vrpn/wiki>

## 2.2 Devices

- **Eyes**

Die Eyes von WorldViz sind Brillen mit Polfilter und zwei montierten Infrarot-Trackern. Somit lassen sich die Position des Kopfes und dessen Rotation auf zwei Achsen (Yaw und Roll<sup>17</sup>) bestimmen. Die dritte Achse (Pitch) kann mit lediglich zwei Trackern nicht ermittelt werden. Dazu wären mindestens drei LEDs oder ein Gyrometer<sup>18</sup> notwendig.



Abbildung 6: PPT Eyes, Quelle: [www.worldviz.com](http://www.worldviz.com)

- **Wand**

Der Wand von WorldViz ist das primäre Eingabegerät. Nebst zwei Infrarot-Trackern, welche für die Positions- und Rotationsbestimmung verwendet werden, ist ein Gyrometer integriert, um noch präziser Drehungen feststellen zu können. Dadurch wird auch die fehlende Rotationsachse (fehlende bei den Eyes) kompensiert und es können Yaw, Roll und Pitch ermittelt werden.

Als Inputs dienen ein analoger Joystick, vier Buttons auf der Vorderseite sowie ein Button auf der Rückseite.

<sup>17</sup> Yaw, Pitch, Roll – [https://en.wikipedia.org/wiki/Six\\_degrees\\_of\\_freedom](https://en.wikipedia.org/wiki/Six_degrees_of_freedom)

<sup>18</sup> Gyrometer – <https://de.wikipedia.org/wiki/Gyrometer>

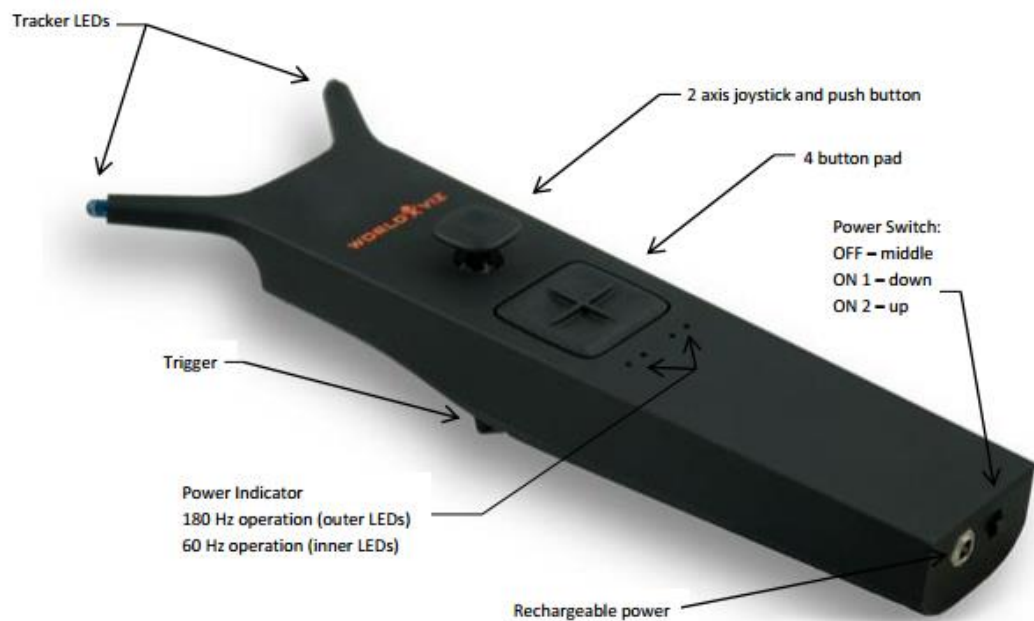


Abbildung 7: PPT Wand, Quelle: [www.worldviz.com](http://www.worldviz.com)

- **Gamepad**

Ein weiteres Inputgerät ist ein handelsübliches Gamepad. Frei von jeglichem Tracking wird einzig die Unity Applikation gesteuert.



Abbildung 8: Gamepad, Quelle: [www.androidrundown.com](http://www.androidrundown.com)

## 2.3 Unity Server

Der leistungsstarke Unity Server ist der Knotenpunkt des gesamten Systems. Auf diesem Rechner läuft die Unity Applikation mit dem konfigurierten Unity Plugin, welches die Trackingdaten vom Trackingserver abgreift und das korrekte Rendering in der Unity Applikation für die Projektoren Aufteilung sicherstellt. Mittels Mosaic, einem speziellen Treiber von Nvidia der die Aufteilung auf mehrere Grafikkartenausgänge übernimmt, werden alle Projektoren korrekt für die stereoskopische Projektion angesprochen.

## **2.4 Audio**

Um die Immersion weiter zu steigern, ist ein 3D Soundsystem mit vier Lautsprechern in Betrieb.

## **2.5 Video Matrix Switch**

Weil parallel weitere Clients in Betrieb sind und Bilddaten für den CAVE liefern können, wird ein Video Matrix Switch eingesetzt, um die verschiedenen Inputquellen auf die 8 bestehenden Projektoren abbilden zu können.

### 3 Architektur der Komponenten

Damit das Unity Plugin einwandfrei läuft, ist eine enge Zusammenarbeit zwischen realen und virtuellen Komponenten vonnöten. Beispielsweise haben die Inputdevices Wand und Eyes ein virtuelles Pendant, um dessen Eigenschaften besser verwerten und weiterverarbeiten zu können. Zusätzlich ermöglicht dieses Konzept ein sauberes Debugging und eine ansprechende Visualisierung.

Die folgende Grafik zeigt die Verschmelzung der realen und virtuellen Welt mit je einem halben CAVE und deren spezifischen Komponenten auf.

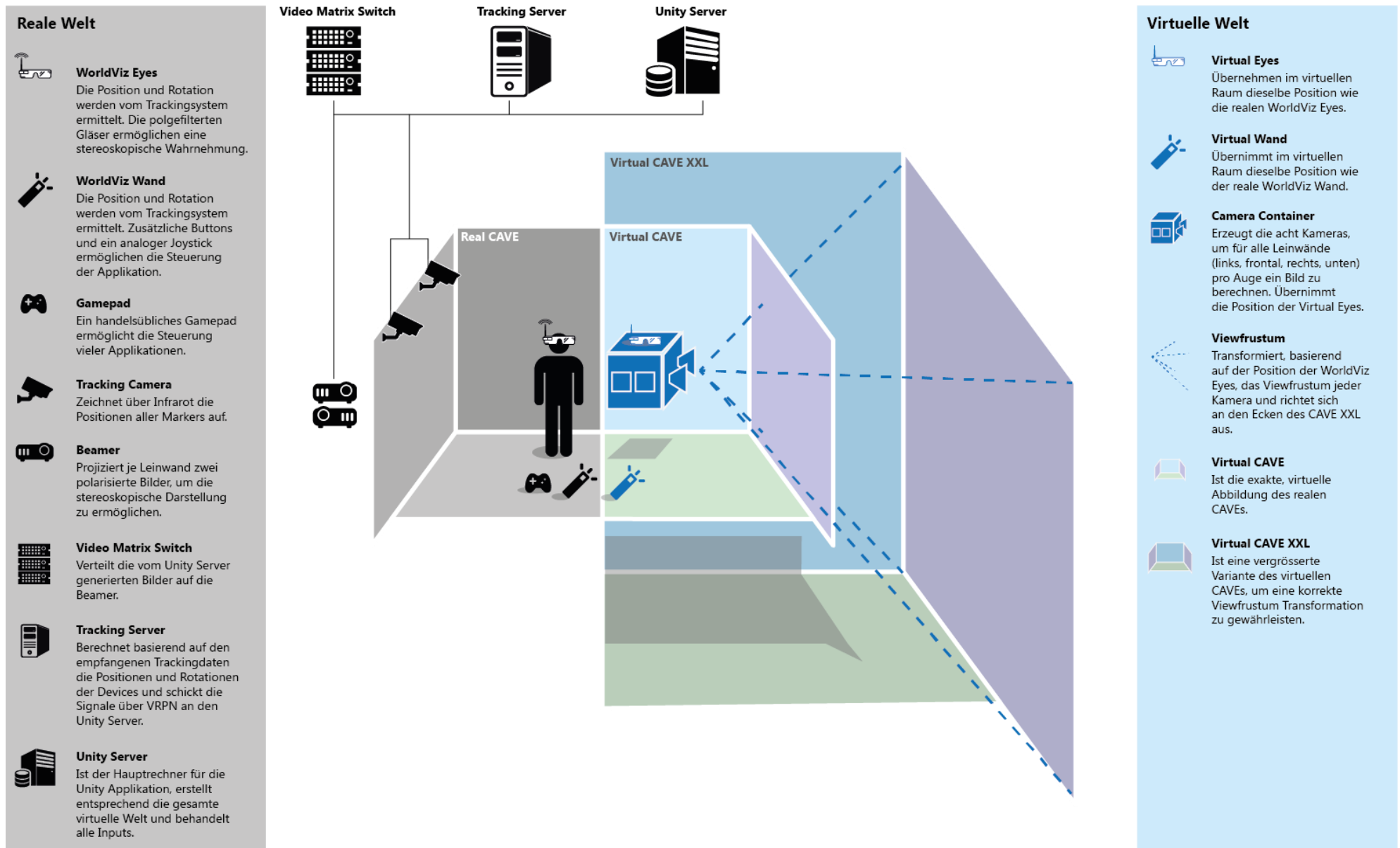


Abbildung 9: Übersicht der Komponenten

### 3.1 Unterteilung Module

Das modulare Konzept der objektorientierten Programmierung wurde streng eingehalten. Jede einzelne Komponente verfügt über einen klar abgegrenzten Aufgabenbereich und hat möglichst wenige Verknüpfungen zu anderen Modulen. Diese für das menschliche Denken intuitive Aufteilung erleichtert die Programmierung enorm und vereinfacht den Einstieg und die Einarbeitung in den Code für Aussenstehende.

### 3.2 Simplifiziertes Paketdiagramm

Folgendes Diagramm zeigt das Zusammenspiel zwischen den Komponenten Unity, Unity-MonoBehaviour, dem Plugin und externen Libraries. Aus Simplifizierungsgründen wurde bei Unity nur die Mainkamera angegeben. Dort könnte auch ein Fahrzeug, «First Person Controller» oder ähnliches sein. Komponenten welche von MonoBehaviour abhängig sind, verwenden die Update, FixedUpdate und weitere Routinen.

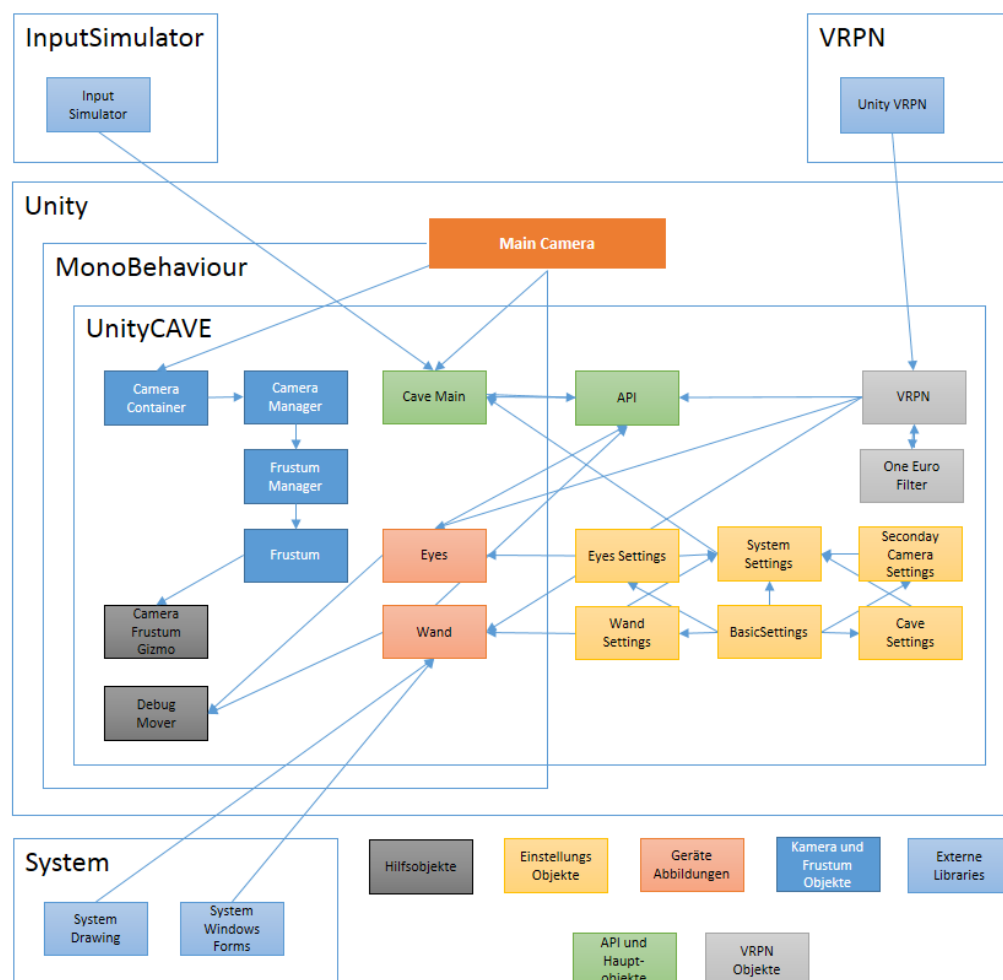


Abbildung 10: Paketdiagramm simplifiziert

### 3.3 Sequenzdiagramm

Folgendes Diagramm zeigt ein Update, also eine Abfolge für die Berechnung eines Frames, welches Unity durchführt. Die physikalischen Komponenten Wand und Eyes werden nicht dargestellt, jedoch das virtuelle Pendant in Unity. Der Benutzer macht im CAVE eine Bewegung, die vom Trackingsystem wahrgenommen und mittels Unity Plugin weiterverarbeitet wird um schlussendlich ein Bild auf den Projektoren anzuzeigen.



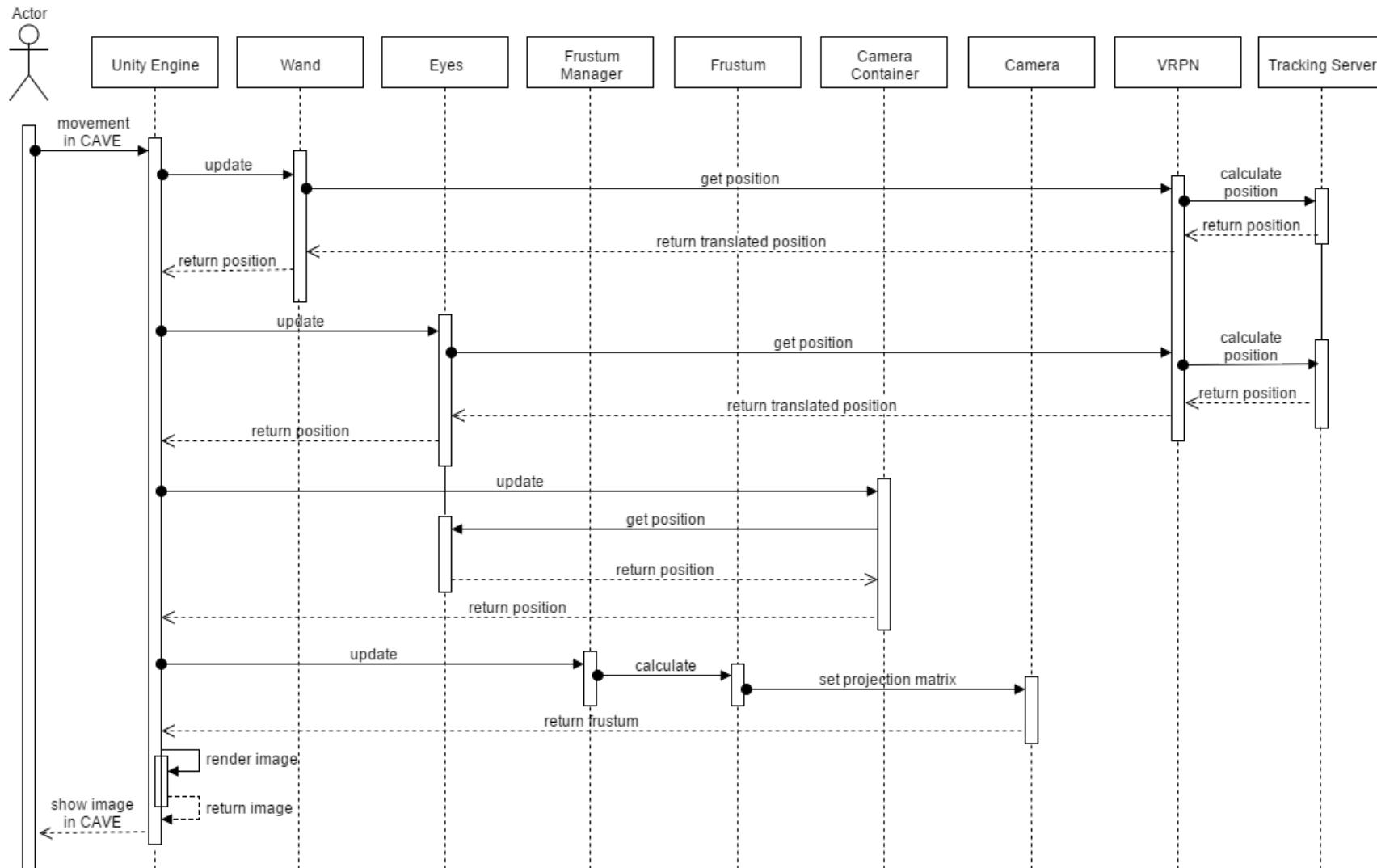


Abbildung 11: Sequenzdiagramm

## 4 Realisation

Basierend auf der gegebenen Infrastruktur, der Aufgabenstellung und der konzeptuellen Erarbeitung erfolgt die Realisation. Auch hier wurden die zentralen Aspekte wie eine gute Immersion und die Einfachheit der Bedienung in den Mittelpunkt gerückt.

Das folgende Kapitel zeigt die Behandlung der vier Hauptstandbeine Stereoskopie, Immersion, Devices und VRPN. Zum Schluss wird das Plugin und dessen Einstellungsmöglichkeiten erläutert. Als zusätzliche Anforderung wurde eine Analyse vorgenommen, wie der von den Beamern erzeugten Verzerrung der Bilder entgegengewirkt werden kann.

### 4.1 Stereoskopie

Damit der Benutzer die Applikation in der dritten Dimension erleben und eine Polfilterung der Bilder erfolgen kann, muss das Plugin etliche Kameras zur Verfügung stellen.

#### 4.1.1 Kameraeinstellungen

Für jede Seitenwand des CAVEs werden mehrere Kameras instanziiert und dem CameraContainer hinzugefügt:

- 3D Render Kamera für das linke Auge
- 3D Render Kamera für das rechte Auge
- GUI Render Kamera, aktiv falls sich GUI Elemente auf dieser Seite befinden
- Cursor Kamera, aktiv falls sich der Cursor auf dieser Seite befindet

Die 3D Render Kameras werden wie folgt instanziiert:

- An der Position der Hauptkamera (übernommen von der Applikation)
- Rotiert um jeweils 90, 0, -90 Grad um Y für die Seitenwände
- Rotiert um 90 Grad um X für den Boden
- Verschoben um die halbe Augendistanz für den passiven Stereoeffekt

Frustum Mode	CAVEXXL
GUI Location	Front
Eye Distance	0.07

Abbildung 12: Einstellungen Augendistanz

```

Left = new CameraInfo
{
    Cam = _cameraLeftLeft,
    CamGUI = API.Instance.Cave.CaveSettings.GUILocation == BasicSettings.Sides.Left ?
    Instantiate(_cameraLeftLeft) : null,
    CamCursor = Instantiate(_cameraLeftLeft),
    Offset = new Vector3(0f, 0f, -(API.Instance.Cave.CaveSettings.EyeDistance / 2))
},

Right = new CameraInfo
{
    Cam = _cameraLeftRight,
    CamGUI = API.Instance.Cave.CaveSettings.GUILocation == BasicSettings.Sides.Left ?
    Instantiate(_cameraLeftRight) : null,
    CamCursor = Instantiate(_cameraLeftRight),
    Offset = new Vector3(0f, 0f, +(API.Instance.Cave.CaveSettings.EyeDistance / 2))
}

```

Quellcode 1: Kamerainstanziierung

#### 4.1.2 GUI Kameras

Der Benutzer kann die Position der GUI-Elemente auf eine CAVE-Seite festlegen. Beim initialisieren der Applikation werden sämtliche 2D-Elemente gesucht und so im virtuellen Raum platziert, dass die Darstellung direkt auf der CAVE-Leinwand erfolgt und kein Tiefeneffekt entsteht. Gleichzeitig werden spezielle GUI-Kameras für das linke sowie das rechte Auge instanziiert, die einzig dazu da sind, GUI-Elemente zu rendern. Diese zusätzlichen Kameras sind notwendig, damit keine 3D-Objekte die Sicht auf die UI-Elemente nehmen.

#### 4.1.3 Cursor Kameras

Ähnlich wie für die GUI-Elemente werden auch für den Cursor spezielle Kameras erstellt, jedoch insgesamt 8 Stück (pro Seite je eine Kamera pro Auge). Je nachdem wohin der Wand zielt, also wo der Cursor dargestellt werden soll, aktivieren sich diese Cursor Kameras und stellen einen Windowscursor dar. Der Systemcursor kann nicht verwendet werden, weil der nur einmal vorhanden ist und somit nur für ein Auge angezeigt werden könnte. Deshalb wird die Position bestimmt und mit Hilfe von Unity zwei Kopien des Cursors dargestellt. Gleichzeitig wird der richtige Cursor ausgeblendet, damit kein Konflikt entsteht.

Falls ein Custom Cursor gewünscht ist, kann dieser dem Plugin angegeben werden und das entsprechende Sprite wird angezeigt.

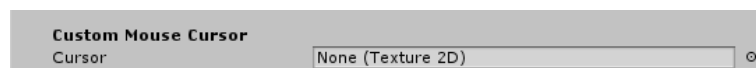


Abbildung 13: Custom Cursor

#### 4.1.4 Sekundäre Kameras

Je nach 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 Aussenkamera oder auch nur Rückspiegel bei einem Fahrzeug sein.

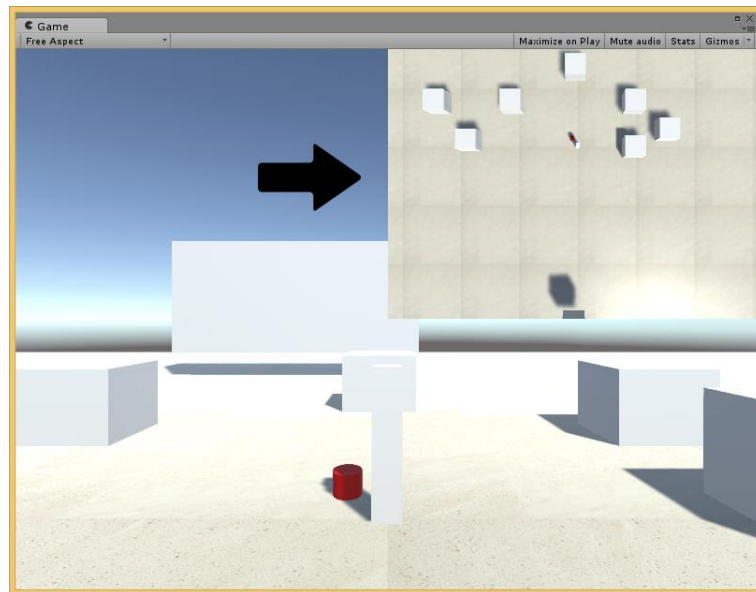


Abbildung 14: Minimap

Das Plugin löst diese Anforderung wie folgt:

- Der Benutzer kann entscheiden, auf welche Seite er die sekundäre Kamera zuordnen will.
- Es müssen lediglich all die gewünschten Kameras dem Plugin übergeben werden.

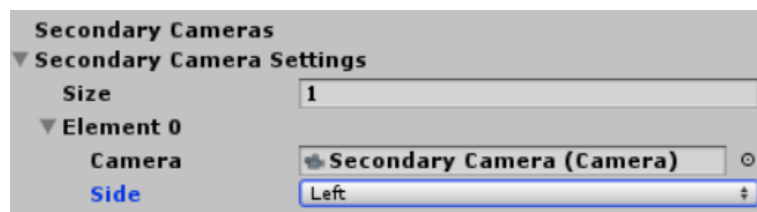


Abbildung 15: Sekundäre Kameras Einstellungen

#### 4.1.5 VR Namespace Unity

Unity verfügt seit Version 5.1 über einen VR Namespace<sup>19</sup>. Dieser wird verwendet, um externe Plugins/ SDKs, wie das der Oculus Rift, abzugrenzen. Mit diesem Namespace möchte Unity folgende Ziele erreichen:

- Vermeiden von Konflikten der Plugins untereinander
- Mehrere VR Geräte brauchen nicht mehr mehrere Plugins (Reduzierung Aufwand)
- Neue SDKs der Hersteller können mit älteren Versionen der Spiele nicht kompatibel sein

<sup>19</sup> VR Namespace Unity – <http://docs.unity3d.com/Manual/VROverview.html>

Das Basis-API unterstützt aktuell nur folgende Features:

- **Automatisches Stereodisplay**

Es ist nicht mehr nötig, wie bisher zwei Kameras zu instanzieren, dies wird von Unity übernommen. Das funktioniert jedoch nur für alle Kameras, die nicht auf eine Textur gerendert werden.

- **Head-Tracking als Input**

Dieser Input wird analog dem umgesetzten Unity Plugin behandelt.

Da dieses Feature erst mit Version 5.1 (Release Juni 2015) verfügbar war und es im Moment nur über Basisfähigkeiten verfügt, wird dies nicht verwendet.

(Unity, 2015)

#### 4.1.6 Mosaic Settings

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

(Nvidia, 2015)

Die Mosaic Einstellungen des Unity Servers sind wie folgt:

- 1280 Pixel auf 1024 Pixel Auflösung pro Ausgang
- Eine 2 auf 4 Verteilung der acht Bildschirme
- Gesamtauflösung von 5120 Pixel auf 2048 Pixel

Karte 1 Ausgang 1 Position 0;0	Karte 1 Ausgang 2 Position 0;1	Karte 2 Ausgang 1 Position 0;2	Karte 2 Ausgang 2 Position 0;3
Karte 1 Ausgang 3 Position 1;0	Karte 1 Ausgang 4 Position 1;1	Karte 2 Ausgang 3 Position 1;2	Karte 2 Ausgang 4 Position 1;3

Tabelle 1: Mosaic Setting schematisch

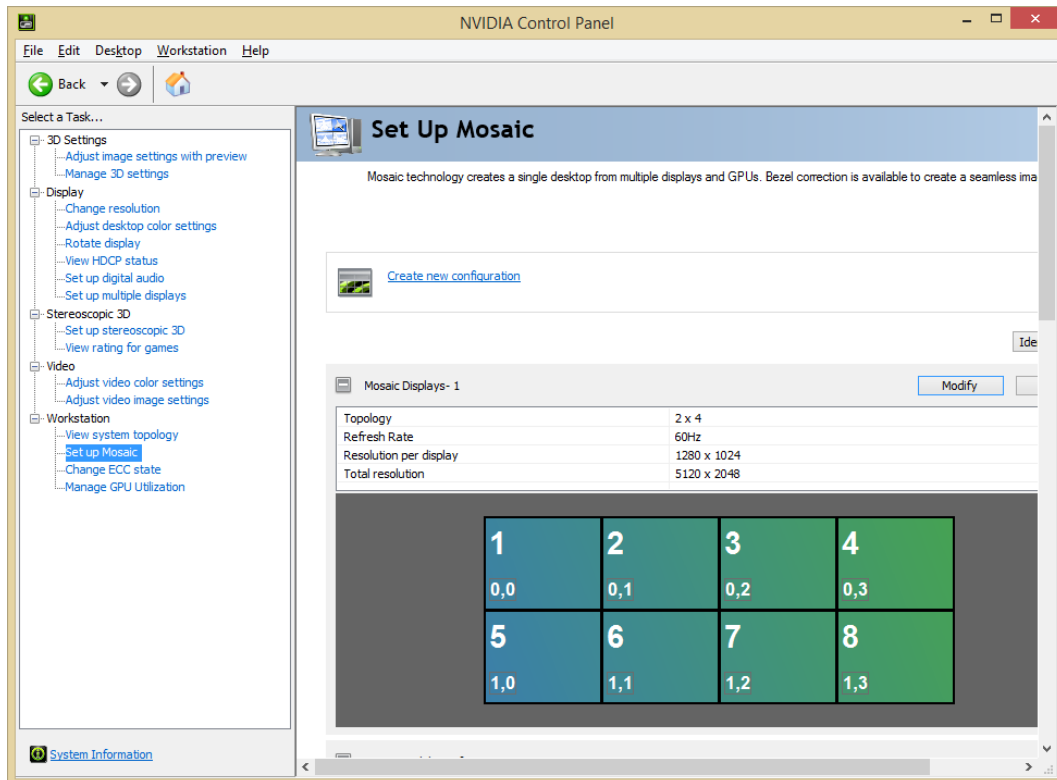


Abbildung 16: Mosaic Setting Unity Server

## 4.2 Immersion

Das Ziel ist, die Wahrnehmung der eigenen Person weitgehend zu verhindern und den Benutzer in die virtuelle Welt eintauchen zu lassen. Dazu gehört eine realistische Perspektive auf die künstliche Welt zu gewährleisten.

### 4.2.1 Frustum allgemein

Für eine optimale Projektion muss das bereits vorhandene Frustum der Applikation angepasst werden. Das Frustum wird im CAVE direkt bestimmt durch die geometrischen Eigenschaften der Wände des CAVES (initial) und wird durch die Position des Benutzers im CAVE aktualisiert (runtime). Ganz allgemein bildet das Frustum das 3D Bild der Computergraphik auf einen zweidimensionalen Bildschirm ab.

Es gibt mehrere Möglichkeiten ein Frustum aufzubauen, die folgenden zwei Methoden wurden umgesetzt:

- Frustum mit einer Projektionsfläche XXL (Seitenwand des CAVE virtuell ~30m entfernt, Dimensionen beibehalten)
- Frustum mit der genauen Projektionsfläche der CAVE-Wand und einer Off-Axis Projektion

Diese unterschiedlichen Möglichkeiten haben sich aus dem Adaptieren des bereits vorhandenen Frustums, den eigenen Überlegungen und der Adaptierung der generalisierten Projektionsmatrix des Ur-CAVEs ergeben.

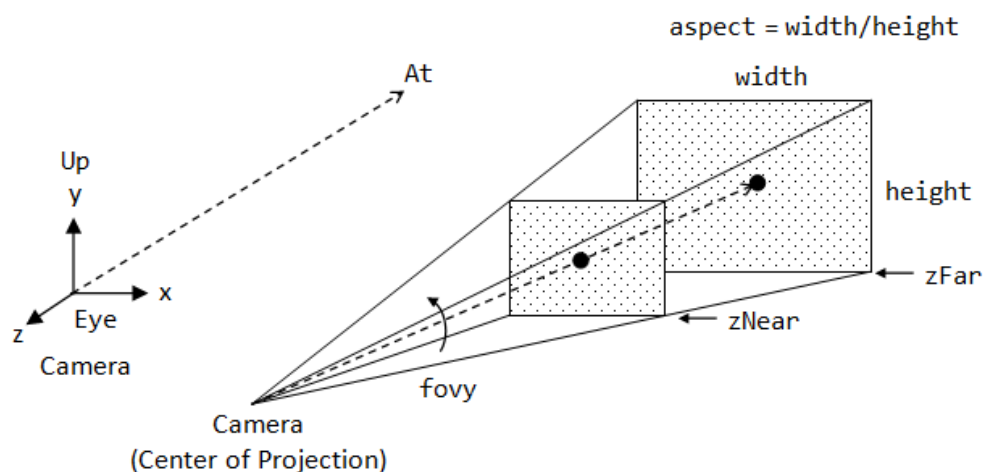


Abbildung 17: Frustum, Quelle: [www.stackoverflow.com](http://www.stackoverflow.com)

Ein Frustum wird in Unity wie folgt dargestellt. Sichtbar hier ist, dass die Seitenwand des CAVE XXL die Plane bildet, woran das Frustum aufgespannt wird.

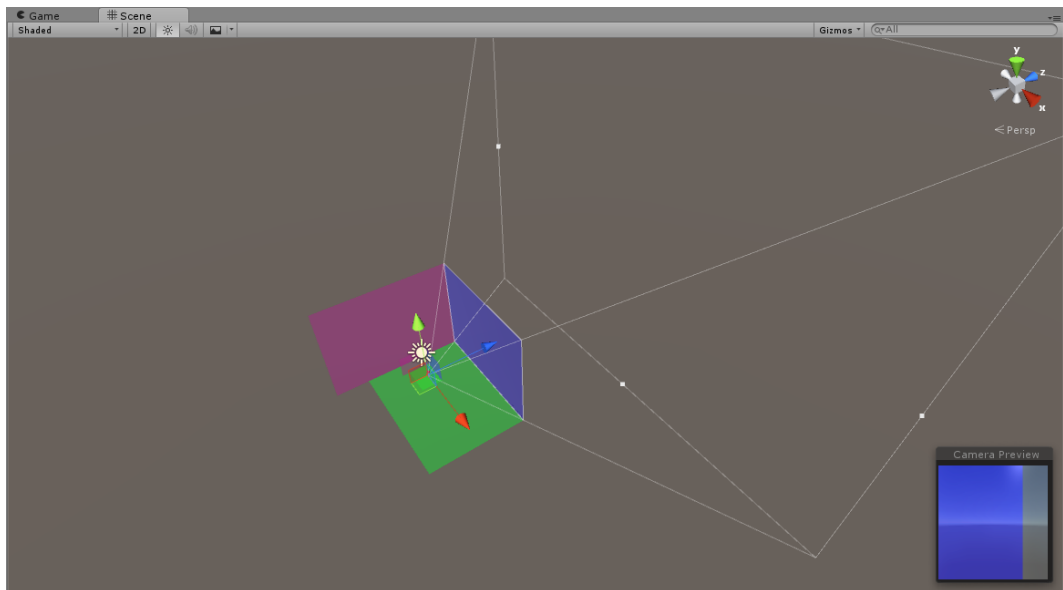


Abbildung 18: Unity Plugin Frustum 1

Mit allen 8 Kameras ergibt sich folgendes Bild:

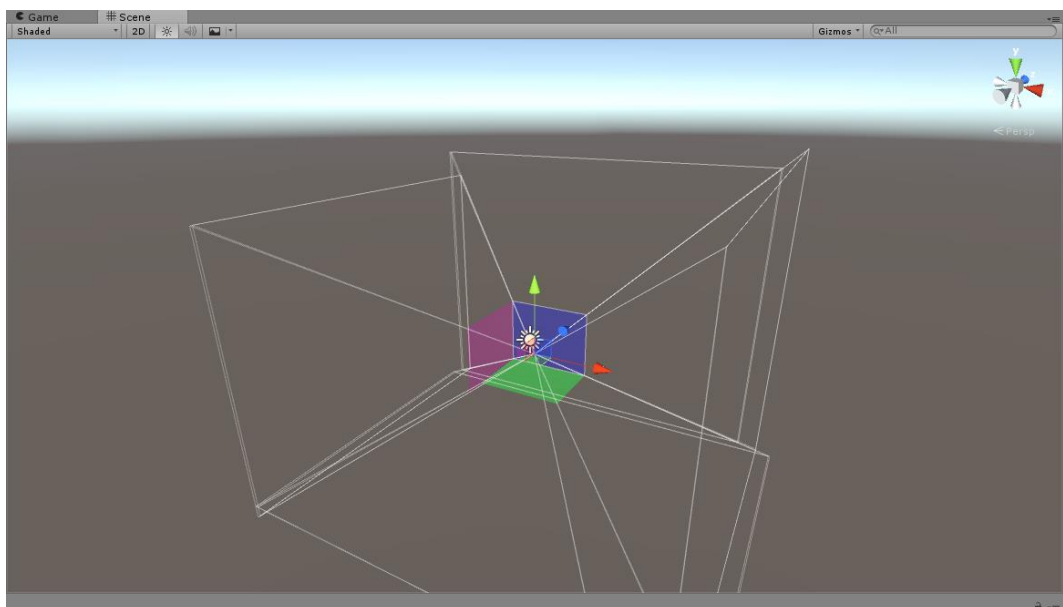


Abbildung 19: Unity Plugin Frustum 2

#### 4.2.2 CAVE XXL Frustum

Falls das Frustum auf die realen CAVE Seitenwand projiziert wird, vergrößert sich das Field of View<sup>20</sup> (FoV) wenn man sich der Leinwand nähert. Bei einem FoV von beispielsweise 120 Grad werden die Objekte verzogen, obwohl die Projektionsfläche dieselbe bleibt.

<sup>20</sup> Field of View – [https://en.wikipedia.org/wiki/Field\\_of\\_view](https://en.wikipedia.org/wiki/Field_of_view)



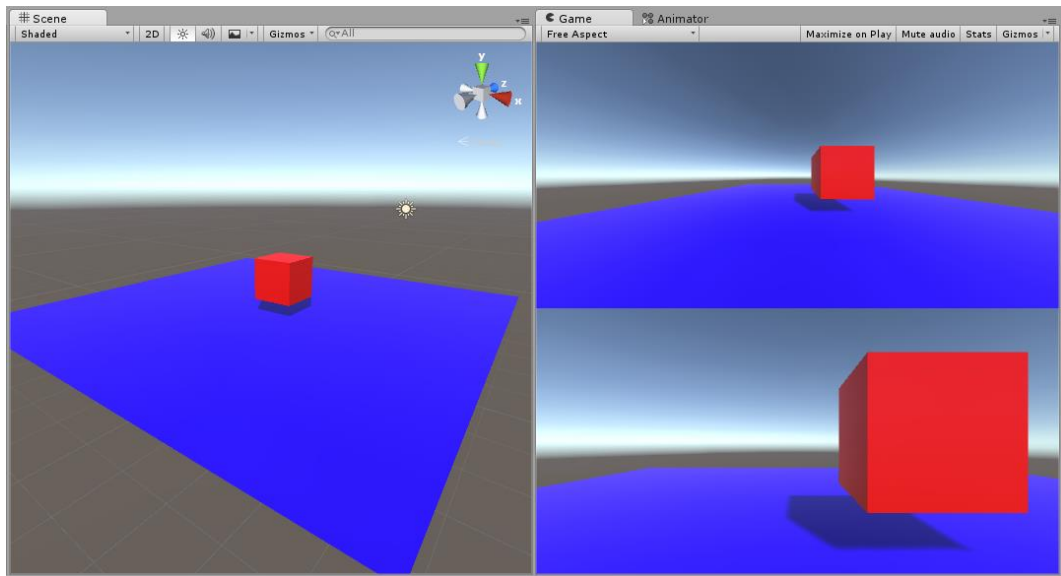


Abbildung 20: Vergleich FoV 120 & 60 Grad

Um diesen Effekt abzuschwächen, wird in diesem Betriebsmodus nicht die Seitenwand des CAVEs verwendet, um das Frustum aufzuspannen, sondern die Seitenwand des CAVE XXL.

- Es kann die «Generalized Perspective Projection» von Robert Kooima verwendet werden, welche 2008 publiziert wurde. (Kooima, 2008)
- Da sich der Betrachter der Seitenwand des CAVE XXL nie gross nähert, tritt der oben genannte Effekt nicht auf.
- Die Frustums überschneiden sich nicht, noch haben sie Lücken.

#### 4.2.3 General Projection Matrix Frustum

In diesem Betriebsmodus wird das FoV nach der Berechnung des Frustums noch angeglichen.

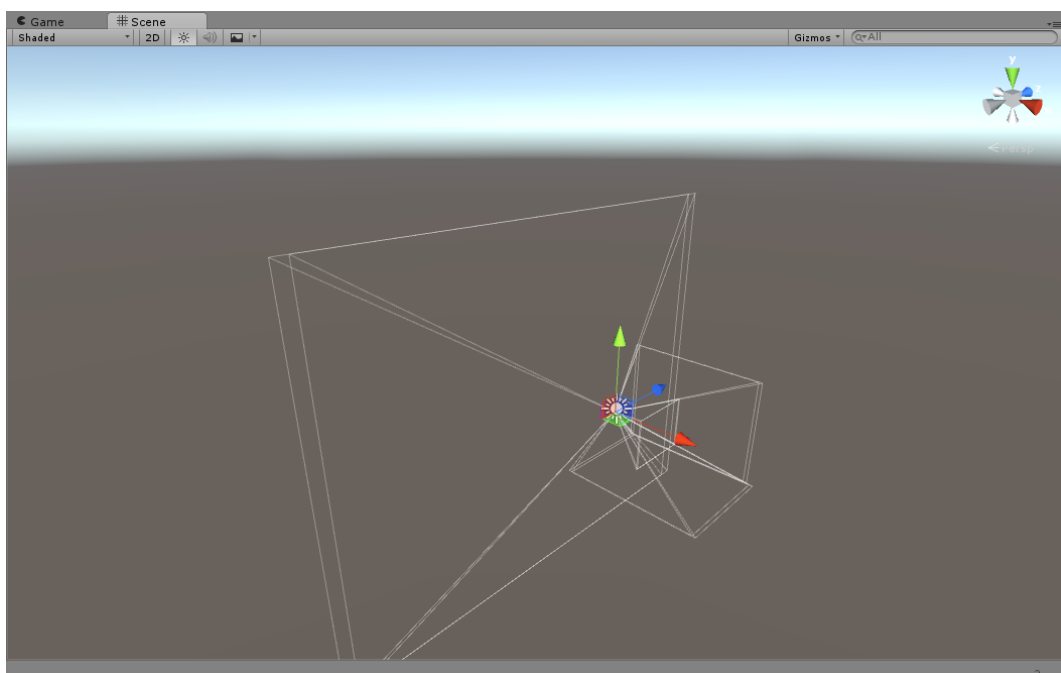


Abbildung 21: Unity Plugin Frustum GGM

Es ist sichtbar, dass hier unterschiedlich grosse Frustums in Verwendung sind, je nachdem wie nahe an der CAVE Wand man sich befindet. Die FoVs werden aufgrund des Verhältnisses von Höhe und Breite des Frustums nachgeglichen.

#### 4.2.4 Vergleich CAVE XXL und General Projection Matrix

Grundlegend basieren beide Berechnungen auf der «GPP» (Kooima, 2008). Das Frustum wird anhand einer Plane im 3D Raum aufgespannt.

##### Vorteile CAVE XXL

- Der Benutzer nähert sich der Seitenwand XXL nie so stark, dass ein unrealistisches FoV erzeugt wird
- Die Vergrößerung aller Distanzen erzeugt eine grössere Genauigkeit
- Weniger Berechnung sind nötig, da nicht noch das FoV nachberechnet werden muss und dies dann die bereits berechnete Projektionsmatrix adjustiert

##### Vorteile General Projection Matrix

- Konstrukt CAVE XXL nicht notwendig
- Anpassung FoV nach verbreiteter und akzeptierter Methode

Da der CAVE XXL für weitere Features (Raycast<sup>21</sup>, 2D GUI) verwendet wird, halbieren sich die Vorteile der General Projection Matrix. Zusätzlich ergibt das Frustum nach CAVE XXL ein besseres 3D Bild. Somit wird standardmässig diese Methode verwendet. Zu Demonstrationszwecken sind aber beide Methoden vorhanden.

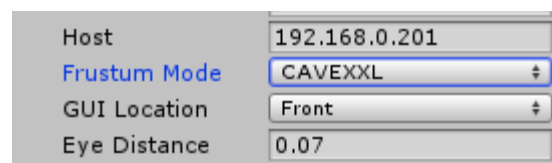


Abbildung 22: Frustum Mode

<sup>21</sup> Raycast – [https://en.wikipedia.org/wiki/Ray\\_casting](https://en.wikipedia.org/wiki/Ray_casting)

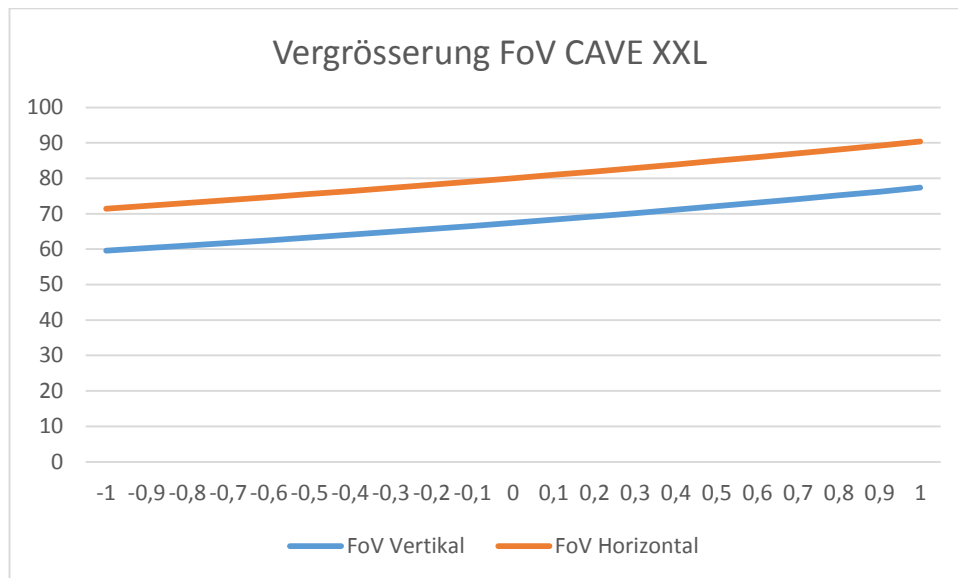


Abbildung 23: Frustum CAVE XXL FoV Anpassung

Die obige Abbildung zeigt die Vergrößerung des FoVs bei einer Bewegung im CAVE (-1 Ausserhalb des CAVEs, 1 direkt an der Projektionswand, Unity Einheiten).

## 4.3 Devices

Anstelle einer Maus oder Tastatur stehen drei Input-Devices zur Verfügung, welche das Plugin interpretieren muss. Dazu gehören der WorldViz Wand, die WorldViz Eyes sowie ein handelsübliches Gamepad. Alle Geräte haben eigene Anforderungen und Eigenschaften, die es zu verarbeiten gilt.

### 4.3.1 Wand

Die Interpretation des Wands ist ein zentrales Element für die Verwendung des CAVEs mit Unity. Er dient dazu, Objekte in der virtuellen Welt zu bewegen oder zu rotieren.

- **Virtueller CAVE**

Damit die Verarbeitung der vom Wand gelieferten Informationen vereinfacht werden können und eine visuelle Darstellung möglich ist, wurde ein virtueller CAVE in Unity erstellt, welcher dieselben Dimensionen wie der reale CAVE der BFH hat. Diese genaue Adaptierung ist möglich, weil in Unity die verwendete Grösseneinheit einem Meter in der realen Welt entspricht.

Implementiert wurde das mittels einem Prefab<sup>22</sup>, welches einmalig in der Hierarchie liegt.

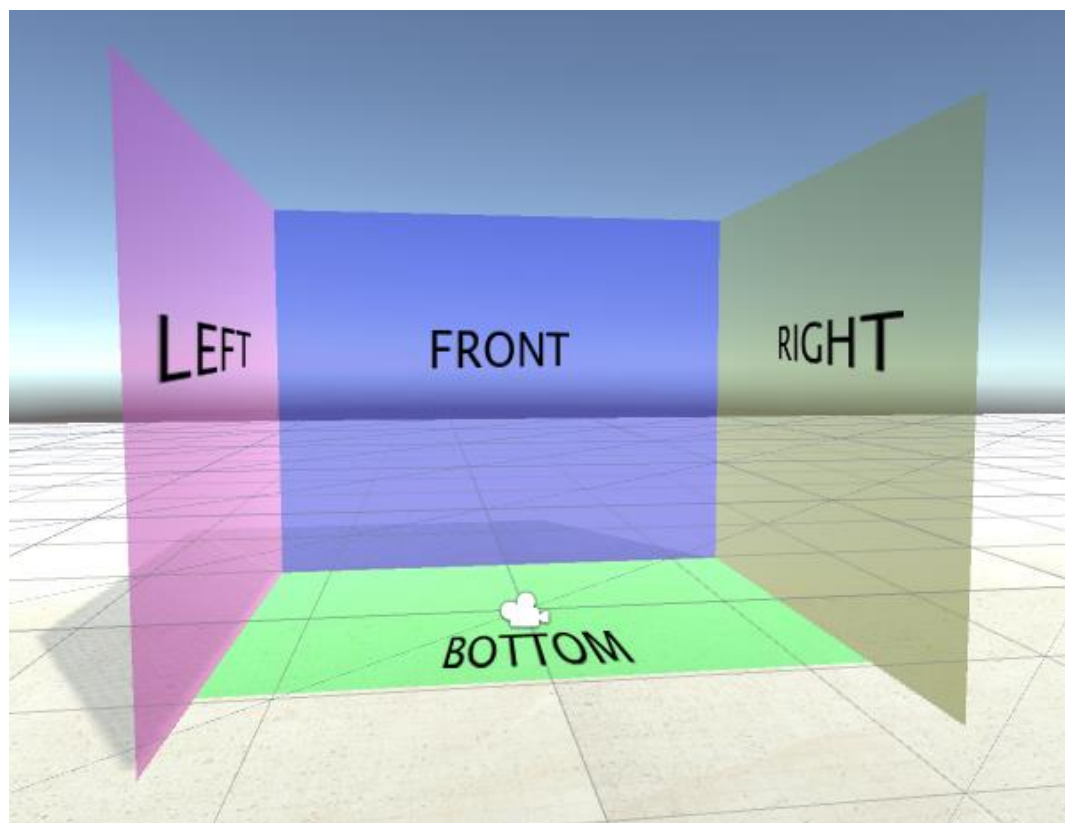


Abbildung 24: Unity Plugin, virtueller CAVE

Zusätzlich zum CAVE sind auch die beiden Devices Eyes und Wand als virtuelle Objekte in der Hierarchie der Applikation.

Der virtuelle CAVE übernimmt die Position und Rotation der Hauptkamera der Applikation. Die Korrelation zwischen CAVE und Wand / Eyes kann nur bestehen, wenn sich der Spieler in der

<sup>22</sup> Unity Prefab – <http://docs.unity3d.com/Manual/Prefabs.html>

virtuellen Welt auch im virtuellen CAVE befindet. Die Hauptkamera wird rein durch die Unity Applikation gesteuert und das Plugin hat keinerlei Einfluss darauf, um den Spielmechanismus nicht zu stören. Damit aber die Checks, wie sich die Devices im CAVE befinden, nach wie vor gemacht werden können, verändert der CAVE die Position und Rotation analog der Hauptkamera.

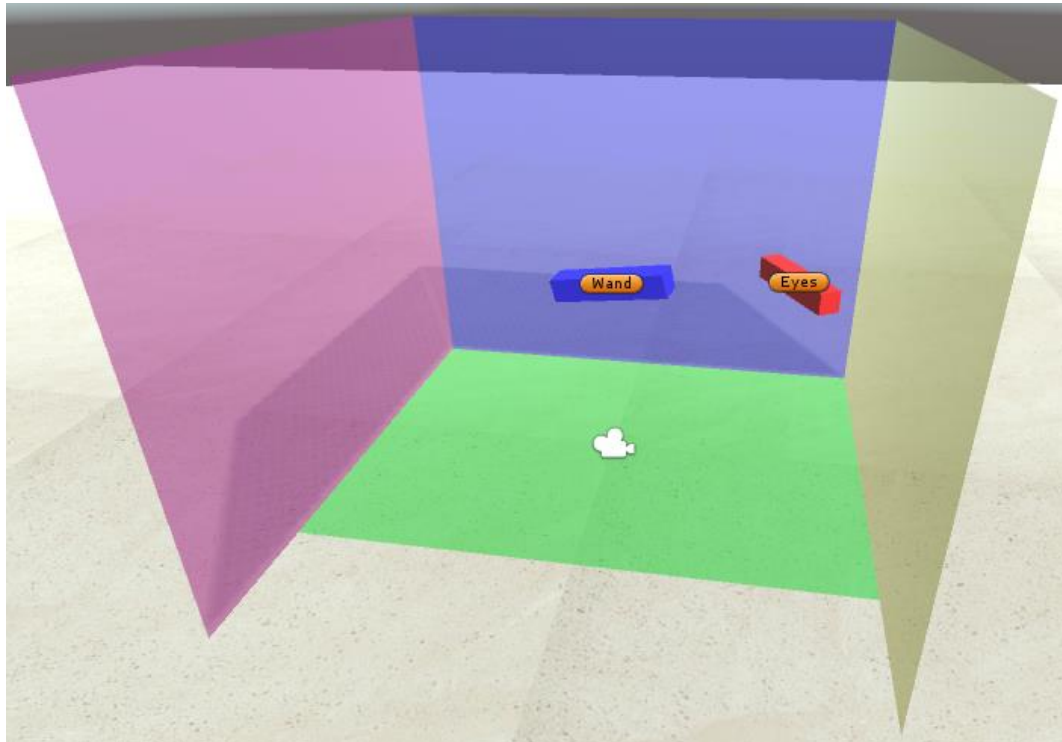


Abbildung 25: Virtueller Wand und Eyes

- **Position und Rotation**

Die realen Positions- und Rotationsveränderungen werden direkt über VRPN auf die Objekte abgebildet und ermöglichen somit ein einfaches Auslesen dieser Daten über das API.

```
var wandRotation = API.Instance.Wand.transform.rotation;
```

Quellcode 2: Zugriff über das API

Das Plugin lässt jedoch zu, einzelne Achsen bei der Positions- und Rotationsfestlegung auszuschliessen. Hierzu wird die Position, bzw. Rotation, vor der Berechnung des neuen Frames zwischengespeichert und auf die blockierten Achsen auf den ursprünglichen Wert zurückgesetzt.

Statt eine Achse komplett zu blockieren, lässt sich auch die Sensibilität adaptieren. Der Standardwert ist 1, was bedeutet, dass im realen Cave eine Verschiebung von einem Meter genau einem Meter in der virtuellen Welt entspricht. Wird die Sensibilität jedoch unter 1 gesetzt, bewegt sich der Wand langsamer und es wird eine kleinere Bewegung virtuell ausgeführt.

- **Mouse Cursor**

Um möglichst generisch die Steuerung der Applikationen übernehmen zu können, ist es unerlässlich, die Mausposition mittels Wand setzen zu können, weil das häufig das primäre Inputgerät ist. Wird auf dieser Ebene des Betriebssystems bereits Hand angelegt, entfallen spezifische Mappings auf Applikationslevel um die Steuerung übernehmen zu können.

Dazu wird ermittelt, wohin der Wand zielt. Die Verlängerung der x-Achse, also eine Gerade definiert durch die Rotation des Wands, kann einen Schnittpunkt mit einer Leinwand des CAVEs haben.

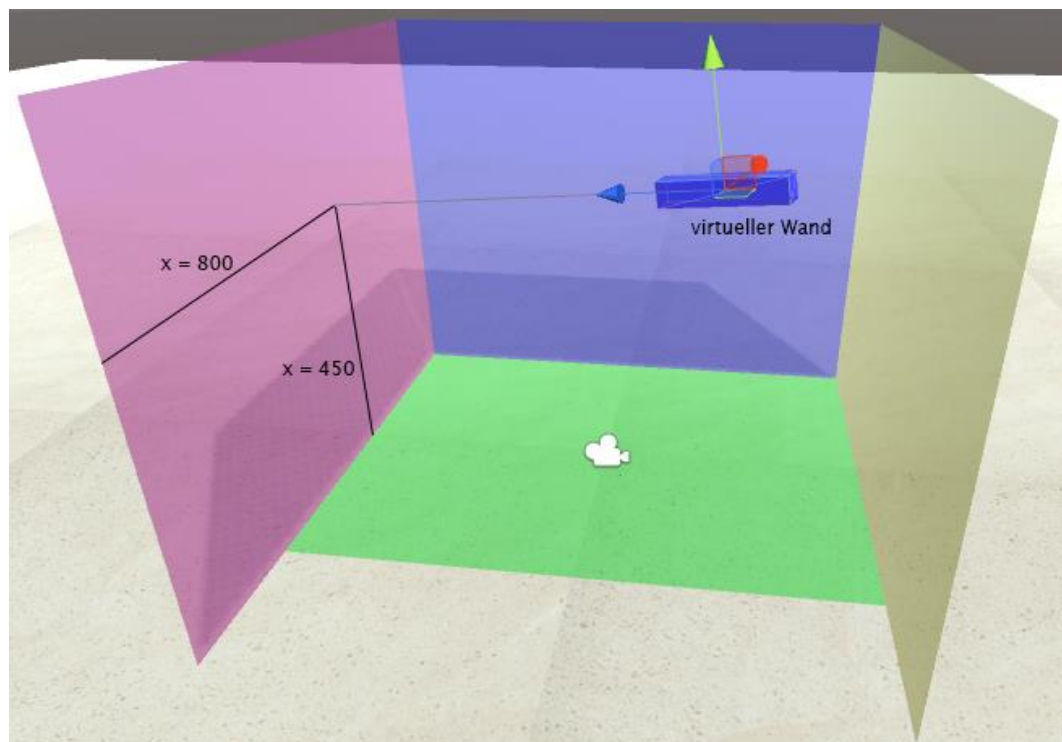


Abbildung 26: Unity Plugin, Schnittpunkt Wand / CAVE

Der Schnittpunkt wird von Unity mit einem Raycast ermittelt.

```
var fwd = transform.TransformDirection(Vector3.forward);  
Ray ray = new Ray(transform.position, fwd);  
RaycastHit hit;  
if (Physics.Raycast(ray, out hit, 100))  
{  
    Vector3 localSpaceHitPoint = hit.transform.worldToLocalMatrix.MultiplyPoint(hit.point);  
    ...  
}
```

Quellcode 3: Raycast

Ausgehend vom virtuellen Wand wird ein sogenannter Ray geschossen, welcher wahlweise nach der ersten Kollision abbricht und das getroffene Hit-Objekt zurückgibt oder durch sämtliche Colliders weiterfliegt und alle Ergebnisse als Rückgabewert liefert.

Der exakte Schnittpunkt auf der getroffenen Fläche liefert nun die benötigten Informationen, um den Mouse Cursor auf Betriebssystemebene festzulegen. Zunächst muss aber noch unterschieden werden, welche Leinwand betroffen ist. Das Mapping funktioniert so, dass bei einem Auftreffen auf die linke Leinwand der Cursor im ersten, oberen Achtel des Bildschirms festgelegt wird und der Cursor auf den zweiten, oberen Achtel mit Hilfe des GUI-Systems dupliziert wird, damit bei der stereoskopischen Projektion beide Augen den Cursor sehen. Bei der Front-Leinwand dasselbe mit dem 3. und 4. Achtel. Findet der Schnittpunkt auf der rechten oder unteren Leinwand statt, wird der Cursor in der unteren Hälfte des Betriebssystems platziert. Weitere Informationen zur Aufteilung der Viewports auf dem Unity-Server mittels Unity Plugin werden im Kapitel „Unity Plugin“ behandelt.

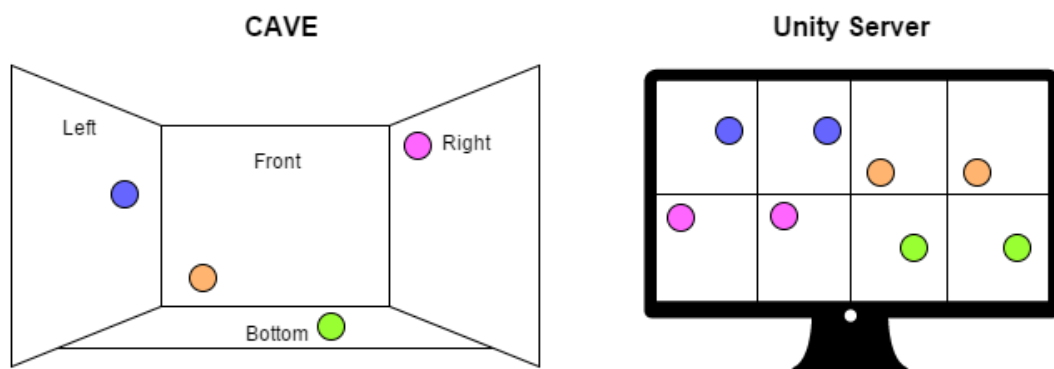


Abbildung 27: Zuordnung Schnittpunkt CAVE und Unity Server

Auf dem Unity Server ist somit jeweils der erste Punkt die reale Position des Cursors und der zweite, gleichfarbige Punkt eine Duplikation für das rechte Auge. Der reale Cursor wird jedoch nie dargestellt, weil die Kopie, welche mittels GUI-System dargestellt wird, immer einen zeitlichen Versatz aufweist und für den Benutzer ein Störfaktor darstellt. Aus diesem Grund wird der Cursor zwar platziert, damit sämtliche über den Cursor laufenden Inputs nach wie vor funktionieren, jedoch ausgeblendet und angezeigt wird eine Cursor-Grafik.

- **Buttons**

Der Wand verfügt über verschiedene Inputs. Nebst dem analogen Joystick sind ein Button auf der Rückseite und vier weitere Buttons auf der Vorderseite angebracht. Zusätzlich lässt sich der Joystick nach unten drücken.

Dem Benutzer soll die Freiheit haben zu entscheiden, welche Aktionen beim Drücken dieser Buttons ausgeführt werden und damit wiederum eine möglichst grosse Bandbreite von Applikationen abgedeckt werden können, ist für jeder Wand-Input eine Keyboard- oder Mauseingabe auswählbar.

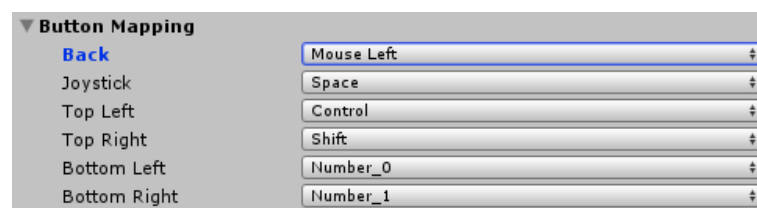


Abbildung 28: Wand Button Zuordnung

Die Auswahlmöglichkeiten beschränken sich auf eine erstellte Liste von Enums, die direkt an einen virtuellen Keycode der InputSimulator<sup>23</sup>-Bibliothek geknüpft sind. Wird nun also beim Wand ein Button betätigt, wird diese Information über das VRPN-Protokoll ans Unity Plugin geliefert und mittels dem InputSimulator ein Tastendruck simuliert. Der grosse Vorteil hier ist wiederum, dass Unity nicht unterscheiden kann, ob dieser Tastendruck von einer Tastatur kommt oder im Unity Plugin initialisiert wurde. Die Abfrage in der Applikation, ob eine Taste gedrückt wurde, kann also wie gewohnt über die Input-Klasse gemacht werden und braucht keine spezielle, vom Unity Plugin abhängige Implementierung. Das untenstehende Codesnippet zeigt, wie auf herkömmliche Weise in Unity das betätigen der Leertaste abgefragt wird und auch weiterhin mit dem umgesetzten Unity Plugin Gültigkeit hat.

```
if (Input.GetKeyDown(KeyCode.Space))
{
    // Applikationsspezifischer Code beim Betätigen der Leertaste
}
```

Quellcode 4: Unity Tastaturabfrage

Eine Ausnahme bilden die Mausclicks. Aus Sicherheitsgründen ist es nicht gestattet, mittels InputSimulator Maus-Inputs zu simulieren. Deshalb musste die user32.dll<sup>24</sup> zugezogen werden, welche erlaubt, Maus-Events zu senden. Dies geschieht wiederum auf Betriebssystem-Level und hat entsprechend keinen Einfluss auf die Interpretation in der Unity Applikation.

Weil, basierend auf der Unity-Architektur, die Button-Abfrage bei jedem Frame geschieht, würden die Inputs einmal pro gerendertem Frame simuliert werden. Das hätte zur Folge, dass selbst bei einem kurzen Klick von weniger als einer Sekunde, der Input mehrfach ausgeführt werden würde. Deshalb werden beim Auslösen desselben Buttons mittels Coroutinen mehrfache Ausführungen blockiert.

- **Joystick**

Weiter verfügt der Wand über einen analogen Joystick, welcher sich stufenlos auf zwei Achsen bewegen kann. Über die API des Unity Plugins lassen sich diese Werte einfach mittels Delegates auslesen. Es muss lediglich eine Funktion definiert werden, welche bei einem Joystick-Update aufgerufen wird. Der folgende Beispielcode zeigt, wie ein Objekt basierend auf den Joystick-Werten bewegt wird.

<sup>23</sup> Inputsimulator – <http://inputsimulator.codeplex.com/>

<sup>24</sup> User32.dll – [https://en.wikipedia.org/wiki/Microsoft\\_Windows\\_library\\_files](https://en.wikipedia.org/wiki/Microsoft_Windows_library_files)



```

void Start()
{
    // Register
    API.Instance.Wand.OnJoystickAnalogUpdate += OnJoystickUpdate;
}

private void OnJoystickUpdate(float x, float y)
{
    Vector3 posNew = transform.position;
    posNew.x += x / 10f;
    posNew.z += y / 10f;

    transform.position = posNew;
}

```

Quellcode 5: Joystick Handling

Die Joystick-Werte sind Teil der VRPN-Daten, welche der Trackingserver über das PPT Studio 2013 liefert.

#### 4.3.2 Eyes

Die Eyes von WorldViz sind notwendig für das stereoskopische Sehen und die Positions- sowie Rotationsbestimmung des Anwenders.

- **Position und Rotation**

Mit Hilfe zweier Infrarot-Trackern werden die Position im Raum und die Rotation auf zwei Achsen (Yaw und Roll) im PPT Studio 2013 aufbereitet und über VRPN an das Unity Plugin übermittelt. Eine Erfassung der dritten Rotationsachse (Pitch) kann ohne weiteren Tracker nicht erfolgen. Im Gegensatz zum Wand verfügen die Eyes auch nicht über ein Gyrometer.

Im CAVE-Prefab des Plugins sind die Eyes ebenfalls als Objekt in der Hierarchie und übernehmen die Werte vom Trackingsystem. Über das API ist somit ein einfacher Zugriff möglich. Die selektive Fixierung und Adaption der Sensibilität funktionieren analog dem Wand.

#### 4.3.3 Gamepad

Das direkt am Unity Server angeschlossene Gamepad ist nicht konfigurierbar und wird bewusst als Standard-Input belassen, um gängige Applikationen steuern zu können. Der Input-Manager von Unity deckt diese Art von Devices bereits sehr gut ab.

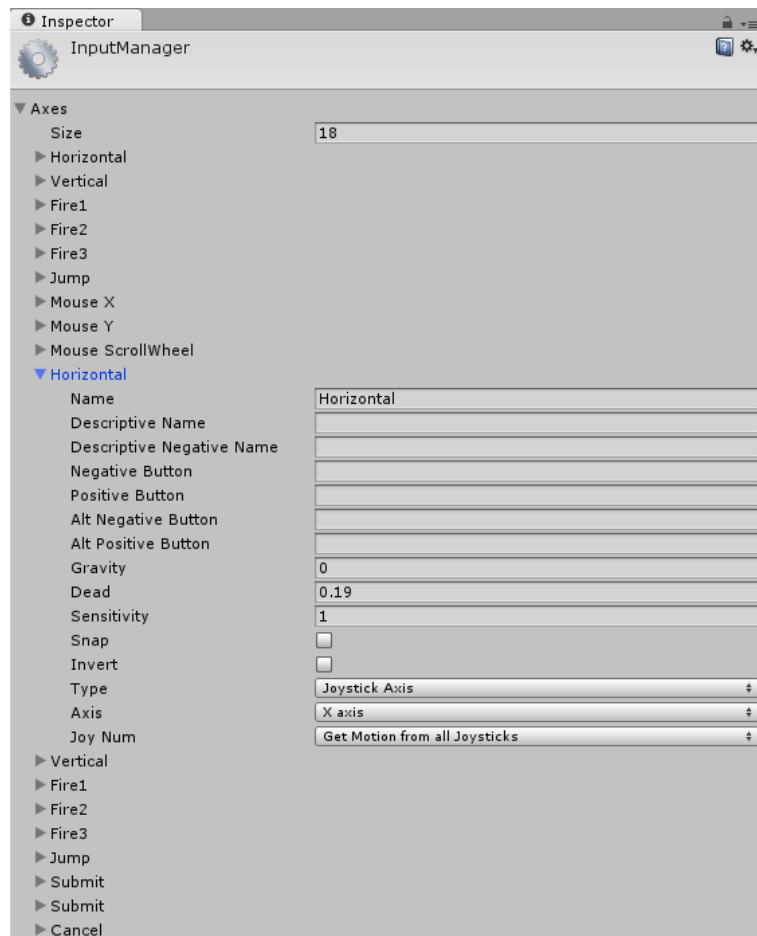


Abbildung 29: Inputmanager Unity

## 4.4 VRPN

Virtual-Reality Peripheral Network (VRPN) ist eine Klassenbibliothek und ein Server-Interface für Client-Programme und physikalische Geräte, welche in einem VR-System verwendet werden. Das Ziel von VRPN ist es, ein allgemeines Interface für Devices wie Motion-Tracking, Joysticks und weitere Geräte bereitzustellen.

### 4.4.1 Verwendung mit PPT Studio WorldViz

Das PPT Studio stellt seine Geräte über verschiedene Ausgänge zur Verfügung und verfügt über einen eingebauten VRPN Server. Dieser liefert über verschiedene Trackernamen die gewünschten Informationen.

```

#define Machine Address of PPT machine
hostname = 'localhost'

#define markerID of Wand in PPT
markerID = 3

#create a tracker object for the 6DOF data
tracker = vrpn.addTracker('PPT0@'+ hostname, markerID-1)

#create analog device for the joystick
analogDev = vrpn.addAnalog('PPT_WAND%d@s:%d' % (markerid, 8945))

#create button device for the buttons
buttonDev = vrpn.addButton('PPT_WAND%d@s:%d' % (markerid, hostname, 8945))

```

Abbildung 30: Beispielverbindung VRPN, Quelle WorldViz Dokumentation

#### 4.4.2 Datenverarbeitung im Unity

Folgende Wrapperbibliothek wurde verwendet, welche über das offizielle Git-Repository von VRPN erhältlich ist:

- Offizielles Git-Repository  
<https://github.com/vrpn/vrpn/wiki>
- UnityWrapper  
<https://github.com/arviceblot/unityVRPN>

Dieser Sourcecode wurde in das projekteigene Git-Repository verlinkt, mit CMake<sup>25</sup> kompiliert und als DLL ins Asset eingefügt. Im Unity wurde dann ein kurzer Wrapper geschrieben, welche den managed Code der Library zur Verfügung stellt:

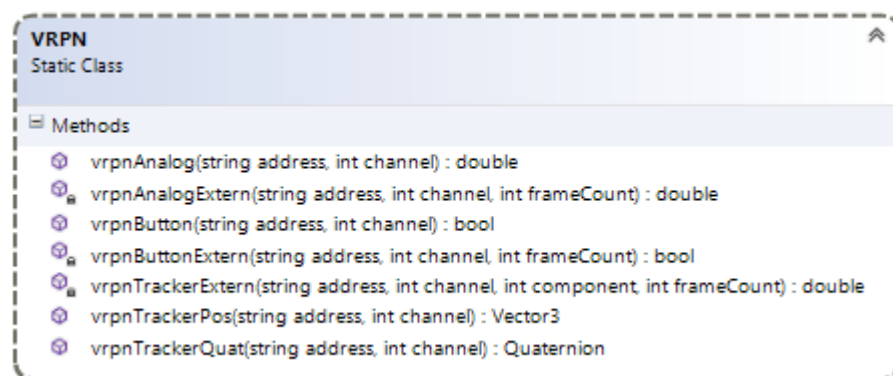


Abbildung 31: VRPN Wrapper C#

<sup>25</sup> CMake – <https://cmake.org/>

Die Weiterverarbeitung dieser Trackerdaten übernehmen dann die Klassen der Objekte, namentlich Wand und Eyes. Im Updatezyklus werden diese Daten ausgelesen, verarbeitet (Anwendung eines Smoothing-Filters) und zugewiesen (Rotation und Position).

```
private void HandlePosition()
{
    if (API.Instance.Cave.WandSettings.TrackPosition)
    {
        // Position
        var posOri = transform.localPosition;
        var pos = VRPN.vrpnTrackerPos(
            API.Instance.Cave.WandSettings.WorldVizObject + "@" +
            API.Instance.Cave.Host, API.Instance.Cave.WandSettings.Channel);

        if (_usePositionSmoothing)
        {
            Vector3 filteredPos = Vector3.zero;
            Vector3 filteredVelocity = Vector3.zero;
            OneEuroFilter.ApplyOneEuroFilter(pos, Vector3.zero, posOri,
                Vector3.zero, ref filteredPos, ref filteredVelocity,
                _posJitterReduction, _posLagReduction);
            pos = filteredPos;
        }

        // Block Axis
        if (API.Instance.Cave.WandSettings.PositionAxisConstraints.X)
            pos.x = posOri.x;
        if (API.Instance.Cave.WandSettings.PositionAxisConstraints.Y)
            pos.y = posOri.y;
        if (API.Instance.Cave.WandSettings.PositionAxisConstraints.Z)
            pos.z = posOri.z;

        transform.localPosition = pos;
    }
}
```

Quellcode 6: VRPN-Positionshandling

#### 4.4.3 Datenveredlung im Unity

Da unter Umständen ein etwas unruhiger Input über das VRPN geliefert wird, wurde ein Lowpass<sup>26</sup> Filter hinzugefügt. Dieser kann aktiviert, deaktiviert und mit zwei Parametern eingestellt werden. Der „1€ Filter“, welcher zum Einsatz kommt, beschreibt sich wie folgt:

The 1€ filter (“one Euro filter”) is a simple algorithm to filter noisy signals for high precision and responsiveness. It uses a first order low-pass filter with an adaptive cutoff frequency: at low speeds, a low cutoff stabilizes the signal by reducing jitter, but as speed increases, the cutoff is increased to reduce lag. The algorithm is easy to implement, uses very few resources, and with two easily understood parameters, it is easy to tune. In a comparison with other filters, the 1e filter has less lag using a reference amount of jitter reduction. (Géry Casiez, 2012)

<sup>26</sup> Lowpass Filter – [https://en.wikipedia.org/wiki/Low-pass\\_filter](https://en.wikipedia.org/wiki/Low-pass_filter)

Der 1€-Filter ist nur auf 2D ausgelegt, wurde jedoch um eine Dimension erweitert. Sowohl bei tiefen wie auch bei hohen Frequenzen und schnellen Bewegungen wurden optimale Resultate erzielt. Aufgrund dieser guten Resultate kam dieser Filter schlussendlich zum Einsatz. Folgende Grafik zeigt für die verschiedenen Geschwindigkeitsintervalle die durchschnittliche Distanz zwischen dem gefilterten und der aktuellen Cursorposition. (Géry Casiez, 2012)

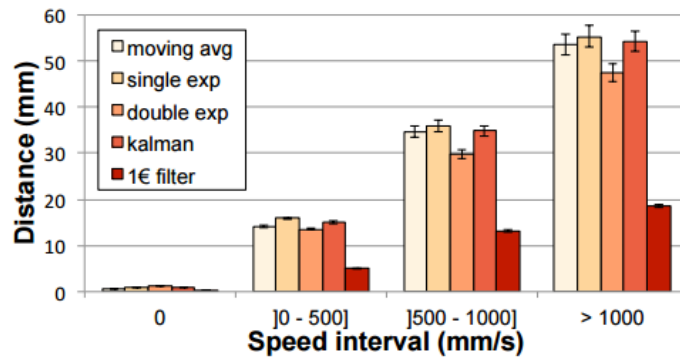


Abbildung 32: 1€ Smoothing Vergleich

## 4.5 Unity Plugin

Das resultierende Produkt ist ein Unity Plugin, welches sich in wenigen Augenblicken in die eigene Applikation integrieren lässt.

### 4.5.1 Konfiguration

Um eine möglichst weite Bandbreite von Unity Applikationen abdecken zu können, werden etliche Einstellungsmöglichkeiten zur Verfügung gestellt. Diese gliedern sich in fünf relevante Sektionen.

- **Wand**

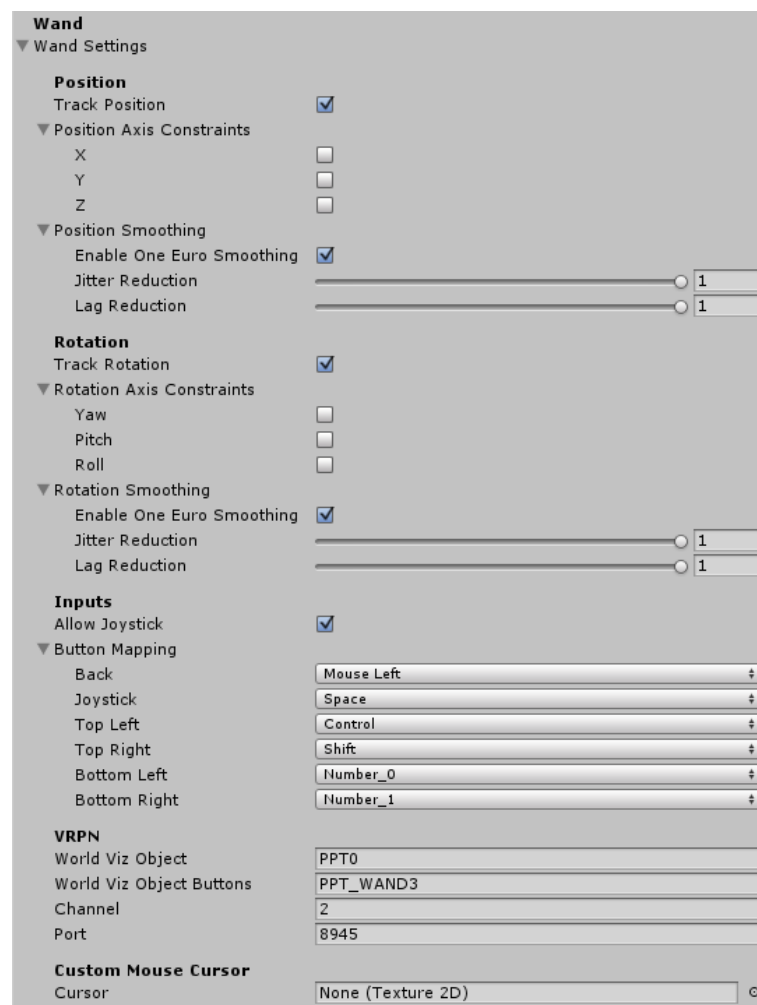


Abbildung 33: Unity Plugin, Settings Wand

#### Position

Falls die Position des Wands in der aktuellen Applikation unerheblich ist, kann die an dieser Stelle deaktiviert werden. Somit übernimmt der virtuelle Wand, welcher im Prefab liegt, keine Translationen vom realen Wand. Ist diese Option aber aktiviert, besteht die Möglichkeit, achsenabhängig die Sensibilität einzustellen. Das heisst, bei einer hohen Sensibilität auf der y-Achse vollführt der virtuelle Wand eine grosse Bewegung im Vergleich zu der realen Bewegung im CAVE. Gegenteilig, wird der Regler unter 1 gestellt, ist die virtuelle Bewegung kleiner als die reale Bewegung. Zusätzlich können einzelne Achsen auch komplett deaktiviert werden.

Weiter kann auf Wunsch die Interpolation (Smoothing) deaktiviert werden, es wird aber empfohlen, diese Option aktiv zu halten.

### Rotation

Auch die Rotation kann auf Wunsch komplett deaktiviert werden. Ebenfalls ein Smoothing ist standardmässig aktiv.

### Inputs

Der Wand verfügt über mehrere Buttons, die aus einer umfassenden Auswahlliste auf die Applikation abgebildet werden können. Zusätzlich kann der Joystick aktiviert oder deaktiviert werden.

### Custom Mouse Cursor

Falls im Spiel ein spezifischer Cursor Verwendung findet, kann die Textur hier angegeben werden, damit sie für beide Augen gerendert wird. Ansonsten wird der normale Windows-Cursor angezeigt.

### VRPN

Der VRPN-Block wird für die Anmeldeinformationen beim VRPN-Server verwendet. Diese müssen nur bei einer Umstellung des PPT-Studios (auf dem Tracking-Server) adaptiert werden.

- **Eyes**

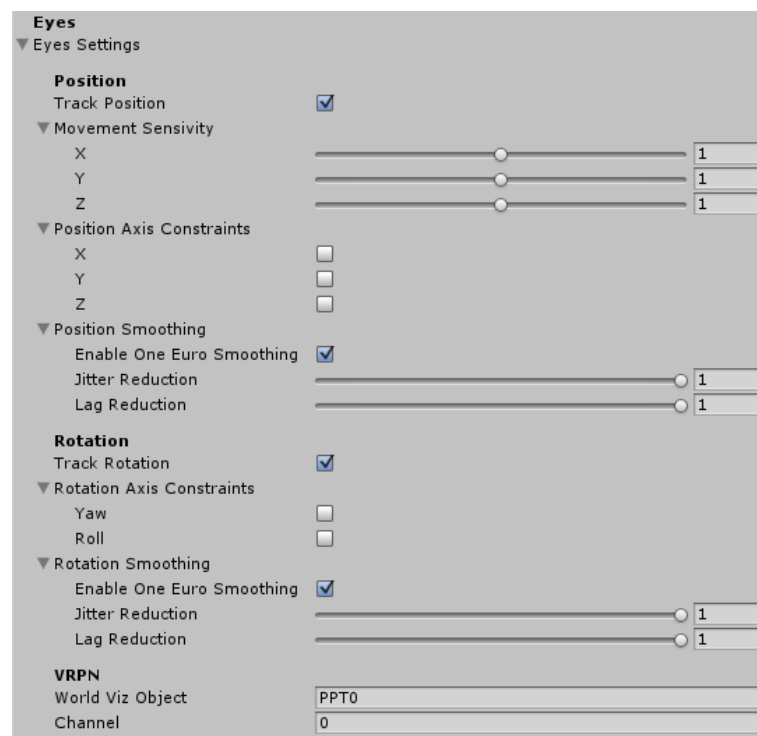


Abbildung 34: Unity Plugin, Settings Eyes

## Position

Falls die Position der Eyes in der aktuellen Applikation unerheblich ist, kann die an dieser Stelle deaktiviert werden. Somit übernehmen die virtuellen Eyes, welche im Prefab liegen, keine Translationen von den realen Eyes. Ist diese Option aber aktiviert, besteht die Möglichkeit, achsenabhängig die Sensibilität einzustellen. Das heisst, bei einer hohen Sensibilität auf der y-Achse vollführen die virtuellen Eyes eine grosse Bewegung im Vergleich zu der realen Bewegung im CAVE. Gegenteilig, wird der Regler unter 1 gestellt, ist die virtuelle Bewegung kleiner als die reale Bewegung. Zusätzlich können einzelne Achsen auch komplett deaktiviert werden.

Weiter kann auf Wunsch die Interpolation (Smoothing) deaktiviert werden, es wird aber empfohlen, diese Option aktiv zu halten.

## Rotation

Auch die Rotation kann auf Wunsch komplett deaktiviert werden. Ebenfalls ein Smoothing ist standardmässig aktiv.

## VRPN

Der VRPN-Block wird für die Anmeldeinformationen beim VRPN-Server verwendet. Diese müssen nur bei einer Umstellung des PPT-Studios (auf dem Tracking-Server) adaptiert werden.

- **Sekundäre Kameras**

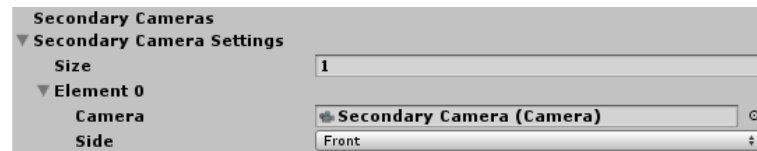


Abbildung 35: Unity Plugin, Settings Sekundäre Kameras

Möglicherweise werden für die Applikation nicht nur eine Hauptkamera, sondern auch eine oder mehrere sekundäre Kameras parallel gerendert. Eine beliebige Anzahl an Kameras können hier angegeben werden und auf welcher Seite der Leinwand die Kamera erscheinen soll.

- **Cave**

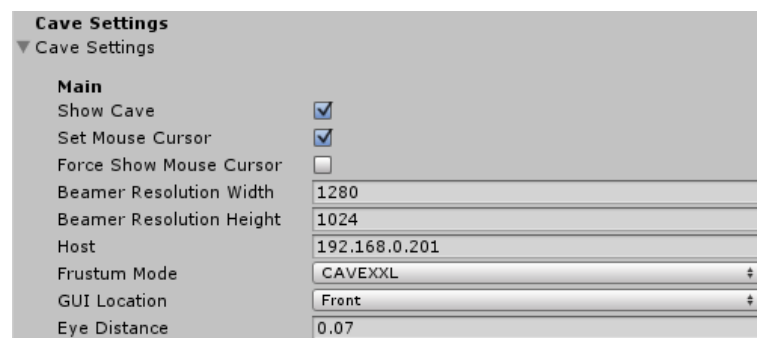


Abbildung 36: Unity Plugin, Settings Cave

In dieser Sektion muss in den meisten Fällen keine Einstellungen vorgenommen werden. Möglicherweise will der Benutzer aber die GUI-Elemente auf einer anderen Leinwand darstellen.



- **System**

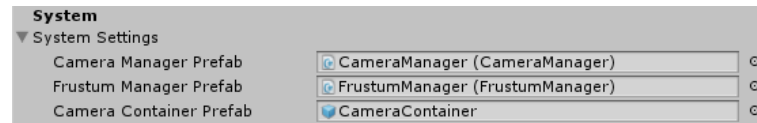


Abbildung 37: Unity Plugin, Settings System

Die zugewiesenen Prefabs werden beim Initialisieren des Plugins instanziiert und müssen nicht adaptiert werden.

Das CameraContainer Gameobject beinhaltet alle Kameras, die sich jeweils der Hauptkamera unterordnen und die Bildaufteilung für die verschiedenen Beamer übernimmt.

Einstellungen müssen nur bei grundlegenden Veränderungen gemacht werden und empfehlen sich nur für erfahrene Benutzer.

All diese Einstellungen sind ebenfalls über das bereitgestellte API manipulierbar. Folgender Beispielcode zeigt eine einfache Einstellungsänderung in den Wand Settings.

```
API.Instance.Cave.WandSettings.TrackPosition = false;
```

Quellcode 7: Einstellungsänderung über API

#### 4.5.2 Aufgabenverteilung

Der zentrale Knoten des Plugins ist die Klasse „CaveMain“. Hier sind alle Einstellungsmöglichkeiten, die Referenzen auf sämtliche Objekte und die Geometrie der virtuellen CAVEs gespeichert.

Ein direkter Zugriff auf verschiedene Komponenten sollte grundsätzlich nicht erfolgen, dafür wird ein API als Schnittstelle für Unity Applikationsprogrammierer zur Verfügung gestellt.

Das untenstehende Klassendiagramm zeigt sämtliche Klassen und deren wichtigsten Assoziationen auf.

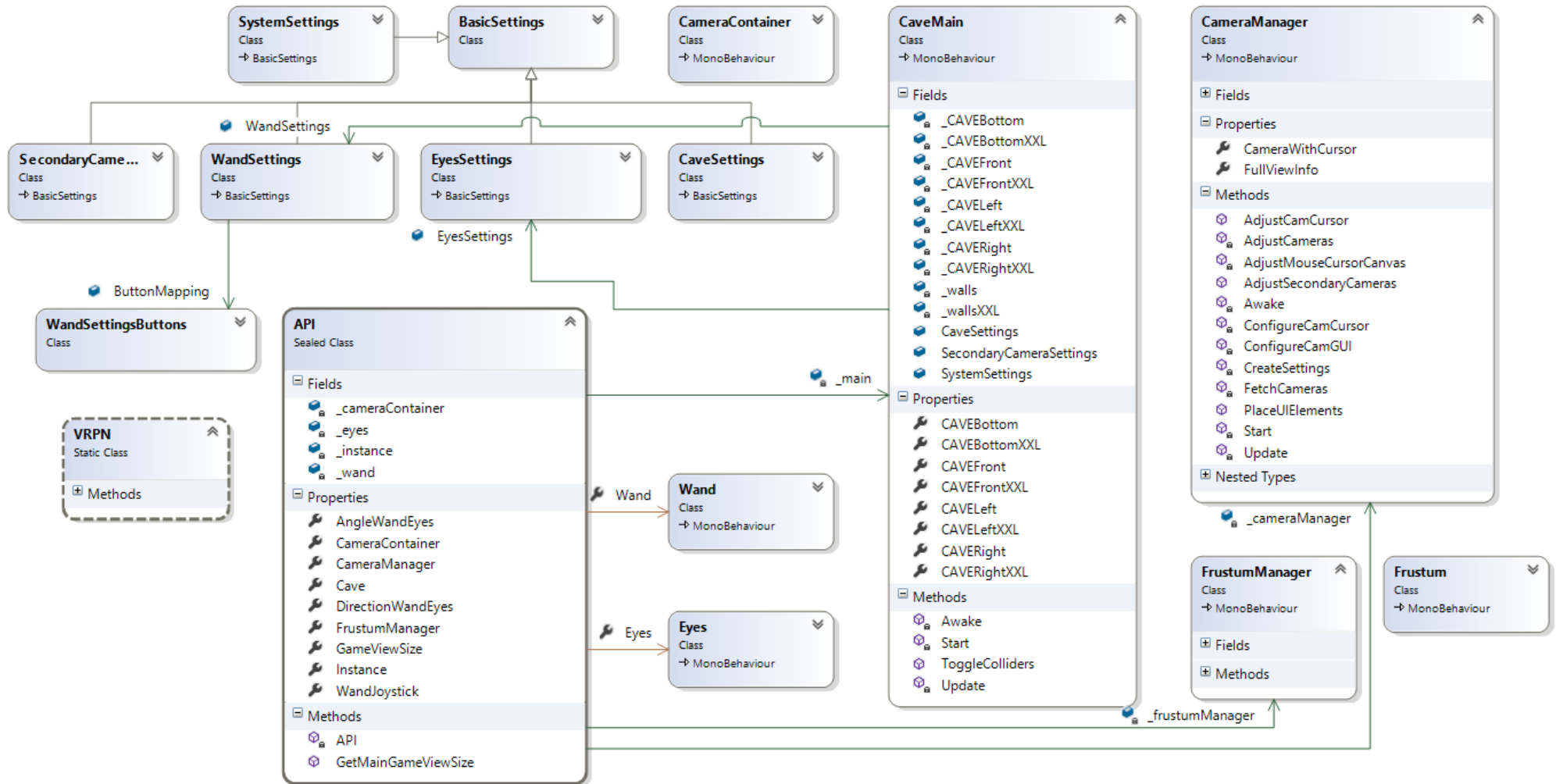


Abbildung 38: Unity Plugin, Klassendiagramm

Die wichtigsten Klassen und deren Methoden werden folgend kurz beschrieben.

- **CaveMain.cs**

Das ist der zentrale Knoten des Plugins. In der Unity-Hierarchie sind sämtliche Plugin-relevanten Objekte Child-Elemente dieser Instanz. Wichtige Management-Klassen werden während des Startvorgangs vom CaveMain instanziiert und die Geometrie des virtuellen CAVEs wird hier zusammengetragen. Der Zugriff auf die benutzerspezifischen Einstellungen, welche in Model-Klassen ausgelagert sind, erfolgt über diese Stelle. Je nach Einstellung, ob der CAVE zu Debug-Zwecken dargestellt werden soll, deaktiviert diese Klasse sämtliche Renderers.

Methode	Aufgabe
Awake()	Instanziiert CameraManager, FrustumManager und CameraContainer
Start()	<ul style="list-style-type: none"><li>• Erstellt eine Kollektion der Cave-Transforms</li><li>• Deaktiviert auf Wunsch sämtliche Renderer</li><li>• Deaktiviert die Collider des virtuellen Caves</li></ul>
Update()	Übernimmt die Position sowie Rotation der Hauptkamera.
ToggleColliders()	Schaltet die Collider der CAVE-Wände ein, aus.

Tabelle 2: CaveMain Methoden

Enum	Aufgabe
FrustumMode	Auswahl für CAVE XXL und GPP_Kooima. Steuert, wie das Frustum berechnet wird.

Tabelle 3: CaveMain Enums

- **Eyes.cs**

Die virtuellen Eyes interpretieren die Daten des Trackingsservers über VRPN und beachten die benutzerspezifischen Einstellungen des Plugins. Nach dieser Bearbeitung wird das in der Hierarchie liegende Eyes-Objekte aktualisiert, damit mittels API dessen Werte ausgelesen werden können.

Methode	Aufgabe
Start()	Speichert etliche über das API erreichbare Daten in privaten Variablen.
Update()	Führt in jedem Frame die Methoden „HandlePosition“ und „HandleRotation“ aus.
HandlePosition()	<ul style="list-style-type: none"> <li>• Position von VRPN übernehmen</li> <li>• Smoothing durchführen</li> <li>• Blockierte Achsen zurücksetzen</li> <li>• Neue Position setzen</li> </ul>
HandleRotation()	<ul style="list-style-type: none"> <li>• Rotation von VRPN übernehmen</li> <li>• Smoothing durchführen</li> <li>• Blockierte Achsen zurücksetzen</li> <li>• Neue Rotation setzen</li> </ul>

Tabelle 4: Eyes Methoden

- **Wand.cs**

Der Wand interpretiert analog den Eyes die Daten des Trackingsservers über VRPN und aktualisiert unter Berücksichtigung der benutzerspezifischen Einstellungen das Wand-Objekt. Der Wand ist aber noch für etliche weitere Aufgaben zuständig. Alle Inputs, die der Wand liefert, werden ebenfalls an dieser Stelle bearbeitet. Das umfasst das Setzen der Cursor-Position und dessen eventuell vorhandenen Cursor-Textur, die Bereitstellung der Joystick-Daten und die Input-Simulation der verschiedenen Buttons.

Methode	Aufgabe
Start()	Speichert etliche über das API erreichbare Daten in privaten Variablen und setzt, falls vom Benutzer gewünscht, eine spezifische Cursor-Textur.
Update()	Führt in jedem Frame die Methoden „HandlePosition“, „HandleRotation“, „HandleButtons“, „HandleJoystick“ und „SetCursor“ aus.
HandlePosition()	<ul style="list-style-type: none"> <li>• Position von VRPN übernehmen</li> <li>• Smoothing durchführen</li> <li>• Blockierte Achsen zurücksetzen</li> <li>• Neue Position setzen</li> </ul>
HandleRotation()	<ul style="list-style-type: none"> <li>• Rotation von VRPN übernehmen</li> <li>• Smoothing durchführen</li> <li>• Blockierte Achsen zurücksetzen</li> <li>• Neue Rotation setzen</li> </ul>
HandleButtons()	<ul style="list-style-type: none"> <li>• Empfängt die Button-Inputs über VRPN</li> <li>• Simuliert die im Inspector zugewiesenen Inputs (Tastatur sowie Maus)</li> <li>• Stellt sicher, dass derselbe Input nicht versehentlich mehrmals hintereinander ausgeführt wird</li> </ul>

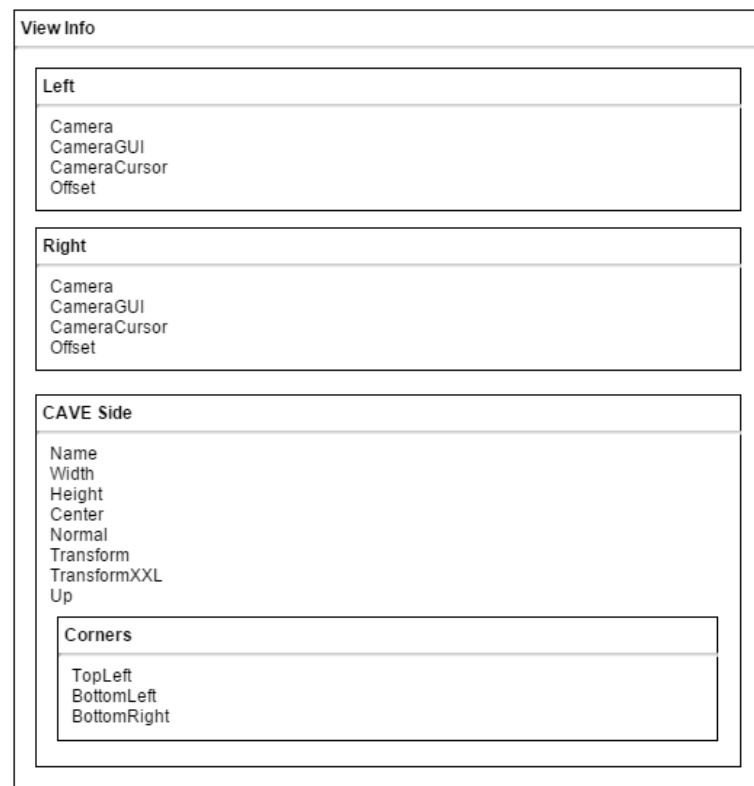
HandleJoystick()	<ul style="list-style-type: none"> <li>• Empfängt die Joystick-Inputs über VRPN</li> <li>• Führt die registrierten Delegates aus mit der aktuellen Joystick-Position</li> </ul>
SetCursor()	<ul style="list-style-type: none"> <li>• Raycast auf den virtuellen Cave</li> <li>• Leinwand ermitteln</li> <li>• Cursor-Kamera adaptieren</li> <li>• 2D-Position auf Screen ermitteln</li> <li>• Cursor-Position setzen</li> <li>• Cursor-Position des Duplikats setzen</li> </ul>

Tabelle 5: Wand Methoden

- **CameraManager.cs**

Das Aufgabenspektrum des CameraManagers befasst sich mit allen Tasks, die in Verbindung mit einer Kamera stehen. Vorgängig wird hier das grundlegende Setting der multiplen Kamera-Konstruktion für das stereoskopische Sehen vorgenommen und alle relevanten Informationen zu den CAVE-Seiten werden in Structs gespeichert.

Jede Seite des CAVEs, also 4 Stück, enthält Daten über die Kameras je Auge, die Abmessungen der Leinwand, die Eckpunkte der Leinwand usw. Diese Daten finden während dem gesamten Prozess Verwendung, hauptsächlich jedoch bei der Frustumberechnung. Folgend sind die verwendeten Structs.



Quellcode 8: CameraManager DataStruct

Methode	Aufgabe
Awake()	Setzt CaveMain als Parent.
Start()	Verhindert das Rendern von UI-Elementen auf der Hauptkamera und führt die Methoden „FetchCameras“, „CreateSettings“, „AdjustCameras“, „AdjustSecondaryCameras“ und „PlaceUIElements“ aus.
Update()	Ermittelt, auf welcher Kamera sich der Systemcursor befindet.
FetchCameras()	Speichert die Referenzen auf die verschiedenen Kameras.
CreateSettings()	Füllt die oben abgebildeten Structs mit Daten ab und speichert sie in einem Dictionary.
AdjustCameras()	Übernimmt die Grundeinstellungen der Hauptkamera, macht spezifische Einstellungen je Seite und positioniert die Viewports.
AdjustSecondaryCameras()	Konfiguriert die vorhandenen sekundären Kameras.
PlaceUIElements()	Konfiguriert die GUI-Kameras und platziert die Canvas.
AdjustCamCursor()	Die für den Cursor zuständigen Kameras werden je nach Cursorposition ausgerichtet und ein- / ausgeschaltet.

Tabelle 6: CameraManager Methoden

Enum	Aufgabe
CameraDepths	Steuert, auf welchem Layer sich die Kamera befindet. Sodass beispielsweise das GUI nicht durch das Spiel verdeckt wird.

Tabelle 7: CameraManager Enums

- **CameraContainer.cs**

Dieser Container beinhaltet alle dynamisch, vom Unity Plugin generierten Kameras, um sie zentral verschieben zu können. Zur Laufzeit wird geprüft, ob eine andere Kamera als Hauptkamera definiert wurde und verschiebt sich entsprechend dorthin.

Die Position CameraContainer-Objects ist direkt an die Position der Eyes gebunden. Und weil die Sensibilität, also in welchem Verhältnis sich die virtuellen Eyes zu den realen Eyes verschieben, eingestellt werden können, muss diese Berechnung hier geschehen. Beim Setzen der Position wird eine einfache Vektormultiplikation durchgeführt.

Methode	Aufgabe
Start()	Die Bewegungssensibilität wird zwischengespeichert.
Update()	<ul style="list-style-type: none"> <li>• Prüfung, ob der Container noch an der Hauptkamera angehängt ist und eventueller Parentwechsel.</li> <li>• Setzen der Position basierend auf den Eyes sowie Sensibilität beachten.</li> </ul>

Tabelle 8: CameraContainer Methoden

- **FrustumManager.cs**

Der FrustumManager sammelt die benötigten geometrischen CAVE-Daten und die dazugehörigen Kameras, um mittels Frustum-Klasse das auf die Kameras abzubildende Viewfrustum zu berechnen.

Methode	Aufgabe
Awake()	Setzt CaveMain als parent.
Update()	Nimmt die vom CameraManager zusammengetragenen Daten und bereitet sie für die Frustum-Berechnung auf.

Tabelle 9: FrustumManager Methoden

- **Frustum.cs**

Berechnet für die ihm angegebene Kamera das Viewfrustum, basierend auf der Eyes-Position, den drei Eckpunkten einer CAVE-Seite sowie der Near- und Farplane. Siehe dazu Kapitel 4.2.1

- **VRPN.cs**

Diese Klasse übernimmt die Schnittstelle zwischen C# (Unity) und C++ (VRPN). Alle Daten, die vom Trackingserver aufbereitet und übers VRPN-Protokoll verschickt werden, werden hier empfangen und in C# Datentypen umgewandelt, um die Verwendung zu vereinfachen.

- **API.cs**

Grundsätzlich kann der gesamte Sourcecode des Plugins eingesehen und verändert werden. Um die Verwendung jedoch zu vereinfachen und die eigene Anwendung applikationsspezifisch mit dem Plugin zu verknüpfen, werden gewissen Werte, Berechnungen und Objekte in der API zur Verfügung gestellt. Mittels Singleton-Pattern wird sichergestellt, dass die Verwaltung der besagten Properties zentral an einem Ort geschieht und dort abgegriffen werden können.

Der untenstehende Code der Klasse API zeigt, wie beispielsweise der Wand gesucht, referenziert und zurückgegeben wird. Ebenfalls werden gewisse Berechnungen, in diesem Beispiel der relative Winkel zwischen dem Wand und den Eyes, direkt berechnet und können abgefragt werden.

```

public sealed class API
{
    static readonly API _instance = new API();

    public Wand Wand
    {
        Get
        {
            if(_wand == null)
            {
                return _wand = GameObject.Find("WorldVizWand")
                    .GetComponent<Wand>();
            }

            return _wand;
        }
    }

    ...

    public Quaternion AngleWandEyes { get { return Quaternion
        .Inverse(Eyes.transform.rotation) * Wand.transform.rotation; } }

    public static API Instance
    {
        get
        {
            return _instance;
        }
    }

    ...
}

```

Quellcode 9: API

Der Zugriff auf das API erfolgt, weil es sich um ein Singleton-Pattern handelt, über die selber erstellte Instanz.

```

var angle = API.Instance.AngleWandEyes;

```

Quellcode 10: Zugriff über das API

### 4.5.3 Deployment

Das entwickelte Unity Plugin muss möglichst unkompliziert und rasch in die gewünschte Applikation integriert werden können. Um das zu erreichen, wird der gleiche Ansatz wie das Deployment über den integrierten Unity Asset Store gewählt. Dazu werden sämtliche Verzeichnisse und Dateien in eine .unitypackage-Datei gepackt und können in jedes beliebige Projekt importiert werden.

Um diese Bündelung der Dateien zu erreichen, gibt es in Unity den Befehl, ein Package zu exportieren.



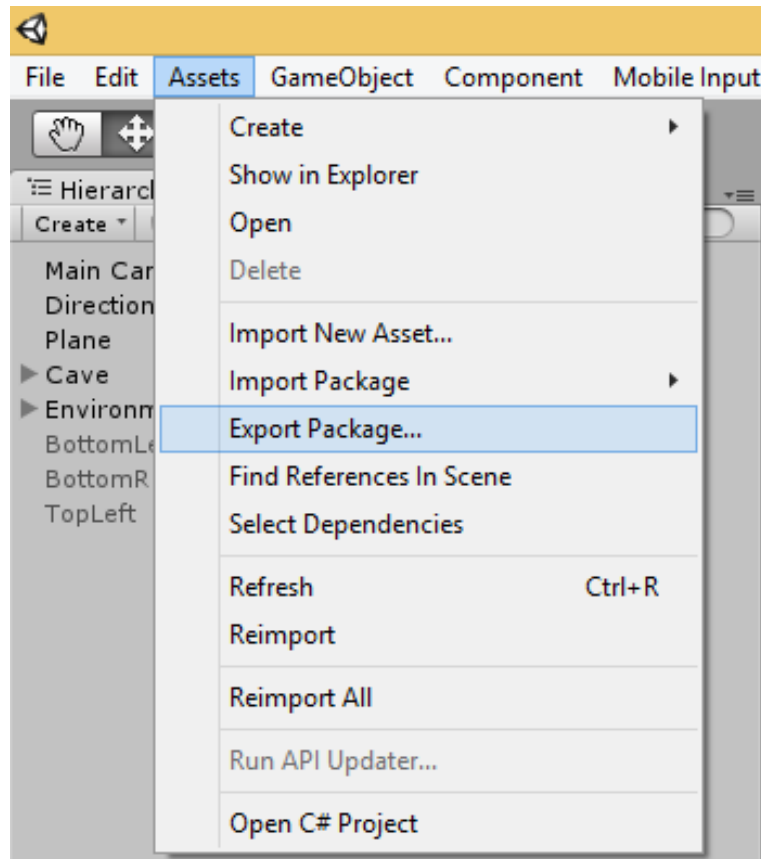


Abbildung 39: Unity Plugin, Export Package

Beim anschließenden Popupmenu alle Assets auswählen und den Export starten.

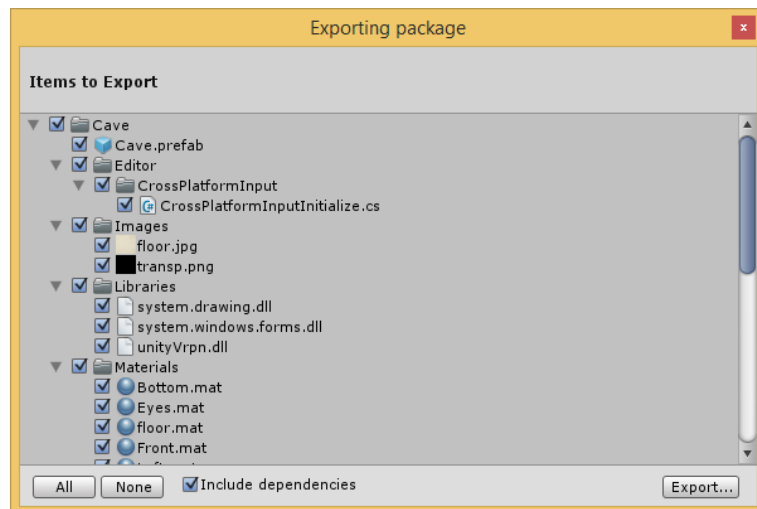


Abbildung 40: Unity Plugin, Export Package Selection

Die Verwendung des Plugins mit sämtlichen Abhängigkeiten erfolgt in wenigen Schritten:

### 1. Import des Plugins

Abhängig der Unity-Version kann der Import zu Problemen führen. Sind gewisse, bereits existierende Files in Verwendung, werden die Klassen / Prefabs / usw. nicht überschrieben, sondern lediglich hinzugefügt (CaveMain 1.cs, CaveMain 2.cs usw.). Darum empfiehlt sich, den

Import manuell über den Explorer vorzunehmen und nicht die exportierte .unitypackage-Datei zu verwenden.

Als erstes muss das Unity-Projekt geschlossen werden, damit alle Dateien und Verzeichnisse über Schreibrechte verfügen. Falls bereits eine andere Version des Plugins ins Projekt integriert wurde, den Ordner „Cave“ im Assets-Verzeichnis löschen und die neue Version reinkopieren.

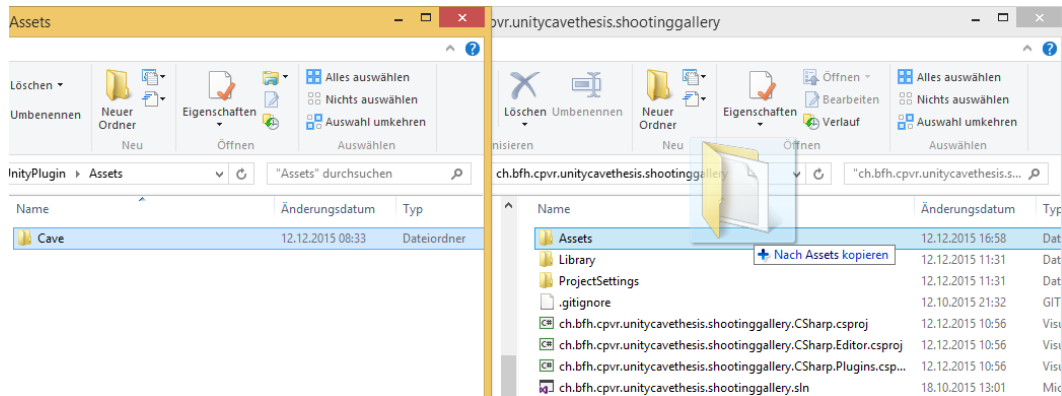


Abbildung 41: Kopieren des Unity Plugins

Alternativ kann der Asset Import-Mechanismus von Unity verwendet werden. Diese Methode, auch wenn sie elegant erscheinen mag und ursprünglich so angedacht war, führt leider oft zu einem Fehlverhalten und es wird an dieser Stelle abgeraten, das Asset auf diese Weise zu importieren.

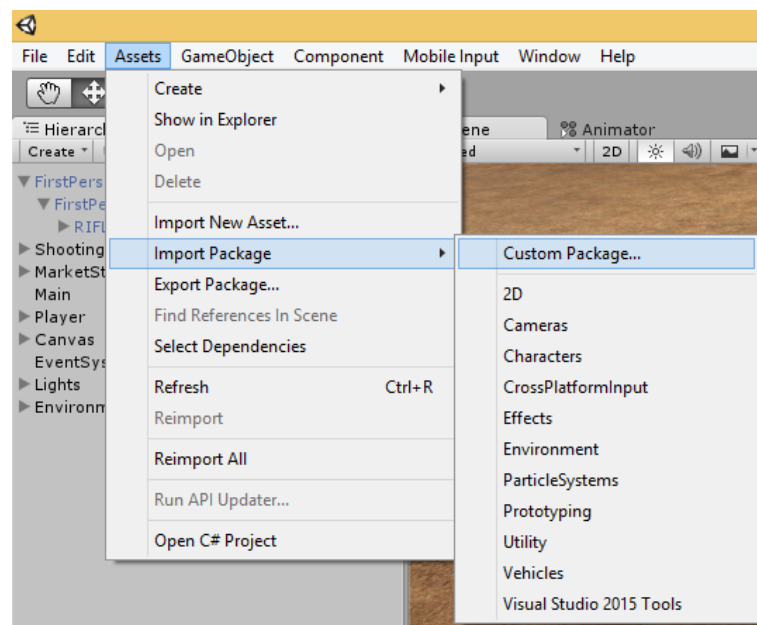


Abbildung 42: Unity Plugin, Import Package

Beim anschließenden Popupmenu sind alle Assets auszuwählen und mit „Import“ zu bestätigen.

## 2. Cave-Prefab

In einem zweiten Schritt gilt es, das Cave-Prefab, welches sich direkt im Verzeichnis „Cave“ befindet, an einer beliebigen Stelle in der Szene zu platzieren.

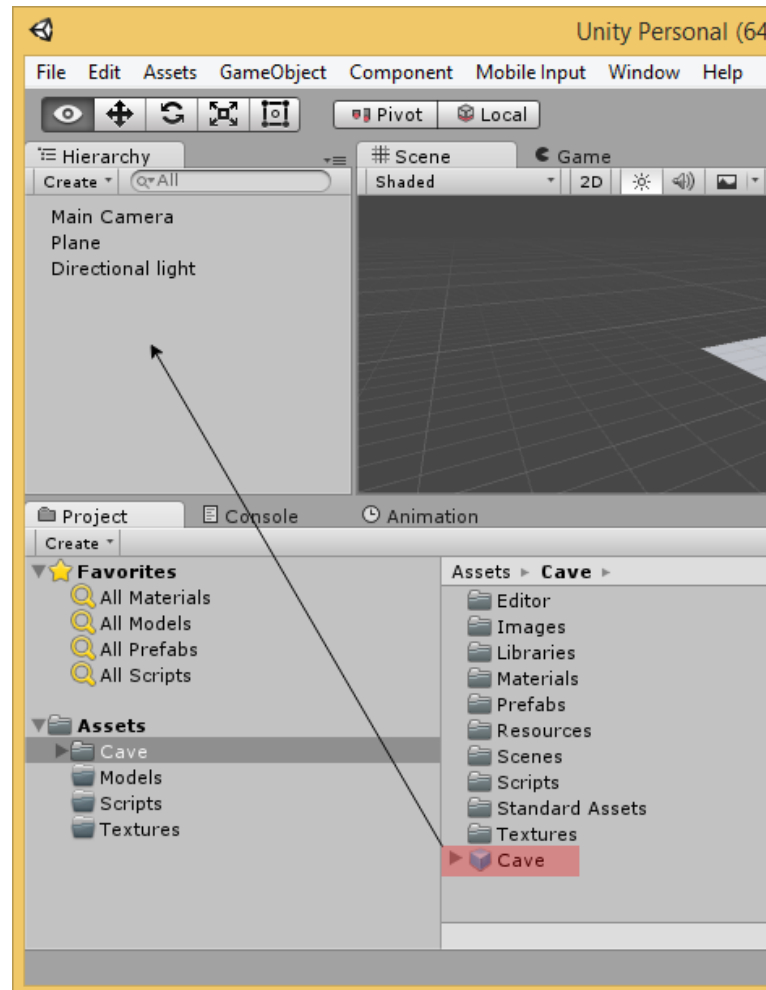


Abbildung 43: Hinzufügen Cave Prefab

### 3. API Kompatibilitätslevel

Weil das Unity Plugin sich weiterer DLLs bedient, muss bei den Player-Settings das „Api Compatibility Level“ auf „.NET 2.0“ (ohne Subset) gesetzt werden. Damit wird die Verwendung externer DLLs ermöglicht.

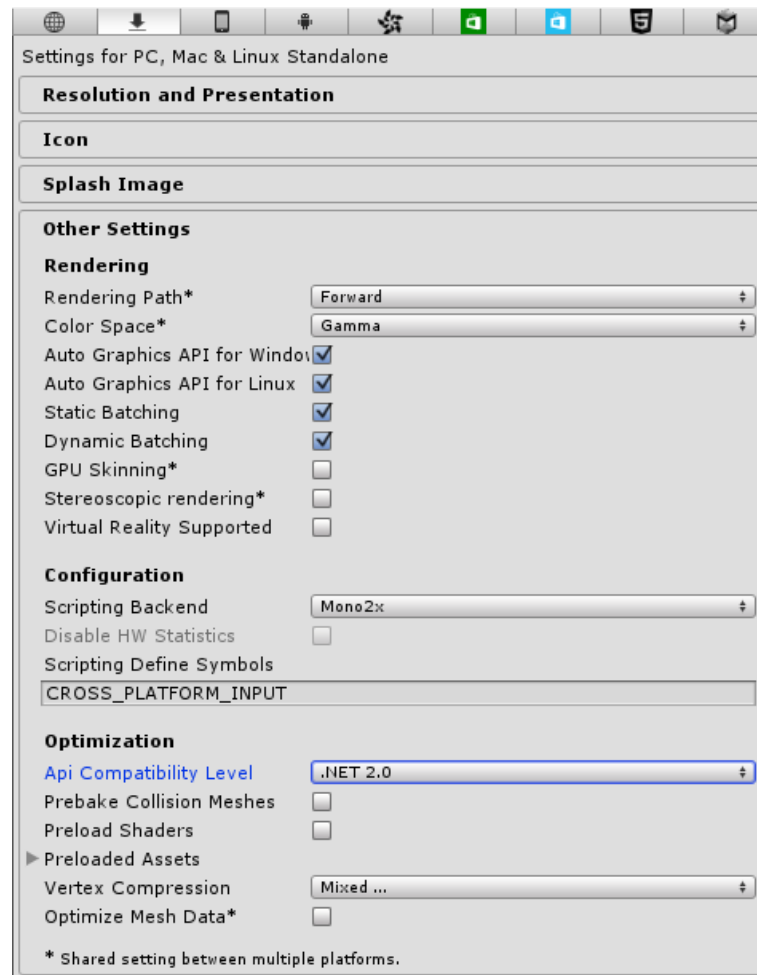


Abbildung 44: Player Settings, Api Compatibility Level

#### 4. Plattform Einstellungen

Um sicherzustellen, dass die zusätzlichen DLLs für die Zielplattform exportiert werden, im Projekt zu den DLLs navigieren (Cave -> Libraries) und bei allen DLLs die entsprechenden Häkchen setzen.

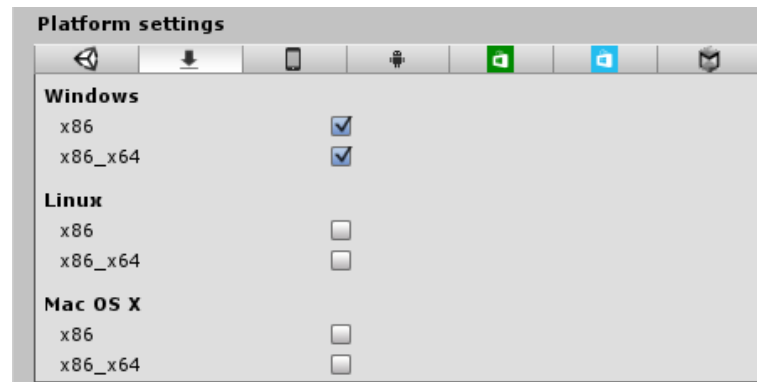


Abbildung 45: Platform Settings

#### 5. Einsatzbereit

Nach diesen einfachen Schritten wurde das Plugin erfolgreich in die Applikation eingebettet und kann verwendet werden.

##### 4.5.4 Troubleshooting

- **Fehlende Verlinkung Prefabs**

Es besteht die Möglichkeit, dass die Verbindung zu den Prefabs vom CameraManager, FrustumManager und CameraContainer unter den System Settings beim Kopieren verloren geht. In diesem Fall direkt in Unity das entsprechende Prefab aus dem Ordner „Cave / Prefabs“ an das dafür vorgesehene Property hängen.

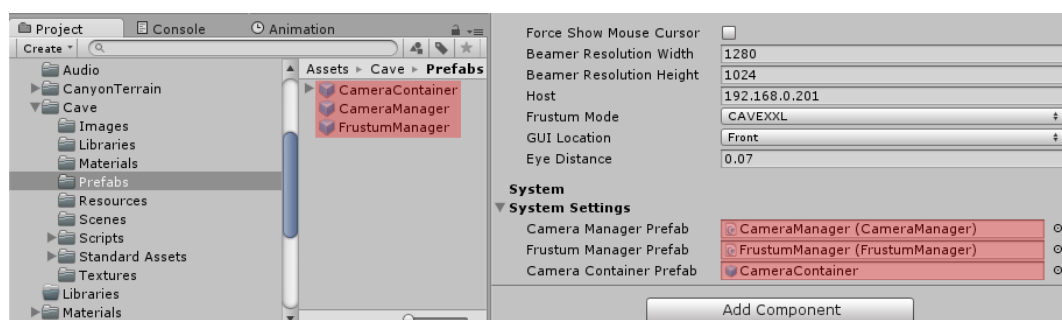


Abbildung 46: Zuweisung Prefabs

- **Falsche Architektur**

Falls beim Exportieren Fehler auftreten sollten, muss die Architektur möglicherweise auf 64bit angepasst werden.

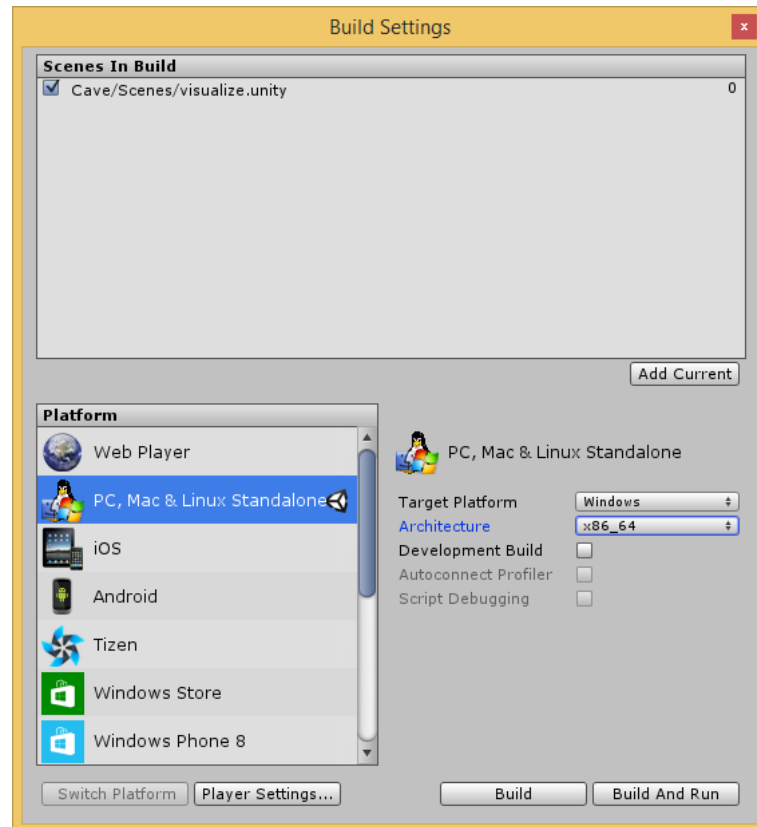


Abbildung 47: Export 64bit

#### 4.5.5 Performance Verbesserungen

Um eine möglichst gute Performance zu erreichen, wurde bereits während der Entwicklung darauf geachtet, Scripts nicht unnötig auszuführen. Dennoch wurde zum Schluss der Code nochmals analysiert und geprüft, wo Verbesserungen durchgeführt werden können.

Im folgenden Profiling-Screenshot ist deutlich zu sehen, dass fast sämtliche Ressourcen nur noch für die Berechnung der Kameras verwendet werden. Alle anderen Aufgaben sind weitgehend optimiert und haben keinen markanten Einfluss auf den Spielfluss. Das relativ kostspielige Rendering aller Kameras lässt sich nicht umgehen und ist ein wichtiger Bestandteil des Plugins.

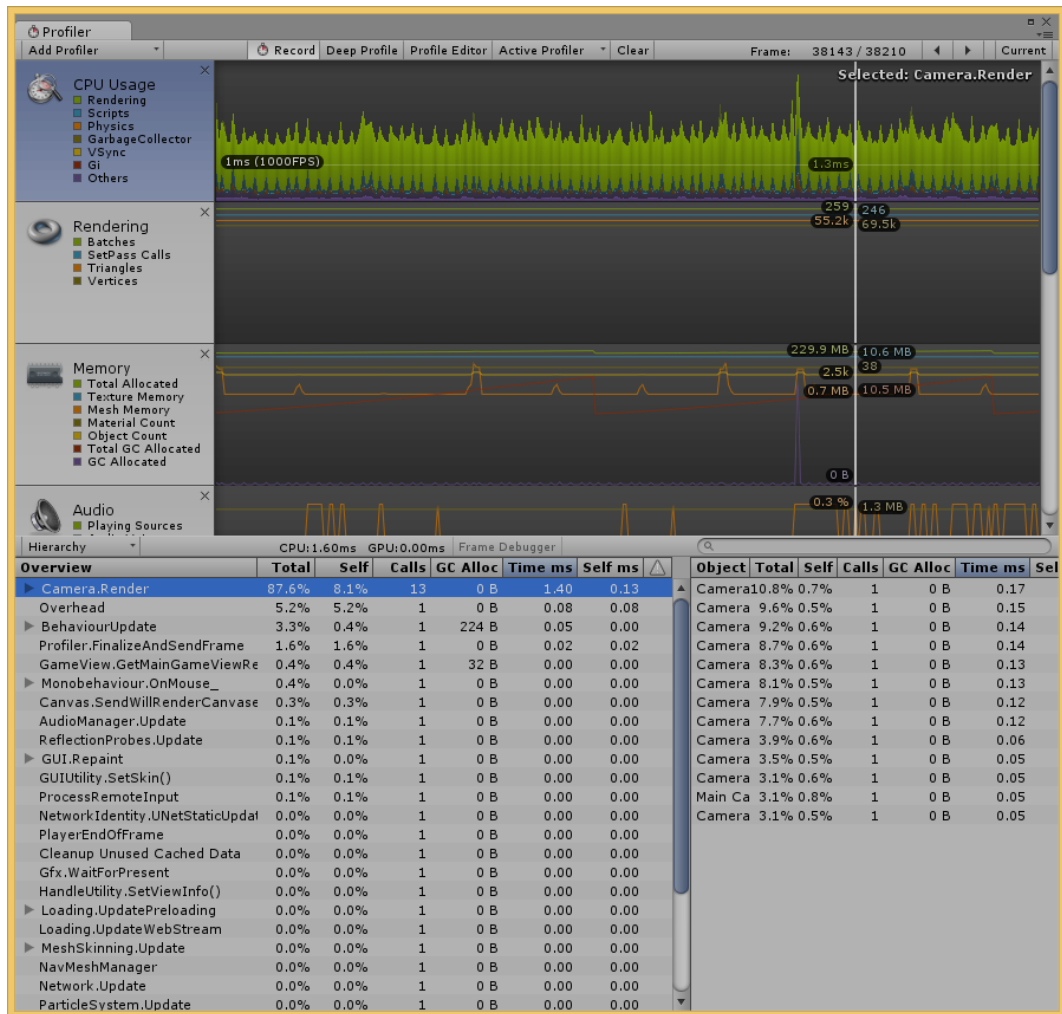


Abbildung 48: Performance analysieren mit Unity Profiler

Die folgenden Codeänderungen wurden im Zuge der Performanceanalyse durchgeführt.

- **Komponenten suchen**

Bei etlichen Modulen bestehen Verbindungen zu anderen Objekten. Es wurde sichergestellt, dass sämtliche Referenzen nicht mehrmals neu geholt werden, sondern alle Verknüpfungen in der Startphase gemacht werden.

- **Deaktivieren Rendering Hauptkamera**

Weil das Plugin eigene Kameras erstellt und die Darstellung der ursprünglichen Kamera obsolet macht, indem die neuen Viewports den alten Viewport verdecken, wird das Rendering dieser Kamera nach der Konfigurationsphase deaktiviert.

- **UI-Kameras orthografische Projektion**

Weil die Berechnung einer orthografischen im Vergleich zu einer perspektivischen Projektion einfacher ist, wird die Projektion für alle UI-Kameras auf orthografisch gesetzt. Dies ist möglich, weil die Perspektive keinen Einfluss auf die Darstellung der 2D-UI-Elemente hat.

- **FixedUpdate<sup>27</sup> statt Update**

Unity stellt eine Methode namens FixedUpdate() zur Verfügung, die im Vergleich zu Update() nicht so häufig wie möglich (also bei jedem berechneten Frame) ausgeführt wird, sondern nur zu fix definierten Zeitpunkten. Standardmässig sind das 30 Ausführungen pro Sekunde. Alle Berechnungen vom Wand, den Eyes und dem Viewfrustum werden in dieser FixedUpdate-Methode gemacht. Der Unterschied ist für den Benutzer nicht spürbar und es lassen sich somit etliche Berechnungen vermeiden.

- **Wand Raycast**

Der Wand führt in gewissen Zeitabständen Raycasts aus, um den Punkt auf dem virtuellen CAVE, in welche er ausgerichtet ist, zu bestimmen. Die Distanz, wie weit dieser Raycast in die virtuelle Welt geschossen werden soll, kann bestimmt werden. Im Falle der CAVE-Schnittpunktsuche ist eine maximale Distanz von den räumlichen Gegebenheiten bereits gegeben und muss nur so weit erfolgen.

- **Clipping Planes**

Wie weit, bzw. nah, sich die Clipping Planes befinden, wird von der Unity Applikation festgelegt und darf vom Plugin nicht beeinflusst werden. Jedoch für die UI-Kameras ist die maximale Distanz nur bis zum CAVE XXL und lässt sich entsprechend verkleinern, um eine bessere Performance zu erreichen.

<sup>27</sup> FixedUpdate Unity – <http://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html>



## 4.6 Warping<sup>28</sup>

Bei einer stereoskopischen Projektion mit zwei unterschiedlichen Beamern muss für eine vollständige Immersion die Kalibrierung dieser zwei Output Geräte sehr genau sein. Aufgrund verschiedener Umstände (Wärmeausdehnung, Freiheitsgrade durch Gelenke und Spiegel) wäre es wünschenswert, den gerenderten Output noch zu «warpen», um eine bessere Übereinstimmung beider Bilder auf einer Leinwand zu erreichen.

„Warping (von englisch warp = verformen, verzerren) von Bildern gehört in der Computergrafik zu den bildbasierten Techniken. Falls zu einem Bild die dazugehörigen Tiefenwerte existieren, ist es mittels der Warping-Gleichung möglich, das Bild von einem anderen Blickpunkt zu betrachten. Das Verfahren ist echtzeitfähig, bringt jedoch einige Artefakte wie beispielsweise Aufdeck- oder Verdeckungsfehler mit sich.“ (Warping, 2016)

In Unity ist dies durch einen Shader oder über die Projektionsmatrix der Kamera möglich. Da im CAVE die Projektionsmatrix bereits für die Frustumtransformation neu berechnet wird, empfiehlt sich hier die Verwendung des Shaders.

Im folgenden Beispiel ist ein einfacher Shader verwendet worden, um eine homographische Transformation der Kameraperspektive durchzuführen. (chiragraman, 2016)

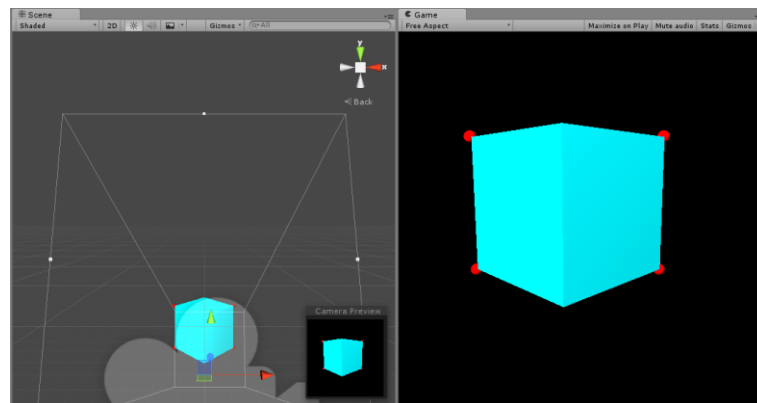


Abbildung 49: Initialposition Warpingbeispiel

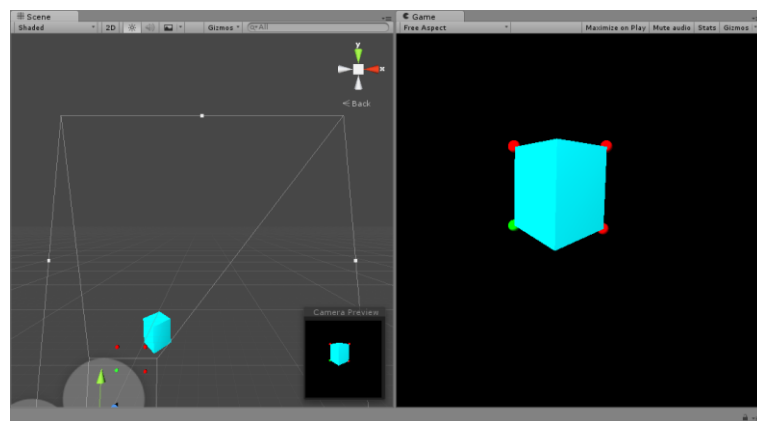


Abbildung 50: Verkleinerung Perspketive Warping

<sup>28</sup> Warping – [https://en.wikipedia.org/wiki/Image\\_warping](https://en.wikipedia.org/wiki/Image_warping)

## **Fazit**

Das Warping ist prinzipiell möglich und es empfiehlt sich, dies über einen Shader durchzuführen. Dadurch wird die Projektionsmatrix der Unity Kamera nicht mehr verändert und der Shader kann für sich selbst behandelt werden. Es ist möglich, bereits vorhandene GLSL<sup>29</sup>-Shader in Unity zu verwenden. Eine Manipulation der Projektionsmatrix kann dann u.U. ähnliche Verzerrungseffekte hervorrufen wie in Kapitel 4.2.2 erwähnt wird.

## **Offene Punkte, Abgrenzung**

Folgende Arbeiten wurden in diesem Thema durchgeführt:

- Recherche Warping Machbarkeit in Unity
- Prototyp Variante Shader (Fisheye von Unity)
- Prototyp Projektionsmatrix (chiragraman, 2016)

Adaptionen direkt im Plugin und adjustierte Shader wurden aus Zeit- und Komplexitätsgründen weggelassen. Hier empfiehlt es sich, die Verzerrungen zuerst zu erfassen, in einem zweiten Schritt diese durch Vektoren abzubilden um damit einen Shader aufzubauen. Weiter ist nicht bewiesen, ob die Verzerrungen sich während des Betriebes noch durch zusätzliche Temperatúrausdehnungen verändern. Falls dies der Fall wäre, müsste der Shader durch eine Nachkonfiguration während des Betriebes verändert werden.

<sup>29</sup> GLSL Shader in Unity – <http://docs.unity3d.com/Manual/SL-GLSLShaderPrograms.html>

## 5 Demo Apps

### 5.1 Shooting Gallery



Abbildung 51: Shooting Gallery Ingame

Das Setting dieses Demospiels ist eine Schiessbude im Wilder Westen Stil, wie sie auf einem Jahrmarkt anzutreffen ist. Die Galerien mit den abzuschliessenden Zielen verteilen sich rund um den Spieler. Mit Hilfe des Head Trackings kann sich der Spieler in der gesamten Szenerie umschauchen, Bewegungen ausführen und die Objekte aus verschiedenen Perspektiven betrachten. Das Wand-Device steuert das Gewehr, um die Zielobjekte anzuvisieren und abzuschliessen. Die Buttons des Wands werden gebraucht um das Gewehr abzufeuern.

- **Umgebung**

Die Landschaft ist ein Model aus dem Asset Store von Unity mit riesigen Ausmessungen. Um die abzuschliessenden Ziele platzieren zu können, Hindernisse zu schaffen und der Szenerie Leben einzuhauchen, wurden verschiedene Marktstände, Heukarren, Büsche und Zäune eingefügt.

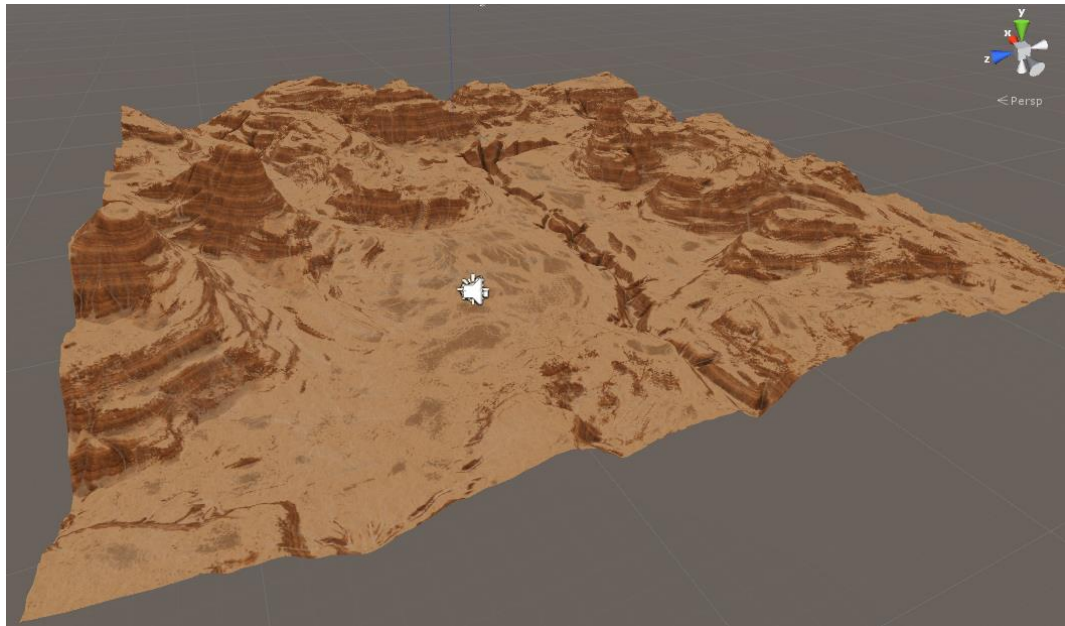


Abbildung 52: Desert Model, Quelle: Unity Asset Store

- **Gewehr**

Das Model des Gewehrs ist ebenfalls aus dem Unity Asset Store. Die Enten und Zielscheiben werden mit der Cursorposition anvisiert, welche durch das Unity Plugin vom Wand gesetzt wird.

**Rotation**

Das Gewehr dreht sich entsprechend dessen Position. Die Rotation wird zweierlei beeinflusst.

**a) Rotation der Eyes**

Basierend auf der Rotation der Eyes auf der y-Achse (Yaw) und der z-Achse (Roll) dreht sich auch das Gewehr im Spiel. Somit wird ermöglicht, dass sich der Benutzer des CAVEs drehen kann und das Gewehr immer in seine Blickrichtung zielt. Neigt er den Kopf leicht auf eine Seite, übernimmt die Flinte ebenfalls diese Manipulation rollt sich auf die Seite.

**b) Relativer Winkel zwischen Eyes und Wand**

Zusätzlich zu der Blickrichtung, welche mittels Eyes festgestellt wird, folgt das Gewehr der aktuellen Cursorposition. Dazu wird der relative Winkel zwischen dem Wand und den Eyes berechnet und darauf basierend erfolgt eine zusätzliche Rotation des Gewehrs.

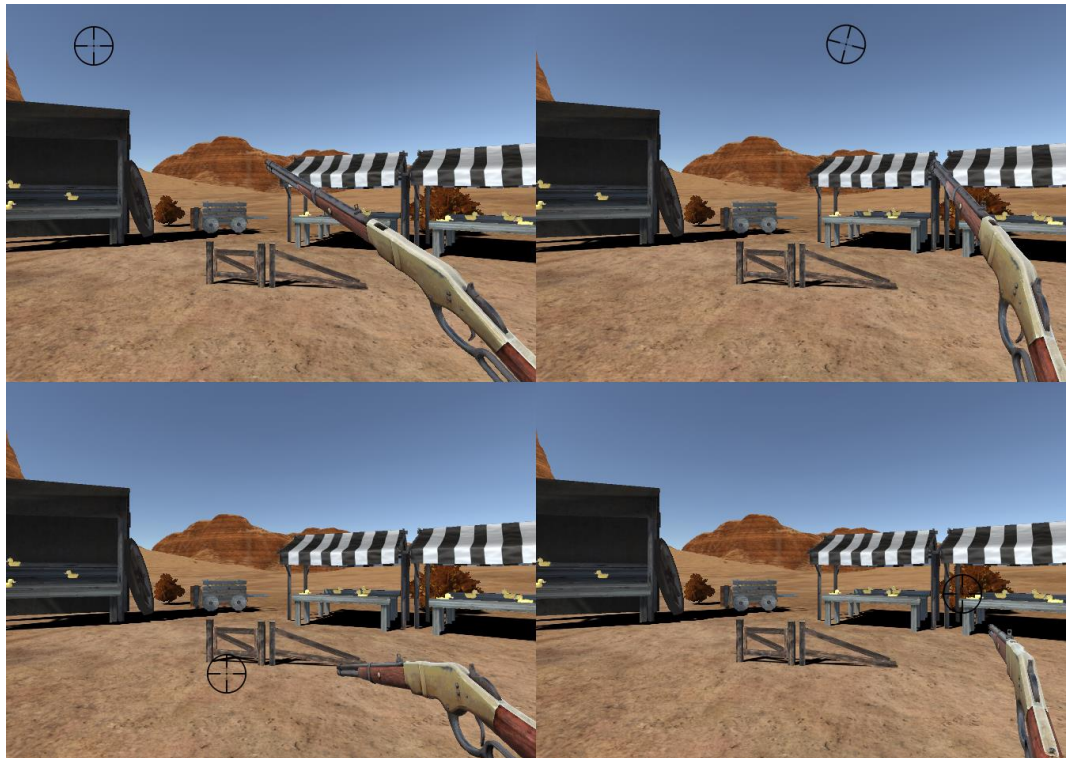


Abbildung 53: Shooting Gallery, Rotation des Gewehrs

### Schuss

Dort wo sich der Cursor auf der 2D-Ebene befindet, wird auch präzise der Schuss in der 3D-Welt auftreffen. Dazu wird ein Raycast mit der Ausrichtung der Kamera an der Position des Cursors in die Umgebung geschossen und geschaut, welches Objekt als erstes im Wege steht.

Beim Auftreffen dieses virtuellen Schusses wird dem Ziel mitgeteilt, ob nun eine Interaktion erfolgen soll oder nicht. Zusätzlich wird ein Partikeleffekt, welcher den Einschuss verdeutlicht, an der getroffenen Stelle erzeugt. Ein weiterer Raucheffekt wird bei der Flinte direkt gezeigt.



Abbildung 54: Shooting Gallery, Rauch

### Audio

Verschiedene Audiofiles wurden integriert, um das Spielerlebnis zu verbessern. Folgende Ereignisse provozieren einen Audioeffekt:

- Feuern des Gewehrs
- Treffen einer Ente
- Treffen der Zielscheibe

- **Zielscheibe**

Eines der beiden abzuschliessenden Ziele ist eine Holzkonstruktion mit drei Zielscheiben, welche sich zu zufälligen Zeitpunkten nach oben, bzw. nach unten klappen und somit angreifbar, bzw. nicht angreifbar werden.

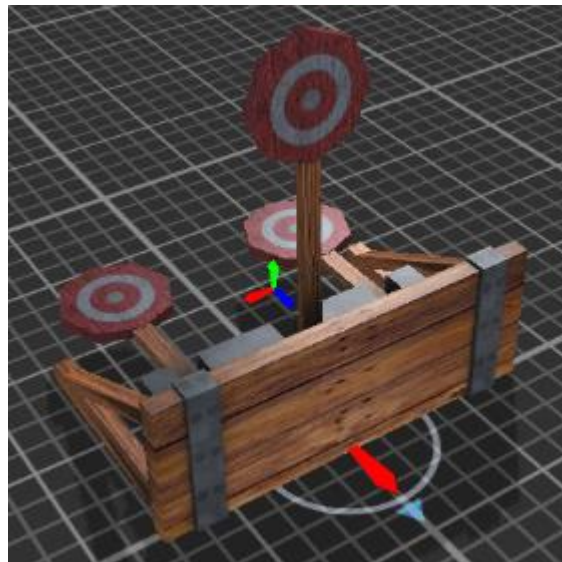


Abbildung 55: Shooting Gallery, Zielscheibe

Die Animationen sind im FBX<sup>30</sup>-Model gespeichert und werden mittels einem AnimationController ausgelöst. Zu zufälligen Zeitpunkten wird eine der Show-States aktiviert um die Animation abzuspielen. Trifft während einer gewissen Zeitspanne kein Schuss die Zielscheibe, aktiviert sich der Hide-State und geht anschliessend zurück in den Idle-State. Im Gegenzug, trifft der Spieler auf die Zielscheibe, wird die Animation beim Hit-State abgespielt und es werden Punkte gutgeschrieben. Zudem wird als Audiofeedback ein entsprechender Sound gehört. Anschliessend fängt die Sequenz von vorne an.

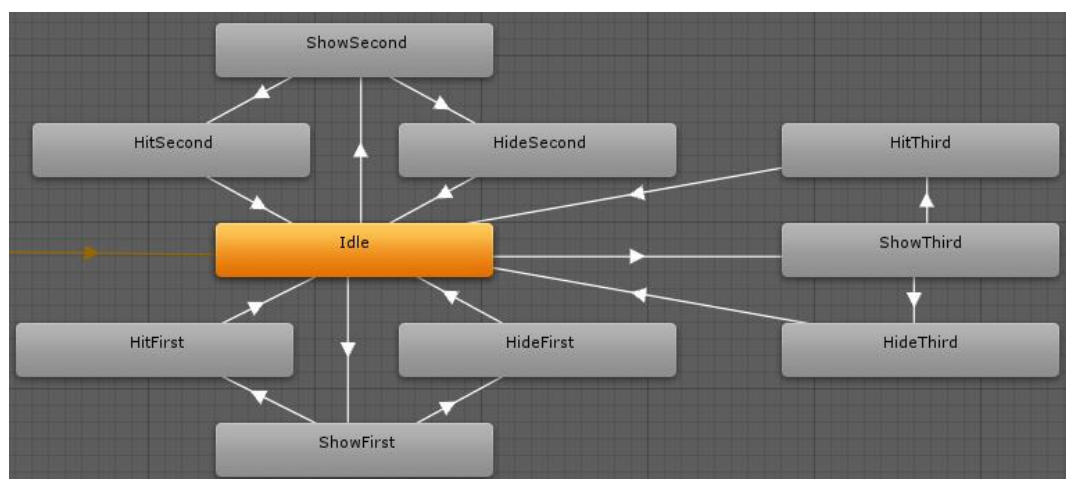


Abbildung 56: Shooting Gallery, Zielscheibe Animation Controller

<sup>30</sup> FBX – <https://en.wikipedia.org/wiki/FBX>



- **Ente**

Das zweite abzuschliessende Objekt ist eine Gummiente, die sich auf einer festgelegten Bahn mit zufälliger Geschwindigkeit nach vorne und hinten bewegt.



Abbildung 57: Shooting Gallery, Ente

Am Anfang und Ende der Bahn befinden sich jeweils Trigger, die ausgelöst werden, sobald sich das Mesh der Ente damit überschneidet. In diesem Moment ändert sich die Bewegungsrichtung der Ente und eine neue Geschwindigkeit wird zufällig zwischen einem definierten Bereich gewählt.

Bei einem Treffer wird der Winkel zwischen der aktuellen Kamera und der Ente berechnet und entsprechend ein Impuls auf die Ente angewandt, damit sie in die korrekte Richtung davonfliegt. Sobald die Ente keinen Kontakt mit der Oberfläche mehr hat auf der sie sich bewegt, wird ein Event losgeschickt um nach einer gewissen Zeitspanne die Position und Rotation wiederherzustellen und Punkte gutzuschreiben. Bei einem Treffen ist zudem ein Audiofeedback (Quaken) zu hören.

Weil das gefundene Model der Ente für die Anwendungszwecke viel zu detailliert war, musste aus Performancegründen mit Blender<sup>31</sup> eine Reduktion der Faces erfolgen. Von ehemals 7160 sind noch 1288 Faces übrig geblieben. Dank der Textur und der Distanz, die zwischen dem Spieler und der Ente liegt, fällt der Unterschied nicht auf.

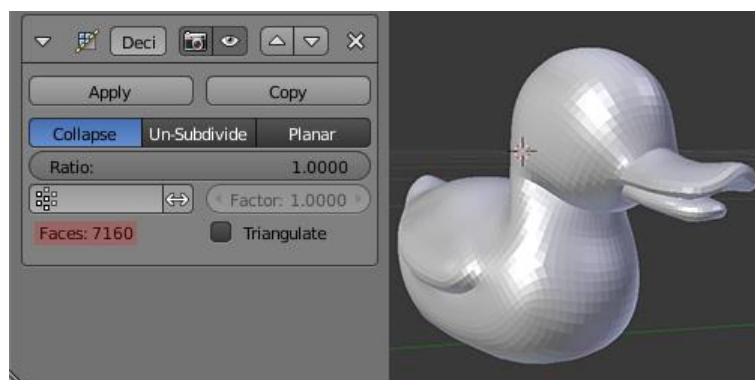


Abbildung 58: Shooting Gallery, Ente mit vielen Details

<sup>31</sup> Blender – <https://www.blender.org/>

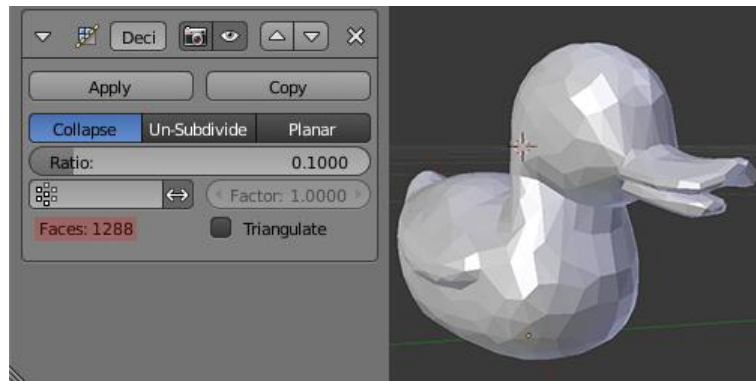


Abbildung 59: Shooting Gallery, Ente mit reduzierten Details

- **Punktesystem**

Nebst der bereits verstrichenen Zeit ist in einer Ecke als UI-Element die erspielte Punktzahl sichtbar. Die beiden abzuschliessenden Ziele (Ente und Zielscheibe) stellen öffentliche, statische Delegates (Events) zur Verfügung, mittels denen andere Objekte Methoden registrieren können. Wird eine beliebige Ente abgeschossen, wird die Methode aufgerufen. Die Klasse, welche für die Punkteberechnung und -darstellung zuständig ist, registriert entsprechend eine Methode, welche als Parameter die erspielten Punkte erhält. Somit kann dort sauber Buch geführt werden über den aktuellen Punktestand und ihn als GUI-Element anzeigen.

```
Duck.cs
public delegate void DelegateHit(int points);
public static event DelegateHit OnHit;

public void Hit()
{
    if (!_alreadyHit)
    {
        _alreadyHit = true;
        OnHit(POINTS);
    }
}

UIPoints.cs
Duck.OnHit += OnHit;

void OnHit(int points)
{
    _points += points;
    _text.text = "Punktzahl: " + _points.ToString();
}
```

Quellcode 11: Shooting Gallery, Ente und Punkte



## 5.2 Model-Viewer

Da der CAVE nicht nur zum Spielen gedacht ist, zeigt diese zweite Demo noch ein weiteres Anwendungsgebiet. Es handelt sich um mehrere Operationssäle, welche begangen werden können.

Die gesamte Umgebung ist statisch, einige Personen haben jedoch vordefinierte Animationen.

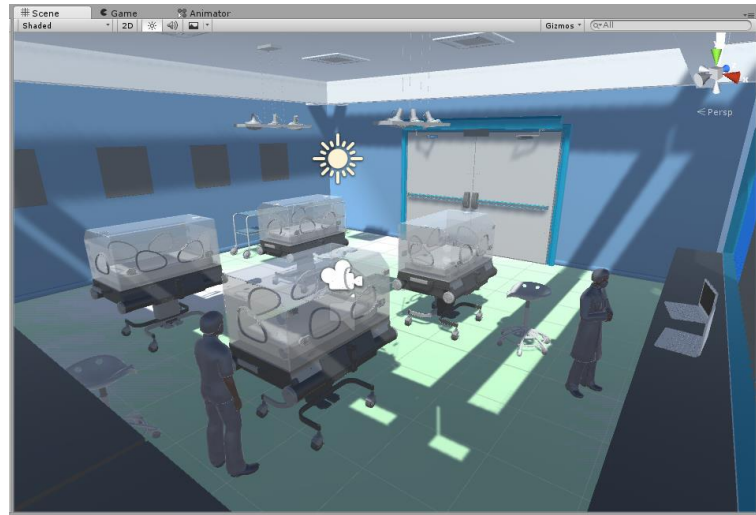


Abbildung 60: Model-Viewer OP Raum 1

- **Modelle**

Es werden Modelle von Dosch<sup>32</sup> verwendet, welche bereits im Besitz der BFH sind.

Es folgt eine Auflistung der verfügbaren Personen und Räume. Auf die Detailliste aller Equipments wird hier Verzichtet.

Person	Infos	
Surgeon	Person a1	19 Actions, 2 Sounds
SurgeonAssistant	Person a2	4 Actions
TOA	Person a3	6 Actions
TOA	Person a4	4 Actions
OPattendant	Person a5	8 Actions
Anesthetist	Person a6	7 Actions

Tabelle 10: Personen Modelviewer

<sup>32</sup> Dosch – [www.doschdesign.com](http://www.doschdesign.com)

Raum	Infos	
MedicalRoom01	CT Raum mit Kontrollraum	
MedicalRoom02	Zahnarzt	
MedicalRoom03	Zahnarzt schlicht	
MedicalRoom04	Bettenzimmer	
MedicalRoom05	Babystation	
MedicalRoom06	Zahnarzt mit Vorzimmer	
MedicalRoom07	Patientendoppelzimmer	
MedicalRoom08	Quadratisches Zahnarztzimmer	
MedicalRoom09	Psychiater	
MedicalRoom10	Fitnessstudio	

Tabelle 11: Räume Modelviewer

- **Szenen**

Für jeden verfügbaren Raum ist eine vordefinierte Szene erstellt, in welcher mit dem Gamepad navigiert werden kann.

- **Hauptmenu**

Die Szenen können über ein statisches Menu geladen werden.

### 5.3 App Drittpartei

Dieser Anwendungsfall soll aufzeigen, dass nicht nur konkret auf das umgesetzte Unity Plugin hin erarbeitete Applikationen problemlos verwendet werden können. Ausgewählt wurde das mit Unity 5 ausgelieferte Beispielprojekt namens „Standard Assets Example Project“.

Die Integration des Plugins verlief einwandfrei und das Spiel konnte erfolgreich in 3D im CAVE gespielt werden. Alle Elemente, zum Beispiel die Anzeige der UI Elemente auf einem spezifischen Screen, verursachen keinerlei Schwierigkeiten. Der nachfolgende Screenshot zeigt die Aufteilung der Viewports nach dem Aktivieren des Plugins.

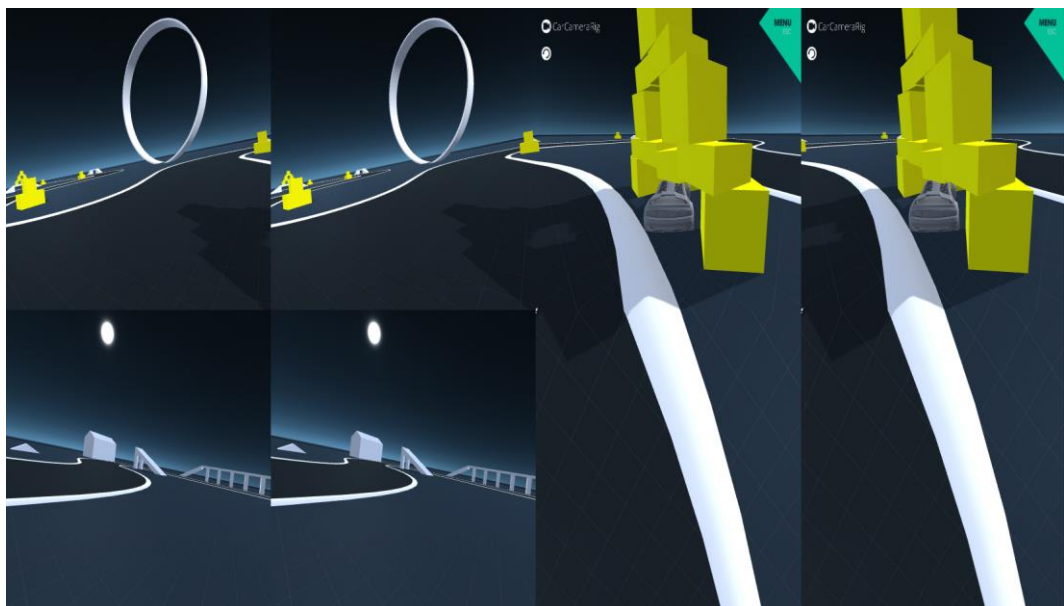


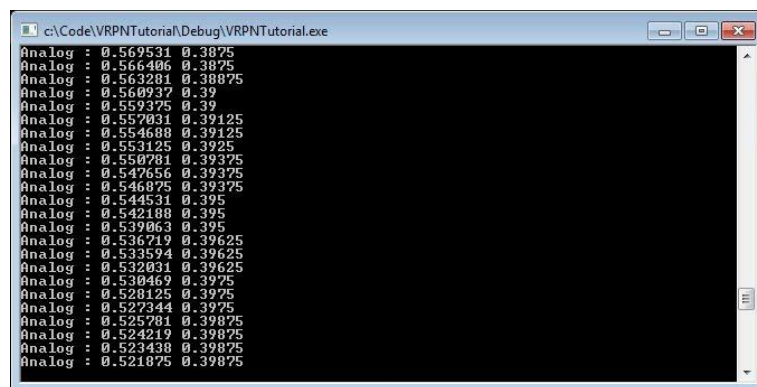
Abbildung 61: Standard Assets Example Project mit Unity Plugin

## 6 Testing

Dieses Kapitel beinhaltet die Tests der Prototypen, welche für verschiedene Module erstellt wurden, das Testen des Plugins mit der eigenen Testszene, sowie Performancetests der gesamten Anwendung.

### 6.1 PPT-Studio und VRPN

Die Funktionalität des internen VRPN Servers des PPT Studios kann mit einem einfachen Konsolencient überprüft werden, welcher die Trackerdaten ausgibt. Das Programm «vrpn\_print\_devices.exe» von vrpn.org liefert diese Informationen.



```
c:\Code\VRPNTutorial\Debug\VRPNTutorial.exe
Analog : 0.569531 0.3875
Analog : 0.566406 0.3875
Analog : 0.563281 0.38875
Analog : 0.560937 0.39
Analog : 0.557375 0.39
Analog : 0.557031 0.39125
Analog : 0.554688 0.39125
Analog : 0.553125 0.3925
Analog : 0.550781 0.39375
Analog : 0.547656 0.39375
Analog : 0.546875 0.39375
Analog : 0.544531 0.395
Analog : 0.542188 0.395
Analog : 0.539063 0.395
Analog : 0.536719 0.39625
Analog : 0.533594 0.39625
Analog : 0.532031 0.39625
Analog : 0.530469 0.3975
Analog : 0.528125 0.3975
Analog : 0.527344 0.3975
Analog : 0.525781 0.39875
Analog : 0.524219 0.39875
Analog : 0.523438 0.39875
Analog : 0.521075 0.39875
```

Abbildung 62: VRPN Debug Ausgabe – <http://www.cs.unc.edu/Research/vrpn/index.html>

### 6.2 Wrapping VRPN in C# und Unity

Die Funktionalität der kompilierten DLL des VRPN Projekts sowie die .net Wrapperklasse wurde Anhand eines kleinen Unity Testprojektes überprüft. Dies verwendete die erhaltenen Trackerdaten direkt als Positions- und Rotationsangaben der Kamera. Weiter ist eine Plane im Raum sichtbar, welche die Ausrichtung im Unity bekanntgibt, damit dies dann mit der realen Trackingposition überprüft werden kann.

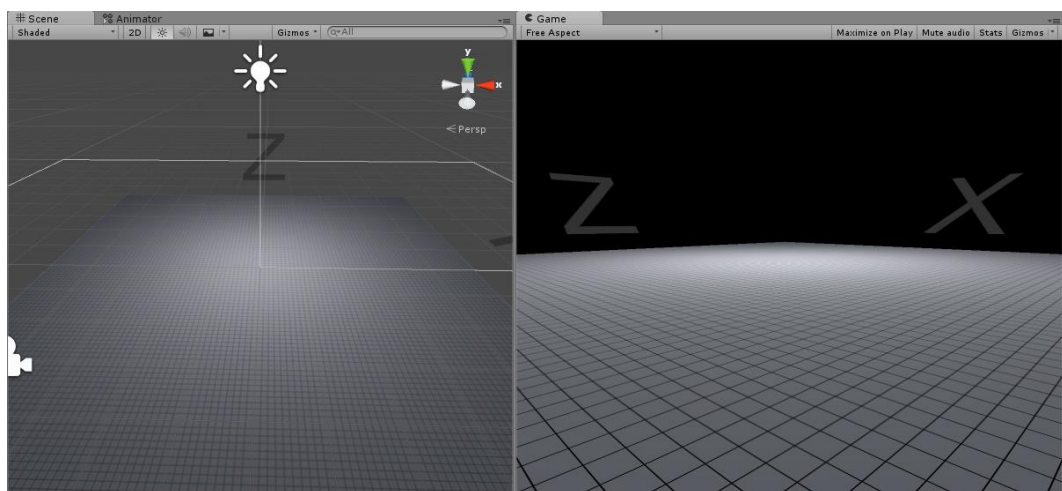


Abbildung 63: VRPN Prototyp

```

public class Tracker : MonoBehaviour
{
    public string host = "localhost";
    public string obj;
    public int channel = 0;

    public bool trackPosition = true;
    public bool trackRotation = true;

    // Update is called once per frame
    void Update()
    {
        if (trackPosition)
        {
            transform.position = VRPN.vrpnTrackerPos(obj + "@" + host, channel);
        }
        if (trackRotation)
        {
            transform.rotation = VRPN.vrpnTrackerQuat(obj + "@" + host, channel);
        }
    }
}

```

Quellcode 12: VRPN Prototyp

### 6.3 Unity Plugin Projekt als Debugger

Das Unity Plugin kommt mit einer Testszene, mit welcher folgende Elemente getestet werden können:

- Stereoskopie Einstellungen der Kameras
- Sekundäre Kameras
- GUI Elemente
- Mousecursor
- Verhalten der Frustumtransformation bei Bewegungen im CAVE.

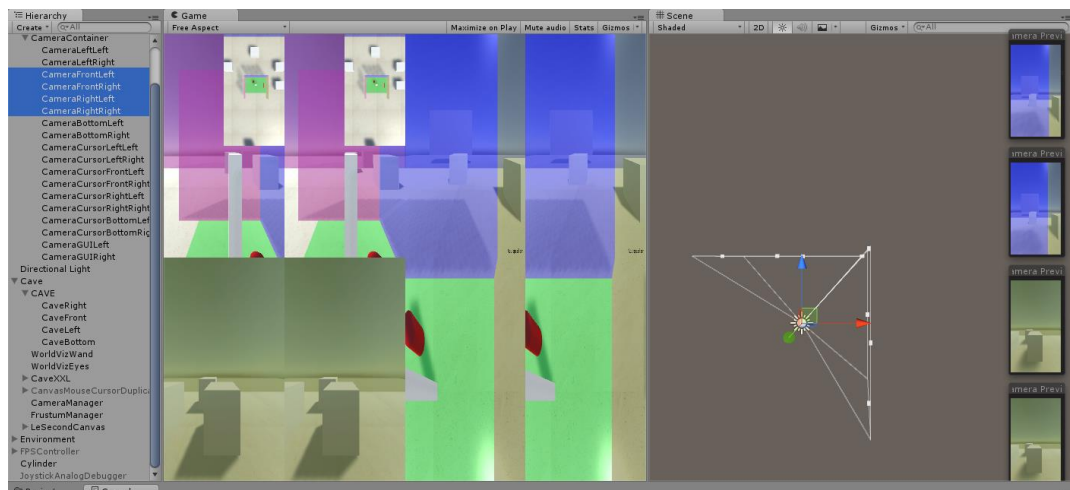


Abbildung 64: CAVE Testszene

Auf der obigen Abbildung sind die Frustums von vier Kameras sichtbar und ob die Aufspannung der Projektion korrekt ist. Eine sekundäre Kamera ist auf die linke Projektionsseite gemappt, so dass auch dieser Output direkt im Plugin überprüft werden kann. Zusätzlich können die Frustums noch über Gizmos<sup>33</sup> von Unity angezeigt werden.

Die Simulationen der Bewegung übernimmt ein Debugscript, welches bei Bedarf auf den Komponenten aktiviert werden kann. So führt die Komponente eine Bewegung im Raum durch zusammen mit einer Rotation.

```
void Update()
{
    TimeCounter += Time.deltaTime * speed;
    float x = Mathf.Abs(Mathf.Cos(TimeCounter) * width);
    float y = Mathf.Abs(Mathf.Sin(TimeCounter) * height);
    float z = 0.1f;

    transform.position = new Vector3(x, y, z);
    transform.rotation = new Quaternion(x, y, z, 1f);
}
```

Quellcode 13: Debugmover

## 6.4 Performance Tests Unity Server

Die Grafik Performance des Unity Servers ist von zentraler Bedeutung. Die Konfiguration des Unity Servers ist folgendermassen:

Auflösung: 5120 x 2048  
Qualität Unity: Fantastic (maximum)

Um eine aussagekräftige Analyse zu machen, wurden die Testapplikationen zweimal ausgeführt und analysiert. Im ersten Durchlauf ohne das Unity Plugin, in einem zweiten Durchlauf mit aktiviertem Unity Plugin. So ist die Einbusse der Performance klar ersichtlich.

- **Shooting Gallery**

Diese eigens entwickelte Applikation wurde als erstes getestet.

FPS ohne Plugin: 218  
FPS mit Plugin: 171

Verlust in %: 21.6 %

<sup>33</sup> Unity Gizmo – <http://docs.unity3d.com/ScriptReference/Gizmos.html>



Abbildung 65: Shootinggallery Demospiel ohne Plugin

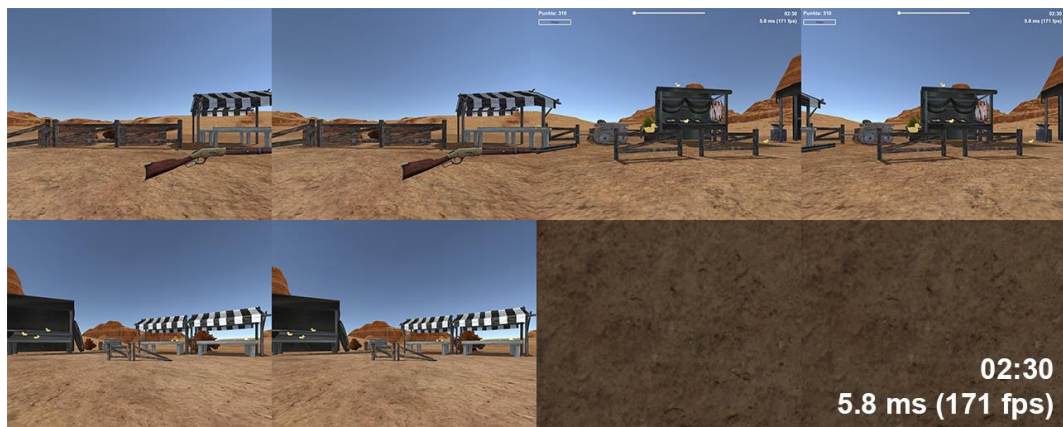


Abbildung 66: Shootinggallery Demospiel mit Plugin

- **Standard Assets Example Project**

Die von Unity gelieferte Applikation wurde ebenfalls getestet.

FPS ohne Plugin: 127

FPS mit Plugin: 92

Verlust in %: 27.7 %

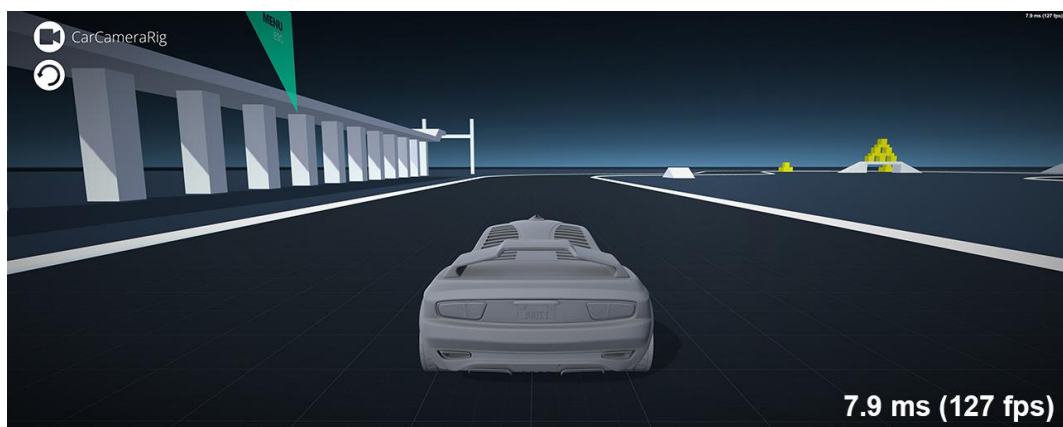


Abbildung 67: Unity Example Project ohne Plugin



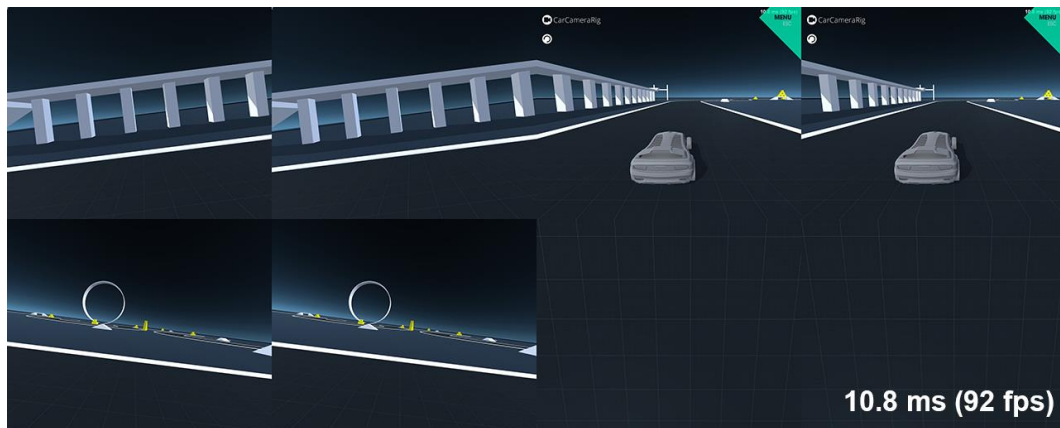


Abbildung 68: Unity Example Project mit Plugin

- **Bowling**

Das Bowling Spiel wurde im Rahmen des Moduls BT17534p Advanced Game Development für den CAVE umgesetzt.

FPS ohne Plugin: 92

FPS mit Plugin: 55

Verlust in %: 40.2 %



Abbildung 69: Bowlingdemo AdvGameDev ohne Plugin

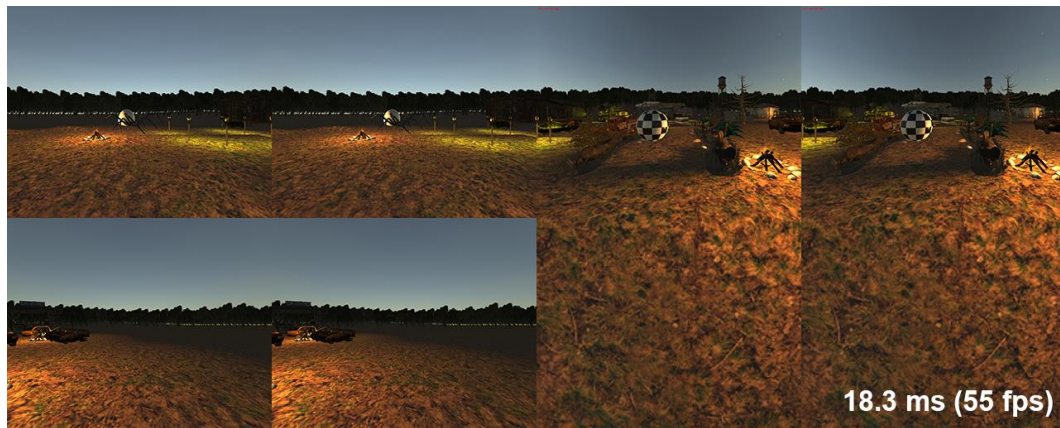


Abbildung 70: Bowlingdemo AdvGameDev mit Plugin

### Fazit

In Anbetracht dessen, dass sich der Viewport vervielfacht, ist eine Einbusse von durchschnittlich 29% tragbar. Statt einem Viewport müssen nun sieben weitere, plus eventuell zusätzliche GUI-Viewports berechnet werden. Je nach Applikationen ist der Verlust auch schwindend gering. Die starke Hardware im BFH CAVE kann diesen Verlust gut ausgleichen und die Grafikeinstellungen waren bei sämtlichen Tests auf das Maximum gestellt. Werden diese auf ein Mittelmaß runtergeschraubt, vergrößert sich die Framerate wiederum um ein x-faches. Keine Applikation wurde durch das Plugin unspielbar.



## 7 Projektmanagement

### 7.1 Aufgabenstellung

Die Aufgabenstellung lautet «Neben der bestehenden CAVE Cluster-Rendering Lösung soll ein Unity 3D Render-Server in Betrieb genommen werden, um der zunehmenden Bedeutung von Unity im CPVRLab und im Unterricht Rechnung zu tragen. Es ist dabei eine Lösung zu entwickeln, welche eine einfache Integration von neuen oder bestehenden Unity Applikationen in das Multi-Screen Rendering Setup des CAVEs ermöglicht»

Folgende Anforderungen sollen dabei umgesetzt werden:

- Entwicklung eines geeigneten Setups oder APIs für ein einfaches Unity Multi-Screen Rendering
- Integration des WorldWiz Tracking Systems für User Head Tracking und Kamerasteuerung
- Implementation einer Demo-Applikation welche die Features des Frameworks demonstriert
- Erstellen eines Entwickler-Handbuches mit detaillierte Anleitung und Tutorial

### 7.2 Zieldefinitionen

Durch die Ist-Soll Analyse werden die Ziele definiert.

Beschrieb	Ist	Soll
<b>Stereoskopisches Rendering von Unity im CAVE</b>	Der neue Unityserver ist noch unkonfiguriert. Keine Mosaiceinstellungen vorhanden und kein Plugin vorhanden.	Mosaic soll so konfiguriert werden, dass das Unity Plugin verwendet werden kann.
<b>Verwenden des WorldViz Trackings</b>	Die Hardware (Infrarotkameras sowie Wand und Eyes) sind vorhanden, der Trackingserver ist neu und unkonfiguriert.	Die vorhandene Hardware soll mit dem neuen Trackingserver verwendet und die Daten müssen über VPRN vom Unity Plugin empfangen werden können.
<b>Erstellen von Demo-Applikationen</b>	Keine Unity Demos vorhanden für den CAVE.	Demo-Applikationen programmieren für den CAVE.
<b>Erstellen eines Handbuches mit Tutorial</b>	Kein Unity Handbuch für den CAVE (zusammen mit dem Plugin) vorhanden.	Handbuch, Tutorial und weitere Hilfe sollen angeboten werden.

Tabelle 12: Ist-Soll Analyse

Somit ergeben sich folgende primären Ziele, welche **im Rahmen der Bachelorthesis erreicht wurden**:

- Entwicklung eines Unity Plugins, welches das Multi-Screen Stereo Rendering, das Tracking, sowie die Berechnung der Frustums übernimmt.
- Abdeckung Spezialfälle wie sekundäre Kameras, Darstellung GUI, benutzerdefinierter Cursor usw.
- Verwenden der VR-Inputdevices (PPT Wand / Eyes) mittels einem neuen Interface im Unity Plugin.
- Demoapplikation (ShootingGallery und ModelViewer) programmiert und im CAVE vorhanden.
- Handbuch mit einer Schritt-für-Schritt Anleitung zur Verwendung des Unity Plugins erstellt.

## 7.2.1 Funktionale Anforderungen

Durch die Zieldefinition ergeben sich folgende funktionale Anforderungen, welche erreicht wurden:

	Prio.	Keyfeatures	Kritische Punkte	Ziel erreicht
Adaption Unity Anwendung für den CAVE	1	<ul style="list-style-type: none"> <li>– Automatische Erstellung der zusätzlichen Unitykamas</li> <li>– Rotation der Unitykamas</li> <li>– Justierung des Frustums und FoV</li> </ul>	<ul style="list-style-type: none"> <li>– Optimale Frustums, welches die genaue Position im CAVE als Unitykamera übernehmen.</li> </ul>	100% erreicht
VRPN Unterstützung	1	<ul style="list-style-type: none"> <li>– Implementierung des Protokolls für VR-Eingabegeräte</li> </ul>	<ul style="list-style-type: none"> <li>– VRPN Server von WorldViz ist eine Blackbox (Gerätenamen)</li> </ul>	100% erreicht
Kompatibilität	2	<ul style="list-style-type: none"> <li>– Aktuelle Unityversion und zukünftige sollen supportet werden</li> </ul>	<ul style="list-style-type: none"> <li>– Abhängig von Unity</li> </ul>	100% erreicht (Version 5.2.3)
Tracking	1	<ul style="list-style-type: none"> <li>– Erkennung der Positionen und Rotationen von Eyes und Wand im CAVE</li> </ul>	<ul style="list-style-type: none"> <li>– Abhängig von den vorhandenen Kameras, sowie deren Bildqualität</li> </ul>	100% erreicht
Einstellungsmöglichkeiten	2	<ul style="list-style-type: none"> <li>– Alle Freiheitsgrade müssen pro Anwendung einstellbar oder direkt sperrbar sein</li> </ul>		100% erreicht
Setup, Dokumentation	2	<ul style="list-style-type: none"> <li>– Plug &amp; Play für Unityanwendungen</li> </ul>	<ul style="list-style-type: none"> <li>– Spezielle Anwendungen können eventuell nicht durchgängig unterstützt werden</li> </ul>	100% erreicht
Demo-Applikationen	3	<ul style="list-style-type: none"> <li>– Siehe Tabelle 2 (Anhang Pflichtenheft)</li> </ul>		100% erreicht
Unity Output Warping	4	<ul style="list-style-type: none"> <li>– Das Ausgabebild soll auf die Eigenheiten des CAVEs angepasst werden können</li> </ul>	<ul style="list-style-type: none"> <li>– Optionales Feature</li> </ul>	Abklärungen getroffen und Prototypen getestet

Tabelle 13: Funktionale Anforderungen

## 7.3 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.

### 7.3.1 Projektteam

Daniel Inversini [daniel.inversini@students.bfh.ch](mailto:daniel.inversini@students.bfh.ch)

Julien Villiger [julien.villiger@students.bfh.ch](mailto:julien.villiger@students.bfh.ch)

### 7.3.2 Betreuer

Prof. Urs Künzler [urs.kuenzler@bfh.ch](mailto:urs.kuenzler@bfh.ch)

### 7.3.3 Experte

Dr. Harald Studer [harald.studer@iss-ag.ch](mailto:harald.studer@iss-ag.ch)

## 7.4 Projektplanung, Meilensteine

Der Gantt<sup>34</sup>-Projektplan ist unter <https://pm.ti.bfh.ch/projects/unity-cave-thesis/issues/gantt> erreichbar.

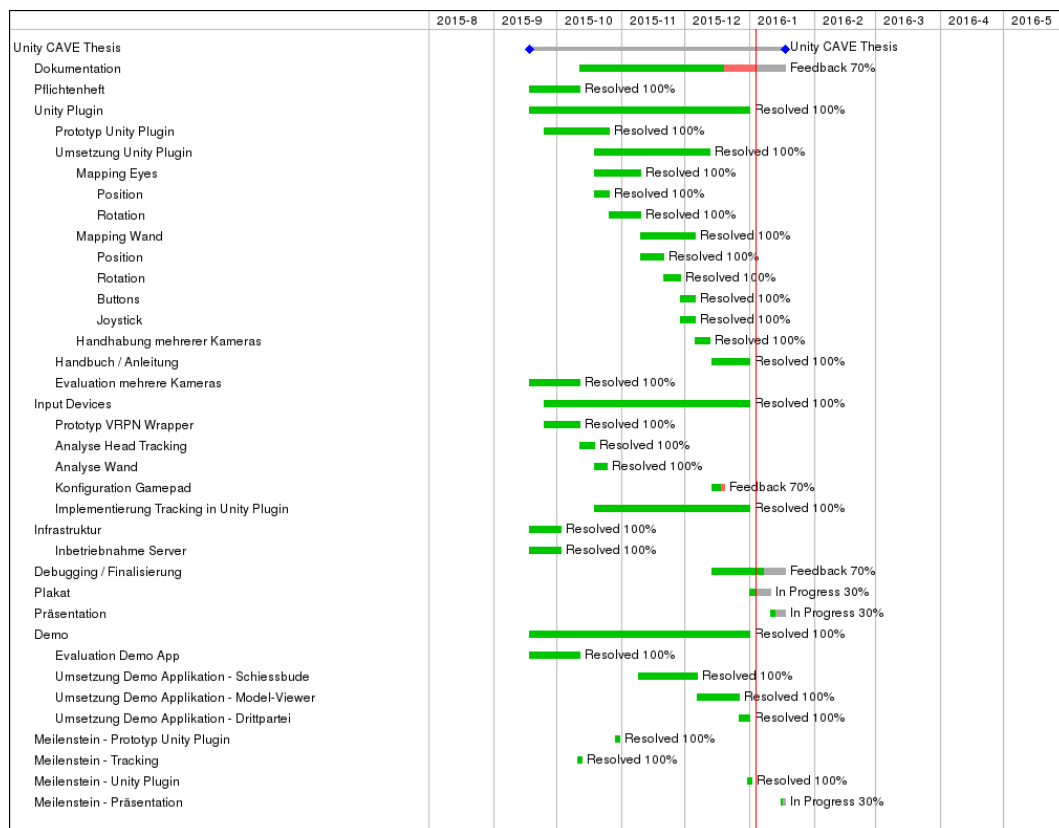


Abbildung 71: Gantt Plan

<sup>34</sup> Gantt – <http://de.wikipedia.org/wiki/Gantt-Diagramm>

Folgende Meilensteine wurden in der Voranalyse und im Pflichtenheft bereits festgelegt:

Bezeichnung	Fälligkeit
Prototyp Unity Plugin	29.10.2015
Tracking	29.10.2015
Unity Plugin / Handbuch / Dokumentation	31.12.2015
Präsentation	17.01.2015

Tabelle 14: Meilensteine

- **Prototyp Unity Plugin**  
Eine erste Implementierung des Unity Plugins mit grundlegender Funktionalität wurde umgesetzt.
- **Tracking**  
Die Analysephase des VRPN Protokolls wurde abgeschlossen, damit die Integration des WorldViz Tracking Systems in das Unity Plugin erfolgen kann. Zusätzlich wird die Umsetzung einer Demo-Applikation gestartet.
- **Unity Plugin / Handbuch / Dokumentation**  
Das Unity Plugin wurde fertiggestellt und getestet. Kleinere Anpassungen und die Finalisierung erfolgen noch. Zusätzlich wurden ein Handbuch und eine Anleitung erstellt, um die Verwendung des Plugins zu vereinfachen. Die Dokumentation ist an dieser Stelle ebenfalls möglichst weit fertiggestellt.
- **Präsentation**  
Sämtliche Dokumente und Arbeiten sind abgeschlossen und eine Präsentation wurde erstellt.

## 7.5 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, bei denen Fortschritte und Probleme diskutiert werden können.

## 7.6 Risikoanalyse

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

A: Auswirkungen W: Wahrscheinlichkeit

Risiko	Beschreibung	Massnahmen	A	W
Neue Infrastruktur nicht lauffähig	Der Unity Render Server sowie der Trackingserver für WorldViz wurden erneuert, jedoch noch nicht konfiguriert und getestet.	Intensives Behandeln der WorldViz Trackingsoftware zu Beginn der Bachelorarbeit um Probleme frühzeitig zu erkennen.	Gross	Mittel
Fehlerhafte Hardware im Bereich Trackingsystem	Defekte oder störende Kameras / Devices können die Trackingpositionen beeinflussen oder verunmöglichen.	Neukalibrierung der Kameras und das Erkennen von defekten Geräten, eventuelles Austauschen der Komponenten.	Gross	Mittel
Softwareprobleme	Die vorgesehenen Softwares (Mosaic) könnten u.U. Probleme bereiten oder nicht alle gewünschten Features unterstützen.	Frühes Testen mit Prototypen, Spezialfälle eruieren, Alternativen suchen.	Mittel	Mittel
Performance	Die Leistung der Hardware ist nicht ausreichend oder das Plugin ist zu ressourcenintensiv.	Regelmässige Tests und Optimierungen des Plugins mittels Profiler.	Mittel	Klein
VRPN	Die Daten können in Unity nicht ausgelesen oder interpretiert werden. Die Netzwerkkapazität ist nicht ausreichend.	In einer frühen Phase das Auslesen und Interpretieren sicherstellen, ev. auf externe Libraries zugreifen oder Beispiele zu Hilfe nehmen. Frequenz der ausgelieferten Daten reduzieren.	Mittel	Mittel

Tabelle 15: Risikoanalyse

## 8 Fazit

Hervorzuheben ist die dankbare Unity Plattform. Durch unser Vorwissen, dem Besuch der Module „BTI7527a Game Development“ und „BTI7534p Advanced Game Development“ sowie guten Kenntnissen in C# konnte rasch und ohne grosse Einarbeitung entwickelt werden. Viele komplexe Berechnungen, beispielsweise die Ermittlung des Schnittpunktes auf einer Ebene, sind bereits in Unity integriert und können dank einem umfassenden API ausgelagert werden und erfordern keine eigene, mathematische Implementierung.

Leider sorgte das Trackingsystem von WorldViz regelmässig für unangenehme Überraschungen. Die Bestimmung der Position und Rotation ist häufig unzuverlässig und ein Device war eine Zeit lang sogar defekt und wurde dank der guten Zusammenarbeit mit dem CPVR-Lab rasch repariert. Trotzdem beeinträchtigt die Ungenauigkeit dieses Systems die verknüpften Applikationen.

Die vorgängig detaillierte Planung konnte uns vor Engpässen bewahren und dank dem bereits zu Beginn grossen Einsatz eine Zuspitzung der Arbeitsleistung am Ende des Semesters verhindern. Der grosse Vorteil einer Partnerarbeit ist die stetige Kommunikation. Dies fördert die Kreativität bei der Lösungserarbeitung eines Problems und verhindert den Gang in eine gedankliche Sackgasse.

Sehr Motivierend ist die Tatsache, dass wir grosses Potenzial in der zukünftigen Verwendung des Plugins sehen und den CAVE der BFH beleben wird. Die beiden Unity Module sorgen in Zukunft für eine grosse Bandbreite an geeigneten Unity Applikationen, die im CAVE Verwendung finden könnten.

Ein grosser Lerneffekt wurde in der Verwendung von Unity erzielt. Durch die intensive Nutzung vieler Komponenten verfestigte sich der Umgang mit der Spielentwicklungsplattform erneut. Weiter war die Anwendung des Trackingsystems, also die Interaktion mit Soft- und Hardware, eine gute Erfahrung und zeigte dessen Schwierigkeiten auf, die es zu meistern galt.

Basierend auf dem umgesetzten Plugin sind weitere Arbeiten denkbar. Dazu gehört die Umsetzung des Warpings, also eine Behebung der Bildverzerrung durch die Projektoren. Möglich ist auch eine Erweiterung der unterstützten Input-Devices, beispielsweise des bereits vorhandenen Trackingballs. Damit das Plugin auch weiterhin problemfrei funktioniert, müssen möglicherweise bei neuen Unity-Versionen Adaptionen gemacht werden.

Im Allgemeinen wurde das Produkt zielstrebig umgesetzt und bietet einen Mehrwert für den CAVE der BFH.

## 9 Abbildungsverzeichnis

Abbildung 1: Vergleich Varianten Projekt 2	5
Abbildung 2: Prototyp 1 Projekt 2	7
Abbildung 3: Prototyp 2 Projekt 2	7
Abbildung 4: CAVE BFH	8
Abbildung 5: Infrastruktur CAVE	9
Abbildung 6: PPT Eyes, Quelle: <a href="http://www.worldviz.com">www.worldviz.com</a>	10
Abbildung 7: PPT Wand, Quelle: <a href="http://www.worldviz.com">www.worldviz.com</a>	11
Abbildung 8: Gamepad, Quelle: <a href="http://www.androidrundown.com">www.androidrundown.com</a>	11
Abbildung 9: Übersicht der Komponenten	14
Abbildung 10: Paketdiagramm simplifiziert	15
Abbildung 11: Sequenzdiagramm	17
Abbildung 12: Einstellungen Augendistanz	18
Abbildung 13: Custom Cursor	19
Abbildung 14: Minimap	20
Abbildung 15: Sekundäre Kameras Einstellungen	20
Abbildung 16: Mosaic Setting Unity Server	22
Abbildung 17: Frustum, Quelle: <a href="http://www.stackoverflow.com">www.stackoverflow.com</a>	23
Abbildung 18: Unity Plugin Frustum 1	24
Abbildung 19: Unity Plugin Frustum 2	24
Abbildung 20: Vergleich FoV 120 & 60 Grad	25
Abbildung 21: Unity Plugin Frustum GGM	25
Abbildung 22: Frustum Mode	26
Abbildung 23: Frustum CAVE XXL FoV Anpassung	27
Abbildung 24: Unity Plugin, virtueller CAVE	28
Abbildung 25: Virtueller Wand und Eyes	29
Abbildung 26: Unity Plugin, Schnittpunkt Wand / CAVE	30
Abbildung 27: Zuordnung Schnittpunkt CAVE und Unity Server	31
Abbildung 28: Wand Button Zuordnung	31
Abbildung 29: Inputmanager Unity	34
Abbildung 30: Beispielverbindung VRPN, Quelle WorldViz Dokumentation	35
Abbildung 31: VRPN Wrapper C#	35
Abbildung 32: 1€ Smoothing Vergleich	37
Abbildung 33: Unity Plugin, Settings Wand	38
Abbildung 34: Unity Plugin, Settings Eyes	39
Abbildung 35: Unity Plugin, Settings Sekundäre Kameras	40
Abbildung 36: Unity Plugin, Settings Cave	40
Abbildung 37: Unity Plugin, Settings System	41
Abbildung 38: Unity Plugin, Klassendiagramm	42

Abbildung 39: Unity Plugin, Export Package	49
Abbildung 40: Unity Plugin, Export Package Selection	49
Abbildung 41: Kopieren des Unity Plugins	50
Abbildung 42: Unity Plugin, Import Package	50
Abbildung 43: Hinzufügen Cave Prefab	51
Abbildung 44: Player Settings, Api Compatibility Level	52
Abbildung 45: Platform Settings	53
Abbildung 46: Zuweisung Prefabs	53
Abbildung 47: Export 64bit	54
Abbildung 48: Performance analysieren mit Unity Profiler	55
Abbildung 49: Initialposition Warpingbeispiel	57
Abbildung 50: Verkleinerung Perspektive Warping	57
Abbildung 51: Shooting Gallery Ingame	59
Abbildung 52: Desert Model, Quelle: Unity Asset Store	60
Abbildung 53: Shooting Gallery, Rotation des Gewehrs	61
Abbildung 54: Shooting Gallery, Rauch	61
Abbildung 55: Shooting Gallery, Zielscheibe	62
Abbildung 56: Shooting Gallery, Zielscheibe Animation Controller	62
Abbildung 57: Shooting Gallery, Ente	63
Abbildung 58: Shooting Gallery, Ente mit vielen Details	63
Abbildung 59: Shooting Gallery, Ente mit reduzierten Details	64
Abbildung 60: Model-Viewer OP Raum 1	65
Abbildung 61: Standard Assets Example Project mit Unity Plugin	66
Abbildung 62: VRPN Debug Ausgabe – <a href="http://www.cs.unc.edu/Research/vrpn/index.html">http://www.cs.unc.edu/Research/vrpn/index.html</a>	67
Abbildung 63: VRPN Prototyp	67
Abbildung 64: CAVE Testszene	68
Abbildung 65: Shootinggallery Demospiel ohne Plugin	70
Abbildung 66: Shootinggallery Demospiel mit Plugin	70
Abbildung 67: Unity Example Project ohne Plugin	70
Abbildung 68: Unity Example Project mit Plugin	71
Abbildung 69: Bowlingdemo AdvGameDev ohne Plugin	71
Abbildung 70: Bowlingdemo AdvGameDev mit Plugin	72
Abbildung 71: Gantt Plan	75



## 10 Tabellenverzeichnis

Tabelle 1: Mosaic Setting schematisch	21
Tabelle 2: CaveMain Methoden	43
Tabelle 3: CaveMain Enums	43
Tabelle 4: Eyes Methoden	44
Tabelle 5: Wand Methoden	45
Tabelle 6: CameraManager Methoden	46
Tabelle 7: CameraManager Enums	46
Tabelle 8: CameraContainer Methoden	46
Tabelle 9: FrustumManager Methoden	47
Tabelle 10: Personen Modelviewer	65
Tabelle 11: Räume Modelviewer	66
Tabelle 12: Ist-Soll Analyse	73
Tabelle 13: Funktionale Anforderungen	74
Tabelle 14: Meilensteine	76
Tabelle 15: Risikoanalyse	77

## 11 Quellcodeverzeichnis

Quellcode 1: Kamerainstanziierung	19
Quellcode 2: Zugriff über das API	29
Quellcode 3: Raycast	30
Quellcode 4: Unity Tastaturabfrage	32
Quellcode 5: Joystick Handling	33
Quellcode 6: VRPN-Positionshandling	36
Quellcode 7: Einstellungsänderung über API	41
Quellcode 8: CameraManager DataStruct	45
Quellcode 9: API	48
Quellcode 10: Zugriff über das API	48
Quellcode 11: Shooting Gallery, Ente und Punkte	64
Quellcode 12: VRPN Prototyp	68
Quellcode 13: Debugmover	69

## 12 Glossar

### I

**1€ Filter** 36

Smoothingfilter um die Positionen von VRPN zu glätten.

### A

**AnimationController** 62

Ist eine Zustandsmaschine welche die zu spielende Animation auf einem Objekt bestimmt sowie Übergängen zwischen Animationen ermöglicht.

**API** 21, 29, 32, 33, 41, 43, 44, 47, 48, 52

Eine Programmierschnittstelle, genauer Schnittstelle zur Anwendungsprogrammierung, häufig nur kurz API, ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.

**Asset Store** 48, 59, 60

Der Unity Asset Store ist eine Sammlung von frei verfügbaren wie auch kommerziellen Assets (Plugins) für Unity. Es können Scripts, Models, Texturen, Sound, etc. sein. Alles was Unity unterstützt.

### B

**BFH** 1, 2, 8, 28, 65

Berner Fachhochschule BFH  
Die Institution des Studiums.

**Blender** 63

Blender ist eine freie, mit der GPL lizenzierte 3D-Grafiksoftware. Sie enthält Funktionen, um

dreidimensionale Körper zu modellieren, sie zu texturieren, zu animieren und zu rendern.

### C

**CameraContainer** 18, 41, 43, 46, 53

Das plugineigene Konstrukt um die Kameras zu verwenden.

**CAVE** 1, 2, 5, 8, 9, 12, 13, 16, 19, 23, 24, 25, 26, 27, 28, 30, 31, 38, 40, 43, 45, 47, 56, 57, 65, 66, 68, 71, 72, 73

Bezeichnet den Raum, der verwendet wird um die dreidimensionale Illusionswelt zu erschaffen.

**Chromium** 5

**Chromium Rendering System**  
Bezeichnet ein System, welches fähig ist, interaktives Rendering auf mehreren Clusters durchzuführen.

**Cluster** 73

Ein Rechnerverbund oder Computercluster, meist einfach Cluster genannt, bezeichnet eine Anzahl von vernetzten Computern.

**CMake** 35

CMake (cross-platform make) ist ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.

**Coroutinen** 32

Coroutinen sind eine Verallgemeinerung des Konzepts einer Prozedur oder Funktion. Der prinzipielle Unterschied zwischen Coroutinen und Prozeduren ist, dass Coroutinen ihren Ablauf unterbrechen und später wieder aufnehmen können, wobei sie ihren Status beibehalten.

Cursor 18, 19, 31, 39, 44, 45, 46, 61

Der Mausfeil unter Windows.

## ***D***

Delegates 32, 45, 64

Objektorientierte Variante von Methodenzeiger unter C#.

DLL 35, 67

Auch oft als dynamic linked Library verwendet, enthält Programmcode, Daten und Ressourcen für eine Anwendung (EXE).

## ***E***

Equalizer 5

Equalizergraphics Parallel Rendering Middleware-Software, um OpenGL basierende Software auf mehrere Nodes, Graphikkarten und Prozessoren zu verteilen.

Eyes 10, 13, 16, 28, 29, 33, 36, 39, 40, 43, 44, 46, 47, 56, 60, 73

WorldViz Eyes

Brillengestütztes Trackinggerät, welches mit Infrarot LEDs ausgerüstet ist.

## ***F***

FBX 62

Dateiformat für 3D Modelle von Autodesk.

Field of View 24

Sichtfeld des Blickes, wie viele Grad sichtbar sind.

FixedUpdate 56

Unityfunktion, welche ein Update zu bestimmten Zeitpunkten durchführt und nicht jedes Frame.

FoV *Siehe Field of View*

Frustum 23, 24, 25, 26, 27, 43, 47

Frustum ist in der Computergrafik die mathematische Abbildung eines 3D-Universums auf den Bildschirm.

## ***G***

Git 35

Git ist eine freie Software zur verteilten Versionsverwaltung von Dateien.

Gizmo 69

Visuelles Hilfsmittel von Unity, um in der Szenenview Objekte anzuzeigen.

GPU 6

Grafikprozessor, ist ein auf die Berechnung von Grafiken spezialisierter und optimierter Prozessor für Computer, Spielkonsolen und Smartphones.

GUI 2, 18, 19, 26, 31, 40, 46, 64, 68

Grafische Benutzeroberfläche GUI bezeichnet eine Form von Benutzerschnittstelle eines Computers. Sie hat die Aufgabe, Anwendungssoftware auf einem Rechner

mittels grafischer Symbole, Steuerelemente oder auch Widgets genannt, bedienbar zu machen.

Gyrometer 10, 33

Ein Gyrometer registriert beziehungsweise misst Drehbewegungen.

## **H**

Hauptkamera 18, 19, 28, 40, 41, 43, 46, 55

Das Objekt in Unity, welche den standartmässigen Videooutput rendert.

## **I**

immersive *Siehe Immersion*

InputSimulator 32

C# Interface zum Simulieren von Tastatur- und Mauseingaben über den Win32 SendInput Befehl.

## **K**

KeyCode 32

Definition der Tastaturtaste, Moustaste, und weiteren Eingabegeräten (Mausrad etc).

## **L**

Lowpass 36

Filter, welcher die Signalanteile mit Frequenzen unterhalb ihrer Grenzfrequenz annähernd ungeschwächt passieren lässt, Anteile mit höheren Frequenzen dagegen dämpft.

## **M**

Mesh 63

Untereinander mit Kanten verbundene Punkte bilden in der Computergrafik ein Mesh, Polygonnetz, das die Gestalt eines Polyeders definiert.

MiddleVR 5, 77

MiddleVR for Unity  
Kommerzielle Applikation, welche die Erstellung von VR Anwendungen unterstützt. Verschiedene Soft- und Hardware werden unterstützt und verbindet sie untereinander,

Minimap 19, 20

Stellt einen Überblick über die Spielwelt dar. Abstrahiert und simplifiziert die Details, sodass nur wichtige Akteure (Spieler, Quests, Gelände) angezeigt werden.

Mosaic 6, 11, 21, 22, 73, 77

Die NVIDIA Mosaic Mehrbildschirm Technologie dient zur einfachen Skalierung jeder Anwendung auf mehrere Bildschirme, und das ohne Softwareanpassungen oder Leistungseinbussen.

Motion-Tracking 34

Das Verfolgen von Körperbewegungen des Benutzers.

## **N**

Nvidia 6, 11, 21  
Einer der weltweit grössten Hersteller von Chipsätzen und Graphikprozessoren.

## **P**

Polfilter 2, 8, 10

Ein Polarisationsfilter (kurz auch Polfilter) ist ein Polarisator für Licht, der auf Dichroismus beruht, also komplementär polarisiertes Licht absorbiert, statt es wie polarisierende Strahlteiler zu reflektieren. Dadurch eignet es sich, Lichtstrahlen, die in der „falschen“ Ebene schwingen, zu unterdrücken.

PPT Studio 2013 9, 33

Software von WorldViz welche die Installation im CAVE der BFH softwareseitig unterstützt (bietet Kalibrierungseinstellungen, VRPN Output und vieles mehr an).

Prefab 28, 33, 38, 40, 51, 53

Unity Hilfsmittel, um Spielobjekte zu Instanzieren.

## R

Raycast 26, 30, 45, 56, 61

Das Verwenden von Rays (Strahlen) in der Computergraphik um Bilder zu Rendern oder 3D Objekte in der Tiefe zu erkennen.

Render 18, 73, 77

Das Berechnen der Computergraphik. Meist über die Graphikkarte.

## S

Smoothing 36, 37, 39, 40, 44  
*Siehe 1€ Filter*

Stereoskopie 2, 18, 68

Die Stereoskopie ist die Wiedergabe von Bildern mit einem räumlichen Eindruck von Tiefe, der physikalisch nicht vorhanden ist.

Stereoskopische *Siehe Stereoskopie*

## U

Unity 1, 2, 5, 6, 11, 13, 16, 19, 20, 21, 22, 24, 25, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 43, 46, 47, 48, 49, 50, 52, 53, 56, 57, 59, 60, 66, 67, 68, 69, 70, 71, 73, 76, 77

Unity ist eine Laufzeit- und Entwicklungsumgebung für Spiele und Simulationen

## V

Video Matrix Switch 12

Hardwareswitch, um verschiedene Video/Bild Ein- und Ausgänge zu koppeln.

Viewfrustum *Siehe Frustum*

Viewport 19, 55

Je nach Anwendungsgebiet ein Ausschnitt eines Bildes, eines Videos, einer virtuellen oder realen Welt bezeichnet oder der für die Darstellung zur Verfügung stehende Bereich.

VR Namespace 20

Unity Namespace für VR Geräte wie die Oculus Rift.

VRPN 9, 29, 32, 33, 34, 35, 36, 39, 40, 43, 44, 45, 47, 67, 68, 76

Virtual Reality Peripheral Network

Stellt verschiedene Tools (Server, Client, Klassen und Protokolle) zur Verfügung, um ein transparentes Interface zwischen einer Applikation und physikalischen Geräten zu bieten.

## **W**

Wand 10, 11, 13, 16, 19, 23, 26, 28, 29, 30, 31, 32, 33, 36, 38, 39, 44, 45, 47, 56, 59, 60, 73, 74

Einhändiger Controller, welcher über Infrarot LEDs verfügt, sowie mehrere Buttons und einen analogen Joystick.

Warping 57

Technologie, um ein Bild zu verzerren.

Windows 21, 39

Microsoft Windows  
Betriebssystem von Microsoft, aktuell in der Version 10.

WorldViz 9, 10, 33, 34, 73, 76, 77

Kommerzielle Firma welche komplette Systeme mit Infrarotkameras und Software anbietet.

## **13 Literaturverzeichnis**

chiragraman, G. (01. 01 2016). *github.com/chiragraman*. Von [github.com/chiragraman](https://github.com/chiragraman/Unity3DProjectionMapping): <https://github.com/chiragraman/Unity3DProjectionMapping> abgerufen

Géry Casiez, N. R. (5. 5 2012). *http://cristal.univ-lille.fr/*. Von <http://cristal.univ-lille.fr/>: <http://cristal.univ-lille.fr/~casiez/publications/CHI2012-casiez.pdf> abgerufen

Kooima, R. (01. 01 2008). *Generalized Perspective Projection*. Louisiana: Louisiana State University, Computer Science and Engineering.

Nvidia. (01. 01 2015). *www.nvidia.de*. Von [www.nvidia.de](http://www.nvidia.de): <http://www.nvidia.de/object/nvidia-mosaic-technology-de.html> abgerufen

Unity. (1. 3 2015). *Unity 3D*. Von <http://docs.unity3d.com/Manual/VROverview.html> abgerufen

Warping, W. I. (01. 01 2016). *Wikipedia*. Von Wikipedia: [https://en.wikipedia.org/wiki/Image\\_warping](https://en.wikipedia.org/wiki/Image_warping) abgerufen

## 14 Anhang

- Benutzerhandbuch
- Benutzerhandbuch WorldViz
- Selbstständigkeitserklärung
- Pflichtenheft
- Plakat
- Bucheintrag
- Aufgabenstellung