



Hochschule
Kaiserslautern
University of
Applied Sciences

Informatik und
Mikrosystemtechnik
Zweibrücken

Studiengang
Medieninformatik

PO Version 2011

Bachelorarbeit

Immersive Strömungsvisualisierung in Unity

Immersive Flow Visualization in Unity

vorgelegt von

Karim Stock

19. Dezember 2016

Betreuung: Prof. Dr. Manfred Brill

Zweitkorrektur: Prof. Dr. Michael Bender



Hochschule
Kaiserslautern
University of
Applied Sciences

Informatik und
Mikrosystemtechnik
Zweibrücken

Studiengang
Medieninformatik

PO Version 2011

Bachelorarbeit

Immersive Strömungsvisualisierung in Unity

Immersive Flow Visualization in Unity

vorgelegt von

Karim Stock

19. Dezember 2016

Betreuung: Prof. Dr. Manfred Brill

Zweitkorrektur: Prof. Dr. Michael Bender

Immersive Strömungsvisualisierung in Unity

Karim Stock

1. August 2018

Kurzfassung

Diese Bachelorarbeit dokumentiert die Erstellung einer Unity3D Anwendung zur Strömungsvisualisierung mit Virtual Reality Support. Zur Realisierung wurde Unity Engine, Version 5.5 Beta verwendet und es wurde mit der HTC Vive als Head-mounted Display gearbeitet. Sie richtet sich an Studenten und Informatiker, sowie Ingenieure im Bereich Strömungsmechanik, welche grundlegende Erfahrung zur Entwicklung mit Unity Engine und dem Arbeiten mit Virtual Reality Hardware haben, sowie Verständnis der Programmiersprache C#.

Abstract

This scientific paper deals with the creation of a Unity3D application on flow visualization in virtual reality. Unity Engine 5.5 Beta was used for the implementation of the project and the HTC Vive was used as the Head-mounted Display.

This work is directed toward students and computer scientist as well as engineers with an understanding of fluid mechanics, that acquired basic experiences with developing in Unity Engine and working with virtual reality hardware as well as an understanding of the programming language C#.

Inhaltsverzeichnis

1 Einleitung	3
1.1 Einführung	3
1.1.1 Aufgabenstellung	3
1.1.2 Gliederung der Bachelorarbeit	4
2 Auswahl des Plugins Mega Flow	5
2.1 Strömungsvisualisierung	5
2.2 Unity Asset Packs	5
2.3 Anforderungen an Fluid Flow Plugin	6
2.3.1 Dokumentation und Support	6
2.3.2 Frameraten und Performanz	6
2.3.3 Import von Vektorfeldern	7
2.3.4 Partikel	7
2.3.5 Stromlinien	8
3 Realisierung	9
3.1 Erstellung des standard Raumes	9
3.1.1 Lichteinstellungen der Szene	13
3.2 SteamVR	17
3.3 VRTK - SteamVR Unity Toolkits	21
3.4 Mega Flow	23
3.4.1 Skalierung	27
3.4.2 Trail Renderer	32
3.4.3 Particle System	34
3.4.4 Particle Trigger	37
3.5 Küchenszene	43
3.5.1 Import VTK Daten in Unity	43
3.5.2 Erstellung Küchenszene	44
3.5.3 Physikalisches Regler Menü	50
3.5.4 Regler in Küchenszene	54
3.5.5 Tooltips	57
4 Fazit	59
4.1 Features aus der Zukunft	59
Literaturverzeichnis	61

1 Einleitung

1.1 Einführung

Diese Bachelorarbeit dokumentiert die Erstellung einer Unity3D Anwendung mit Virtual Reality Support, sie setzt Grundkenntnisse zur Entwicklung mit Unity Engine, dem Arbeiten mit Virtual Reality Hardware, sowie Verständnis der Programmiersprache C# voraus. Zum Einstieg in Unity empfehlen sich die offiziellen Unity Tutorials [Tec16b]. Zum Verständnis von Strömungsvisualisierung und damit verbundenen Vektorfeldern empfiehlt sich *Computergrafik. Ein anwendungsorientiertes Lehrbuch* [BB03].

1.1.1 Aufgabenstellung

Das Ziel dieser Bachelorarbeit ist, den Prototyp einer interaktiven VR Applikation in Unity zu entwickeln, welche zum Visualisieren von Strömungsdaten im dreidimensionalen Raum genutzt werden kann.

Dafür nötig ist, Vektorfelder in Unity3D zu nutzen, Strömungsdaten zu Importieren, VR Elemente einzubeziehen:

Wo befindet sich der Nutzer im Raum, wie groß sind die dargestellten Objekte?

Findet es um den Nutzer herum statt, oder guckt er auf eine vordefinierte Plattform?

Genutzt wurde für die Umsetzung Unity 5.5 mit der Entwicklungsumgebung Visual Studio 2015 für den c# Code. Als VR Hardware wurde die HTC Vive verwendet, mit dem dazugehörigen SteamVR Unity Plugin wurde der Support für das *Head-Mounted Display* bewerkstelligt.

Weitere Plugins, die im Laufe der Realisierung genutzt worden sind: *Mega Flow* [Chr16a] zum Erstellen von Vektorfeldern und Importieren von Strömungsdaten, *VRTK - SteamVR Unity Toolkit* [Sys16] für die Interaktion der Vive Controller, sowie *VRBasics* [Zac16] für ein physikalisches Menü, welches ebenfalls mit den Vive Controllern genutzt werden kann. Zusätzliche – rein visuelle – Plugins werden an den jeweiligen Stellen in Kapitel 3 Realisierung erwähnt.

1.1.2 Gliederung der Bachelorarbeit

Kapitel 1 Einleitung

Das Kapitel stellt die Aufgabenstellung, sowie die damit verbundene Entwicklungsumgebung und die vorausgesetzten Grundkenntnisse vor.

Kapitel 2 Auswahl des Plugins Mega Flow

In diesem Kapitel werden Anforderungen, die an ein Visualisierungsplugin für Unity gestellt werden offenbart und die Auswahl des Plugins *Mega Flow* begründet.

Kapitel 3 Realisierung

Es wird Schritt für Schritt die Entwicklung der Anwendung mit Vorstellung und Nutzen aller zugehörigen Plugins und Importverfahren beschrieben.

Kapitel 4 Fazit

Das letzte Kapitel bietet nochmal eine Zusammenfassung der erstellten Anwendung, sowie den Ausblick auf zukünftige Funktionen, die zu erwarten sind.

2 Auswahl des Plugins Mega Flow

2.1 Strömungsvisualisierung

Strömungsvisualisierung ist ein Verfahren um Strömungsvorgänge darzustellen. Zur Veranschaulichung können z.B. Stromlinien (*Streamlines*) oder Streichlinien (*Streaklines*) verwendet werden, welche beide dazu dienen, den Pfad in einem Vektorfeld darzustellen.

Ein Vektorfeld beschreibt eine Funktion, welche jedem Punkt eines Raumes einen Vektor zuordnet. Demnach beschreiben Stromlinien Kurven – welche den Pfad der Strömung zeigen – indem sie in jedem Punkt eine parallele Tangente zum Ortsvektor des Feldes aufweisen.

Zur Visualisierung mit Streichlinien werden an einem Startpunkt kontinuierlich Partikel injiziert, welche sich an den jeweiligen Ortsvektoren des Vektorfeldes fortbewegen [BGZ13]. Eine einfache Möglichkeit, solche Partikel in festen Abständen zu injizieren, bietet Unity mit dem Partikelsystem bereits, weshalb diese Methode der Visualisierung nahe liegt. Jedoch wurde ein Plugin verwendet um Vektorfelder darzustellen und für diese Partikel die jeweiligen Ortsvektoren zu übernehmen, während sie in der Szene gerendert wurden. Das verwendete Plugin *Mega Flow* stellt diese Funktionen zur Verfügung, das erwähnte *Computergrafik Lehrbuch* [BB03] führt Methoden auf, Vektorfelder selbst darzustellen.

2.2 Unity Asset Packs

Asset Packages bieten eine praktische Möglichkeit, eine selbst bestimmbare Ansammlung von Assets oder Teilen eines Projektes zu Exportieren und zu Importieren [Uni16a].

Sie stellen eine Ansammlung von Dateien und Daten eines Unity Projektes — oder Teil eines Projektes – dar, welche komprimiert und in einer Datei — ähnlich eines Zip-Archives – gespeichert werden [Uni16b].

Diese Asset Packages stellen für den weiteren Verlauf des Projektes die genutzten Plugins dar und durch den Unity Asset Store ist es möglich, bereits erstellte Plugins, die für Strömungsvisualisierung und Virtual Reality relevant sind zu importieren.

2.3 Anforderungen an Fluid Flow Plugin

Da Strömungsvisualisierung in Unity voraussetzt, dass ein Vektorfeld dargestellt wird, war es notwendig, ein Asset Package bzw. Plugin zu verwenden, welches dies ermöglicht.

In der anfänglichen Planungszeit für die Realisierung mit Unity tauchten allerdings nur wenige Plugins für die Stichwörter *fluid flow* oder *vectorfield* im Unity Asset Store auf, wovon die meisten für eine ältere Version von Unity angelegt worden sind und durch mehrere Jahre ohne Update nicht in Frage kamen.

Ein Plugin – *Mega Flow* – konnte jedoch mit vielen Funktionen und aktuellem Code überzeugen. Für dieses Projekt wurde also *Mega Flow* verwendet, mehr zur Arbeit mit *Mega Flow* kann in Kapitel 3.4 gelesen werden.

Die folgenden Anforderungen wurden dennoch zur Bestimmung des Plugins formuliert und können zur zukünftigen Weiterentwicklung der Applikation genutzt werden um eine Alternative zu *Mega Flow* festzustellen.

2.3.1 Dokumentation und Support

- Wie *aktiv* ist die Entwicklung des Plugins? Gibt es regelmäßige Updates?
- Wie gut ist die Dokumentation des Plugins?

Mega Flow Dokumentation und Support

Das Plugin wurde für viele unterschiedliche Unity Versionen zwischen 4.2.0 und 5.5.0 aktuell gehalten, mit dem letzten Update während der Bachelorarbeit am 12.11.2016.

Die *Mega Flow* Website [Chr16b] bietet Dokumentation und Beispiele, sowie Verweise an Videos, die eine Hilfestellung für den Umgang mit dem Plugin bieten. Eine Umfangreiche Dokumentation kann nach dem Import des Plugins im Unity Package gefunden werden.

2.3.2 Frameraten und Performanz

- Kann mit dem Plugin eine, für eine VR-Anwendung geeignete Framerate erreicht werden? Hängt dies von der Größe des verwendeten Vektorfelds ab?
- Unterstützt das Plugin neben stationären auch instationäre Vektorfelder?
- Welche Visualisierungsmethoden werden unterstützt?
- Ist es möglich das Plugin selbst zu erweitern, um Visualisierungsmethoden anzupassen oder eigene zu implementieren?

Mega Flow Frameraten und Performanz

Das Plugin hat keine offensichtliche Limitierung für die Vektorfelder. Frameraten und Performanz sind mehr von der Art der Darstellung der Partikelsysteme abhängig, als vom erstellten Vektorfeld. Es können lediglich stationäre Vektorfelder erstellt werden und es sind Elemente enthalten um Partikel sowie jegliche anderen Objekte in Unity von erstellten Strömungen beeinflussen zu lassen. Der *Mega Flow* Code ist komplett einsehbar und modifizierbar.

2.3.3 Import von Vektorfeldern

- Können Vektorfelder aus Dateien importiert werden?
- Welche Formate werden unterstützt?
- Können Konvertierung aus Formaten in andere, vom Plugin unterstützte Dateiformate implementiert werden? Wie groß ist der Aufwand? Als Beispiele können die Dateien verwendet werden, die im *Computergrafik* Buch [BB03] verwendet werden.

Mega Flow Import von Vektorfeldern

Unterstützt werden Import von *Maya Fluids*, *FumeFX* und *FGA* Dateien, sowie ein eigenes, XML basiertes Dateiformat, *FLW* Dateien [Chr16b]. Der Import eigener Daten, sowie die Konvertierung einer *VTK* Datei als *FLW* erfolgen in Kapitel 3.5.1, wobei die genutzten *VTK* Daten aus den *VTK* Beispielen entsprungen und ebenfalls in *Informatik im Fokus: Virtuelle Realität* [Bri08] zur Visualisierung verwendet wurden.

2.3.4 Partikel

- Wird das Unity Partikelsystem verwendet?
- Wie werden Partikel gerendert?
- Welche Attribute der Partikel können im SDK und interaktiv verändert werden?
- Kann die Zeit *umgedreht* werden? Kann in der Anwendung die Zeit rückwärts laufen, zum Beispiel um einen Verlauf der Partikel genauer zu untersuchen?

Partikel in Mega Flow

Das Unity Partikelsystem [Uni16c] kann verwendet werden und *Mega Flow* bietet eine weitere Komponente, welche ermöglicht zusätzlich die Masse der Partikel für deren Wirkung in der Strömung anzupassen. Die normalen Unity Partikelsystem Attribute können auch

jederzeit angepasst werden, jedoch gibt es keine Möglichkeit durch *Mega Flow* die Simulation *zurückzuspulen*.

2.3.5 Stromlinien

- Können Stromlinien berechnet und gespeichert werden?
- Wie können Stromlinien gerendert werden? Unterstützt das Plugin das von sich aus?
- Welche Attribute der Linien können im SDK und interaktiv verändert werden?

Stromlinien in *Mega Flow*

Das berechnen der Stromlinien erfolgt mit *Mega Flow* ausschließlich im Unity Editor, nicht während der Runtime. Es gibt diverse Anpassungen, die man mit *Mega Flow* für die dargestellten Stromlinien vornehmen kann, jedoch sind diese nur für die Vorschau des Vektorfeldes und der Strömung gedacht, weshalb für die Runtime eine eigene Form der Darstellung erfolgen müsste.

3 Realisierung

3.1 Erstellung des standard Raumes

Zur Erstellung des Raumes wurden Teile des Asset Packs *3D Showroom: Level Kit Vol 1* [Cre15] verwendet. Diese befinden sich unter folgendem Pfad:

```
/Assets/_ShowRoom/_Prefabs/_Rooms/
```

und es können verschiedene Teile zur Erstellung eines simplen Raumes gefunden werden.

Als erstes Prefab zur Erstellung der Default Umgebung, wurde *Seperator_Closed_01* (Abb. 3.1) verwendet, eine Wand mit dazugehörigen Logo, welches später zum Logo der Hochschule Kaiserslautern geändert wurde.

Da die Prefabs des Assetpacks eigentlich zusammengehörig für einen anderen Demo Raum gedacht waren, wurden sie hier manuell ausgelegt, in dem Fall wurde das Prefab als Ausgangspunkt für den Rest des Raumes genutzt und ihm wurde die Position (0,0,0) zugewiesen.

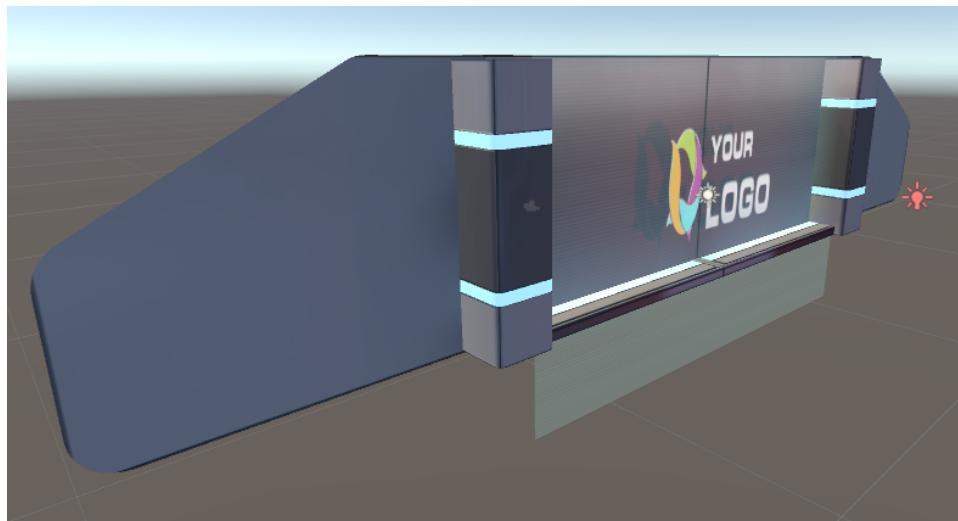


Abbildung 3.1: Seperator_Closed_01 aus 3D Showroom Level Kit Vol 1

Nun wurde das nächste Prefab, BigRoom_01_part_01 hinzugefügt und ebenfalls auf die Position (0,0,0) gelegt, wie es mit der *Object Origin* bei diesem Asset gedacht ist, um direkt an dem vorigen Prefab Seperator_Closed_01 anzuliegen (Abb. 3.2).

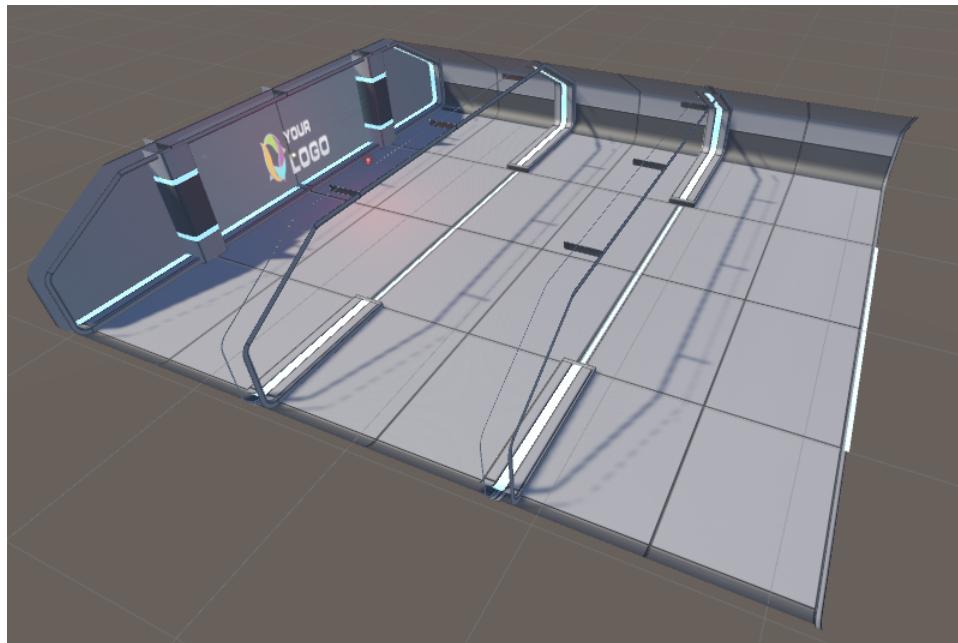


Abbildung 3.3: Raumelemente aus 3D Showroom Level Kit Vol 1

Um diesen Abschnitt – für einen größeren Raum – mehrmals darzustellen, wurde das Prefab BigRoom_01_Part_01 zwei weitere Male in die Szene eingefügt, mittels des Unity Objekt Rasters (an Position (0,0,14) und Position (0,0,28)), um perfekt an den anderen anzuliegen (Abb. 3.3).

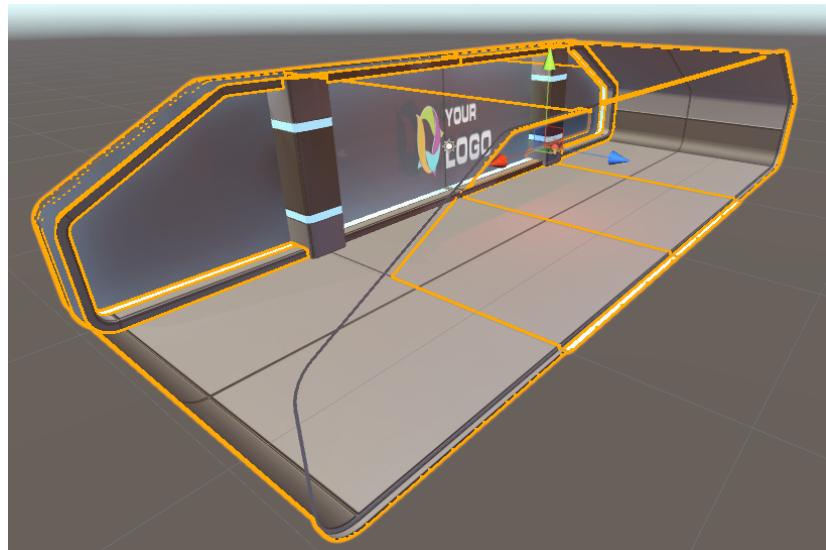
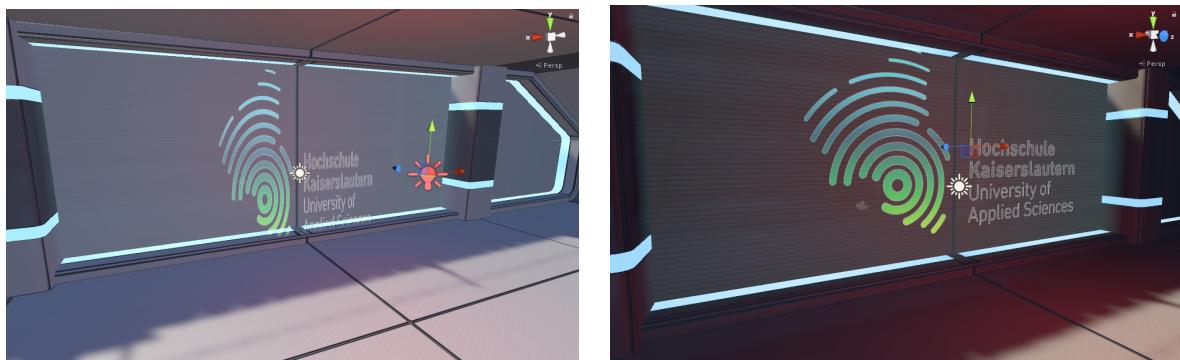


Abbildung 3.2: BigRoom_01_Part_01 aus 3D Showroom Level Kit Vol 1



(a) Verzerrtes Hochschullogo

(b) Korrektes Logo

Abbildung 3.4: Falsche und korrekte Darstellung des Hochschullogos

Da die Größe bzw. das Verhältnis der Skalierungswerte des Quad Objektes nicht mit dem, des Logos der Hochschule übereinstimmt (Abb. 3.4a), musste ein anderer Wert genutzt werden, welcher der eigentlichen Auflösung des Logos entspricht.

Dafür wurde eine entsprechende Skalierung von (1.5525, 0.8082, 1) gewählt, welche durch Multiplikation von 0,9 mit der ursprünglichen Auflösung (1725x898) ermittelt wurde (Abb. 3.4b).

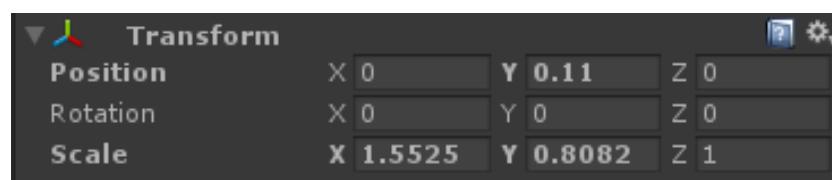


Abbildung 3.5: Transform Component des Logos

Zusätzlich dazu wurde die Position auf der Y-Achse des Logos um 0,11 verändert, dass es zentral auf der Fläche anliegt, wodurch sich Werte der *Transform* Komponente von Abb. 3.5 für das Quad ergeben haben.

Auffallend störend war jedoch das rote Licht,

Szenehierarchie: Seperator_Closed_01 > Logo > Point light

welches die Wand und das eingefügte Hochschullogo bestrahlt hat (Abb. 3.4b) und eher unpassend für einen praktischen Prototypen ist, weshalb es auf ein angenehmeres Blau (#837FB0FF) angepasst wurde (Abb. 3.6).



Abbildung 3.6: Angepasstes Point Light des Logos

Abschließend wurde das fertig angepasste Seperator_Closed_01 Prefab auf das andere Ende des Raumes kopiert (Position (0,0,41)) und auf der Y-Achse um 180° gedreht.

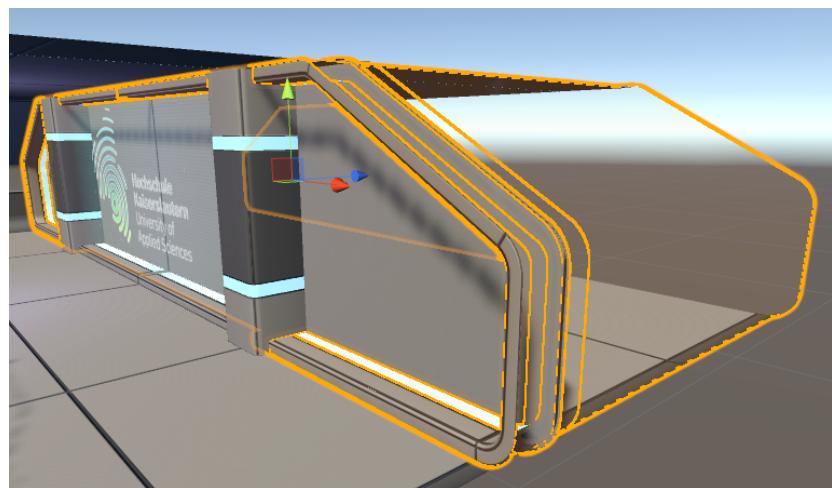


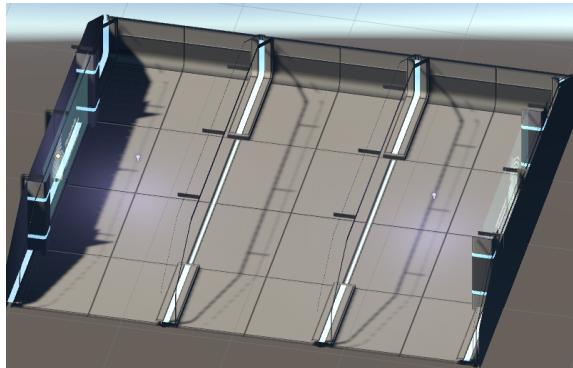
Abbildung 3.7: Rückseite des Raumes

Um ebenfalls die Bögen der einzelnen Raumabschnitte an der hinteren Wand mit dem Logo anzufügen, wurde eine weitere, vierte Kopie von

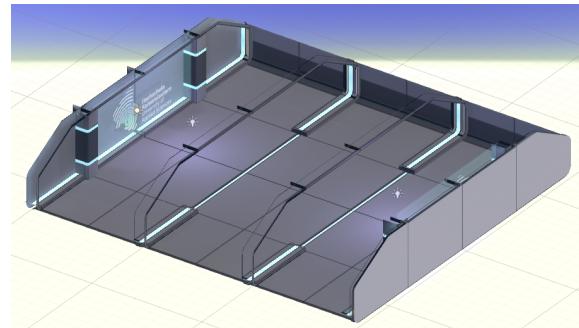
BigRoom_01_Part_01 angelegt und die Bögen an die Wand am Raumende angelehnt, wobei der Rest dieses Prefabs nun gelöscht werden konnte, da lediglich die Bögen gefehlt hatten (Abb. 3.7).

An dieser Stelle war die Geometrie des Demo Raumes fertiggestellt. Es war weiterhin nur noch nötig, die Lichteinstellungen und Lichtobjekte in der Szene anzupassen.

3.1.1 Lichteinstellungen der Szene



(a) Szene ohne erweiterten Lichteinstellungen



(b) Szene mit angepasstem Licht

Abbildung 3.8: Szene vor und nach der Umsetzung der Lichtsektion

Trotz Fertigstellung der Geometrie des Raumes war es an dieser Stelle nötig, Anpassungen im *Lighting-Tab* der Szene zu machen.

Abb. 3.8 zeigt hierfür die Szene vor und nach den Anpassungen, die in folgendem Abschnitt vorgenommen werden und zeigt bei Abb. 3.8b die einheitlich beleuchtete Umgebung als Resultat.

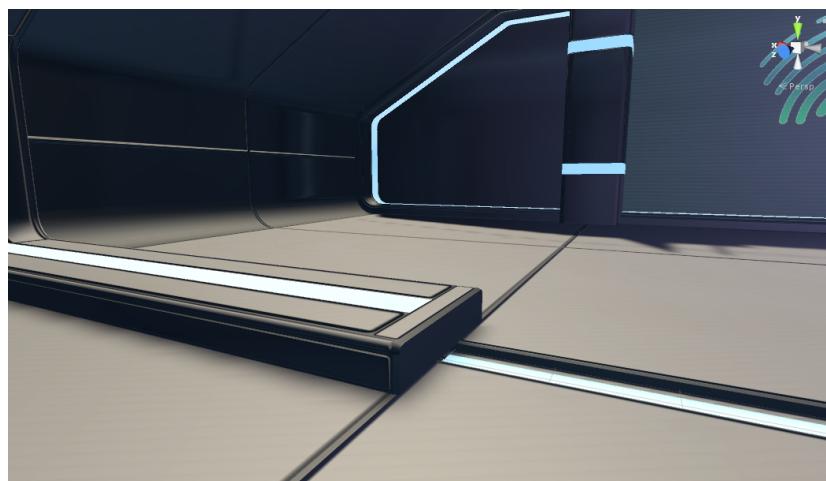


Abbildung 3.9: Zu dunkle Kanten in der Szene

Das größte Problem – Licht und allgemeine Helligkeit im Raum betreffend – war, dass selbst nach *Baken* der Szene mit *Global Illumination* und Setzen der Lichter auf *Baked (Light Component: Baking Type: Baked)* einige Schatten und Kanten, die nicht in der direkten Beleuchtung von einer Lichtquelle liegen viel zu dunkel erschienen sind (Abb. 3.9).

Es haben sich hier mehrere Möglichkeiten ergeben, wie man mit diesem Problem umgehen könnte. Zum einen wäre es natürlich möglich gewesen, die Anzahl der Lichtquellen im Raum zu erhöhen und einige *Point Lights* im Raum zu verteilen, welche vom *Baking Type: Mixed* sind.

Da der Raum jedoch sehr groß ist und die Ergebnisse von solchen Lichterweiterungen erst nach erneutem *Baking* der Szene ersichtlich gewesen wären, dieser Vorgang sich für jede Lichtquelle in der *Baking Pipeline* allerdings verlängert hätte, wurden die anderen Möglichkeiten damit umzugehen in Betracht gezogen.

Eine weitere, simplere Lösung des Problems, mit den dunklen Schatten umzugehen, wäre es, im *Lighting Tab* der Szene unter dem Punkt *Environment Lighting* die *Reflection Bounces* zu erhöhen.

Standardmäßig reflektiert eine Lichtquelle vom *Baking Type: Mixed* oder *Baked* auf den Oberflächen von statischen Objekten einmal, wodurch keine der Schatten in der Szene absolut schwarz sind, jedoch in diesem Fall trotzdem zu dunkel.

Erhöht man also die Anzahl der *Reflection Bounces*, so wird mehr Licht auf den statischen Objekten reflektiert und der Raum an sich würde erhellt dargestellt werden.

Das Problem, was sich hier wiederum ergäben hätte, ist, dass der *Baking* Prozess sich lediglich auf statische Objekte bezieht und alle Objekte, die während der Demo bewegt werden, weiterhin von den zu dunklen Schatten befallen sein würden.

In dieser Szene gab es diese Objekte zwar noch nicht, aber da im weiteren Verlauf der Realisierung Support für die HTC Vive implementiert wird, ist es nötig, die Darstellung bewegbarer Objekte, wie die *tracked Controller* dementsprechend in Betracht zu ziehen.

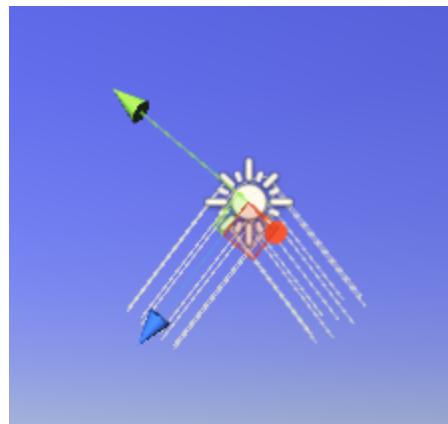


Abbildung 3.10: Zwei direktionale Lichter in unterschiedlichen Winkeln

Letztendlich ist die Lösung für dieses Problem, die angewandt wurde, ein weiteres – zweites – Direktionales Licht in die Szene einzufügen, welches aus der entgegengesetzten Richtung auf der Y-Achse des ersten Lichts strahlt.

Dafür wurde das *Directional Light* dupliziert und um 180° auf der Y-Achse rotiert (Abb. 3.10).

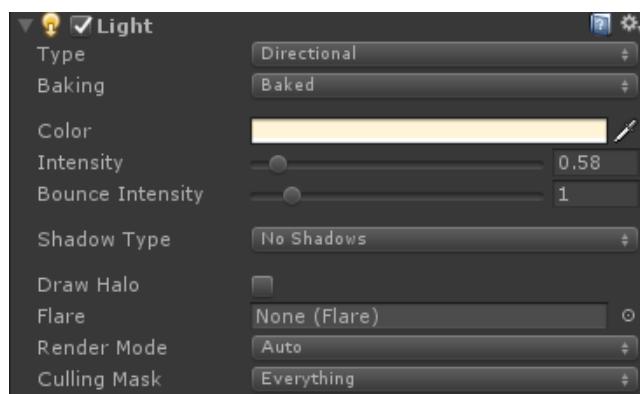


Abbildung 3.11: Light Component des zweiten Directional Light

Zusätzlich wurden für diese Lichtquelle die Schatten deaktiviert, die Intensität herabgesetzt und der Baking Type auf Baked gesetzt (Abb. 3.11).

Als letzten Schritt für die Lichtanpassungen der Demo Szene war es nötig, eine neue *Skybox* zu erstellen. Obwohl es von der Innenseite des Raumes nicht möglich ist, die *Skybox* zu sehen, spielt diese eine große Rolle für die Beleuchtung der Szene.

Im *Lighting Tab* steht die *Ambient Source* des *Environment Lighting* standardmäßig auf *Skybox*, was bedeutet, dass reflektive Materials und Shader, welche von ihrer Umgebung als Reflektionsquelle Gebrauch machen – wie auch der Unity Standard Shader – auf die *Skybox* zurückgreifen.

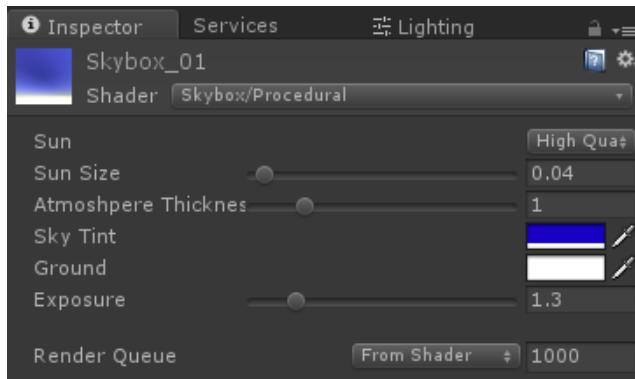


Abbildung 3.12: *Skybox Material*

Um also das *Ambient Lighting* an die eigentliche Vorstellung des Raumes anzupassen, wurde eine neue *Skybox*, *Skybox_01* als Material, *Skybox (Procedural)* generiert, welche auf einem stärkeren blau basiert (Abb. 3.12).



Abbildung 3.13: *Environment Lighting*

Mit dieser *Skybox* wurde die Standard *Skybox* überschrieben und mit den Einstellungen des *Environment Lighting* auf Abb. 3.13 wurde die Szene letztendlich *gebaked*.

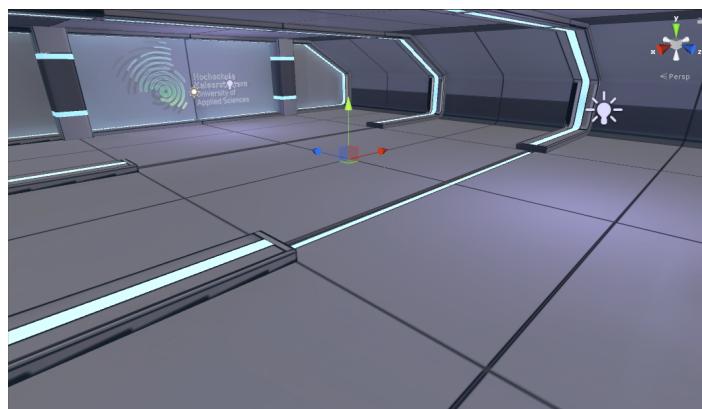


Abbildung 3.14: Ergebnis der Abschnitts

3.2 SteamVR



Abbildung 3.15: SteamVR Plugin im Asset Store

Das SteamVR SDK ermöglicht die Entwicklung für OpenVR. Es bietet Support vieler unterschiedlicher Virtual Reality Headsets, unabhängig davon, ob es sich um eine Anwendung handelt, welche das Stehen oder Sitzen voraussetzt, oder diese die *getrackten* HTC Vive Controller nutzt.

Zusätzlich bietet es Zugriff zur Entwicklung mit den Vive Controllern und enthält Beispiele für den Umgang mit diesen, sowie weitere Beispiele um Unity's UI Systeme in VR darzustellen.

Alles in Allem ist das SteamVR Plugin nötig, um die Build Pipeline für neue Build Targets zu erweitern, in diesem Fall eben die HTC Vive via OpenVR [Val16].

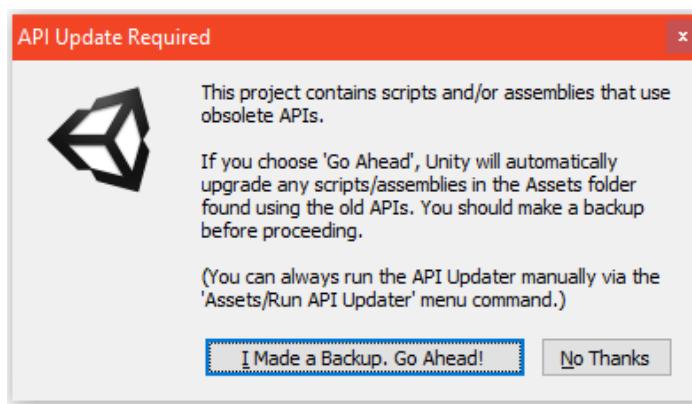


Abbildung 3.16: Unity API Updater

Nach dem Download und Import des Asset Packs, empfiehlt Unity den API Updater die Scripts zu erneuern (Abb. 3.16).

Da die Unity Beta 5.5.0b4 [Uni16d] zur Erstellung der Demo benutzt wurde, das letzte Update des SteamVR Plugins jedoch für Unity 5.4.0 [Tec16a] war, wurden manche Methoden in der API verändert oder entfernt, weshalb es möglich ist, dass das Plugin alten Code verwendet, der nicht mehr unterstützt wird.

Um eben diesen veralteten Code für die genutzte Version zu erneuern, wurde das Fenster bestätigt und kontrolliert, dass die Konsole keine weiteren Fehler ausgegeben hat.

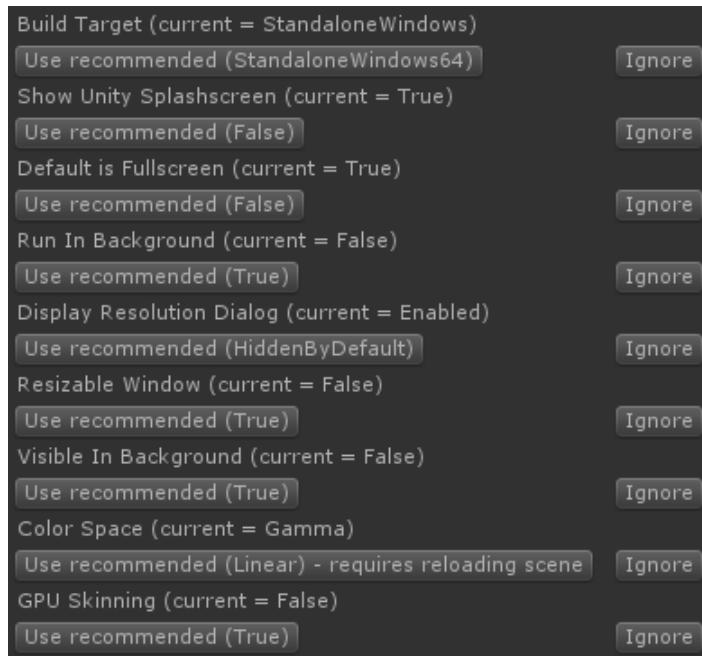


Abbildung 3.17: SteamVR Setup

Nach erfolgreichen Import und Updaten des Codes startet das SteamVR Plugin das VR Setup, welches Einstellungen vorstellt und empfiehlt um mit VR Support zu entwickeln (Abb. 3.17).

Es besteht jedoch die Möglichkeit, diese Vorschläge einzeln zu bestätigen bzw. zu ignorieren oder die Einstellungen zu einem späteren Zeitpunkt vorzunehmen.

Bei diesem Projekt wurden an der Stelle alle Vorschläge bestätigt und die von SteamVR empfohlenen Einstellungen genutzt, wobei jedoch später ein VR Splash Screen eingeführt wurde, ein weiteres Feature der genutzten Unity Beta.

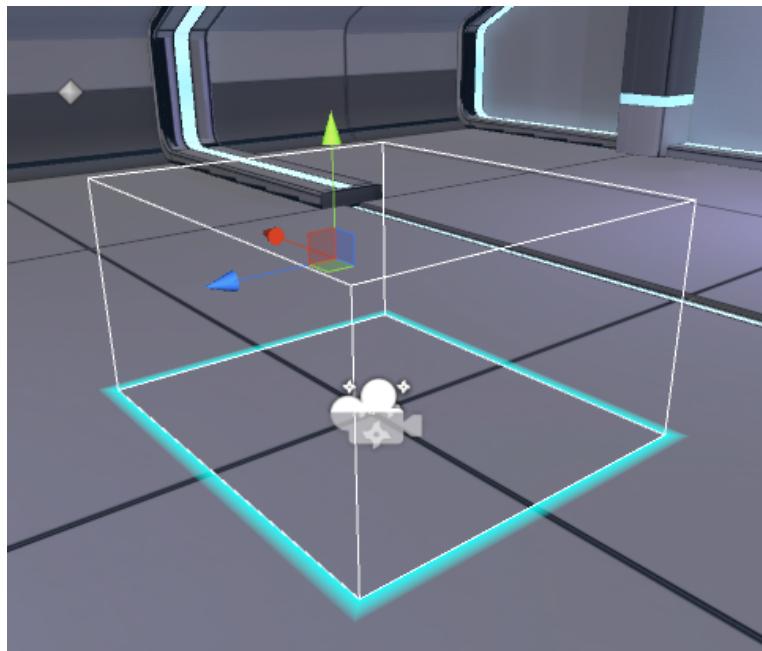


Abbildung 3.18: SteamVR Prefab [CameraRig]

Um den SteamVR Support in der Szene zu nutzen, wurde die alte 2D Kamera der Szene, *MainCamera* deaktiviert und das SteamVR Prefab [CameraRig] aus

```
/Assets/SteamVR/Prefabs/
```

in die Szene geladen (Abb. 3.18).

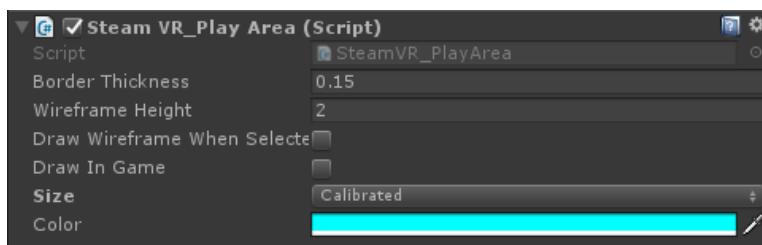


Abbildung 3.19: SteamVR Play Area

Für die Roomscale *Play Area* – die während der Runtime auf dem Boden dargestellt werden soll um die Grenzen des Raumes zu sehen – wurden die Standardeinstellungen des Steam VR_Play Area Scripts verwendet, jedoch war es nötig, die Größe (*Size*) von einem festen Wert auf *Calibrated* zu stellen (Abb. 3.19).

Build Settings

Um die Anwendung mit VR Support zu *builden*, mussten in den Build Settings wie gewohnt die Szenen bzw. die Szene ausgewählt werden. Danach war es nötig Anpassungen an den Player Settings vorzunehmen.

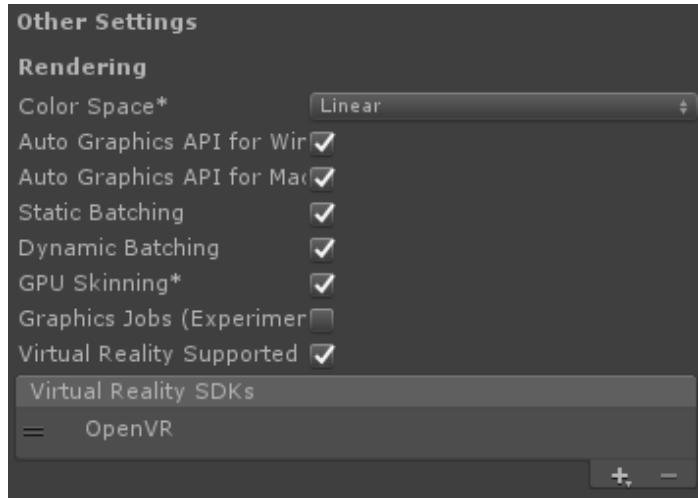


Abbildung 3.20: Other Settings (Player Settings)

Unter dem Reiter *Other Settings* der Player Settings ist es möglich, die Einstellungen für Virtual Reality Support zu finden. Es ist lediglich notwendig, die *Virtual Reality Supported* Checkbox zu bestätigen und *OpenVR* als Virtual Reality SDK hinzuzufügen (Abb. 3.20).



Abbildung 3.21: VR Splash Screen Einstellungen

Schließlich – zum einbinden eines Virtual Reality Splash Images – wurden die Einstellungen unter *Splash Image* in den Player Settings ignoriert und die Checkbox *Show Splash Screen* deaktiviert, da diese lediglich für die zweidimensionale Darstellung auf dem Desktop sind.

Etwas anderes, was diese Version von Unity jedoch bietet, ist das *Virtual Reality Splash Image*, welches ermöglicht, einen experimentellen *Splash Screen* für VR darzustellen.

Es wurde erstmal das Logo der Hochschule verwendet (Abb. 3.21), jedoch ist es ratsam mehr Informationen auf ein Splash Image abzubilden, als nur das Hochschullogo.

3.3 VRTK - SteamVR Unity Toolkits

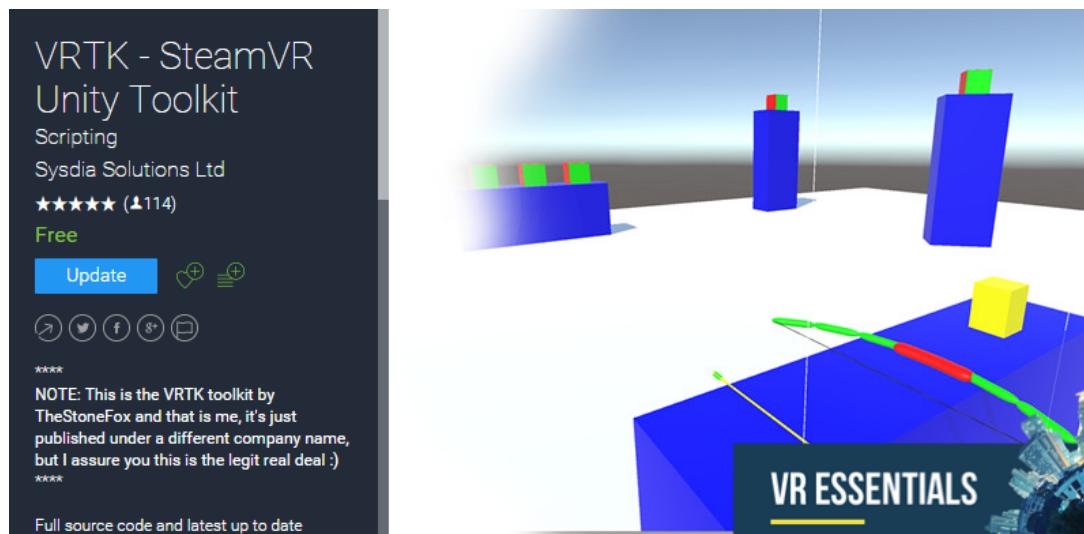


Abbildung 3.22: VRTK im Asset Store

VRTK ist ein weiteres Plugin. Es bietet simplen und strukturierten Code, sowie eine Vielzahl von Anwendungsbeispielen zum Darstellen mit SteamVR und vor allen Dingen zur Handhabung des Controller Inputs, sowie Beispielen der verschiedenen Interaktionsmöglichkeiten der Buttons, Trigger und Touchpads der SteamVR Controller in Kombination mit unterschiedlichen anderen Objekten im Raum oder als Teil von Unity's User Interface.

Für diese Szene ist erstmal die Teleport Funktion wichtig, welche schnelles Ändern der Position des Users durch Markieren im Raum ermöglicht.

Ebenfalls wichtig für die spätere Navigation durch unterschiedliche Partikeleinstellungen sind die Beispiele für die Touchpad *Wheel* Navigation.

Diese lässt mit einfachem *Swipe*n auf dem Touchpad eine Auswahl im Menü *RadialMenu* verbinden.

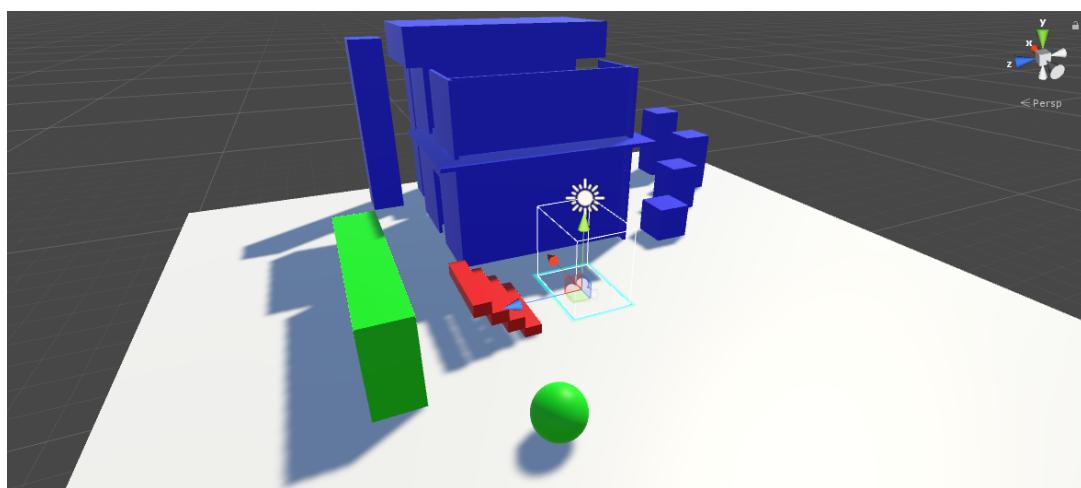
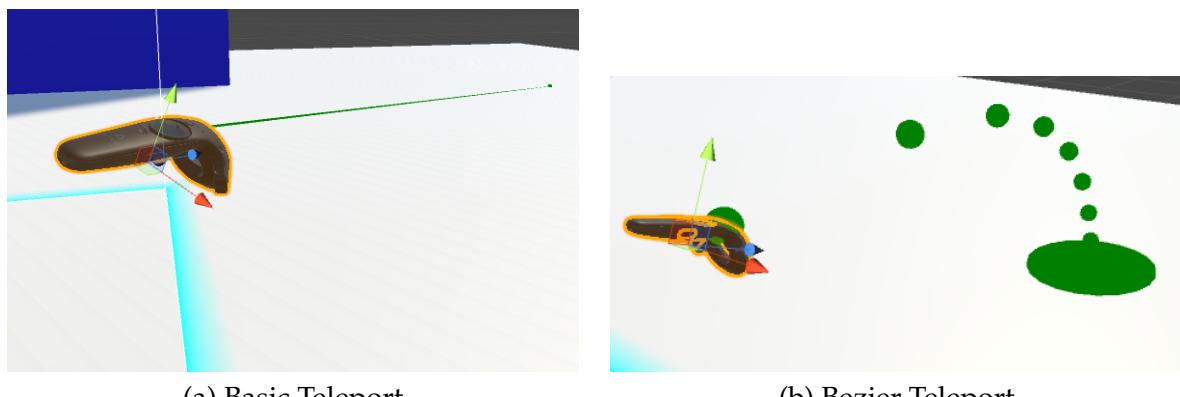


Abbildung 3.23: Szene

Es ist ratsam, Szene009_Controller_BezierPointer
`/Assets/SteamVR/Unity/Toolkit/Examples/`

zu betrachten um sich ein Bild von den Features zu machen und die Funktion zu überprüfen.

In der Szene befinden sich alle für das Plugin und die Teleportfunktion relevanten Scripts, zu finden sind diese auf den SteamVR Elementen [CameraRig], Controller(left) und (right).



(a) Basic Teleport

(b) Bezier Teleport

Abbildung 3.24: Unterschiedliche Teleportprefabs aus VRTK

Es wurden die Komponenten des linken Controllers in dieser Szene in die bereits erstellte Szene des Demo Raumes kopiert, um die Teleportfunktion für den Bezier Teleport (Abb. 3.24b) mit der linken Hand nutzen zu können.

3.4 Mega Flow



Abbildung 3.25: *Mega Flow* im Unity Asset Store

Ziel dieses Abschnittes war es, eine Szene zu entwerfen, welche Features und Funktionen von *Mega Flow* [Chr16a] in der, in Kapitel 3.1 erstellten Beispielumgebung demonstriert. Dafür wurde mit einem weiteren Plugin von Chris West – Entwickler der Mega Plugins – *Mega Shapes* [Chr12] eine Form erstellt, welche dann eine Strömung, *Mega Flow Source* berechnet hat, die in der Szene benutzt werden konnte.

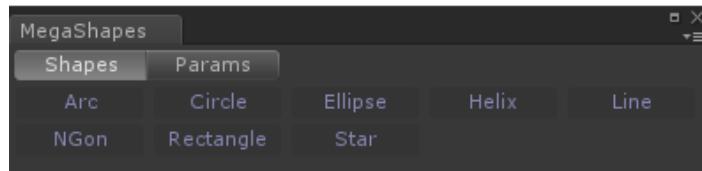


Abbildung 3.26: *Mega Shapes* Interface

Um eine Form mittels *Mega Shapes* zu erstellen, wurden im Tab `Assets` oben im Unity Editor `MegaShapes` gewählt. Diese Auswahl öffnet ein Fenster, welches verschiedene *Shapes* als Basis präsentiert (Abb. 3.26).

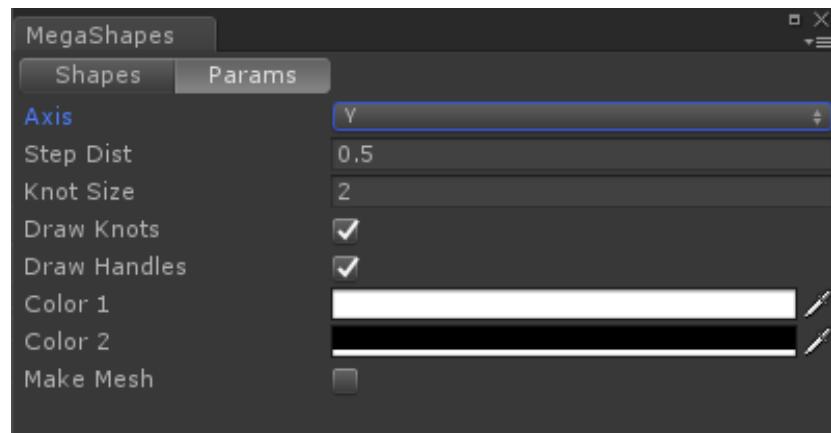


Abbildung 3.27: *MegaShapes* Fenster: Interface, Params

Es können jedoch im Reiter *Params* weitere, eigene *Shapes* direkt erstellt werden. Da das Ziel dieses Abschnittes lediglich die Vorarbeit der Strömungsdarstellung und Interaktion in Virtual Reality ist, war die eigentliche Form des *Shape* Objekts irrelevant, da später andere Strömungsdaten zur Simulation benutzt werden.

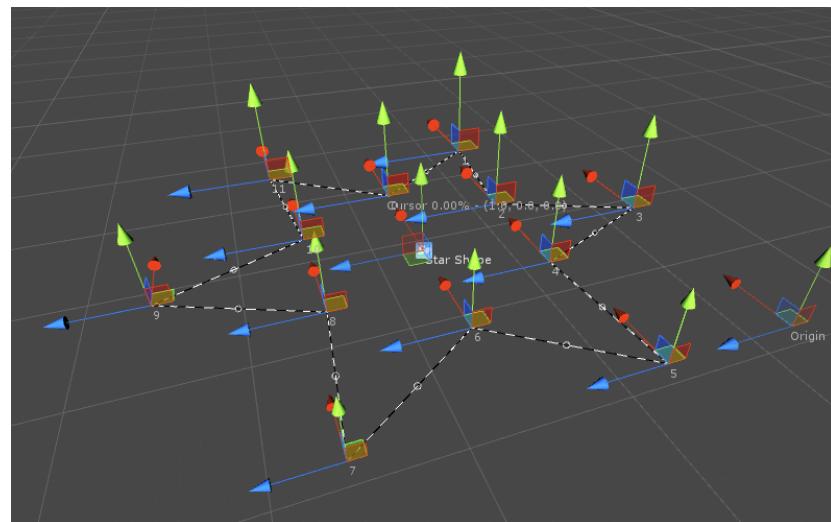
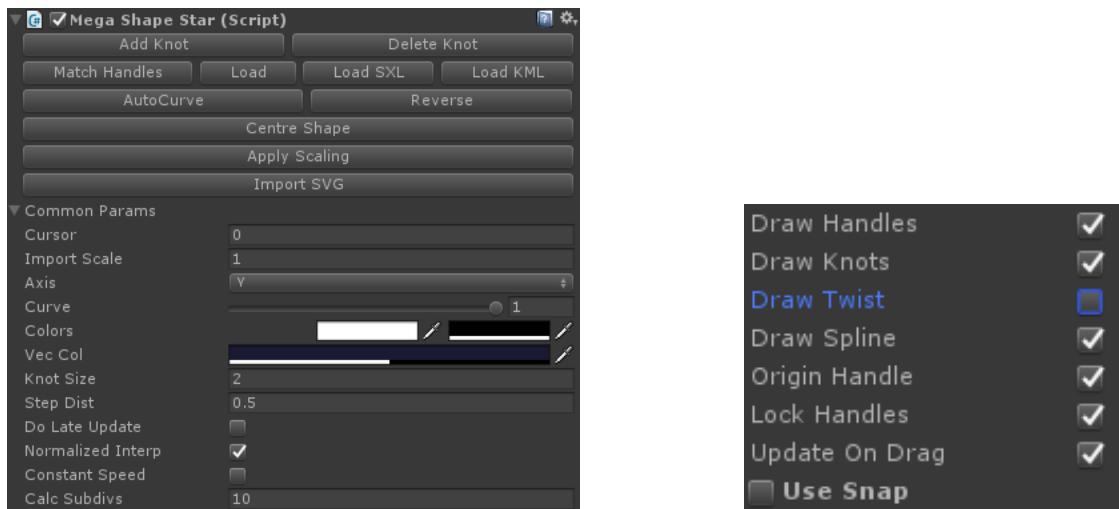
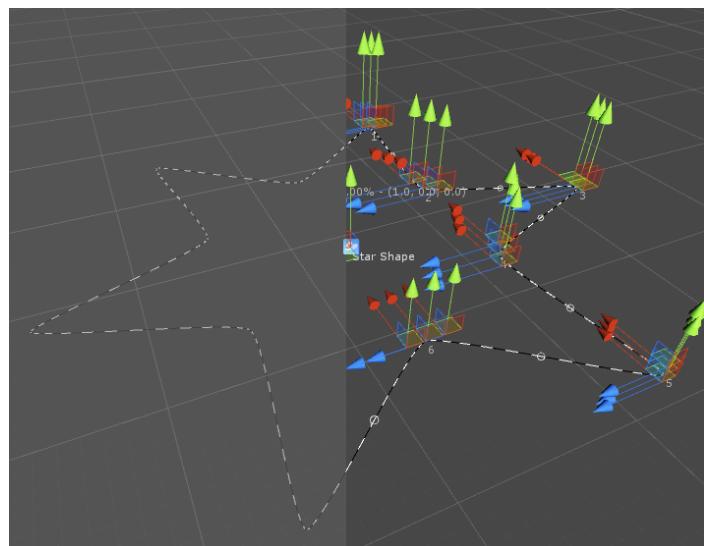


Abbildung 3.28: *Mega Shapes*: Star

Es wurde schließlich im *MegaShapes* Fenster aus Abb. 3.27 die *Star Shape* gewählt, wodurch ein *GameObject* *StarShape* in der Szene erstellt wurde, welches die Komponente *Mega Shape Star* enthält (Abb. 3.28).

(a) *Mega Shapes*: Überblick(b) *Mega Shapes* Komponente: Steuerungselemente**Abbildung 3.29:** *Mega Shapes*: Überblick

Die Mega Shape Komponente enthält unzählige Möglichkeiten, eine erstellte *Shape* zu verändern, z.B. sind weitere Anpassungen, welche an dieser Stelle mit der Stern Form möglich waren: Neue Knoten hinzufügen, alte Knoten löschen, Pivot Position Anpassen, das Ändern der Darstellung auf den Axen, sowie die Darstellung von Elementen in der Szene, um per Hand Punkte bzw. Knoten der Form zu Skalieren, Rotieren und zu Transformieren (Abb. 3.29).

**Abbildung 3.30:** *Mega Shapes*: *Star*

Da der Stern an dieser Stelle noch zu eckig war, wurde der *Smoothness* Wert in der *Mega Shape Star* Komponente auf 0.250 hochgeschoben, was zusätzliche Punkte zur Ab rundung der Kanten der Stern Form erstellt hat.

Um die Form für die Strömung später noch etwas interessanter zu machen, wurde nun eine Zweite Mega Shape hinzugefügt, nämlich ein Kreis, welcher den Stern einschließen soll.

Hierfür wurde erneut eine Mega Shape über den Assets Tab und Auswahl von *MegaShapes* ein *Circle* im *Mega Shapes* Fenster gewählt.

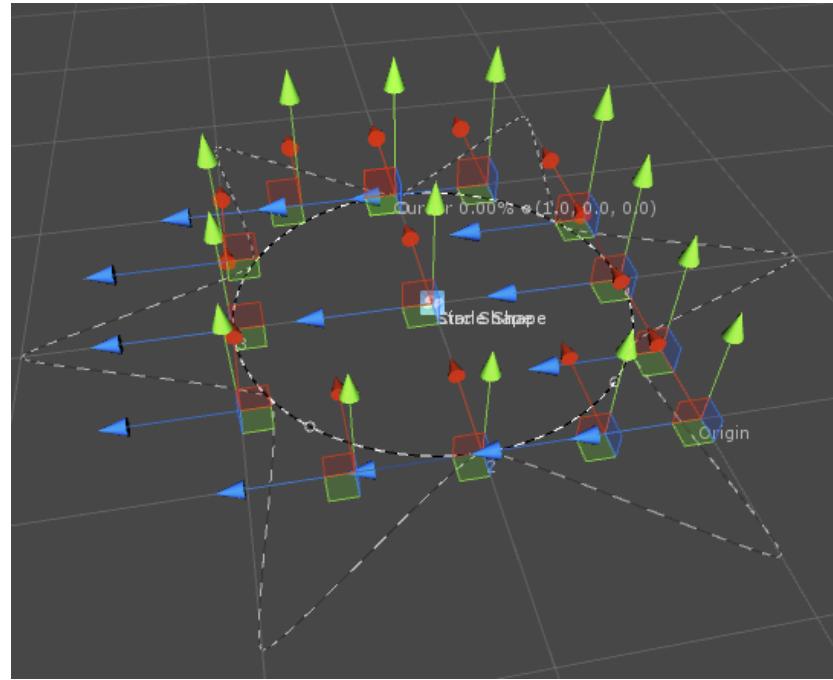


Abbildung 3.31: *Mega Shapes: Star Circle*

Die bereits erstellte Stern Form und der Kreis wurden auf den Ursprung $(0, 0, 0)$ gesetzt, sodass sie auf der gleichen Ebene liegen. Jedoch ist an dieser Stelle der Kreis noch kleiner als der Stern und befindet sich damit im Inneren (Abb. 3.31).

3.4.1 Skalierung

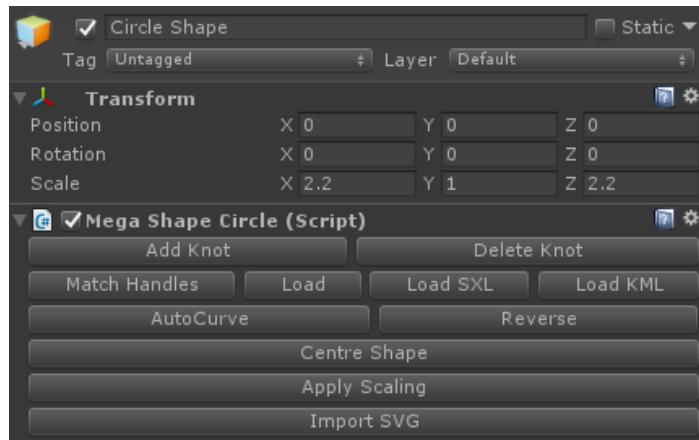


Abbildung 3.32: *Mega Shapes* Transform Element und Skalierung

In *Mega Shapes* ist es möglich zur Skalierung die Unity-eigene *Transform* Komponente zu verändern, um diese dann in die eigentlichen Knoten berechnen zu lassen. Dafür wurde die *Transform Scale* des Kreis Objekts auf $(2.2, 1, 2.2)$ hochskaliert und die Anpassungen wurden mit *Apply Scaling* (Abb. 3.32) in die Form berechnet.

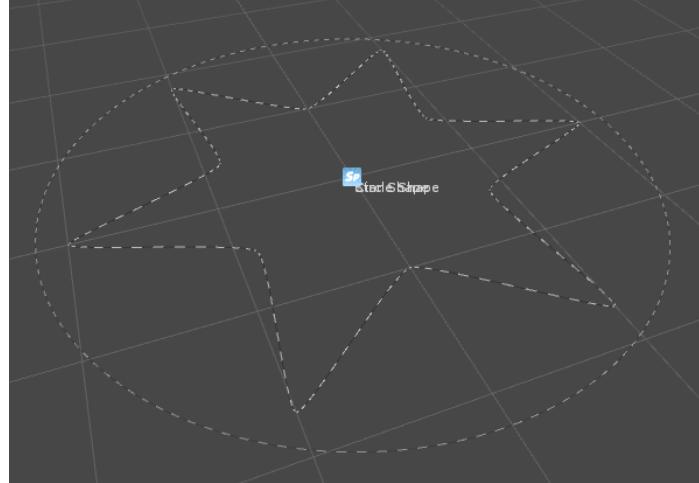


Abbildung 3.33: *MegaShapes* Transform Element und Skalierung

Nun ist die Ausgangsform für die Demoströmung erstellt und es geht darum, die *Mega Shapes* Komponente mittels *Mega Flow* Integrierung als ein Vektorfeld berechnen zu lassen. Für das Arbeiten mit den *Mega Flow* Komponenten ist es nötig, zwei neue Objekte in der Szene zu erstellen, welche *Mega Flow* Komponenten zugewiesen bekommen. Ein Objekt, welches das – letztendlich ausgegebene – verwendbare Vektorfeld beinhalten soll, sowie ein weiteres Objekt mit der Komponente, die verantwortlich für das Umwandeln der *Mega Shape* zu einer *MegaFlow Source* ist.

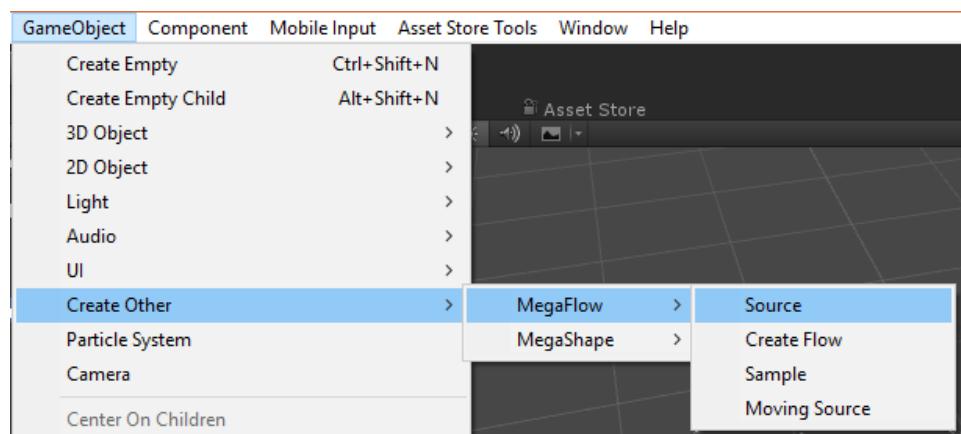


Abbildung 3.34: *Mega Shapes Transform Element und Skalierung*

Jeweilige Objekte wurden durch Auswahl von `GameObject`, `Create Other`, `MegaFlow`, `Create Flow` und `GameObject`, `Create Other`, `MegaFlow`, `Source` erstellt (Abb. 3.34).

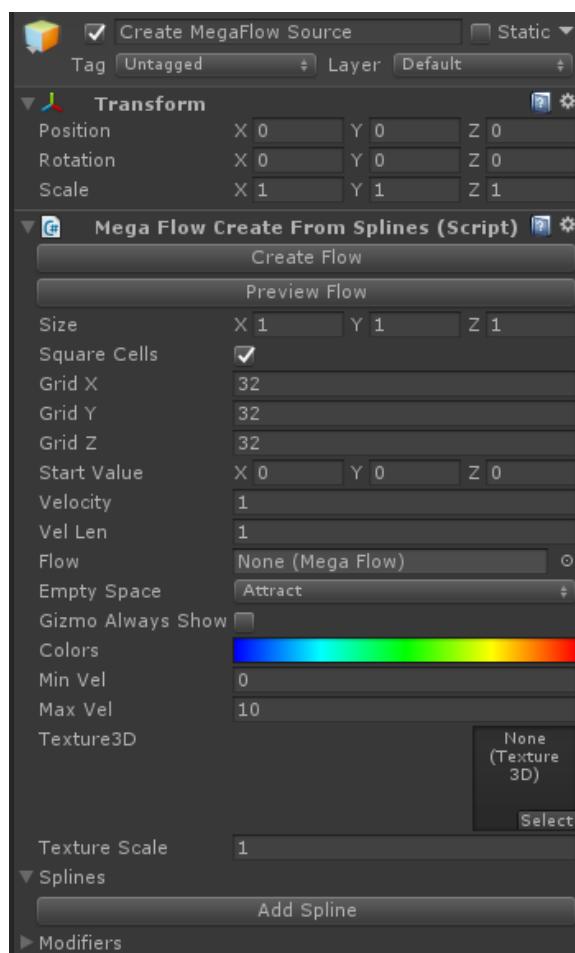


Abbildung 3.35: *Mega Flow Create Source Objekt*

Das neue Objekt Create MegaFlow Source wurde nun ebenfalls auf Position $(0, 0, 0)$ verschoben und enthält Informationen zur Erstellung des Vektorfeldes (Abb. 3.35). Mit Size und Grid X,Y,Z war es hier möglich, Größe und Genauigkeit des Rakes zu bestimmen.

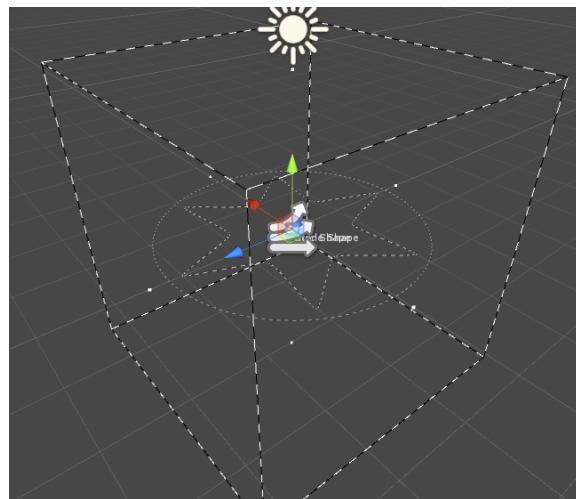


Abbildung 3.36: *Mega Flow Create Source* Objekt

Die Size des Vektorfeldes wurde auf $(5, 5, 5)$ festgelegt, was in Relation zur Grid Größe $(32, 32, 32)$ (Abb. 3.36) passt und es war nun nötig, die bereits erstellten Formen als Splines hinzuzufügen.

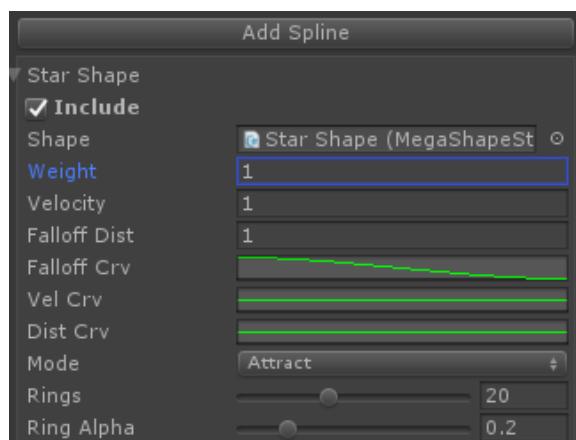


Abbildung 3.37: *MegaFlow Create Source Spline*

Durch ein Drücken auf Add Spline war es hier möglich, die erstellten Shapes einzubinden und weitere Einstellungen vorzunehmen, jedoch wurden an dieser Stelle Objekt Star Shape und Circle Shape mit den Standardeinstellungen hinzugefügt.



Abbildung 3.38: *MegaFlow Create Source Flow Verlinkung*

Nach dem Einbeziehen der Formen als Splines war es lediglich nötig, im Create MegaFlow Source Objekt unter Flow das andere Objekt, MegaFlow Source einzubinden, bevor man auf Create Flow klicken konnte um das Vektorfeld zu generieren (Abb. 3.38).

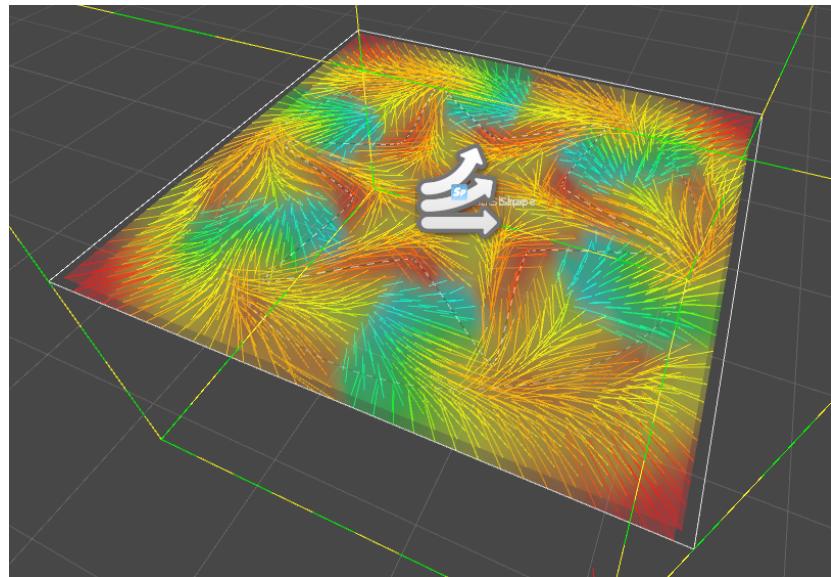


Abbildung 3.39: *MegaFlow* Vektorfeld

Nach der Berechnung und Erstellung des Vektorfeldes, ist es möglich im *MegaFlow Source* Objekt alle möglichen Einstellungen zur Visualisierung des Vektorfeldes im Unity Editor zu bestimmen.

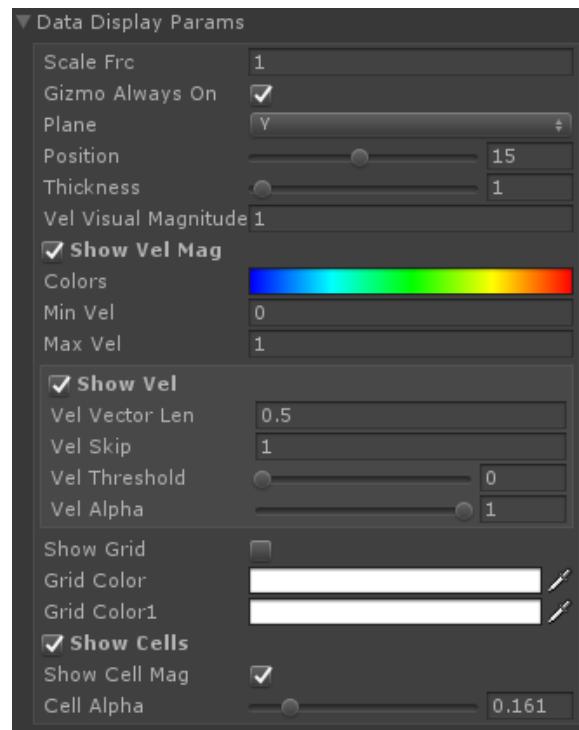


Abbildung 3.40: *MegaFlow* Data Display Params

Für die Darstellung wie in Abb. 3.39 wurden unter Data Display Params abgebildete Einstellungen festgelegt (Abb. 3.40), um eine Schicht des Feldes wie in Abb. 3.39 darzustellen.

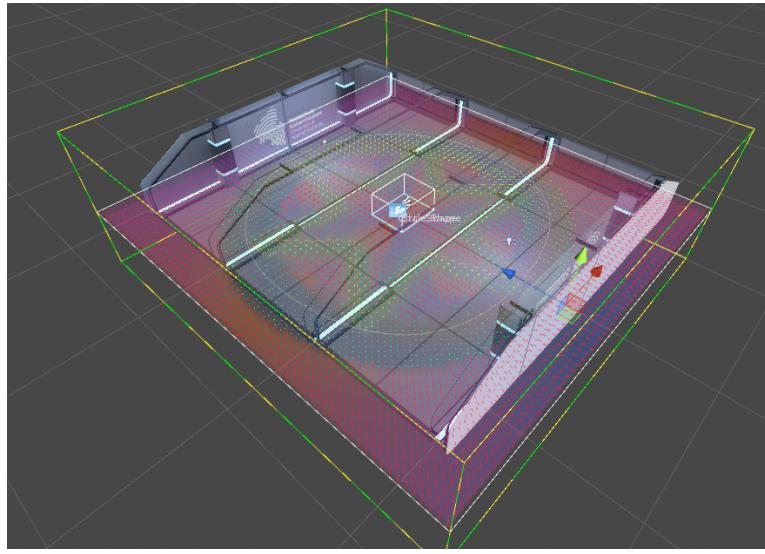


Abbildung 3.41: *MegaFlow* Resultat in Szene

Das generierte Vektorfeld wurde nun als Prefab für die Bereits erstellte Szene übernommen (Abb. 3.41) und auf die Größe des Raumes skaliert. Es wäre natürlich möglich gewesen

den gleichen Vorgang für die andere Szene zu wiederholen, jedoch wurde das Vektorfeld des erstellten Sterns lediglich zur Integration der Partikel- und Controllerfunktionen der folgenden Sktionen verwendet.

3.4.2 Trail Renderer

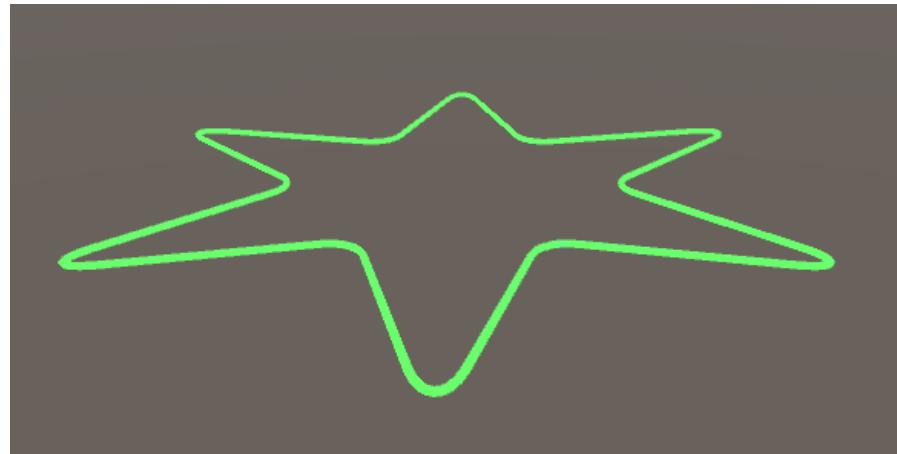


Abbildung 3.42: *MegaFlow* Darstellung des Trails

Als nächstes sollte auch während der Runtime eine Möglichkeit bestehen, das Vektorfeld bzw. die Linien der erstellten *MegaShape* zu sehen.

Dafür wurde das *Mega Path Follow* Script verwendet. Wenn die *Mega Path Follow* Komponente einem Objekt in der Szene hinzugefügt wird und als *target* die erstellte *MegaShape* (*Star Shape*) gewählt wird, ist es möglich, das Objekt den erstellten Pfad abzufahren zu lassen, wenn *Animate* gesetzt ist.

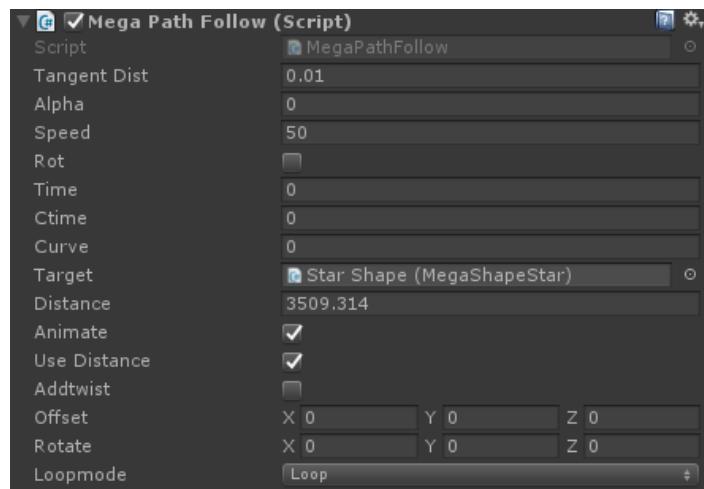


Abbildung 3.43: Trail Komponente *Mega Path Follow*

In diesem Fall wurde ein `EmptyObject` erstellt und die Komponente wurde manuell hinzugefügt. Durch angeben der Geschwindigkeit – für das Beispiel 50 – kann angepasst werden, wie schnell das Objekt dem Pfad der Shape folgt.

Da das `EmptyObject` allerdings keine `Mesh` besitzt oder ein anderes Objekt beinhaltet, das während der Runtime gerendert wird, war es nötig, etwas hinzuzufügen.

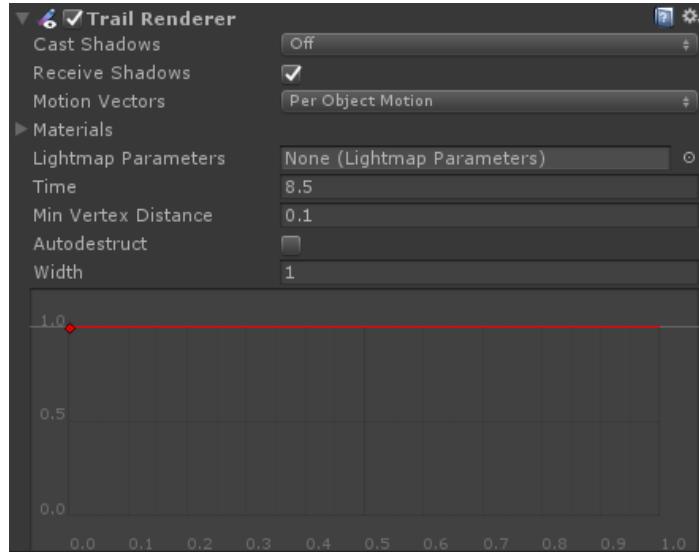


Abbildung 3.44: Trail Komponente Trail Renderer

Um prozedural einen Pfad darzustellen bzw. ein dynamisches Objekt beim Abfahren des erstellten Pfades darzustellen, ist simples Nutzen eines einfachen 3D Modells oder eines Partikels für das erstellte Objekt keine gute Idee, weshalb mit einem `Trail Renderer` gearbeitet wurde.

Ein `Trail Renderer` wird benutzt um Spuren bzw. Pfade hinter einem `GameObject` darzustellen, während es in der Szene bewegt wird.

Er ermöglicht Einstellungen für die Materialien, die von ihm benutzt werden sollen, deren Farben, sowie Breite über Zeit (`Width` und Diagramm in Abb. 3.44), die eigentliche Zeit des Renderns (`Time`) und die minimale Distanz für die Update Rate der `Trail Mesh` (`Min Vertex Distance`).

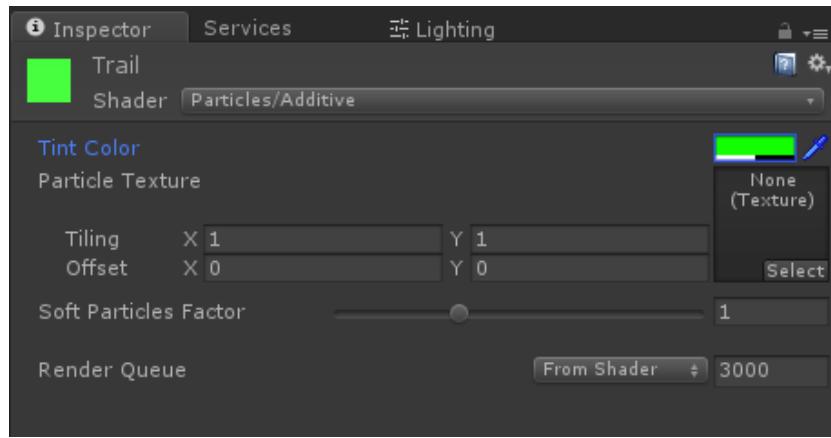


Abbildung 3.45: Erstelltes Material für Trail Renderer

Da zur Darstellung des Pfades der erstellten Form ein Material benötigt wird, wurde ein neues Material mit dem Shader `Particles/Additive` erstellt (Abb. 3.45). Ein additiver Shader erlaubt es durch setzen des Alpha Wertes bei der Farbwahl Durchlässigkeit bzw. Transparenz darzustellen.

Es wurde ein helles grün benutzt, wodurch der Trail Renderer nun einen grünen, halbdurchsichtigen Pfad wie in Abb. 3.42 darstellen kann.

Das erstellte Trail Material wurde im `Trail Renderer` angegeben und die Zeit der Darstellung wurde auf 8,5 Sekunden festgelegt, genug um in Kombination mit der Animationsgeschwindigkeit von 50 des `Mega Follow` Scripts den ganzen Stern kontinuierlich darzustellen.

3.4.3 Particle System

Anders als für den Trail Renderer – welcher lediglich die erstellte Form abfährt – ist es möglich für die Darstellung einer Strömung mittels Partikelsystem das generierte Vektorfeld zu nutzen.

Verantwortlich ist die Komponente `MegaFlowParticle`, ein Script, welches ein Unity Particle System dazu nutzt um von der erstellten Strömung affektiert zu werden.

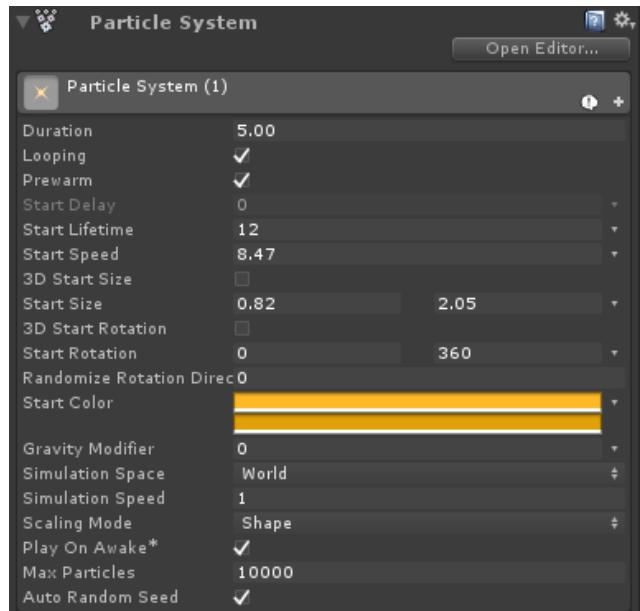


Abbildung 3.46: Particle System

Zu Beginn wurde das Particlesystem, welches in der *MegaFlow* Demo Szene zu finden war (Abb. 3.46) als Prefab für andere Szenen gespeichert. Die Szene *flowscene* befindet sich in

```
/Assets/Mega-Fiers/MegaFlow/DemoScene/
```

Jedoch wurde von dem Beispiel Particlesystem das Prefab *Star Particle* erzeugt, welches unter

```
/Resources/Particle Effects/
```

abgelegt wurde.

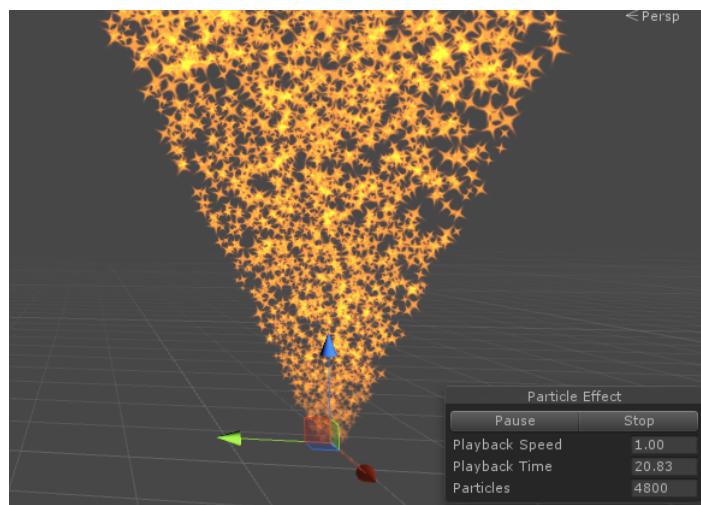


Abbildung 3.47: Particle System

Nach einfügen des Prefabs, damit das Particlesystem auch das erstellte Vektorfeld in der Szene verwendet, musste bei der `MegaFlowParticle` Komponente – welche genutzt wird um ein Particle System für *MegaFlow* zu verwenden – als Source das erstellte Vektorfeld `MegaFlow Source` angegeben werden.

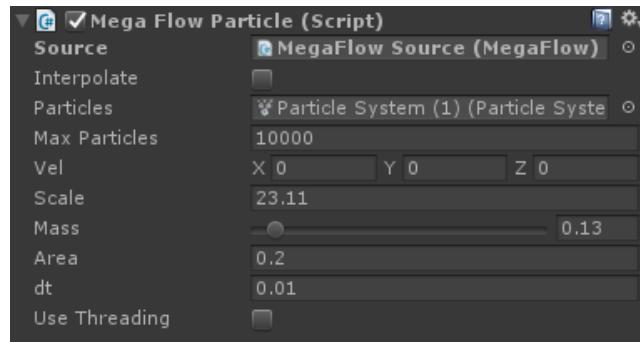


Abbildung 3.48: Particle System

Nachdem `MegaFlowParticle` die erstellte `MegaFlow Source` verwendet, ist es möglich, die Einflüsse des Vektorfeldes während der Runtime zu sehen (Abb. 3.49). Es wurde nun erfolgreich ein Partikelsystem in die Szene übernommen, welches die Strömung darstellen kann.

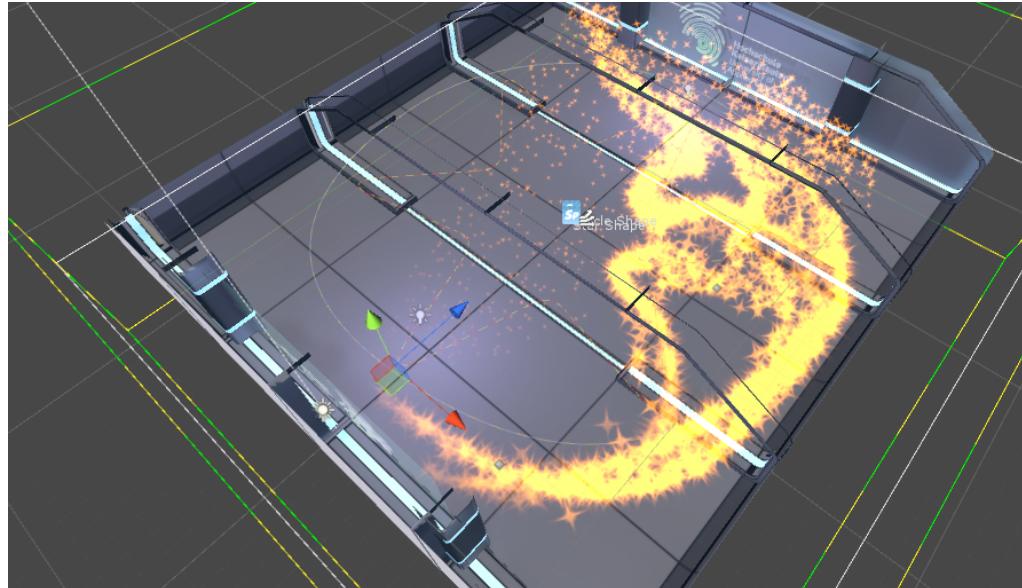


Abbildung 3.49: Particle System

3.4.4 Particle Trigger

Ziel dieses Abschnittes war es, dass die VIVE Controller verwendet werden können um während der Runtime Partikel in die Szene zu *versprühen*, sodass der User live unabhängig der Art des Vektorfeldes den Verlauf der Strömung darstellen kann.

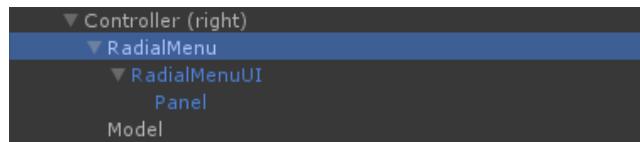


Abbildung 3.50: RadialMenu in Controller

Dem rechten Controller in der erstellten Szene wurde das VRTK Touchmenü hinzugefügt. Dafür wurde dem Controller Objekt das RadialMenu Prefab hinzugefügt (Abb. 3.50).

Assets/Assets/SteamVR_Unity_Toolkit/Prefabs/RadialMenu



Abbildung 3.51: RadialMenu in Runtime

Beim Nutzen des Touchpads werden nun mehrere Buttons eingeblendet, welche durch Drücken die Auswahl bestätigt. Dieses Menü sollte genutzt werden um eine Auswahl unterschiedlicher Partikelsysteme zu bieten (Abb. 3.51)

Durch Auswahl des `Panel` Objekts – dem untersten Child in der `RadialMenu` Hierarchie – war es möglich Einstellungen für die `RadialMenu` Komponente zu tätigen (Abb. 3.52).

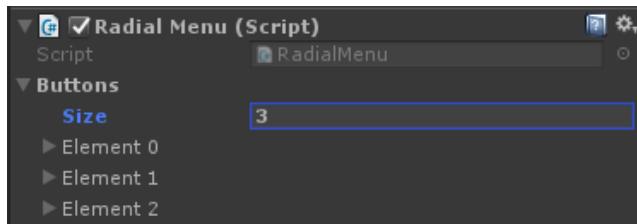


Abbildung 3.52: RadialMenu Komponente

Da den Buttons jedoch beim Klicken auch eine Funktion zugewiesen werden soll, war es nun nötig ein eigenes Script für Interaktionen mit den Partikelsystemen zu schreiben. Dafür wurde das Script `ParticleTrigger` erstellt, dessen Zweck es war, das genutzte Partikel- system durch einen Methodenaufruf des Touchinputs auszuwechseln und beim betätigen des Triggers das gewählte Partikelsystem abzuspielen.

Genutzt werden sollte eine Vielzahl an Particlesystems, weshalb ein Array benutzt wurde, sowie ein Integer Wert `ParticlePlaying` für das momentan zu verwendende Partikelsys- tem, welches durch das Touchinterface zugewiesen wird.

```
public ParticleSystem[] PS;
public int ParticlePlaying;
```

Eine public Methode `ChangeTheParticles` wurde erstellt, welche vom `RadialMenu` ge- nutzt werden sollte um den jeweiligen Button zu übergeben.

```
//Wechseln der Partikel
public void ChangeTheParticles(int i){
    foreach(ParticleSystem PSys in PS) {
        PSys.Stop();
    }
    ParticlePlaying = i;
}
```

Durch die Übergabe des Buttonwertes aus dem Touchmenü werden in dieser Methode nun sämtliche Partikelsysteme gestoppt und der Wert wird in `ParticlePlaying` gespeichert, sodass die Nummer des aktuell gewählten Partikelsystems immer in `ParticlePlaying` zu finden ist.

Dem `RadialMenu` musste jedoch noch vermittelt werden, welche Methode es Aufrufen soll um den Wert zu übergeben.

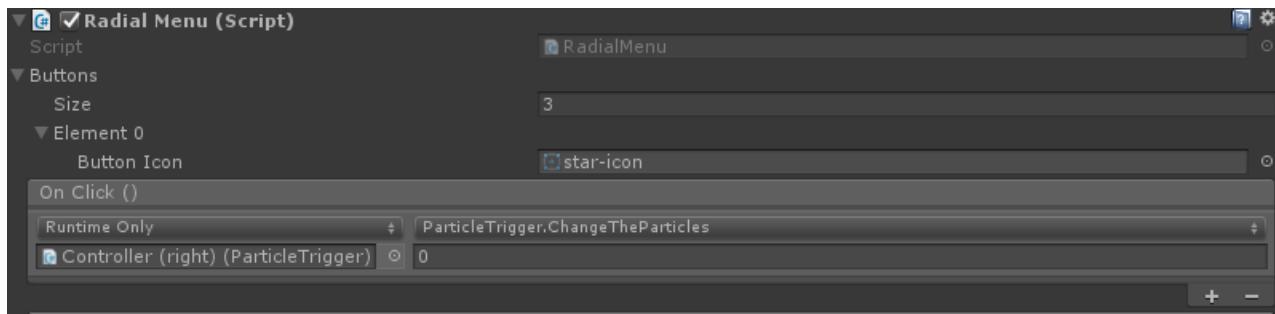


Abbildung 3.53: RadialMenu Buttons

Dafür wurde dem rechten Vive Controller Objekt (`Controller (right)`) das erstellte Script `ParticleTrigger` hinzugefügt und für jedes Button Element im `RadialMenu` ein Verweis auf das `ParticleTrigger` Script des Controllers, mit der Methode `ChangeTheParticles` angegeben.

Übergeben wird der jeweilige Wert des Elements, sodass Element 0 den Wert 0, Element 1 den Wert 1 usw. an die Methode übergibt, für das Ereignis `On Click()` des `RadialMenu`. Dadurch wurde erfolgreich festgelegt, wie die Interaktion zum Austauschen der Partikel erfolgt. Es fehlte aber, die Partikelsysteme auch wie gewollt in der Szene abzuspielen, weshalb erneut Funktionen des VRTK Plugins genutzt wurden. •• Das `ParticleTrigger` Script wurde also erweitert.

```
using VRTK;
```

Am Anfang des Scripts wurde nun angegeben, dass der Namespace `VRTK` benutzt wird um die `VRTK` Events für die Controllereingabe zu verwenden.

```
private void Start()
{
    //Setup controller event listeners
    //TRIGGER
    GetComponent<VRTK_ControllerEvents>().TriggerPressed += 
        new ControllerInteractionEventHandler(DoTriggerPressed);
    GetComponent<VRTK_ControllerEvents>().TriggerReleased += 
        new ControllerInteractionEventHandler(DoTriggerReleased);
    //Button
    GetComponent<VRTK_ControllerEvents>().ApplicationMenuPressed += 
        new ControllerInteractionEventHandler(DoApplicationMenuPressed);
```

In der `Start()` Methode waren diese Aufrufe nötig, um die Events für das Drücken und Loslassen des Triggers, sowie das Drücken des Menü Buttons aufzufangen.

```
//Trigger gedrueckt => spiele Partikel ab
private void DoTriggerPressed(object sender,
                               ControllerInteractionEventArgs e)
{
    PS[ParticlePlaying].Play();
}
```

```
//Trigger gelöst
private void DoTriggerReleased(object sender,
                                ControllerInteractionEventArgs e)
{
    PS[ParticlePlaying].Stop();
}
```

Diese Methoden wurden nun beim Drücken und Lösen des Triggers aufgerufen und sie spielen das gewählte Partikelsystem ab bzw. stoppen es beim Aufruf. Alles was jetzt noch nötig war um das erstellte Menü mit den Funktionen zu verwenden war, ein tatsächliches Partikelsystem einzubinden.

Dafür wurde das in Kapitel 3.4.3 verwendete Partikelsystem benutzt und abgeändert, da die Größe der Partikel viel zu hoch war.

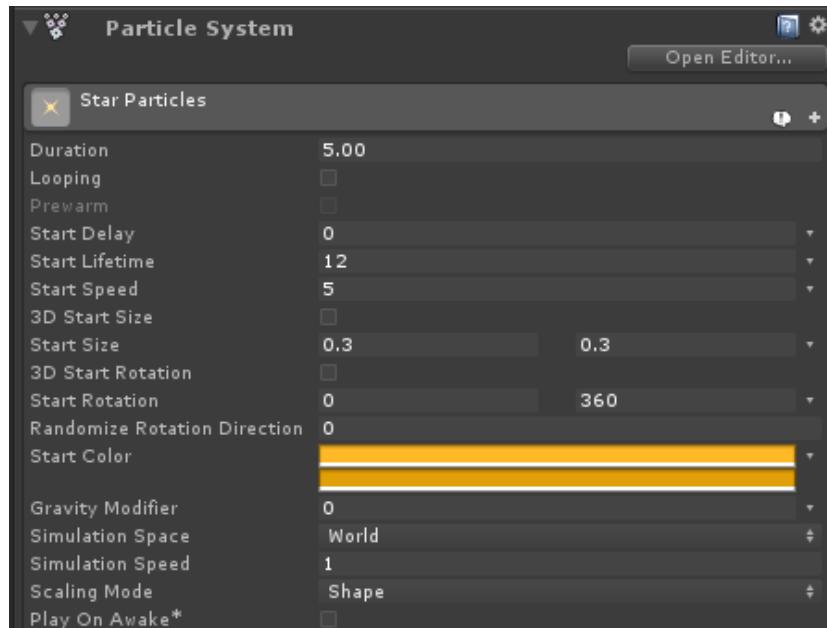


Abbildung 3.54: Particlesystem Komponente

Zusätzlich war es nötig Play On Awake zu deaktivieren (Abb. 3.54), da das Particlesystem sonst beim Starten der Szene abgespielt werden würde, nicht erst beim Drücken des Triggers.



Abbildung 3.55: Controller (right) Hierarchie

Drei Kopien des Particlesystems wurden dem rechten Controller untergewiesen und in Controllermitte zentriert, lokale Position $(0, 0, 0)$ (Abb. 3.55).

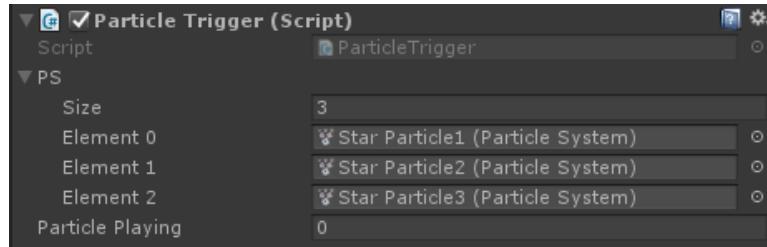


Abbildung 3.56: ParticleTrigger

Auf dem rechten Controller konnten die Partikelsysteme nun der erstellten Komponente ParticleTrigger zugewiesen werden (Abb. 3.56).



Abbildung 3.57: Partikelsystem an Controller

Während der Runtime konnten schließlich Partikel vom rechten Controller ausgehend *versprüht* werden und deren Einfluss auf die Szene beobachtet werden (Abb. 3.57).

Trail Renderer Script Methode

```
//Menu Button gedrueckt
private void DoApplicationMenuPressed(object sender,
                                      ControllerInteractionEventArgs e)
{
    Traill.SetActive(!Traill.activeSelf);
}
```

Um den zuvor erstellten Trail Renderer, der die Sternform darstellt über den Menübutton optional ein- und auszublenden, wurde das `ParticleTrigger` Script um diese Methode erweitert, wobei `Trail1` ein Verweis auf das erstellte Trailobjekt in der Szene ist.

Trail Particle

Die zwei weiteren Kopien des Partikelsystems wurden so angepasst, dass die Partikel Trails hinter sich herziehen, wofür das bereits erstellte Trail Material einmal in grün und einmal in rot benutzt wurde.



Abbildung 3.58: Traileigenschaften des Partikelsystems

Das Ergebnis war Rendern der Trails für die einzelnen Partikel während der Runtime.

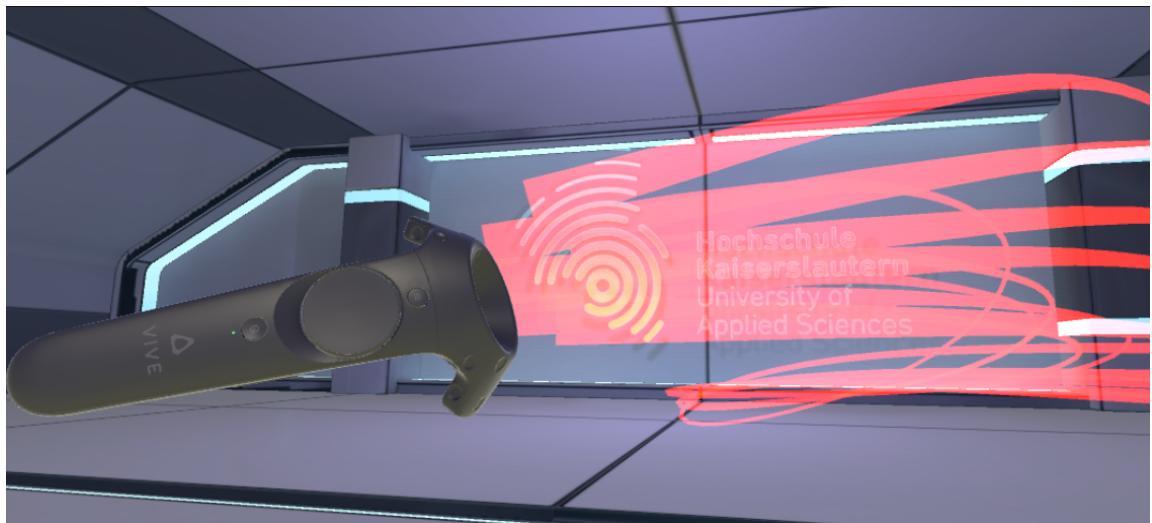


Abbildung 3.59: Trail Partikelsystem in der Szene

3.5 Küchenszene

3.5.1 Import VTK Daten in Unity

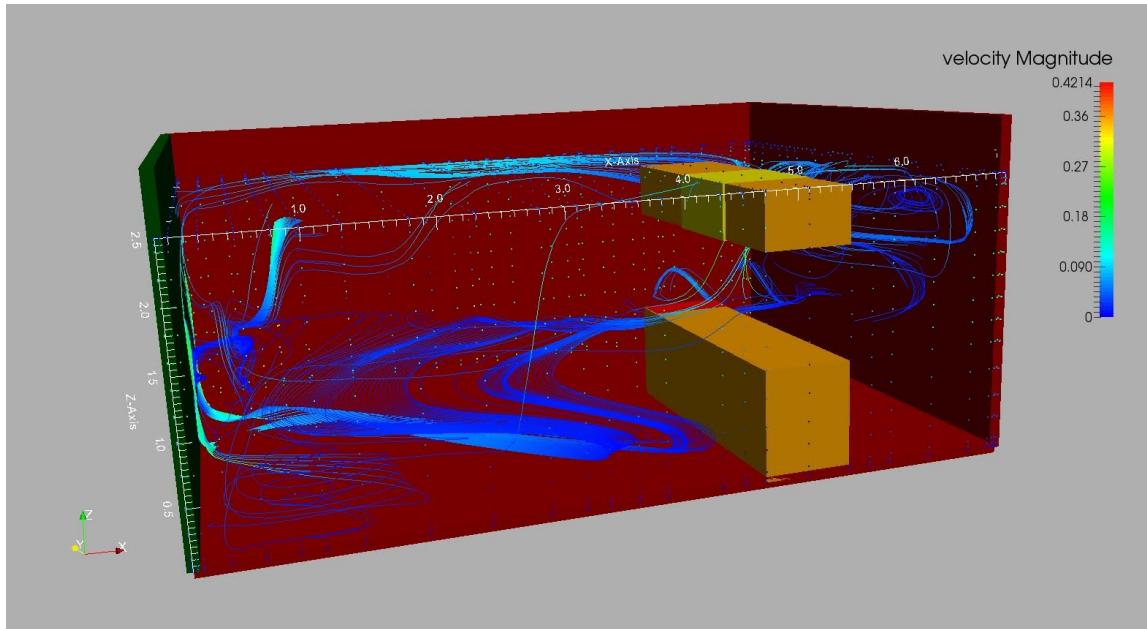


Abbildung 3.60: ParaView Darstellung der VTK Küchendaten

Zur weiteren Arbeit der Bachelorarbeit wurden echte Strömungsdaten einer Küche genommen, die aus existierenden VTK Daten (The Visualization Toolkit [Kit16]) entsprungen sind. Jedoch mussten diese Daten erst etwas angepasst werden, um von Mega Flow gelesen zu werden.

```
//VTK Daten
DATASET STRUCTURED_GRID
DIMENSIONS 28 24 17
POINTS 11424 float

POINT_DATA 11424
VECTORS velocity float
-0.0062880628 -0.006037035 0.0123251
0.020995639 -0.01016393 0.0033478111
0.0066849641 -0.001696447 0.004561712
0.00028177281 0.002587439 -0.002266933
...
```

Damit MegaFlow in Unity die Daten lesen konnte, mussten diese in einer .FLW Datei im XML Format gespeichert werden. Hierfür wurde die *Suchen und Ersetzen* Funktion in Notepad++ [Ho16] genutzt um alle Leerzeichen für die Trennung der Vektor Werte mit Komma zu ersetzen.

```
//FLW Daten
<Fluid grid="28,24,17" size="0.0,0.0,0.0,3.6,3.1,2.5">
    <Vel data="-0.0062880628,-0.006037035,0.0123251
               ,0.020995639,-0.01016393,0.0033478111
               "/>
</Fluid>
```

Die Strömungsdaten wurden auf diese Art als FLW Datei gespeichert und werden in den nächsten Abschnitten in Unity verwendet. Die Größe (Size) lässt sich in diesem Fall durch die Maße der Vive Playarea des VR-Labors bestimmen, da für die Küche eine Szene ohne Teleportfunktion realisiert werden sollte.

3.5.2 Erstellung Küchenszene

Um die in Kapitel 3.5.1 erstellte FLW Datei zu importieren, musste zunächst in einer neuen Szene ein `MegaFlow Source` GameObject erstellt werden (`GameObject`, `Create Other`, `MegaFlow, Source`).

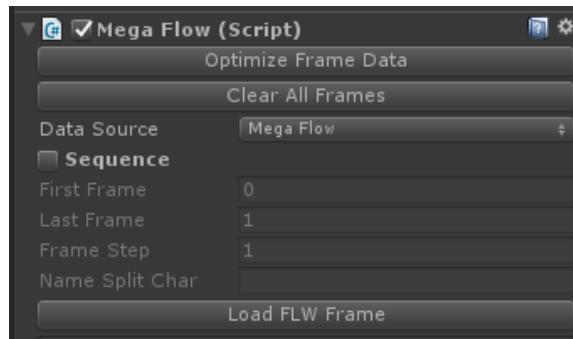


Abbildung 3.61: MegaFlow Source Komponente

Dort wurde über `Load FLW Frame` die erstellte Datei `Kitchen.flw` abgerufen (Abb. 3.61). Diese befindet sich ebenfalls im Unityprojekt, dies ist für den Import jedoch nicht nötig.

`Assets\Resources\FLW\Kitchen.flw`

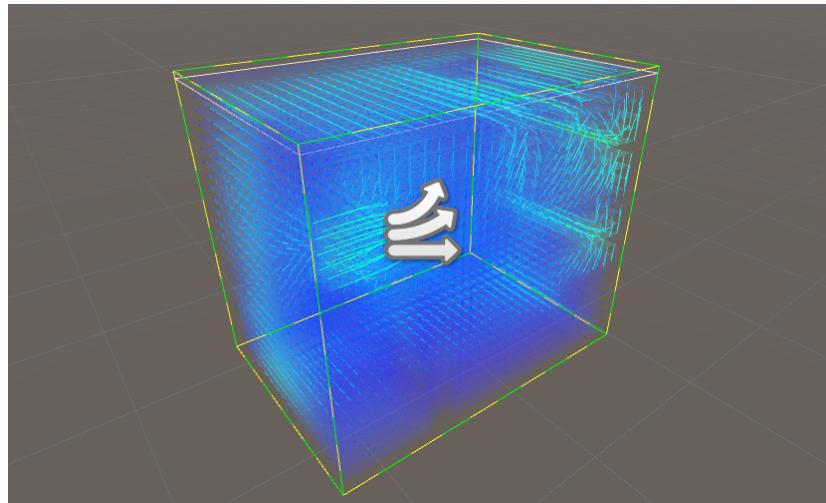


Abbildung 3.62: MegaFlow Küche

Wie auch zuvor, war es an dieser Stelle möglich, die Data Display Params der MegaFlow Komponente anzupassen um eine Darstellung wie in Abb. 3.62 zu bekommen. Da bei den genutzten VTK Daten die Z-Achse nach oben zeigt, in Unity jedoch die Y-Achse, war es nötig das MegaFlow Source Objekt um 90° auf der X-Achse zu rotieren.

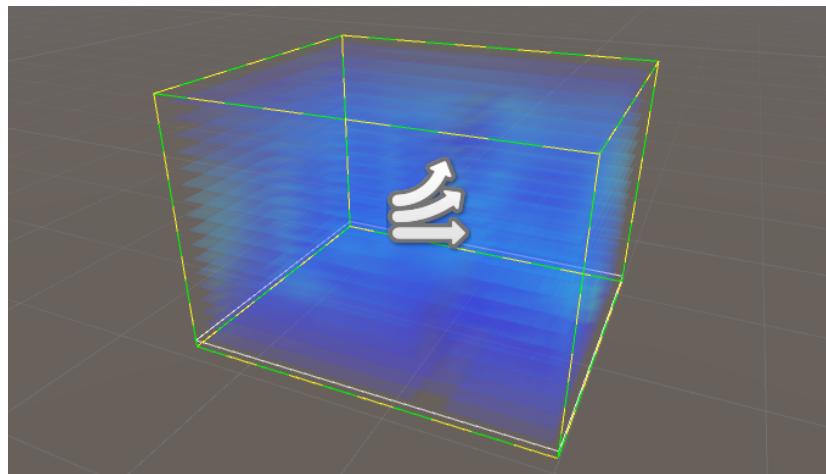


Abbildung 3.63: MegaFlow Küche, rotiert

Um die Küche in einem Raum mit Wänden, Boden und Decke darzustellen, wurden in Unity sechs Cubes erstellt, welche so transformiert und skaliert wurden, dass sie einen Raum mit den Maßen $10\text{m} \times 10\text{m} \times 10\text{m}$ umschließen konnten.

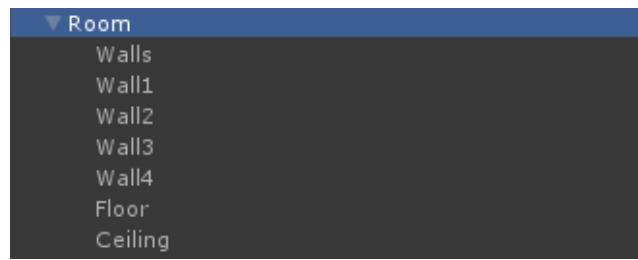


Abbildung 3.64: Küchengeometrie in Hierarchie

Anschließend wurden diese einem `EmptyObject` *Room* als Child Objects untergeordnet (Abb. 3.64).

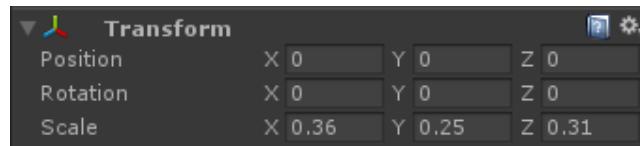


Abbildung 3.65: *Room* Objekt Transform

Um den erstellten Würfel aus Wänden, Boden und Decke auf die Maße der Küchendaten zu skalieren, wurde das Parent Object *Room* auf (0.36, 0.25, 0.31) festgelegt, so dass Länge, Höhe und Breite der Wände mit den Maßen des Vektorfeldes übereinstimmen (Abb. 3.65).

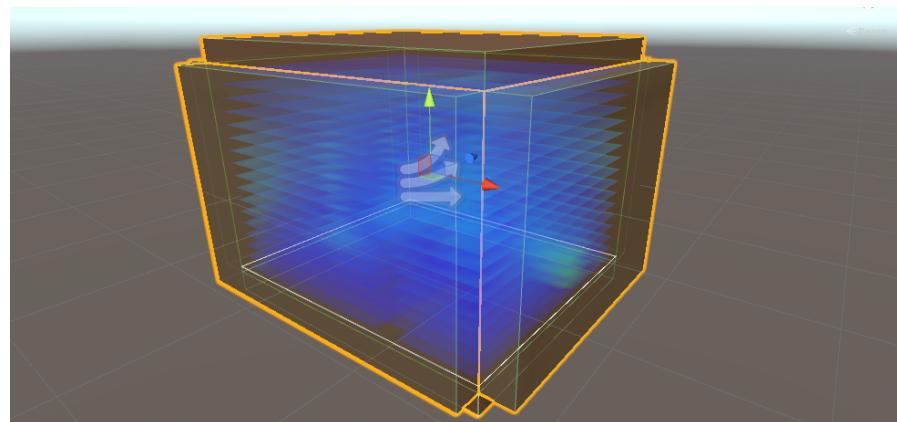
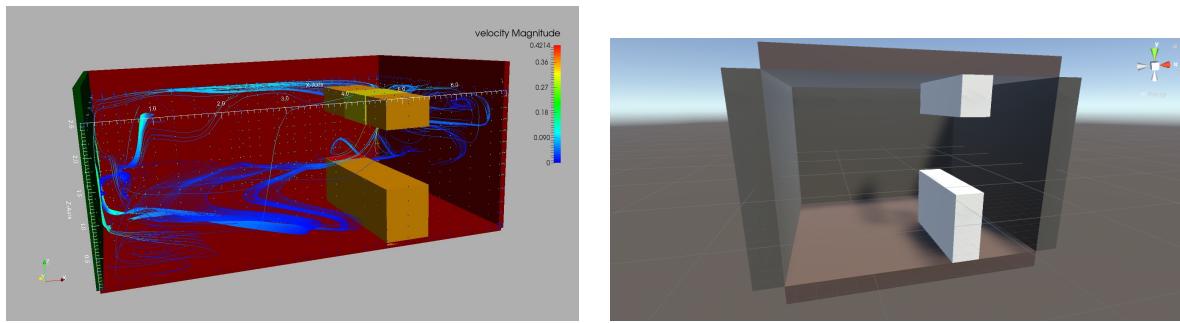


Abbildung 3.66: Küchengeometrie in Szene



(a) ParaView Abbildung

(b) Unity Szene

Abbildung 3.67: Vergleich Geometrie bei Skalierung

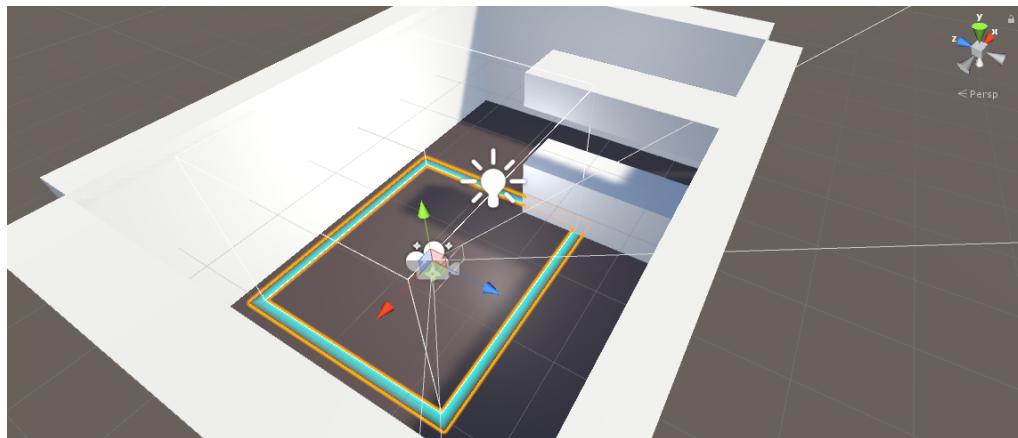
Zusätzlich wurde die restliche Geometrie des Raumes eingefügt, Theke und Arbeitsfläche. Diese wirken jedoch gestreckt, da sie als Objekte des Raumes ebenfalls auf die eigentlichen Raummaße (3,6m x 3,1m) skaliert wurden (Abb. 3.67).

Küche Vive Integration

Zunächst ist das zuvor gespeicherte Prefab [CameraRig] aus der alten Szene in die Küche übernommen worden.

Assets/Resources/ [CameraRig]

Es beinhaltet die Vive Komponenten und die erstellten Partikelsysteme, sowie die erstellten Interaktionen der Vive Controller.

**Abbildung 3.68:** [CameraRig] in Küchenszene

Da die Play Area der Vive standardmäßig auf *Calibrated* steht um Werte vom jeweiligen PC aufzunehmen, die Küche jedoch im fest definierten Raum stattfinden soll, ist es nötig die Playarea Größe ebenfalls auf 3,6m x 3,1m zu stellen. Diese Werte sind von sich aus nicht verfügbar, weshalb es nötig war, SteamVR_PlayArea zu bearbeiten und einen weiteren Eintrag unter `public enum Size` für die Maße des VR Labors anzulegen.

```
public class SteamVR_PlayArea : MonoBehaviour
{
    public float borderThickness = 0.15f;
    public float wireframeHeight = 2.0f;
    public bool drawWireframeWhenSelectedOnly = false;
    public bool drawInGame = true;

    public enum Size
    {
        Calibrated,
        _400x300,
        _360x310, //Neuer Eintrag
        _300x225,
        _200x150
    }
}
```

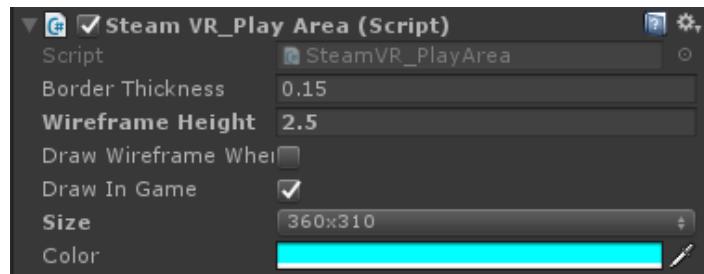


Abbildung 3.69: SteamVR_PlayArea Komponente

Danach war es möglich die definierte Größe der *Playarea* auszuwählen (Abb 3.69).

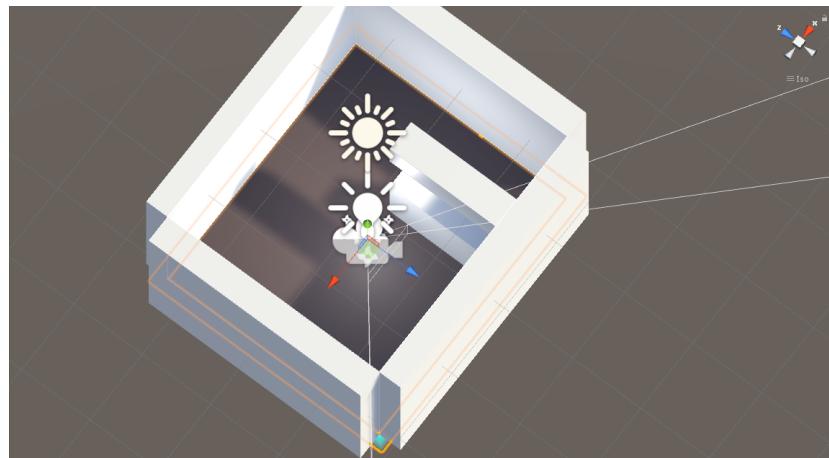


Abbildung 3.70: *Playarea* in Küchenszene

Nachdem dies erfolgreich bestimmt wurde, schließt die *Playarea* den ganzen Raum ein (Abb 3.70).

Rauch Partikelsystem

Da ein interessantes und authentisches Partikelsystem, das im Raum benutzt werden könnte Rauch ist, wurde hier nun eines der bereits genutzten Stern Partikelsysteme zu Rauch Partikeln abgeändert.



Abbildung 3.71: White Smoke Particle System Asset

Als Textur bzw. Material Ressource wurde ein weiteres Unity Asset hinzugefügt, White Smoke Particle System. Es wurde eine weitere Kopie des Star Particle Objekts angelegt und es wurden Anpassungen an Start Speed, Start Rotation und den Farben vorgenommen (Abb. 3.72).

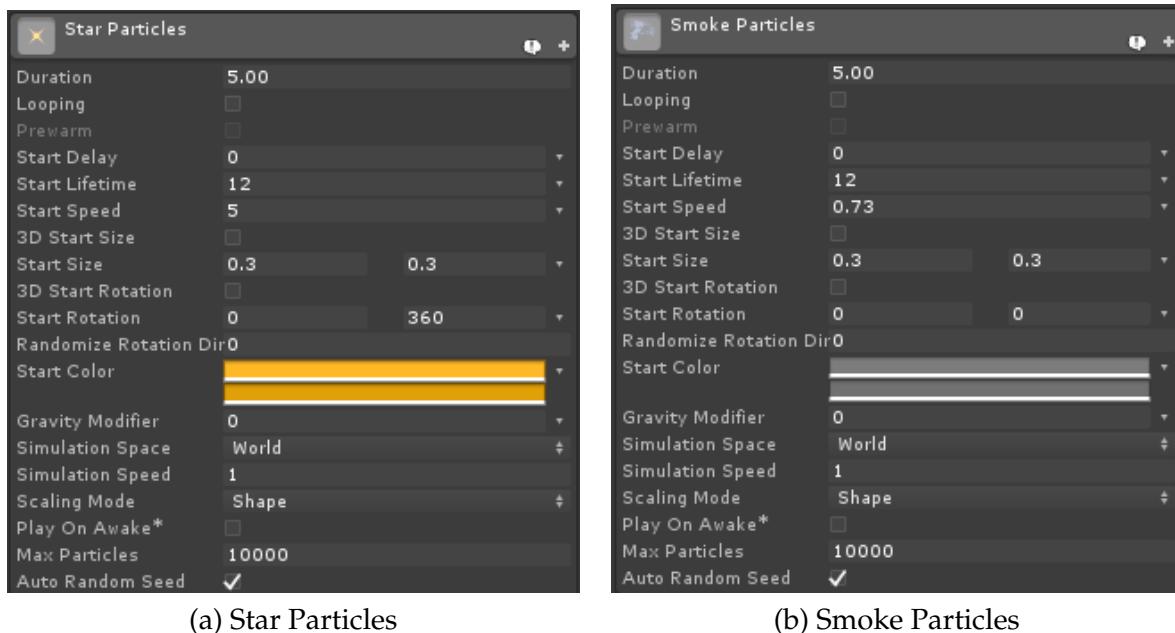


Abbildung 3.72: Unterschiedliche Einstellungen der Partikelsysteme

Zusätzlich wurde für die Partikel das *Smoke* Material aus dem importierten Asset gewählt (Abb. 3.73).

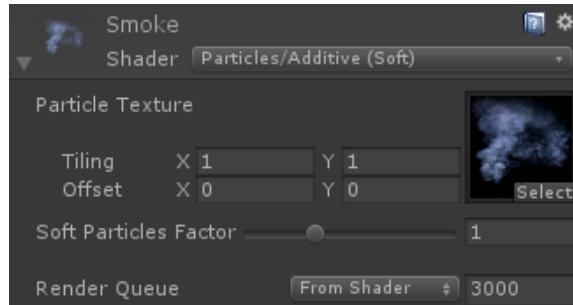
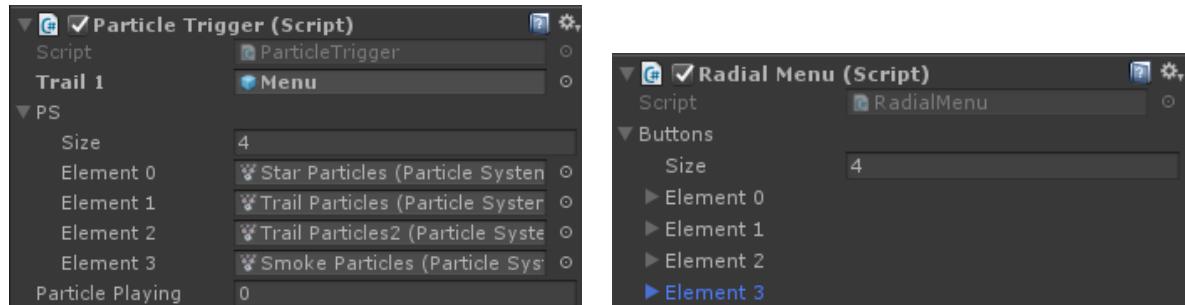


Abbildung 3.73: *Smoke* Material

Zum Hinzufügen in der existierenden *ParticleTrigger* Komponente des rechten Controllers wurde das Rauchobjekt als viertes Element für das PS Array angegeben, sowie im RadialMenu, um das Betätigen des Touch Buttons zu verlinken.(Abb. 3.74).



(a) Particle Trigger

(b) Radial Menu

Abbildung 3.74: Updates um weiteres Partikelsystem hinzuzufügen

3.5.3 Physikalisches Regler Menü

Zum Modifizieren der Partikelsysteme während der Runtime – Ändern von Größe und Masse der Partikel – wurde ein weiteres Asset zur Implementierung eines zusätzlichen Menüs importiert, *VRBasics*.

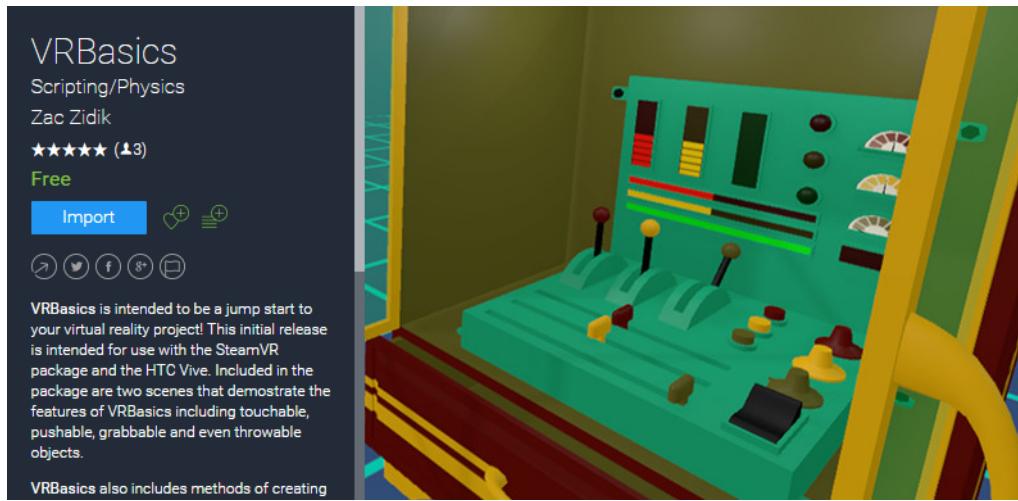


Abbildung 3.75: VRBasics Asset

VRBasics [Zac16] bietet eine einfache Umsetzung von physikalischen Interaktionsmöglichkeiten für die Vive Controller, wie etwa Knöpfe, Hebel und Schieberegler. Zum Nutzen der VRBasics Objekte, war es nötig, eine weitere Ebene mit dem Namen *Ignore Collisions* hinzuzufügen, dafür wurde unter Tags and Layers (Edit, Project Settings, Tags and Layers) die User Layer 8 beschrieben (Abb. 3.76).

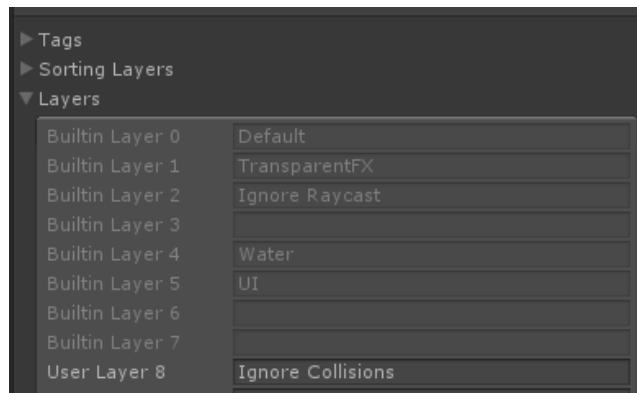


Abbildung 3.76: Tags and Layers

Danach musste im *PhysicsManager* (Edit, Project Settings, Physics) in der Kollisionsmatrix die Kollisionen der neu erstellten Ebene mit anderen Ebenen deaktiviert werden (Abb. 3.77).

▼ Layer Collision Matrix						
	Default	TransparentFX	Ignore Raycast	Water	UI	Ignore Collisions
Default	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TransparentFX	<input checked="" type="checkbox"/>					
Ignore Raycast	<input checked="" type="checkbox"/>					
Water	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
UI	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ignore Collisions	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abbildung 3.77: Kollisionsmatrix, PhysicsManager

Dies war nötig, dass es zu keinen ungewollten Kollisionen zwischen der Komponente, die dem Controller hinzugefügt wird und der Umgebung kommen kann.

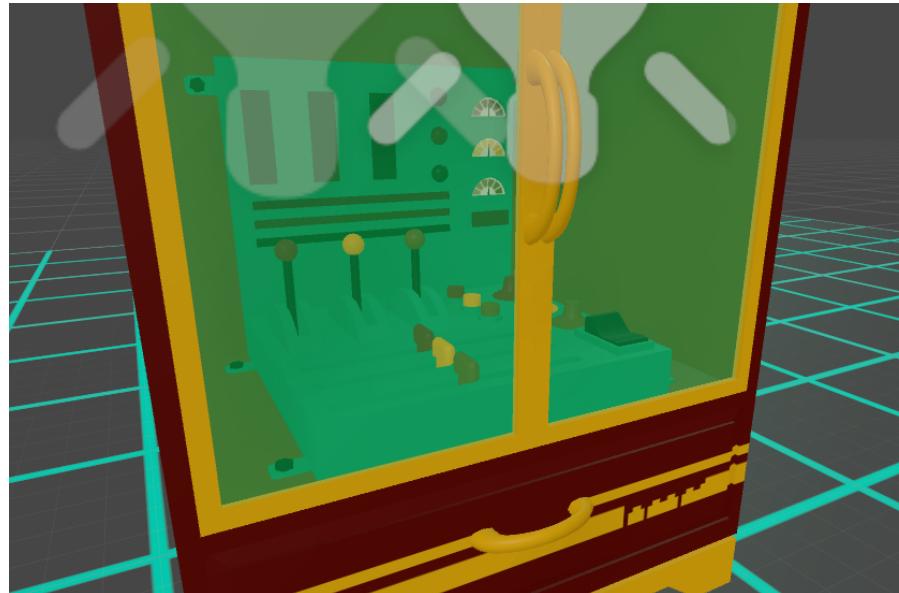


Abbildung 3.78: *demoStation* mit Controlpanel

Nun wurde die Szene *demoStation*(Abb. 3.78) aus *VRBasics*

Assets/VRBasics/Examples/Scenes/demoStation

geladen und dem Controlpanel in der Szene wurde der Schieberegler `rail_greenSlider` entnommen und kopiert.

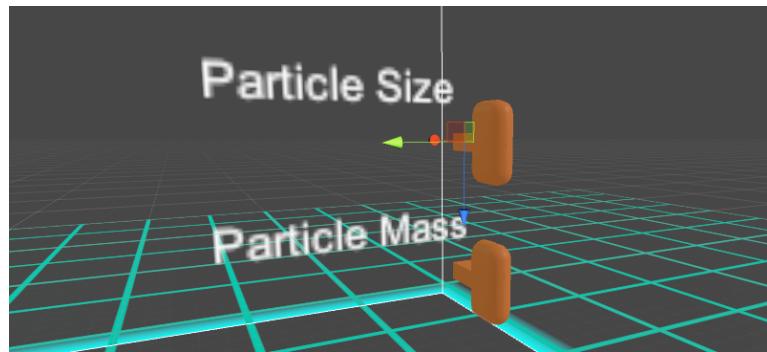


Abbildung 3.79: Erstellung Reglermenü

Der Schieberegler wurde ein weiteres Mal dupliziert und zwei Versionen wurden übereinander angeordnet um ein Menü zu Bilden. Per Unity Text Objekt wurden Überschriften für die Regler hinzugefügt, welche die späteren Funktionen Beschreiben (Abb. 3.79).

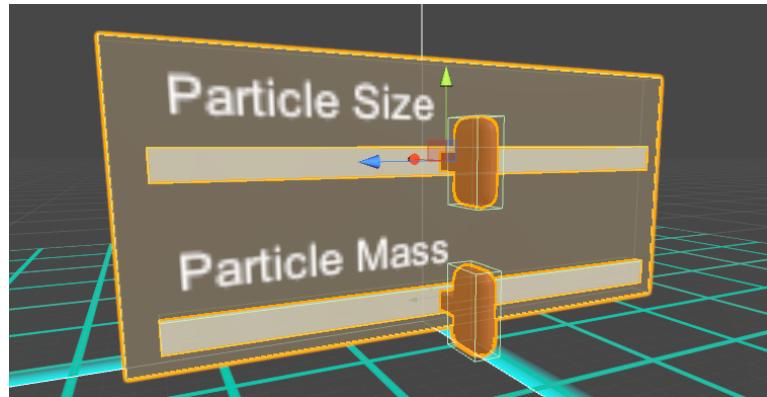


Abbildung 3.80: Erstellung Reglermenü

Durch Erstellen und Skalieren von drei Cube Objekten wurde das dreidimensionale Menü fertig gestellt und als Prefab für die Küchenszene gespeichert.

Assets/Resources/Menu

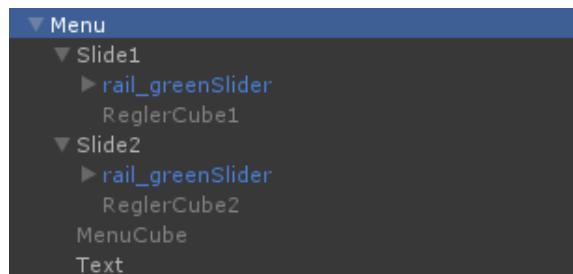


Abbildung 3.81: Hierarchie und Anordnung der Objekte des Menüs

3.5.4 Regler in Küchenszene

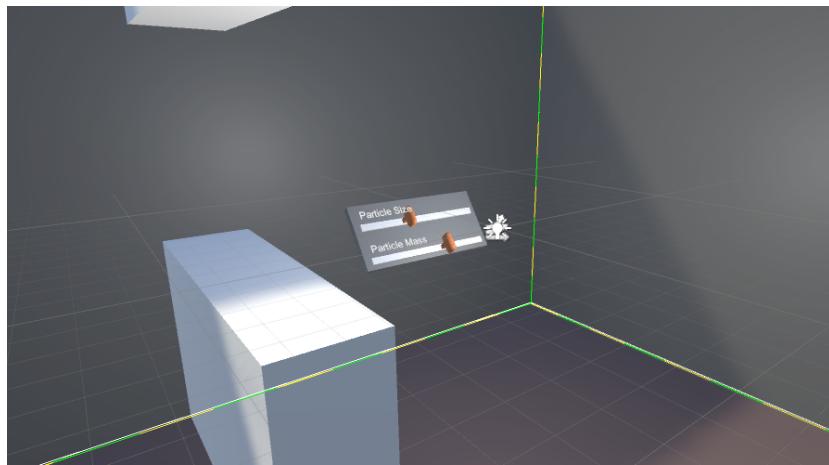


Abbildung 3.82: Menü in der Küchenszene

Das erstellte Menü wurde nun in die Mitte der Küche auf Oberkörperhöhe gesetzt und um -35° auf der Z-Achse rotiert. Damit *VRBasics Objekte* in der Szene funktionieren, musste jedoch noch das Prefab *VRBasics*

Assets\VRBasics\Prefabs

in die Szene geladen werden und das Prefab *Toucher*

Assets/VRBasics/Prefabs/Widgets/Toucher

dem linken Controller untergeordnet werden (Abb. 3.83).



Abbildung 3.83: Untergeordnetes Prefab *Toucher* in Hierarchie



Abbildung 3.84: VRTK Teleport Komponente

Auf dem linken Controller konnte die Komponente *VRTK_Bezier Pointer*, der Teleporter für die zuvor erstellte Szene deaktiviert werden (Abb. 3.84) und die Komponente *VR Basics_Controller* hinzugefügt werden (Abb. 3.85).

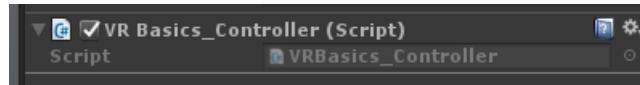


Abbildung 3.85: VRBasics Komponente

Da die *Slider* Rigidbody Objekte sind, würden sie ohne passende Einstellung beim Verschieben noch *nachziehen* und weiter mit der Einwirkung gleiten, weshalb der *Drag* Wert des Rigidbody *slider* erhöht werden musste.

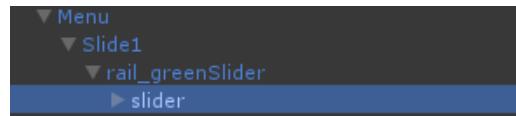


Abbildung 3.86: slider im Menü Objekt

Dafür wurde für jeden Regler jeweils das *slider* Objekt (Abb. 3.86) gewählt und deren *Drag* Wert in der Rigidbody Komponente auf 100 erhöht (Abb. 3.87).

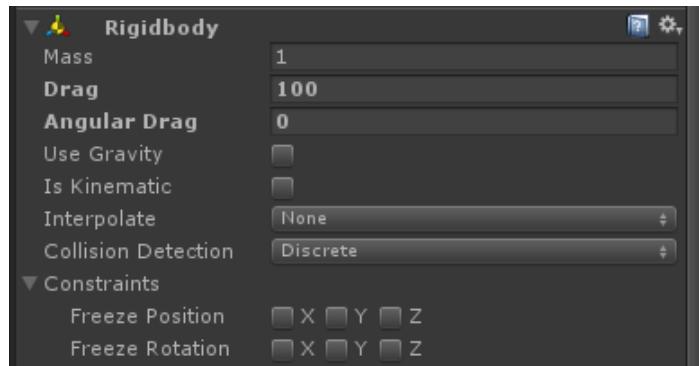


Abbildung 3.87: Rigidbody Komponente von slider

Jetzt war es noch nötig, das *ParticleTrigger* Script zu erweitern, sodass die Änderungen an der Reglerposition auf die Partikelsysteme übernommen werden konnten.

```
//Slider der UI
public VRBasics_Slider slider_PS_Size;
public VRBasics_Slider slider_PS_Mass;
```

Am Anfang des Scripts wurden nun *Slider* angelegt um die benutzten Reglerwerte zu bekommen.

```
//Partikelgr e ber Slider
void ChangeSize(float s)
{
    var ma = PS[ParticlePlaying].main;
    ma.startSize = s;
}
```

Die erstellte Methode *ChangeSize* passt die Größe der Partikel an den übergebenen float Wert *s* an. Seit Unity 5.5 erfolgt dies über `ParticleSystem.main` [Uni16d].

```
//Masse zum Slider hin anpassen
void ChangeMass(float s)
{
    PS[ParticlePlaying].gameObject
        .GetComponent<MegaFlowParticle>().mass = s / 12;
}
```

Die erstellte Methode *ChangeMass* passt die Masse der Partikel an ein Zwölftel von dem übergebenen Wert *s* an. Es wurde der Faktor 12 gewählt, dass ein maximaler Wert von 0.083 möglich ist, da das Verhalten einer größeren Masse sich unwesentlich ändert.

```
void LateUpdate()
{
    ChangeSize(slider_PS_Size.percentage);
    ChangeMass(slider_PS_Mass.percentage);
}
```

Zu guter letzt wurde *LateUpdate* hinzugefügt. *LateUpdate* wird jeden Frame ausgeführt und ruft die beiden erstellten Methoden zur Übergabe der Reglerwerte auf.

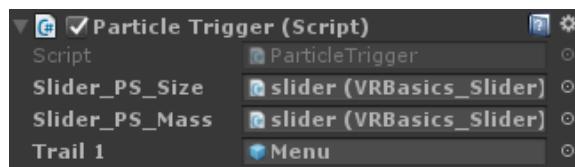


Abbildung 3.88: Angepasste *ParticleTrigger* Komponente

Die public Objekte konnten schließlich angegeben werden und *Trail1* wurde statt mit dem Trail des Sterns aus der vorigen Szene nun mit dem Menü gefüllt, um Ein- und Ausblenden des Menüs mit den Vive Controller Menu Buttons zu ermöglichen (Abb. 3.88).

3.5.5 Tooltips

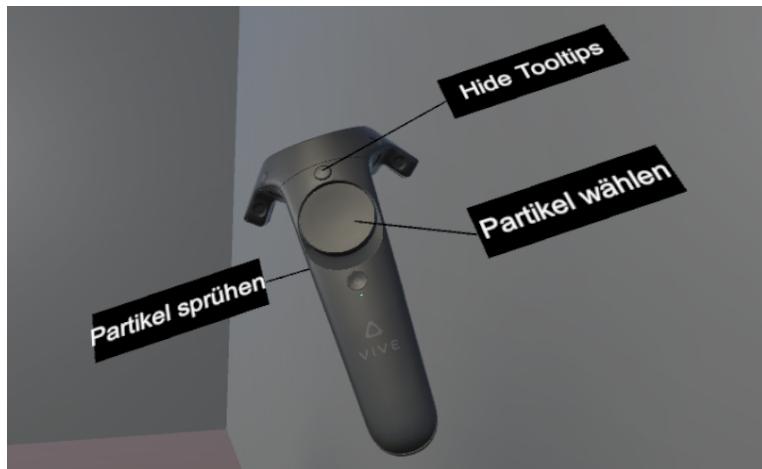
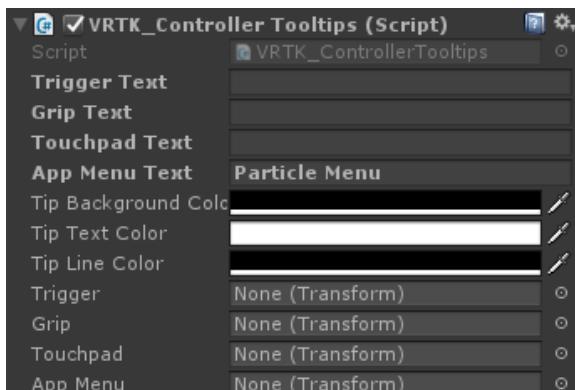
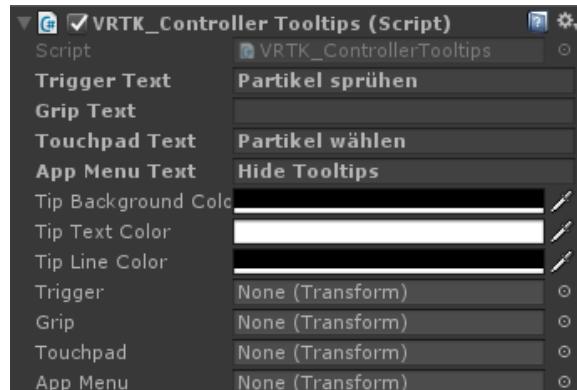


Abbildung 3.89: Tooltips in Szene

Abschließend, dass Nutzer der Anwendung erkennen können, welchen Buttons welche Funktionen zugewiesen sind, wurde erneut von einem Prefab des VRTK Assets Nutze gemacht, ControllerTooltips.



(a) Tooltips Controller(left)



(b) Tooltips Controller(right)

Abbildung 3.90: VRTK Tooltips Komponente

Das Prefab ControllerTooltips

```
Assets/assets/SteamVR_Unity_Toolkit/Prefabs/ControllerTooltips
```

wurde je beiden Controllern in der Szene hinzugefügt und die Bezeichnung der Buttons wurde wie in Abb. 3.90b vorgenommen.

4 Fazit

Es wurden die Grundlagen der Arbeit mit *Mega Flow*, der Unity Engine und Darstellung für VR Hardware, der HTC Vive gezeigt und manuell eine Szene erstellt, die all dies mit VTK importierten Strömungsdaten vereint, um die Strömungen live, interaktiv in einer VR Applikation zu betrachten.

Es wurde mit den Vive Controllern gearbeitet und Asset Packages, welche Interaktionen mit diesen vereinfachen. Die Küchenszene wurde für übersichtliche Demonstrationen der Funktionen der Visualisierung in der Größe des VR-Labors der Hochschule Kaiserslautern (Standort Zweibrücken) angelegt.

Die zuerst erstellte Demoszene, die noch das Hochschullogo beinhaltet, sowie die umgesetzte Küchenszene bieten eine anschauliche Basis für die weitere Arbeit an diesem Projekt mit *Mega Flow* und anderer VR Hardware, die zum jetzigen Zeitpunkt noch nicht verfügbar ist.

Für die Weiterentwicklung dieser Anwendung ist Automatisierung des Imports, sowie Nutzerfreundlichkeit in UI/UX priorisiert. Ebenso Anpassung visueller Elemente für eine rundum fertiggestellte Applikation, die mehr als nur auf einer Demonstration der Funktionen auf einem technischen Level basiert, sondern auch wie die erstellte Demoszene im futuristischen Raum visuell beeindrucken kann und mehr Professionalität ausstrahlt, was jedoch im Rahmen einer Bachelorarbeit natürlich nicht zur eigentlichen Funktionalität beiträgt.

Zusätzlich verlief die Realisierung des Unity Projektes linear, zu keinem Zeitpunkt war der nächste Schritt unklar und die Arbeit mit den Unity Assets konnte zu schnellen Ergebnissen führen, ohne die Notwendigkeit, jede Funktion der Umsetzung selbst zu implementieren.

4.1 Features aus der Zukunft

Letztendlich noch eine Ansammlung an Funktionen, die sich in der weiteren Entwicklung dieser Applikation ergeben werden, sowie die geplanten Methoden zur Verwirklichung.

- Zeitsteuerung bzw. Steuerung einer Zeitachse der Visualisierung, sowie Darstellen interstationärer Vektorfelder.
Eine Zeitsteuerung, welche auch auf Partikelsysteme wirkt, bietet das Asset *Chronos* -

Time Control [lud15], es sollte hiermit zumindest möglich sein, einfach eine Steuerung zum Zurückspulen und Verlangsamem umzusetzen.

- UI mit erweiterten Optionen (zur Anpassung der Partikel, der Szene, des Vektorfeldes usw.).
Ähnlich der momentanen UI sollte es möglich sein ein Menü mit weniger Einschränkungen vorzubringen. Das Asset *CurvedUI* [Chi16] bietet die perfekte Basis für ein VR User Interface, unabhängig der Art des Inputs und kann in unterschiedlichen, selbst-bestimmten Formen, wie z.B. einem Ausschnitt eines Zylinders oder einer Hemisphäre dargestellt werden.
- Größere Auswahl an unterschiedlichen, durchdachten Partikelsystemen.
Durch Generieren einer 3D Textur in *Mega Flow* sollte es möglich sein, mittels Implementierung eines eigenen Shaders z.B. GPU basierte Partikelsysteme zu unterstützen, oder Schnittebenen durch Verschieben geometrischer Objekte während der *Runtime* darzustellen.
- Automatisierter Export/Import von VTK Daten, beispielsweise durch Python, Unity Editor selbst oder externe Anwendung
- Automatisches Anlegen der Raumgeometrie für Imports, falls nötig.
Unity Editor Script könnte beim Import der Daten feststellen, welche Maße das Vektorfeld hat, wie und ob es skaliert werden soll und wo die Geometrie sich im Raum befindet und selbst Objekte für die Positionen generieren.
- Darstellung von Strömungslinien in VR.
Anlaufstelle hierfür sind die Methoden zur Darstellung von Ribbons im *Mega Flow* Editor Script. Anders wäre es möglich Partikel oder Trails in einer sehr hohen Frequenz zu injizieren.
- Steuerung bzw. Einstellungen für Desktop Kamera um bei Demonstrationen mehr Darstellungsmöglichkeiten für Zuschauer zu haben, die lediglich die nicht-VR Ansicht vor sich haben.
- Oculus Touch Support, Google Daydream Support bzw. Unterstützung einer Vielzahl aktueller VR Hardware (auch mobil).
- Online Multiplayer; Für remote Demonstrationen, sowie die Möglichkeit, mit anderen Nutzern gleichzeitig Visualisierungen in VR zu betrachten.

Literaturverzeichnis

- [BB03] BENDER, MICHAEL und BRILL, MANFRED: *Computergrafik. Ein anwendungsorientiertes Lehrbuch.* Hanser, 2003.
- [BGZ13] BUNGARTZ, H.J., M. GRIEBEL und C. ZENGER: *Einführung in die Computergrafik: Grundlagen, Geometrische Modellierung, Algorithmen.* Mathematische Grundlagen der Informatik. Vieweg+Teubner Verlag, 2013.
- [Bri08] BRILL, MANFRED: *Virtuelle Realität.* Informatik im Fokus. Springer Berlin Heidelberg, 2008.
- [Chi16] CHISELY: *Curved UI - VR Ready Solution To Bend / Warp Your Canvas*, 2016. Online unter <https://www.assetstore.unity3d.com/en/#!/content/53258>, Version 1.8, zuletzt geprüft 11.12.2016.
- [Chr12] CHRIS WEST: *Mega Shapes*, Juni 2012. Online unter <https://www.assetstore.unity3d.com/en/#!/content/24340>, Version 2.21, zuletzt geprüft 15.12.2016.
- [Chr16a] CHRIS WEST: *Mega Flow*, November 2016. Online unter <https://www.assetstore.unity3d.com/en/#!/content/24340>, Version 1.27, zuletzt geprüft 15.12.2016.
- [Chr16b] CHRIS WEST: *west-racing.com*, November 2016. Online unter http://www.west-racing.com/mf/?page_id=5892, zuletzt geprüft 11.12.2016.
- [Cre15] CREEPYCAT: *3D Showroom Level Kit Vol 1*, Februar 2015. Online unter <https://www.assetstore.unity3d.com/en/#!/content/29679>, Version 1.8, zuletzt geprüft 11.12.2016.
- [Ho16] HO, DON: *Notepad++*, September 2016. Online unter <https://notepad-plus-plus.org>, zuletzt geprüft 27.11.2016.
- [Kit16] KITWARE: *The Visualization Toolkit*, September 2016. Online unter <http://www.vtk.org>, zuletzt geprüft 27.11.2016.
- [lud15] LUDIQ: *Chronos - Time Control*, 2015. Online unter <https://www.assetstore.unity3d.com/en/#!/content/31225>, Version 2.4.4, zuletzt geprüft 14.12.2016.

- [Sys16] SYSDIA SOLUTIONS LTD: *VRTK - SteamVR Unity Toolkit*, November 2016. Online unter <https://www.assetstore.unity3d.com/en/#!/content/64131>, Version 2.2.1, zuletzt geprüft 12.12.2016.
- [Tec16a] TECHNOLOGIES, UNITY: *Unity 5.4*, September 2016. Online unter <https://unity3d.com/de/unity/whats-new/unity-5.4.0>, Version 5.4, zuletzt geprüft 15.12.2016.
- [Tec16b] TECHNOLOGIES, UNITY: *Unity Tutorials*, September 2016. Online unter <https://unity3d.com/de/learn/tutorials>, zuletzt geprüft 15.12.2016.
- [Uni16a] UNITY TECHNOLOGIES: *Asset Packages*, Dezember 2016. Online unter <https://docs.unity3d.com/Manual/AssetPackages.html>, Version 5.5, zuletzt geprüft 15.12.2016.
- [Uni16b] UNITY TECHNOLOGIES: *Asset Workflow*, Dezember 2016. Online unter <https://docs.unity3d.com/Manual/AssetWorkflow.html>, Version 5.5, zuletzt geprüft 15.12.2016.
- [Uni16c] UNITY TECHNOLOGIES: *Particle System*, Dezember 2016. Online unter <https://docs.unity3d.com/ScriptReference/ParticleSystem.html>, Version 5.5, zuletzt geprüft 15.12.2016.
- [Uni16d] UNITY TECHNOLOGIES: *Unity 5.5.0b4*, November 2016. Online unter <https://unity3d.com/de/unity/beta/unity5.5.0b4>, Version 5.5.0b4, zuletzt geprüft 15.12.2016.
- [Val16] VALVE CORPORATION: *SteamVR Plugin*, November 2016. Online unter <https://www.assetstore.unity3d.com/en/#!/content/32647>, Version 1.1.1, zuletzt geprüft 15.12.2016.
- [Zac16] ZAC ZIDIK: *VRBasics*, August 2016. Online unter <https://www.assetstore.unity3d.com/en/#!/content/69323>, Version 1.2, zuletzt geprüft 15.12.2016.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, **Karim Stock**, geboren am **12.01.1995, Heidelberg**, ehrenwörtlich,

- dass ich meine Bachelorarbeit/Masterarbeit mit dem Titel

Immersive Strömungsvisualisierung in Unity

selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe;

- dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Zweibrücken, 19.12.2016

Karim Stock