Systemspezifikation Vortex-Tunnel

|  |  |  |
| --- | --- | --- |
| **Version** | **Beschreibung** | **Bearbeiter** |
| 1.0 | Erster Entwurf des Berichts | Marc Nussbaumer |
| 1.1 | Allgemeine Entwurfsentscheide | Philipp Gröbelbauer |
|  |  |  |
|  |  |  |

Inhaltsverzeichnis

[Einführung 4](#_Toc502754851)

[Anforderungen 4](#_Toc502754852)

[Abgrenzung der Anforderungen 5](#_Toc502754853)

[Use Cases 6](#_Toc502754854)

[Use Case Diagramm 6](#_Toc502754855)

[Use Case: Konfigurieren 6](#_Toc502754856)

[Use Case: Tunnel durchlaufen 6](#_Toc502754857)

[Allgemeine Entwurfsentscheide 8](#_Toc502754858)

[Effektgenerierung der Drehung 8](#_Toc502754859)

[Prototyp Spotlight 8](#_Toc502754860)

[Prototyp Pointlight 9](#_Toc502754861)

[Prototyp Textur 10](#_Toc502754862)

[Vergleich der Prototypen 12](#_Toc502754863)

[Varianten zur Parameter-Eingabe 13](#_Toc502754864)

[Variante 1: Eingabemaske beim Start der Tunnel-Simulation 13](#_Toc502754865)

[Variante 2: Parameter in einem XML-File 13](#_Toc502754866)

[Variante 3: Ein simpler Editor zum Tunnelbau 13](#_Toc502754867)

[Vergleich der drei Varianten 13](#_Toc502754868)

[Konfigurierbarkeit Steg 15](#_Toc502754869)

[Wahl der Tunneltextur 16](#_Toc502754870)

[Tunnelgenerierung 17](#_Toc502754871)

[Variante 1: Generieren eines Tunnel-Meshes in C# 17](#_Toc502754872)

[Variante 2: Tunnel-Mesh modellieren und in Unity laden 17](#_Toc502754873)

[Entscheidungstreffung 17](#_Toc502754874)

[Systemübersicht 18](#_Toc502754875)

[Komponentendiagramm 19](#_Toc502754876)

[Hardware-Anforderungen 19](#_Toc502754877)

[Schnittstellen 20](#_Toc502754878)

[Konfigurations-Datei 21](#_Toc502754879)

[Struktur 21](#_Toc502754880)

[Vortex-Tunnel 22](#_Toc502754881)

[Abschnitte 22](#_Toc502754882)

[Abschnitt 23](#_Toc502754883)

[Wandmuster 23](#_Toc502754884)

[Textur 24](#_Toc502754885)

[Lichter 25](#_Toc502754886)

[Farbe 26](#_Toc502754887)

[Stegtransparenz 27](#_Toc502754888)

[Rotation der Lichter 28](#_Toc502754889)

[Drehende Textur 29](#_Toc502754890)

[Umgang mit Renderer 29](#_Toc502754891)

[UV Mapping 30](#_Toc502754892)

[SketchUp Modell zu Unity 31](#_Toc502754893)

[Transformation Elemente 32](#_Toc502754894)

[XML-Mapping 34](#_Toc502754895)

[Best Practices VR 35](#_Toc502754896)

[Deployment 36](#_Toc502754897)

[Ausblick 37](#_Toc502754898)

# Einführung

Im Auftrag des iHomeLab der Hochschule Luzern soll nicht technikaffinen Personen «Virtual Reality» anhand eines Vortex-Tunnels nähergebracht werden. Ein Vortex-Tunnel ist eine drehende Röhre, welche von einer Person auf einem Steg durchschritten werden soll. Dabei ist es in der Röhre dunkel, sichtbar sind nur sich drehende Lichtpunkte und Muster. Dies führ zu einer Beeinträchtigung des Gleichgewichts-Sinnes der Person. Diese Beeinträchtigung soll in der virtuellen Welt reproduziert werden, damit Personen die Möglichkeiten von Virtual Reality sehen können.

## Anforderungen

Gemäss Aufgabenstellung sind folgende Ergebnisse zu erarbeiten:

* Konzept zur Erreichung des maximalen Effekts auf den Gleichgewichtssinn der Probenden
* Erstellung Softwarekonzept und Testkonzept
* Softwareprototyp mit Unity3d und Oculus Rift oder HTC Vive
* Testresultate mit den Probanden und Diskussion der erarbeiteten Ergebnisse
* Code, Dokumentation und HowTo’s auf GitHub.

Nach einem Workshop mit Herr Biallas (siehe Anhang «20171011\_Protokoll Sitzung Vortex-Tunnel») wurden folgende konkreten Anforderungen für den Tunnel definiert:

Muss:

* Benutzer soll sich durch laufen durch den Tunnel bewegen können
* Es existiert ein gerades Tunnelmodell durch welches gelaufen werden kann
* Der Tunnel soll aufgrund einer XML-Konfiguration generiert werden
* Versuchsbeschreibung zur Effekterreichung mit Auswertung über Fragebögen   
  (mindestens 10 Datensätze)

Kann:

* Der Tunnel soll Kurven aufweisen

Aufgrund der Zwischenpräsentation der Prototypen wurde folgende Muss-Anforderung zusätzlich formuliert (siehe Anhang «20171108\_Protokoll Zwischenpräsentation Vortex-Tunnel»)

Muss:

* In der XML-Konfiguration sollen zusätzlich die Segmente des Tunnels definiert werden können

Basierend auf den Erkenntnissen aus der Versuchsdurchführung wurde an der Präsentation der Versuchsresultate folgende Änderungen an der Zielsetzung durchgeführt:

Die Kann-Anforderung der Kurve fällt weg, stattdessen soll die Konfigurierbarkeit des Stegs mit folgenden zusätzlichen Parametern erweitert werden (siehe Anhang «20171207\_Protokoll Versuchsresultate Vortex-Tunnel»):

* + ***Handgeländer ein/ausschaltbar***
  + ***Breite des Stegs***
  + ***Transparenz-Grad des Stegs***
  + ***Textur***
  + ***Höhe***

## Abgrenzung der Anforderungen

Mittels eines Workshops mit Herr Biallas (siehe Anhang «20171011\_Protokoll Sitzung Vortex-Tunnel») wurden folgende Einschränkungen an der Aufgabenstellung beschlossen:

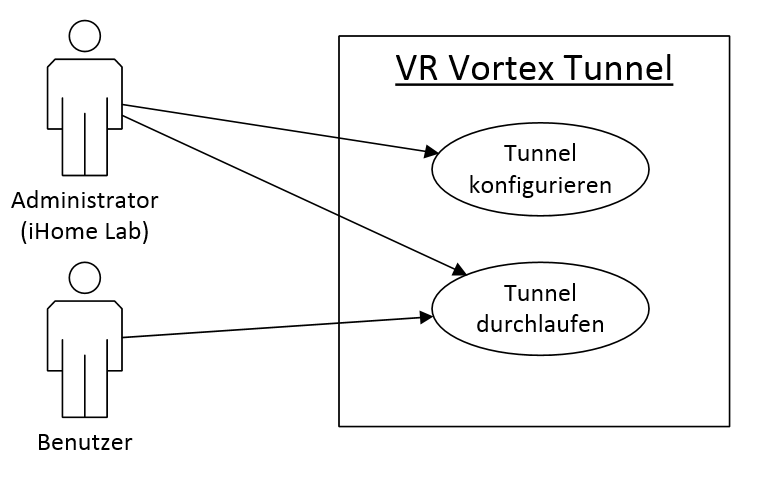
* Kurven sollen nur im 90° Grad Winkel konzipiert werden
* Lichtmuster sind nicht Teil der Konfiguration und können zufällig sein
* Form der Lichtpunkte ist nicht Teil der Konfiguration und kann beliebig sein
* Tunnel ist immer ein Kreis / Zylinder, eine Ellipse wird explizit ausgeschlossen
* Toleranz der Grössenunterschiede zwischen Konfigurationsdatei und Vortex-Tunnel beträgt 10cm

## Use Cases

Da es sich beim VR Vortex Tunnel nicht um ein herkömmliches Softwareprojekt handelt, kann man dieses schlecht in einzelne Use Cases unterteilten. Es kann jedoch klar zwischen zwei Anwendungsfällen unterschieden werden: Dem Konfigurieren des Tunnels und dem Durchlaufen des Tunnels. Die folgenden Abschnitte zeigen diese zwei Use Cases auf.

### Use Case Diagramm

Das nachfolgende Diagramm zeigt die beiden Use Cases auf.



Es ist hierbei anzumerken, dass das System selbst keine Rollen und Zugriffsberechtigungen kennt. Ein normaler Benutzer (nicht Administrator) könnte somit auch den Tunnel konfigurieren, falls er Zugriff auf das XML-Konfigurationsfile im Dateisystem erlangt.

### Use Case: Konfigurieren

1. Der Administrator öffnet die Datei „vortexparams.xml“ (im Unterordner Assets des Projektsordners) mit einem beliebigen Text-Editor.
2. Der Administrator ändert die XML-Datei ab, unter Beachtung der Richtlinien und Richtwerte, die in diesem Dokument im Kapitel „Konfigurations-Datei“ beschrieben sind.
3. Der Administrator speichert die Datei und startet die Applikation VRVortexTunnel.exe.
4. Das System generiert den Vortex Tunnel anhand der neu gesetzten Parameter.

**Sonderfall:** Nach Abändern der Datei und starten der Applikation wird der VR Vortex Tunnel nicht mehr richtig dargestellt. Dies zeichnet sich beispielsweise durch fehlende Texturen und Lichter aus. In diesem Fall wurde die vorgegebene XML-Struktur verletzt und der Administrator muss die Datei berichtigen, sodass ein funktionaler Vortex Tunnel generiert werden kann.

### Use Case: Tunnel durchlaufen

1. Der Administrator/Benutzer startet die Applikation VRVortexTunnel.exe.
2. Der Benutzer setzt die HTC-Vive VR-Brille auf.
3. Der Benutzer durchläuft den VR Vortex Tunnel unter Beaufsichtigung oder unter Zuhilfenahme einer Notfall-Stütze.

Fehlt hier ein Titel? Evt Kontextdiagramm?

System

Exe zu SteamVR

Umgebung

# Allgemeine Entwurfsentscheide

## Effektgenerierung der Drehung

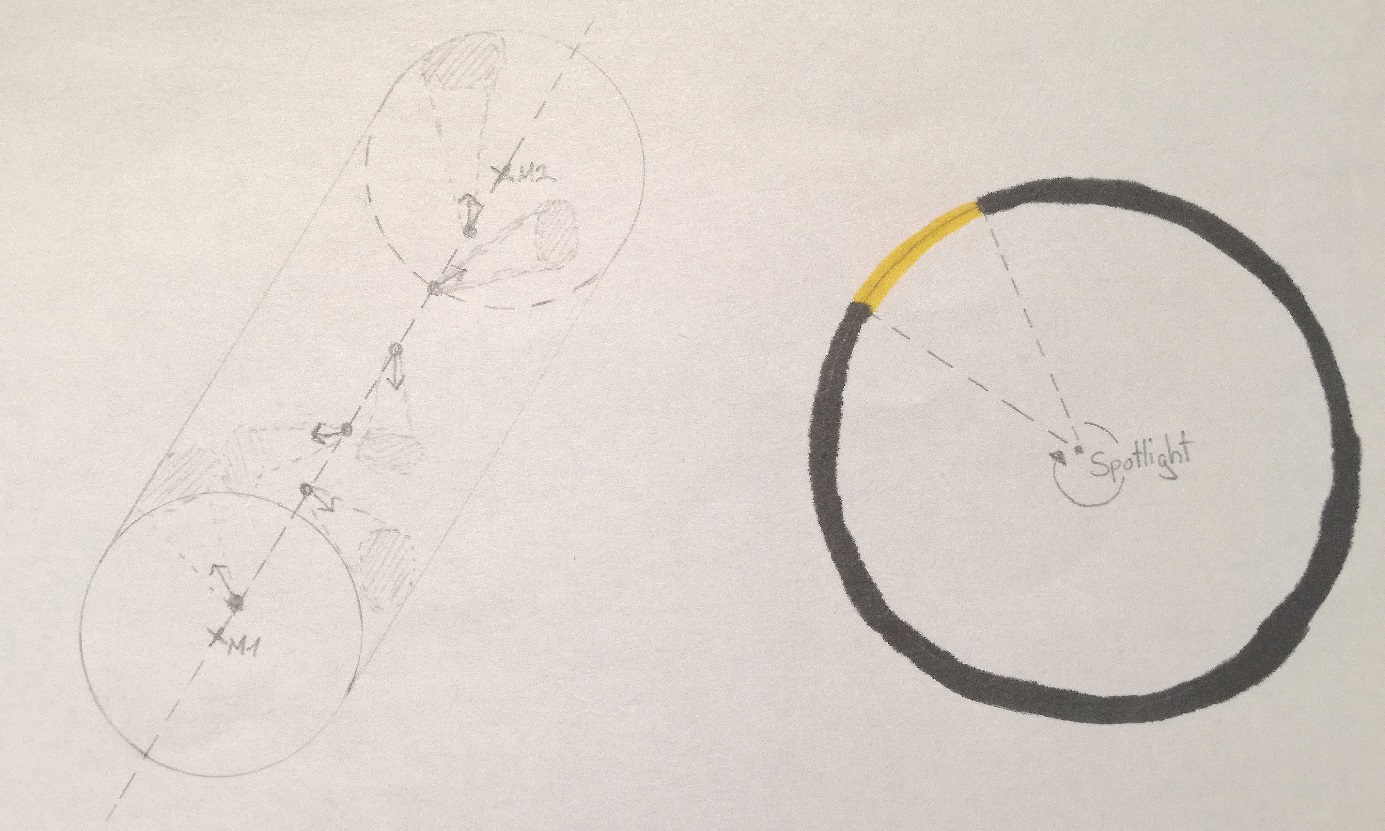
Als Entscheidungsbasis für das weitere Vorgehen zur Effekterzeugung wurden drei Prototypen mit unterschiedlichen Ansätzen erstellt und anschliessend in einer Sitzung vorgestellt.

### Prototyp Spotlight

In einem ersten Ansatz wurde versucht, den Dreh-Effekt innerhalb des Tunnels mit Unity-Spotlights zu erzeugen. Spotlights sind kegelförmige Lichter, die alle Oberflächen von Meshes, die sich mit dem Lichtkegel überschneiden, beleuchten.

Jedes einzelne Spotlight sollte schlussendlich an der Tunnel-Innenwand als Lichtpunkt sichtbar sein und sich an dieser entlangbewegen. Um dies zu erzielen wurde folgender Algorithmus erdacht:

1. Beim Programmstart werden die Spotlights dynamisch entlang der Z-Achse im Wertebereich 0 bis l erzeugt, wobei l die Tunnellänge ist, welche durch die Konfigurationsdatei gegeben wird. Die Z-Achse stellt zugleich die Tunnelmitte dar.
2. Jedes Spotlight wird initial um einen zufälligen Winkel um die Z-Achse rotiert. Dies sorgt für eine gute Verteilung der Lichter. Weitere Parameter wie beispielweise Lichtfarben werden randomisiert und/oder gesetzt.
3. Jedem Spotlight wird ein Rotations-Script zugewiesen. Dieses soll bewirken, dass sich das Licht mit jedem Frame (mit jedem Aufruf der Unity Update()-Methode) um einen kleinen Winkel um die Z-Achse weiterdreht. Somit bewegen sich alle Lichtpunkte stetig um die Tunnelwand und erzeugen einen Dreh-Effekt.



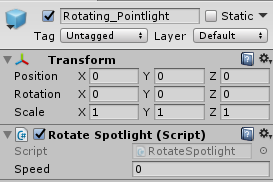
Aufgrund eines Unity-Bugs (oder einer Unity Eigenheit) funktionierte dieser erste Ansatz jedoch nicht wie erwartet. Das Problem war, dass Spotlights ab einer bestimmten Entfernung nicht mehr sichtbar waren. Der genaue Grund dafür ist uns nicht bekannt. Klar war aber, dass auch die Lichter am Ende des Tunnels sichtbar sein müssen – daher wurde dieser erste Ansatz verworfen.

### Prototyp Pointlight

Basierend auf dem ersten Ansatz wurden in einem zweiten Prototyp Unity Pointlights verwendet. Pointlights erzeugen keinen Lichtkegel, sondern eine Lichtkugel. Aus diesem Grund müssen die Lichter auf der Tunnelwand platziert werden und sich dieser entlangbewegen, damit sie als Lichtpunkte sichtbar werden. Um dieses Verhalten zu erzielen, wurde zur Vorbereitung folgendes Prefab definiert:



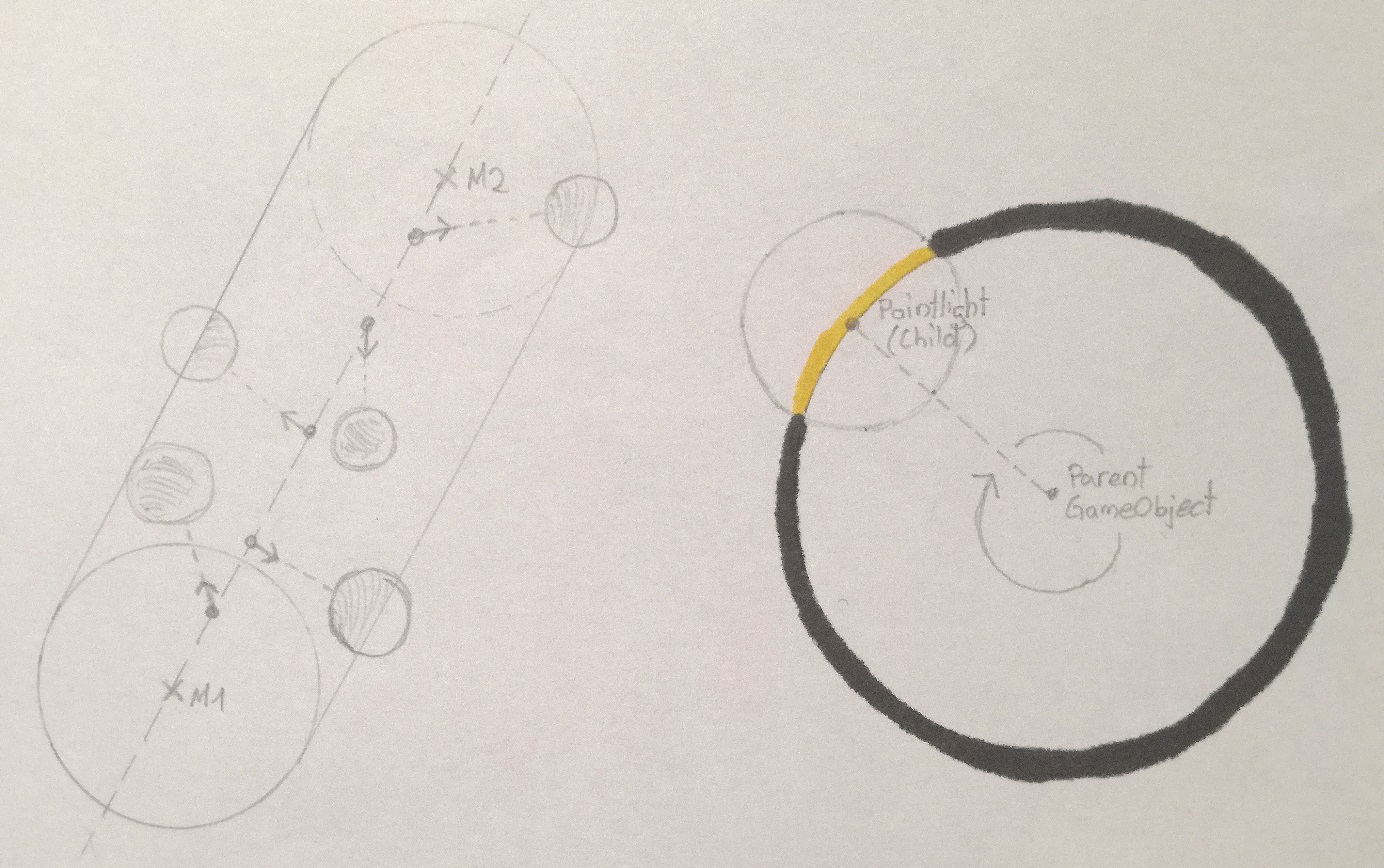
Wie das obenstehende Bild zeigt, besteht das Prefab aus zwei GameObjects. Das erste (Rotating\_Pointlight) ist lediglich ein unsichtbarer Punkt im Raum. An diesem Objekt angehängt ist dasselbe Script aus dem ersten Prototypen, welches eine stetige Rotation um die Z-Achse bewirkt.



Das zweite GameObject des Prefabs (Point light) ist das eigentliche Pointlight. Dieses befindet sich zu Beginn auf denselben Koordinaten wie das erste GameObject. Wichtig ist aber, dass Rotating\_Pointlight und Point light in einer Parent-Child Beziehung stehen. Eine Transformation des Parents (Rotating\_Pointlight) wird somit auch eine Transformation des Childs, also des eigentlichen Pointlights mit sich ziehen, aber nicht umgekehrt.

Mit diesem Prefab als Ausgangslage wurde zur Erzielung des Dreh-Effektes der folgende Algorithmus verwendet:

1. Beim Programmstart werden Objekte nach dem oben beschriebenen Prefab instanziiert. Diese werden entlang der Z-Achse im Wertebereich 0 bis l erzeugt, wobei l die Tunnellänge ist, welche durch die Konfigurationsdatei gegeben wird. Die Z-Achse stellt zugleich die Tunnelmitte dar.
2. Alle eigentlichen Pointlights (Children) werden um die Hälfte des Tunneldurchmessers auf der X-Achse verschoben. Somit liegen die Lichter nun mitten in der Tunnelwand, während deren Parent-Objekte sich noch in der Tunnelmitte befinden.
3. Alle Parent-Objekte werden um einen zufälligen Winkel um die Z-Achse rotiert. Dies führt auch zu einer Rotation der Pointlights (Children). Deren Drehpunkt liegt aber auf dem Parent-Objekt – in der Tunnelmitte. Somit bewegen sich die Lichter stets auf der Tunnelwand und sind nach dieser ersten Rotation gut verteilt.
4. Weitere Parameter der Pointlights werden gesetzt. Dies sind der Radius der Lichtkugel, die Farbe des Lichts und die Lichtintensität. Zudem wird die Drehgeschwindigkeit über ein C#-Property auf dem am Prefab angehängten Script gesetzt.



Dieser Lösungsweg führte zu einem funktionierenden Prototyp. Das an jeder Instanz von Rotating\_Pointlight angehängte Script sorgte für eine stetige Drehung der Parent-Objekte zur Laufzeit der Simulation. Dies wiederum führte dazu, dass sich deren Children, die Pointlights, stets der Tunnelwand entlang bewegten und als wandernde Lichtpunkte sichtbar waren.

### Prototyp Textur

Nebst der Verwendung von vorgefertigten Unity-Lichteffekten wurde zusätzlich eine mögliche Lösung mittels Innen-Textur des Tunnel-Modells untersucht. Dabei wird auf jegliche Beleuchtung ausser dem Ambient-Light verzichtet. Stattdessen werden die Lichtpunkte durch Punkte auf der Grafik für die Textur dargestellt.

Um eine Textur auf einem Modell abzubilden ist eine sogenannte UV-Map notwendig, diese bestimmt welche Bildpunkte auf welche Modell-Punkte abgebildet werden sollen (siehe Kapitel «UV Mapping»).

Um die Textur zu drehen wurden folgende Wege untersucht:

* Drehung des Modells
* Drehung der Kamera
* Drehung der Textur

Eine Drehung des Modells ist solange möglich, bis es an Kurven angewendet werden soll. Wie in Abbildung 1 Problemfall Modelldrehung sichtbar wird dreht sich das Kurvenstück (rot) vom anschliessenden Tunnel-Teilstück weg und der Tunnel bekommt ein Loch.

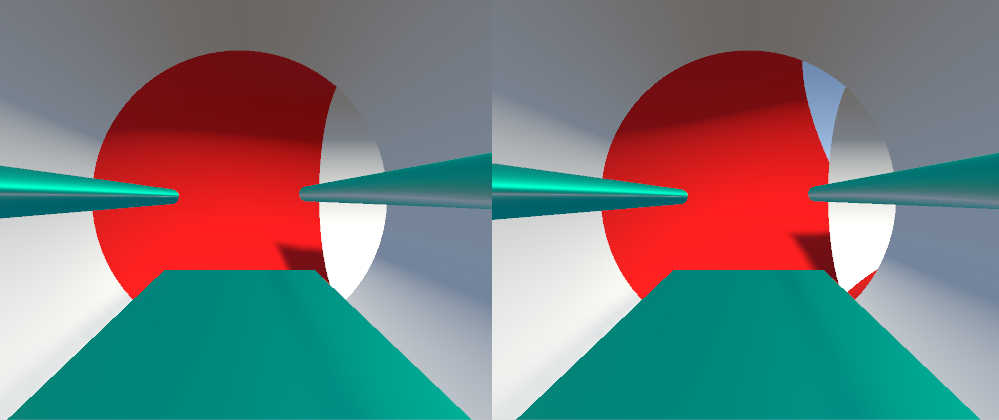


Abbildung 1 Problemfall Modelldrehung

Die Drehung der Kamera hat das Problem, dass diese nicht in Zusammenhang mit Kurven verwendet werden kann. Wie in Abbildung 2 Problemfall Kameradrehung zu sehen ist dreht sich der Ausgang (rot) der Kurve mit der Kamera mit und verunmöglicht somit die korrekte Darstellung einer Kurve.

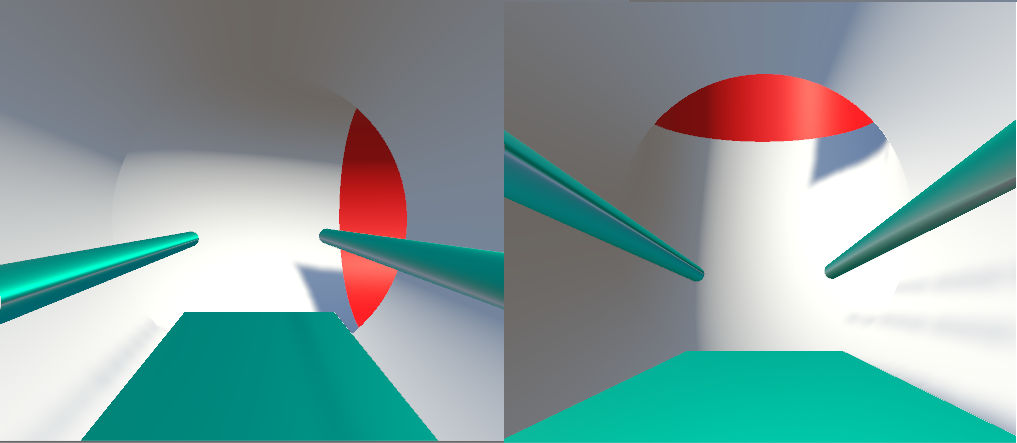


Abbildung 2 Problemfall Kameradrehung

Wird die Drehung über die Textur gelöst, so kann eine Drehung auch in der Kurve erzeugt werden. Dabei wird es jedoch zu Verzerrungen oder Clipping kommen, das genaue Verhalten kann per UV-Map (siehe Kapitel «UV Mapping») beschrieben werden. Eine mögliche Lösung ist in «Abbildung 3 Lösungsansatz Texturdrehung» zu sehen.

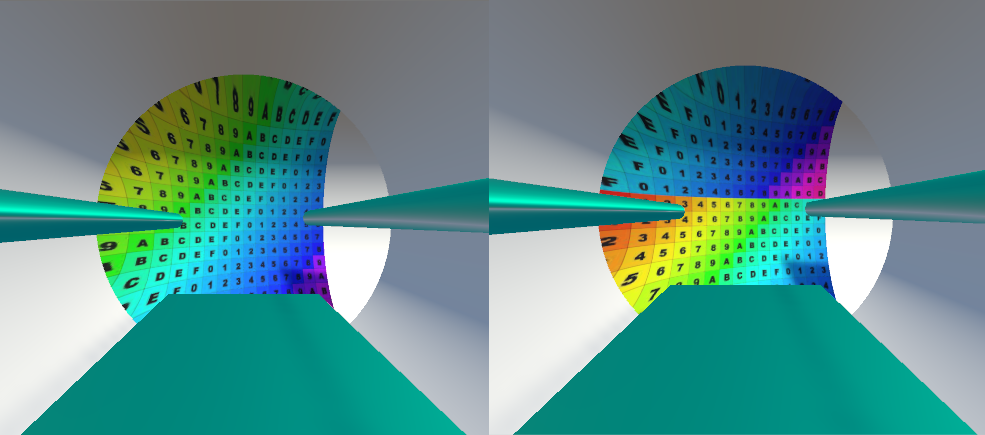


Abbildung 3 Lösungsansatz Texturdrehung

Da die Lösung mittels Drehung der Textur einen Lösungsansatz für das Problem der Drehung der Kurve bietet wurde dieser Lösungsansatz weiterverfolgt.

### Vergleich der Prototypen

Die zwei funktionierenden Prototypen wurden an der Sitzung vom 08.11.2017 (siehe Anhang «20171108\_Protokoll Zwischenpräsentation Vortex-Tunnel») präsentiert. Es wurde dabei entschieden beide Lösungsansätze zu verknüpfen und in zukünftigen Versionen zu verwenden.

## Varianten zur Parameter-Eingabe

In den folgenden Abschnitten werden verschiedene Varianten zur Parameter-Eingabe angeschaut und im Anschluss verglichen. Die Parameter-Eingabe muss nicht durch den Endbenutzer erfolgen, sondern durch Vertreter des iHome-Labs. Aus diesem Grund muss die Eingabe nicht zwingend über ein im Programm integriertes GUI erfolgen. Das Verändern der Parameter soll aber das Öffnen von Unity/einer IDE nicht voraussetzen.

### Variante 1: Eingabemaske beim Start der Tunnel-Simulation

Die Parameter können direkt vom Benutzer, der die VR-Brille trägt, vor Start der Simulation gesetzt werden. Unity bietet einfache Steuerelemente an, die dies ermöglichen.

### Variante 2: Parameter in einem XML-File

Die Parameter werden strukturiert in einem XML-File eingegeben. Anschliessend wird die VR-Simulation gestartet und die Werte werden aus der Datei übernommen.

### Variante 3: Ein simpler Editor zum Tunnelbau

Ein simples Tool zum Zeichnen des Tunnels. Die Benutzeroberfläche des Tools könnte den Tunnel aus der Vogelperspektive darstellen und das Setzen von neuen Tunnelabschnitten ermöglichen. Das Tool wird eine Datei erzeugen (xml, txt oder anderes Format), die dann beim Programmstart eingelesen und zur Generierung des Tunnels verwendet wird. Im Vergleich zu den ersten beiden Varianten wird bei dieser Lösung kein zufälliger Tunnel generiert.

### Vergleich der drei Varianten

Zur Auswahl der besten Variante werden alle Konzepte verglichen. Die Kriterien an eine Parametereingabe sind die folgenden:

* Einfachheit der Eingabe: Die Parametereingabe soll möglichst kein technisches Know-How erfordern. (1= Gelernter Informatiker, 3=Technischer Assistent, 5= Microsoft-Office Anwender)
* Komfort: Wie angenehm ist es für den Benutzer, die Parameter einzugeben. (1=Mehrere Zusatzprogramme notwendig, 3=Ein Zusatzprogramm notwendig, 5=Alles in der Applikation)
* Entwicklungsaufwand: Wie aufwändig ist die Entwicklung? (1= sehr aufwändig und grosser Zeitaufwand, 3=tragbarer Aufwand ohne zusätzliche Kenntnisse/Libraries, 5=wenig Aufwand und kann gut in C# umgesetzt werden.)
* Genauigkeit: Wie gezielt und präzise können die Parameter gesetzt werden? (1=grobe Definition ohne Massangabe, 3=Masse in bestimmten Intervallen, 5=freie Massangaben)
* Erweiterbarkeit: Wie einfach können neue Parameter hinzugefügt werden? (1=Entwickler für die Anpassung benötigt, 3=Technischer Assistent, 1=Laie)

Die drei Varianten wurden an den genannten Kriterien gemessen und verglichen. Jedes Kriterium wurde zusätzlich gewichtet.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Gewichtung** | **Variante 1 -GUI** | **Variante 2 -XML** | **Variante 3 -Editor** |
| **Einfachheit** | 3 | 5 | 3 | 5 |
| **Komfort** | 2 | 3 | 3 | 4 |
| **Entwicklungsaufwand** | 3 | 2 | 5 | 1 |
| **Genauigkeit** | 1 | 3 | 5 | 3 |
| **Erweiterbarkeit** | 1 | 1 | 5 | 1 |
| **Gesamt** | 10 | 31 | 40 | 30 |

Tabelle Vergleich Konfigurationsmethoden

Aus dem Vergleich geht hervor, dass sich die Parameter-Eingabe über eine XML-Datei am besten eignet. Deshalb wird dieser Ansatz weiterverfolgt.

## Konfigurierbarkeit Steg

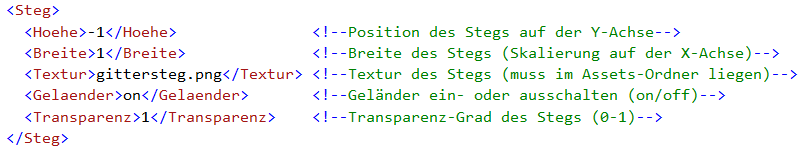
In der Sitzung vom 11.10.2017 wurden die Anforderungen an das Projekt geklärt. Unter anderem wurden unter dem Stichwort „Bodenbeschaffenheit“ die gewünschten Konfigurationsmöglichkeiten des Stegs bekanntgegeben: Es soll mindestens aus drei verschiedenen Steg-Varianten gewählt werden können. Vorgeschlagen wurde ein solider Steg (grau oder schwarz), ein Gittersteg und ein Glassteg.



Abbildung 4 Erste Version des Stegs

Bereits an der Zwischenpräsentation vom 08.11.2017 kam die Idee eines engeren Stegs auf, sodass man beispielsweise nicht mehr beide Füsse nebeneinander setzen könnte. Es könnte sein, dass es dann noch schwieriger ist, das Gleichgewicht im VR Vortex Tunnel zu halten, weil man zusätzlich noch balancieren muss.

Anhand der Resultate und Rückmeldungen des Versuchs mit Probanden (siehe Wissenschaftlicher Bericht) wurde in der Sitzung vom 07.12.2017 entschieden, die Konfigurationsmöglichkeiten des Stegs zu erweitern und stattdessen die Implementation der Kurve wegzulassen. Dieser Entscheid führte zu folgenden Parametern in der XML-Datei, die neu frei konfigurierbar sind:



Da der Steg jeweils pro Tunnelabschnitt definiert ist, sind auch hier wiederum eine Vielfalt von Konfigurationsmöglichkeiten und Spielereien möglich. Beispielsweise könnte der Steg anfangs ein Geländer haben, welches aber nach zwei Metern aufhört. Der Steg könnte gar Lücken aufweisen, über die gesprungen werden muss oder er könnte komplett weggelassen werden.

## Drehende Tunneltextur

Bei der Erstellung des Prototyps mit der drehenden Textur (siehe «Prototyp Textur») wurde eine PNG-Datei verwendet, welche Spotlights darstellt (siehe Abbildung «Abbildung 5 Erste Tunneltextur»). Dabei wurde noch keine Endlos-Textur verwendet, da es darum ging den grundlegenden Effekt zu simulieren.

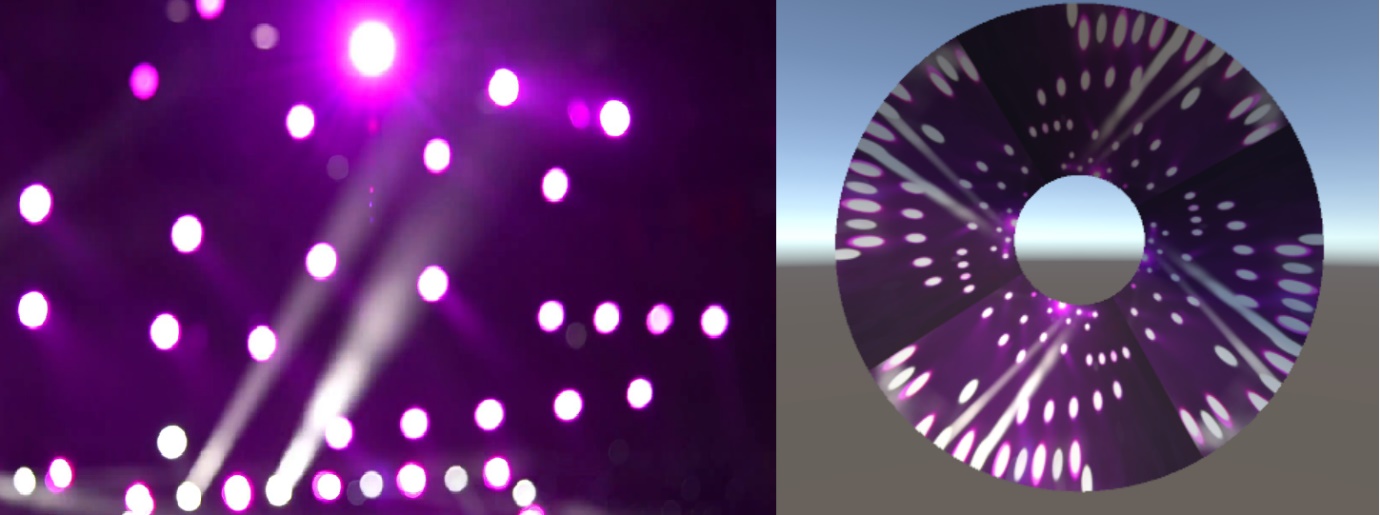


Abbildung 5 Erste Tunneltextur

Die Bild-Datei befand sich dabei im Resouce-Order welcher einen zeitnahen Zugriff in Unity erlaubt, so kann auf jede Datei per «Resources.Load(name)» zugegriffen werden wie in Abbildung 6 Textur über Resources sichtbar ist. Dabei wird die Textur für ein beliebiges GameObject gesetzt, wobei dies in diesem Fall das Tunnel-Modell ist.

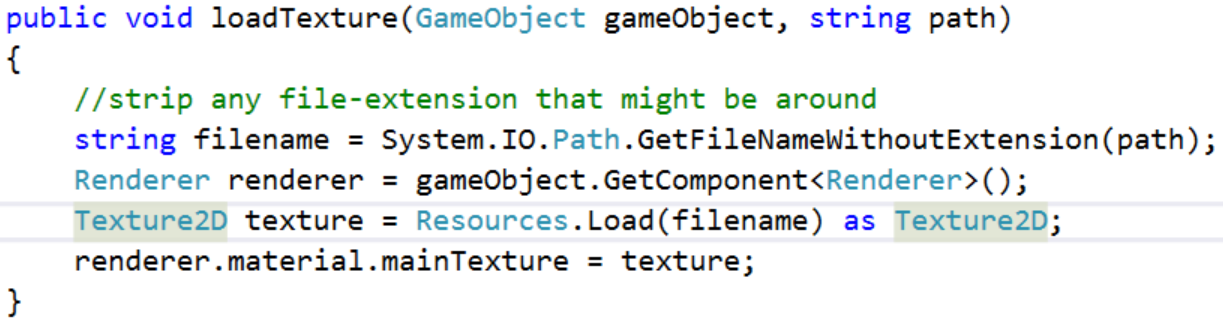


Abbildung 6 Textur über Resources

Bei der Zwischenpräsentation (siehe Anhang «20171108\_Protokoll Zwischenpräsentation Vortex-Tunnel») wurde entschieden, dass die Tunneltextur frei gewählt werden soll, um Effekte mit unterschiedlichen Grafiken untersuchen zu können. Die Bild-Datei kann aufgrund dieser Entscheidung nicht mehr im Resource-Ordner platziert werden, daher funktioniert der bisherige Zugriff nicht mehr.

Um den Zugriff auf eine beliebige Bild-Datei sicherzustellen musste wie in Abbildung 7 Textur über WWW ersichtlich über die WWW-Methode zugegriffen werden. Diese Funktion erlaubt Zugriff sowohl auf Internet-Ressourcen oder auch lokale Dateien. Beim Zugriff auf lokale Dateien ist es zwingend den Pfad mit «file://» zu beginnen.

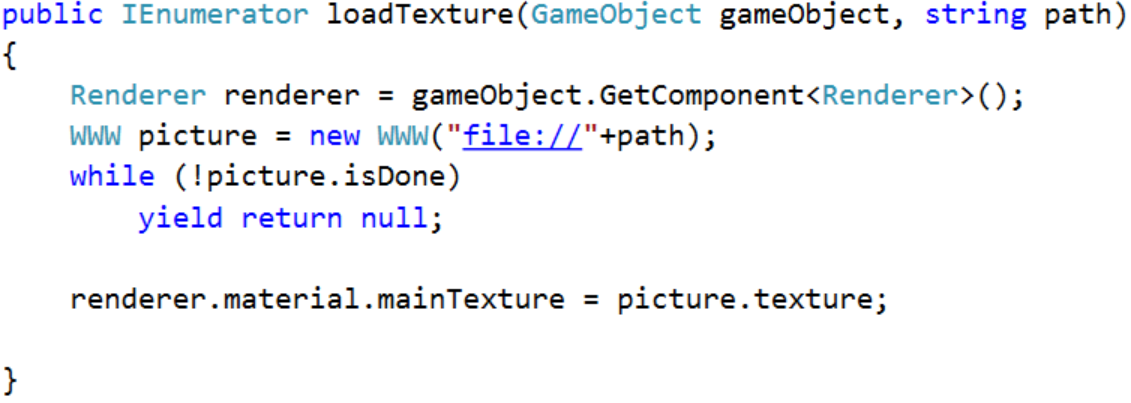


Abbildung 7 Textur über WWW

Zusätzlich erfordert die Implementation der WWW-Methode, dass während des Ladens die Frames übersprungen werden. Dies wird in Unity über «yield return null» mit dem dazugehörigen Rückgabe-Datentyp «IEnumerator» sichergestellt. (Stackoverflow - IEnumerator's yield return null, 2017)

Basierend auf den Rückmeldungen der Testprobanden des Versuchs (siehe «Wissenschaftlicher Artikel») wurde entschieden eine sich wiederholende Textur mit Spiralen-Muster zu erzeugen (siehe «Abbildung 8 Textur Spirale»). Bei sich wiederholenden Texturen ist es wichtig, dass die UV-Map passgenau ist, da sonst Linien in dem Modell wahrgenommen werden können. (siehe Kapitel «UV Mapping»)



Abbildung 8 Textur Spirale

## Tunnelgenerierung

Nach den initialen Projekt-Anforderungen sollte ein konfigurierbarer VR Vortex-Tunnel erstellt werden, der sowohl gerade, als auch kurvige Abschnitte enthalten kann. Um einen solchen Tunnel zu erhalten, kamen zwei grundlegend verschiedene Vorgehensweisen in Frage, die in den Folgekapiteln beschrieben sind.

### Variante 1: Generieren eines Tunnel-Meshes in C#

Ein Tunnel, der sich durch eine Neukonfiguration stets verändert und beliebig viele enge oder weite Kurven beinhalten kann, ist nicht vormodellierbar in einem CAD-Programm. Aus diesem Grund wurde vorerst das programmatische Generieren eines Tunnel-Meshes untersucht.

Dazu bietet UnityEngine die Klasse Mesh. Einem Objekt dieser Klasse kann ein Array von Vertices zugewiesen werden, welche jeweils zu dreieckigen, sichtbaren Oberflächen verbunden werden können. Mit dieser Vorgehensweise kann ein gesamtes 3D-Gitterobjekt per C#-Code zusammengebaut werden. Wie eine Implementation dieses Ansatzes ungefähr aussehen könnte ist unter <https://docs.unity3d.com/ScriptReference/Mesh.html> zu finden.

### Variante 2: Tunnel-Mesh modellieren und in Unity laden

Eine zweite Variante ist das Modellieren des dreidimensionalen Tunnels mithilfe einer CAD-Software. Dies stellte die Frage nach der Software. Hierbei wurde Google Sketchup gewählt, aus den folgenden Gründen:

* Einfach zu erlernen
* Simpel gehalten - daher optimal für Benutzer, die sich noch wenig auskennen in der 3D-Modellierung
* Optimal zur Erstellung eines statischen Objekts, wie beispielsweise einem Gebäude – oder einem Tunnel

Die anfänglichen Bedenken dieses Ansatzes waren, dass unter Umständen viele verschiedene Tunnels – oder modulare Tunnelabschnitte – erstellt werden müssten, um später einen hochkonfigurierbaren Vortex Tunnel daraus generieren zu können.

### Entscheidungstreffung

In der Sitzung vom 11.10.2017 (siehe Sitzungsprotokoll) wurden die Anforderungen an das Projekt spezifiziert. Dabei stellte sich heraus, dass ausschliesslich 90° Kurven und gerade Tunnelabschnitte gefordert sind.

Dies kam sehr zugunsten der zweiten Variante – dem Modellieren der Tunnelstücke, die später in Unity nur noch geladen werden müssen. Das programmatische Generieren des Tunnels wäre im Vergleich dazu sehr zeitaufwändig und viel fehleranfälliger gewesen. Aus diesem Grund wurde die Entscheidung getroffen, sowohl ein gerades, als auch ein 90° Tunnelstück in Google Sketchup zu modellieren. Diese Tunnelstücke sollten dann beim Start der Simulation aneinandergehängt werden – jedoch maximal ein gerades Stück und eine Kurve.

An der Zwischenpräsentation vom 08.11.2017 wurde in einer kleinen Diskussionsrunde ein weiterer, interessanter Ansatz vorgeschlagen. Die XML-Konfigurationsdatei könnte es ermöglichen, eine beliebig lange Liste von Tunnelabschnitten zu definieren, die dann in Unity generiert und aneinandergereiht werden. Dieser Vorschlag wurde weiterverfolgt und schliesslich implementiert. Dies einerseits weil die Anzahl an Konfigurationsmöglichkeiten dadurch enorm anstieg. Andererseits war auch der Aufwand dazu relativ gering – der bereits bestehende Code zur Tunnelgenerierung änderte sich nur minimal und wurde neu für jeden Tunnelabschnitt ausgeführt.

# Systemübersicht

Im Gegensatz zu einem klassischen Software-Entwurf ist bei Unity das Konzept auf sogenannte «Prefabs» und Scripts ausgelegt. Dabei sind die Prefabs immer Objekte, welche sichtbar oder unsichtbar auf der Szene platziert werden.

Sobald ein Prefab geladen wird werden sämtliche angeschlossenen Scripts ausgeführt. Dazu erben sämtliche Scripts (ein Script besteht hierbei meist aus einer Klasse) von der Klasse «MonoBehavior» des Imports «UnityEngine».

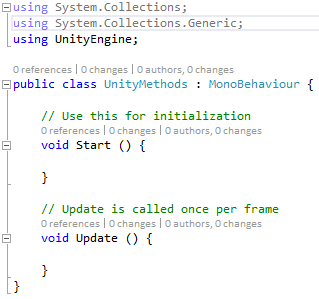


Abbildung 9 Unity-Methoden

Wie in «Abbildung 4 Unity-Methoden» sichtbar ist werden dabei zwei Methoden zur Verfügung gestellt, die Start-Methode welche bei erstmaliger Instanziierung stattfindet und die Update-Methode welche pro Frame ausgeführt wird.

Die Generierung neuer Objekte basierend auf «Prefabs» setzt einen nicht performanten Suchprozess (siehe «Abbildung 5 Instanzierung neues Objekt») in Gange und sollte daher möglichst wenig in der Update-Methode passieren.

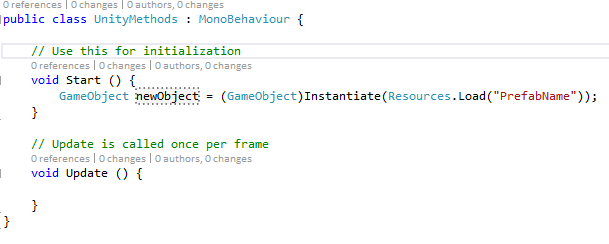


Abbildung 10 Instanzierung neues Objekt

## Komponentendiagramm

Aufgrund des nicht performanten Prozesses zur neuen Objektinitialisierung haben wir uns entschieden, dies mit dem Script «BuildTunnel» zu lösen. Am Anfang der Szene ist nur das Prefab «Tunnel\_Builder» sichtbar welches dann sofort das «BuildTunnel»-Script startet. Dort wird aufgrund der XML-Konfiguration eine oder mehrere Instanzen der Tunnel-Teile instanziiert wie in «Abbildung 6 Komponentendiagramm Vortex-Tunnel» aufgezeigt wird.

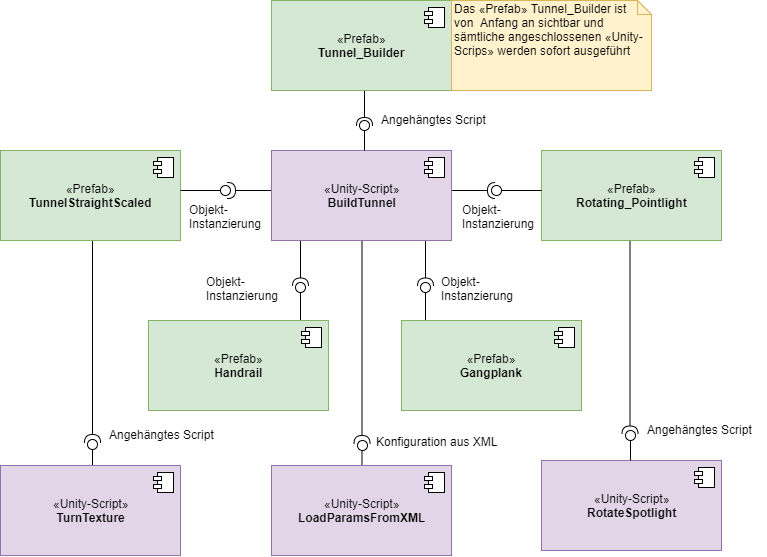


Abbildung 11 Komponentendiagramm Vortex-Tunnel

Aufgrund der Verwendung eines einzelnen Scripts kann so sichergestellt werden, dass alle generierten Objekte zum gleichen Zeitpunkt sichtbar werden. Zusätzlich wird der Ressourcen-Intensive Suchprozess (siehe «Abbildung 5 Instanzierung neues Objekt») nicht durch andere Scripts verlangsamt. Aus Performance-Sicht kann somit die schnellstmögliche Generierung der benötigten Objekte sichergestellt werden. Dies ist insbesondere Wichtig bei einer hohen Anzahl Lichter im Tunnel.

## Hardware-Anforderungen

VR-Computer mit folgenden Eigenschaften:

* nVidia GeForce 970+ / AMD Radeon RX 480 oder besser
* Intel Core i5 / AMD FX 8350 oder besser
* Steam-Account
* Vive-Account
* Stabile Internet-Verbindung

# Schnittstellen

In Unity ist die Verwendung von Schnittstellen anders als in anderen Programmiersprachen. Anstatt etwa eine Library zu importieren erhält man ein oder mehrere Prefabs, welche man anschliessend verwenden kann. Um solche Prefabs zu finden bietet Unity einen integrierten Marktplatz auf welchem sowohl kostenlose als auch kostenpflichtige Prefabs angeboten werden.

Steam-VR

Zur Kommunikation zwischen HTC Vive und Unity wird «SteamVR» verwendet. Dabei erhält man drei Prefabs zur Verwendung:

* [CameraRig]

Dient dazu die Hardware-Informationen der VR-Hardware auszulesen dazu enthält es wiederum folgende Prefabs:

* + Controller (left)

Enthält Positions-Informationen des linken Controllers, als auch sämtliche Benutzerinteraktionen mit diesem.

* + Controller (right)  
    Enthält Positions-Informationen des rechten Controllers, als auch sämtliche Benutzerinteraktionen mit diesem.
  + Camera (head)  
    Erhält die Position des VR-Brille im Raum und funktioniert zusätzlich als Kamera in Unity. Es kann nur Ursprungsposition gesetzt werden, sämtliche anderen Werte werden laufend zur Laufzeit durch Hardware-Werte gesetzt.
* [Status]

Dient dazu Menüs darzustellen, wird von uns nicht verwendet.

* [SteamVR]

Dient dazu das Spielfeld im virtuellen Raum festzulegen, die Grösse wird über die Raum-Konfiguration in SteamVR festgelegt. Erreicht der Benutzer den physikalischen Rand des definierten Spielfelds so wird dies mit einer Gitter-Linie dargestellt.

Wichtig ist bei Verwendung der SteamVR-Schnittstelle, dass keine anderen Kameras verwendet werden dürfen. Dies da ansonsten der Anwender fälschlicherweise die Perspektive dieser Kamera einnehmen könnte.

Die Einrichtung der Kamera-Perspektive über [CameraRig] muss bei über das Objekt in der Szene geschehen bevor diese abgespielt oder kompiliert wird. Zur Laufzeit werden diese Werte automatisch über die Hardware gesetzt, wobei dies relativ zur Ursprungsposition geschieht.

Eine Manipulation der Kamera wäre höchsten über eine Verschachtelung in ein anderes Prefab möglich, jedoch verletzt dies diverse Best Practices (siehe Kapitel «Best Practices VR») und ist daher nicht empfehlenswert.

# Konfigurations-Datei

## Struktur

<VortexTunnel durchmesser=**"6"**>

<Abschnitte>

<!--Tunnelabschnitt -->

<Abschnitt>

<Typ>**gerade**</Typ>

<Steg>**gitter.png**</Steg>

<Laenge>**20**</Laenge>

<Wandmuster>

<Textur>

<Name>**spotlight\_texture.png**</Name>

<Drehrichtung>**rechts**</Drehrichtung>

<Drehgeschwindigkeit>**4**</Drehgeschwindigkeit>

</Textur>

<Lichter>

<Reichweite>**2**</Reichweite>

<Intensitaet>**10**</Intensitaet>

<Anzahl>**50**</Anzahl>

<Drehrichtung>**rechts**</Drehrichtung>

<Drehgeschwindigkeit>**4**</Drehgeschwindigkeit>

<!--Festlegen des Farbspektrums, in welchem das Muster erzeugt wird (Werte 0-1) -->

<Farbe>

<MinimumRot>**0**</MinimumRot>

<MaximumRot>**1**</MaximumRot>

<MinimumGruen>**0**</MinimumGruen>

<MaximumGruen>**1**</MaximumGruen>

<MinimumBlau>**0**</MinimumBlau>

<MaximumBlau>**1**</MaximumBlau>

</Farbe>

</Lichter>

</Wandmuster>

</Abschnitt>

<!-- Weitere Abschnitte können hinzugefügt werden(Reihenfolge!)

<Abschnitt>

...

</Abschnitt>

-->

</Abschnitte>

</VortexTunnel>

## Vortex-Tunnel

<VortexTunnel durchmesser=**"6"**>

Attribut «**durchmesser**» legt den Durchmesser in **Meter** des gesamten Tunnels fest.

**Erlaubte Werte:**

Gleitkomma-Zahlen welche grösser als die Grösse des Tunnelbenutzers sind. Ansonsten liegt die Kamera ausserhalb des Tunnels und sieht nichts.

## Abschnitte

<Abschnitte>

<!--Tunnelabschnitt -->

<Abschnitt>

**<<Abschnitt 1>>**

</Abschnitt>

<Abschnitt>

**<<Abschnitt 2>>**

</Abschnitt>

</Abschnitte>

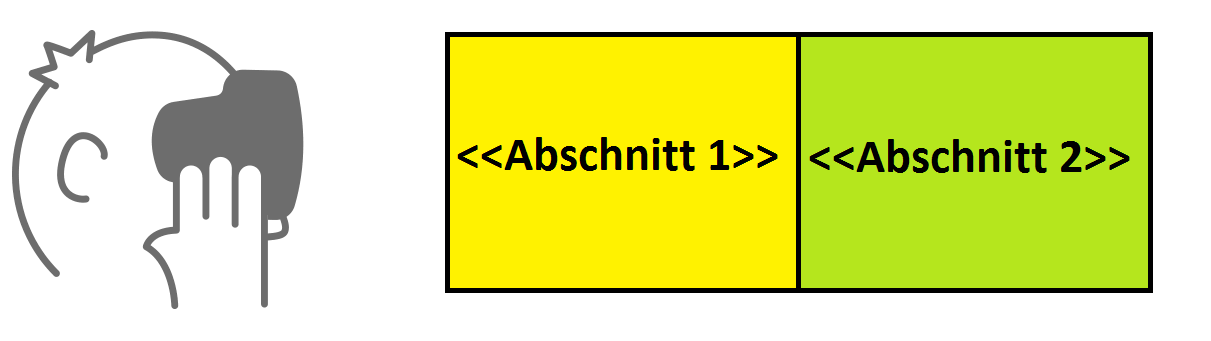
In «**Abschnitte**» werden die einzelnen Tunnel-Abschnitte (siehe weiter unten) spezifiziert. Dabei werden die Tunnel-Abschnitte von oben nach unten abgearbeitet.   


Bild-Quelle: <http://www.view-master.com/en-US/Images/troubleshooting-reel_tcm1147-254727.png>

## Abschnitt

<Abschnitt>

<Steg>**gitter.png**</Steg>

<Laenge>**20**</Laenge>

<Wandmuster>

**<<Definition Wandmuster>>**

</Wandmuster>

</Abschnitt>

«**Steg**» legt fest welche Bild-Datei als Textur für den Steg verwendet werden soll.

Erlaubte Werte:

PNG-Datei - Die PNG Datei muss folgende Eigenschaften für eine gute Darstellung

* Auflösung soll einem Vielfachen von 2 entsprechen. 40x40px ist beispielsweise okay, 41x42px jedoch nicht.
* Es muss eine PNG-Datei sein, andere Bildformate werden nicht unterstützt
* Die Datei kann durch den Namen spezifiziert werden, falls sie bei der EXE-Datei liegt. Ansonsten soll der vollständige Pfad verwendet werden.

«**Laenge**» legt fest, wie lange der Tunnel-Abschnitt in Meter sein soll.

**Erlaubte Werte:**  
  
positiver Integer - positiver Ganz-Zahlwert von 1 bis 2147483647

## Wandmuster

<Wandmuster>

<Textur>

<<Textur-Definition>>

</Textur>

<Lichter>

<<Lichter-Definition>>

</Lichter>

</Wandmuster>

Im Wandmuster werden die Eigenschaften der Tunnelwand beschrieben, dabei müssen sowohl Textur, als auch Lichter definiert sein. Ansonsten dient dieses Tag der Übersicht im XML als Trennung zwischen dem Tunnelaufbau und der Tunneldarstellung.

## Textur

<Textur>

<Name>**spotlight\_texture.png**</Name>

<Drehrichtung>**rechts**</Drehrichtung>

<Drehgeschwindigkeit>**4**</Drehgeschwindigkeit>

</Textur>

«**Name**» legt fest welche Bild-Datei als Tunneltextur für den momentanen Abschnitt verwendet werden soll.

Erlaubte Werte:

PNG-Datei - Die PNG Datei muss folgende Eigenschaften für eine gute Darstellung

* Auflösung soll einem Vielfachen von 2 entsprechen. 40x40px ist beispielsweise okay, 41x42px jedoch nicht.
* Es muss eine PNG-Datei sein, andere Bildformate werden nicht unterstützt
* Die Linke-Seite sollte direkt an die rechte Seite anschliessen können. Ansonsten gibt es Linien im Tunnel
* Die Datei kann durch den Namen spezifiziert werden, falls sie bei der EXE-Datei liegt. Ansonsten soll der vollständige Pfad verwendet werden.

«**Drehrichtung**» legt fest in welche Richtung vom Betrachter aus die Textur dreht.

**Erlaubte Werte:**

«rechts» - dreht die Textur vom Betrachter aus im Uhrzeigersinn  
«link» - dreht die Textur vom Betrachter aus im Gegenuhrzeigersinn

«**Drehgeschwindigkeit**» legt die Kreisbewegung in Unity-Meter / Sekunde fest. Zu beachten ist, dass die Textur viermal im Tunnel angezeigt wird

**Erlaubte Werte:**

Gleitkomma-Zahlen - Empfohlen sind Werte zwischen 0 und 1.

## Lichter

<Lichter>

<Reichweite>**2**</Reichweite>

<Intensitaet>**10**</Intensitaet>

<Anzahl>**50**</Anzahl>

<Drehrichtung>**rechts**</Drehrichtung><Drehgeschwindigkeit>**4**</Drehgeschwindigkeit><Farbe>

**<<Erlaubter Farbraum>>**

</Farbe>

</Lichter>

«**Reichweite**» legt fest wie viele Meter das Licht auf anderen Objekten sichtbar ist.

Erlaubte Werte:

Gleitkommazahlen - Licht ist nur bei positiven Werten sichtbar

«**Intensitaet**» legt fest wie stark eine Lichtquelle sein soll.

Erlaubte Werte:

Positiver Integer - Licht wird bei höheren Werten stärker

«**Anzahl**» legt fest wie viele Lichter in diesem Tunnelabschnitt dargestellt werden sollen.

Erlaubte Werte:

Positiver Integer - Anzahl Lichter die zufällig verteilt werden im Tunnelabschnitt

«Drehrichtung» legt fest in welche Richtung die Lichter im Tunnel-Abschnitt drehen sollen.

Erlaubte Werte:

«rechts» - dreht die Textur vom Betrachter aus im Uhrzeigersinn  
«link» - dreht die Textur vom Betrachter aus im Gegenuhrzeigersinn

«**Drehgeschwindigkeit**» legt die Drehgeschwindigkeit in Unity-Meter pro Sekunde fest

**Erlaubte Werte:**

Gleitkomma-Zahlen - Empfohlen sind Werte zwischen 0 und 1

## Farbe

<Farbe>

<MinimumRot>**0**</MinimumRot>

<MaximumRot>**1**</MaximumRot>

<MinimumGruen>**0**</MinimumGruen>

<MaximumGruen>**1**</MaximumGruen>

<MinimumBlau>**0**</MinimumBlau>

<MaximumBlau>**1**</MaximumBlau>

</Farbe>

«MinimumRot» legt fest, welcher Rot-Wert das Licht mindestens haben muss.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der minimale Rot-Anteil

«MaximumRot» legt fest, welcher Rot-Wert das Licht höchstens haben darf.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der maximale Rot-Anteil

«MinimumGruen» legt fest, welcher Grün-Wert das Licht mindestens haben muss.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der minimale Grün-Anteil

«MaximumGruen» legt fest, welcher Grün-Wert das Licht höchstens haben darf.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der maximale Grün-Anteil

«MinimumBlau» legt fest, welcher Blau-Wert das Licht mindestens haben muss.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der minimale Blau-Anteil

«MaximumBlau» legt fest, welcher Blau-Wert das Licht höchstens haben darf.

Erlaubte Werte:

0.0000 – 1.0000 - je höher der Wert desto höher der maximale Blau-Anteil

# Stegtransparenz

Die Transparenz des Stegs wurde durch das «Alpha Blending» erzielt, hierbei wird der Alpha-Wert einer Grafik dazu verwendet diese Transparent gegenüber anderen Grafik-Elementen darzustellen.

Um dies in Unity anwenden zu können muss man beim Modell-Material, wie in Abbildung 12 Einstellung Rendering Mode gezeigt, den Renderer-Mode auf «Transparent» setzten.

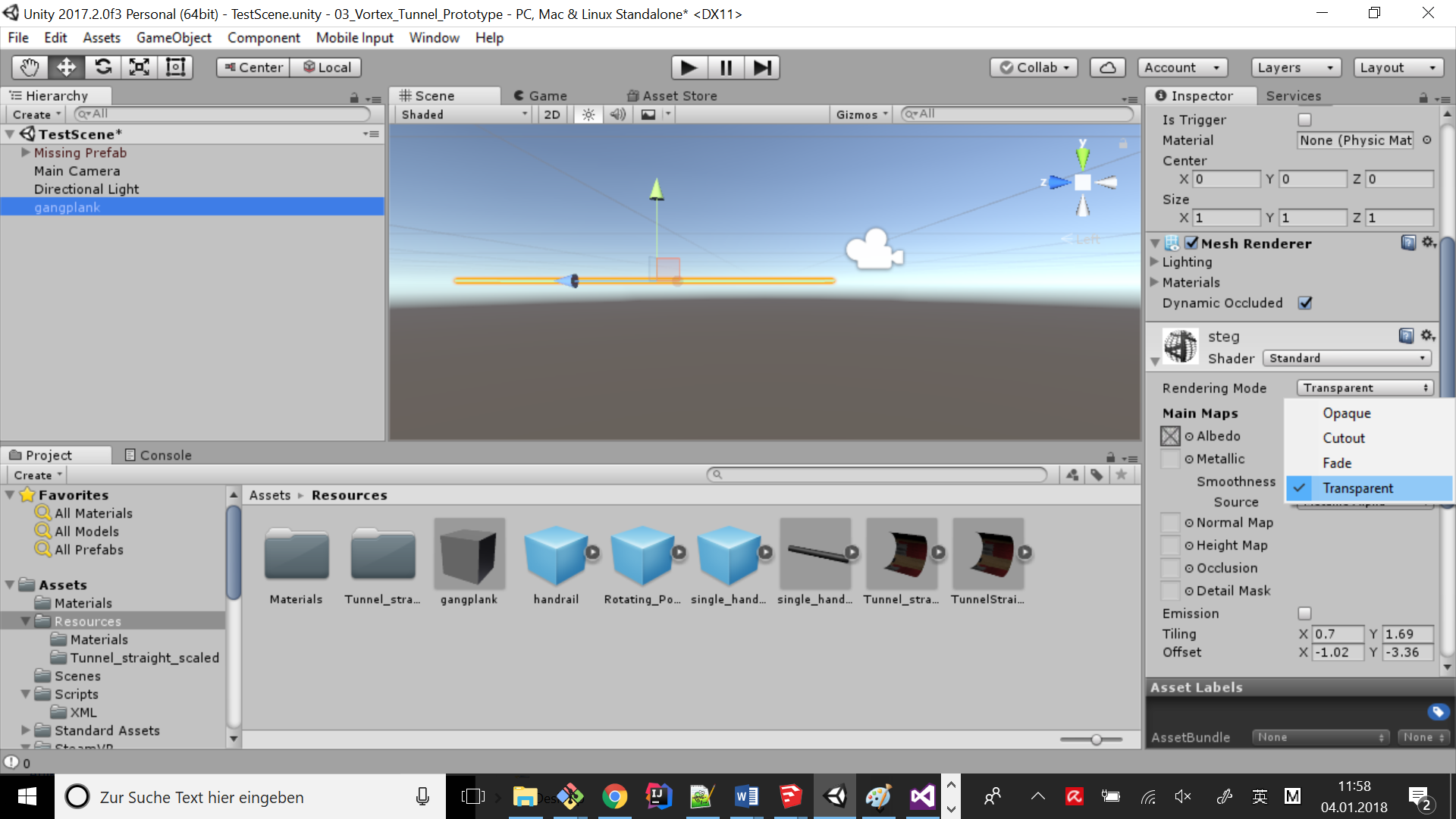


Abbildung 12 Einstellung Rendering Mode

Die Transparenz kann man, wie in Abbildung 13 Code Transparenzänderung gezeigt, per Code manipulieren. Dazu benötigt man zuerst die momentane Farbe des angewandten Materials in einer Zwischenvariable. Auf dieser kann man nun den Alpha-Channel der Grafik wie gewünscht anpassen, wobei Werte zwischen 0 und 1 erlaubt sind, wobei 0 komplette Transparenz bedeutet.

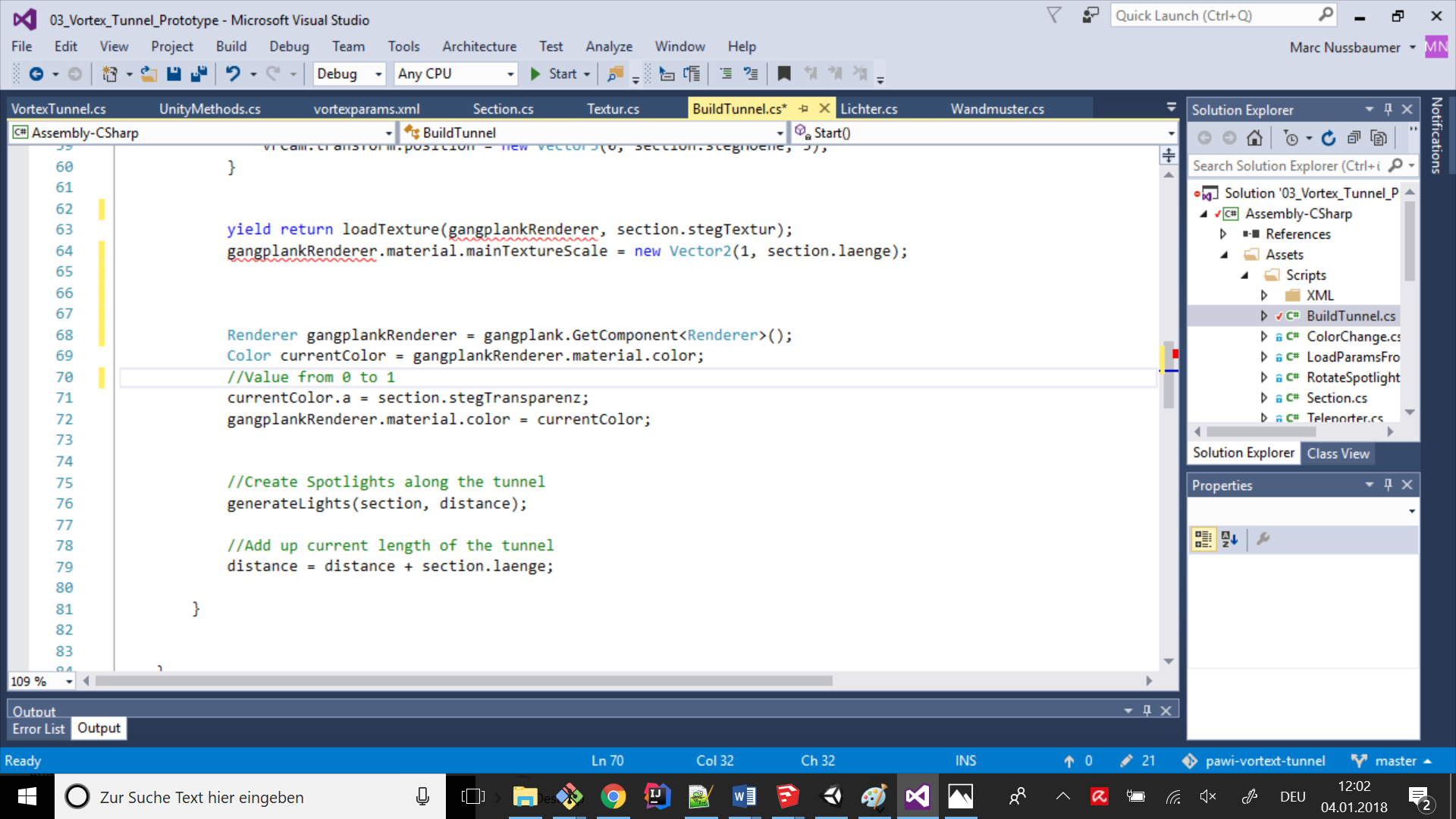


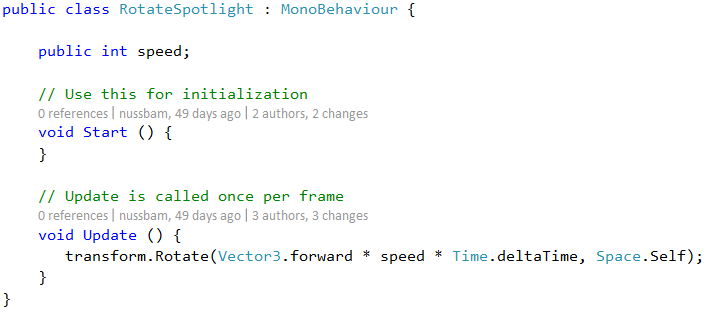
Abbildung 13 Code Transparenzänderung

# Rotation der Lichter

In diesem Kapitel wird die Rotation der Lichter zur Laufzeit der Simulation erläutert. Das dazu erstellte Prefab, sowie der Algorithmus zur Erstellung der Ausgangslage werden im Kapitel „Prototyp Pointlight“ beschrieben.

Die Rotation der Lichter zur Laufzeit geschieht über das C#-Script „RotateSpotlight“. Dieses Script kann theoretisch an ein beliebiges GameObject als Komponente angehängt werden, welches sich dann stetig um die eigene Z-Achse drehen würde. Im Falle des VR Vortex Tunnels wird es jedoch nur für das Rotieren der Lichter verwendet.

Der folgende Code ist das gesamte Script zum Rotieren der Lichter:



Es folgen einige Erklärungen zur Implementation:

* In der Klasse RotateSpotlight wird ein Property „speed“ definiert. Auf dieses kann von ausserhalb des Scripts zugegriffen werden. Da die Drehgeschwindigkeit der Lichter von den aktuellen Werten in der XML-Konfigurationsdatei abhängt, ist diese innerhalb der Klasse RotateSpotlight nicht bekannt. Sie muss von einem anderen Script aus gesetzt werden. Dies geschieht in der Methode generateLights([…]) der Klasse BuildTunnel. An dieser Stelle werden die Lichter (Prefabs) instanziiert, an welchen das RotateSpotlight-Script hängt. Das Setzen des speed-Properties erfolgt in der Klasse BuildTunnel wie folgt:



* Das Script bezieht sich direkt auf das GameObject, an welchem es angehängt wird. Deshalb muss in der Update-Methode keine Referenz auf das zu rotierende Objekt geholt werden. Das Property „transform“ bezieht sich bereits auf die Transformation des PointLight-Prefabs.
* Die Rotate-Methode wird mit zwei Parametern aufgerufen:
  + Der erste Parameter sagt aus, dass um die Z-Achse rotiert wird. Vector3.forward bedeutet nichts anderes als der Einheitsvektor der Z-Achse, also der Vektor (0, 0, 1). Während die Richtung dieses Vektors die Rotationsachse festlegt, bestimmt die Länge des Vektors den Winkel. Da die Methode für jedes neue Frame aufgerufen wird – und nicht zwischen allen Frames gleichviel Zeit verstreicht – muss eine Zeitkomponente hinzugefügt werden. Deshalb wird der Vektor mit Time.deltaTime multipliziert. Time.deltaTime liefert die Zeit in Sekunden, die zum Komplettieren des letzten Frames benötigt wurde (<https://docs.unity3d.com/ScriptReference/Time-deltaTime.html>). Gemäss der Unity Dokumentation von Transform.Rotate wird sich ein Objekt mit einem Grad pro Sekunde drehen, wenn dieser erste Parameter ein Einheitsvektor multipliziert mit Time.deltaTime ist. (<https://docs.unity3d.com/ScriptReference/Transform.Rotate.html>) Es wird hier jedoch ein dritter Multiplikator verwendet: Das Property Speed, welches in der XML-Konfigurationsdatei gesetzt werden kann. Daraus ergibt sich die effektive Rotationsgeschwindigkeit der Lichter, die nur von der Variable speed abhängt:
  + Als zweiter Parameter der Rotate-Methode kann entweder Space.Self oder Space.World angegeben werden. Hiermit wird bestimmt, ob für die Transformation das lokale Koordinatensystem des Objekts oder das globale Koordinatensystem der Welt verwendet wird. Das globale Koordinatensystem bewegt sich nie. Das lokale Koordinatensystem verändert sich jedoch mit jeder Transformation am Objekt. Die folgende Abbildung soll dies veranschaulichen:

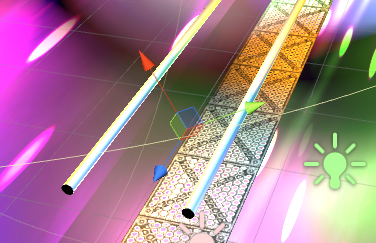


Abbildung 14 Das lokale Koordinatensystem

Die drei farbigen Pfeile in der Mitte des Bildes zeigen das lokale Koordinatensystem des Drehpunkts eines Pointlights auf. Dieser wurde bereits um die Z-Achse rotiert. Die X-Achse (roter Pfeil) und die Y-Achse (grüner Pfeil) zeigen daher nicht mehr in dieselbe Richtung wie die entsprechenden Achsen im globalen Koordinatensystem. Wird das Objekt jedoch immer nur um die Z-Achse rotiert, dann spielt es gar keine Rolle, ob das lokale oder globale Koordinatensystem verwendet wird. Die zeigen in diesem Fall nämlich immer in dieselbe Richtung und wären daher beide als Rotationsachsen geeignet.

# UV Mapping

Als UV Mapping bezeichnet man den Prozess wie man eine 2D-Textur auf ein 3D-Modell umrechnet. Dabei wird durch das UV-Mapping festgelegt welche Koordinaten der Textur an welche XYZ-Koordinaten des 3D-Modells gelegt werden sollten.

Zur Erstellung der UV-Map des Tunnels wurde das Plugin «SketchUV» für SketchUp verwendet. Mit diesem wird eine UV-Map für die ausgewählte Fläche generiert. Das verwendete Verfahren des Plugins war dabei «Box Map», welches für gerade Flächen empfohlen wird. Um die momentan angewandte UV-Map sichtbar zu machen und die Bearbeitung zu erlauben wird eine Hilfstextur (siehe «Abbildung 13 Hilfstextur UV-Map») verwendet.

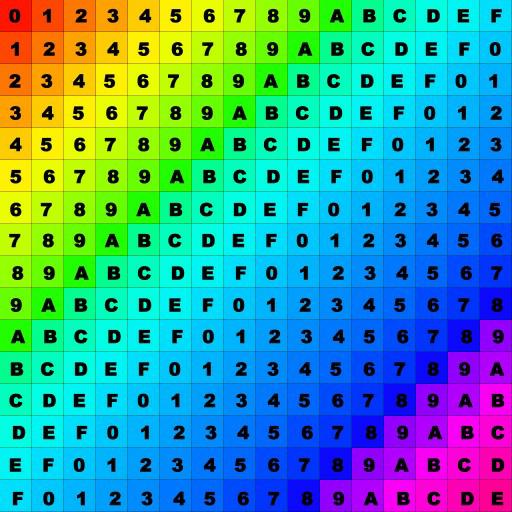


Abbildung Hilfstextur UV-Map

Dabei stellt die Hilfstextur dar, wie anschliessend das Bild auf das Objekt gelegt werden soll. Will man eine sich wiederholende Textur verwenden so muss jede Instanz vollständig auf dem Objekt dargestellt werden. Ist dies nicht der Fall so kommt es zu grafischen Fehlern wie in Abbildung 14 Unkonfigurierte UV-Map zu sehen ist. Diese bestehen mit einer eingestellten UV-Map nicht wie in Abbildung 15 Konfigurierte UV-Map sichtbar ist.

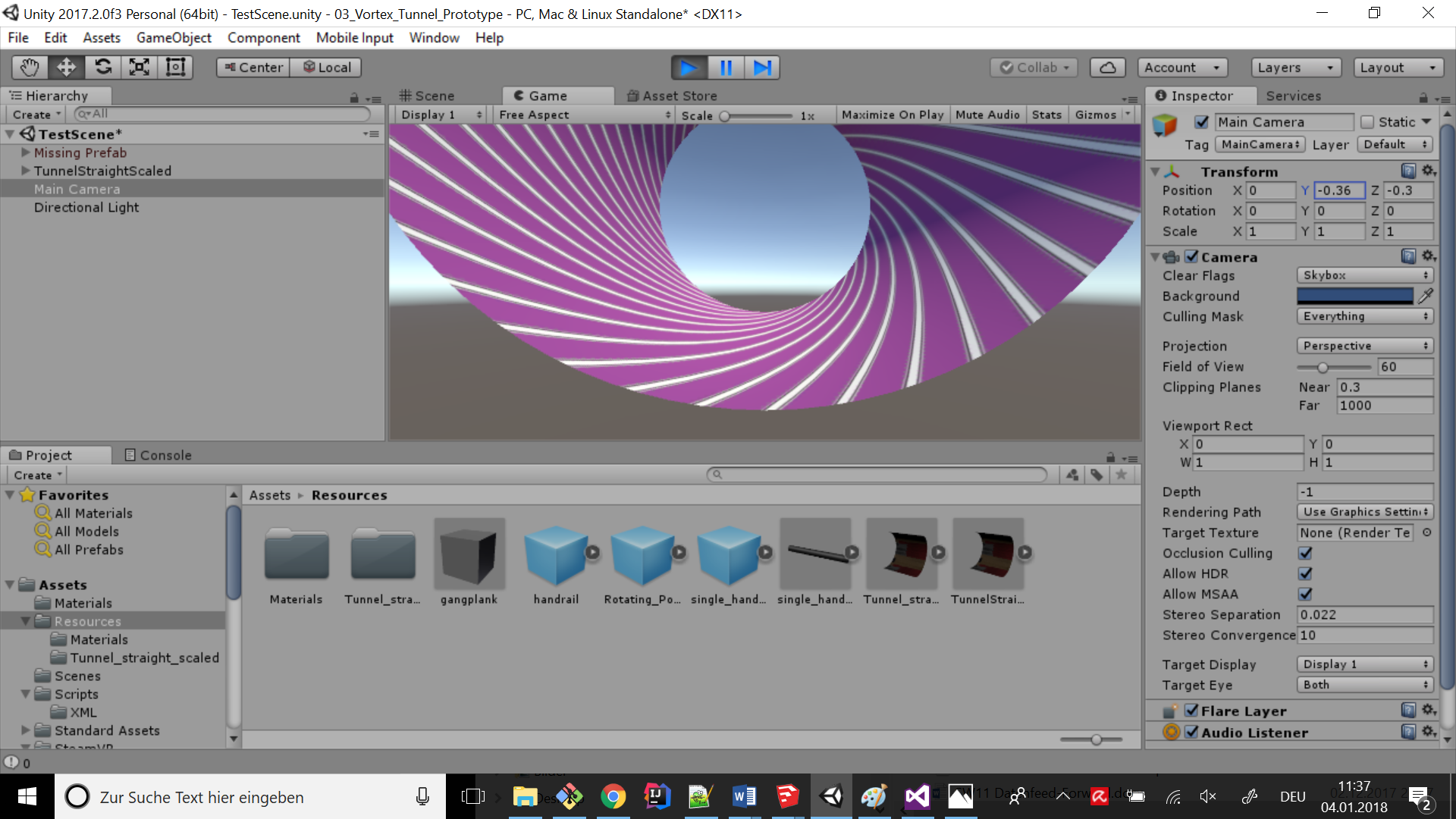
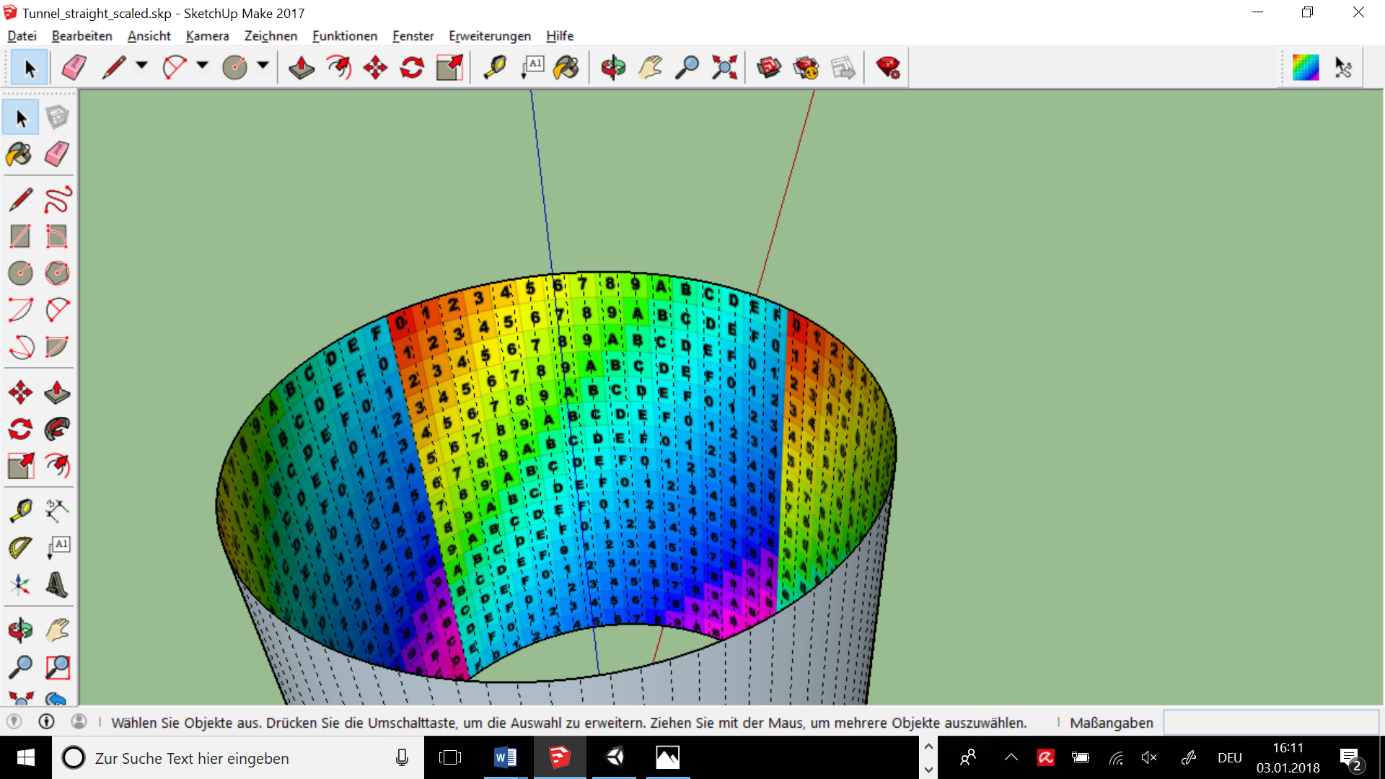
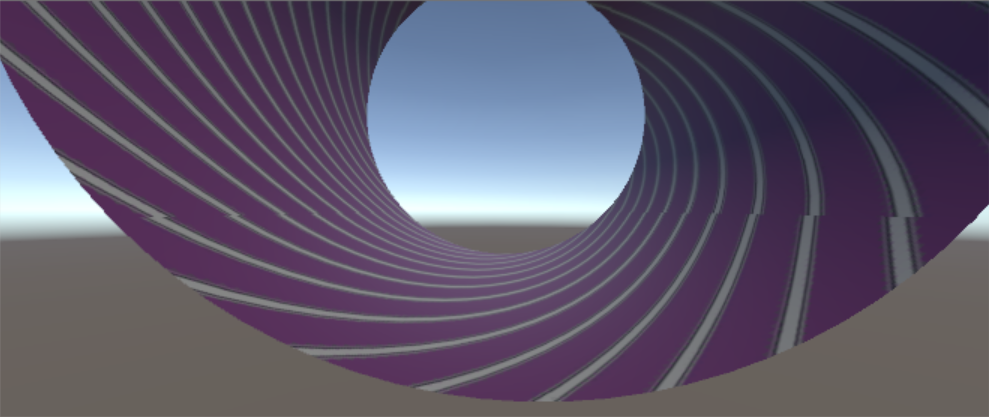
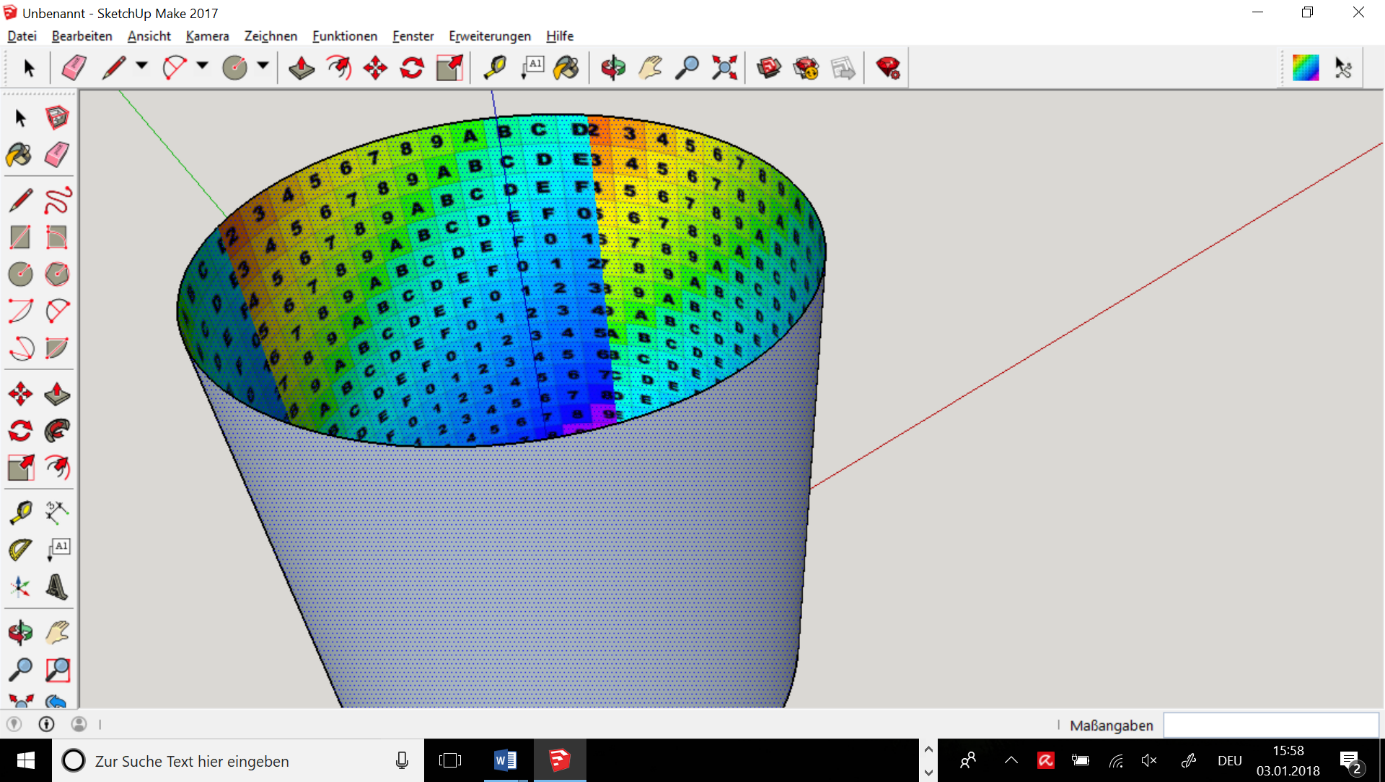


Abbildung Unkonfigurierte UV-Map

Abbildung 17 Konfigurierte UV-Map

# SketchUp Modell zu Unity

Datentyp

Einheit-> Inch

Kreis-Segmente (24 -> 90) = Runder

# Transformation Elemente

Vertex-Transformationen

Siehe Einheit import Unity -> 1m

Transformation in Z-Achse = Länge

Transformation XY = Durchmesser / Breite

Code-Beispiel?

# XML-Mapping

XML-Format auf Klassen ummünzen

# Best Practices VR

Um den Effekt der «VR-Sickness» zu reduzieren haltet sich die Implementierung des Vortex-Tunnels an geltende Best Practices solange diese im Einklang mit dem Projekt stehen.

Folgende Best Practices wurden beachtet:

* Es werden keine Standbilder dargestellt, alles in einer Szene basiert auf einem Modell
* Laufzeitgeschwindigkeitsorientierte Programmierung (nicht unter 60FPS)
* Kamera folgt immer Kopf-Bewegungen
* Keine künstliche Beschleunigung des Benutzers
* Keine Kamera-Wechsel
* Keine Zoom-Funktion
* Keine Laufbewegung (head bopping)
* Keine Beeinflussung der Kamera
* Bodenbeschaffenheit stimmt mit VR Welt überein

Folgende Best Practices konnten explizit nicht verfolgt werden mit Begründung:

* Rotieren von grossen Objekten vermeiden

Ziel des Projektes ist es einen Vortex-Tunnel zu simulieren, welche genau diesen Gleichgewichtsverlust erzeugen will.

(Yao, Heath, Davies, Forsyth, & Mitchell, 2014)

# Deployment

# Ausblick

* Implementation Kurve
* Kurve -> Beliebig (nicht nur 90°)
* Muster der Lichtpunkte
* Physikalischer Steg (wissenschaftlicher bericht)