Umsetzung komplexer Projekte mit einer Virtual-Reality Toolchain

Hauptprojekt Ausarbeitung

Johann Bronsch

Sommersemester 2017

HAW Hamburg

Zusammenfassung. Das Ziel dieser Arbeit ist die Umsetzung eines komplexen Projektes mithilfe einer Virtual-Reality Toolchain. Darüber hinaus sollen einige Interaktionskonzepte vorgestellt und implementiert werden. Diese sollen während einer Veranstaltung von einer Vielzahl von Anwendern getestet werden. Zusätzlich sollen Unity-Plugins und Unity-Assets vorgestellt und deren Funktionsweise erläutert werden.

1 Einleitung

Die Erstellung von komplexen Projekten ist meist sehr schwierig und kann bei einer fehlerhaften Planung oder bei der Auswahl der falschen Werkzeuge scheitern. Daher ist es notwendig sich vorab zu überlegen welche Werkzeuge eingesetzt werden sollen um ein bestimmtes Ziel zu erreichen. Für einige Fälle gibt es jedoch bereits Lösungsansätze oder auch Negativbeispiele.

Zum Erstellen von virtuellen Welten existieren bereits Toolchains, welche die Arbeit erleichtern. In dieser Ausarbeitung wird daher die Toolchain aus [3] verwendet. Mithilfe jener Toolchain soll ein komplexes Projekt umgesetzt werden, welches nicht nur eine komplexe Anwendungslogik besitzt, sondern auch verschiedene Interaktionskonzepte implementiert.

1.1 Bisherige Arbeit

Die vorangegangenen Arbeiten beschäftigten sich sowohl mit dem Thema Virtual- und Augmented Reality im allgemeinen, vgl. [1], als auch mit dem Aufbau einer Toolchain für die Erstellung von virtuellen Welten, vgl. [3]. Das Ziel der letzten Arbeit war die Analyse und Erprobung der benötigten Softwarepakete, die für die Erstellung von virtuellen Welten benötigt wurden. Die ausgewählten Softwarepakete sind in Abbildung 1 dargestellt.

Hierbei lag der Fokus auf der Benutzerfreundlichkeit. Dennoch hat auch der Funktionsumfang der jeweiligen Softwarepakete bei der Entscheidung eine wichtige Rolle gespielt. Die Wahl für die Erstellung von virtuellen Welten fiel daher, im Bereich der

Abb. 1: Die gewählte Toolchain aus der vorangegangenen Arbeit [3, S. 15]

3D-Modellierung, auf Blender und auf Unity3D. Um die Machbarkeit der Toolchain zu überprüfen wurde eine simple Anwendung entwickelt. Dazu wurden in Blender einige Modelle modelliert und in Unity3D importiert. Anschließend wurden einige Objekte mit einer Animation versehen, vgl. [3, S. 20]

Darüber hinaus wurde in vergangenen Arbeiten auch das Thema Interaktion behandelt. Zum einen auf der technischen Ebene, vgl. [3, S. 10], und zum anderen auf der soziologischen Ebene, vlg. [2, S. 2]. Das Ziel dieser Arbeit war sowohl die Analyse des Begriffes Interaktion aus soziologischer Sicht, als auch die Analyse aus technischer Sicht in virtuellen Welten.

1.2 Zielsetzung

Die Machbarkeit einer Toolchain wurde in [3] nachgewiesen. Ziel dieser Arbeit ist es die Machbarkeit der vorher erwähnten Toolchain bei der Umsetzung von komplexen Projekten nachzuweisen. Darüber hinaus sollen einige Interaktionskonzepte aus [2] implementiert werden.

Zu diesem Zweck wird eine Raumschiff-Simulation erstellt. Jene soll hierfür die verschiedenen Plugin-Arten von Unity3D nutzen. Abschließend werden einige Interaktionskonzepte implementiert und überprüft.

2 Raumschiff - Simulation

Wie in Unterabschnitt 1.2 schon erwähnt soll ein komplexes Projekt mithilfe der Toolchain realisiert werden. Hierzu wird eine Raumschiff-Simulation entworfen. Diese soll sich an bekannten Science-Fiction Serien orientieren. Daher wird eine begehbare Brücke entworfen, in der sich der Anwender frei bewegen kann und aus dem *virtuellem Fester* schauen kann. Darüber hinaus sollen dem Anwender einige Möglichkeiten für die Interaktion bereitgestellt werden, wie z.B. das Steuern des Raumschiffes oder das Bedienen eines Interplanetaren-Navigationssystems. Das zuvor erwähnte Interplanetaren-Navigationssystem soll es dem Anwender ermöglichen, mithilfe eines Joysticks, zu dem gewählten Himmelskörper zu reisen. Des Weiteren soll die Simulation sich an unserem Sonnensystem orientieren. Um einen hohen Grad an Immersion zu erzeugen sollen sowohl die Oberflächen der Himmelskörper, als auch die Entfernungen der Himmelskörper zu einander möglichst realistisch dargestellt werden. Zusätzlich soll auch die Bewegung der Himmelskörper möglichst realistisch simuliert werden. Dies

schließt sowohl die Position, als auch die Rotation der Himmelskörper im Sonnensystem mit ein.

Hierfür werden nicht nur Blender-Modelle benötigt, sondern auch einige Unity3D-Plugins. Diese werden für die Simulationslogik, Middleware, vgl. [4], und weitere Elemente benötigt.

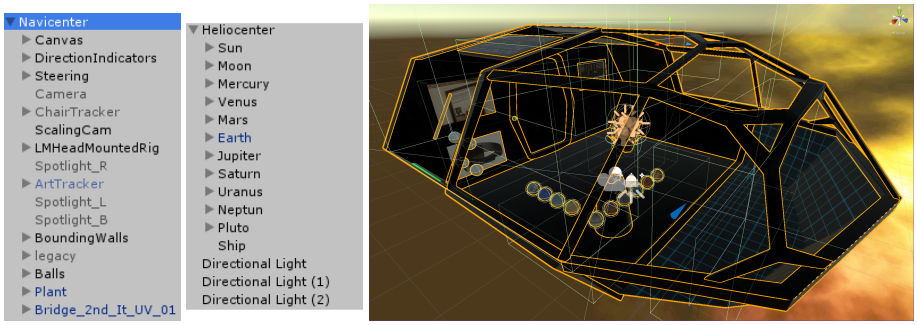
2.1 Projektaufbau

In diesem Abschnitt sollen der Aufbau, bzw. die Struktur des Projektes erläutert werden. Dazu wird zunächst die Projekthierarchie dargestellt und anschließend die verwendeten Elemente genannt. Diese werden als Plugin, Asset oder auch 3D-Modell, im FBX-Dateiformat, in das Projekt importiert bzw. implementiert.

Die Planung und Umsetzung des Projektes wurde in Zusammenarbeit mit Gerald Melles[1] durchgeführt.

Projekthierarchie

In Abbildung 2a und Abbildung 2b ist die Projekthierarchie zu sehen. Dabei ist direkt zu erkennen, dass die Szene in zwei Bereiche unterteilt ist.



(a) Navicenter (b) Heliocenter (c) Cockpit

Abb. 2: Links und Mitte werden die zwei Bestandteile der Projekthierarchie abgebildet. Rechts ist das gesamte Navicenter-Objekt dargestellt.

Der Inhalt vom Objekt **Heliocenter** ist das gesamte Sonnensystem. Inhalt sind unter anderem die Planeten, auch Pluto, und einige Monde. Darüber hinaus ist an das **Heliocenter** die Logik für die Rotation und Positionierung aller dort enthaltenen Elemente angehängt. In diesem Skript kann man des Weiteren auch die Skalierung der verschiedenen Elemente bestimmen und auch die Entfernung dieser zu einander.

Das Objekt **Navicenter** beinhaltet alle Elemente, Modelle und Scripts, die das *Raumschiff* betreffen. Diese sind zum einen das Cockpit und alles was sich in diesem

[1] Gerald Melles - CSTI - HAW Hamburg, vgl. [5]

befindet, aber auch unsichtbare Objekte wie z.B. Kollisionsboxen. An diesem Objekt ist auch die Kamera angehängt. Die Kamera wiederum beinhaltet die Modelle des LeapMotion-Plugins zur Handerkennung und Darstellung. Des Weiteren befindet sich an dem **Navicenter** auch die Logik für die *Bewegung* des Raumschiffs.

Die *Bewegung* ist hierbei der Grund der Trennung des Sonnensystem-Objektes (**Heliocenter**) vom Raumschiff-Objekt (**Navicenter**). Die Herausforderung hierbei ist das Koordinatensystem von Unity3D. Dieses basiert auf dem Datentyp *float* und ist daher begrenzt. Um eine fehlerhafte Darstellung zu vermeiden wurde eine auf den ersten Blick ungewöhnliche Art der Bewegung vom Raumschiff implementiert. Hierbei bleibt das Cockpit stets im Mittelpunkt des Unity3D-Koordinatensystems. Sollte das Raumschiff beschleunigen, so bewegt sich nicht das Cockpit, sondern das Sonnensystem. Dies verhindert eine fehlerhafte Darstellung des Cockpits. Um Fehler bei der Anzeige weit entfernter Objekte zu vermeiden werden diese, ab einer vorher definierten Entfernung, deaktiviert bzw. aktiviert. Dies wäre jedoch bei der Sonne nicht sinnvoll, diese wird daher nicht deaktiviert.

Verwendete Elemente im Projekt

Im vorangegangenen Abschnitt wurde die Projekthierarchie vorgestellt. Darüber hinaus wurden einige Elemente in der vorgestellten Hierarchie genannt. In diesem Abschnitt sollen einige der genannten Elemente näher beschrieben werden. Auch die Techniken, die bei der Implementierung eingesetzt wurden, werden hier benannt.

Sonnensystem

Das Sonnensystem besteht in dieser Simulation aus allen bekannten Planeten unseres Sonnensystems und Pluto. Darüber hinaus sind der Mond der Erde und die beiden Monde vom Mars implementiert. Bis auf die Monde des Mars, Sonne und Erde, sind alle anderen genannten Objekte einfache Kugeln mit einer Textur. Die benötigten Texturen, sowie die beiden Modelle für die Marsmonde, kamen von einer externen Quelle[2].

Sowohl die Erde als auch die Sonne wurden aus dem Asset-Store heruntergeladen und implementiert. Da diese beiden Objekte als Unity3D-Asset verfügbar sind konnten sie einfach implementiert werden und verfügten bereits über Texturen. Die Verwendung von Assets wird in Abschnitt 2.1 näher erläutert.

Cockpit

Für das implementierte Navigationssystem, welches mithilfe eines Pfeils die Richtung zum gewählten Planeten anzeigt, wurde eine verkleinerte Variante des Sonnensystems implementiert. Jedoch ohne die Marsmonde. Darüber hinaus wurde eine ”Weltraum-Blume” als Platzhalter implementiert. Die ”Weltraum-Blume” wurde, wie einige andere Objekte, aus dem Asset-Store bezogen.

Die Modellierung des Cockpits erfolgte mit der Anwendung *Maya*. Die Modellierung und Texturierung wurde von Uli Meyer[3] durchgeführt. Als Vorlage für die Modellierung

[2] Planetery Pixel Emporium - http://planetpixelemporium.com/planets.html; letzter Zugriff: 20.07.2017

[3] Uli Meyer - CSTI - HAW Hamburg - http://csti.haw-hamburg.de/uli-meyer/; letzter Zugriff: 20.07.2017

wurden die Maße der Traverse, vgl. [3, S. 3], verwendet. Diese betragen 5m x 5m. Das Modell des Joysticks, welcher zur Steuerung des Raumschiffs verwendet wird, wurde aus [3, S. 12] entnommen und mit einer Steuerungslogik versehen. Um weitere Interaktionskonzepte testen zu können wurden darüber hinaus einige andere Elemente implementiert. Wie z.B. Bälle, die mithilfe der Leap-Motion, vgl. [3, S. 11], bewegt oder auch auf ein Ziel geschleudert werden können. Auch eine Schiebetür, welche sich durch einen virtuellen Knopf öffnen lässt, wurde getestet.

Unity3D - Plugins

In [3, S. 20] wurde die Plugin-Entwicklung für Unity3D kurz beschrieben. In diesem Abschnitt soll das Thema aufgegriffen werden.

Die Einsatzgebiete von Unity3D-Plugins sind sehr umfangreich. In den meisten Fällen werden entweder komplexe Berechnungen oder das Ansteuern von Hardware in Plugins ausgelagert. Dabei wird zwischen zwei Plugin-Arten unterschieden. In den folgenden Abschnitten werden diese beiden Arten, anhand von Beispielen, beschrieben.

Managed-Plugins (Astro-Lib)

Managed-Plugins basieren auf einem .NET[4] Code. Daher unterscheiden sich Managed-Plugins kaum vom Unity3D Script Code. Darüber hinaus sind Managed-Plugins auch Plattform unabhängig. Somit müssen Managed-Plugins nicht auf andere Plattformen portiert werden.

Das Erstellen und Implementieren von Managed-Plugins ist in [7] genau beschrieben. Zum Erstellen von Managed-Plugins wird generell die *UnityEngine.dll* benötigt, welche als Referenz im Projekt eingetragen werden muss. Der Aufruf erfolgt dabei über die öffentlichen (public) Methoden.

Mithilfe der oben genannten Quelle wurde für die Raumschiff-Simulation ein Managed-Plugin erstellt. Die AstroLib[5] ist für die Berechnung der Planetenposition zuständig. Hierfür werden die folgenden Methoden bereitgestellt:

- AstroLib: Berechnet die Position der Planeten anhand eines Datums und übergibt den Größenmultiplikator der Himmelskörper.
  - *public static double[] calcAstroElement(AstroElementEnum aee, double day)*: Berechnet die Position des übergebenen Himmelskörper (AstroElementEnum) für das übergebene Datum (day).
  - *public static double[] calcAstroElement(AstroElementEnum aee, double day, CoordinateSystemTypeEnum cste)*: Wie die vorangegangene Methode mit dem Zusatz das die Position in einem bestimmten Koordinatensystemtyp übergeben wird.
  - *public static double getSizeMultiplierComparedToEarth(AstroElementEnum aee)*: Übergibt den Größenmultiplikator für den übergebenen Himmelskörper, dabei hat die Erde die Größe 1. Z.B. die Sonne, dabei ist der Rückgabewert 109.3, die Sonne ist somit 109.3 mal größer als die Erde.

[4] Dot-Net - Microsoft - http://www.microsoft.com/net; letzter Zugriff: 20.07.2017

[5] AstroLib - Johann Bronsch - CSTI - http://git.csti.haw-hamburg.de/Johann.Bronsch/AstroLibCS; letzter Zugriff: 20.07.2017

- DateCalculator: Für die Berechnung der Planetenpositionen wird ein Datum benötigt. Die Schnittstelle berechnet das Datum, entweder anhand von übergebenen Werten oder das Datum zum Abfragezeitpunkt.
  - *public DateCalculator(int day, int month, int year, int hour, int minute, int seconds)*: Konstruktor für den Calculator, dabei wird das gewünschte Datum übergeben.
  - *public DateCalculator()*: Konstruktor für den Calculator, hierbei wird das Datum der Initialisierung des Objektes verwendet.
  - *public double addSecond(int second)*: Fügt der erstellten Instanz n-Sekunden hinzu.
  - *public static double calculateCurrentDate()*: Statische-Methode zur Berechnung des Datums, es wird das Abfragedatum für die Berechnung verwendet.
  - *public static double calculateDate(int day, int month, int year, int hour, int minute, int seconds)*: Statische-Methode zur Berechnung des Datums anhand der übergebenen Werte.
  - *private static double calculateHours(double hour, double minute, double seconds)*: Statische-Methode zur Berechnung des Datums. Es wird das Datum zur Abfragezeit verwendet, die Stunden, Minuten und Sekunden werden der Methode übergeben.

Die beschriebenen Methoden können, solange die Compilierte-Bibliothek im DLL-Dateiformat sich im Plugin-Ordner befindet, wie folgt im Unity3D Script Code aufgerufen werden:

```
double[] positionEarth = AstroLib.calcAstroElement(AstroElementEnum.EARTH,
DateCalculator.calculateCurrentDate());
```

Hierbei wird die Position der Erde zum jetzigen Zeitpunkt abgefragt. Diese befinden sich in einem Double-Array, welcher wie folgt aufgeschlüsselt wird:

- positionEarth[0]: X-Achse
- positionEarth[1]: Y-Achse
- positionEarth[2]: Z-Achse

Native-Plugins (Middleware/Microservices)

Native-Plugins eignen sich sehr gut um direkt auf die Ressourcen eines Betriebssystems zuzugreifen, wie z.B. bei aufwendigen Algorithmen, oder auf Bibliotheken von anderen Systemen. Diese sind jedoch, im Gegensatz zu den Managed-Plugins, Plattform-Spezifisch und müssen daher für jede Plattform einzeln entwickelt werden. Sie sind daher nicht so flexibel wie die Managed-Plugins.

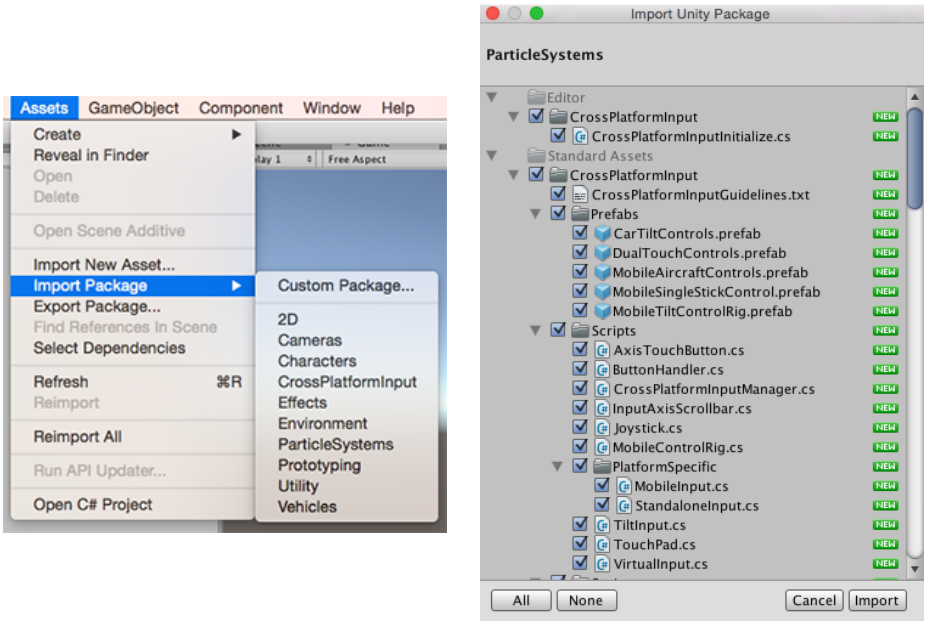
Der Vorteil ist jedoch, dass mithilfe von Nativen Plugins unter anderem angeschlossene Hardware direkt angesprochen werden kann und das jener nicht auf Dot-Net basieren muss. Dies ermöglicht eine freiere Gestaltung des benötigten Plugins.

Die Laborumgebung des CSTI verfügt über eine eigene Middleware. Diese ermöglicht es den Studenten Prototypen zu entwickeln, die über eine gemeinsame Schnittstelle kommunizieren können. Daher wurde ein Natives Plugin entwickelt, welches diese Funktionalität in Unity3D bereitstellt. Jenes ermöglicht sowohl den Empfang, als auch den Versand von Daten aus der Middleware. Dieses Plugin ist zum Zeitpunkt der Veröffentlichung dieser Ausarbeitung nur für Windows-Systeme verfügbar.

Unity3D - Assets

Wie in [3, S. 20] bereits beschrieben sind Unity-Assets Container, die ein ganzes Projekt, oder nur ein Script, für die Lösung von einem Problem beinhalten können. Sie einigen sich besonders für die Wiederverwendung von sich wiederholenden Ereignissen, wie z.B. das Leap-Motion-Asset[6]. Dies beinhaltet nicht nur alle notwendigen Bibliotheken, sondern auch die Handmodelle und Beispielszenarios.

Unity-Assets können dabei von einer externen Quelle kommen, oder aus dem Asset-Store, welcher sich auch in der Entwicklungsumgebung befindet. Das Importieren von Assets aus einer externen Quelle wird wie in Abbildung 3 durchgeführt. Dazu wählt man, im Abbildung 3a dargestellten Menü, den Unterpunkt *Custom Package* und sucht im Dateisystem das gewünschte Unity-Asset aus. Nach dem man das gewünschte Package ausgewählt hat erscheint das *Import Unity Package*. Dies zeigt alle Bestandteile eines Assets an. Es ist unter anderem auch möglich nur bestimmte Teile eines Asstes zu importieren.



(a) Import Package Menu [6] (b) Import Package Dialog [6]

Abb. 3: Importieren von Unity-Assets aus einer externen Quelle

Um ein eigenes Unity-Asset zu exportieren muss, in dem Abbildung 3b dargestellten Menu, der Unterpunkt *Export Package* ausgewählt werden. Anschließend sollte der

[6] Leap-Motion Asset - LEAP MOTION, INC. - http://developer.leapmotion.com/unity; letzter Zugriff: 20.07.2017

in Abbildung 4 dargestellte Dialog erscheinen. In jenem können dann die einzelnen Elemente ausgewählt werden, die in das eigene Asset implementiert werden sollen. Diese können dann in andere Projekt importiert werden und, wenn benötigt, auch angepasst werden.

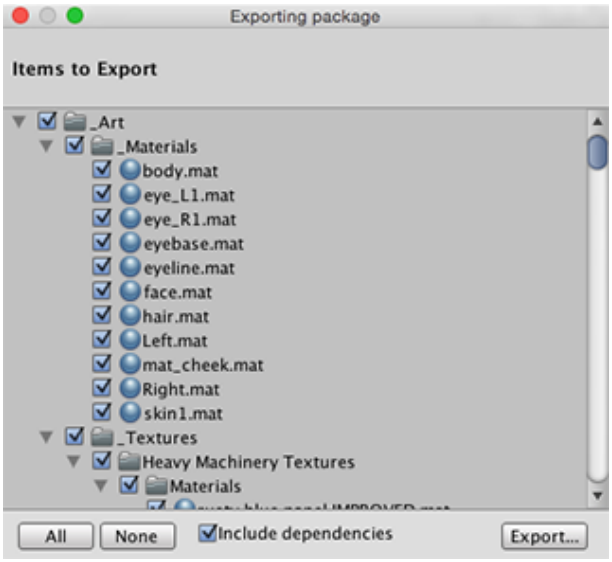


Abb. 4: Export Package Dialog [6]

2.2 Arten der Interaktionen

Die Interaktionen in virtuellen Welten trägt entscheidend zum Grad der Immersion bei. Hierbei existieren verschieden Interaktionsmodi, welche auch auf unterschiedlichste Art und Weise klassifiziert werden können. In dieser Arbeit wird jedoch die Klassifizierung aus [2] gewählt. Anhand dieser Klassifizierung wurden einige Interaktionen in die Simulation implementiert und werden an dieser Stelle beschrieben.

Realmodell

Wie bereits in Abschnitt 2.1 beschrieben wurde ein Joystick aus [3, S. 12] verwendet um das Raumschiff zu steuern. Dieser wurde nach seinem realen Vorbild modelliert, daher entspricht die Darstellung in der virtuellen Welt weitestgehend dem realen Abbild. Da dieser die selben Funktionen in der realen und virtuellen Welt besitzt, kann jener nach [2] als Realmodell eingestuft werden.

Der Vorteil hierbei ist, dass der Anwender, anhand von Erfahrungswerten oder durch eine Erwartungshaltung, meist ohne eine lange Einweisung mit dieser Art der Interaktion zurecht kommt. Ein Beispiel der Implementierung wird in Abbildung 5

dargestellt. Hierbei werden mithilfe der Leap-Motion die Hände in die virtuelle Welt übertragen. Des Weiteren wird das virtuelle Modell des realen Joysticks angezeigt.

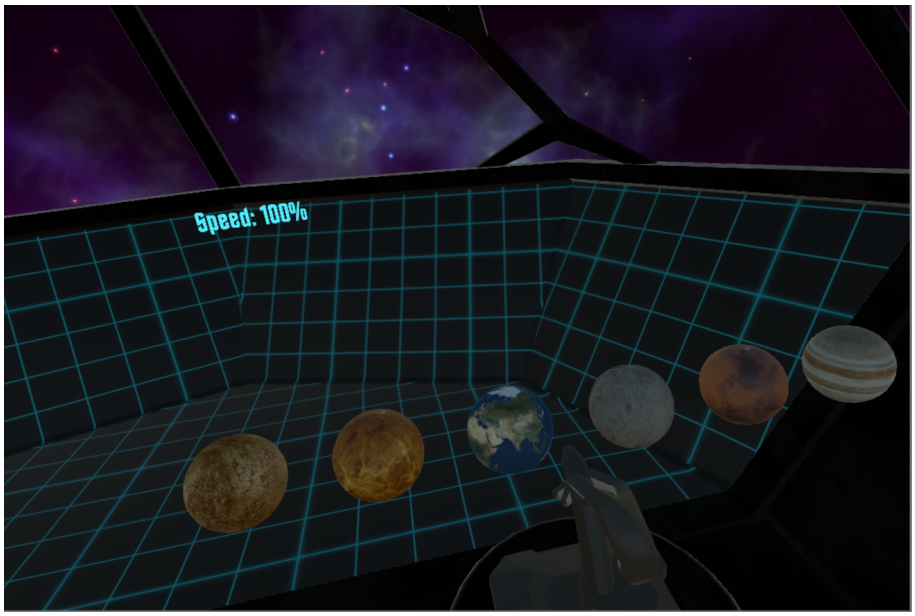


Abb. 5: Anzeige eines Joysticks und der Hände, mithilfe der Leap-Motion, in der virtuellen Realität.

Mixed-Reality Interfaces

Mixed-Reality Interfaces sind reale Objekte, die in der virtuellen Welt mit zusätzlichen Funktionen erweitert werden. Je ähnlicher das virtuelle Abbild dem realen Objekt entspricht, desto immersiver wirkt die Simulation.

Ein Beispiel solch eines Mixed-Reality Interfaces ist das Lichtschwert aus *Lightblade VR*[7]. Hierbei wird der HTC Vive Controller als Lichtschwert dargestellt, was ihn nach [2] zu einem Mixed-Reality Interface macht.

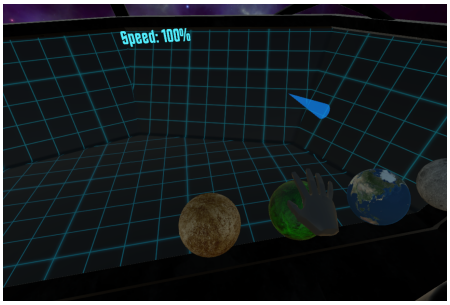
In diesem Projekt entspricht die Laborumgebung, genauer die Traverse, in der die Simulation statt findet, einem Mixed-Reality Interfaces. Dabei wird die bereitgestellte Fläche innerhalb der Traverse zu einem Raumschiff-Cockpit. Die Wände der Traverse werden zu der Innenverkleidung des Cockpits.

[7] Lightblade VR - Andreas Hager Gaming - http://lightbladevr.mazebert.com/; letzter Zugriff: 20.07.2017

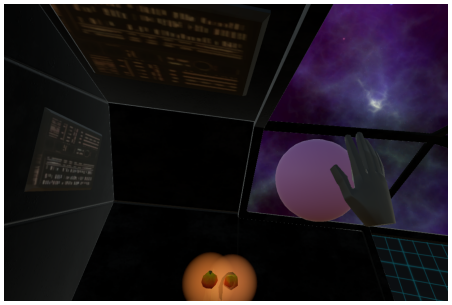
Vollständig virtuelle Interfaces

Eines der Alleinstellungsmerkmale von Virtual Reality ist die Darstellung von Objekten oder Umgebungen, die in der Natur nicht vorkommen oder mit heutigem Technologiestand nicht umsetzbar sind. Solche Objekte können zum einen ein Lichtschwert oder auch ein fliegendes Schwein sein. Darüber hinaus können Umgebungen geschaffen werden, die zwar vorstellbar sein können, aber zum jetzigen Zeitpunkt nicht umsetzbar sind. Z.B. ein Raumschiff-Cockpit mit eigener Schwerkraft, wie in diesem Projekt.

Um die Funktionsweise und Interaktion mit solchen Objekten und Umgebungen zu Testen wurden diese im Projekt umgesetzt, vgl. Abbildung 6a und Abbildung 6b.



(a) Bedienung des Navigationssystems mithilfe der Leap-Motion



(b) Bewegung eines virtuellen Balles mithilfe der Leap-Motion

3 Fazit

3.1 Derzeitiger Stand

Die Umsetzung der Simulation wurde anhand den Abbildungen in dieser Ausarbeitung dargestellt. Dies zeigt, dass die Umsetzung komplexer Projekte mit einer Virtual-Reality Toolchain machbar ist.

Hierfür wurde ein komplexer Interaktionsraum geschaffen in dem der Anwender nicht nur mit den Objekten im Cockpit interagieren konnte, sondern auch mit der virtuellen Welt außerhalb des begehbaren Raumes. Um die Interaktion in der somit geschaffenen Welt attraktiver zu gestalten, wurde eine möglichst realistisches Sonnensystem nachgebildet. Hierbei lag der Fokus auf eine gewohnte Darstellung aus namhaften Science-Fiction Filmen.

Innerhalb des Projektes konnte die Immersion nicht anhand von Fragebögen, wie in [2] beschrieben, gemessen werden. Die Nutzererfahrungen während einer Vorstellung der Simulation zeigten, dass der Großteil der Anwender sich als Teil der Simulation sah. Darüber hinaus hatten einige Nutzer auch reale Emotionen gezeigt, wie z.B. beim Öffnen der virtuellen Tür, vgl. Abbildung 7. Einige Nutzer fühlten dabei ein Unbehagen oder Furcht.

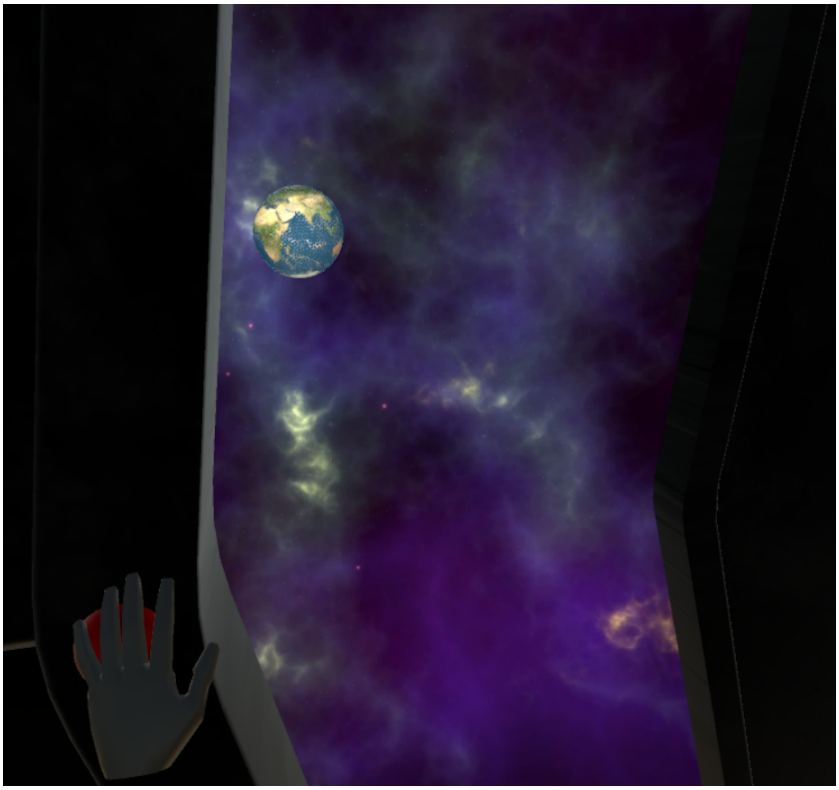


Abb. 7: Offene Cockpit-Tür mit Ausblick auf die virtuelle Erde

3.2 Ausblick

Zukünftige Arbeiten können sich mit der Implementierung eines haptischen Feedbacks beschäftigen und jene als Unity-Asset anbieten. Jene Arbeiten können dabei auf den Erkenntnissen von [8] aufbauen.

Ein weiterer interessanter Forschungsbereich ist der Einsatz der Virtual-Reality Technologie im eLearning. Somit könnte z.B. angehende Triebwerksmechaniker an einem virtuellen Triebwerk üben ohne reale Konsequenzen befürchten zu müssen. Die hier aufgestellte Fragestellung könnte z.B. lauten: *"Wie hoch ist der Lerneffekt in der virtuellen Welt im Vergleich zur realen Welt?"*.

Mit der hier beschriebenen Toolchain können jedoch auch weitere Szenarien erstellt werden, welche die unterschiedlichsten Fragestellungen beantworten könnten. Solch eine Frage könnte z.B. die folgende sein: *"Wann sind virtuelle Welten Erwartungskonform[8]?"*.

[8] ISO 9241:10 - Handbuch Usability - http://www.handbuch-usability.de/erwartungskonformitaet.html; letzter Zugriff: 20.07.2017

Literatur

1. Bronsch, J.: Vergleich von virtual-und augmented-reality in bezug auf deren gemeinsamkeiten und probleme. Tech. rep., Hochschule für Angewandte Wissenschaften Hamburg, http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master2015-gsem/bronsch/bericht.pdf (2016)
2. Bronsch, J.: Interaktionen in der virtuellen realität. Tech. rep., Hochschule für Angewandte Wissenschaften Hamburg, http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master2016-hsem/bronsch/bericht.pdf (2017)
3. Bronsch, J.: Virtual-reality toolchain. Tech. rep., Hochschule für Angewandte Wissenschaften Hamburg, http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master2017-proj/bronsch.pdf (2017)
4. Eichler, T.: Agentenbasierte middleware zur entwicklerunterstützung in einem smart-home-labor (2014), http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/eichler.pdf
5. Melles, G.: Internetauftritt von gerald melles (2017), http://geraldmelles.com/
6. Unity3D: Asset packages (2017), http://docs.unity3d.com/Manual/AssetPackages.html
7. Unity3D: Unity3d plugin-tutorial (2017), http://docs.unity3d.com/Manual/UsingDLL.html
8. Wortmann, H.: Iterative konstruktion eines individualisierten force-feedback handschuhs mittels rapid-prototyping-techniken (2017), http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/wortmann.pdf