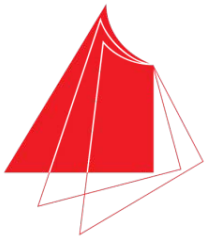

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hochschule Karlsruhe

Kommunikation und Medienmanagement

Projektbericht

Erstellung einer Serious-Gaming-Anwendung für die Oculus Rift

T3M120/130 – Media-Engineering

Fabian Steiner

13. Februar 2015

Prüfer: Prof. Dr. Martin Schober (Hochschule Karlsruhe)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abk.	Abkürzung
ANE	Acquiring Native Extension
DK2	Development Kit 2
Full HD	Full High Definition
GUI	Graphical User Interface
HMD	Head Mounted Display
HUD	Head-up-Display
SDK	Source Development Kit
VR	Virtual Reality
XML	Extensible Markup Language

1. Grundlagen und Überblick

1.1 Ansatz der Projektarbeit

Die Projektarbeit soll in erster Linie die Möglichkeiten der Oculus Rift für einen potenziellen Einsatz im Bereich der Technischen Redaktion betrachten. Dazu wird exemplarisch eine Anwendung konzipiert, die sich die Funktionalitäten der Oculus Rift für die freie Navigation innerhalb einer virtuellen Umgebung zu Nutze macht.

Mit der Anwendung soll der Benutzer Abläufe in einer sicheren Umgebung durchführen und unter annähernd identischen Bedingungen jederzeit wiederholen können.

1.2 Aspekte der Mediendidaktik

1.2.1 Erkennbarkeit und Klarheit

Der Benutzer muss im Stande sein, Interaktionselemente vom Rest der Anwendung unterscheiden zu können. In einer 3D-Umgebung gibt es verschiedene Objekte, die sich in ihrer Geometrie, Ausrichtung, Farbe sowie evtl. auch Textur voneinander unterscheiden. Elemente, mit denen der Benutzer interagieren kann, müssen sich daher zusätzlich von den verschiedenen Formen der statischen Objekte abheben. Dies kann z. B. durch die Größe der Hervorhebung oder den Einsatz von Signalfarben erreicht werden, die bei den restlichen 3D-Modellen nicht oder nur in einem geringen Anteil vorkommen.

1.2.2 Vermeidung von Wahrnehmungsüberlastung

Der Benutzer sollte nicht mit zu vielen gleichzeitig dargestellten Informationen überlastet werden. Dies kann durch Ausblenden nicht relevanter Informationen erfolgen. Dabei ist aber zu beachten, dass der Benutzer die mit dem Ausblenden verbundenen Änderungen erkennen und auch nachvollziehen kann.

1.2.3 Verstehen der Anwendung

Soll mit einer Anwendung ein praktischer Nutzen verbunden sein, muss der Benutzer wissen, welche Möglichkeiten sie ihm bietet und wie er diese nutzen kann. Er muss die logisch-funktionalen Zusammenhänge erkennen und verstehen können. Dazu ist es essenziell, dass der Benutzer weiß, wie er die Anwendung bedienen muss. Im Falle einer 3D-Umgebung muss er wissen, wie er im virtuellen Raum navigieren und wie er mit den in 1.2.1 beschriebenen Interaktionselementen nutzen kann.

1.2.4 Benutzungsmotivation

Die Anwendung sollte anregend gestaltet sein. Wenn es für die mit der Anwendung durchgeführte Aufgabe angebracht ist, soll sie die Aufmerksamkeit des Benutzers auf sich ziehen und ihn dazu ermutigen, sich gezielt mit ihr auseinanderzusetzen. Dies kann durch eine Kombination von hohem Realitätsgrad und einer hohen Anzahl an Interaktionsmöglichkeiten erreicht werden.

1.2.5 Fehlertoleranz

Der Benutzer muss die Möglichkeit haben, auch falsche Eingaben machen zu dürfen. Diese müssen von der Anwendung erkannt und verarbeitet werden, ohne dass sie sich schwerwiegend auf die Stabilität der Anwendung auswirken. Darüber hinaus soll der Benutzer wieder an die vorherige Position wechseln können, nachdem er z. B. ein falsches Element gewählt hat.

1.2.6 Kognitive Aspekte

Für die Entwicklung muss berücksichtigt werden, dass Benutzer unterschiedlich schnell mit einer Anwendung und ihren Bedienmöglichkeiten zurechtkommen. Persönliche Auffassungsgabe sowie die Zeit, die sich ein Benutzer mit der Anwendung beschäftigt, haben ebenso Auswirkung auf den Lerneffekt, wie die Komplexität der Anwendung und das umgesetzte Informationsdesign.

1.2.7 Weitere Anforderungen

Andere Anforderungen wie die Berücksichtigung von Wahrnehmungsunterschieden - z. B. bei Personen mit Farbenblindheit - werden für dieses Projekt gezielt nicht betrachtet, da diese in Hinblick auf die mit dem Prototyp zu prüfende technische Umsetzbarkeit keine zentrale Rolle spielen. Dies trifft auch auf Kriterien wie z. B. Lesbarkeit zu, da dies für die Umsetzung nicht von zentraler Relevanz ist.

2. Oculus Rift DK2

Bei der Oculus Rift handelt es sich um ein sog. *Virtual Reality Head-Mounted Display* (VR HMD) des amerikanischen Herstellers *Oculus VR*. Die Abkürzung DK2 steht für “Development Kit 2”, also einer Entwicklerversion.

Das Headset der Oculus Rift besteht im Kern aus einem Display mit einer Auflösung von 1920 x 1080 px (Full HD) und verfügt über eine Reihe von Sensoren für die Erkennung von Rotations- und Neigungsbewegungen sowie der Ausrichtung des Headsets innerhalb des Erdmagnetfelds zur Richtungserkennung.

Die DK2 kann nativ an einen HDMI-Anschluss oder mit Hilfe des beiliegenden Adapters auch an DVI angeschlossen werden. Eine weitere Anschluss-möglichkeit bildet der DisplayPort, hier ist allerdings auch ein Adapter auf HDMI erforderlich und die Performance bei der Darstellung kann unter Umständen beeinträchtigt sein.

Die Oculus Rift rendert für jedes Auge ein separates Bild, das die 3D-Szene leicht versetzt zeigt. Dies hilft bei der Simulation der räumlichen Darstellung.

Nr.	Beschreibung
1	Ein-/Aus-Schalter
2	Status-LED
3	Zusätzlicher USB- und Sync-Kabel-Anschluss
4	Anschlussbox für das HDMI-/USB-Datenkabel
5	Schaumstoffpolster für angenehmen Sitz am Kopf
6	Linsen
7	kombiniertes HDMI-/USB-Datenkabel
8	verstellbares Kopfband
9	Einstellrad für Abstand zwischen Augen und Display

Abbildung 1: Übersicht der Oculus Rift

Zur Erkennung der relativen Position im Raum verfügt die Oculus Rift über mehrere Infrarot-LEDs, die von der zusätzlich im DK2-Set enthaltenen Tracking-Kamera erkannt werden. Die Kamera kann so die relative Position der VR-Brille im Raum berechnen und die Bewegungen des Benutzers nachverfolgen. Die Kamera wird ähnlich wie eine Webcam über USB mit dem Computer sowie über ein separates Sync-Kabel mit der Anschlussbox der DK2 verbunden.

Abbildung 2: Tracking-Kamera des DK2-Sets

3. Konzeption

3.1 Idee

Die Kernidee zur Umsetzung in der Anwendung ist ein mögliches Szenario für einen sog. Preflight-Check eines Flugzeugs. Die Konzeption dieses Szenarios ist dabei rein exemplarisch und erhebt keinerlei Anspruch auf Vollständigkeit oder technische Korrektheit.

Vor dem Start macht der Pilot einen obligatorischen Rundgang um das Flugzeug, bei dem er sicherheitsrelevante Aspekte sowie den technischen Zustand der Maschine prüft.

Basierend darauf werden im Rahmen dieser Anwendung folgende exemplarische Aufgaben für den Benutzer betrachtet:

- Einsammeln von „Remove before flight“-Tags
- Aufräumen von Werkzeug
- Prüfen von Flugzeug und Umgebung auf Fremdkörper und Trümmerteile
- Prüfen des Flugzeugs auf Beschädigungen

Der Benutzer wird mit der Anwendung in die Rolle eines Piloten versetzt und muss diese Aufgaben bei einem Rundgang um das Flugzeug durchführen. Um den Benutzer zu einer gründlicheren Durchführung der Aufgaben zu verleiten, wird die Anzahl einzusammelnder RBF-Tags, Werkzeuge, Trümmerteile oder zu reparierender Beschädigungen im Vorfeld zwar nicht explizit bekannt gegeben – er kann sich aber jederzeit über seinen aktuellen Fortschritt durch Einblendung einer Statistik informieren.

3.2 Funktionsumfang der Anwendung

Die Anwendung soll es dem Benutzer ermöglichen, die in 3.1 beschriebenen Aktionen in einer virtuellen Umgebung durchführen und auf diese Weise die damit verbundenen Abläufe trainieren zu können. Dabei macht sich die Anwendung in erster Linie die Funktionsweise der Oculus Rift zu Nutze, mit der sich der Benutzer in der virtuellen Welt genauso umsehen und Gegenstände betrachten kann, wie er es von der realen Welt gewohnt ist.

3.3 Software und Tools

3.3.1 Entwicklungsumgebung

Es gibt verschieden Möglichkeiten, Anwendungen für die Oculus Rift zu erstellen. Je nach persönlicher Vorliebe und Vorkenntnisse des Entwicklers kommen andere Softwaretools in Frage.

Im Rahmen des Projekts wurden folgenden Entwicklungsmöglichkeiten in die engere Wahl genommen:

- nativ mit der Oculus SDK
- mit Unity3D über das *Oculus Integration Package*
- Oculus ANE für Adobe Flash

Jede dieser Entwicklungsarten bietet verschiedene Möglichkeiten, eine Anwendung zu erstellen.

3.3.2 Auswahl einer Entwicklungsumgebung

Basierend auf der in 3.3.1 getroffenen Vorauswahl werden die einzelnen Entwicklungsmöglichkeiten im Folgenden anhand einer Reihe von Kriterien verglichen und hinsichtlich ihrer Tauglichkeit bewertet.

		Maximum		Oculus Rift SDK (nativ mit Visual Studio o.ä.)		Unity 4 (Oculus Integration Package)		Adobe Flash Professional CS6 (Flare3D + Oculus ANE)	
Kriterium	**G**	**P**	**G x P**	**P**	**G x P**	**P**	**G x P**	**P**	**G x P**
Allgemeine Bedienung									
WYSIWYG-Editor	3	3	9	0	0	3	9	1	3
Erlernbarkeit	3	3	9	1	3	3	9	2	6
Umfang Dokumentation	2	3	6	3	6	3	6	2	4
Σ			**24**		**9**		**24**		**13**
3D-Funktionalitäten									
Umfang	1	3	3	3	3	3	3	2	2
Handhabung	3	3	9	1	3	3	9	1	3
Unterstützung gängiger 3D-Formate	3	3	9	3	9	3	9	2	6
Performance	3	3	9	3	9	3	9	2	6
Σ			**30**		**24**		**30**		**17**
Unterstützung Oculus Rift									
Darstellung	3	3	9	3	9	3	9	3	9
Sensoren für Kopfbewegung	3	3	9	3	9	3	9	2	6
Tracking-Kamera	3	3	9	3	9	3	9	0	0
Σ			**27**		**27**		**27**		**15**
Gesamt			**81**		**60**		**81**		**45**
Prozent		100%		**74%**		**100%**		**56%**	

Tabelle 1: Auswahl einer Entwicklungsumgebung für die Oculus Rift

Adobe Flash und die 3D-Engine Flare3D bieten einfache Möglichkeiten, 3D-Inhalte in gewohnter Flash-Umgebung mit ActionScript zu erstellen. Zum Zeitpunkt der Recherche ist die Oculus ANE für Adobe Flash allerdings noch nicht so ausgereift, dass alle Funktionen der DK2 unterstützt werden. Aus diesem Grund scheidet diese Variante für die Auswahl einer Entwicklungsumgebung aus.

Die native Entwicklung mit Visual Studio und der Oculus SDK ist der direkteste Weg, Anwendungen für die DK2 zu erstellen. Die Entwicklung erfolgt direkt in C++ und eignet sich daher nur für versierte Programmierer.

Unity bietet einen guten Kompromiss zwischen Funktionalität und einfacher Handhabung. Bei Unity handelt es sich um ein Autorensystem, in dem Benutzer sowohl 2D- als auch 3D-Inhalte in einer Art Baukastensystem im erstellen können. Es eignet sich für Personen, die im Vorfeld keine oder nur wenig Erfahrung mit 3D-Software oder auch Programmierung gemacht haben. Eine im Hintergrund arbeitende Gameengine kümmert sich um die nötige Darstellung – der Benutzer kann sich auf den Inhalt konzentrieren und die (3D-)Szene nach seinen Wünschen zusammenstellen. Interaktivität lässt sich dabei über Skripte ergänzen. Diese müssen zwar auch programmiert werden, sind aber nicht so komplex wie die zuvor beschriebene native Entwicklung.

3.4 Gestaltung der Anwendung

3.4.1 Bildschirmeinteilung

Eine VR-Anwendung dieser Art benötigt keine reguläre Bildschirmeinteilung im Sinne von Navigations- oder Inhaltsbereichen. Die gesamte Darstellung der 3D-Szene ist Teil des Inhalts. Stattdessen gliedert sich der Aspekt der Einteilung darstellbarer Inhalte in zwei Bereiche:

- Elemente im 3D-Raum
- On-Screen-Elemente, z. B. Head-up-Display (HUD) o.ä.

Bei HUDs sind Informationselemente relativ zum Benutzer in dessen Sichtfeld positioniert und folgen den HUD-Elemente sind entweder dauerhaft sichtbar, nur bei Bedarf (z. B. zur Kennzeichnung einer Interaktionsmöglichkeit) oder werden auf Eingabe des Benutzers (z. B. Tastendruck) eingeblendet.

3.4.2 Navigation in der Anwendung

Da es sich um sog. „Serious Game“ handelt (also ein Spiel mit ernstem Hintergrund), orientieren sich die Navigationsmöglichkeiten in der 3D-Szene an denen eines normalen 3D-Spiels, z. B. eines First-Person-Shooters.

Die Anwendung nutzt dabei die Möglichkeiten der Oculus Rift. Der Benutzer soll sich in der virtuellen Umgebung genauso umsehen können, wie in der realen Welt. Für Bewegung innerhalb der 3D-Szene werden Steuerungsmöglichkeiten bereitgestellt, wie sie in 3D-Spielen geläufig sind.

3.4.3 Interaktionselemente

Im Laufe der Anwendung hat der Benutzer die Möglichkeit, mit verschiedenen Objekten in Interaktion zu treten. Die Objekte sind alle Teil der 3D-Welt und werden teilweise zur besseren Unterscheidung von statischen Objekten farblich hervorgehoben.

4. Entwicklung mit Unity für die Oculus Rift DK2

4.1 Voraussetzungen

Um die Oculus Rift mit Unity nutzen zu können, werden folgende Softwarepakete benötigt:

- Unity 4.5 (oder neuer)
- Oculus Runtime
- Unity 4 Integration

Unity kann in einer Free-Edition direkt von der offiziellen Herstellerseite heruntergeladen werden: http://unity3d.com/unity/download

Die Oculus Runtime sowie das Integration-Package für Unity werden von Oculus VR ebenfalls zum direkten Download angeboten: https://developer.oculus.com/downloads/. Beim Herunterladen ist dabei zu achten, dass die Pakete in derselben Version (z. B. 0.4.3-beta) heruntergeladen werden.

Im Folgenden wird nur grob auf die einzelnen Installtionen eingegangen. Der Schwerpunkt dieses Kapitels liegt auf der Einrichtung von Unity und dem nötigen Zusammenspiel für die Verwendung der Oculus Rift.

4.2 Installation der Oculus Rift Runtime

Damit die Oculus Rift an einem Computer funktioniert, muss die Oculus Rift Runtime installiert werden. Dabei werden alle nötigen Treiber installiert. Die Runtime stellt später das Hauptinterface zwischen dem Computer und der Oculus Rift dar und ermöglicht die Nutzung der Funktionalitäten der VR-Brille.

4.3 Einrichten der Oculus Rift

Nach Installation der Runtime muss die Rift eingerichtet werden. Dazu wird im Configuration Utility mit der Schaltfläche „+“ ein neues Benutzerprofil angelegt (Abbildung 3).

Abbildung 3: Konfigurationsfenster für die wichtigsten Einstellungen der Rift

Neben einem Profilnamen (Abbildung 4) gibt man dabei das Geschlecht sowie die Körpergröße an (Abbildung 5). Die eingetragenen Werte haben später Einfluss auf die Darstellung der 3D-Szenen in der Oculus Rift.

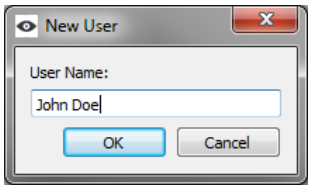

Abbildung 4: Anlegen eines neuen Benutzerprofils für die Oculus Rift

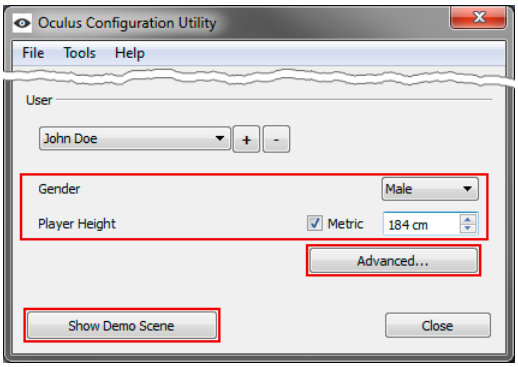

Abbildung 5: Allgemeine Einstellungen für das Benutzerprofil

Genauere Angaben können in einem separaten Dialogfeld gemacht werden, das über die Schaltfläche „Advanced“ zugänglich ist (Abbildung 5). Dort können

detaillierte Angaben zum Augenabstand gemacht werden, die direkten Einfluss auf die Simulation des räumlichen Sehens haben. Für eine exaktere Kalibrierung kann über die Schaltfläche „Measure“ eine automatisierte Messung durchgeführt werden.

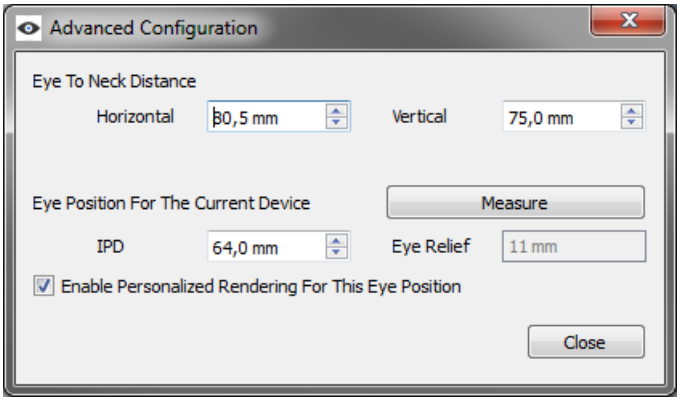

Abbildung 6: Dialogfenster für erweiterte Einstellungen

Zum Testen und Verfeinern der vorgenommenen Einstellungen bietet das Konfigurationsfenster eine kleine Testszene, die über die Schaltfläche „Show Demo Scene“ aufgerufen werden kann (Abbildung 5, unten).

Es wird empfohlen, für jeden Benutzer ein eigenes Profil anzulegen, da kleine Abweichungen der Werte großen Einfluss auf die visuelle Darstellung haben können und die Benutzer visuell etwas anderes vermittelt bekommen könnten, als ihre Sinnesorgane gewohnt sind.

Das Konfigurationsfenster bietet darüber hinaus die Möglichkeit, bei Bedarf den Anzeigemodus der Rift umzuschalten. Im Direktmodus wird die Oculus Rift nur dann als Anzeigegerät genutzt, wenn eine Anwendung gestartet wird, die die VR-Brille unterstützt. Der andere Anzeigemodus bindet die Oculus Rift als zweiten Monitor im erweiterten Desktop ein.

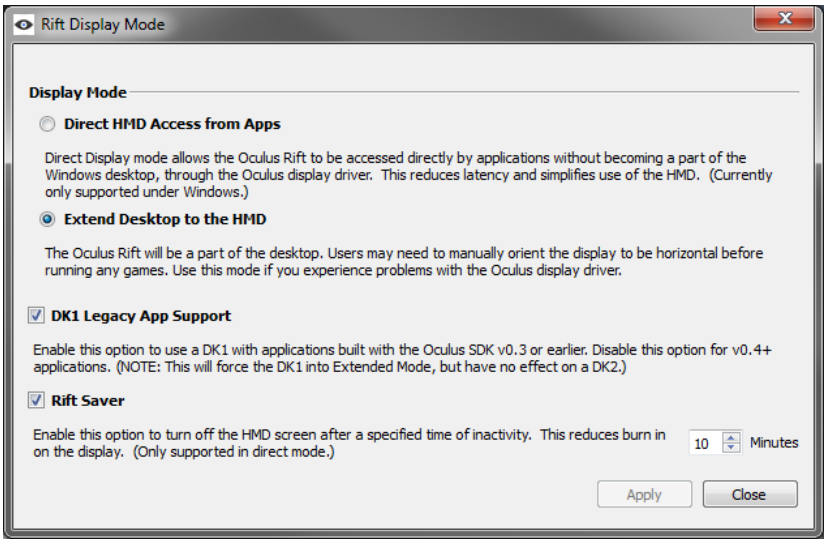

Abbildung 7: Anzeigemodi der Oculus Rift

Bei Verwendung der DK2 an einem HDMI-Port reicht in den meisten Fällen der Direktmodus – dieser bietet auch die beste Performance. Soll die VR-Brille an einem Computer ohne HDMI-Ausgang angeschlossen werden, benötigt man einen Adapter (z. B. den mitgelieferten DVI-zu-HDMI- oder einen DisplayPort-zu-HDMI-Adapter). Bei Bedarf muss der Anzeigemodus auf „Erweiterter Desktop“ umgestellt und die Bildschirmausrichtung der Oculus Rift in den Systemeinstellungen gedreht werden.

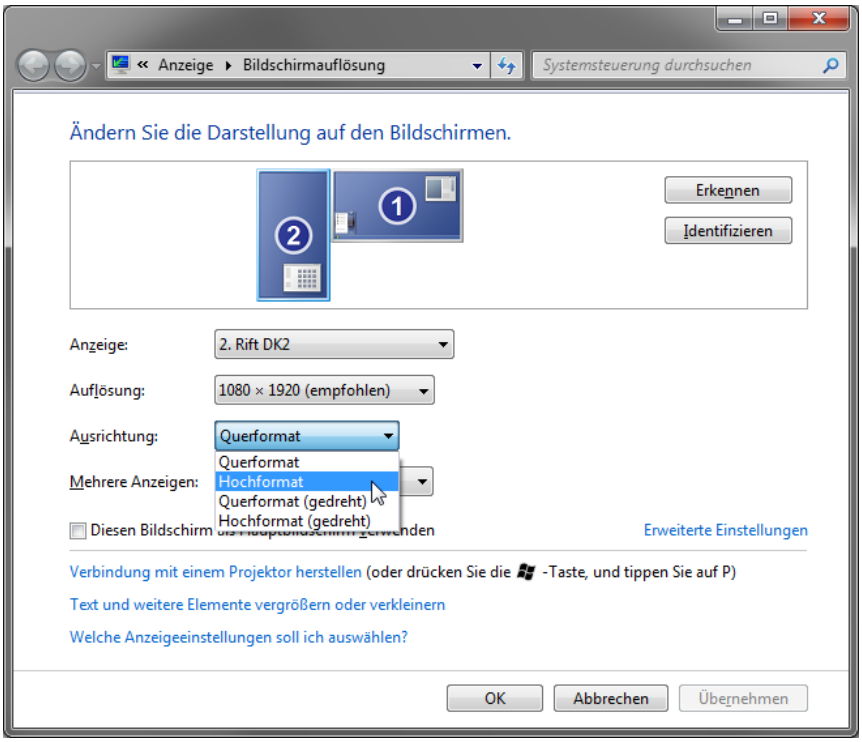

Abbildung 8: Im "Erweiterten Desktop"-Modus ist eine Anpassung der Bildschirmausrichtung notwendig

4.4 Installation Unity

Nach der Einrichtung der Oculus Rift am Computer erfolgt die Installation von Unity. Für die Oculus Rift ist mind. die Unity-Version 4.5 erforderlich. Die Installation einer neueren Version oder ein entsprechendes Update kann sinnvoll sein, wenn gezielt Funktionen der neueren Unity-Versionen genutzt werden sollen.

Die Anwendung in diesem Projekt selbst wird mit Unity 4.6 erstellt, da mit dieser Version ein neues GUI-System eingeführt wurde, dass sich einfacher in Verbindung mit der Oculus Rift nutzen lässt.

4.5 Einrichten von Unity zur Verwendung mit der Oculus Rift

4.5.1 Anlegen eines neuen Projekts

Unity organisiert die Entwicklung von Anwendungen in einzelnen Projekten. In einem Unity-Projekt werden neben den verschiedenen Anwendungsszenen (z. B. für eine Aufteilung in verschiedene Levels) auch alle 3D-Modelle inkl. der dafür verwendeten Materialien, Texturen und Shader sowie alle Interaktionsskripte verwaltet.

Ein neues Projekt wird über `File` → `New Project` angelegt. Die Auswahl des Speicherorts sowie die Benennung des Verzeichnisses können dabei frei erfolgen.

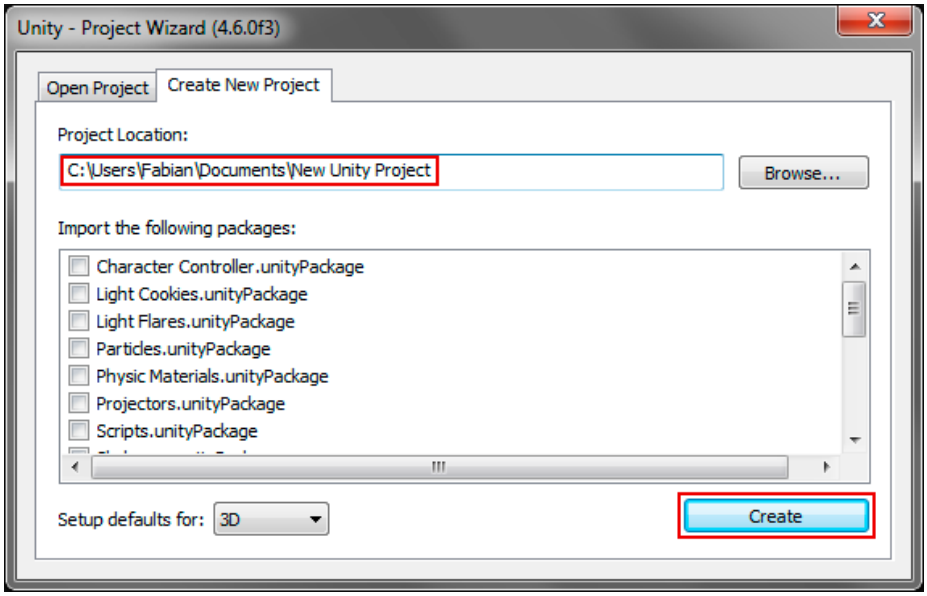

Abbildung 9: Anlegen eines neuen Unity-Projekts

Im Anschluss erstellt Unity ein neues Projekt und öffnet eine leere Szene.

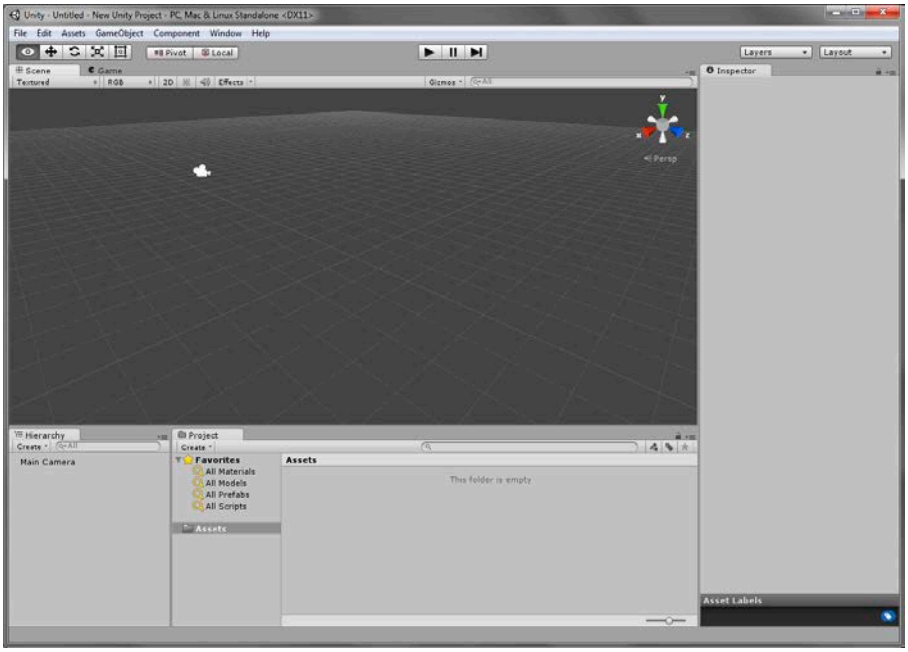

Abbildung 10: Ansicht von Unity nach Öffnen eines neuen Projekts

Jetzt kann die im Vorfeld heruntergeladene Unity 4 Integration entpackt und die darin enthaltene „OculusUnityIntegrationTuscanyDemo.unitypackage“ über `Assets` → `Import Package` → `Custom Package` importiert werden.

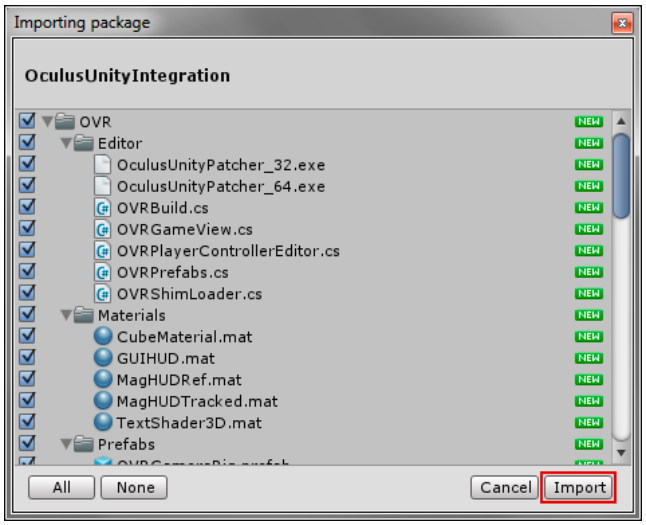

Abbildung 11: Bestätigung für den Import der Tuscany-Demo

Anschließend richtet Unity das importierte Demo-Projekt automatisch ein und bereitet alle Projektdateien für die Verwendung im Editor vor. Dies kann je nach Auswahl des importierten Packages einige Minuten in Anspruch nehmen.

Nach erfolgreichem Abschluss der Übernahme öffnet man im Unterordner `Tuscany/Scenes` die `VRDemo_Tuscany`-Szene. Die daraufhin erscheinende Frage, ob man die zuvor beim Erstellen des Projekts angelegte (und damit leere) Szene speichern möchte, kann man mit Nein beantworten.

Die Toskana-Demo ist nun installiert und geöffnet. Die Szene kann wie jedes andere Unity-Projekt beliebig verändert werden.

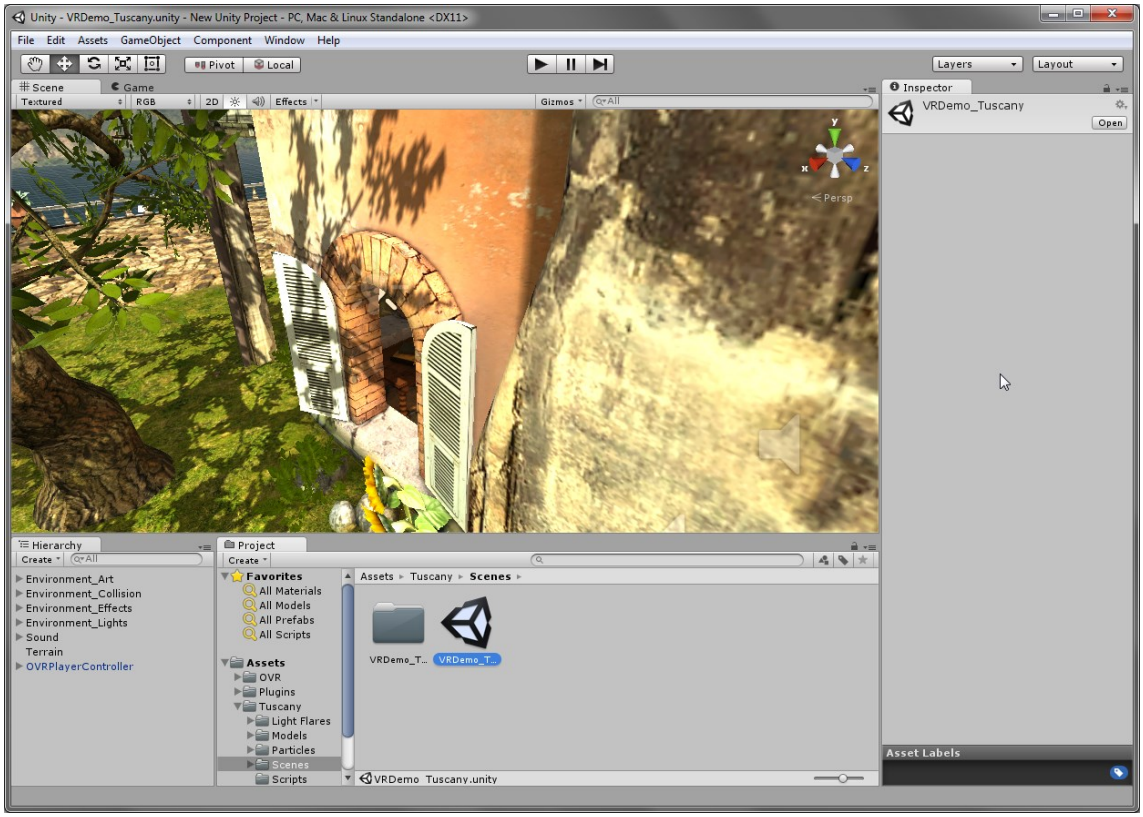

Abbildung 12: Tuscany-Szene zur Bearbeitung geöffnet

4.6 Exportieren in eine ausführbare Anwendung

An Ende der Entwicklung steht der Export in eine ausführbare Anwendung.

Dazu definiert man unter `File` → `Build Settings` die aktuelle Szene durch Klick auf die Schaltfläche „Add Current“ als Teil der zu erstellenden Anwendung. Mit der Schaltfläche „Build“ wird nach Abfrage eines Ablageverzeichnisses die Anwendung kompiliert.

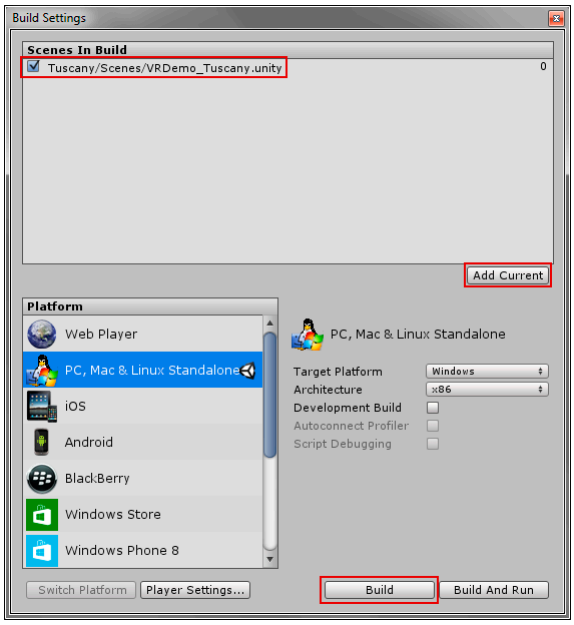

Abbildung 13: Kompilieren der Anwendung in Unity

Nach Abschluss öffnet Unity das beim Kompilieren angegebene Verzeichnis. Der „Data“-Ordner enthält alle Anwendungsdaten in kompilierter Form. Die große Datei darunter ist die eigentliche Anwendung, die (z. B. zum Testen) auch ohne die Oculus Rift gestartet werden kann. Die mit „DirectToRift“ gekennzeichnete Datei startet die Anwendung hingegen direkt auf der DK2.

VRDemo_Tuscany_Data	07.02.2015 18:51	Dateiordner	
VRDemo_Tuscany	18.11.2014 17:48	Anwendung	11.324 KB
VRDemo_Tuscany_DirectToRift	24.10.2014 08:13	Anwendung	161 KB

Abbildung 14: Die beim Kompilieren der Anwendung erstellten Dateien im Überblick

Abbildung 15: Screenshot der kompilierten Tuscany-Demo

5. Umsetzung

5.1 Aufbereitung der 3D-Daten

5.1.1 Flugzeug

Zentrales Element der Anwendung ist das Flugzeugmodell, das vom Benutzer geprüft werden soll.

Das Modell wurde ursprünglich für einen Flugsimulator erstellt. Es ist daher sehr detailliert und enthält viele Modellteile (z. B. interne Mechaniken), die für die Anwendung nicht erforderlich sind. Aus diesem Grund wird das Modell vorab in Autodesk 3ds Max auf die wirklich wichtigen Bestandteile reduziert.

Abbildung 16: Ansicht des Flugzeugmodells

Der Vorteil dieser Vorgehensweise ist, dass das Autodesk-Dateiformat `.max` von Unity problemlos unterstützt wird. Das 3D-Modell inkl. seiner Materialien, Texturen und auch evtl. eingebundener Animationen wird nach der Speicherung im Asset-Unterordner automatisch von Unity erkannt und eingelesen. Dies spart den sonst üblichen Umweg über Austauschformate, wie z. B. FBX. Spätere Änderungen am Modell werden ebenfalls automatisch erkannt und übernommen. Ein erneutes Importieren des Modells in Unity ist nicht erforderlich.

5.1.2 Hangar-Szene

Der Hangar wurde nachträglich für das Flugzeugmodell erstellt, um die Atmosphäre – trotz seiner einfacheren Aufmachung - realistischer erscheinen zu lassen und der 3D-Szene auch optisch eine abgesteckte Grenze zu verleihen.

Abbildung 17: Innenansicht des Hangars, später Schauplatz der Anwendung

5.1.3 RBF-Tags

„Remove before flight“-Tags (RBF-Tags) kennzeichnen empfindliche Teile eines Flugzeugs, die vor dem Start auf ihre Unversehrtheit geprüft werden müssen. Zu diesem Zweck sind sie meist durch eine Abdeckung geschützt. Da der Pilot diese zum Überprüfen abnehmen und einsammeln muss, wird gewährleistet, dass er die entsprechenden Bauteile auch wirklich aus der Nähe untersucht.

Für die Anwendung orientiert sich die Ausgestaltung der Tags an der Form eines Bandes. Da für den Benutzer ein schmales 3D-Modell mit der DK2 nur schwer von anderen Objekten unterscheidbar ist, wurde dieses Modell optisch angepasst, so dass es von allen Seiten gleichermaßen erkannt wird.

Abbildung 18: RBF-Tag mit größerem Grundriss für bessere Sichtbarkeit in der 3D-Szene

5.1.4 Werkzeuge

Die Modelle der Werkzeuge sind einfach gehalten, da diese nur exemplarisch in der Anwendung verwendet werden. Die Art der Modelle orientiert sich an bekannten Werkzeugen, die den meisten Anwendern vertraut sind und die aufgrund ihrer markenten Form auffallen.

Abbildung 19: Die verschiedenen Werkzeugtypen

5.1.5 Trümmerteile

Die Trümmerteile wurden ebenfalls in 3ds Max erstellt. Da diese in der Anwendung nur exemplarisch verwendet werden, handelt es sich um einfach aufgebaute Modelle.

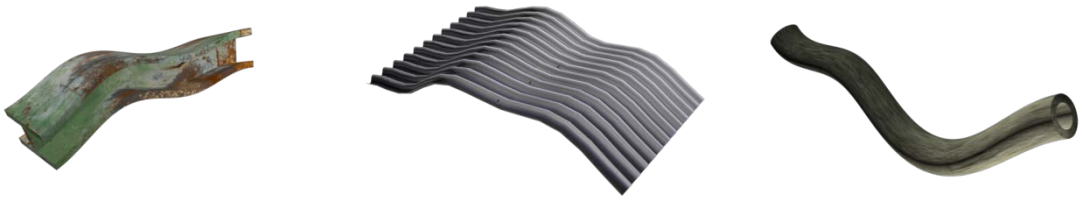

Abbildung 20: Die verschiedenen Trümmerteile

Die Modelle der Trümmerteile werden später an mehreren Stellen in der 3D-Szene wiederverwendet. Da der Schwerpunkt der Anwendung auf den Möglichkeiten der Oculus Rift liegt, wurde bei den Modellen gezielt auf ein Variantenmanagement (z. B. mittels Textur- oder Formänderung) verzichtet.

5.2 GUI

Unity verfügt über ein integriertes System, mit dem sich für jede Anwendung ein Graphical User Interface (GUI) erstellen lassen. Teil eines GUI können z. B. Text- oder Symboleinblendungen sein. Einblendungen im GUI werden in der Anwendung verwendet, um den Benutzer auf eine Interaktionsmöglichkeit hinzuweisen.

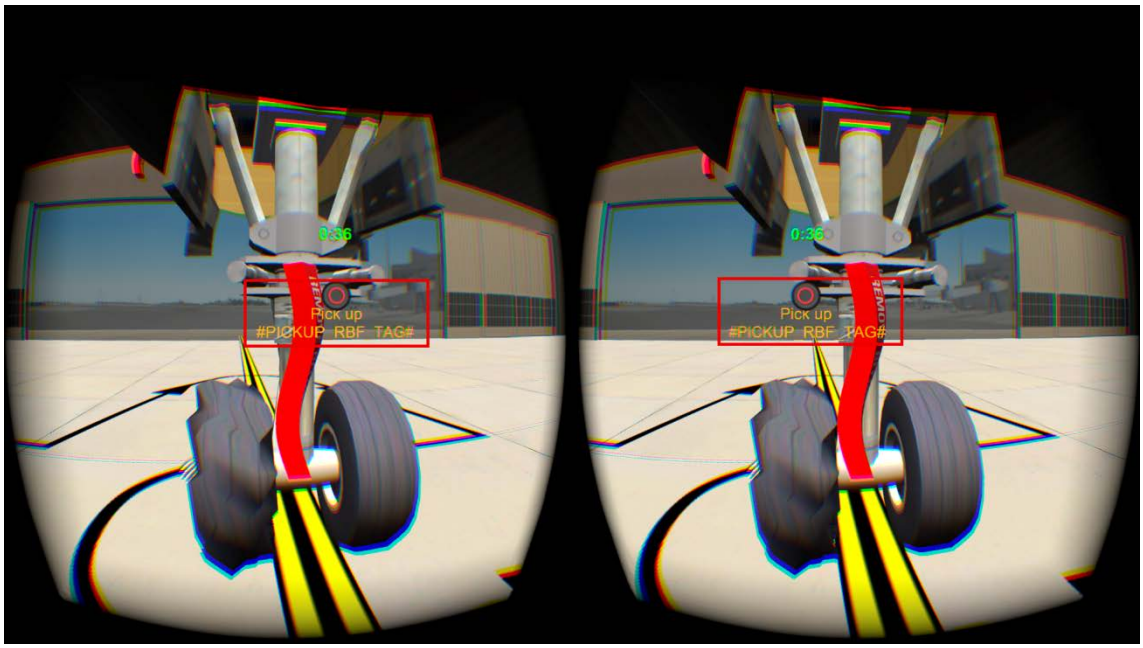

Abbildung 21: Einblendungen im GUI für beide Augen

Das GUI in der Anwendung ist relativ zur Blickrichtung des Benutzers ausgerichtet. Es wurde in Unity unmittelbar vor den beiden Kamerareferenzen des OVRPlayerController platziert, so dass die ausgegebenen Anzeigen auch für beide Augen dargestellt werden.

5.3 Programmierung

5.3.1 Navigation in der Anwendung

Grundlegende Steuerungsmöglichkeiten innerhalb einer 3D-Szene bringt das *Oculus Integration Package* bereits mit. Dies schließt neben der Bewegungskontrolle mit den Pfeiltasten bzw. der Kombination W-A-S-D auch die Steuerung der Ansicht mit der Maus ein (jedoch nur in horizontaler Richtung).

Die Steuerung mittels Gamepad ist zwar schon vorbereitet, funktioniert von Haus aus aber nicht mit allen Gamepads. Original-Gamepads für Microsoft's Xbox oder Sony's PlayStation 3/4 werden unterstützt – kompatible Gamepads von Drittherstellern erfordern ggf. Anpassungen, da sich die interne Tastenbelegung (z. B. für die Taste mit der Bezeichnung „joystick button 1“) unterscheiden kann.

Die später in der Anwendung verwendeten Tastenbelegungen können in Unity mit dem sog. Input Manager definiert und nach freien Wünschen zugeordnet werden.

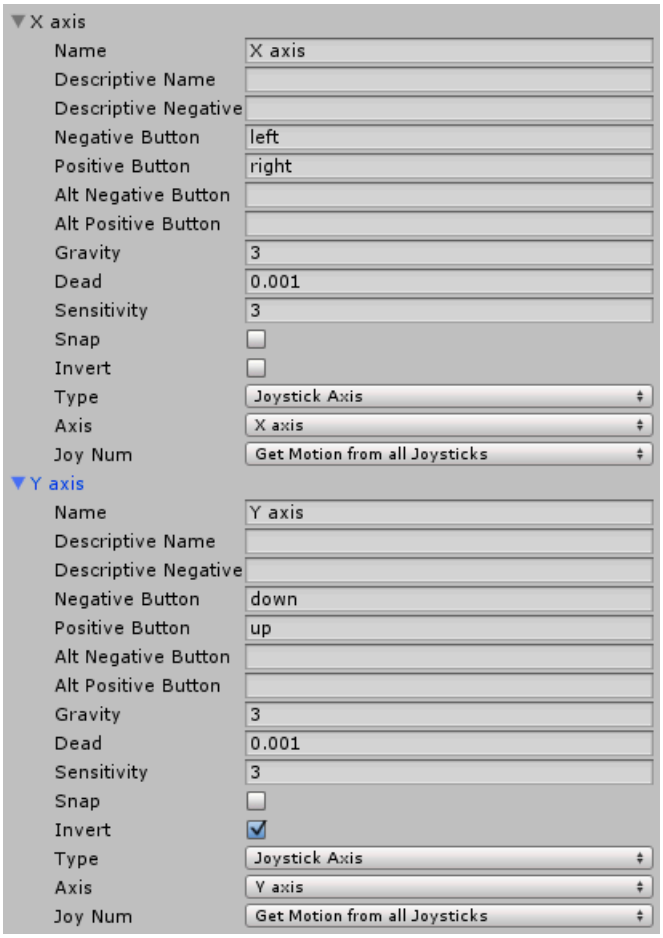

Abbildung 22: Beispielhafte Definitionen im Input Manager für die X- und Y-Achse des Gamepad-Controllers

5.4 Interaktivität per Skript

Wenn es um die Erstellung interaktiver Anwendungen in Unity geht, spielen Skripte eine zentrale Rolle. Unity bietet einfache Möglichkeiten um Interaktivität in eine 3D-Szene zu bringen. Mit Unity können Skripte in Unity, C# sowie Boo[1] (einem Derivat von Python) geschrieben werden. In allen Skriptsprachen können die gleichen Funktionalitäten von Unity verwendet werden. Abbildung 23 zeigt einen Vergleich der drei Skriptsprachen anhand eines exemplarischen Beispiels.

UnityScript (JavaScript for Unity)

```
function Update () {
  var fwd = transform.TransformDirection (Vector3.forward);
  if (Physics.Raycast (transform.position, fwd, 10)) {
    print ("There is something in front of the object!");
  }
}
```

C#

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        Vector3 fwd = transform.TransformDirection(Vector3.forward);
        if (Physics.Raycast(transform.position, fwd, 10))
            print("There is something in front of the object!");

    }
}
```

Boo (Python)

```
import UnityEngine
import System.Collections

public class ExampleClass(MonoBehaviour):

  def Update() as void:
    fwd as Vector3 = transform.TransformDirection(Vector3.forward)
    if Physics.Raycast(transform.position, fwd, 10):
      print('There is something in front of the object!')
```

Abbildung 23: Exemplarischer Vergleich der selben Prozedur in den verschiedenen Skriptsprachen, die in Unity verwendet werden können[2]

[1] Mit der Veröffentlichung von Unity 5 wird Boo als offizielle Skriptsprache aufgrund der geringen Verwendung in Unity-Projekten voraussichtlich nicht mehr weiter unterstützt (siehe dazu http://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/)

[2] Beispiel zur Funktion `Physics.Raycast` aus der offiziellen Script Reference der Unity Documentation, http://docs.unity3d.com/ScriptReference/ Physics.Raycast.html

Die Skripte für diese Anwendung sind alle in C# geschrieben. Die kompletten Quellcodes der nachfolgend beschriebenen Skripte befinden sich im Unterordner `Assets\Scripts` des Projektverzeichnisses.

5.4.1 Interaktion mit Objekten allgemein

Ausgangsbasis für alle Interaktionsmöglichkeiten des Benutzers ist das zentrale `OBJECT_INTERACTION`-Skript.

Um mit Objekten in der virtuellen Umgebung interagieren zu können, müssen diese erst einmal greifbar gemacht werden. Zu diesem Zweck verwendet das Skript einen sog. Raycast. Dabei handelt es sich um einen imaginären Strahl, der in die jeweils aktuelle Blickrichtung geschickt wird. Sobald dieser auf ein Objekt trifft, gibt es eine entsprechende Rückmeldung.

Das Skript prüft bei jeder Frame-Aktualisierung, ob sich innerhalb eines bestimmten Abstands Objekte in aktueller Blickrichtung befinden. Wird ein Objekt gefunden, liest das Skript die Kennzeichnung (Tag) aus. Anhand des vergebenen Tags weiß das Skript, was es machen muss. Objekte ohne Tag werden generell ignoriert, bei Objekten mit Tag wird der Benutzer über die Interaktionsmöglichkeit informiert. Dazu wird im GUI der Anwendung eine entsprechende Meldung eingeblendet.

```
void Update () {
  ...
  Vector3 fwd = ForwardRef.transform.TransformDirection
(Vector3.forward);
  fwd = fwd + new Vector3 (0f, 0.25f, 0f);

  if (Physics.Raycast (transform.position, fwd, out hitCheck)) {

    if(hitCheck.distance <= Distance) {
      //Object reference
      obj = hitCheck.collider.gameObject;

      //Display text
      switch(obj.tag) {
      //Pickup items
      case "PICKUP_RBF_TAG":
      case "PICKUP_TOOLS":
      case "PICKUP_DEBRIS":
        UITextCenter.text = "Pick up\n#" + obj.tag + "#";
        IconUse.SetActive (true);
        break;
      case "PICKUP_BROKEN_PARTS":
        UITextCenter.text = "Fix\n#" + obj.tag + "#";
        IconUse.SetActive (true);
        break;
      // Handles, Buttons, etc.
      case "DOOR":
        Animation AnimationComponent = obj.GetComponent<Animation>();
        if (AnimationComponent == null) {
          //UITextCenter.text = "DOOR (no animation)";
        } else {
          //UITextCenter.text = "DOOR (animated, " +
AnimationComponent.GetClipCount() + ")";
          UITextCenter.text = "Open door";
          IconUse.SetActive (true);
        }
        break;
      default:
        //UITextCenter.text = "";
        break;
      }
      ...
    }
  }
  ...
}
```

Abbildung 24: Ausschnitt des Interaktions-Skriptes für Einblendungen im GUI

5.4.2 Pickup-Objekte

Die im Vorfeld erstellten und einmalig in der 3D-Szene plazierten Pickup-Objekte dienen als Referenz für weitere solche Objekte, die später beim Starten der Anwendung dynamisch platziert werden. Zu diesem Zweck werden die Positionen in einer externen XML-Datei gespeichert. Diese enthält neben dem Typ zur Identifizierung des zu platzierenden Objekts auch die Position und ein

Kommentarfeld für den Entwickler zur genaueren Beschreibung des (eher kryptisch anmutenden) Eintrags.

Als Konvention wurden die Objekttypen wie folgt definiert:

Objekttyp	XML-Definition	3D-Referenzobjekt
RBF-Tag	`rbf_tag`	RBF_TAG
Werkzeug	`tool`	TOOL_HAMMER TOOL_SCREWDRIVER TOOL_PAINT
Trümmerteile	`debris`	DEBRIS_PIPE DEBRIS_PLATE DEBRIS_TRUSS

Tabelle 2: Typdefinitionen der Pickup-Objekte

Entsprechend dieser Definition werden die einzelnen Einträge in der XML-Datei kategorisiert.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <tags>
3    <!-- RBF TAGS -->
4    <tag type="rbf_tag" x="0.2" y="-1.7" z="-15.1" comment="RBF tag 1" />
5    <tag type="rbf_tag" x="4.9" y="-2.6" z="5.4" comment="RBF tag 2" />
6
7    <!-- TOOLS -->
8    <tag type="tool" x="-3.1" y="-5.2" z="-6.3" comment="tool 1" />
9    <tag type="tool" x="15.4" y="2.8" z="3.9" comment="tool 2" />
10
11   <!-- DEBRIS -->
12   <tag type="debris" x="-3.4" y="-5" z="-31" comment="debris 1" />
13   <tag type="debris" x="0.1" y="2.1" z="16" comment="debris 2" />
14 </tags>
```

Abbildung 25: Beispieldefinitionen in der PICKUP_OBJECTS.xml

Das Einlesen der XML-Datei erfolgt im INIT-Skript beim Laden der Anwendung. Dabei werden die einzelnen Einträge durchlaufen, der Objekttyp aus dem Attribut `type` sowie die Positionsdaten der Attribute `x`, `y`, `z` ausgelesen. Entsprechend der angegebenen Typdefinition wird das jeweilige Referenzmodell in der 3D-Szene ausgewählt. Für die Objekttypen `tool` und `debris` sind im Inspector jeweils mehrere Modelle als Array angegebenen (siehe Tabelle 2 und Abbildung 26). Aus diesen Elementen wird per Zufall eines als Referenz ausgewählt.

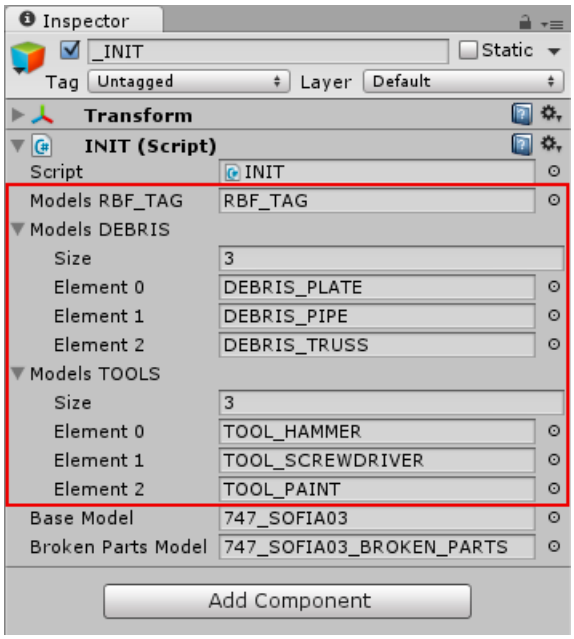

Abbildung 26: Inspector-Panel mit Angaben des INIT-Skripts für die Pickup-Objekte

Danach wird das referenzierte Objekt instanziiert, d. h. das 3D-Referenzobjekt aus der 3D-Szene geklont und gleich per Zufall gedreht. Da die Größe beim Klonen nicht direkt mit angegeben werden kann, muss diese Angabe im Anschluss nachgeholt werden – aus Sicherheitsgründen allerdings nur, wenn auch tatsächlich ein neues Objekt erstellt wurde.

```
1  void InitPickupObjects() {
2    //XML öffnen
3    XmlReader reader = XmlReader.Create("PICKUP_OBJECTS.xml");
4
5    //XML auslesen
6    while(reader.Read()) {
7      //Typ erkennen
8      if(reader.IsStartElement("tag")) {
9        //Typ auslesen
10       type = reader.GetAttribute("type");
11
12       //Position auslesen
13       position = new Vector3();
14       position.x = float.Parse(reader.GetAttribute("x"));
15       position.y = float.Parse(reader.GetAttribute("y"));
16       position.z = float.Parse(reader.GetAttribute("z"));
17
18       //Objekt auswählen
19       switch(type) {
20       case "rbf_tag":
21         GameObjectRef = ModelsRBF_TAG;
22         NUM_RBF_TAGS_MAX++;
23         break;
24       case "tool":
25         GameObjectRef =
             ModelsTOOLS[Random.Range(0,ModelsTOOLS.Length-1)];
26         NUM_TOOLS_MAX++;
```

```
27          break;
28        case "debris":
29          GameObjectRef =
              ModelsDEBRIS[Random.Range(0,ModelsDEBRIS.Length-1)];
30          NUM_DEBRIS_MAX++;
31          break;
32        default:
33          clone = null;
34          break;
35        }
36
37        //neue Instanz des jeweiligen Objekts erstellen
38        clone = Instantiate(GameObjectRef, position,
            Quaternion.Euler(0, Random.Range(0.0f, 359.0f), 0))
            as GameObject;
39
40        //Wenn Objekt erfolgreich erstellt wurde -> Objekt skalieren
41        if(clone != null) {
42          clone.transform.localScale = new Vector3(1, 1, 1);
43        }
44      }
45    }
46  }
```

Abbildung 27: Laden der externen XML-Datei mit direkter Erstellung der jeweiligen Pickup-Objekte in der 3D-Szene

5.4.3 Beschädigte Bauteile

Analog zu den allgemeinen Pickup-Objekten, gibt es auch für die Objektgruppe der beschädigten Bauteile („Broken Parts“) eine interne Definition. Ursprünglich war die Definition dieser Objekte ebenfalls als Teil der externen XML-Datei geplant. Da die beschädigten Bauteile jedoch nur einmal zu Beginn platziert werden müssen, wurde kein Schwerpunkt auf eine Verformung der Meshes zur Laufzeit gelegt.

Objekttyp	(XML-)Definition	3D-Referenzobjekt
Beschädigung	`broken_part`	Alle Teilmodelle des separaten 747_SOFIA03_BROKEN_PARTS-Modells

Tabelle 3: Definition für die "Broken Parts"

Aus diesem Grund kommt in der Anwendung eine abgewandelte Kopie des Flugzeugmodells zum Einsatz. Das Modell wurde in 3ds Max soweit reduziert, dass nur einzelne Teilmodelle enthalten sind. Anschließend wurden diese so bearbeitet wurden, dass sie einen verbogenen und beschädigten Eindruck vermitteln.

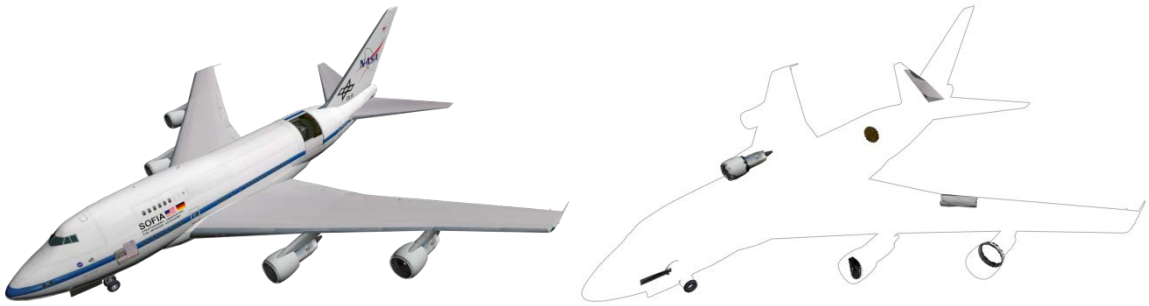

Abbildung 28: Gegenüberstellung des Basismodells und des Modells mit den beschädigten Bauteilen (Umriss zur Veranschaulichung)

Anders als bei den zuvor beschriebenen Pickup-Objekten wird die Platzierung des modifizierten Flugzeugmodells direkt in Unity vorgenommen, indem es an dieselbe Stelle wie das Originalmodell (Basismodell) gesetzt wird.

Da auf diese Weise jedoch beide Modelle überlagert angezeigt werden, müssen beim Laden der Anwendung all diejenigen Meshes des Basismodells ausgeblendet werden, die denen im „beschädigten“ Modell („Broken Parts“-Modell) entsprechen. Dies übernimmt die Funktion `InitBrokenParts` im INIT-Skript. Das Skript erhält im Inspector jeweils eine Referenz auf das Basismodell und das veränderte „Broken Parts“-Modell.

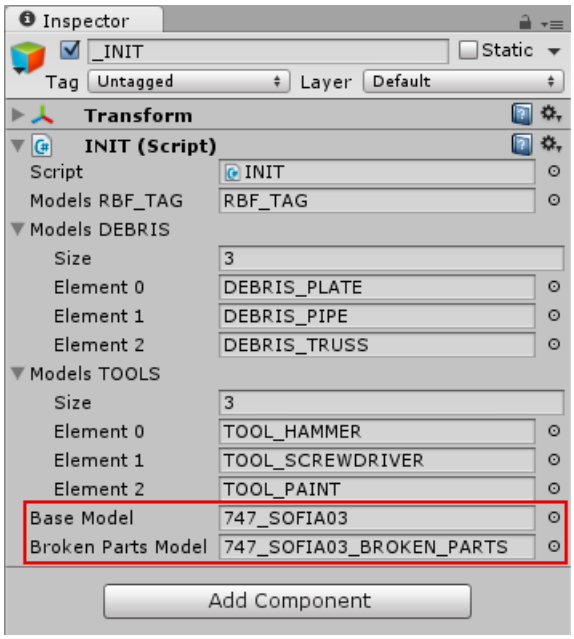

Abbildung 29: Inspector-Panel mit Angaben des INIT-Skripts für die "Broken Parts"

Zu Beginn werden alle Teilmodelle des „Broken Parts“-Models mit dem Tag `PICKUP_BROKEN_PARTS` versehen. Als zweiter Schritt wird der Inhalt des

Basismodell mit dem „Broken Parts“-Models verglichen. Da während der Bearbeitung in 3ds Max keine Änderung an den internen Modellbezeichnungen im „Broken Parts“-Model vorgenommen wurden, kommt es beim Vergleich zweifelsfrei zu Übereinstimmung der jeweiligen Teilmodellbezeichnungen. Im diesem Fall wird das entsprechende Teilmodell im Basismodell aufgeblendet, so dass an dessen Stelle nur noch das beschädigte Bauteil zu sehen ist.

```
1  void InitBrokenParts() {
2    //Liste der Submodels holen
3    BaseModelChildren = BaseModel.GetComponentsInChildren<Transform>();
4    BrokenPartsModelChildren = ↵
      BrokenPartsModel.GetComponentsInChildren<Transform>();
5
6    //Tags verteilen
7    foreach (Transform BrokenPartsModelSingleChild in
       BrokenPartsModelChildren) {
8      BrokenPartsModelSingleChild.gameObject.tag = "PICKUP_BROKEN_PARTS";
9    }
10
11   //Submodels des Referenzmodells im Basismodell ausblenden und ↵
       jeweilige Änderung im Log ausgeben
12   foreach (Transform BaseModelSingleChild in BaseModelChildren) {
13     //Für jedes Submodel des Basismodells prüfen, ob es auch im ↵
         Referenzmodell vorhanden ist
14     foreach(Transform BrokenPartsModelSingleChild in ↵
         BrokenPartsModelChildren) {
15       if(BrokenPartsModelSingleChild.name == BaseModelSingleChild.name) {
16         //Referenzmesh im Basismodell ausblenden
17         BaseModelSingleChild.gameObject.SetActive(false);
18         NUM_BROKEN_PARTS_MAX++;
19       }
20     }
21   }
22 }
```

Abbildung 30: Initialisierung der beschädigten Bauteile

Die spätere Interaktion mit den beschädigten Bauteilen übernimmt das zentrale OBJECT_INTERACTION-Skript. Über den während der Initialisation vergebenen Tag `PICKUP_BROKEN_PARTS` weiß das Skript, was es mit diesen Objekten machen muss.

Konkret durchsucht das Skript das Basismodell nach dem Gegenstück zum jeweiligen Teilmodell, mit dem der Benutzer gerade interagieren möchte. Dieser Test ist nicht zwingend erforderlich, bietet aber eine Möglichkeit, um sicherzustellen, dass das Teilmodell auch richtig übernommen wurde. Kann das Bauteil im Basismodell gefunden werden, wird es wieder eingeblendet. Das

beschädigte Bauteil wird anschließend aus der Szene gelöscht, da es im Anschluss nicht mehr benötigt wird.

```
void Update () {

  if (Physics.Raycast (transform.position, fwd, out hitCheck)) {

    if(hitCheck.distance <= Distance) {
      //Object reference
      obj = hitCheck.collider.gameObject;

      //Aktion mit einem Object
      if(Input.GetKeyDown(KeyCode.E) || Input.GetButton("Use")) {
        //Auswahl der Aktion je nach Objekttyp
        switch(obj.gameObject.tag) {
          ...
          case "PICKUP_BROKEN_PARTS":
            //Submodel des Originalmodells wieder einblenden
            foreach(Transform child in BaseModel_Children) {
              if(child.name == obj.name) {
                child.gameObject.SetActive(true);
                Destroy(obj);
                NUM_BROKEN_PARTS_FOUND++;
                break;
              }
            }
          break;

        }
      }
    }
  }

}
```

Abbildung 31: Ausschnitt des OBJECT_INTERACTION-Skripts mit Austausch des beschädigten Bauteils gegen das Gegenstück im Basismodell

5.4.4 Türen und Klappen

In der Szene befinden verschiedene Objekte, die in ihrer Form Türen und Klappen darstellen. Während der Laufzeit kann der Benutzer mit den Türen in Interaktion treten um sie zu öffnen oder zu schließen. Das Skript soll der Einfachheit halber zwischen dem geschlossenen und dem offenen Zustand umschalten und für den Übergang die entsprechenden Animationen abspielen. Der Animationsbedarf einer Tür beschränkt sich im Allgemeinen auf das Öffnen und Schließen. Die damit verbundenen Transformationsänderungen betreffen auch meist nur eine der Rotationsachsen.

Für die Anwendung werden die entsprechenden Animationen direkt in Unity erstellt. Eine vorherige Definition in 3ds Max wäre ebenfalls möglich, da Unity

diese beim Import des 3D-Modells ebenfalls übernehmen und ansprechbar machen kann.

Animationen werden in Unity im Animation-Fenster erstellt. Nach Auswahl eines 3D-Modells kann mit der Schaltfläche „Add Curve“ eine neue Animation erstellt werden.

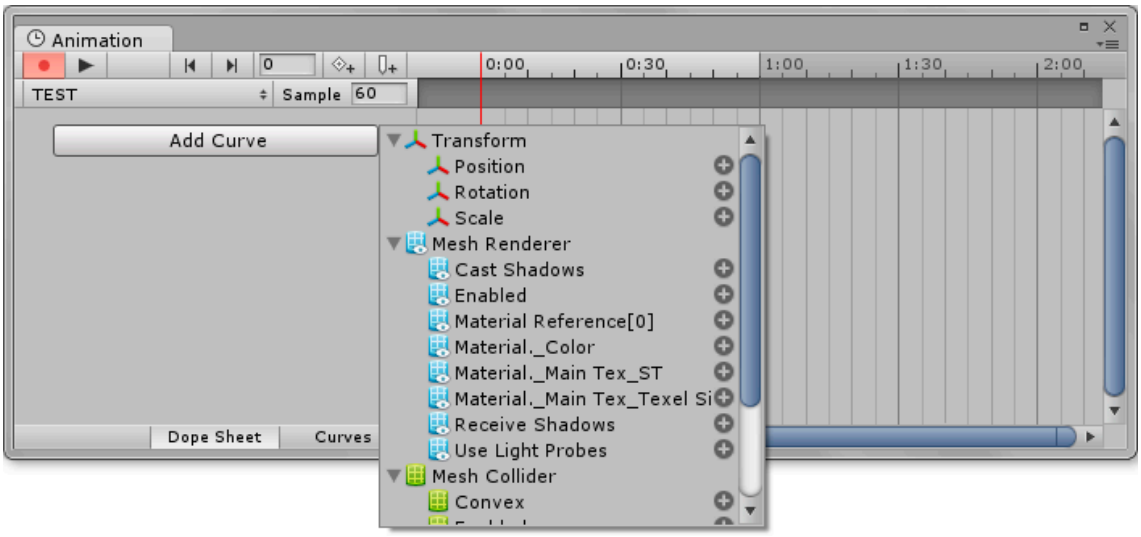

Abbildung 32: Auswahl einer Objekteigenschaft für die spätere Animation

Unity speichert Animationen wie Skripte in einzelnen Dateien und fordert daher zur Vergabe eines Namens sowie eines Speicherortes auf. Nach Auswahl einer zu animierenden Animation können anschließend in in der Zeitleiste des Reites „DopeSheet“ einzelne Keys definiert werden, an denen die gewählten Eigenschaften einen gewünschten Wert annehmen sollen.

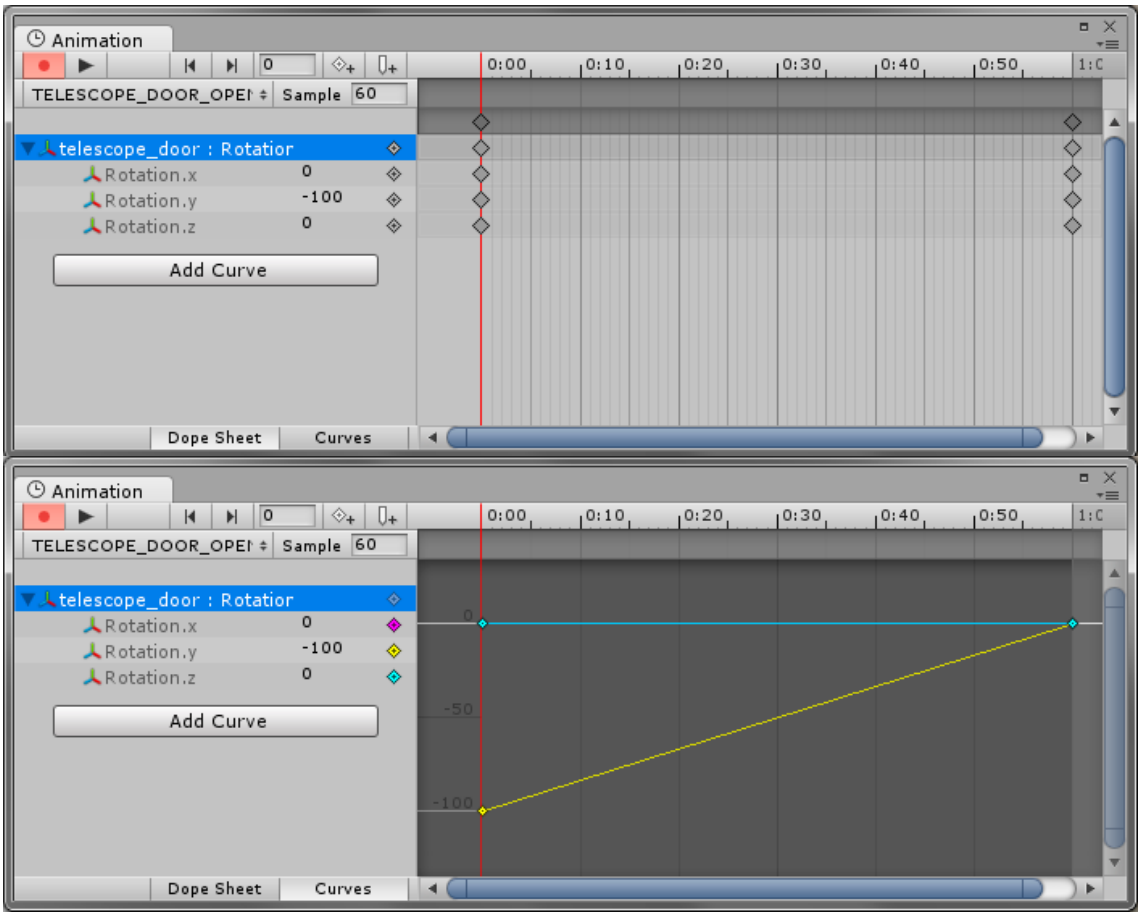

Abbildung 33: Animation-Tab mit Reiter „Dope Sheet“ (oben) und in der Kurvenansicht (unten) zur Darstellung und Bearbeitung der animierten Eigenschaften; hier: Rotation entlang der y-Achse

Um ein einzelnes 3D-Objekt in der Szene (z. B. die Tür des Startgebäudes) animieren zu können, wird es im Inspector mit einer Animations-Komponente versehen. Darin können beliebig viele Animationen für das Objekt hinterlegt werden.

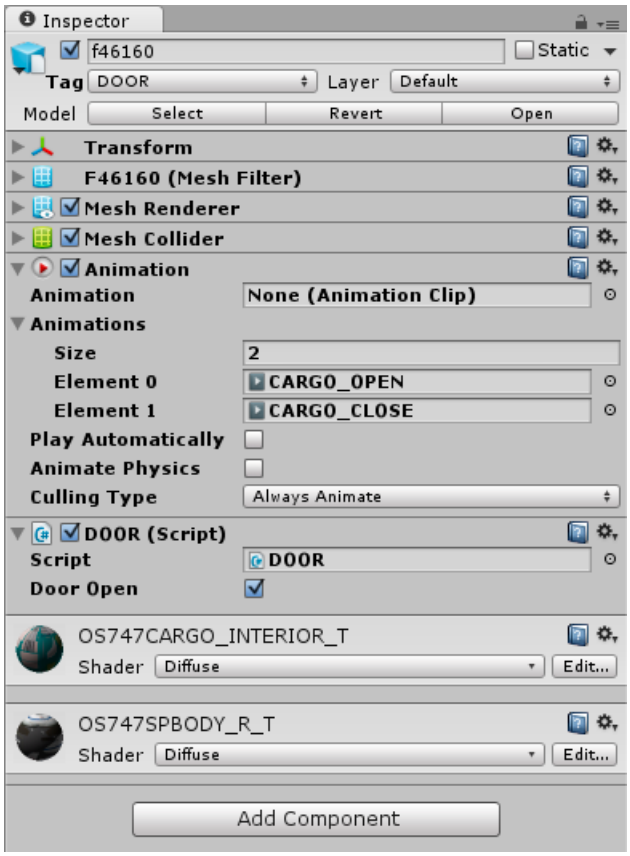

Abbildung 34: Inspector-Panel einer Tür mit Animations-Komponente und DOOR-Skript

Um eine einheitliche Funktionsweise in Zusammenspiel mit dem nachfolgend erstellten Tür-Skript zu erreichen, wird für diese Projekt eine Konvention getroffen: Jedes als Türe zu verwendende 3D-Objekt wird jeweils mit zwei Animationen ausgestattet. Dazu werden jeweils die zwei zuvor für das jeweilige Objekt erstellten Animationen in fester Reihenfolge zugewiesen. Element 0 ist dabei immer die Animation des Öffnens, Element 1 die des Schließens.

Animationen werden in Unity-Skripten mit dem Namen angesprochen, d. h. um eine Animation zu starten, muss der Name explizit bekannt sein. Da das Skript universell für mehrere 3D-Objekte eingesetzt werden soll, können diese Namen nicht einfach hart in den Quellcode geschrieben werden[3]. Stattdessen müssen bei Initialisierung des Skriptes zuerst einmal die in der Animations-Komponente zugewiesenen Animationen ausgelesen werden.

[3] Anmerkung des Autors: Soll ein solches Skript nur bei einer geringen Anzahl von Objekten angewendet werden, kann durchaus auch jeweils ein einzelnes Skript erstellt und die Animationsnamen fest in den jeweiligen Quellcode übernommen werden. Davon wird generell aber abgeraten, da nachträgliche Änderungen an den Skripten in kleinen Stückzahlen zwar noch recht einfach zu bewerkstelligen sind – bei vielen einzelnen Skripten mit gleicher Funktionsweise ist die Grenze der Übersichtlichkeit allerdings schnell erreicht.

Im Rahmen des Skripts zum Öffnen der Tür, können diese entsprechend den Variablen `AnimationOpen` und `AnimationClose` zugewiesen werden.

```
1  //Initialisierung
2  void Start () {
3    //Animationen holen
4    Animation AnimComponent = this.gameObject.GetComponent<Animation> ();
5
6    //Auf Vollständigkeit prüfen
7    if (Animation_Component.GetClipCount () == 2) {
8      string[] Animations = new string[2];
9
10     //Name der Animation holen
11     int i = 0;
12     foreach (AnimationState anim in AnimComponent.animation) {
13     Animations[i] = anim.name;
14       i++;
15     }
16
17     //Animationen zuweisen
18     AnimationOpen = Animations[0];
19     AnimationClose = Animations[1];
20   }
21 }
```

Abbildung 35: Initialisierung des DOOR-Skripts mit Zuweisung der Animationsnamen

Die Interaktion mit der Tür erfolgt im zentralen Interaktions-Skript, das die Tür anhand des im Editor vergebenen Tags `DOOR` erkennt und die Public-Funktion `UseDoor` des jeweiligen GameObjects aufruft.

```
void Update () {

  if (Physics.Raycast (transform.position, fwd, out hitCheck)) {

   if(hitCheck.distance <= Distance) {
    //Object reference
    obj = hitCheck.collider.gameObject;

    //Aktion mit einem Object
    if(Input.GetKeyDown(KeyCode.E) || Input.GetButton("Use")) {
     //Auswahl der Aktion je nach Objekttyp
     switch(obj.gameObject.tag) {

      case "DOOR":
       //DOOR-Skript im GameObject suchen
       DOOR door = hitCheck.transform.GetComponent<DOOR>();

       //wenn DOOR-Skript gefunden...
       if(door) {
        //Tür öffnen/schließen
        door.UseDoor();
       }
       break;

     }
    }
   }
  }

 }
}
```

Abbildung 36: Ausschnitt des OBJECT_INTERACTION-Skripts mit Aufruf der UseDoor-Funktion des DOOR-Skripts

In der aufgerufenen `UseDoor`-Funktion wird zuerst der aktuelle Zustand der Tür anhand des Wertes der Boolean `DoorOpen` geprüft. Bei Initialisierung der Anwendung erhält diese Variable automatisch, für alle mit dem DOOR-Skript versehenen 3D-Objekte, den Wert `false`. Wurde im Inspector die Checkbox DoorOpen aktiviert (siehe Abbildung 34), wird die Variable entsprechend auf `true` gesetzt. Je nach ermitteltem Zustand der Tür wird anschließend entweder die Animation zum Öffnen oder Schließen gestartet.

Bei Steuerung über die Tastatur ist diese Vorgehensweise ausreichend. Die `GetKeyDown`-Funktion prüft nur den speziellen Fall ab, dass eine Taste gedrückt wurde. Möchte man einen Gamepad-Controller verwenden, muss man zusätzlich den Umstand beachten, dass mechanische Schalter zum Prellen neigen. Durch die Vibrationen der Kontakte lösen sie bei beabsichtigter einmaliger Betätigung mehrfache Einzeleingaben am Computer aus. Da die

GetButton-Funktion bei jeder Aktualisierung auf diese falschen Eingaben reagiert, wird der davon abhängige Programmcode - in diesem Fall die UseDoor-Funktion - mehrmals hintereinander ausgeführt. Dies hat zur Folge, dass die nächste Animation gestartet wird, bevor die aktuelle überhaupt fertig ausgeführt werden konnte. Aus diesem Grund stellt die UseDoor-Funktion sicher, dass nicht bereits eine Animation gestartet wurde, bevor der Zustand der Tür verändert wird.

```
1  //Aufruf durch OBJECT_INTERACTION-Skript
2  public void UseDoor() {
3      //Zustand der Tür prüfen
4      if (DoorOpen) {
5          //Sicherstellen, dass die andere Animation nicht gerade läuft
   (Toggle-Effekt bei Gamepad-Controllern vorbeugen)
6          if(!animation.IsPlaying (AnimationOpen)) {
7              //Tür schließen
8              animation.Play (AnimationClose);
9              DoorOpen = false;
10         }
11     } else {
12         //s.o.
13         if(!animation.IsPlaying (AnimationClose)) {
14             //Tür öffnen
15             animation.Play (AnimationOpen);
16             DoorOpen = true;
17         }
18     }
19 }
```

Abbildung 37: UseDoor-Funktion des DOOR-Skripts mit Entprellung bei Verwendung für Gamepad-Controllern

5.4.5 Taschenlampe

Für die Funktionalität der Taschenlampe ist das dafür zuständige FLASHLIGHT-Skript ähnlich aufgebaut wie das zuvor beschriebene DOOR-Skript. Beim Starten der Anwendung werden die im Inspector zugewiesenen Animationen ausgelesen und ihre Bezeichnung für spätere Verwendung in Variablen gespeichert.

```
void Start () {
  //Animationen auslesen
  Animation Animation_Component =
    this.gameObject.GetComponent<Animation> ();
  if (Animation_Component.GetClipCount () == 2) {
    string[] Animations;
    Animations = new string[2];

    int i = 0;
    foreach (AnimationState anim in Animation_Component.animation) {
      Animations[i] = anim.name;
      i++;
    }

    AnimationOn = Animations[0];
    AnimationOff = Animations[1];
  }

  //Taschenlampe standardmäßig aus
  FL.enabled = false;
  animation.Play (AnimationOff);
}
```

Abbildung 38: Ausschnitt des FLASHLIGHT-Skript beim Start der Anwendung (vgl. dazu auch Abbildung 37)

Bei Interaktion des Benutzers wird die Taschenlampe ein- bzw. ausgeschaltet. Beim Einschalten wird die zugeordnete Lichtquelle aktiviert und das 3D-Modell der Taschenlampe entsprechend so animiert, dass sie im Sichtfeld des Benutzers erscheint.

```
void Update () {
  //Taschenlampe an-/ausschalten
  if(Input.GetKeyDown(KeyCode.F) || Input.GetButtonDown("R1")) {
    if (FL.enabled) {
      if(!animation.IsPlaying (AnimationOn)) {
        //Taschenlampe ausschalten
        animation.Play (AnimationOff);
        FL.enabled = false;
      }
    } else {
      if(!animation.IsPlaying (AnimationOff)) {
        //Taschenlampe einschalten
        animation.Play (AnimationOn);
        FL.enabled = true;
      }
    }
  }
}
```

Abbildung 39: Skript für die Funktionalität der Taschenlampe

Das FLASHLIGHT-Skript ist ein gutes Beispiel für die einfache Möglichkeit, Objekte unabhängig voneinander mit Funktionalitäten auszustatten.

6. Fazit

Die Möglichkeiten der Oculus Rift für Anwendungen im Bereich Gaming sind unumstritten. Für Anwendungen mit ernstem Hintergrund, besonders im Bereich Schulung, ist sie aber durchaus genauso interessant.

Bemerkenswert ist der hohe Grad der Immersion, der mit relativ einfachen Mitteln erzeugt werden kann. Die Oculus Rift als (quasi) reines „Anzeigegerät“ bietet allein schon aufgrund der speziell auf die Augen abgestimmten Darstellung einen hohen Realitätsbezug. Die zusätzlichen Steuerungsmöglichkeiten verstärken das Gefühl, sich in der virtuellen Umgebung zu befinden.

Die Entwicklung einer Oculus Rift-Anwendung in Unity bietet einen guten Ausgangspunkt für die erste Auseinandersetzung mit der Technik. Unity ist als Autorensystem besonders für solche Personen interessant, die sich nicht ausgiebig mit 3D-Software oder Anwendungsprogrammierung beschäftigen wollen oder aufgrund ihrer Tätigkeit nicht die Zeit dafür haben.

Anhang

DVD

Auf der beigelegten DVD befinden sich die Quelldaten sowie eine ausführbare Version der Anwendung.

Für das Öffnen der Quelldateien wird Unity 4.6 benötigt. Eine Testversion sowie die für die Oculus Rift notwendigen Softwarepakete sind im Ordner Programme zu finden.