
Prüfungsstudienarbeit

Projekt:
Interaktiver Schaukasten



Bachelor Media Engineering

5. Semester - WS 2019/20

6. Semester - SS 2020

René Sammeth

Matrikel-Nr.: 3088843

Selbstständigkeitserklärung

René Sammeth

Ich bestätige, dass ich die Prüfungsstudienarbeit mit dem Titel:

Out A Time (Interaktiver Schaukasten)

selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum:

Unterschrift:

Ich bestätige, dass ich der Technischen Hochschule Nürnberg für die in der Prüfungsstudienarbeit angefertigten Unterlagen, Dokumente, Fotografien und Filme sowie für die dabei entwickelte Soft- oder Hardware das uneingeschränkte nicht-ausschließliche Nutzungsrecht einräume:

Datum:

Unterschrift:

Abstrakt

Im folgenden Bericht werden die Projektabläufe und Resultate der Projektarbeit aus dem Wintersemester 19/20 und Sommersemester 2020 des Bachelor Studiengangs Media Engineering wiedergegeben.

Das Projekt behandelt die fehlende Weiterentwicklung von Schaukästen. Sie dienen dem Zweck der Vermittlung von Informationen oder auch zum darbieten von Produkten. Dies geschieht trotz dem Zeitalter der Digitalisierung meist noch nicht dynamisch und ohne eine richtige Interaktion. Zur Problemlösung entsteht hierbei eine interaktiver Schaukasten, welcher mit Hilfe von Gestensteuerung, Bewegungserkennung und Ton arbeitet, um eine interaktive Art der Informationsvermittlung zu erhalten. Der Nutzer soll dadurch nicht nur Informationen erhalten können, sondern diese auch wenn möglich deutlicher wahrzunehmen. In diesem Projekt wird das Thema des Klimawandels aufgegriffen, welches im aktuellen Zeitgeschehen ein wichtiger Bestandteil von Politik und Alltagsleben darstellt.

Projekt-Roadmap

Tätigkeit	Beteiligung
Projektmanagement	
Gruppeneinweisung für Github (Gitkraken)	René Sammeth
Organisation des Githubs	René Sammeth
Unterstützung bei Integrationsproblemen	René Sammeth
Unterstützung bei Unity-Problemen	René Sammeth
Konzept	
Erstellung Konzept	René Sammeth
Erstellung Physikalischer Prototyp	René Sammeth
Technische Umsetzung	
Main Menu	
Delay des Szenenübergangs	René Sammeth
Gestensteuerung (Hand-Cursor) + Bedienrechte	René Sammeth
World Scene	
3D Modellierung (Erdkugel + Kontinente)	René Sammeth
Shader-Erstellung	René Sammeth
Erstellung der Hand-Cursor Sprites	René Sammeth
Gestensteuerung (Hand-Cursor)	René Sammeth
Bedienrechte	René Sammeth
Interaktionen	René Sammeth

Tätigkeit	Beteiligung
Veraltet	
Gestensteuerung (Swipe, Zoom)	René Sammeth
OSC-Integration	René Sammeth
Eventplanung	
Zwischenpräsentation des Projektes	René Sammeth
Endpräsentation des Projektes (nur Aufnahme und Folie des eigenen Vortrags)	René Sammeth

Glossar

Begriff	Erklärung
Application Programming Interface	Das Application Programming Interface ist die sogenannte Schnittstelle zur Anwendungsprogrammierung oder auch Programmierschnittstelle genannt.
Asset (-Package)	In diesem Bericht wird der Begriff <i>Asset</i> für alle Ressourcen verwendet, die von Unity benötigt werden, um bestimmte Erweiterungen ausführen zu können oder Ressourcen die von frei verfügbaren Quellen erhalten wurden.
Augmented Reality	Bedeutet auf deutsch <i>Erweiterte Realität</i> und bezieht sich auf alle sichtbaren Objekte, welche nicht in der Realität zu sehen sind.
Bitmap	Ein Bilddateiformat für das Speichern von Bildern.
Blender	Eine kostenfreie Software, welche das Bearbeiten, Erstellen, Animieren usw. von 3D-Objekten ermöglicht.
Bloom	Ein Lichteffekt, welcher die Illusion eines sehr hellen Lichtes, ähnlich wie die Sonne, erzeugt.
Browser/Webbrowser	Ein Programm, welches die Darstellung von Webseite im Internet ermöglicht.
Game Over	Ein Ausdruck aus der Spieleindustrie der bedeutet, dass das Spiel zu Ende ist oder der Spieler an seiner Aufgabe gescheitert ist.
Hover	Bezeichnet das Schweben oder Verbleiben eines Cursors über ein Interaktionsobjekt.
Key-Frames	Bezeichnet in der Medienbranche, für z.B. Video, den Start oder das Ende einer Überleitung auf einer Zeitleiste.

Begriff	Erklärung
Lightweight Render Pipeline	Vorgebauter Renderer von Unity, welcher grafische Skalierung (z.B. für Mobile Anwendungen) ermöglicht, ohne Shaderberechnungsprogramme zu benötigen.
Mesh	Ansammlung von Vertices, Edges (ger. Kanten) und Polygone (auch Faces genannt), die ein drei-dimensionales Objekt beschreiben.
MipMap	Antialiasing-Technik für Texturen, die zur Verbesserung von Bildqualität und Geschwindigkeit eingesetzt wird.
Point-Of-No-Return	Bezeichnet einen Grenzpunkt, ab welchem eine gewisse Aktion nicht mehr aufzuhalten ist.
Post Processing	Hinzufügen von Full-Screen Effekten, die an die Kamera übergeben werden, bevor das Bild auf dem Bildschirm dargestellt wird.
Quaternion	Format für die Representierung von Rotationen in Unity.
Skybox	Eine Skybox ist ein Bild oder Textur, die für Hintergründe in Spielen oder Ähnlichem verwendet, um eine Illusion eines Horizontes und einer weiterführende Landschaft zu erwecken.
Software Development Kit	Bietet hilfreiche Ressourcen zur Erstellung von Software (z.B. Dokumentationen oder Werkzeuge).
Sprite	Ein zwei-dimensionales Objekt, das eine grafische Textur besitzt.
Standalone-Version	Die Software ist unabhängig von anderen Programmen oder Hardware einsetzbar.
Time-Of-Flight	Der Begriff bedeutet auf deutsch Flugzeit, die sich vom Auf treffen eines visuellen Strahls bis zur Rückkehr des Ausgangspunktes berechnet.
Touch	Auf deutsch <i>Berührung</i> und bezieht sich auf alle Interaktionen der Hand einer Person mit einem Bildschirm.

Begriff	Erklärung
Tracking	Deutsch für Verfolgen und bedeutet im Kontext dieses Berichtes das Verfolgen und Erfassen von Bewegungen einer Person.
User Interface	Beschreibt Schnittstellen, mit welcher der Nutzer mit dem System interagieren kann, z.B. in Form von Menüs oder Klickbaren Objekten.
Vertex/Verticies	Ein Punkt oder Position im drei-dimensionalen Raum.
Virtual Machine	Eine Virtual Machine ist eine Software oder ein Betriebssystem, das im Stande ist alle Funktionen eines separaten Rechners, auf dem verwendeten Computer, auszuführen.
Virtual Reality	Bezeichnung für die computergestützte Erweiterung der Realitätswahrnehmung.
ZIP	Der Name eines Dateiformates, welches für das verlustfreie Komprimieren von Dateien verwendet wird.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
2D/3D	zwei-dimensional/drei-dimensional
API	Application Programming Interface
Covid-19	Coronavirus Disease 2019
IP	Internet Protocol
OSC	Open Sound Control
PC	Personal Computer
RGB	Red Green Blue (= Grundfarben eines digitalen Bildschirms)
SDK	Software Development Kit
UI	User Interface
VGB	Visual Gesture Builder
WPF	Windows Presentation Foundation

Inhaltsverzeichnis

1 Konzept	13
1.1 Motivation	13
1.2 Konzeptidee	13
1.3 Physikalischer Prototyp	15
2 Komponenten / Bestandteile	16
2.1 Microsoft Kinect v2	16
2.2 Unity	17
2.3 Open Sound Control (veraltet)	18
3 Technische Umsetzung	19
3.1 3D Modellierung	19
3.1.1 3D Modelle	20
3.1.2 Shaders, Textures und Materials	21
3.2 extOSC (veraltet)	24
3.3 Body Tracking	25
3.3.1 WPF-Application	25
3.3.2 Unity	25
3.4 Bedienrechte	28
3.5 Gestensteuerung	29
3.5.1 Visual Gesture Builder	29
3.5.2 Joint Positionen und Hand States	33
3.5.3 Hand-Cursor	36
3.5.4 Raycasting	39

INHALTSVERZEICHNIS

3.5.4.1 Physic Raycast	39
3.5.4.2 Graphic Raycast	40
3.6 Interaktionen	41
3.6.1 Erdkugel-Rotation	41
3.6.2 Kontinente	42
3.6.3 User Interfaces	45
3.6.4 Szenenwechsel	47
3.7 Aufteilung der Szenen	48
3.7.1 Intro	48
3.7.2 Main Menu	49
3.7.3 World Scene	49
3.7.4 Quiz	50
3.7.5 Impressum	50
4 Evaluation des interaktiven Schaukastens	51
4.1 Implementierung der 3D Modelle	51
4.2 Bedienrechte	52
4.3 Gestensteuerung	52
4.3.1 Gesten	52
4.3.2 Hand-Cursor	52
4.4 Interoperabilität	53
4.5 Wiederverwendbarkeit	53
5 Soll- / Ist-Stand	54
6 Fazit und Ausblick	56

INHALTSVERZEICHNIS

6.1 Fazit des erarbeiteten Konzepts	56
6.2 Möglichkeiten der Verbesserung	56
6.3 Ausblick und Weiterentwicklung	57
7 Literaturverzeichnis	58

1 Konzept

1.1 Motivation

Im Zeitalter der Digitalisierung ist es einfacher denn je Zugang zu Informationen jederzeit und überall zu erhalten. Trotzdem sind viele Arten der Kommunikation bzw. der Informationsverbreitung immer noch nicht in dieser Neuzeit angekommen. Aus diesem Grund soll der Schaukasten in die heutige Zeit transformiert werden, um nicht als ein rein sekundäres Kommunikationsmedium dahin zu verwelken.

1.2 Konzeptidee

Die ursprüngliche Konzeptidee beinhaltete einen Schaukasten, welcher hauptsächlich der Informationsdarstellung und Wegweisung innerhalb der Hochschule Georg Simon Ohm dient. Hierbei war der Ansatz mit Hilfe eines großen Touch-Bildschirms die neuesten Informationen bezüglich Hochschulnews und Studienplanänderungen öffentlich im Gebäude zugänglich zu machen. Außerdem sollte ein 3D Wegweiser für Räume und Gebäude Studienanfängern helfen sich an der Hochschule zurecht zu finden. Dies könnte jedoch auch mit einer mobilen App gelöst werden und ist in dieser Art bereits erstellt worden.

Aufgrund dessen kam das Projektteam zu dem Entschluss einen interaktiven Schaukasten zu entwickeln, der als Hauptinteraktion eine Gestensteuerung nutzt und für mehrere Bereiche einsetzbar ist. Da Touch-Steuerung fast überall im Alltag anzutreffen ist, von Autos über Handys und viele mehr, wollen wir die Gestensteuerung, die bisher nur in wenigen Teilen des digitalem Zeitalters verwendet wird, als Interaktionsart verwenden. Das erlaubt uns außerdem den Nutzer in unser Projekt zu investieren, da die Zeitspanne heutzutage nur gering ist, um die Aufmerksamkeit des Benutzers zu erhalten.

Das Thema, welches der Schaukasten in unserem Konzept darbieten soll ist der Klimawandel. Der Nutzer soll durch Verwendung der Gestensteuerung sich interaktiv damit auseinandersetzen und informieren können. Hierbei wird auch das Annähern an den

1 KONZEPT

Schaukasten als Interaktion eingebunden, um dem Benutzer mitzuteilen, dass der Klimawandel ab einem gewissen Punkt nicht mehr aufzuhalten ist. Des Weiteren ist für diesen Fall auch geplant den Schaukasten innerhalb eines Raumes zu platzieren, damit mit Hilfe von Ton- und Lichtmanipulation ein noch stärker Eindruck hinterlassen wird. Das Konzeptdesign ist in der folgenden Abbildung dargestellt.

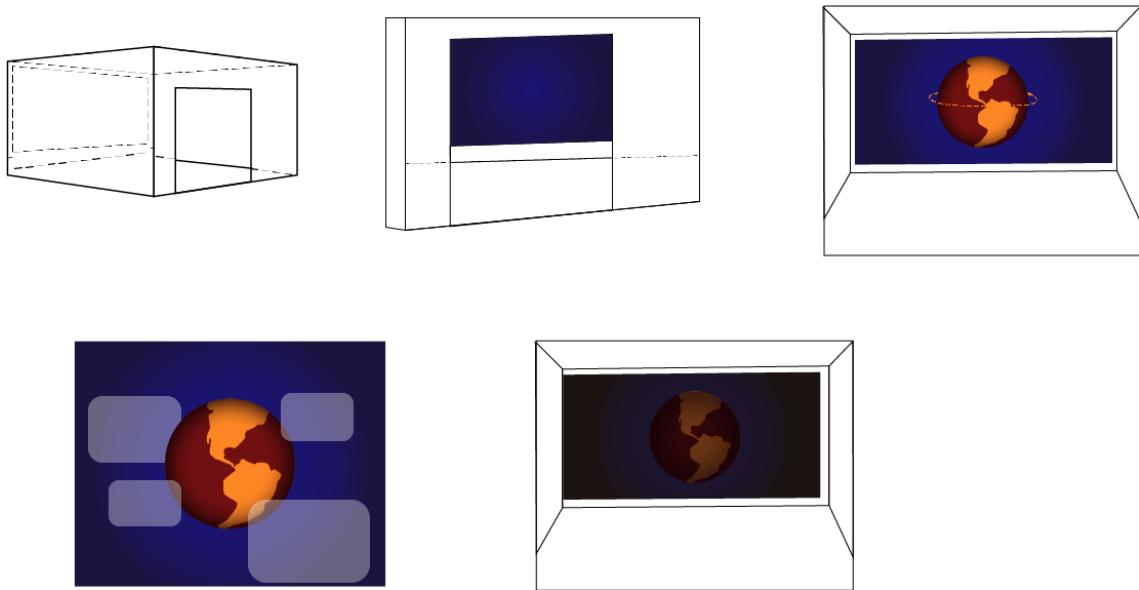


Abbildung 1: Konzeptdesign

1.3 Physikalischer Prototyp

Um das Projekt besser zu visualisieren und die Hauptfunktionen anschaulich darzustellen, wurde ein physikalischer Prototyp erstellt.

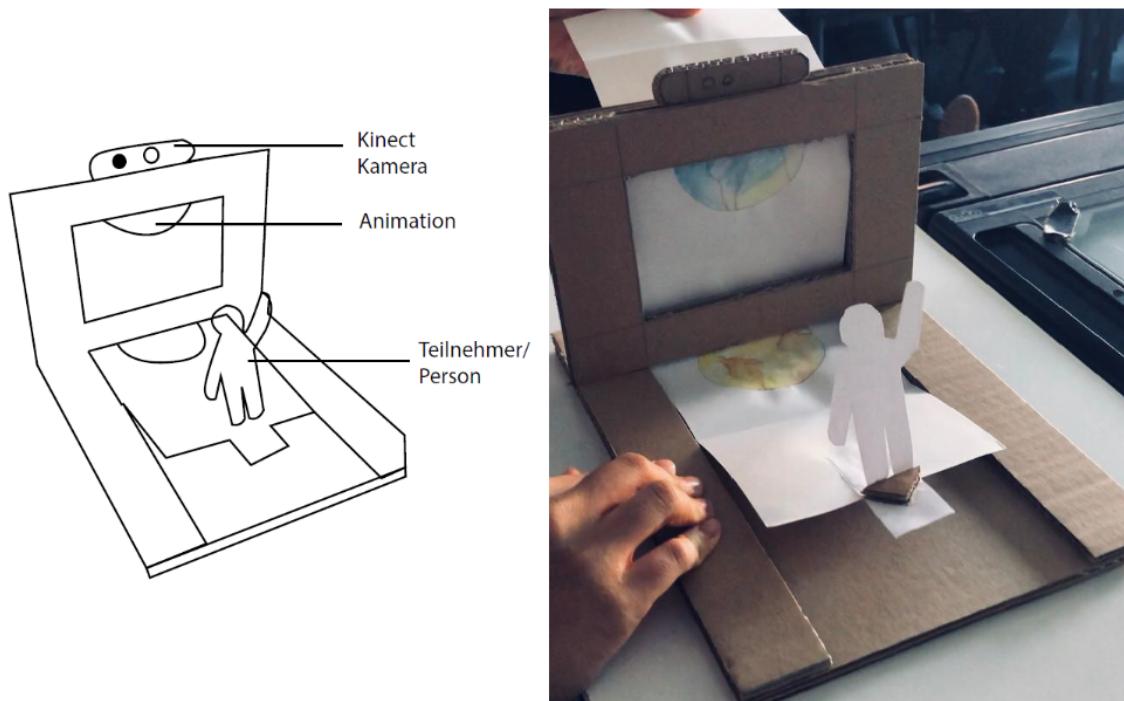


Abbildung 2: Physikalischer Prototyp

2 Komponenten / Bestandteile

2.1 Microsoft Kinect v2

Für das ausgearbeitete Projekt Konzept bietet sich die Kinect v2 gut an, da diese bereits eine SDK (Software Development Kit) für das *Tracking* (ger. Verfolgen) von Personen beinhaltet. Des Weiteren besitzt sie die Möglichkeit die Positionen von einzelnen Körperteilen, mit Hilfe von sogenannten *Joints* (ger. Gelenke), von Personen zu ermitteln. Insgesamt können 25 Joints erkannt und ausgelesen werden. Die maximale Anzahl von Personen, die gleichzeitig als sogenanntes Skelett getrackt werden können, bestehend aus den Joints und *Bones* (ger. Knochen), ist sechs. Die Kinect besitzt eine RGB-Kamera (1920x1080), welche mit einer Framerate von 30fps (fps= Frames per Second) aufnimmt, einen Tiefensensor mit Infrarotkamera (512x424) und Infrarotsender, sowie ein Mikrofon. Die Reichweite der Kamera hierbei befindet sich zwischen 50 cm und 4,5 m. Dadurch kann sowohl mit Tiefenbilddaten als auch mit Farbbilddaten gearbeitet werden.

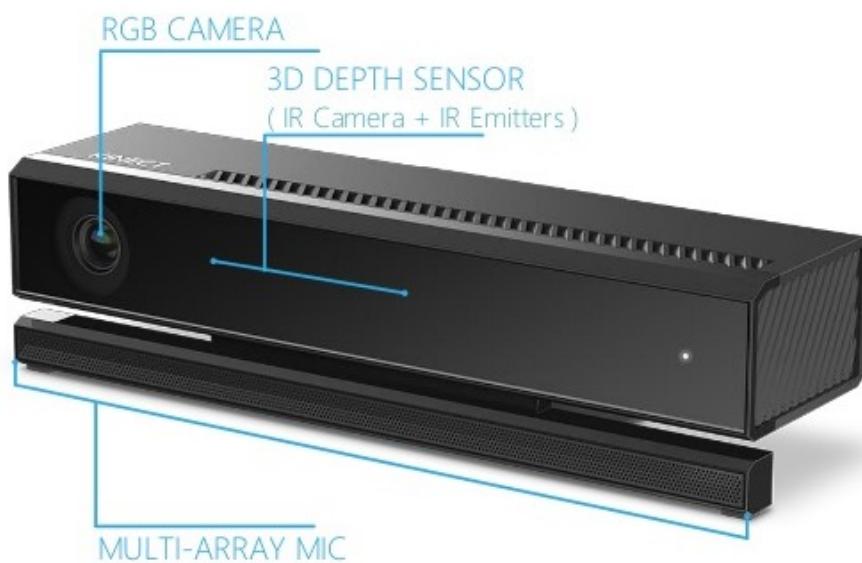


Abbildung 3: Microsoft Kinect v2 Hardware

Für das Erkennen von Personen oder Gegenständen verwendet die Kinect die sogenannte time-of-flight Technologie als Tiefenerkennung, welches in der Abbildung 2 dargestellt ist.

Dies bedeutet, dass die Zeit für das Aussenden und Empfangen des Infrarotlichtsignals gemessen wird und dadurch die Entfernung zu den erfassten Objekten, Personen etc. errechnet werden kann.

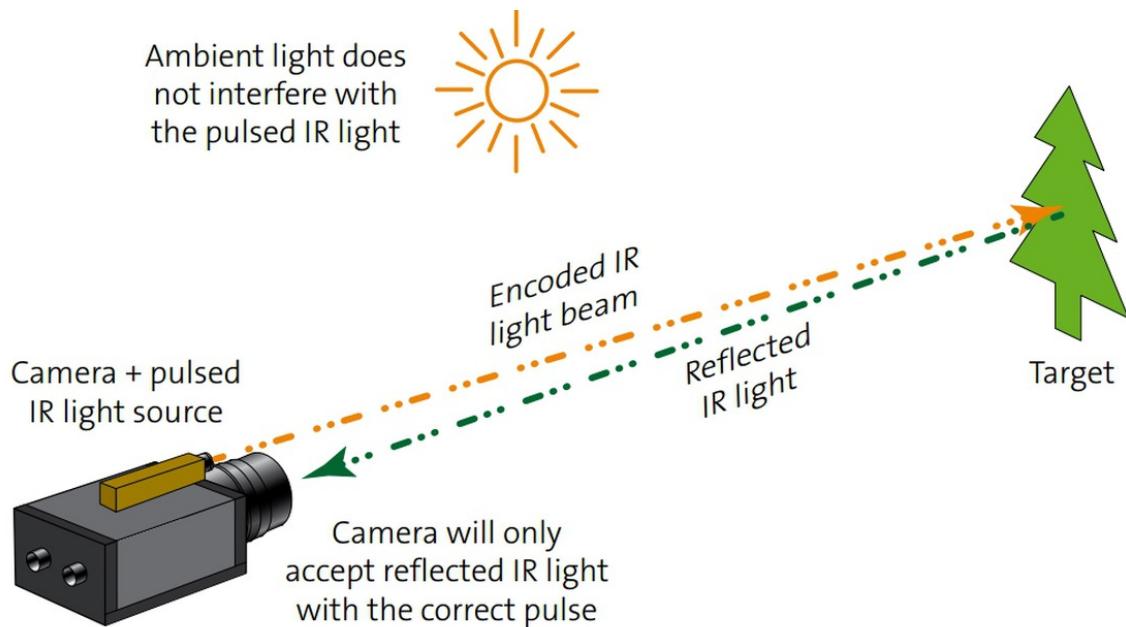


Abbildung 4: Time-Of-Flight Prinzip

Des Weiteren sind in der Kinect v2 SDK drei verschiedenen Anwendungen inkludiert. Diese sind das Kinect Studio, der Kinect v2 SDK Browser und der Visual Gesture Builder.

2.2 Unity

Unity ist eine plattformübergreifende Game Engine, entwickelt von Unity Technology, und ist seit dem Jahr 2018 auf über 25 verschiedenen Plattformen verfügbar. Die Engine kann für 3D, 2D, *virtual reality* und *augmented reality games* verwendet werden und sogar für Simulationen. Unity wird auch außerhalb der Videospielbranche verwendet, beispielsweise für Filme/Animationen, Architektur, Konstruktionen oder Ingenieursprojekte. Unity bietet eine *primary scripting API* (Application programming interface) in C für den Unity Editor und auch in Form von Plugins oder Spiele. Für Nutzerfreundlichkeit bzw. Bedienfreundlichkeit verfügt die Engine über eine *Drag and Drop* Funktionalität. Innerhalb 2D Spiele erlaubt Unity das Importieren von Sprites und fortgeschrittene *2D world*

renderer. Für 3D Spiele bietet Unity die Möglichkeit Anpassungen von Spezifikationen für Texturkompression und MipMaps vorzunehmen. Des Weiteren unterstützt Unity viele weitere Optionen für das Erstellen einer 3D Umgebung. Es wird geschätzt, dass seit 2018 etwa die Hälfte aller neuen mobile games und 60 Prozent von *augmented reality* und *virtual reality* Inhalte mit Hilfe von Unity erstellt wurden.

2.3 Open Sound Control (veraltet)

Open Sound Control, kurz OSC, ist ein Protokoll für die Kommunikation zwischen Computern, Sound Synthesizers und anderen multimediale Geräte, welche mit moderner Netzwerktechnologie ausgestattet sind. Dieses Protokoll bietet alles Notwendige für eine Echtzeitübertragung von Steuersignalen oder anderen Medienprozessen und bleibt dabei flexibel und einfach zu implementieren. Die übertragenen Signale werden in Form von Nachrichten, sogenannten OSC-Messages, an eine Schnittstelle gesendet, um dadurch gewisse Ausgaben steuern zu können. Eine OSC-Message ist wie folgt aufgebaut:

- **Address Pattern:** Gibt Empfangsadresse an und ist beginnend mit einem „/“ gekennzeichnet
- **Type Tag:** Ein Type Tag ist ein String, welcher Anzahl und Typ der zu übertragenen Parameter angibt, beginnend mit einem Komma, und enthält pro Parameter einen Kennbuchstaben (i = Int, f = Float, s = String, b = BLOB)
- **String Arguments:** Die String Arguments enthalten die eigentlichen Daten, also die einzelnen Werte der Parameter, die übertragen werden sollen.

Mit diesem System wurde in dem Projekt das Problem der Framemanipulation (Frame = ger. Bild) innerhalb von Unity, auf dem Windows Betriebssystem, umgangen, indem die notwendigen Joint-Koordinaten an einen MAC IOS Computer gesendet werden.

Aufgrund der Tatsache, dass im aktuellen Projekte eine Virtual Machine für Windows benutzt werden konnte, um die Kinect v2 auf Apple geräten verwenden zu können, ist OSC überflüssig. Deshalb ist auch entschieden worden nicht weiter mit einer OSC-Übertragung in diesem Projekt zu arbeiten.

3 Technische Umsetzung

3.1 3D Modellierung

Das 3D Modell der Erdkugel ist neben dem Intro der zweite Eye-Catcher des interaktiven Schaukastens. Aus diesem Grund fiel der Beschluss auf eine realistische Erdkugel, die in Blender mit Hilfe von Open-Source Texturen erstellt wurde. Der Nutzer kann mit Hilfe der Gesteuerung sich Informationen der einzelnen Kontinente, in Zusammenhang mit dem Klimawandel, beschaffen. Aufgrund der Tatsache, dass die einzelnen Kontinente anklickbar sein sollen, müssen diese ein eigenständiges 3D-Objekt sein. Dies gilt auch für den Kontinentrand, welcher als Hilfsmittel für das highlighten der Kontinentgrenzen dient. Für die Erstellung der 3D-Modelle fiel die Wahl auf das Entwicklungstool *Blender*, da hierbei bereits eine gute Grunderfahrung vorhanden ist. Im Folgenden wird der Modellierungsprozess für die Erdkugel und ihrer Kontinente beschrieben.



Abbildung 5: Kontinent Endstand

3.1.1 3D Modelle

Die Kontinente werden zusammen mit der Erdkugel erstellt, da diese zum Einen an die richtige Position der Erdkugel gesetzt werden müssen und zum Anderen sich an die Oberfläche der Erde anpassen sollen. Die Erdkugel wird durch das Verformen eines rechteckigen Meshes, mit Hilfe des *Simple Deform Modifiers*, erstellt. Hierbei wird das Mesh des Rechtecks um 180 der Z-Achse und um 360 der X-Achse eines leeren Objektes verformt. Die Kontinente hingegen mussten zunächst per Hand modelliert werden. Für die Erstellung der Kontinentränder werden das Kontinent-Mesh verdoppelt und die Vertices des Randes extrudiert. Im Anschluss werden diese mit einem *Solidify-Modifier* versehen, welcher die Dicke des Randes manipulieren kann. Hiernach können die Vertices zwischen dem Rand entfernt werden.

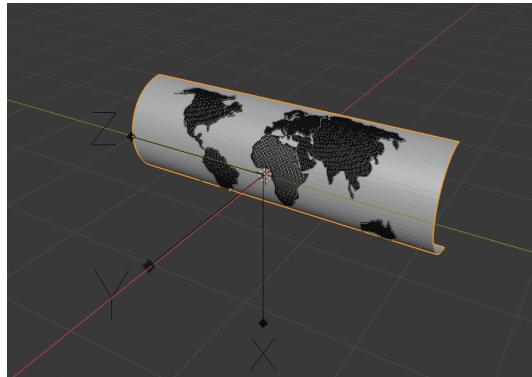


Abbildung 6: Deform Z-Achse um 180 Grad

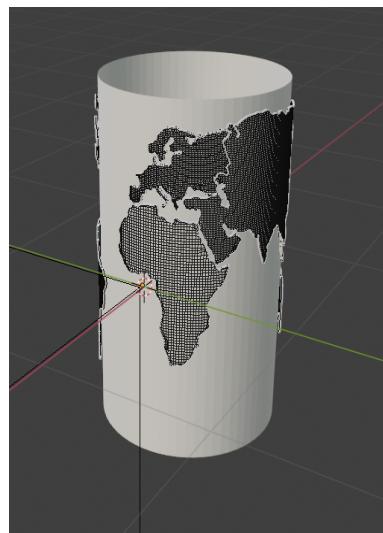


Abbildung 7: Deform
X-Achse um 360 Grad

Jetzt können die Kontinente an die gewünschte Position verschoben werden. Hierbei ist es hilfreich eine Textur auf dem Rechteck (Mesh der Erdkugel) zu verwenden. Wenn die Positionen korrekt eingestellt sind, werden die Kontinente jetzt mit dem *Surface Deform Modifier*, welcher das Mesh der Kontinente auf das Mesh der Erdkugel anpasst, an die Erdkugel gebunden.

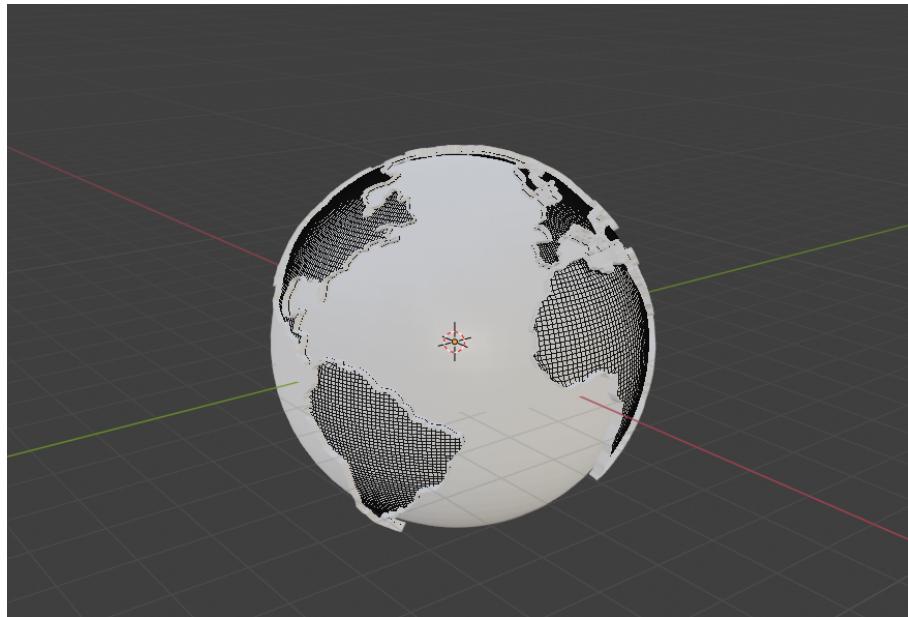


Abbildung 8: Kombination der Deformungen

Das 3D Modell kann nun in Unity als .fbx oder .blend Datei in Unity importiert werden. Alle Objekte müssen einzeln exportiert werden. Hierbei wird das Mesh automatisch übernommen, die Texturen müssen jedoch im Objekt-Inspektor nochmals importiert werden, weil diese externe Texturen sind und nicht in Unity erstellt wurden.

3.1.2 Shaders, Textures und Materials

Für die farbliche und texturelle Gestaltung der Kontinente und der Erdkugel werden drei Komponenten verwendet, welche im Folgenden zunächst erklärt werden.

- **Shaders:** Shader sind kleine Skripte, welche die mathematischen Berechnungen und Algorithmen für die Berechnung der Farbtöne eines jeden dargestellten Pixels enthält, basierend auf Lichteinflüsse und Materialien-Einstellungen.
- **Textures:** Textures (ger. Texturen) sind Bitmap-Bilder. Textures können jedoch neben Oberflächeninformationen bezüglich Farbe, auch Informationen für das Reflexionsverhalten oder Rauheit beinhalten.
- **Materials:** Materials (ger. Materialien) sind Definitionen, wie die Oberfläche eines Objektes dargestellt wird. Hierbei werden Informationen bezüglich benutzter

Textur-Verweise, Farbtöne und mehr. Die verfügbaren Optionen des Materials sind vom zuvor gewählten Shader abhängig.

Für das Projekt werden in Unity zwei verschiedene Arten von Shader verwendet. Für die Erdkugel ist ein bereits vorgefertigter *Surface-Shader* (ger. Oberflächen-Shader) benutzt worden, der in dem frei verfügbaren *Asset-Package* enthalten ist, das bereits im vergangenen Semester verwendet wurde. Hierbei ist außerdem auch das Material der Skybox aus dem Paket verwendet, jedoch unter Verwendung von Hintergrundbildern, die in Blender erstellt wurden.

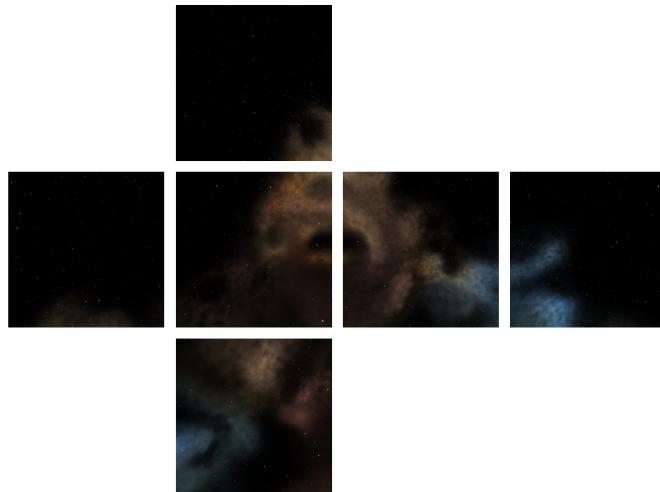


Abbildung 9: Skybox

Die Kontinente hingegen sind mit einem *Shader Graph* ausgestattet, welcher ein Bestandteil von Unity ist. Dieser hat sehr viele Ähnlichkeiten mit dem Node-Editor von der 3D-Software Blender, wobei kein Code geschrieben werden muss, sondern einzelne Nodes (vorgefertigte Funktionen mit UI), die miteinander verbunden werden können um einen gewissen Effect für die Oberflächendarstellung zu erreichen. Um den Shader Graph in einer Scene verwenden zu können, müssen zunächst die zusätzlichen Pakete *Lightweight Render Pipeline* und *Post Processing* importiert werden. Im Anschluss kann dem Projekt unter *Bearbeiten > Projekteinstellungen > Grafik* ein *Lightweight Render Pipeline Asset* zugewiesen werden, das zuvor erstellt worden ist. Hierbei ist zu beachten, dass nach dem

3 TECHNISCHE UMSETZUNG

Einbinden dieses Assets die Standard-Shader von Unity nicht mehr funktionsgerecht eingesetzt werden können.

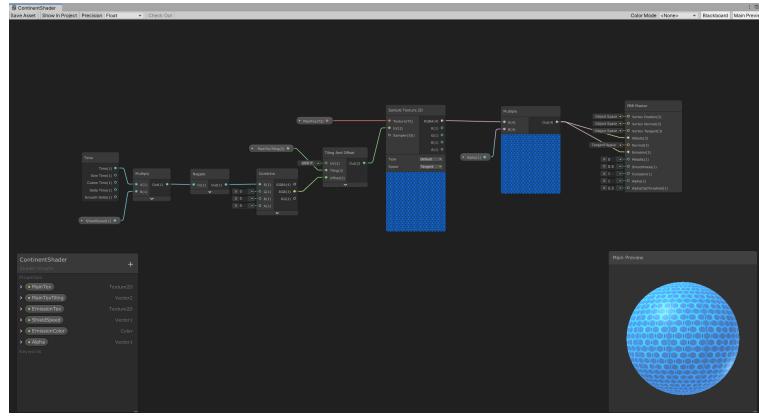


Abbildung 10: Shader Graph

Um den Rand eines jeden Kontinenten ein leichtes Strahlen/Leuchten zu geben, wurde die Kamera mit dem Post-Processing-Element *Bloom* ausgestattet. Durch das Abändern der Sichtbarkeitsebene der Kontinentränder auf eine Post-Processing-Ebene, kann verhindert werden, dass andere Objekte den Bloom-Effekt zugewiesen bekommen. Das Post-Processing-Element der Kamera muss die ausgewählte Ebene ebenfalls als Ziel des Effektes auswählen.

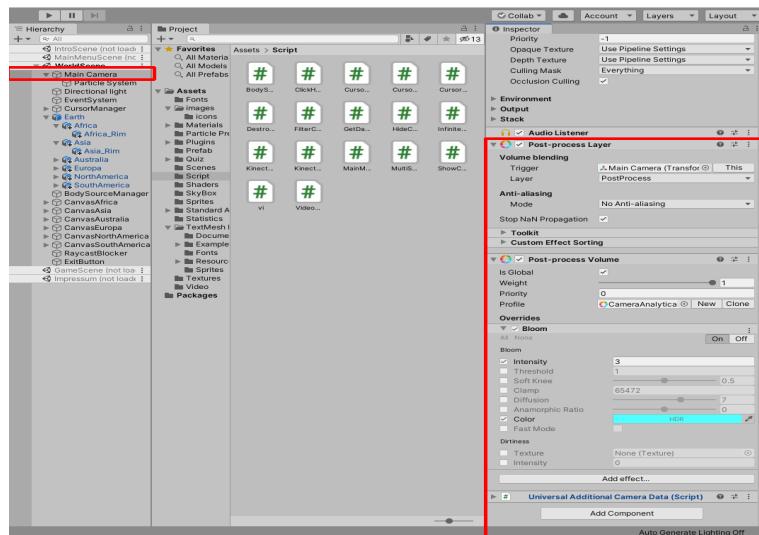


Abbildung 11: Post Processing

3.2 extOSC (veraltet)

Das Open Sound Control Protocol (OSC) wird mit Hilfe des Plugins extOSC in das Unity Projekt eingebunden, welches im Assetstore zur freien Verfügung heruntergeladen und importiert werden kann. Um den Datentransfer der OSC-Messages zu gewährleisten, muss beim Sender zum einen die OSC Manager und die Simple Message Transmitter Klasse an ein leeres Gameobjekt gebunden werden. Beim Empfänger hingegen muss statt dem Transmitter ein Receiver verwendet werden. Anschließend muss nur noch die IP des Empfängers im Transmitter eingegeben werden.

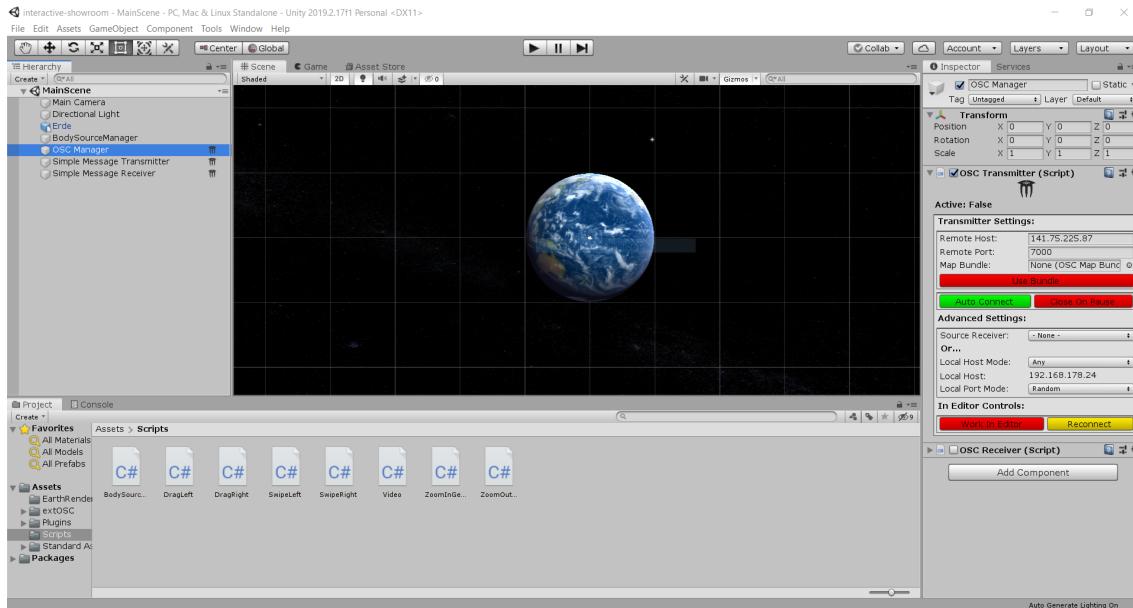


Abbildung 12: extOSC

Wie bereits im Abschnitt Open Sound Control erwähnt, wurde das Übertragen von OSC-Messages im Projekt ausgelassen, da die Anforderung für Interoperabilität über Virtual Machines und Ähnliches erfolgt.

3.3 Body Tracking

3.3.1 WPF-Application

Windows Presentation Foundation, kurz WPF, ist ein User Interface Framework, welches für die Erstellung von Desktop-Anwendungen auf Windows Rechnern verwendet werden kann. Die WPF Entwicklungsplattform unterstützt ein breites Spektrum an Entwicklungsoptionen wie ein Anwendungsmodell, Ressourcen, Steuerungselemente, Grafiken, Layouts und Sicherheitseinstellungen. WPF ist ein Subset von dem .NET Framework unter der Verwendung von ASP.NET oder Windows Forms. WPF verwendet die Extensible Application Markup Language (XAML), um ein deklaratives Modell für Anwendungsprogrammierung bieten zu können. Die unterstützten Programmiersprachen sind *Visual Basics* und *C#*. Beim Erstellen der Applikation mit der Kinect v2 SDK ist zu beachten, dass das richtige .NET Framework verwendet wird und die Verweise für die Kinect SDK hinzugefügt werden. Beim Testen der Applikation kam es jedoch zu einem unerwartenden *Unidentified Framework Error*, wofür aktuell noch keine Lösung gefunden wurde.

3.3.2 Unity

Aufgrund der Tatsache, dass die Standard WPF-Applikation nicht funktionierte, ist als Alternative die Game Engine Unity verwendet worden, da diese bereits von Microsoft ein Paket zur Verfügung gestellt bekommen hat. Um die Kinect v2 SDK in Unity verwenden zu können, muss zunächst das Unity Pro Package von Microsoft <https://developer.microsoft.com/en-us/windows/kinect/> heruntergeladen werden. Nach Download und Entpacken der Zip-Datei kann das Paket folgendermaßen in Unity installiert werden.

1. Erstellen eines neuen Unity Projektes
2. Assets > Import Package > Custom Package > wähle Kinect.xxx.unitypackage in der heruntergeladenen Zip-Datei aus
3. Bestätige mit Import-Button

Für das Testen der SDK in Unity kann von der heruntergeladenen Datei der Ordner

KinectView per Drag and Drop in den Ordner Assets des Unity-Projektes kopiert werden. Dieser beinhaltet bereits eine Szene für die Darstellung der aufgenommenen Personen.

Der Kern des Body Tracking ist der sogenannte **Body Source Manager**, welche für jeden Frame alle Body-Daten inklusive der Joint-Daten enthält. Da in unserem Projekt nur die Joint-Koordinaten ausschlaggebend sind, brauchen wir keine Farbdaten der Kinectkamera. Hierdurch können sowohl Speicherauslastung gespart als auch die Performance stabil gehalten werden.

```
1  public class BodySourceManager : MonoBehaviour {  
2      private KinectSensor _Sensor;  
3      private BodyFrameReader _Reader;  
4      private Body[] _Data = null;  
5  
6      public Body[] GetData() {  
7          return _Data;  
8      }  
9  
10  
11      void Start () {  
12          _Sensor = KinectSensor.GetDefault();  
13          if (_Sensor != null){  
14              _Reader = _Sensor.BodyFrameSource.OpenReader();  
15              if (!_Sensor.IsOpen){  
16                  _Sensor.Open();  
17              }  
18          }  
19      }  
20  
21      void Update () {  
22          if (_Reader != null){  
23              var frame = _Reader.AcquireLatestFrame();
```

```
24     if (frame != null){  
25         if (_Data == null){  
26             _Data = new  
27                 ↳ Body[_Sensor.BodyFrameSource.BodyCount];  
28         }  
29         frame.GetAndRefreshBodyData(_Data);  
30         frame.Dispose();  
31         frame = null;  
32     }  
33 }  
34  
35 void OnApplicationQuit(){  
36     // dispose _Reader and close _Sensor  
37 }  
38 }
```

Das folgende Bild zeigt alle Darstellungsmöglichkeiten, welche die Kinect v2 besitzt. Diese Szene ist bereits in dem Unity Paket von Microsoft enthalten.

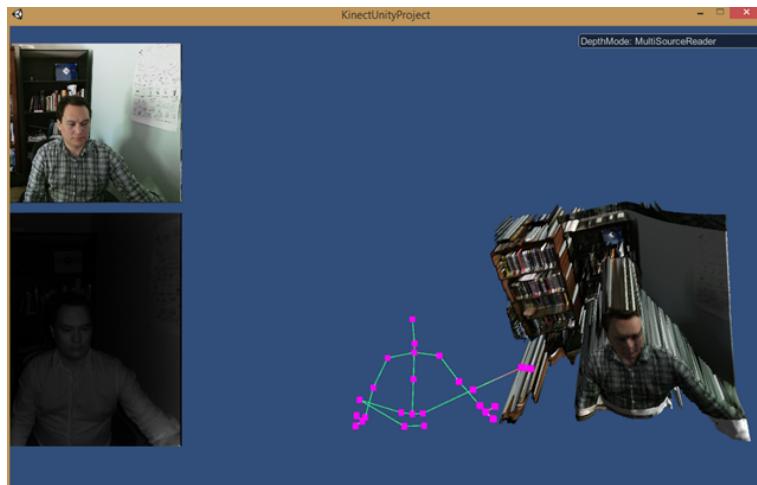


Abbildung 13: KinectView

3.4 Bedienrechte

Um unerwünschte Überlappungen von Gesten und Personen beim Interagieren mit dem Schaukasten zu vermeiden, wird die Person mit dem geringsten Abstand zur Kinect als *aktiver* Benutzer ausgewählt, welcher als einzige Person die Gestensteuerung nutzen kann. Der unten gezeigte Code zeigt die Implementation der Bedienrechte für die Person mit dem geringsten Abstand zur Kinect.

```
1  foreach(var body in bodies){  
2      if(body == null){  
3          Continue;  
4      }  
5      float distance = 10;  
6      float distanceNew = body.Joints[JointType.Head].Position.Z;  
7      if(distanceNew != 0 && distanceNew <= distance){  
8          distance = distanceNew;  
9          if(body.IsTracked){  
10              //do something  
11          }  
12      }  
13  }
```

Der Array **bodies** enthält alle getrackten Personen, die durch die Klasse Body Source Manager übergeben werden. Der Variable **distanceNew** wird die Z-Koordinate des getrackten Kopfes gegeben. Wenn diese sich zwischen 0 und der maximalen Distanz befindet, sowie den geringsten Abstand zur Kinect hat (die Z-Koordinate aller getrackten Personen werden durch Iteration berücksichtigt), bekommt die Person mit besagter Koordinate die Bedienrechte.

Eine weitere Möglichkeit ist dem aktiven Benutzer eine ID zuzuweisen, welche erst unter bestimmten Bedingungen an eine andere Person weitergegeben wird und diese folglich zum aktiven Nutzer wird.

3.5 Gestensteuerung

3.5.1 Visual Gesture Builder

Der Visual Gesture Builder, kurz VGB, ist eine Anwendung, welche bereits in der Kinect v2 SDK enthalten ist und es ermöglicht eigene Gesten zu erstellen. Für die Verwendung mit Unity muss das Visual Gesture Paket von Microsoft in das Unity Projekt importiert werden.

Geste aufnehmen

Das Aufnehmen einer Geste ist mit Hilfe von Kinect Studio möglich. Nach Start der Anwendung ist ein Reiter *Record* zu sehen, welcher eine Ansicht für 2D als auch 3D besitzt. Um eine Geste aufzunehmen muss nun zunächst die Kinect an dem Computer angeschlossen sein und mit Kinect Studio verbunden werden. Die Verbindung zwischen Kinect und Kinect Studio kann hergestellt und getrennt werden durch das Betätigen des *connected*-Icons. Nach stoppen der Aufnahme wird diese automatisch in das Standard Repository als *myClip.xef*-Datei gespeichert. Hierbei ist es von Vorteil, wenn Personen mit unterschiedlichen Körperbauten die Geste öfters ausführen, um die Präzision der Geste zu steigern.

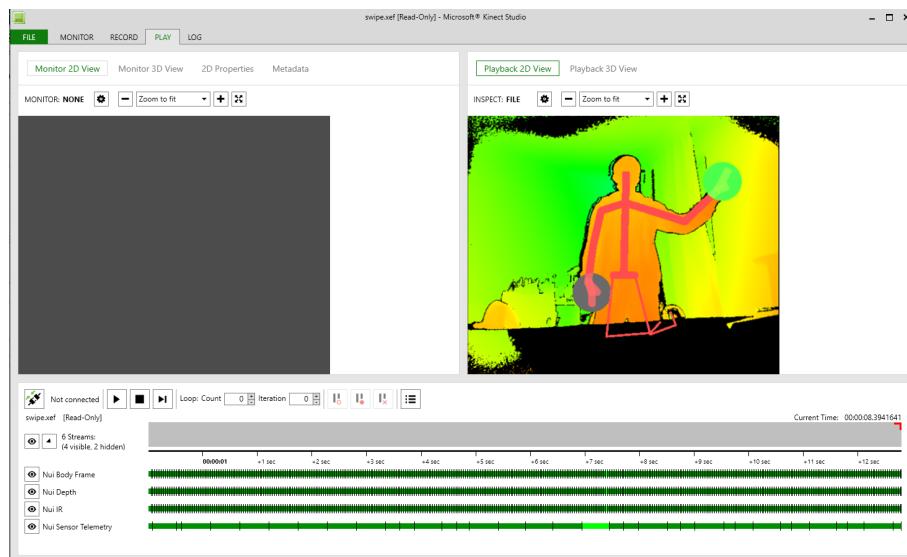


Abbildung 14: Gestenaufnahme mit Kinect Studio

Geste erstellen

Zunächst muss eine neue **Solution** erstellt werden indem über den Reiter *File > New Solution* ausgewählt wird und mit einem Namen versehen wird. Mit einem Rechtsklick auf die Solution muss ein neues Projekt angelegt werden. Hierbei steht zusätzlich die Auswahl mit *Wizard* zur Verfügung, welcher mehr Hilfestellung und Erklärung bietet als die Auswahl ohne Wizard und eignet sich insbesondere für Erstversuche. Beim Erstellen des Projektes der Geste können bestimmte Körperbereiche ignoriert werden, um unerwünschte Probleme beim Ausführen der Geste zu verhindern.

Man unterscheidet beim VGB zwischen einer kontinuierlichen und diskreten Geste. Eine diskrete Geste ist von Natur aus ein Boolean, das heißt sie kann nur zwei verschiedene Zustände besitzen. Ein Beispiel für eine solche Geste ist das Peace-Zeichen oder das formen eines X'es mit den Armen. Diese Art von Geste ist im Gegensatz zu einer kontinuierlichen Geste recht simpel umzusetzen, denn eine kontinuierliche Geste kann aus mehrfachen Zuständen bestehen und folgt einem bestimmten Ablauf.

Nach Erstellung des Gesten-Projektes können die Key-Frames für die Geste im VGB markiert werden. Wenn die Geste innerhalb der Aufnahme öfters vorkommt, können die Key-Frames auch mehrfach gesetzt werden, um diese leichter einsetzbar zu machen. Bei der abschließenden Speicherung kann die Geste direkt in die Streaming Assets des Projektes gespeichert werden.

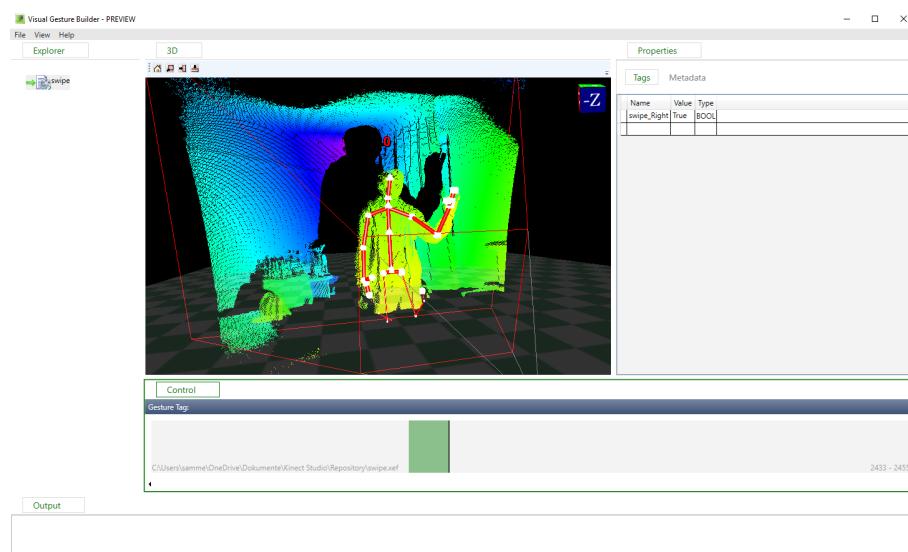


Abbildung 15: Gestenerstellung mit VGB

Geste verwenden

Der folgende Code-Ausschnitt zeigt den Initialisationscode für das Importieren der Gesten-Datenbank und das Registrieren von Events, wenn neue Daten ankommen. In diesem Beispiel wird sowohl eine diskrete Geste **myGesture** und eine kontinuierliche Geste **gestureProgress**, die den Fortschritt der Geste zeigt, verwendet.

```
1  _gestureDatabase = new
   ↵  VisualGestureBuilderDatabase(@"Gestures/myGesture.gbd");
2  _gestureFrameSource = new VisualGestureBuilderFrameSource(_kinect, 0);
3
4 // Add all gestures in the database to the framesource..
5 _gestureFrameSource.AddGestures(_gestureDatabase.AvailableGestures);
6
7 foreach (var gesture in _gestureDatabase.AvailableGestures){
8     if (gesture.Name == "myGesture"){
9         _gesture = gesture;
10    }
11    if (gesture.Name == "gestureProgress"){
12        _gestureProgress = gesture;
13    }
14 }
15 _gestureFrameReader = _gestureFrameSource.OpenReader();
16 _gestureFrameReader.IsPaused = true;
17 _gestureFrameReader.FrameArrived += reader_FrameArrived;
```

Der nächste Ausschnitt zeigt wie die Stati der einzelnen Gesten, in Form eines einfachen UI-Elementes, ausgegeben werden können.

```
1 void reader_FrameArrived(VisualGestureBuilderFrameReader sender,
2   VisualGestureBuilderFrameArrivedEventArgs args){
3   using (var frame = args.FrameReference.AcquireFrame()){
4     if (frame != null && frame.DiscreteGestureResults != null){
5       var result = frame.DiscreteGestureResults[_salute];
6       if (result.Detected == true){
7         var progressResult =
8           frame.ContinuousGestureResults[_saluteProgress];
9         Progress.Value = progressResult.Progress;
10      }else{
11        Progress.Value = 0.0;
12      }
13    }
14  }
15 }
```

Vor- und Nachteile des VGB

Die folgenden Vorteile bietet der Visual Gesture Builder für das Projekt:

- Ausgeführte Gesten können mit Hilfe des Toleranzwerts gut erkannt werden.
- Jede Geste muss nur einmal geladen werden.
- Gesten funktionieren für Personen mit unterschiedlichem Körperbau (wenn diese in der aufgenommenen Geste enthalten sind).

Die Implementierung des VGB's bringt jedoch auch gewisse Nachteile mit sich:

- Im aktuellen Stand innerhalb von Unity nicht einsetzbar, da dies einen Absturz verursacht.
- Der VGB kann nur in Verbindung mit der Kinect verwendet werden.

3.5.2 Joint Positionen und Hand States

Aufgrund der Tatsache, dass der Visual Gesture Builder Unity zum Absturz bringt, wurde eine vorübergehende Alternative für die Gesteuerung erstellt. Diese verwendet die Möglichkeit des Zugriffs auf die Joint Koordinaten eines Bodies sowie die Tatsache, dass die Kinect drei verschiedene Hand States(Abbildung 9) unterscheiden kann. Für die jeweilige Geste wird eine Klasse erstellt, welche dem Gameobjekt der Erde angefügt wird.



Abbildung 16: Hand States

Der Zugriff auf die Joint Koordinaten und der Hand States ist mit Hilfe des sogenannten Body Source Managers möglich. Dieser muss zunächst einem leeren Gameobjekt angehängt werden, sowie für die Verwendung der Gesten an die jeweiligen Gesten-Klassen, was durch den Inspektor von Unity vereinfacht wird.

Joint Koordinaten:

Der folgende Code-Ausschnitt zeigt den Aufruf und Speichern der Joint-Koordinate Z des Kopfes aller getrackten Bodies in einer lokalen Variable **headCoord**. Für alle 25 Joints gibt es einen **JointType**, welcher mit einem bestimmten Namen und Index versehen ist und mit diesem aufrufbar ist.

```
1 void Update(){
2     bodies = bodyManager.GetData();
3     foreach(var body in bodies){
4         if(body.IsTracked){
5             var headCoord =
6                 body.Joints[JointType.Head].Position.Z;
7         }
8     }
```

Hand State:

Im Folgenden wird der Zugriff auf die verschiedenen Hand States, welche die Kinect erkennen kann.

```
1 void Update(){
2     ... // siehe Joint Koordinaten
3     var leftHandClosed = (body.HandLeftState == HandState.Closed);
4     var leftHandOpen = (body.HandLeftState == HandState.Open);
5     var leftHandLasso = (body.HandLeftState == HandState.Lasso);
6 }
```

Vor- und Nachteile von Joint Positionen und Hand States

Vorteile:

- Einfache diskrete Gesten wie beispielsweise *salutieren*, welche nur zwei verschiedene Stati besitzen (0 und 1), sind schnell und einfach umsetzbar.

Nachteile:

- Kontinuierliche Gesten, welche aus mehreren diskreten Gesten bestehen sind aufwendig und schwer umsetzbar.
- Gesten haben auf diese Weise keine Toleranzbereiche, was diese für die alltägliche Verwendung eher impraktikabel macht.

Temporäre Gesten

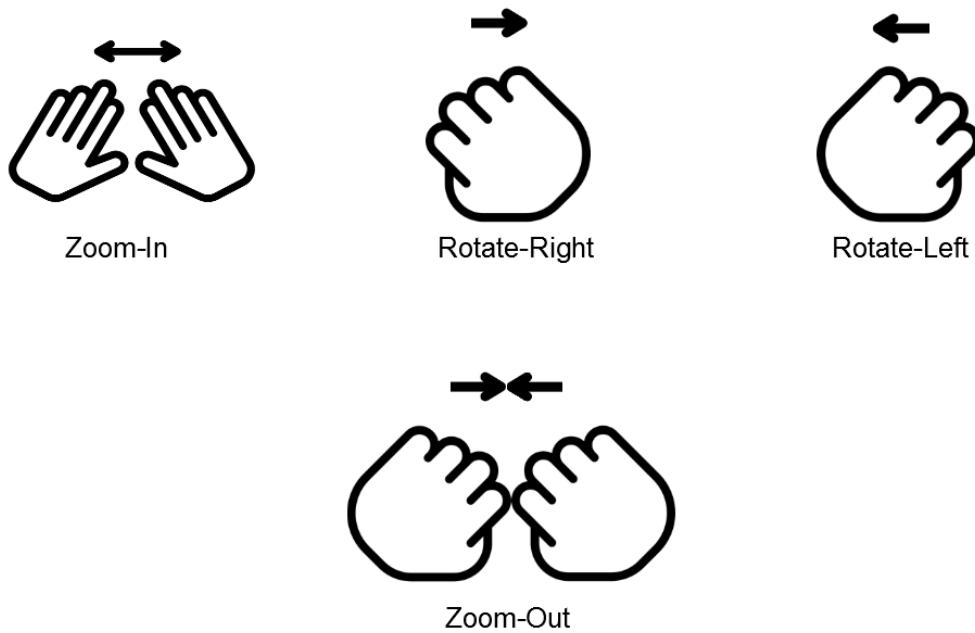


Abbildung 17: Gesten

3.5.3 Hand-Cursor

Das Feedback von der Projektzwischenpräsentation des vergangenen Semesters zeigte auf, dass die Gestensteuerung recht unpräzise und nach längerer Nutzungszeit anstrengend ist. Als Folge kam das Projektteam zu dem Entschluss einen Hand-Cursor als neue Gestensteuerung zu verwenden. Einer der Vorteile des Hand-Cursors ist, dass der Nutzer sich besser im Programm orientieren kann sowie die Steuerung präziser ist. Des Weiteren ist der Hand-Cursor Szenenübergreifend verwendbar. Es werden drei verschiedene Cursor-Icons verwendet, wobei sich das Cursor-Icon für die jeweilige Interaktionsart ändert, um den Nutzer auf Interaktionsmöglichkeiten hinzuweisen.



Abbildung 18: Hand-Cursor-Icons

Das Abändern des Cursor-Icons wird ermöglicht, indem der SpriteRenderer als Variabel gespeichert wird. Diese kann dann folgendermaßen verändert werden.

```
1 GameObject cursorObject;
2 Sprite cursorHandSprite;
3 SpriteRenderer spriteRend;
4 void Start(){
5     // get SpriteRenderer from cursor
6     spriteRend = cursorObject.GetComponent<SpriteRenderer>();
7     // change cursor sprite
8     spriteRend.sprite = cursorHandSprite;
9 }
```

Der Standard-Cursor ermöglicht es dem Nutzer sich im Programm zurecht zu finden und zu bewegen. Der Drag-Cursor ist nur in der **World Scene**, der Hauptzene, vorhanden,

welcher dem Nutzer die Möglichkeit gibt die Erdkugel nach seinem Belieben zu drehen. Der letzte Cursor ist der sogenannte Hover-Cursor. Dieser dient als Hauptwerkzeug, um mit Objekten zu interagieren. Für die Verwendung des Standard- und Hover-Cursors muss der Nutzer die Hand, beim Bewegen, geöffnet lassen. Beim Drag-Cursor hingegen muss die Hand geschlossen bleiben. Um mit einem Objekt interagieren zu können, muss der Nutzer drei Sekunden lang über dem gewünschten Interaktionsobjekt, mit dem Hover-Cursor, verweilen.

Die Positionsdaten der Hand des Nutzers werden, wie bereits im Kapitel *Body Tracking* erklärt, in Echtzeit von der Kinect v2 erhalten. Die Bedienrechte sind unverändert geblieben (siehe Kapitel Bedienrechte). Das bedeutet, dass die Person mit dem geringsten Abstand zur Kinect die Bedienrechte über die Gestensteuerung erhält. Im Gegensatz zu den ursprünglichen Gesten muss beim Hand-Cursor jedoch nur abgefragt werden, ob sich eine der beiden Hände über der Anderen befindet, um ihn zu aktivieren.

```
1 void Update(){
2     ... // siehe Body Tracking
3     if(body.IsTracked){
4         //Joint variables
5         var handLeft = body.Joints[JointType.HandLeft];
6         var handRight = body.Joints[JointType.HandRight];
7
8         // Create Raycast if left Hand is higher than right hand
9         if(handLeft.Position.Y > handRight.Position.Y &&
10            ↳ body.HandLeftState == HandState.Open){
11             ...
12             // Create Raycast if right Hand is higher than left hand
13             }else if(handRight.Position.Y > handLeft.Position.Y &&
14                ↳ body.HandRightState == HandState.Open){
15                 ...
16             }
```

15 }

16 }

Der Code für das Bewegen mit dem Drag-Cursor ändert sich im Code ausschließlich der Handstate zu **HandState.Closed** (siehe Code Zeile 9 und 12).

3.5.4 Raycasting

Raycasting beschreibt den Prozess des Werfens eines unsichtbaren Strahls, in eine bestimmte Richtung. Falls dieser Strahl auf ein Objekt mit einer Kollisionskomponente trifft, gibt er einen *Hit* (Boolean) zurück. In dem erstellten Programm werden zwei verschiedene Arten von Raycasts verwendet.

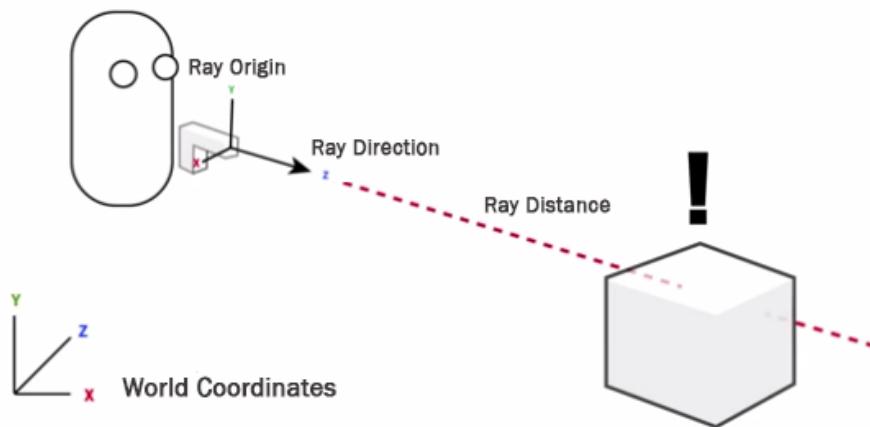


Abbildung 19: Raycasting

3.5.4.1 Physic Raycast

Der Physic Raycast ist für das Erkennen von 3D-Objekten zuständig. Um einen Physic Raycast zu generieren sind die folgenden Parameter notwendig:

- **Origin** = Ursprungspunkt des Raycasts in Welt-Koordinaten.
- **Direction** = die Strahlungsrichtung des Raycasts.
- **Length** = maximale Distanz, in welcher der Raycasts auf *Hits* überprüfen soll. Falls dieser Parameter nicht gesetzt ist, geht die Länge des Strahls ins Unendliche.
- **LayerMask** = ein Hilfsmittel, das ermöglicht, Kollisionsobjekte auf gewissen Layern (ger. Ebenen) der Scene, vom Raycast ignorieren zu lassen.

Der Code für die Erstellung eines Raycasts sieht folgendermaßen aus:

```
1 RaycastHit hit;
2 // Does the ray intersect any objects excluding the player layer
3 if (Physics.Raycast(transform.position,
4     ↪ transform.TransformDirection(Vector3.forward), out hit,
5     ↪ Mathf.Infinity, layerMask))
6 {
7     Debug.DrawRay(transform.position,
8         ↪ transform.TransformDirection(Vector3.forward) * hit.distance,
9         ↪ Color.yellow);
10    Debug.Log("Did Hit");
11 }
```

3.5.4.2 Graphic Raycast

Im Gegensatz zum Physic Raycast ist der der Graphic Raycast für das erfassen von UI-Elementen zuständig. Im Code finden sich offensichtliche Verschiedenheiten gegenüber des Physic Raycasts. Der Graphic Raycast generiert einen Strahl vom *PointerEventData*-Objekt aus in Richtung der Z-Achse. Alle getroffenen Objekte werden innerhalb einer Liste **results** gespeichert.

```
1 //Code to be place in a MonoBehaviour with a GraphicRaycaster
2     ↪ component
3
4 GraphicRaycaster gr = this.GetComponent<GraphicRaycaster>();
5
6 //Create the PointerEventData with null for the EventSystem
7 PointerEventData ped = new PointerEventData(null);
8
9 //Set required parameters, in this case, mouse position
10 ped.position = Input.mousePosition;
11
12 //Create list to receive all results
13 List<RaycastResult> results = new List<RaycastResult>();
14
15 //Raycast it
16 gr.Raycast(ped, results);
```

3.6 Interaktionen

Aufgrund der Tatsache, dass mit Hilfe des Raycastings, ersichtlich wird mit welchem Objekt der Nutzer interagieren möchte, können eine Reihe von Interaktionen Objekten zugewiesen und implementiert werden. Die folgenden Interaktionen und Funktionen sind in das Programm eingebaut.

3.6.1 Erdkugel-Rotation

Wie bereits im Abschnitt *Hand-Cursor* erwähnt, kann der Nutzer mit Hilfe des Drag-Cursor die Erdkugel nach Belieben drehen. Hierbei muss sowohl die vorherige Position des Cursors als auch die aktuelle Position abgefragt werden, um die Rotationsrichtung zu bestimmen.

```
1  if(...){
2      // difference vector
3      mPosDelta = cursor.transform.position - mPrevPos;
4      if(Vector3.Dot(transform.up, Vector3.up) >= 0){
5          earth.transform.Rotate(transform.up, -Vector3.Dot(mPosDelta,
6              → Camera.main.transform.right * rotSpeed), Space.World);
7      }else{
8          earth.transform.Rotate(transform.up, Vector3.Dot(mPosDelta,
9              → Camera.main.transform.right * rotSpeed), Space.World);
10     }
11     rend.sprite = mainCursor;
12 }
13 // new previous position = current position
14 mPrevPos = cursor.transform.position;
```

3.6.2 Kontinente

Alle interaktiven Kontinente werden zu Beginn/Start des Programms versteckt, indem der Alpha-Wert, welcher die Helligkeit bestimmt, des zugehörigen Materials auf 0 gesetzt wird. Alle Kontinente werden hierfür mit dem folgenden Skript ausgestattet. Der Alpha-Wert kann auf diese Weise verwendet werden, da zuvor, im Shader-Graph, ein Parameter für diesen Wert angelegt worden ist.

```
1 // Start is called before the first frame update
2 void Start(){
3     // Get rim object MeshRenderer
4     rend = this.GetComponent<MeshRenderer>().material;
5     // Set parent and child material alpha value to 0
6     alpha = 0.0f;
7     rend.SetFloat("_Alpha", alpha); //rend is the material of the
8         ↳ continent rim
9     mat.SetFloat("_Alpha", alpha); //mat is the material of the
10        ↳ continent
11 }
```

Wenn der Nutzer mit dem Cursor einen Kontinent trifft, also der Kontinent von dem Raycast getroffen wurde, wird dieser sichtbar gemacht, indem der Alpha-Wert des zugehörigen Materials wieder auf 1 gesetzt wird.

```
1 if(Physics.Raycast(_ray, out hit, rayLength, layermask)){
2     // Get MeshRenderer from hit continent and continent rim
3     main = hit.collider.gameObject.GetComponent<MeshRenderer>();
4     child =
5         ↳ main.transform.GetChild(0).gameObject.GetComponent<MeshRenderer>();
6     if(main.name == main.name + "Material"){
7         alpha = 1.0f;
8         main.material.SetFloat("_Alpha", alpha);
```

```

8         child.material.SetFloat("_Alpha", alpha);
9     }
10 }else{
11     ....
12 }
```

Des Weiteren wird bei aktivem User Interface eines Kontinentes, dieser ebenfalls aktiviert. Hierbei wird geprüft, welches Canvas-Element aktiv ist und den zugehörigen Kontinent zugeordnet. Dann kann der Alpha-Wert des Materials vom Kontinent auf 1 gesetzt werden. Dieses Skript wurde zunächst, für vereinfachten Handhabung, jedem Canvas-Element (enthält alle UI-Elemente) zugewiesen.

```

1 void Start(){
2     // continent name string array for assigning with canvas ui
3     conti = new string[] {"Africa", "Asia", "Australia", "Europa",
4     → "NorthAmerica", "SouthAmerica"};
5
6     // Get parent and child MeshRenderer for the belonging continent,
7     → if selected
8     foreach(string contiName in conti){
9         string coName = "Canvas" + contiName;
10        if(obj.name == coName){
11            continent = GameObject.Find(contiName);
12            parent = continent.GetComponent<MeshRenderer>();
13            child = parent.transform.GetChild(0).gameObject
14            .GetComponent<MeshRenderer>();
15        }
16    }
17
18 }
```

```

19 void Update(){
20     // if ui is active, show belonging continent and rotate it to
21     // front
22     if(obj.activeSelf == true){      //obj is the 3D object, which
23         // contains this script
24         alpha = 1.0f;
25         parent.material.SetFloat("_Alpha", alpha);
26         child.material.SetFloat("_Alpha", alpha);
27         if(rotation == true){
28             rotation = false;
29             RotateContinentToFront();
30         }
31     }

```

Zusätzlich wird während dieser Aktion auch der Kontinent zur Front rotiert. Diese Methode wird innerhalb der if-Schleife, in der oben gezeigten Update-Methode, eingefügt. Der **rotation**-Boolean verhindert das mehrfache Ausführen dieser Methode. Die **Quaternion.Euler**-Methode setzt die neue Rotationsausrichtung. Diese kann anschließend der **Quaternion.RotateTowards**-Methode übergeben werden, welche die aktuelle Ausrichtung, in Abhängigkeit von Rotationsgeschwindigkeit und Zeit, zur neu gesetzten Rotation transformiert.

```

1 void RotateContinentToFront(){
2     rotation = true;
3     // Get earth gameobject
4     GameObject earth = GameObject.Find("Earth");
5     if(obj.name == "Canvas" + conti[i]){
6         Quaternion targetRotation = Quaternion.Euler(-90.0f, 0.0f,
7             -70.0f);

```

```

7     earth.transform.rotation =
8         ↳ Quaternion.RotateTowards(earth.transform.rotation,
9             ↳ targetRotation, rotationSpeed * Time.deltaTime);
10    }else if(){
11        ...
12    }

```

3.6.3 User Interfaces

Die User Interfaces dienen zur Vermittlung von Informationen. In diesem Fall sollen diese über den Klimawandel informieren. Sie dienen jedoch auch zur Navigation durch die einzelnen Szenen des Programms. Das UI ist als Canvas-Objekt in der Szenen integriert. Zu beachten ist, dass das User Interface innerhalb des *World Spaces* (3-dimensionaler Raum) sein muss und nicht im *Screen Space* (2-dimensionaler Raum), da der Hand-Cursor nicht innerhalb des Screen Spaces interagieren kann. Nur Mausinteraktionen sind innerhalb des *Screen Spaces* möglich.

Beim Programmstart werden alle Canvas-Objekte, welche die UI-Elemente enthalten, zunächst auf inaktiv gestellt. Anschließend muss das User Interface dem entsprechenden Kontinent zugeordnet werden.

```

1 // Hide all Objects with Tag UI
2 uiCanvas = GameObject.FindGameObjectsWithTag("UI") as GameObject[];
3
4 foreach(GameObject canvas in uiCanvas){
5     if(canvas.layer == 5){
6         canvas.SetActive(false);
7     }
8 }

```

Wenn der Nutzer länger als drei Sekunden über einen Kontinent verbleibt, wird zunächst

überprüft, zu welcher UI der Kontinent gehört. Anschließend kann das UI-Element aktiviert werden. Hierfür ist ein **timer** innerhalb des Raycasts eingebaut, welcher sich jede Sekunde um eins erhöht. Wenn der timer die **waitTime** (in diesem Fall 2) überschreitet, dann wird das UI aufgerufen.

```

1 // show canvas ui, if user hovers longer than 2 sec on continents
2 void HoverGesture(MeshRenderer hitObject){
3     string newName = "Canvas" + hitObject.name;
4     foreach(GameObject canvas in uiCanvas){
5         // Show UI on hover on continent for over 2 seconds
6         if(canvas.name == newName && timer > waitTime){
7             canvas.SetActive(true);
8             exitButton.SetActive(true);
9             raycastBlocker.SetActive(true);
10            timer = 0.0f;
11        }
12    }
13 }
```

Um zu vermeiden, dass der Nutzer einen neuen Kontinent auswählt, während das UI aktiv ist, wurde ein Raycastblocker, in Form eines Rechtecks mit einer Kollisionskomponente zwischen der Erdkugel und dem UI erstellt, welcher dies verhindert.

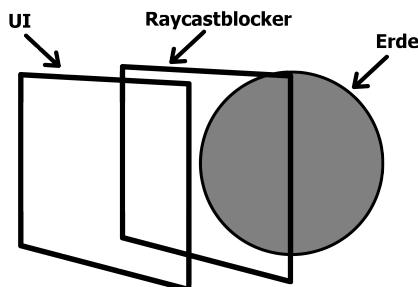


Abbildung 20: Raycastblocker

Um das User Interface zu schließen, muss der Benutzer drei Sekunden lang über dem Exit-Objekt hovern.

```
1 // Hide UI on hover on ExitButton for over 2 seconds
2 if(hitObject.name == "ExitButton" && timer > waitTime){
3     exitButton.SetActive(false);
4     raycastBlocker.SetActive(false);
5     timer = 0.0f;
6 }
```

3.6.4 Szenenwechsel

Für das Laden und Entladen von Szenen ist zunächst ein sogenannter SceneManager notwendig, welcher in dem zugehörige Skript verwendet werden muss. Da immer eine Szenen aktiv sein muss, ist der Code-Abschnitt hierfür zum Großteil immer gleich.

```
1 if(...){
2     SceneManager.UnloadSceneAsync("CurrentSceneName");
3     SceneManager.LoadSceneAsync("NewSceneName");
4 }
```

3.7 Aufteilung der Szenen

Das Projekt beeinhaltet eine Reihe von Szenen, in welcher der Nutzer sich Wissen über den Klimawandel aneignen und testen kann. Folgend werden die einzelnen Szenen vorgestellt und welchem Zweck sie erfüllen.

3.7.1 Intro

Die Intro-Scene ist der Eye-Catcher des Programms, welcher die Neugierde des Nutzers wecken soll, indem der Nutzer durch das Hinzubewegen und Hinwegbewegen auf den Bildschirm das Video, welches die Klimaerwärmung von 1880 bis 2018 zeigt, vor- und zurückspulen kann. Wenn der Nutzer eine gewisse Distanz überschreitet, kann er keine Änderungen mehr durch das Hinwegbewegen vom Bildschirm bewirken. Ab diesem **Point-Of-No-Return** wird das Video zum aktuellen Stand ausgeführt und soll durch einen schmelzenden/auflösenden Effekt, den Untergang der Erde simulieren.

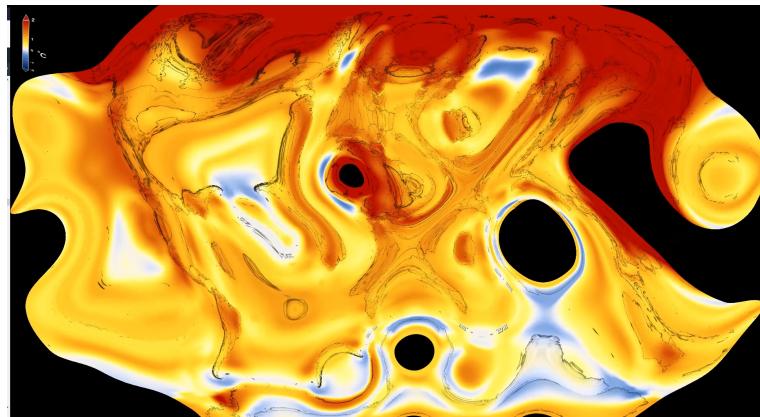


Abbildung 21: Intro Scene

3.7.2 Main Menu

Nachdem der Point-Of-No-Return im Intro erreicht wurde, wird nach dem Ende des Videos der Nutzer in das Main Menu weitergeleitet. Hier besitzt er die Möglichkeit auf die verschiedenen Szenen, mit Hilfe der Gestensteuerung, zuzugreifen.

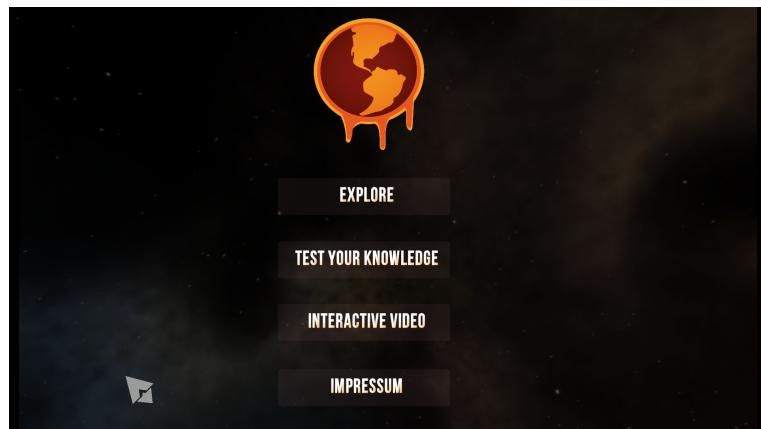


Abbildung 22: Hauptmenü

3.7.3 World Scene

Die Hauptszene des Programms bzw. Projektes ist die sogenannte World Scene, in welcher der Nutzer, unter Verwendung der Gestensteuerung, sich interaktiv über die Auswirkungen des Klimawandels informieren kann. Hierbei können auf sechs Kontinente, ohne der Antarktis, zugegriffen werden.



Abbildung 23: World Scene

3.7.4 Quiz

Das Quiz dient zum Testen des angeeigneten Wissens des Nutzers über den Klimawandel. Der Benutzer kann sich mit Hilfe der Gestensteuerung in der Szene bewegen. Das Quiz bietet Fragen mit vier *Multiple Choice*-Antworten pro Frage. Für jede richtig beantwortete Frage steigt der Punktestand des Nutzers. Nach dreimaliger falscher Antwort ist das Spiel mit einem Game Over vorbei.

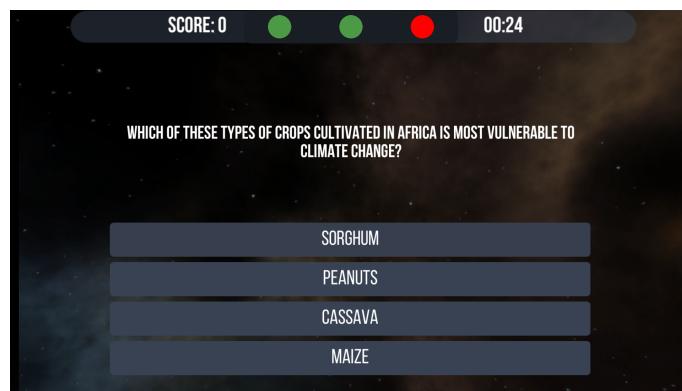


Abbildung 24: Quiz Game

3.7.5 Impressum

Das Impressum dient ausschließlich für die Darstellung der Entwickler. Es ist zum jetzigen Zeitpunkt noch nicht entschieden, ob das Impressum im Endprodukt enthalten sein soll, da das Projekt hauptsächlich zur interaktiven Informationsvermittlung über den Klimawandel dient.

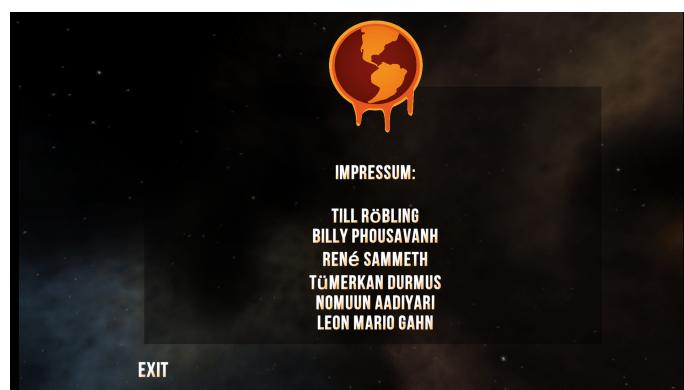


Abbildung 25: Impressum

4 Evaluation des interaktiven Schaukastens

Folgend wird dargestellt, wie die einzelnen Hauptfunktionalitäten des Projektes sich bewährt haben und welche Resultate daraus gezogen werden konnten.

4.1 Implementierung der 3D Modelle

Beim Import des Erdmodells von Blender in Unity kam es zu unerwünschten Problemen. Wenn ein 3D Modell als .fbx-Datei in Blender 2.8 exportiert und in Unity importiert wird, werden die Texturen nicht übernommen. Dieses Problem betrifft ausschließlich Blender 2.8. Da das 3D Modell in Blender 2.7 in kurzer Zeit nicht exakt nachzubauen ist, wurde vorübergehend für Präsentationszwecke ein Asset aus dem Unity Assetsstore verwendet, welches ein 3D Modell der Erde und eine Skybox enthält. Das ursprüngliche wurde in der zweiten Hälfte des Projektes jedoch durch die neuen 3D Modelle ersetzt, welche im Kapitel der 3D Modellierung vorgestellt werden.



Abbildung 26: 3D Erdkugel mit Skybox

4.2 Bedienrechte

Die Implementierung der Bedienrechte, so wie in der technischen Umsetzung beschrieben, funktioniert einwandfrei. Hierbei muss jedoch dem Nutzer besser vermittelt werden, dass dieser auch der aktive Benutzer ist und die Rechte für die Gestensteuerung besitzt. Als Folge der Covid-19 Situation konnte dies leider nicht getestet werden und wurde aus diesem Grund auf dem aktuellen Stand beibehalten.

4.3 Gestensteuerung

4.3.1 Gesten

Bei den Gesten, welche zur Projektzwischenpräsentation eingebaut wurden, ist aufgefallen, dass diese zum einen nicht vollständig ausgeführt werden müssen, um beispielsweise die Rotation der Erde auszuführen. Des Weiteren wurde des Öfteren bemängelt, dass die verwendeten Gesten nur ungenau funktionieren und über eine längere Zeitspanne anstrengend sind. Dies waren Gründe, weshalb zu einem Hand-Cursor gewechselt wurde.

4.3.2 Hand-Cursor

Beim Hand-Cursor konnte aufgrund der Covid-19 Zustände keine Umfassende Usability-Prüfung ausgeführt werden. Aus diesem Grund ist bislang unbekannt, ob Nutzer die Bedienung des Hand-Cursors verständlich finden. Des Weiteren gibt es im aktuellen Zustand des Programms noch das Problem, dass beim Hovern über einen Kontinent, das Standard-Cursor-Icon sich nicht zu dem Hover-Cursor-Icon ändert. Dies könnte leider Auswirkungen auf die User Experience haben, da dies dem Nutzer das Feedback geben soll, dass er mit einem Objekt interagieren kann. Durch die begrenzte Verfügbarkeit der Kinect v2 Hardware, musste für das Testen von einzelnen Funktionen ein Hand-Cursor erstellt werden, welcher mit der Computermaus interagiert. Auf diese Weise wurden unter Anderen die Raycast- und die Cursoränderungen erarbeitet.

Aufgrund von zeitlichen Engpässen, konnte im *Main Menu* der Hand-Cursor nur problem-

behaftet und im Quiz sowie Impressum nicht implementiert werden. Dies bedeutet für den aktuellen Zustand, dass der Nutzer sich im Quiz und Impressum nicht mit dem Hand-Cursors bewegen kann und dies nur mit einer Computermaus funktioniert. Des Weiteren gibt es im Hauptmenü Probleme mit der sogenannten *Hitbox*, welche den Aktionsraum für den Raycast-Hit darstellt. Hierbei wird der Impressum-Button etwas über der UI ausgelöst.

4.4 Interoperabilität

Unity besitzt bereits den Vorteil, dass es plattformübergreifend, darunter Windows-, Linux- und Apple-Systemen, als Standalone-Version verwendet werden kann. Problematisch ist hierbei eher die Kinect v2, da diese hauptsächlich nur mit Windows kompatibel ist. Es gibt zwar Pakete für die Installation auf Apple und Linux Geräten, diese konnten jedoch im Zeitrahmen des Projektes noch nicht getestet werden. Eine funktionierende Alternativmethode für diese Pakete ist die Verwendung einer *Virtual Machine*, auf der ein Windows Betriebssystem läuft.

4.5 Wiederverwendbarkeit

In der ursprünglichen Aufgabenstellung ist die Wiederverwendbarkeit aufgeführt. Aufgrund der Tatsache, dass beim Testen der Kinect v2 mit einer WPF-Applikation nicht behebbare Komplikationen aufgetaucht sind, konnte dies nicht der Anforderung für die Wiederverwendbarkeit gerecht werden. Deshalb wurde der Entschluss gefasst, das Programm als Individual-System aufzufassen und nur den *Showroom* wiederverwendbar zu machen.

5 Soll- / Ist-Stand

Der **Soll-Stand** beinhaltet die folgenden Anforderungen:

- Die Animation/Videotextur, soll durch Hinzubewegen auf den Bildschirm manipuliert werden (vor- oder zurückspulen).
- Die Animationsmanipulation soll mit einem Synthesizer verbunden werden, um den Ton auf die Animation abzustimmen.
- Es soll immer nur eine Person die Rechte für die Gestensteuerung erhalten.
- Mit Hilfe der Gestensteuerung soll die Erde rotiert werden können, sowie zusätzliche Informationen aufgerufen werden.
- Der Nutzer soll sich mit Hilfe der Gestensteuerung zwischen den Szenen bewegen können.
- Für den Zugriff auf zusätzliche Informationen werden klickbare Flächen, in Form von Länder-/Kontinentengrenzen, über das 3D Modell der Erde gelegt.
- Um die zusätzlichen Informationen aufzurufen, soll ein Hand-Cursor implementiert werden und eine Klickbewegung hinzugefügt werden.
- Implementierung eines Quizzes, mit welchem der Nutzer sein angeeignetes Wissen testen kann.
- Der Nutzer soll mit Hilfe eines Hauptmenüs zwischen den Szenen navigieren können.
- Der Showroom soll durch ein klapp- und schiebbares Aufbausystem, einfach auf Messen aufgebaut werden können.
- LEDs sollen, in Abhängigkeit von der Distanz des Nutzers zum Bildschirm, in verschiedenen Farben aufleuchten, um eine Stimmung zu kreieren, die den Nutzer in die Anwendung des Schaukastens eintauchen lässt.

Im **Ist-Stand** sind folgende Funktionen und Methoden implementiert worden:

- Das Intro, bestehend aus dem Video der Klimaerwärmung, kann durch Hinzu- und Hinwegbewegung vor- und zurückgespult werden.
- Das Intro läuft zu Ende wenn der Nutzer eine gewisse Distanz überschritten hat.
- Die Rechte für die Gesteuerung erhält die Person mit dem geringsten Abstand zur Kinect.
- Ein Hand-Cursor wird als Gesteuerung verwendet.
- Funktionsfähigkeit für Links- und Rechtshänder ist integriert.
- Der Hand-Cursor ist aktuell nur im Hauptmenü und in der Hauptszene implementiert.
- Der Hand-Cursor wird durch das Bewegen mit geöffneter Hand aktiviert.
- Die Erde kann durch den Hand-Cursor(Drag) rotiert werden, indem der Nutzer mit geschlossener Hand sich in eine gewünschte Richtung bewegt.
- Durch das Hovern über einen Kontinenten wird dieser gehighlightet.
- Interaktionen werden durch drei sekündiges Hovern, über einem Interaktionsobjekt, ausgeführt.
- Durch Interaktion mit einem Kontinent kann der Nutzer sich detaillierte Informationen anzeigen lassen.
- Der ausgewählte Kontinent wird bei der Interaktion zur Frontansicht rotiert.
- Änderung des Hand-Cursor-Icons, zur Mitteilung von Informationsmöglichkeiten, funktioniert nur für das Drag-Cursor-Icon in der Hauptszene.
- Der Nutzer kann im Hauptmenü Szenen auswählen.
- Das Quiz kann vom Hauptmenü ausgewählt werden, bei welchem der Nutzer Fragen beantworten kann. Bei jeder richtig beantworteten Frage steigt der Punktestand. Wenn drei Fragen falsch beantwortet wurden, dann ist dies ein "Game Over".

6 Fazit und Ausblick

6.1 Fazit des erarbeiteten Konzepts

Der Ansatz des Body Tracking mit Hilfe der Microsoft Kinect v2 erfolgte aufgrund der bereits vorgegebenen Funktionen. Die Plattformunabhängigkeit wurde zwar durch die Verwendung einer Virtual Machine ermöglicht, ist jedoch keine optimale Lösung. Das Ersetzen der unpräzisen Gestensteuerung durch einen Hand-Cursor hat wesentlich mehr intuitive Vorteile für den Nutzer, konnte jedoch noch nicht einem umfangreichen Usability Test unterzogen werden, aufgrund der Covid-19 Situation, bei welchem noch nicht bekannte Probleme auftauchen könnten. Des Weiteren konnte aus zeitlichen Gründen die Gestensteuerung noch nicht in alle Szenen eingebaut werden. Neben den noch nicht behobenen Fehlern, wie die Cursor-Icon-Änderung für Interaktionen und fehlende Interaktionsobjekte, um zurück auf das Hauptmenü zu gelangen, funktionieren alle Methoden wie angefordert. Insgesamt kann gesagt werden, dass nach Behebung der vereinzelten Fehler, der Nutzer eine gute interaktive Erfahrung mit dem Klimawandel erhält.

6.2 Möglichkeiten der Verbesserung

Im Bereich der Gestensteuerung kann es noch Potenzial zur Verbesserung geben, welches sich anhand von Usability Tests herausstellen kann. Das Intro sollte, nachdem der Nutzer die Reichweite der Kinect verlassen hat oder nach einer gewissen Distanz, wieder abgespielt werden, um die Neugierde von neuen Nutzern zu erlangen. Des Weiteren könnte man noch mehr Details im User Interface unterbringen, indem diese auch interaktiv gestaltet wird. Beim Quiz könnte eine Highscore-Liste als kompetitiver Faktor eingebaut werden. Dadurch würden Nutzer sich vermutlich intensiver mit dem Thema beschäftigen. Dies müsste jedoch auch erst durch einen Usability Test evaluiert werden.

6.3 Ausblick und Weiterentwicklung

Im Hinblick auf die Erfahrungen von der Projektpräsentation sowie durch dessen Rückmeldung, lässt sich feststellen, dass das Projekt sich für die zukünftige Weiterentwicklung bewährt. Aufgrund der Tatsache, dass etwa nur jährlich die Daten für den Klimawandel aktualisiert werden, könnten die verwendeten Informationen jedes Jahr, ohne großen Aufwand, erneuert werden. Im aktullen Zustand ist das Projekt gut für die Anwendung auf Messen einsetzbar. Wenn der Schaukasten für andere Zwecke als die aktuelle Ausarbeitung verwendet werden soll, müsste jedoch ein anderer Ansatz gewählt werden, bei dem die Grundstruktur für das Layout und Gestensteuerung gegeben sind.

7 Literaturverzeichnis

Definitionen:

- (1) Vertex/Vertices, Aufruf 19.07.2020, <https://docs.blender.org/manual/en/latest/modeling/meshes/structure.html>
- (2) Mesh, Aufruf 19.07.2020, https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/What_is_a_Mesh%3F
- (3) MipMap, Aufruf 19.07.2020, https://de.wikipedia.org/wiki/Mip_Mapping
- (4) SDK, Aufruf 19.07.2020, <https://www.dev-insider.de/was-ist-ein-sdk-a-730921/>
- (5) Virtual Machine, Aufruf 19.07.2020, <https://www.techopedia.com/definition/4805/virtual-machine-vm>
- (6) Virtual Reality, Aufruf 19.07.2020, https://de.wikipedia.org/wiki/Erweiterte_Realit%C3%A4t
- (7) Material, Aufruf 19.07.2020, <https://docs.unity3d.com/560/Documentation/Manual/Shaders.html>
- (8) Shader, Aufruf 19.07.2020, <https://docs.unity3d.com/560/Documentation/Manual/Shaders.html>
- (9) Textures, Aufruf 19.07.2020, <https://docs.unity3d.com/560/Documentation/Manual/Shaders.html>

Blender:

- (1) 3D Modelling, Aufruf 19.07.2020, <https://blender.stackexchange.com/questions/52904/wrap-curve-around-sphere>

KINECT:

- (1) Kinect v2 Hardware, Aufruf 10.01.2020, <http://michaelkipp.de/interaction/kinect.html>
- (2) Kinect v2 Joint Koordinaten, Aufruf 06.11.2019, <https://medium.com/@lisajamhoury/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16>
- (3) WPF-Applikation, Aufruf 06.11.2019, <https://www.youtube.com/watch?v=GPjSOSBtHwY>

- (4) WPF-Applikation, Aufruf 14.11.2019, <http://kinect.github.io/tutorial/lab01/index.html>
- (5) Body Tracking, Aufruf 14.11.2019, <http://kinect.github.io/tutorial/lab06/index.html>
- (6) Bedienrechte, Aufruf 23.11.2019, <https://social.msdn.microsoft.com/Forums/en-US/57b44f36-130f-415b-9c6e-0c68945013f5/choose-which-body-to-track?forum=kinectv2sdk>
- (7) Visual Gesture Builder, Aufruf 18.12.2019, <https://channel9.msdn.com/Blogs/AdamTuliper/Custom-Gestures-with-the-Kinect-v2-for-Windows>
- (8) Visual Gesture Builder, Aufruf 08.01.2020, <https://peted.azurewebsites.net/visual-gesture-builder-kinect-4-windows-v2/>
- (9) Visual Gesture Builder, Aufruf 08.01.2020, <https://peted.azurewebsites.net/visual-gesture-builder-kinect-4-windows-v2-code/>

UNITY:

- (1) Unity Wikipedia, Aufruf 17.02.2020, [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))
- (2) Unity Tutorials, Aufruf 17.02.2020, <https://learn.unity.com>
- (3) Raycast, Aufruf 04.05.2020,
<https://gamedevacademy.org/learn-and-understand-raycasting-in-unity3d/>
- (4) Graphic Raycast, Aufruf 18.06.2020,
<https://docs.unity3d.com/2017.3/Documentation/ScriptReference/UI.GraphicRaycaster.Raycast.html>
- (5) Graphic Raycast, Aufruf 18.06.2020, <https://answers.unity.com/questions/844158/how-do-you-perform-a-graphic-raycast.html>
- (6) Physic Raycast, Aufruf 04.05.2020, <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>
- (7) 3D Modelling-Post Processing, Aufruf 22.04.2020, <https://docs.unity3d.com/Manual/PostProcessingOverview.html>
- (8) 3D Modelling-Post Processing, Aufruf 22.04.2020, <https://docs.unity3d.com/Manual/PostProcessing-Bloom.html>

- (9) **3D Modelling-LWRP**, Aufruf 22.04.2020, <https://docs.unity3d.com/Packages/com.unity.render-pipelines.lightweight@5.10/manual/index.html>
- (10) **3D Modelling-Shader**, Aufruf 25.04.2020, <https://www.youtube.com/watch?v=6AKR4b4C5jo>
- (11) **Erdkugel-Rotation-Quaternion**, Aufruf 29.06.2020, <https://docs.unity3d.com/ScriptReference/Quaternion.html>
- (12) **Interaktionen-Finde Objekt mit Tag**, Aufruf 14.06.2020, <https://docs.unity3d.com/ScriptReference/GameObject.FindGameObjectsWithTag.html>
- (13) **Interaktionen-Finde Objekt auf Layer**, Aufruf 14.06.2020, <https://answers.unity.com/questions/968709/find-object-by-layer-unity-5-split-screen-camera-f.html>
- (14) **Quiz**, Aufruf 06.07.2020, <https://github.com/swapnilrane24/Quiz>

ASSETS:

- (1) **extOSC Plugin**, <https://assetstore.unity.com/packages/tools/input-management/extosc-open-sound-control-72005>
- (2) **3D Erdkugel mit Skybox**, <https://assetstore.unity.com/packages/vfx/shaders/earth-rendering-free-54914>
- (3) **NASA Video Temperaturdaten**, https://climate.nasa.gov/climate_resources/139/video-global-warming-from-1880-to-2019/

BILDQUELLEN:

- Technische Hochschule Nürnberg Georg Simon Ohm Logo,
https://www.th-nuernberg.de/fileadmin/abteilungen/kom/kom_bilder/Logos_Presse/RGB/TH_Nuernberg_RGB.png
- (Abbildung 1) Konzeptidee, Erstellung in Projektgruppe
- (Abbildung 2) Foto und Skizzierung des physikalischen Prototyps
- (Abbildung 3) Kinect v2 Hardware, (Slide 5), [https://pt.slideshare.net/ DavidLpez60/kinect-v2-descripcin?smtNoRedir=1](https://pt.slideshare.net/DavidLpez60/kinect-v2-descripcin?smtNoRedir=1)
- (Abbildung 4) Time-Of-Flight Prinzip,

https://www.stemmer-imaging.com/media/cache/default_image_1100/uploads/cameras/sis/gl/glossary-3D-time-of-flight-cameras-1-en.jpg

(Abbildung 5) Screenshot des finalen Kontinentdesigns in Unity

(Abbildung 6) Screenshot Blender Simple Deform Z-Achse 180 Grad

(Abbildung 7) Screenshot Blender Simple Deform X-Achse 180 Grad

(Abbildung 8) Screenshot Blender Endstand des 3D Erd- und Kontinentmodell

(Abbildung 9) Skybox (aus Assets)

(Abbildung 10) Screenshot Shader Graph in Unity

(Abbildung 11) Screenshot Post Processing in Unity

(Abbildung 12) Screenshot Unity extOSC

(Abbildung 13) KinectView, <https://www.imaginativeuniversal.com/blog/2015/03/>

(Abbildung 14) Screenshot Kinect Studio

(Abbildung 15) Screenshot Visual Gesture Builder

(Abbildung 16) Hand States

(Abbildung 17) Gesten

(Abbildung 16+17) Hand Zoom Geste, https://www.flaticon.com/free-icon/zoom_2142562

(Abbildung 16+17) Hand Closed Geste, https://www.flaticon.com/free-icon/hand-closed_482657?term=hand

(Abbildung 16+17) Hand Open Geste, <https://www.flaticon.com/search?word=hand>

(Abbildung 16+17) Hand Lasso Geste, https://www.flaticon.com/free-icon/rock-and-roll_487174?term=hand

(Abbildung 18) Hand-Cursor

(Abbildung 19) Raycasting,
<https://i.pinimg.com/originals/e2/50/ea/e250ea279dae7803e05449c52dfab611.png>

(Abbildung 20) Grafik zur Darstellung des Raycastblockers

(Abbildung 21) Screenshot Intro in Unity

(Abbildung 22) Screenshot Main Menu in Unity

(Abbildung 23) Screenshot World Scenen in Unity

(Abbildung 24) Screenshot Quiz in Unity

(Abbildung 25) Screenshot Impressum in Unity

(Abbildung 26) Screenshot Asset 3D Erdkugel mit Skybox (siehe Assets)

OSC:

(1) OSC official, Aufruf 20.02.2020, <http://opensoundcontrol.org/introduction-osc>

(2) extOSC Plugin Repository, Aufruf 10.01.2020, <https://github.com/Iam1337/extOSC>

QUELLCODE:

(1) Git-Repository, <https://github.com/rsamQ/interactive-showroom/devlopment>