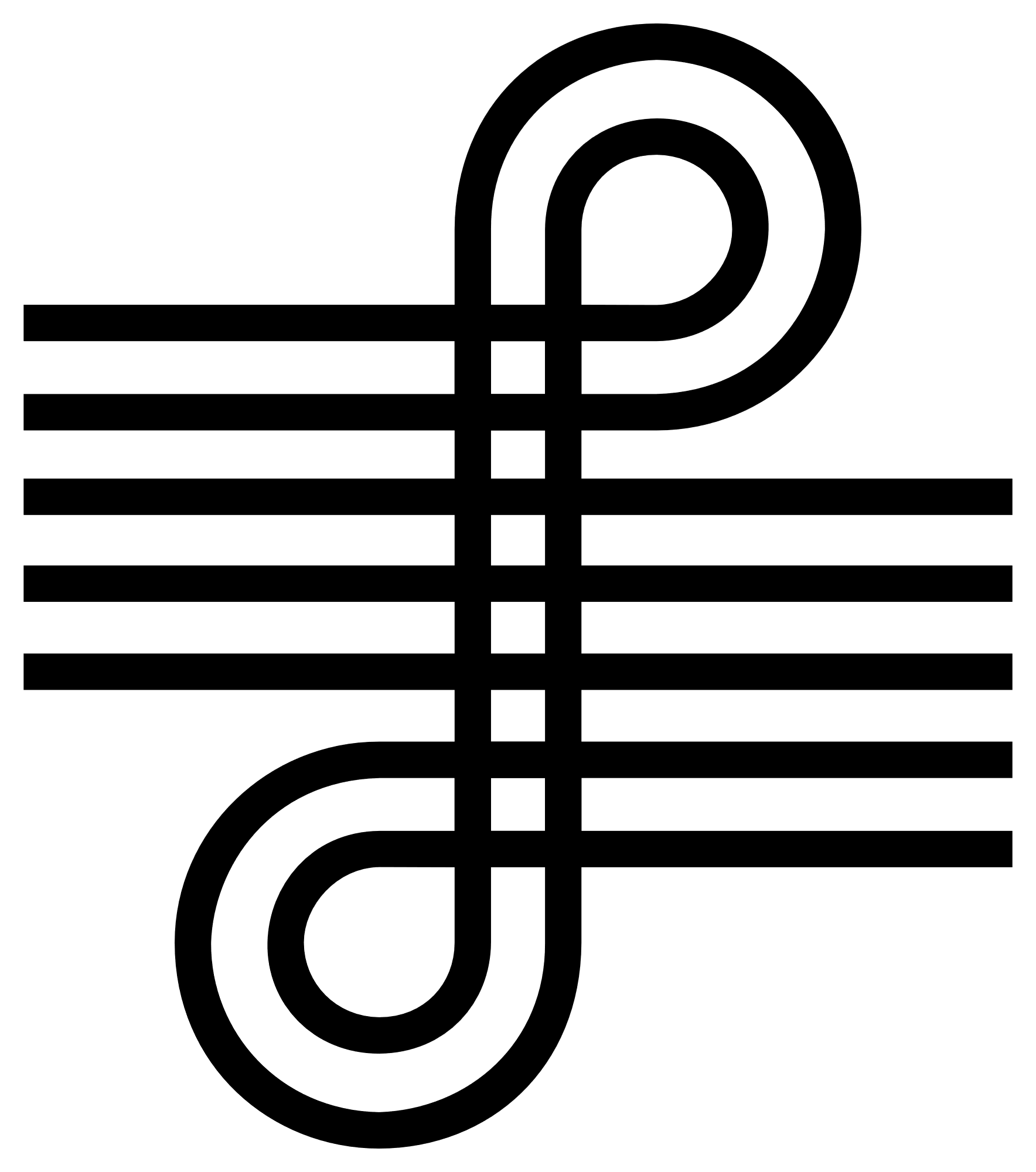
Hochschule für Musik Karlsruhe

IMWI - Institut für Musikinformatik und Musikwissenschaft

Bachelorarbeit zum Thema:

**Interaktion mit Audio-Daten in VR mithilfe von Unity, Head-Mounted-Display und Motion-Controllern**

Zur Erlangung des Grades Bachelor of Arts

Vorgelegt von:

Manuel-Philippe Hergenröder, 12085

[mail@manuelhergenroeder.de](mailto:mail@manuelhergenroeder.de)

Betreuender Dozent: Zweitkorrektor:

Prof. Dr. Damon T. Lee Patrick Borgeat

Studiengang:

Musikinformatik (Hauptfach) / Musikwissenschaften (Nebenfach)

Abgabe:

xx.07.2020

Inhaltsverzeichnis

[2 Einleitung 1](#_Toc45032210)

[3 Grundlagen 1](#_Toc45032211)

[3.1 Virtual Reality 1](#_Toc45032212)

[3.1.1 Definition und Historie 1](#_Toc45032213)

[3.1.2 Anforderungen an VR-Software und -Hardware 3](#_Toc45032214)

[3.2 Diskrete Fourier-Transformation 6](#_Toc45032215)

[4 Praktische Implementation „VrAudioSandbox“ 7](#_Toc45032216)

[4.1 Repository und Systemvoraussetzungen 7](#_Toc45032217)

[4.2 Architektur und externe Bibliotheken 7](#_Toc45032218)

[4.2.1 Unity als Laufzeitumgebung 8](#_Toc45032219)

[4.2.2 SteamVR / OpenVR 9](#_Toc45032220)

[4.2.3 NAudio Bibliothek 9](#_Toc45032221)

[4.2.4 DFT mit FFTW3 10](#_Toc45032222)

[4.2.5 FFTWSharp 10](#_Toc45032223)

[4.2.6 CurvedUI 11](#_Toc45032224)

[4.3 Implementation in Unity 11](#_Toc45032225)

[4.3.1 Import und Fast-Fourier-Transformation 11](#_Toc45032226)

[4.3.2 Visualisierung der Spektrum-Daten 15](#_Toc45032227)

[4.3.3 Manipulation der Spektrum-Meshes 16](#_Toc45032228)

[4.3.4 Inverse Fast-Fourier-Transformation und Wiedergabe 19](#_Toc45032229)

[4.3.5 User Interface und Steuerung 23](#_Toc45032230)

[5 Kritische Betrachtung der Ergebnisse 23](#_Toc45032231)

[6 Fazit 24](#_Toc45032232)

[7 Literatur III](#_Toc45032233)

[8 Sonstige Quellen IV](#_Toc45032234)

[9 Abbildungsverzeichnis VI](#_Toc45032235)

[10 Code-Listing-Verzeichnis VII](#_Toc45032236)

[11 Eidesstattliche Erklärung VIII](#_Toc45032237)

[12 Anhang – Programmcode IX](#_Toc45032238)

# Einleitung

*Virtual Reality* (VR) bietet das Potential unserem Geist direkten und erlebbaren Zugriff auf digitale Daten – losgelöst von der Umgebung in der wir uns aufhalten – zu geben. In dieser Arbeit sollen die Möglichkeiten der Interaktion mit Audio-Daten im virtuellen Raum am Beispiel der Darstellung und Manipulation von FFT-Audio-Daten mithilfe eines Head-Mounted-Display (engl. für „Am-Kopf-befestigter-Bildschirm“) – oder kurz HMD - und Motion-Tracking-Controllern untersucht werden.[[1]](#footnote-2) Gegenstand der Arbeit ist die praktische Implementation dieses Konzepts. Dabei spielen technische Herausforderungen und praktische Implementationsdetails ebenso eine Rolle wie die Entwicklung und Bewertung des Bedienkonzeptes. […]

# Grundlagen

## Virtual Reality

### Definition und Historie

Der Begriff Virtual Reality impliziert, dass eine Realität mithilfe einer Simulation geschaffen wird, welche für den Nutzer als real wahrgenommen wird. Bereits 1965 beschrieb der Computergrafikpionier Ivan E. Sutherland mit dem „Ultimate Display“ einen Raum, in dem die Materie durch den Computer gesteuert wird. Auf einen Stuhl in diesem Raum könne man sich setzen, eine Pistolenkugel wäre tödlich.[[2]](#footnote-3) Dieses Ziel der kompletten Immersion konnte bisher nicht umgesetzt werden. Virtuelle Realität muss aber nicht der „echten Realität“ komplett entsprechen, um wirksam zu sein: „Sobald das Nervensystem genügend Anhaltspunkte hat, um die virtuelle Welt als Grundlage für seine Erwartungen zu akzeptieren, kann sich VR real anfühlen […]“[[3]](#footnote-4), schreibt Jaron Lanier. Das heißt dementsprechend, dass auch eine „Zeichentrick-Welt“ immersiv sein und unser Gehirn diese als Realität annehmen kann. Entscheidend ist also nicht eine möglichst photo-realistische Darstellung oder ein realistisches Szenario innerhalb der virtuellen Welt, sondern eine Stimulation unserer Sinne mithilfe von Technologie, so dass unser Gehirn die Schwelle übertreten kann für eine gewisse Zeit an die virtuelle Welt zu glauben – anstelle der echten. Virtuelle Realität könnte man daher eher als eine Art Bewusstseinserfahrung beschreiben, die geprägt ist von dem Aspekt der Immersion und der Authentizität. Dabei spielt es keine Rolle, welche Technologien genutzt werden, um diesen Zustand zu erreichen.

Die Geschichte der Entwicklung dieser Technologien geht weit zurück. So stellen schon Panoramabilder des 19. Jahrhunderts, deren Zweck es war das Sichtfeld des Betrachters auszufüllen, erste Techniken der Immersion dar. Auch die Entdeckung der Stereoskopie durch Charles Wheatstone[[4]](#footnote-5), bei der der Eindruck des räumlichen Sehens durch zwei blickwinkelverschiedene Bilder hervorgerufen werden kann, ist eine noch heute relevante Technik. Vom heutigem Standpunkt aus, bei dem virtuelle Realität hauptsächlich mit einer mithilfe des Computers simulierten Welt assoziiert wird, spielte die Entwicklung des sogenannten Head-Mounted-Display eine wichtige Rolle. Ein HMD besteht aus einem oder mehreren Displays und stellt am Kopf befestigt eine Schnittstelle zwischen der visuellen Wahrnehmung des Menschen und der Grafikausgabe eines Computers her. Ivan E. Sutherland hat 1968 ein solches System mit dem Namen „The Sword of Damocles“ mithilfe seiner Studenten realisiert.[[5]](#footnote-6) Dies legte die Grundlagen zum einen für die stereoskopische Darstellung des im Computer erzeugten Raums in Vektorgrafik, zum anderen für die notwendige schnellen Synchronisierung der angezeigten Bilder mit den Kopfbewegungen des Benutzers. Ein Sensor mit mechanischem Arm hat zunächst die Kopfposition ermittelt. Später wurde dies für bessere Bewegungsfreiheit mithilfe von Ultraschall-Emittern und -Empfängern umgesetzt.[[6]](#footnote-7) Dieses sogenannte Head-Tracking und die zeitnahe Abstimmung der angezeigten Bilder an Kopfposition und -bewegungen ist auch heute noch ein wichtiger Aspekt bei HMDs. Einerseits um das Präsenzgefühl überhaupt erst zu erzeugen zwischen virtueller Welt und dem physischen Körperempfinden, andererseits um Übelkeit (sogenannte Motion-Sickness), die in unserem Gehirn entsteht, wenn widersprüchliche Reize – in diesem Fall die visuellen Informationen unseres Sehapparates und die körperliche Wahrnehmung bezüglich der Bewegung und Stellung des Kopfes verarbeitet werden – beim Benutzer vorzubeugen.

<NASA Datenhandschuh, VR Hype 90er, 2. Welle Oculus Rift/Vive, Nintendo Virtual Boy, Gaming Industry, Professionelle Anwendungen Architektur, CAD, Ergonomie>

Prinzipiell können alle erdenklichen Sinne durch technische Einrichtungen stimuliert werden – z.B. auch taktile oder olfaktorische. Beispiele sind Aktoren in Ganzkörperanzügen; Geräte, die Duftstoffe absondern; Wind- oder Wärmegeneratoren etc. Sofern diese zur Immersion und dem Präsenzgefühl beitragen, können sie ein Teil des VR-Konzeptes sein.

Ein mit der Virtual Reality verwandtes Feld ist die sogenannte Augmented Reality bei der die echte Realität mit zusätzlichen Informationen angereichert wird. Im Gegensatz zur VR werden reales Umfeld und virtuelle Bestandteile vom Computer vermischt – z.B. durch ein transparentes Display oder mithilfe eines eingeblendeten Live-Kamera-Bildes. Bei einer starken Interaktion zwischen Bestandteilen realer und virtueller Welt wird zudem auch der Begriff Mixed-Reality verwendet. AR ist dementsprechend nicht Gegenstand dieser Arbeit.

### Anforderungen an VR-Software und -Hardware

Gängige etablierte Bedienparadigma, die auf der der Computer- und Medientechnik basieren und sich über die letzten Jahrzehnte entwickelt und etabliert haben, sind für VR-Anwendungen nicht in Gänze übertragbar. Insbesondere die Gestaltung in Form einer zweidimensionalen Bedienoberfläche (User Interface), welche auf einem zweidimensionalen Display dargestellt wird und mit Maus, Tastatur oder via Touch bedient wird, lässt sich nur mit Einschränkungen bzgl. der Ergonomie und Nutzerfreundlichkeit bei Virtual Reality anwenden. Insbesondere die damit einhergehenden verschachtelten Menüstrukturen, die viel Feinmotorik zur Interaktion benötigen, sind nicht für die Bedienung mit Motion-Controllern ausgelegt. Motion-Tracking ist in der Regel Teil eines Virtual-Reality-Konzepts und wird demzufolge häufig in Form eines Gesamtkonzeptes mit HMDs zusammen entwickelt und verkauft. Dies macht deutlich, dass neue Bedienkonzepte – zugeschnitten auf die Bedienung durch Handbewegungen und -gesten – eine große Rolle spielen. Ein Beispiel zur Verdeutlichung sind die in vielen Computerspielen – aber auch in Software fernab des Unterhaltungssektors – vorhandenen HUD-UI-Elemente. HUD steht für Head-Up-Display und beschreibt das Platzieren von informativen Elementen oder Elementen zur Steuerung im sogenannten Screen-Space – d.h. überlagernd ohne Bezug zur dreidimensional dargestellten Szene. Durch die stereoskopische Darstellung in VR kann dieser fehlende Bezug zur Geometrie des virtuellen Raumes irritierend wirken. Auch dass der Benutzer sich im Raum orientieren muss und seine Blickrichtung dabei frei wählbar ist, kann dazu führen, dass UI-Elemente im Sichtbereich stören können. Das VR-Computerspiel „Half-Life Alyx“ der Firma Valve zeigt den Ansatz traditionelle HUD-Elemente durch virtuell-physische Elemente zu ersetzen. Gesundheitszustand des Spielers und Munitionsanzeige sind Teil der Spielwelt und können durch entsprechenden Blick des Spielers bei Bedarf erfasst werden ohne zu stören (siehe Abbildung 1 unten; oben die traditionell genutzte Methode durch HUD-UI-Elemente).

Diese virtuelle Physikalität lässt sich aber auch auf die Steuerung in VR übertragen. Etablierte Eingabegeräte wie Maus, Tastatur oder Gamepad stellen eine Abstraktionsschicht dar, die der Immersion abträglich ist. Das Bewegen einer Spielfigur mit Richtungstasten bspw. erfordert eine Abstraktion und entspricht nicht intuitiv unserer Vorstellung des Gehens. Hand-Controller stellen dabei momentan noch einen Kompromiss dar, deuten aber schon das Wegfallen dieser Abstraktionsebene an, indem Bewegungen des Armes und einzelner Finger auf die virtuelle Welt übertragen werden und für die Interaktion mit virtuellen Objekten genutzt werden können. Sowohl die Eingabe, als auch die Ausgabe von VR-Anwendungen sind also besonders immersiv und intuitiv, wenn sie konzeptionell nach dieser Physikalität gestaltet sind.

Abbildung 1 – Vergleich HUD-basierte Darstellung (oben) von Gesundheit und Munition mit im virtuellen Raum als physische Objekte manifestierter Darstellung (unten)

Einzelquellen (von oben n. unten): Steam Workshop Half-Life: Alyx [HUD], Screenshot von „Alex“ – <https://steamuserimages-a.akamaihd.net/ugc/1016065725020200218/B47282DABAFC22E8A6C5770FE46BCA8FCFFBF718/>; IGDB Half-Life: Alyx Press kit – <https://images.igdb.com/igdb/image/upload/t_original/sc7dad.png>; Screenshot von SuperQGS, reddit –<https://i.redd.it/yojg6f875rg41.jpg> – alle Screenshots © 2019 Valve Corporation, letzter Abruf 17.06.2020

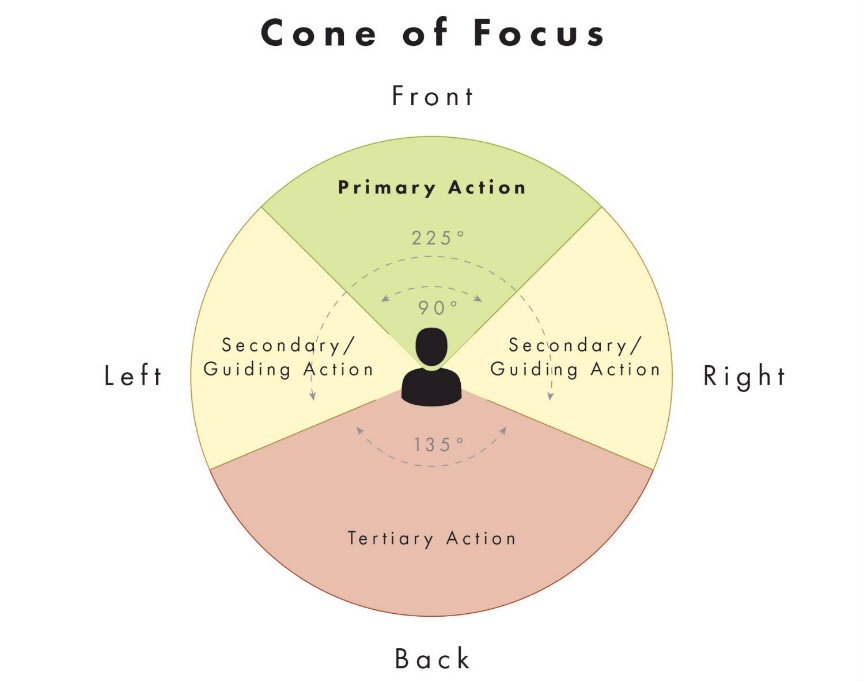
Dadurch, dass der Benutzer seine Blickrichtung jederzeit frei wählen kann bzw. muss, ergeben sich Anforderungen an die Steuerung der Aufmerksamkeit durch das Design der VR-Applikation. Das Cone-Of-Focus-Modell (siehe Abbildung 2) teilt dabei das Umfeld des Spielers in verschiedene Zonen ein. Alle wichtigen Ereignisse sollten sich im Zentrum der Aufmerksamkeit und Blickrichtung des Benutzers abspielen. Sollte es gewünscht sein, dass der Nutzer seine Aufmerksamkeit auf eine andere Zone lenkt, bieten sich visuelle Anhaltspunkte über den primären und sekundären peripheren Sichtbereich an, um den Nutzer zur Änderung seiner Blickrichtung zu motivieren.[[7]](#footnote-8) Diese könnten ein Wechsel der Beleuchtung, Aufblinken bestimmter Elemente oder Animationen sein, die den Blick des Benutzers lenken. HMDs bieten dabei einen größeres Sichtfeld als zwei-dimensionale Bildschirme, so dass dieser Aspekt eine größere Rolle spielt.

Abbildung 2 – Cone of Focus, UploadVR Copyright 2019 UVR Media LLC – <https://mk0uploadvrcom4bcwhj.kinstacdn.com/wp-content/uploads/2016/07/cone-of-focus.jpg>, letzter Abruf 17.06.2020

Bei Szenen- oder Positionswechsel bieten sich Fade-Out und Fade-In an, um den Übergang weicher zu gestalten. Schnelle Bewegungen der virtuellen Figur sollten vermieden werden, wenn sie nicht mit den Bewegungen des Körpers in der realen Welt korrespondieren. Room-Scale-VRist insofern unproblematischer, dass die virtuellen Bewegungen denen des Benutzers in der realen Welt entsprechen. Es bietet sich dabei ein Warnmechanismus z.B. in Form eines Gitters an, welches bei Annäherung an Hindernisse in der realen Welt eingeblendet wird oder das Einblenden der realen Welt als Echtzeit-Kamerabild.[[8]](#footnote-9)

Ein weiterer Aspekt – bei dem insbesondere das Zusammenspiel zwischen Software und Hardware wichtig ist – ist die Latenz zwischen Bewegung (beispielsweise des Kopfes) und der entsprechenden Berechnung des angezeigten Bildes, welche die ausgeführte Bewegung berücksichtigt. Wichtig ist dabei Latenz des gesamten Systems bestehend aus Motion-Tracking, Verarbeitung im Computer (Applikation, Rendering) und die Ausgabe auf dem HMD. Hier spielen auch Faktoren wie die Frequenz des Trackings, Bildwiederholrate des HMD und die Verwendung von Framebuffern eine Rolle. Diese Latenz muss möglichst gering ausfallen, da ansonsten das Präsenz-Gefühl des VR-Nutzers leide und es schnell zu Übelkeit bei der Nutzung kommen könne.[[9]](#footnote-10) Experimente am NASA Ames Research Center[[10]](#footnote-11) legen nahe, dass einerseits Probanden unterschiedlich sensibel auf verschiedene Latenzen reagieren und teilweise bei einem VR-System mit 7.4 ms Latenz noch Unterschiede im Bereich von 3.2 ms wahrgenommen werden konnten. Gängige HMDs (Stand 2015) liegen laut Messungen von Kjetil Raaen und Ivar Kiellmo bei einer Latenz von etwa 40 ms.[[11]](#footnote-12) Auch wenn die Fähigkeiten der eingesetzten Hardware vom Softwareentwickler nur bedingt beeinflusst werden können, sollte die Software auf stabile und hohe Frameraten hin optimiert werden, um Irritationen und Übelkeit zu vermeiden.

## Diskrete Fourier-Transformation

Die diskrete Fourier-Transformation (DFT) ist ein wichtiger Algorithmus in der digitalen Signalverarbeitung und überführt Signal-Daten aus dem Zeitbereich in den Frequenzbereich. Der Frequenzbereich bietet Möglichkeiten Eigenschaften des Signals abzuleiten oder das Signal zu bearbeiten, welche im Zeitbereich nur schwer oder nicht möglich sind. Eine beliebige Wellenform kann in sinoidale Einzelbestandteile aufgespalten werden und zeigt so die Zusammensetzung eines Signals – metaphorisch ähnlich wie bei der Zerlegung von Licht mithilfe eines Prismas. Fast-Fourier-Transform (kurz. FFT) beschreibt dabei eine Untermenge von Implementationen der DFT, die besonders effizient und schnell zu berechnen sind. Die DFT bildet die Grundlage für die Visualisierung und Modifikation der Audio-Daten in VrAudioSandbox (siehe Kapitel 4.2.1).

# Praktische Implementation „VrAudioSandbox“

## Repository und Systemvoraussetzungen

Der Programm-Code der selbst-geschriebenen Klassen findet sich unter „Anhang-Programmcode“. Das gesamte Projekt kann über das Github-Repository unter der URL

<https://github.com/ByteCrunch/vraudiosandbox>

geladen werden. Durch Klonen des Repository kann das gesamte Projekt begutachtet, im Unity Editor ausgeführt bzw. modifiziert oder als Build exportiert werden. Ein solcher Build, welcher auch ohne Unity-Installation nutzbar ist, findet sich unter dem Bereich „Releases“ ebenfalls zum Download.

Es wird Microsoft Windows 7 SP1 oder höher und eine Steam-Installation mit installierter SteamVR-Client-Runtime benötigt.

Außerdem ist das Projekt für ein htc VIVE HMD mit den dazugehörigen Motion-Controllern und Basisstationen konzipiert. Prinzipiell könnten auch andere Hersteller und Modelle unterstützt werden – hier können jedoch Einschränkungen bzgl. der Funktionalität auftreten und es muss ein separates Controller-Mapping erstellt werden.

Für das Öffnen des Projektes in Unity ist Version 2019.3.13f1 notwendig.

Als Entwicklungssystem wurde ein Windows-10--PC mit Intel i9-9900K, 32 Gigabyte DDR4-3200 RAM und einer GeForce RTX 2070 mit 8 Gigabyte VRAM genutzt. Auf diesem System ist die Darstellung bei Audio-Dateien mit einer Länge unter einer Minute bei Sampling-Raten bis 96 kHz und 24 Bit flüssig möglich. Der Speicherbedarf ist hoch: VrAudioSandbox benötigt nach dem Laden einer x Sekunden langen Audio-Datei mit einer Samplingrate von … und … bit … etwa 18 Gigabyte Arbeitsspeicher.

## Architektur und externe Bibliotheken

Im Rahmen dieser Arbeit wurde die Möglichkeit der Darstellung und Manipulation von Audio-Daten innerhalb der Virtual Reality praktisch implementiert. Dazu wurde neben den technischen Voraussetzungen ein Bedienkonzept zugeschnitten auf die Motion Tracking Controller der htc Vive entwickelt. Der Benutzer kann vorhandenes Audiomaterial importieren, mithilfe des htc Vive HMD und der dazugehörigen Motion-Controller modifizieren und das Ergebnis probehören oder als Audio-Datei exportieren. Die Bedienung findet dabei vollständig in VR statt.

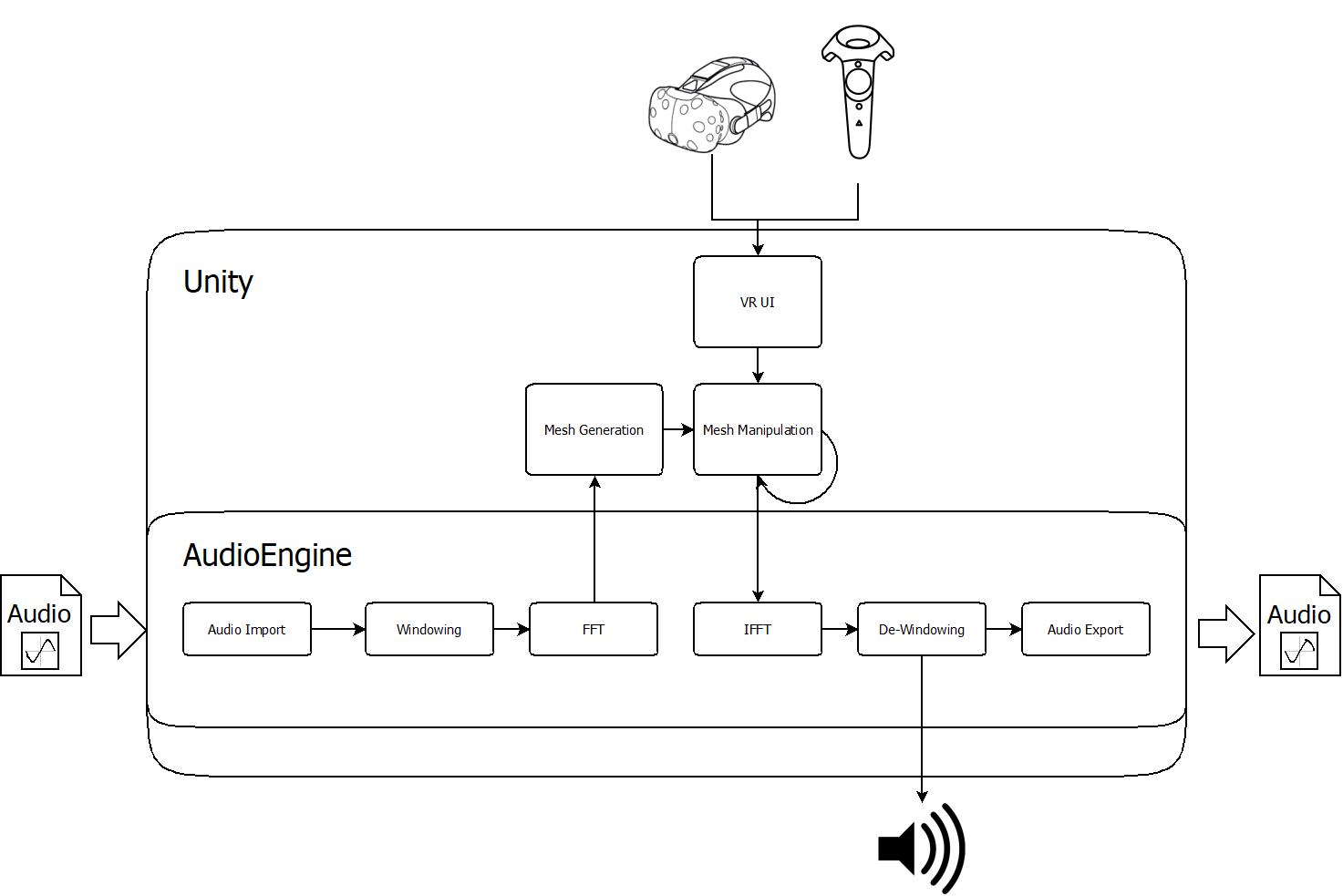
Im Mittelpunkt steht die Laufzeitumgebung Unity in der sämtliche Programmlogik in Form von eigenen Klassen implementiert ist. Darüber hinaus werden einige externe Bibliotheken genutzt, die im Folgenden vorgestellt und näher erläutert werden.

Abbildung 3 – Eine abstrakte Darstellung der Architektur - Eigene Darstellung

### Unity als Laufzeitumgebung

Die Laufzeit- und Entwicklungsumgebung Unity ist eine weitverbreitete Spiele-Engine mit etwa 50% Marktanteil für den Bereich PC/Konsole/Mobile Games.[[12]](#footnote-13) Das dazugehörige Unternehmen „Unity Technologies“ (ehemals „Over the edge“) wurde 2004 von David Helgason, Nicholas Francis und Joachim Ante gegründet. Auch außerhalb der kommerziellen Spiele-Entwicklung ist Unity sehr beliebt – u.a. auch da es eine für nicht-kommerzielle Zwecke kostenlose „Personal“-Lizenz gibt, welche den Hauptteil des Funktionsumfangs von Unity bereitstellt.[[13]](#footnote-14) Momentan (Stand: Juni 2020) unterstützt Unity 18 verschiedene Zielplattformen.[[14]](#footnote-15)

Unity bietet eine Vielzahl von vorgefertigten Komponenten im Bereich Grafik, Animation, Audio, Netzwerk, Physik und anderen Bereichen an. Außerdem liefert Unity viele Werkzeuge innerhalb des Unity-Editors mit, die bei der Spieleentwicklung hilfreich sind. Die Kernkomponenten von Unity sind in nativem C++ geschrieben. Die Logik und auch Erweiterung oder Modifikation der mitgelieferten Komponenten kann in der Sprache C# – ausgeführt innerhalb der freien, alternativen und quelloffenen Implementierung von Microsofts .NET Framework „Mono“ – implementiert werden.[[15]](#footnote-16) Alternativ steht das Scripting-Backend IL2CPP („Intermediate Language To C++“) für einige Zielplattformen zur Verfügung.

### SteamVR / OpenVR

SteamVR ein Sammelbegriff für das Virtual Reality Konzept der Firma Valve und umfasst eine API-Spezifikation (Application Programming Interface), eine Runtime, um mit OpenVR entwickelte Applikationen auszuführen und wird auch als Markenbegriff im Zusammenhang mit eigener VR-Hardware genutzt. OpenVR ist dabei das dazugehörige Software-Development-Kit, das auch auch mit Produkten anderer Hersteller nutzbar ist. In diesem SDK sind bereits viele vorgefertigte Bausteine enthalten – etwa für das Motion-Tracking, Teleportation, etc.

Außerdem bietet der Unity Asset Store das kostenlose „SteamVR Plugin“[[16]](#footnote-17), welches eine Vielzahl von Komponenten mitliefert, die direkt in Unity genutzt werden können.

### NAudio Bibliothek

NAudio ist eine von Mark Heath entwickelte Open-Source-Audio-API und Bibliothek für .NET und ist in C# geschrieben.[[17]](#footnote-18) Es wird für den Import der Audio-Dateien, die Wiedergabe und den Export als Audio-Datei genutzt. Darüber hinaus würde NAudio auch die Möglichkeit bieten Audio-Streams zu manipulieren, zu mixen und MIDI-Daten einzulesen und auszugeben.

### DFT mit FFTW3

FFTW (Akronym für „The Fastest Fourier Transform in the West”) ist eine freie FFT-Bibliothek zur Berechnung der diskreten Fourier-Transformation und wurde von Matteo Frigo und Steven G. Johnson am Massachusetts Institute of Technology entwickelt.[[18]](#footnote-19) Die in C und OCaml geschriebene Bibliothek ist im Quelltext verfügbar, sehr portabel und unterstützt daher viele Plattformen. Außerdem biete sie laut den Autoren im Vergleich zu anderen Implementationen eine sehr gute Performance.[[19]](#footnote-20)

FFTW bietet eine Vielzahl an Algorithmen und Funktionen an. Für die DFT und inverse DFT in VrAudioSandbox wird auf die „1d Discrete Fourier Transform (DFT)“ aus der FFTW3-Biblitohek mit Double-Precision zurückgegriffen, welche durch folgende mathematische Summenformel beschrieben wird:[[20]](#footnote-21)

Die Eingabe X ist dabei ein ein-dimensionales Array komplexer Zahlen der Größe *n* und *Y* das Ausgabe-Array, wobei das *k*-te Element der Frequenz *k/n* entspricht.

Bei der inversen Funktion handelt es sich um die gleiche Funktion der DFT:

FFTW berechnet eine un-normalisierte Transformation ohne Koeffizienten vor der Summe der DFT, d.h. eine vorwärts gerichtete DFT gefolgt von der rückwärtsgerichteten DFT ergibt als Ergebnis die Eingabe multipliziert mit *n*.

### FFTWSharp

FFTWSharp ist ein Wrapper um die C-Bibliothek FFTW3 aus C# heraus zu nutzen.[[21]](#footnote-22) Es vereinfacht und abstrahiert vor allem den Datenaustausch über „unmanaged“ Arrays (Arrays, die nicht vom Garbage-Collector der .NET-Laufzeitumgebung verwaltet werden) zwischen C# und C.

### CurvedUI

Für die Anzeige und Interaktion mit Unity-Canvases im World-Space wurde das CurvedUI-Interface-System des Entwicklers „Chisely“ aus dem Unity Asset Store[[22]](#footnote-23) in das Projekt importiert. Dieses vereinfacht das Raycasting von Canvas-Elementen stark. Unity ermöglicht im Auslieferungszustand nur das Raycasting von Collider-Komponenten, so dass es bei komplexen Canvas-UI-Elementen sehr aufwendig ist, diese Collider zu erstellen und zu verwalten. Zusätzlich werden noch andere Komfort-Funktionen angeboten, u.a. die namensgebene Möglichkeit für einen Canvas eine Biegung für bessere Blickwinkel-Ergonomie zu definieren.

## Implementation in Unity

Im Folgenden Abschnitt werden exemplarisch einige Besonderheiten, technische Details und Herausforderungen näher erläutert. Dazu werden einige Code-Beispiele aufgeführt. Für eine komplette Auflistung des Programmcodes sei auf den Anhang VIII „Programmcode“ oder das in 4.1 genannte Github-Repository verwiesen.

### Import und Fast-Fourier-Transformation

Nachdem der Benutzer eine Audio-Datei für den Import ausgewählt hat, wird diese mithilfe der NAudio-Bibliothek zunächst für die Weiterverarbeitung in ein Array bestehend aus den einzelnen Audio-Frames in Form von 32-Bit-Float-Werten überführt. Je nach Bit-Tiefe der Eingangsdaten werden dabei Bytes der PCM-Rohdaten zusammengefasst– bspw. bei 16-Bit Audio ergeben 2 Bytes ein Audio-Frame, bei 24-Bit-Audio 3 Bytes etc. NAudio beachtet außerdem, ob die Daten in Big-Endian- oder Little-Endian-Reihenfolge gespeichert sind, d.h. ob das höchstwertige Bit an erster oder letzter Stelle positioniert ist. Da NAudio die Ausgabe von Rohdaten nur als 32-Bit-Float-Array unterstützt, findet beim Import von Audio-Daten mit 64 Bit ein Verlust der Genauigkeit bzw. der entsprechenden Nachkommastellen statt.

Bevor die Audio-Daten der FFT zugeführt werden, wird die FFT-Size abhängig von der Sampling-Rate festgelegt. Bei 44.100 kHz hat sich eine FFT Size von 1024 als guter Kompromiss zwischen temporaler Auflösung, frequenzbezogener Auflösung und Rechenleistung für Berechnung und Visualisierung herausgestellt. Dieses Verhältnis zwischen Sampling-Rate und FFT-Size wird auch bei anderen Sampling-Raten verwendet.

Aus der FFT-Size folgt die Anzahl der FFT-Bins[[23]](#footnote-24):

Die Frequenz-Auflösung eines Bins beträgt[[24]](#footnote-25):

Beispielhaft beträgt bei 44.100 kHz Sampling-Rate und einer FFT-Size von 1024 die Frequenz-Auflösung eines Bins etwa 43 Hz – bei einer Anzahl von 512 Bins.

Die Audio-Daten werden nun in einzelne Teile entsprechend der FFT-Size aufgeteilt mit 50 % Überlappung – d.h. bis auf das erste und letzte Segment enthält jedes Segment die zweite Hälfte des vorherigen Segments und die erste Hälfte des nachfolgenden. Der letzte Teil wird – wenn er nicht der FFT-Size entspricht – mit Nullen aufgefüllt.

Auf jedes dieser Segmente werden die Koeffizienten der Von-Hann-Fensterfunktion angewendet. Die Fensterfunktion (oder auch das „Windowing“) vermindert den Leck-Effekt (auch „Spectral leakage“ genannt): Durch das Zerlegen des Signals in einzelne Segmente sind im Regelfall die Endpunkte des FFT-Windows nicht kontinuierlich. Dies führt bei der FFT ohne vorherige Fensterfunktion zu einer spektralen Streuung, die scheinbar existierende Spitzen in benachbarten Frequenzen erzeugt. Das Spektrum erscheint „verschmiert“. Eine Fensterfunktion reduziert die Amplitude zum Randbereich hin, so dass diesem Effekt entgegengewirkt wird. Durch die zuvor durchgeführte Überlappung wird sichergestellt, dass nicht zu viele Informationen, die in diesem Randbereich eines Segments liegen, verloren gehen. Das Anwenden der Fensterfunktion geschieht durch Multiplikation der Audio-Samples innerhalb eines FFT-Segments mit den Fensterfunktionskoeffizienten.

Diese werden entsprechend der FFT-Size *n* für jedes Element *i* eines Segments generiert:

**Fft.cs**

Listing 1 – Generierung der Von-Hann-Koeffizienten

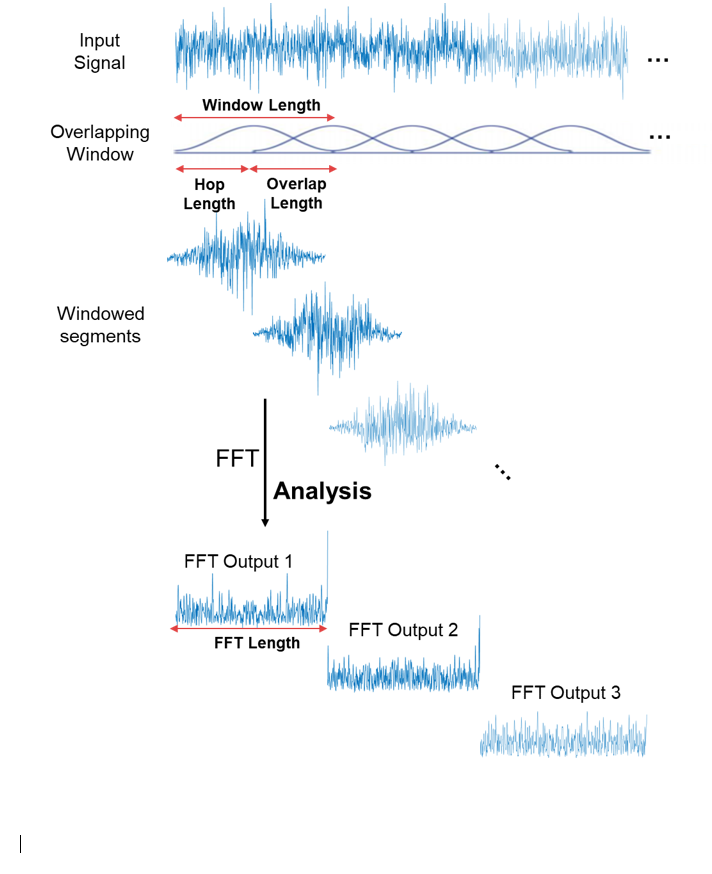
Der Code zur Generierung der verschiedenen Fensterfunktionskoeffizienten wurde von Thomas Buck aus seinem Projekt „AudioTest“ übernommen[[25]](#footnote-26) und unterstützt auch die Fensterfunktionen „Hamming“, „Blackman“ und „Flat“ (mit konstantem Koeffizient 1.0, was keiner Fensterfunktion entspricht). Abbildung 4 zeigt eine zusammengefasste Darstellung der Segmentierung des Original-Signals, dem Overlap und der Von-Hann-Fensterfunktion.

Abbildung 4 – Illustration Windowing mit 50 % Overlap – „Short-time FFT – MATLAB“, © 1994-2020 The MathWorks, Inc. – <https://www.mathworks.com/help/dsp/ref/stft_output.png>, letzter Abruf: 29.06.2020

Die FFT wird jetzt für jedes dieser überlappenden Segmente durchgeführt. Da FFTW für die FFT „dft\_1d“ komplexe Zahlen erwartet, wird dazu ein double-Array mit doppelter Größe initialisiert und alle gerade Indizes mit den Amplituden-Werten gefüllt. Die ungeraden Indizes enthalten laut FFTW-Konvention den Imaginärteil, der in diesem Fall immer 0 beträgt.

Da es sich bei FFTW um eine C-Bibliothek handelt, erfolgt der Austausch über einen separaten Speicherbereich, der manuell instanziert und im Anschluss auch wieder freigegeben werden muss. Nachdem die Fourier-Transformation als sogenannter „Plan“ konfiguriert ist, wird er durch „fftw.execute“ in Zeile 90 ausgeführt. Vorher und nacher werden die Daten zwischen „managed“ Speicherbereich der Mono-Lauftzeitumgebung und dem „unmanaged“ Speicherbereich von FFTW hin und her kopiert.

**Fft.cs**

Listing 2 – Instanzieren des Speicherbereichs, Erstellung fftw-Plan und Ausführung

Anschließende Freigabe der Speicher-Allokation:

**Fft.cs**

Listing 3 – Freigeben der fftw-Speicher-Allokation

Um die Magnituden für die Darstellung des Spektrums zu erhalten, muss nun der mathematische Betrag für jedes der Elemente der FFT-Daten berechnet werden. Dieser ist für komplexe Zahlen wie folgt definiert:

Diese Daten werden nun für die Visualisierung und Manipulation genutzt und entsprechen dem Amplituden-Spektrum.[[26]](#footnote-27)

### Visualisierung der Spektrum-Daten

Aus Performanz-Gründen wird für jedes Segment der Amplituden-Spektrum-Daten ein eigenes Mesh zur Laufzeit erzeugt. Alle Daten in ein Mesh zu laden ist prinzipiell möglich (durch das Vergrößern der Mesh-Indizes von 16 Bit auf 32 Bit[[27]](#footnote-28)), führte jedoch zu extremen Performance-Einbußen und teilweise zum Absturz von Unity.

Zur Erzeugung eines dieser Meshes müssen zunächst Vertices erstellt werden. Dazu wird zunächst ein Raster aus Vertices kreiert, die später die Grundfläche der einzelnen „Pyramiden“ ergeben, welche die Werte des Amplituden-Spektrums repräsentieren. Im Anschluss werden die Vertices für die Spitzen abhängig von der Amplitude skaliert auf der y-Achse generiert.

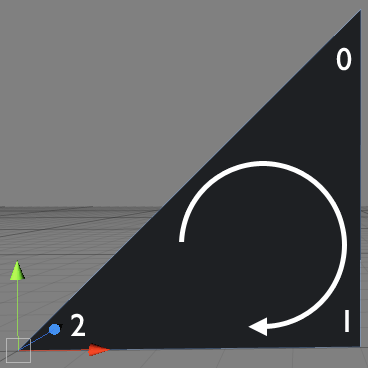
Damit eine sichtbare Visualisierung entsteht, müssen nun je 3 dieser Vertices zu Triangles (auch Polygone genannt) verbunden werden. Dabei ist aufgrund des sogenannten „Cullings“, das vereinfacht ausgedrückt Renderzeit spart, indem es Innenseiten beim Rendering ignoriert, die korrekte Reihenfolge beim Definieren der entsprechenden Vertex-Indizes zu beachten: Unity verwendet die sogenannte „Clockwise-Winding-Order“, d.h. nur Polygone die aus Sicht der Kamera im Uhrzeigersinn angeordnet sind, sind als „vorderseitig“ und damit sichtbar klassifiziert.[[28]](#footnote-29)

Abbildung – Unity Forums: „Unity Winding Order”, Screenshot von Eric5h5 – <https://forum.unity.com/attachments/mesh1-png.244462/>, letzter Abruf: 03.07.2020

Die Triangles werden über ein Integer-Array definiert, welches in Dreiergruppen die Indizes der Vector3-Arrays, die die Vertex-Positionen enthalten, referenziert.

**SpectrumMeshGenerator.cs**

Listing 4 – Definition der Triangles mit korrekter Winding-Order

Zur Einfärbung der Pyramiden werden Vertex-Colors definiert, die über einen benutzdefinierten Shader durch die Grafikkarte dargestellt werden.

Zusätzlich wird für jedes Mesh auch ein MeshCollider zur Laufzeit generiert, der später für die Manipulation der Spektrum-Meshes mithilfe von Raycasting genutzt wird.

### Manipulation der Spektrum-Meshes

Mithilfe der SteamVR\_LasterPointer-Komponente, die Teil des SteamVR Plugins für Unity ist, wurde Raycasting ausgehend vom rechten Motion-Controller implementiert. Ein sichtbarer Strahl vereinfacht das Anvisieren einzelner Bereiche des Meshes. Bei Kollision mit einem Collider-Objekt, wird das Event „PointerIn“ ausgelöst. Zusätzlich gibt es das Event „PointerClick“, falls die Action „Trigger“ während einer Kollision auftritt. Die Klasse „SteamVR\_LaserPointer“ wurde modifiziert, um im Falle eines Events die Koordinaten des Schnittpunktes als Event-Argument mit zu übergeben. Dazu wurde das Vector3-Attribut „point“ dem Struct „PointerEventArgs“ beigefügt und der Code zur Erzeugung des Events entsprechend ergänzt. Dies erleichert die im Folgenden beschriebene Logik zur Modifizierung der Spektrum-Meshes.

In der Klasse „ToolHandler.cs“ wird ausgehend vom Schnittpunkt und dem gewählten Radius werden jene Vertices, welche die Spektrum-Daten repräsentieren (die Spitzen der Pyramiden) und sich innerhalb des Radius befinden durch Skalarmultiplikation oder einem neuen absoluten Wert innerhalb der y-Achse modifiziert.

Die Logik für die Veränderung des Mesh-Spektrums findet sich in der Klasse „SpectrumDeformer.cs“. Als problematisch hat sich dabei die Performance erwiesen: Konzeptionell entsteht ein Flaschenhals innerhalb von Unity beim Transfer einer großen Anzahl an Mesh-Vertices zur Laufzeit. Dies ist dadurch begründet, dass diese Daten vom unmanged Speicherbereich der Mono-Laufzeitumgebung zum C++-Kern von Unity kopiert werden müssen und im Anschluss zur Grafikkarte übertragen werden (bzw. vice versa). Es ist nicht möglich (Stand: Unity 2019.3.13f) auf einzelne Vertices innerhalb des Mesh-Arrays zuzugreifen – sowohl beim Zugriff über Mesh.vertices, wobei es sich nicht um ein Attribut, sondern einen Getter handelt, der ein Array zurückgibt, als auch über die neuere Methode GetVertices() mit Rückgabetyp Liste werden immer alle Vertices des Meshes kopiert, was eine hohe Latenz zur Folge hat. Zusätzlich war zunächst die Laufzeit bei der Identifikation der zu verändernden Vertices aufgrund der Berechnung der Distanz zum Kollisionspunkt durch Iteration über alle Vertices unpraktikabel hoch. Eine Modifikation des Spektrums dauerte mit einer ersten naiven Implementation Minuten und verzögerte das Rendering in für den Benutzer nicht zumutbarer Art und Weise.

Um diese Probleme zu umgehen, wurden zweierlei Maßnahmen ergriffen:

Durch „Unity Coroutines“ wird die Ausführung innerhalb des Main-Threads bei performance-kritischen Operationen ausgesetzt und bei der Berechnung des nächsten Frames wieder fortgeführt.[[29]](#footnote-30) Dies ist wichtig für die Einhaltung einer geringen Latenz. So kann der nächste Frame rechtzeitig gerendert werden. Dies verlangsamt allerdings die Fertigstellung der Berechnung zusätzlich, da diese nun unterbrochen wird. Außerdem müssen durch „yield return;“-Statements Stellen im Code definiert werden, die als „Ausstiegspunkte“ dienen. Dabei muss ein Kompromiss zwischen Latenz und Verzögerung der Berechnung gefunden werden.

Zur drastischen Verringerung der Laufzeit für die Berechnung der zu verändernden Vertices wurde diese Logik mit Multi-Thread-Programmierung neu implementiert. Unity ist per se nicht thread-safe, so dass ein Zugriff auf API-Funktion von Unity außerhalb des Main-Threads im besten Fall nicht-deterministisch ist oder gar zum Absturz führt. Prinzipiell könnten Funktionen, die keine Unity-API-Aufrufe enthalten multi-threaded programmiert werden, allerdings würde eine separate Multi-Thread-Implementation neben Unity bzgl. des Schedulings nicht optimal auf die Anzahl der im System verbauten Kerne skalieren. Unity bietet seit Version 2018.1[[30]](#footnote-31) das „Unity Job System“, welches eine Untermenge der aus C# ansprechbaren Funktionalität innerhalb des nativen, internen Job Systems ausführen kann. So kann ein Teil der Programmlogik mit C# erstellt werden und wird dann innerhalb der intern in Unity bereits vorhandenen „Worker Threads“ mit echter Nebenläufigkeit ausgeführt. Der „Burst Compiler“ kompiliert den entsprechenden Code in nativen Code, der im Kontext von Unity ausgeführt wird – er läuft also nicht innerhalb der Mono-Laufzeitumgebung.[[31]](#footnote-32) Im Zuge dessen müssen spezifische Datentypen genutzt werden, dessen Speicherallokationen selbst verwaltet werden. Der Job-Typ IJobParallelFor führt dabei ähnlich einer for-Schleife den Job-Code für Segmente des Eingabe-Arrays entsprechend der Batch-Size parallel aus. Diese Segmente werden durch Unity automatisch entsprechend der Anzahl der logischen Prozessorkerne auf die Worker-Threads verteilt. Da aus den Threads gemeinsam in eine Datenstruktur geschrieben werden muss, wird der Datentyp NativeQueue genutzt. Nur dieser Datentyp erlaubt das Schreiben der Ergebnisse – also die Vertex-Werte, die verändert werden müssen – in eine gemeinsame Datenstruktur. Diese Änderungen werden dann aus den Main-Thread heraus lokal in die Kopie „modifiedVertices“ eingepflegt und je Mesh über den Mesh-Vertices-Setter zunächst in Unity und dann schlussendlich zur Grafikkarte übertragen.[[32]](#footnote-33) In diesem Zuge werden auch die FFT-Daten selbst, sowie die Mesh-Colliders und Mesh-Colors entsprechend angepasst.

**SpectrumDeformer.cs** 

Listing 5 – Multi-threaded Burst Code zum Finden der Vertices innerhalb des Kollisionsradius

### Inverse Fast-Fourier-Transformation und Wiedergabe

Um die Veränderungen im Amplituden-Spektrum hörbar zu machen bzw. eine Audio-Datei exportieren zu können, muss eine inverse oder rückwärtsgerichtete Fourier-Transformation berechnet werden. Dabei wird prinzipiell die gleiche mathematische Funktion wie bei der vorwärts gerichteten FFT angewandt – jedoch ohne negatives Vorzeichen im Exponenten.

Durch das Bilden des mathematischen Betrags (siehe 4.2.1) sind Informationen verloren gegangen, die für die Konstruktion des zeitbasierten Signals benötigt werden. Dabei handelt es sich um die sogenannten Phaseninformationen der Frequenz-Komponenten. Die Phaseninformationen lassen sich aus den FFT-Daten mit folgender Formel berechnen:[[33]](#footnote-34)

In der praktischen Implementation ist zu beachten, dass für die Berechnung der inversen Winkelfunktion Arkustangens die Funktion „atan2“ (oder alternativ auch „arctan2“ genannt) genutzt werden muss, da nur so der volle Wertebereich von 360° (alle vier Quadranten des Koordinatensystems) korrekt berücksichtigt wird.[[34]](#footnote-35)

Ein weiteres Problem für die Berechnung der Phaseninformation ist die Tatsache, dass kleine Rundungsfehler durch die Nutzung von Fließkommazahlen bei naiver Berechnung der oben genannten Formel zu unbrauchbaren Ergebnissen führen. Die im Web-Blog „GaussianWaves“ vorgestellte Lösung von [Mathuranathan Viswanathan](https://www.linkedin.com/in/mathuranathan/) ist einen Schwellwert zu definieren und alle berechneten Phasenwerte unterhalb des Schwellwertes zu ignorieren.[[35]](#footnote-36) Der Schwellwert wird in diesem Fall als 1/10000 der maximalen Magnitude definiert. Listing 2 zeigt die Implementation dieses Konzeptes in C#.

**Fft.cs**

Listing 6 – Schwellwert-Logik zur Berechnung der Phaseninformationen

Mithilfe der berechneten Phaseninformationen und der Amplituden-Spektrum-Daten ist es nun möglich die Daten der Zeit-Domäne (also die digitale Repräsentation des Audio-Signals) zu rekonstruieren. Dazu müssen zunächst die komplexen FFT-Daten aus jenen beiden Daten erzeugt werden – also den Phaseninformationen und potenziell veränderten Amplituden. Mathematisch betrachtet handelt es sich dabei um die Umwandlung von Polarkoordinaten in kartesische Koordinaten. Das Ergebnis ist eine komplexe Zahl bestehend aus Real- und Imaginärteil, wobei *r* dem Betrag entspricht (also der Magnitude) und der Phase: [[36]](#footnote-37)

Die Implementation in C# entsprechend der FFTW-Konvention der Darstellung komplexer Zahlen als alternierende Folge von Real- und Imaginärteil:

**Fft.cs**

Listing 7 – Berechnung der FFT-Daten aus Phasen- und Amplituden-Daten für die anschließende IFFT

Die rückwärtsgerichtete FFT mit der FFTW-Bibliothek wird nun analog zur FFT angewandt – allerdings mit dem Parameter „fftw\_direction.Backward“, was einem veränderten Vorzeichen im Exponenten entspricht. Das Ergebnis sind Audio-Samples, die noch die Veränderungen durch die Von-Hann-Fensterfunktion mit 50 % Überlappung enthalten. Diese können durch einfaches paarweises Summieren der Werte in den beiden Überlappungs-Regionen wieder entfernt werden. Für das erste und letzte Segment werden die Daten durch die Fensterfunktion-Koeffizienten dividiert.

**AudioEngine.cs**

Listing 8 – De-Windowing eines Von-Hann-Fensters mit 50% Overlap

Die resultierenden Sample-Daten werden nun in ein MemoryStream-Objekt geschrieben und können so durch NAudio wiedergegeben werden oder als Audio-Datei exportiert werden.

### User Interface und Steuerung

* Motion-Controller, Actions
* Konzept: Interaktion ohne verschachtelte 2D-Menüs, Werkzeugfarben, Ablesen im virtuellen Raum

Die Visualisierung der Daten erzeugt ein Raumgefühl, das zur Erkundung einlädt.

# Kritische Betrachtung der Ergebnisse

* Verbesserungsvorschläge
* Ideen für weitere Werkzeuge / Funktionen

# Fazit

# Literatur

**Fischer-Stabel, Peter**: „Datenvisualisierung – Vom Diagramm zur Virtual Reality“, utb Verlagsgemeinschaft, UVK Verlag, München (2018), ISBN 9783825250287

**Frigo, Matteo; Johnson, Steven G.**: „The Design and Implementation of FFTW3” – Proceedings of the IEEE, Volume 93, Number 2, 2005, S. 216-231 – http://www.fftw.org/fftw-paper-ieee.pdf, letzter Abruf: 13.06.2020

**Jerald, Jason** (Ph.d.): „The VR Book: Human-Centered Design for Virtual Reality”, A publication in the ACM Book series #8, Association for Computing Machinery and Morgan & Claypool Publishers, San Rafael (California) (2016), ISBN 9781970001129

**Lanier, Jaron**: „Anbruch einer neuen Zeit – Wie Virtual Reality unser Leben und unsere Gesellschaft verändert”, Hoffmann und Campe Verlag, Hamburg (2018), übersetzt von Heike Schlatterer und Sigrid Schmid (erstveröffentlicht als: „Dawn of the New Everything. Encounters with Reality and Virtual Reality“, Henry Holt and Company, New York [2017]), ISBN 9783455003994

**Norman, Donald A.**: „The Design of Everyday Things”, Basic Books – Hachette Book Group company, New York (2002) (erstveröffentlicht als: „The Psychology of Everyday Things” [1988]), ISBN 9780465067107

**Sutherland, Ivan**: „A head-mounted three dimensional display” – AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I, December 1968 Pages 757–764, S. 757 – <http://www.medien.ifi.lmu.de/lehre/ss09/ar/p757-sutherland.pdf>, letzter Abruf: 10.06.2020

**Sutherland, Ivan**: „The Ultimate Display” – Konferenzband: Information Processing 1965: proceedings of IFIP Congress / Wayne A. Kalenich [Hrsg.]. International Federation for Information Processing, Amsterdam u.a., Washington u.a. (1965)

**Wade, Nicholas J.**: Guest Editorial „Charles Wheatstone (1802 - 1875)”, Perception (Journal), SAGE Publications (ehemals: Pion), Thousand Oaks (2002), Ausgabe 31, ISSN: 0301-0066

# Sonstige Quellen

**FFTW**: „FFT Benchmark Results“ – http://www.fftw.org/speed/, letzter Abruf: 13.06.2020

**FFTW**: 3.3.8 Manual, Kapitel 4.8.1 „The 1d Discrete Fourier Transform (DFT)” – http://www.fftw.org/fftw3\_doc/The-1d-Discrete-Fourier-Transform-\_0028DFT\_0029.html, letzter Abruf: 13.06.2020

**Gaussian Waves**: How to interpret FFT results – obtaining magnitude and phase information – https://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/, letzter Abruf: 01.07.2020

**Github**: AudioTest/fft.cs at master, – Code von Thomas Buck – https://github.com/101010b/AudioTest/blob/master/fft.cs, letzter Abruf: 01.07.2020

**Github**:C# wrapper for FFTW – https://github.com/tszalay/FFTWSharp, letzter Abruf: 23.06.2020

**Github**:NAudio repository – https://github.com/naudio/NAudio, letzter Abruf: 23.06.2020

**Ircam**: AudioSculpt 3.0 User Manual „Introduction - FFT Size“ – http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html, letzter Abruf: 29.06.2020

**Khan Academy**: „Polar & rectangular form of complex number” – https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:complex/x9e81a4f98389efdf:complex-polar/v/polar-form-complex-number, letzter Abruf: 03.07.2020

**Martin Ritter**: „Unity mesh generation: vertices, triangles, winding” – http://www.martin-ritter.com/2019/01/unity-mesh-generation-vertices-triangles-winding/, letzter Abruf: 02.07.2020

**Unity**: “Burst User Guide” – https://docs.unity3d.com/Packages/com.unity.burst@0.2/manual/index.html, letzter Abruf: 07.07.2020

**Unity Asset Store**: „CurvedUI” – https://assetstore.unity.com/packages/tools/gui/curved-ui-vr-ready-solution-to-bend-warp-your-canvas-53258, letzter Abruf: 07.07.2020

**Unity Asset Store**: „SteamVR Plugin” – https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647, letzter Abruf: 07.07.2020

**Unity Documentation**: „C# Job System Overview“ – https://docs.unity3d.com/2019.3/Documentation/Manual/JobSystemOverview.html, letzter Abruf: 07.07.2020

**Unity Documentation**: “Coroutine” – https://docs.unity3d.com/ScriptReference/Coroutine.html, letzter Abruf: 07.07.2020

**Unity Technologies**:„Multiplatform | Unity“ – https://unity.com/features/multiplatform, letzter Abruf: 24.06.2020

**Unity Technologies**: „Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps.” – https://store.unity.com/#plans-individual, letzter Abruf: 11.06.2020

**Unity Technologies**: „Sie fragen sich was Unity ist? Entdecken Sie, wer wir sind, wo wir angefangen haben und wohin wir uns entwickeln | Unity“ – https://unity.com/de/our-company, letzter Abruf: 11.06.2020

# Abbildungsverzeichnis

[Abbildung 1 – Vergleich HUD-basierte Darstellung (oben) von Gesundheit und Munition mit im virtuellen Raum als physische Objekte manifestierter Darstellung (unten) 4](file:///C:\Users\bytecrunch\Documents\Uni\Bachelor%20Arbeit\vraudiosandbox\Manuel-Philippe%20Hergenröder%20-%20Interaktion%20mit%20Audio-Daten%20in%20VR%20mithilfe%20von%20Unity%20und%20der%20htc%20Vive.docx#_Toc44689950)

[Abbildung 2 – Cone of Focus, UploadVR Copyright 2019 UVR Media LLC – https://mk0uploadvrcom4bcwhj.kinstacdn.com/wp-content/uploads/2016/07/cone-of-focus.jpg, letzter Abruf 17.06.2020 5](file:///C:\Users\bytecrunch\Documents\Uni\Bachelor%20Arbeit\vraudiosandbox\Manuel-Philippe%20Hergenröder%20-%20Interaktion%20mit%20Audio-Daten%20in%20VR%20mithilfe%20von%20Unity%20und%20der%20htc%20Vive.docx#_Toc44689951)

[Abbildung 3 – Eine abstrakte Darstellung der Architektur - Eigene Darstellung 8](file:///C:\Users\bytecrunch\Documents\Uni\Bachelor%20Arbeit\vraudiosandbox\Manuel-Philippe%20Hergenröder%20-%20Interaktion%20mit%20Audio-Daten%20in%20VR%20mithilfe%20von%20Unity%20und%20der%20htc%20Vive.docx#_Toc44689952)

[Abbildung 4 – Illustration Windowing mit 50 % Overlap – „Short-time FFT – MATLAB“, © 1994-2020 The MathWorks, Inc. – https://www.mathworks.com/help/dsp/ref/stft\_output.png, letzter Abruf: 29.06.2020 13](file:///C:\Users\bytecrunch\Documents\Uni\Bachelor%20Arbeit\vraudiosandbox\Manuel-Philippe%20Hergenröder%20-%20Interaktion%20mit%20Audio-Daten%20in%20VR%20mithilfe%20von%20Unity%20und%20der%20htc%20Vive.docx#_Toc44689953)

[Abbildung 5 – Unity Forums: „Unity Winding Order”, Screenshot von Eric5h5 – https://forum.unity.com/attachments/mesh1-png.244462/, letzter Abruf: 03.07.2020 15](file:///C:\Users\bytecrunch\Documents\Uni\Bachelor%20Arbeit\vraudiosandbox\Manuel-Philippe%20Hergenröder%20-%20Interaktion%20mit%20Audio-Daten%20in%20VR%20mithilfe%20von%20Unity%20und%20der%20htc%20Vive.docx#_Toc44689954)

# Code-Listing-Verzeichnis

[Listing 1 – Generierung der Von-Hann-Koeffizienten 12](#_Toc45029837)

[Listing 2 – Instanzieren des Speicherbereichs, Erstellung fftw-Plan und Ausführung 14](#_Toc45029838)

[Listing 3 – Freigeben der fftw-Speicher-Allokation 14](#_Toc45029839)

[Listing 4 – Definition der Triangles mit korrekter Winding-Order 16](#_Toc45029840)

[Listing 5 – Multi-threaded Burst Code zum Finden der Vertices innerhalb des Kollisionsradius 19](#_Toc45029841)

[Listing 6 – Schwellwert-Logik zur Berechnung der Phaseninformationen 21](#_Toc45029842)

[Listing 7 – Berechnung der FFT-Daten aus Phasen- und Amplituden-Daten für die anschließende IFFT 22](#_Toc45029843)

[Listing 8 – De-Windowing eines Von-Hann-Fensters mit 50% Overlap 23](#_Toc45029844)

# Eidesstattliche Erklärung

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe.  Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde/Prüfungsstelle vorgelegen hat. Ich erkläre mich damit nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich gemacht wird.  Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hochgeladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öffentlichkeit dar. | | | | |
|  | | | | |
| Oberboihingen,  der xx.07.2020 |  | Manuel-Philippe Hergenröder |  |  |
| Ort, Datum |  | Vorname Nachname |  | Unterschrift |

# Anhang – Programmcode

1. *Anmerkung: Für die Entwicklung wurden das im März 2015 vorgestellte und im April 2016 auf dem Markt verfügbare HMD HTC Vive und die dazugehörigen Motion-Controller verwendet.*  [↑](#footnote-ref-2)
2. Vgl. Sutherland, Ivan: „The Ultimate Display” – Konferenzband: Information Processing 1965: proceedings of IFIP Congress / Wayne A. Kalenich [Hrsg.]. International Federation for Information Processing, Amsterdam u.a., Washington u.a. (1965), S. 508 – <http://worrydream.com/refs/Sutherland%20-%20The%20Ultimate%20Display.pdf>, letzter Abruf: 09.06.2020 [↑](#footnote-ref-3)
3. Lanier, Jaron: „Anbruch einer neuen Zeit – Wie Virtual Reality unser Leben und unsere Gesellschaft verändert”, Hoffmann und Campe Verlag, Hamburg (2018), S. 81-82 [↑](#footnote-ref-4)
4. Vgl. Wade, Nicholas J.: Guest Editorial „Charles Wheatstone (1802 - 1875)”, Perception (Journal), SAGE Publications (ehemals: Pion), Thousand Oaks (2002), Ausgabe 31, ISSN: 0301-0066, S. 265 [↑](#footnote-ref-5)
5. Vgl. Sutherland, Ivan: „A head-mounted three dimensional display” – AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I, December 1968 Pages 757–764, S. 757 – <http://www.medien.ifi.lmu.de/lehre/ss09/ar/p757-sutherland.pdf>, letzter Abruf: 10.06.2020 [↑](#footnote-ref-6)
6. Vgl. Ebd. S. 760-761 [↑](#footnote-ref-7)
7. Vgl. Jerald, Jason (Ph.d.): „The VR Book: Human-Centered Design for Virtual Reality”, A publication in the ACM Book series #8, Association for Computing Machinery and Morgan & Claypool Publishers, San Rafael (California) (2016), S. 254 [↑](#footnote-ref-8)
8. Vgl. Ebd. S. 213 [↑](#footnote-ref-9)
9. Vgl. Ebd. S. 183-184 [↑](#footnote-ref-10)
10. Vgl. Ebd. S. 184-185 – Adelstein et al. 2003 & 2006, Jerald 2009, Ellis et al. 1999, 2004. Mania et al. 2004 [↑](#footnote-ref-11)
11. Vgl. Raaen, Kjetil; Kjellmo, Ivar: „Measuring Latency in Virtual Reality Systems”, 14th International Conference on Entertainment Computing (ICEC), Sep 2015, Trondheim, Norway. pp.457-462, 10.1007/978-3-319-24589-8\_40. hal-01758473

    *Anmerkung: Stand 2015, zwischen 35-45 ms wurden bei der Ocolus Rift DK2 gemessen bei aktivierten V-Sync – d.h. Synchronisierung der Bildwiederholrate mit der Grafikkarte, um Tearing-Effekte zu vermeiden. Dazu wurde eine Unity Szene und eine lichtempfindliche Photozelle verwendet. Latenz durch Eingabegeräte ist nicht berücksichtigt.* [↑](#footnote-ref-12)
12. Vgl. Unity Technologies: „Sie fragen sich was Unity ist? Entdecken Sie, wer wir sind, wo wir angefangen haben und wohin wir uns entwickeln | Unity“ – <https://unity.com/de/our-company>, letzter Abruf: 11.06.2020 [↑](#footnote-ref-13)
13. Vgl. Unity Technologies: „Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps.” – https://store.unity.com/#plans-individual, letzter Abruf: 11.06.2020 [↑](#footnote-ref-14)
14. Vgl. Unity Technologies: „Multiplatform | Unity“ – <https://unity.com/features/multiplatform>, letzter Abruf: 24.06.2020 [↑](#footnote-ref-15)
15. *Anmerkung: In der Vergangenheit konnten alterrnativ die Skriptsprachen Boo und UnityScript verwendet werden. Diese sind aber als veraltet klassifiziert.* [↑](#footnote-ref-16)
16. Unity Asset Store: „SteamVR Plugin” – <https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-17)
17. Vgl. Github – NAudio repository – <https://github.com/naudio/NAudio>, letzter Abruf: 23.06.2020 [↑](#footnote-ref-18)
18. Vgl. Frigo, Matteo; Johnson, Steven G.: “The Design and Implementation of FFTW3” – Proceedings of the IEEE, Volume 93, Number 2, 2005, S. 216-231, S. 231 – <http://www.fftw.org/fftw-paper-ieee.pdf>, letzter Abruf: 13.06.2020 [↑](#footnote-ref-19)
19. Vgl. FFTW: „FFT Benchmark Results“ – <http://www.fftw.org/speed/>, letzter Abruf: 13.06.2020 [↑](#footnote-ref-20)
20. Vgl. FFTW: 3.3.8 Manual, Kapitel 4.8.1 „The 1d Discrete Fourier Transform (DFT)” – <http://www.fftw.org/fftw3_doc/The-1d-Discrete-Fourier-Transform-_0028DFT_0029.html>, letzter Abruf: 13.06.2020 [↑](#footnote-ref-21)
21. Vgl. Github: C# wrapper for FFTW – <https://github.com/tszalay/FFTWSharp>, letzter Abruf: 23.06.2020 [↑](#footnote-ref-22)
22. Unity Asset Store: „CurvedUI” – <https://assetstore.unity.com/packages/tools/gui/curved-ui-vr-ready-solution-to-bend-warp-your-canvas-53258>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-23)
23. Vgl. Ircam: AudioSculpt 3.0 User Manual „Introduction - FFT Size“ – <http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html>, letzter Abruf: 29.06.2020 [↑](#footnote-ref-24)
24. *Anmerkung: In der Literatur findet sich auch oft die äquivalente Formel: „Höchste darstellbare Frequenz / Anzahl der FFT Bins“. Der Quotient ist identisch, da Dividend und Divisor bei dieser Definition jeweils halbiert sind.* [↑](#footnote-ref-25)
25. Vgl. Github: AudioTest/fft.cs at master, – Code von Thomas Buck – <https://github.com/101010b/AudioTest/blob/master/fft.cs>, letzter Abruf: 01.07.2020 [↑](#footnote-ref-26)
26. Vgl. Gaussian Waves: How to interpret FFT results – obtaining magnitude and phase information „3a. Extract amplitude of frequency components (amplitude spectrum)” – <https://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/>, letzter Abruf: 01.07.2020 [↑](#footnote-ref-27)
27. *Anmerkung: Dazu wird nach dem Erstellen des Meshes die Property .indexFormat des Mesh-Objektes = UnityEngine.Rendering.IndexFormat.UInt32 gesetzt. Dies hat zur Folge, dass für den Index Buffer 32bit genutzt werden, so dass bis zu 4 Milliarden Vertices indexiert werden können. Standardmäßig werden 16bit genutzt, um Bandbreite und Speicher zu sparen und da nicht alle Plattformen mehr als 16bit unterstützen.* Siehe auch <https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html>, letzter Abruf: 01.07.2020 [↑](#footnote-ref-28)
28. Vgl. Martin Ritter: „Unity mesh generation: vertices, triangles, winding” – <http://www.martin-ritter.com/2019/01/unity-mesh-generation-vertices-triangles-winding/>, letzter Abruf: 02.07.2020 [↑](#footnote-ref-29)
29. Vgl. Unity Documentation: “Coroutine” – <https://docs.unity3d.com/ScriptReference/Coroutine.html>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-30)
30. Vgl. Unity Documentation: „C# Job System Overview“ – <https://docs.unity3d.com/2019.3/Documentation/Manual/JobSystemOverview.html>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-31)
31. Vgl. Unity: “Burst User Guide” – <https://docs.unity3d.com/Packages/com.unity.burst@0.2/manual/index.html>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-32)
32. *Anmerkung: Mit Unity 2020 wird ein direkter Zugriff auf die Vertices – ohne Umweg über den managed Speicherbereich von Mono – erstmalig ausgehend vom Unity Job System eingeführt. Diese Methode ist allerdings noch als experimentell gekennzeichnet und unzureichend dokumentiert (Stand: 05/2020)* [↑](#footnote-ref-33)
33. Vgl. Gaussian Waves: How to interpret FFT results – obtaining magnitude and phase information „3b. Extract phase of frequency components (phase spectrum)” – <https://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/>, letzter Abruf: 01.07.2020 [↑](#footnote-ref-34)
34. Vgl. Ebd. [↑](#footnote-ref-35)
35. Vgl. Ebd. [↑](#footnote-ref-36)
36. Vgl. Khan Academy: „Polar & rectangular form of complex number” – <https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:complex/x9e81a4f98389efdf:complex-polar/v/polar-form-complex-number>, letzter Abruf: 03.07.2020 [↑](#footnote-ref-37)