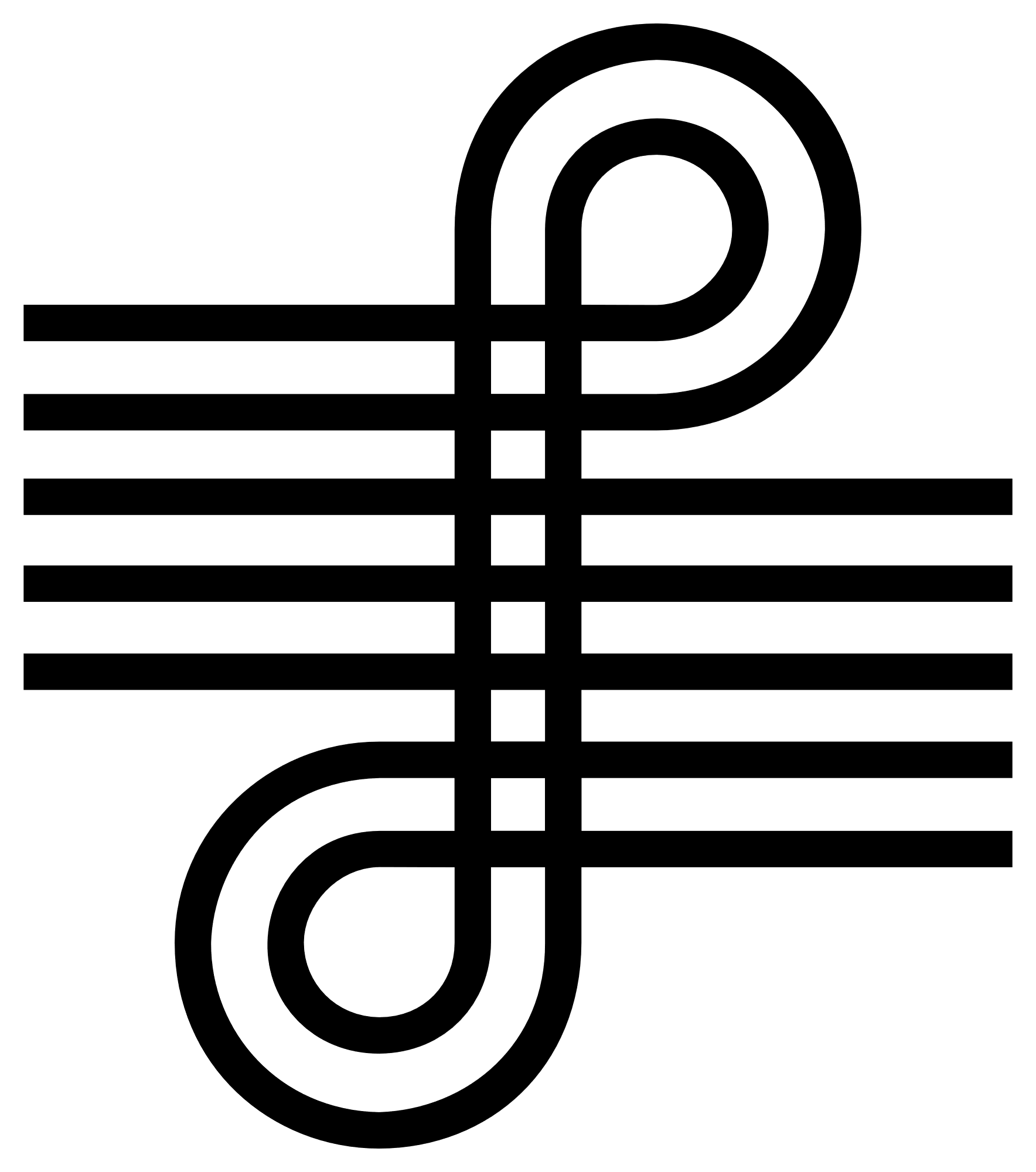
Hochschule für Musik Karlsruhe

IMWI - Institut für Musikinformatik und Musikwissenschaft

Bachelorarbeit zum Thema:

**Interaktion mit Audio-Daten in VR mithilfe von 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:

16.07.2020

Inhaltsverzeichnis

[2 Einleitung 1](#_Toc45809437)

[3 Grundlagen 2](#_Toc45809438)

[3.1 Virtual Reality 2](#_Toc45809439)

[3.1.1 Definition und Historie 2](#_Toc45809440)

[3.1.2 Anforderungen an VR-Software und -Hardware 6](#_Toc45809441)

[3.2 Diskrete Fourier-Transformation 10](#_Toc45809442)

[4 Praktische Implementation „VrAudioSandbox“ 11](#_Toc45809443)

[4.1 Repository und Systemvoraussetzungen 11](#_Toc45809444)

[4.2 Architektur und externe Bibliotheken 12](#_Toc45809445)

[4.2.1 Unity als Laufzeitumgebung 13](#_Toc45809446)

[4.2.2 SteamVR / OpenVR 13](#_Toc45809447)

[4.2.3 NAudio Bibliothek 14](#_Toc45809448)

[4.2.4 DFT mit FFTW3 14](#_Toc45809449)

[4.2.5 FFTWSharp 15](#_Toc45809450)

[4.2.6 CurvedUI 15](#_Toc45809451)

[4.3 Implementation in Unity 15](#_Toc45809452)

[4.3.1 Import und Fast-Fourier-Transformation 16](#_Toc45809453)

[4.3.2 Visualisierung der Spektrum-Daten durch Meshes 19](#_Toc45809454)

[4.3.3 Manipulation der Spektrum-Meshes 21](#_Toc45809455)

[4.3.4 Inverse Fast-Fourier-Transformation, Wiedergabe und Export 23](#_Toc45809456)

[4.3.5 User Interface und Steuerung 27](#_Toc45809457)

[5 Schlussbetrachtung 34](#_Toc45809458)

[6 Literatur III](#_Toc45809459)

[7 Sonstige Quellen V](#_Toc45809460)

[8 Abbildungsverzeichnis VII](#_Toc45809461)

[9 Code-Listing-Verzeichnis VIII](#_Toc45809462)

[10 Eidesstattliche Erklärung IX](#_Toc45809463)

[11 Anhang – Programmcode X](#_Toc45809464)

# Einleitung

Der Traum der Virtual Reality (VR) begleitet die Geschichte unserer technischen Entwicklung schon lange und scheint seit Kurzem durch immer leistungsfähigere und kostengünstigere Technik kurz vor dem Durchbruch für ein breiteres Publikum zu stehen. Abseits vom Medienkonsum durch Computerspiele und sichtfeldfüllenden Filmen stellt sich die Frage, ob VR auch unsere Art und Weise wie wir mit dem Computer arbeiten zu verändern vermag. VR bietet das Potential unserem Geist direkten und erlebbaren Zugriff auf jegliche Form digitaler Daten zu geben und unsere Kreativität zu beflügeln.

In dieser Arbeit sollen die Möglichkeiten der Interaktion mit Audio-Daten im virtuellen Raum am Beispiel der Darstellung und Manipulation von 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 dieser Arbeit ist die praktische Implementation des Konzepts Audio-Daten durch diskrete Fourier-Transformation im virtuellen Raum erlebbar und manipulierbar zu machen. Dabei spielen technische Herausforderungen und praktische Implementationsdetails ebenso eine Rolle wie die Entwicklung eines Bedienkonzeptes und die Frage, ob VR für diesen Anwendungsfall einen Mehrwert bietet.

# 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 fotorealistische Darstellung oder ein realistisches Szenario innerhalb der virtuellen Welt, sondern eine passende 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.

Ein zentraler Aspekt ist die Präsenz. Dieses Konzept ist im Zusammenhang mit der Telerobotik entstanden und beschreibt das Gefühl sich innerhalb der virtuellen Umgebung zu befinden und auf diese so zu reagieren, als ob es sich um die reale Umgebung handeln würde. Weiter präzisieren lässt sich dieser Aspekt in drei Teilaspekte (entsprechend der Terminologie von Slater, 2009)[[4]](#footnote-5):

Die Ortsillusion beschreibt das Gefühl, sich am dargestellten Ort zu befinden. Dies wird insbesondere durch eine korrekte blickwinkelabhängige Darstellung, die unmittelbar auf die Drehung des Kopfes reagiert, unterstützt.

Die Plausibilitätsillusion bezieht sich eher auf den Inhalt bzw. auf die Ereignisse innerhalb der virtuellen Umgebung. Entscheidend ist nicht die Präsentation, sondern die Glaubwürdigkeit von Ereignissen – insbesondere, wenn diese ohne Zutun des Rezipienten geschehen.

Ähnlich verhält es sich bei der Involviertheit, welche das Interesse und die Aufmerksamkeit des Nutzers beschreibt. Auch hier spielt weniger die Qualität der Darstellung eine Rolle. Eine niedrige Involviertheit würde sich demnach als „Langeweile“ beim Nutzer manifestieren.

Ein VR-System entspricht demnach einem Gesamtkonzept aus Hard- und Softwarekomponenten und fungiert als Mensch-Maschine-Schnittstelle. Die Wahrnehmung bestimmt die Realität. 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[[5]](#footnote-6), bei der der Eindruck des räumlichen Sehens durch zwei blickwinkelverschiedene Bilder hervorgerufen werden kann, ist eine noch heute relevante Technik.

Morton Heilig baute 1962 einen VR-Automat, den er „Sensorama“ nannte. Durch Stereoskopie, Stereo-Sound, Rüttelmechanik, sowie einem Wind- und Geruchsystem konnte eine Einzelperson für 25 Cent eine Auswahl von etwa zehn-minütigen immersiven Filmen betrachten (u.a. eine Motorradfahrt durch New York, die Fahrt in einem Strandbuggy oder der Tanz einer exotischen Bauchtänzerin[[6]](#footnote-7)).

Abbildung 1 – Fotografie „Sensorama“, 1962, Copyright Morton Heilig – <http://www.medienkunstnetz.de/assets/img/data/3331/bild.jpg>, letzter Abruf: 08.07.2020

Heilig, der für die Filmindustrie als Kameramann arbeitete, hatte dabei die Vision des „Experience Theaters“. Riesige Kino-Säle – ausgestattet mit Sensorama-Technik und einer sichtfeld-füllenden Leinwand – sollten „The Cinema of the Future“ bilden.[[7]](#footnote-8)

Dabei hat sich Heilig detailliert mit dem menschlichen Sichtfeld, dem Zusammenspiel mit anderen Sinneswahrnehmungen und der Steuerung der Aufmerksamkeit der Rezipienten beschäftigt.[[8]](#footnote-9) All dies war ohne Computer möglich – allerdings als passive Erfahrung ohne Interaktivität.

Vom heutigen 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.[[9]](#footnote-10) Dies legte die Grundlagen zum einen für die stereoskopische Darstellung des im Computer erzeugten Raums in Vektorgrafik, zum anderen für die notwendige schnelle Synchronisierung der angezeigten Bilder mit den Kopfbewegungen des Benutzers. Im Gegensatz zur passiven Sensorama-Erfahrung ist hierbei das angezeigte Bild an die physische Blickrichtung des Rezipienten gekoppelt. Ein Sensor mit mechanischem Arm hat zunächst die Kopfposition ermittelt. Später wurde dies für bessere Bewegungsfreiheit mithilfe von Ultraschall-Emittern und -Empfängern umgesetzt.[[10]](#footnote-11) 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.

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.

VR wird in Wissenschaft und Forschung, beim Militär, im Ingenieurwesen, im Bereich der Architektur, in der Medienkunst und zunehmend im Unterhaltungssektor eingesetzt. Die Firma Virtuality führte 1991 erste VR-Spielautomaten in Spielhallen ein, die u.a. vernetzte Multiplayer-Partien ermöglichten. Produkte für den Privatgebrauch wie der Virtual Boy von Nintendo, der 1995 erschien und keinen kommerziellen Erfolg brachte, waren allerdings technisch unausgereift. Die Kosten für eine überzeugende und immersive Erfahrung wären mit dem damaligen Stand der Technik zu hoch gewesen. So hatte der „Virtual Boy“ ein monochromes Display auf Basis von roten LEDs, konnte nicht am Kopf befestigt, sondern nur in unbequemer Haltung auf den Tisch stehend benutzt werden und hat schnell zu Kopfschmerzen und anderen physiologischen Belastungen geführt. Es ist außerdem fragwürdig, ob es sich beim „Virtual Boy“ trotz entsprechendem Marketing um eine VR-Erfahrung handelt. Konzeptionell entsprachen die Spiele eher den im Markt etablierten 2D-Spielen, die zusätzlich mit Tiefeninformationen angereichert waren und wie zuvor in der Branche üblich mit einem Gamepad gesteuert wurden.[[11]](#footnote-12)

Beim „Sega VR“-Headset der Firma Sega handelte es sich um ein am Kopf befestigtes HMD mit Tracking-Sensoren und Kopfhörern, das aber nie zur Marktreife entwickelt wurde. Zu groß waren die Probleme bzgl. Überanstrengung der Augen und Motion-Sickness.[[12]](#footnote-13) Auch war das Sichtfeld sehr klein, so dass der Eindruck entstand, auf einen entfernten rechteckigen Monitor zu schauen – ohne ein blickfeldfüllendes immersives Gefühl.

Erst ab etwa 2013 wurde die Technik leistungsfähiger und damit insbesondere auch günstiger, so dass sich eine breitere Masse HMDs und andere Technologien leisten konnte. Zuvor waren die Preise – u.a. auch für die Workstations, die leistungsfähig genug waren die Grafiken für die HMDs oder Projektionsflächen zu erzeugen – astronomisch hoch. Somit waren VR-Erfahrungen in der Regel der Industrie, den Universitäten oder dem Militär vorbehalten. Hervorzuheben sind dabei die HMDs „Rift DK 1“ und „Rift DK 2“ der Firma Oculus, welche zu einem erneuten VR-Boom führten, bei dem die Hürde für den Einstieg des Konsums oder der Programmierung von VR-Erfahrungen signifikant gesenkt wurde. Andere Hersteller folgten, so dass es ab etwa 2016 eine große Auswahl an erschwinglicher VR-Hardware für den Privatgebrauch gibt.[[13]](#footnote-14)

Das VR-System „HTC Vive“, welches in Kooperation der Firmen HTC und der Valve Corporation entstanden ist, erschien 2016 für den Endkunden und war gebaut aus einem Set bestehend aus dem HMD, zwei Motion-Tracking-Controllern, sowie zwei Infrarot-Laser-Basisstationen. Der Fokus lag hier auf einer Komplettlösung, die sogenanntes Room-Scale-VR mit sehr präzisem Positional-Tracking für dieses Preissegment realisierte. Room-Scale-VR bedeutet dabei, dass die Dimensionen des realen Raums auf den virtuellen Raum übertragen werden. Dies ermöglichte andere Konzepte als bspw. vorher verbreitete VR-Anwendungen, die primär im Sitzen genutzt wurden (u.a. Cockpit-VR-Erfahrungen für Autorennen- oder Flugsimulatoren).

Ein mit VR verwandtes Feld ist die sogenannte Augmented Reality (AR) bei der die echte Realität mit zusätzlichen Informationen angereichert wird. Im Gegensatz zur VR werden reales Umfeld und virtuelle Bestandteile 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 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 VR anwenden. Speziell 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 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 des Wandels etablierter Konzepte 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 2 – unterer Teil; oberer Teil: 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. Zusätzlich ist durch die Nutzung eines HMDs die Bedienung über die traditionellen Eingabegeräte unpraktikabel. 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. Motion-Tracking des gesamten Körpers – inklusive fein-motorischer Details – stellt dabei den Idealfall dar.

Abbildung 2 – 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 Copyright 2019 Valve Corporation, letzter Abruf 17.06.2020

Sowohl die Eingabe, als auch die Ausgabe von VR-Anwendungen sind also besonders immersiv und intuitiv, wenn sie konzeptionell nach dieser Physikalität gestaltet sind. Dies kann z.B. noch mit taktiler Rückmeldung (Force-Feedback) verstärkt werden.

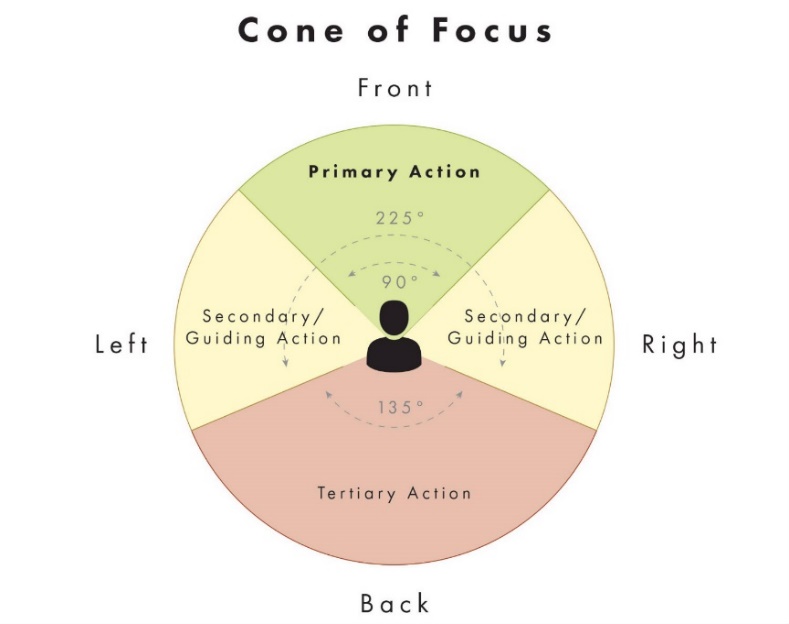
Dadurch, dass der Benutzer seine Blickrichtung jederzeit frei wählen kann bzw. muss, ergeben sich Anforderungen an die Steuerung der Aufmerksamkeit durch das Design der VR-Applikation. Das Cone-Of-Focus-Modell (siehe Abbildung 2) teilt dabei das Umfeld des Spielers in verschiedene Zonen ein.

Abbildung 3 – 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

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.[[14]](#footnote-15) 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.

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.[[15]](#footnote-16)

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.[[16]](#footnote-17) Experimente am NASA Ames Research Center[[17]](#footnote-18) 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.[[18]](#footnote-19) 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. Auch können Effekte wie Lichtempfindlichkeit oder Epilepsie, die auch bei zwei-dimensionaler Darstellung auftreten, durch das große Blickfeld und die Immersion verstärkt werden.

## 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

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.

Zusätzlich findet sich der Programm-Code der selbst-geschriebenen Klassen unter „Anhang-Programmcode“.

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 acht 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. Die Anforderungen an den Arbeitsspeicher sind hoch: VrAudioSandbox benötigt nach dem Laden einer Zehn-Sekunden-langen Audio-Datei mit einer Samplingrate von 48 kHz und 16 Bit etwa 700 Megabyte Arbeitsspeicher. Wird diese Datei innerhalb der Unity-IDE geladen, ist der Speicherverbrauch noch höher.

## 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.

Abbildung 4 – Eine abstrakte Darstellung der Architektur - Eigene Darstellung

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.

### Unity als Laufzeitumgebung

Die Laufzeit- und Entwicklungsumgebung Unity ist eine weitverbreitete Spiele-Engine mit etwa 50% Marktanteil für den Bereich PC/Konsole/Mobile Games.[[19]](#footnote-20) 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.[[20]](#footnote-21) Momentan (Stand: Juni 2020) unterstützt Unity 18 verschiedene Zielplattformen.[[21]](#footnote-22)

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.[[22]](#footnote-23) Alternativ steht das Scripting-Backend IL2CPP (Akronym für 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 (Akronym für 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 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[[23]](#footnote-24), welches eine Vielzahl von Komponenten mitliefert, die direkt in Unity genutzt werden können. Essentielle Funktionen wie das Tracking des HMD und der Motion-Controller im Raum werden durch direkt nutzbare Unity-Komponenten bereitgestellt. Zusätzliche Funktionen wie etwa das Greifen von Objekten, Teleportation oder ein mit Raycasting nutzbarer Laser-Pointer können bei Bedarf hinzugefügt werden. Um diese Funktionalität in der Praxis sinnvoll nutzen zu können, sind jedoch trotzdem eigene Erweiterungen oder Modifikationen in Form von Programmcode notwendig – insbesondere, wenn das Verhalten dieser Funktionen nicht den Vorgaben entsprechen soll oder komplexere Bedienparadigma umgesetzt werden sollen. Eine Abstraktionsschicht für die Steuerung durch externe Controller vereinfacht außerdem die Unterstützung verschiedener Eingabegeräte der etablierten Hersteller.

### NAudio Bibliothek

NAudio ist eine von Mark Heath entwickelte Open-Source-Audio-API und Bibliothek für .NET und ist in C# geschrieben.[[24]](#footnote-25) 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.[[25]](#footnote-26) 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.[[26]](#footnote-27)

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:[[27]](#footnote-28)

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.[[28]](#footnote-29) 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[[29]](#footnote-30) 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 das Kaptiel „11 Anhang - 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 zwei Bytes ein Audio-Frame, bei 24-Bit-Audio drei 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[[30]](#footnote-31):

Die Frequenz-Auflösung eines Bins beträgt[[31]](#footnote-32):

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.[[32]](#footnote-33) 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

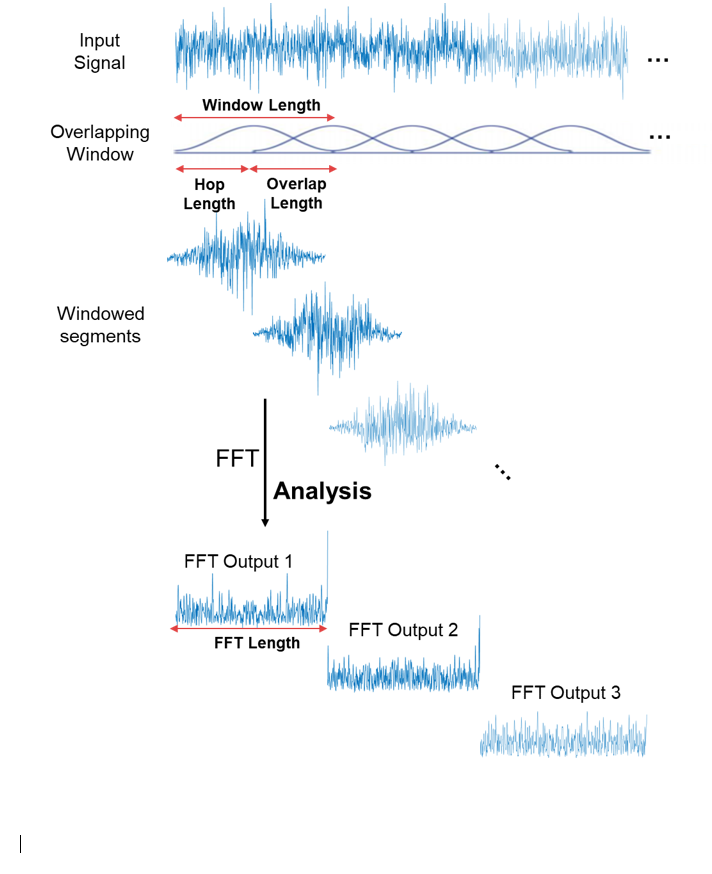
Die FFT wird jetzt für jedes dieser überlappenden Segmente durchgeführt. Da FFTW für die FFT „dft\_1d“ komplexe Zahlen erwartet, wird dazu ein double-Array mit doppelter Größe initialisiert und alle gerade Indizes mit den Amplituden-Werten gefüllt. Die ungeraden Indizes enthalten laut FFTW-Konvention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

Abbildung 5 – 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

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 nachher 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.[[33]](#footnote-34)

### Visualisierung der Spektrum-Daten durch Meshes

Aus Performance-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[[34]](#footnote-35)), führte jedoch zu extremen Performance-Einbußen und teilweise zum Absturz von Unity.

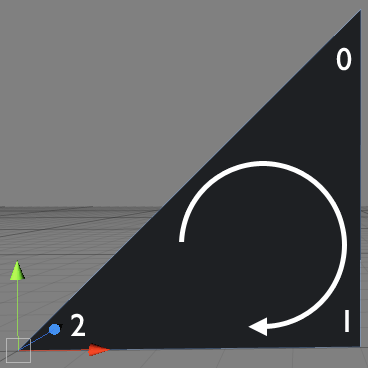
Zur Erzeugung eines dieser Meshes müssen zunächst Vertices erstellt werden. Dazu wird zunächst ein Raster aus Vertices kreiert, die später die Grundfläche der einzelnen „Pyramiden“ ergeben, welche die Werte des Amplituden-Spektrums repräsentieren. Im Anschluss werden die Vertices für die Spitzen abhängig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

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

Damit eine sichtbare Visualisierung entsteht, müssen nun je drei 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.[[35]](#footnote-36)

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 benutzerdefinierten 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 erleichtert die im Folgenden beschriebene Logik zur Modifizierung der Spektrum-Meshes.

In der Klasse „ToolHandler.cs“ wird ausgehend vom Schnittpunkt und dem gewählten Radius 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 einige Zeit dauert. 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.[[36]](#footnote-37) Dies ist wichtig für die Einhaltung einer genügend hohen Framerate. 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. Es muss also ein Kompromiss zwischen Latenz und Verzögerung der Berechnung gefunden werden.

Zur drastischen Verringerung der Laufzeit für die Berechnung der zu verändernden Vertices wurde die entsprechende 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 das „Unity Job System“[[37]](#footnote-38), 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 in Unity bereits vorhandenen Worker-Threads mit echter Nebenläufigkeit und optimiert für die Anzahl der logischen CPU-Kerne ausgeführt. Der „Burst Compiler“ kompiliert den entsprechenden Code in nativen Code, der im Kontext vom Unity-Prozess ausgeführt wird – er läuft also nicht innerhalb der Mono-Laufzeitumgebung.[[38]](#footnote-39) 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 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.[[39]](#footnote-40) In diesem Zuge werden auch die FFT-Daten selbst, sowie die Mesh-Colliders und Mesh-Colors entsprechend angepasst.

Es wurde die Klasse „DeformJob“ definiert, welche als Warteschlange für die einkommenden Aufträge zur Veränderung des Meshes dient. In der Update-Methode wird diese Warteschlange kontinuierlich geprüft und für jeden Auftrag die Coroutine „DeformMeshMultiplePointsWorker“ gestartet, welche wiederum die oben beschriebenen Operationen und den multi-threaded Burst-Code ausführt.

Listing 5 zeigt den Burst-Code für das Multi-Threading.

**SpectrumDeformer.cs** 

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

### Inverse Fast-Fourier-Transformation, Wiedergabe und Export

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:[[40]](#footnote-41)

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.[[41]](#footnote-42)

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 als Matlab-Code vorgestellte Lösung von [Mathuranathan Viswanathan](https://www.linkedin.com/in/mathuranathan/) ist es, einen Schwellwert zu definieren und alle berechneten Phasenwerte unterhalb des Schwellwertes zu ignorieren.[[42]](#footnote-43) 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:[[43]](#footnote-44)

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 auf Wunsch als Audio-Datei exportiert werden.

### User Interface und Steuerung

Für die Visualisierung der Spektrum Daten wurde eine Darstellung gewählt, die alle Daten ähnlich dem Blick aus der Vogelperspektive auf eine Gebirgslandschaft darstellt. Dies sorgt für eine gute Übersicht und Orientierung. Das User Interface soll in den Hintergrund treten – im Zentrum der Aufmerksamkeit sollen die Audio-Daten selbst sein.

Die Prämisse für die Steuerung ist es, dass jegliche Interaktion innerhalb der VR-Umgebung stattfindet, um zu vermeiden, dass das HMD abgesetzt werden muss. Auch wird weitestgehend auf den Einsatz von Menüstrukturen und Unterebenen verzichtet bzw. diese sollen den Fokus auf die Visualisierung des Spektrums nicht stören. Neben der initialen Dateiauswahl gibt es nur ein Hauptmenü, um das Programm zu beenden oder eine neue Datei zu importieren. Außerdem kann dort die Skalierung der Spektrum-Daten bzgl. der y-Achse beeinflusst werden. Da diese Aktionen nicht versehentlich ausgelöst werden sollen bzw. im letzteren Fall selten benötigt werden, ist die Platzierung in einer Unterebene unproblematisch.

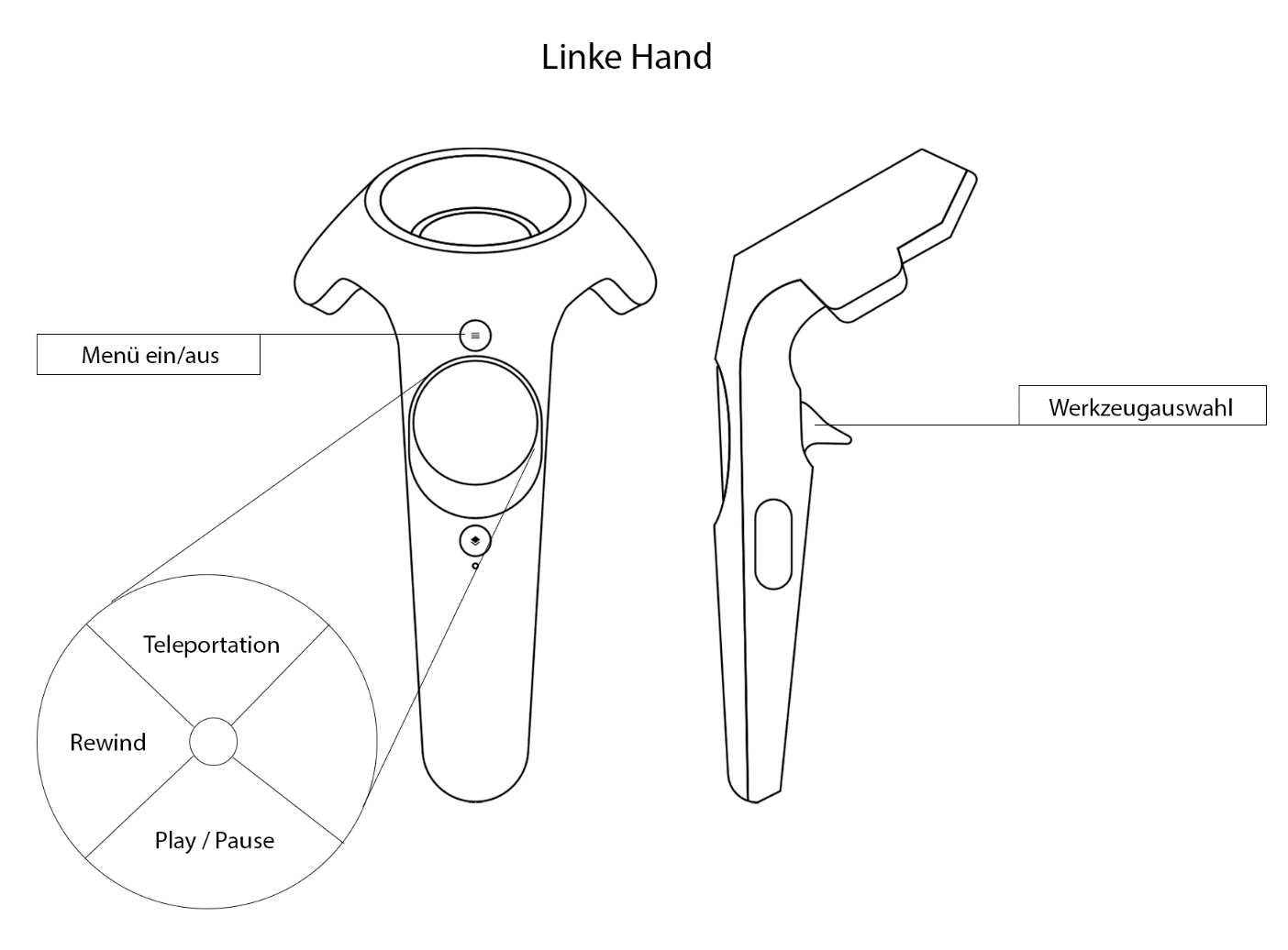
Die Vielzahl der Aktionen wird direkt über die Tasten der Motion-Controller gesteuert. Dabei ist die linke Hand hauptsächlich für die Navigation im Raum und die Steuerung der Wiedergabe zuständig, der rechten Hand sind beinahe alle Werkzeugfunktionen zugeordnet – mit der bewussten Ausnahme, dass die Werkzeugauswahl mit der linken Hand initiiert wird (Abbildungen 7 und 8 zeigen die Tastenbelegung).

Abbildung 7 – Tastenbelegung Controller – linke Hand – eigene Darstellung

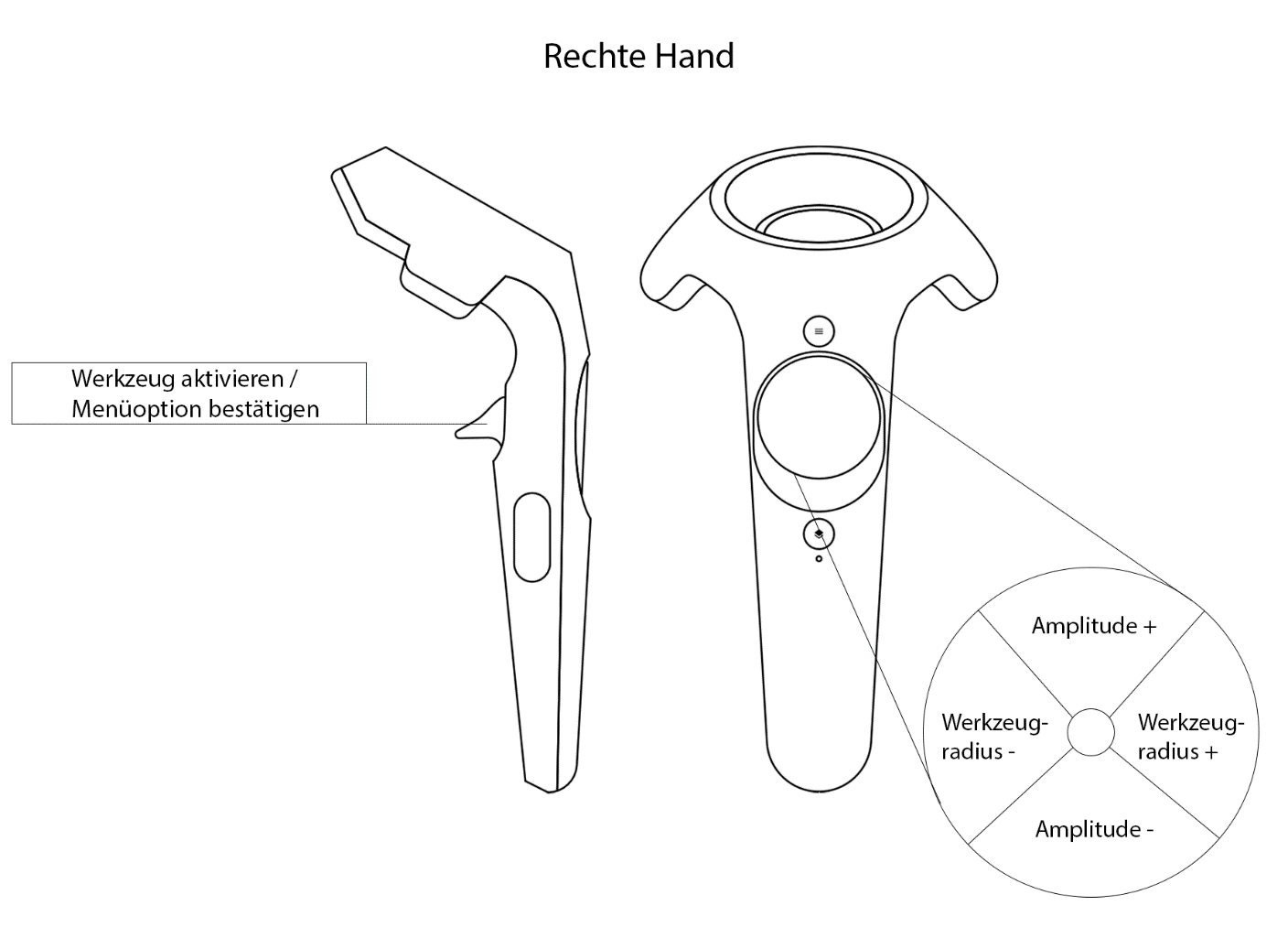
Zu Beginn wird der Dateiauswahldialog (Abbildung 9) als im Raum schwebender Canvas eingeblendet, der als einzig sichtbares Element die Blickrichtung und Aufmerksamkeit des Nutzers an die Stelle leitet, an der nach erfolgtem Import auch die Visualisierung des Spektrums sichtbar wird. Dieser Teil der Interaktion folgt der etablierten Steuerung einer grafischen Oberfläche mit der Maus – hierbei wird der rechte Controller mit eingeblendeten Laser-Pointer zum Zielen und Ausführen der gewünschten Aktion genutzt.

Abbildung 8 – Tastenbelegung Controller – rechte Hand – eigene Darstellung

Nun kann sich der Nutzer frei im Raum bewegen und die Visualisierung erkunden. Mit der linken Hand kann bei gedrückter Teleportationstaste im oberen Teil des Trackpads an eine beliebige Stelle teleportiert werden. Dazu wird als Orientierungshilfe ein Bogen eingeblendet, der den Zielpunkt angibt (siehe Abbildung 10).

Mit der linken Hand kann die Wiedergabe gestartet, pausiert oder zurückgespult werden. Die Spektrum-Daten werden während der Wiedergabe entsprechend der Abspielposition blau eingefärbt, um einen einfachen Abgleich der Visualisierung mit der auditiven Wahrnehmung zu ermöglichen.

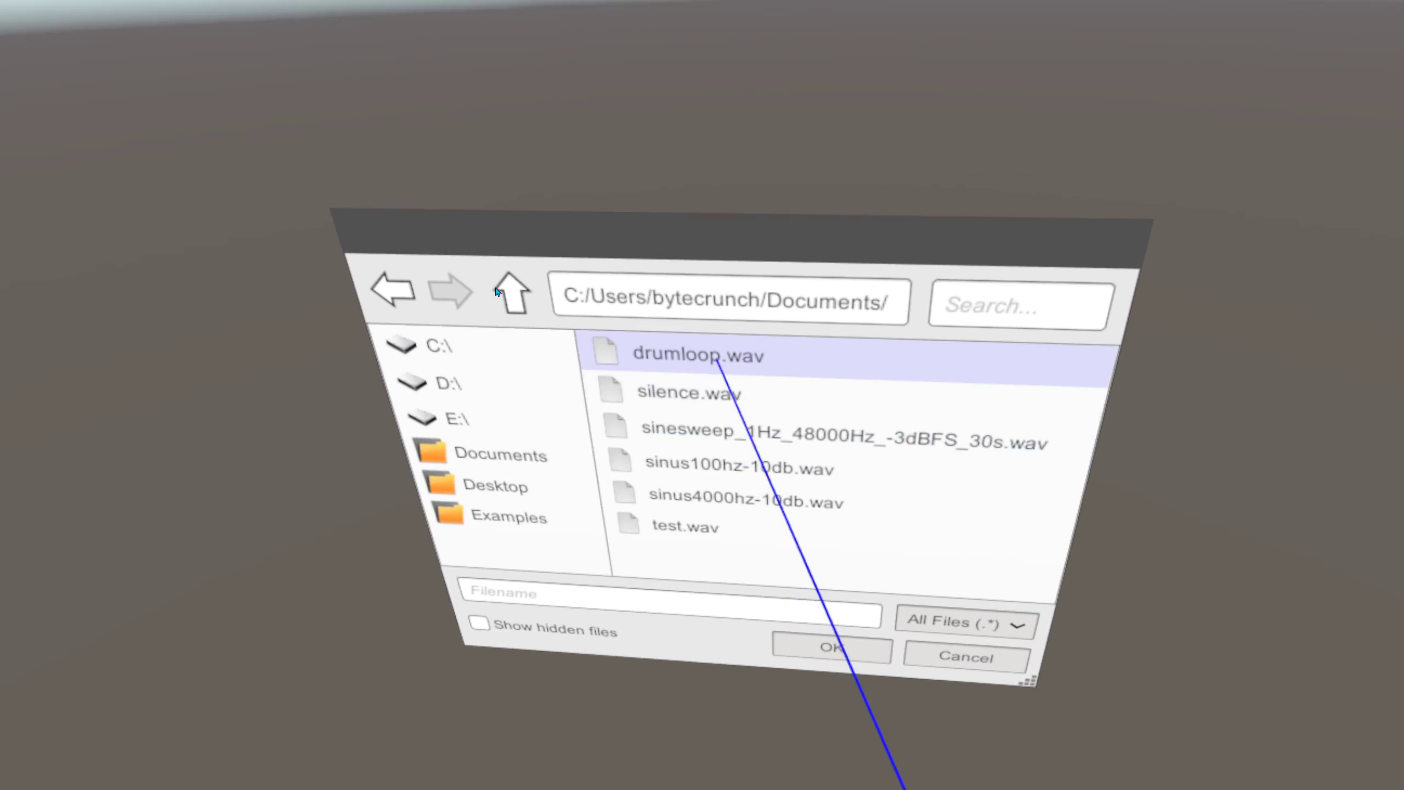
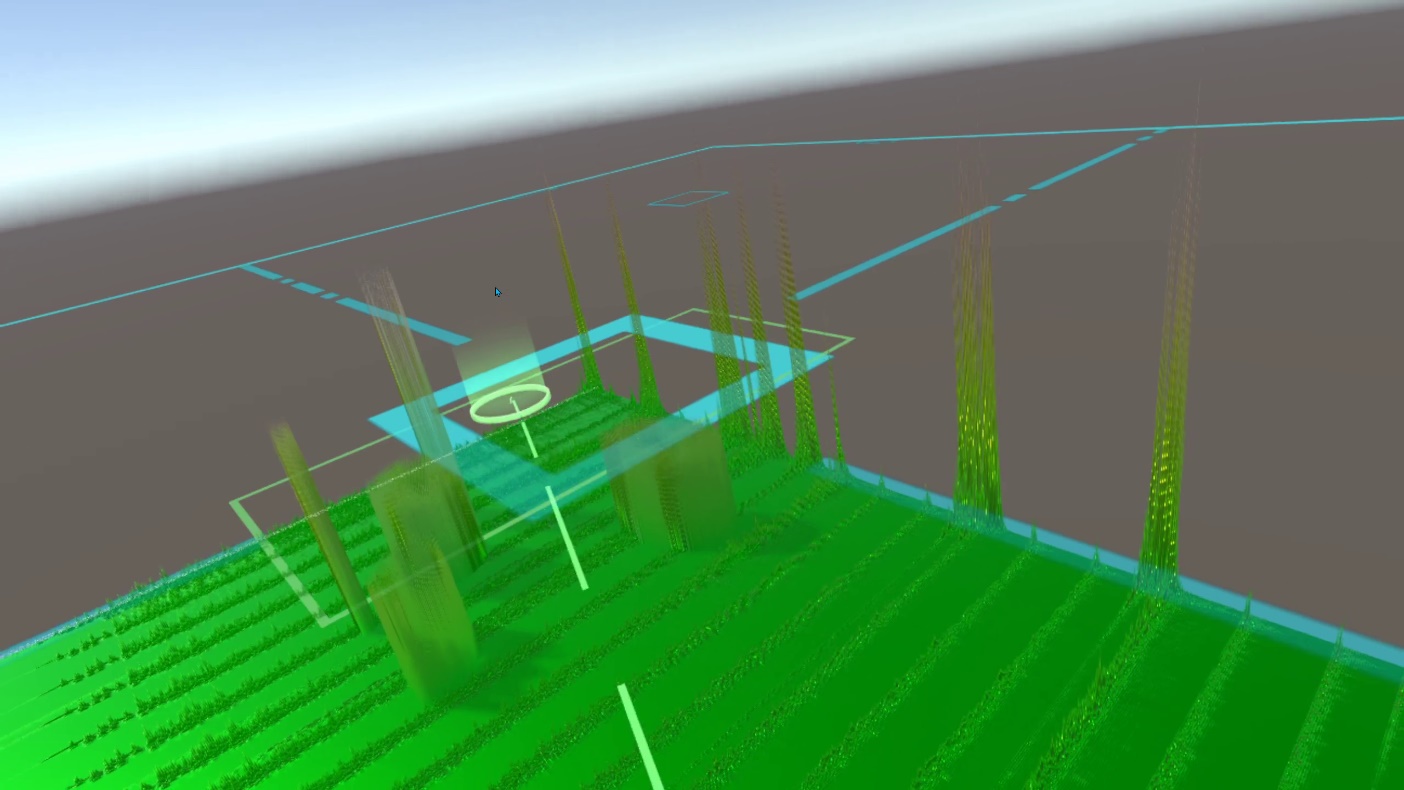


Abbildung – Teleportation – selbst erstellter Screenshot

Abbildung – Dateiauswahldialog – selbst erstellter Screenshot

Die Werkzeuge sind verschiedenfarbig und die individuelle Farbe spiegelt sich auch in der Farbe des Laser-Pointers wider, um die aktuelle Werkzeugauswahl zu verdeutlichen. Das Wechseln des Werkzeuges findet bei gedrücktem Trigger des linken Controllers durch Seitwärts-Bewegungen des Handgelenkes statt. Somit kann die rechte Hand weiter auf den zu bearbeitenden Bereich zielen, ohne, dass nach dem Werkzeugwechsel eine Neuausrichtung ausgeführt werden muss. Während der Auswahl wird im linken peripheren Bereich ein kleines Overlay in Form eines World-Space-Canvas eingeblendet, das den Hauptsichtbereich frei lässt. Dieses verschwindet nach Beendigung der Werkzeugauswahl durch Loslassen des Triggers. Dort finden sich nach dem Prinzip des Skeuomorphismus gestaltete Icons für die verschiedenen Werkzeuge.

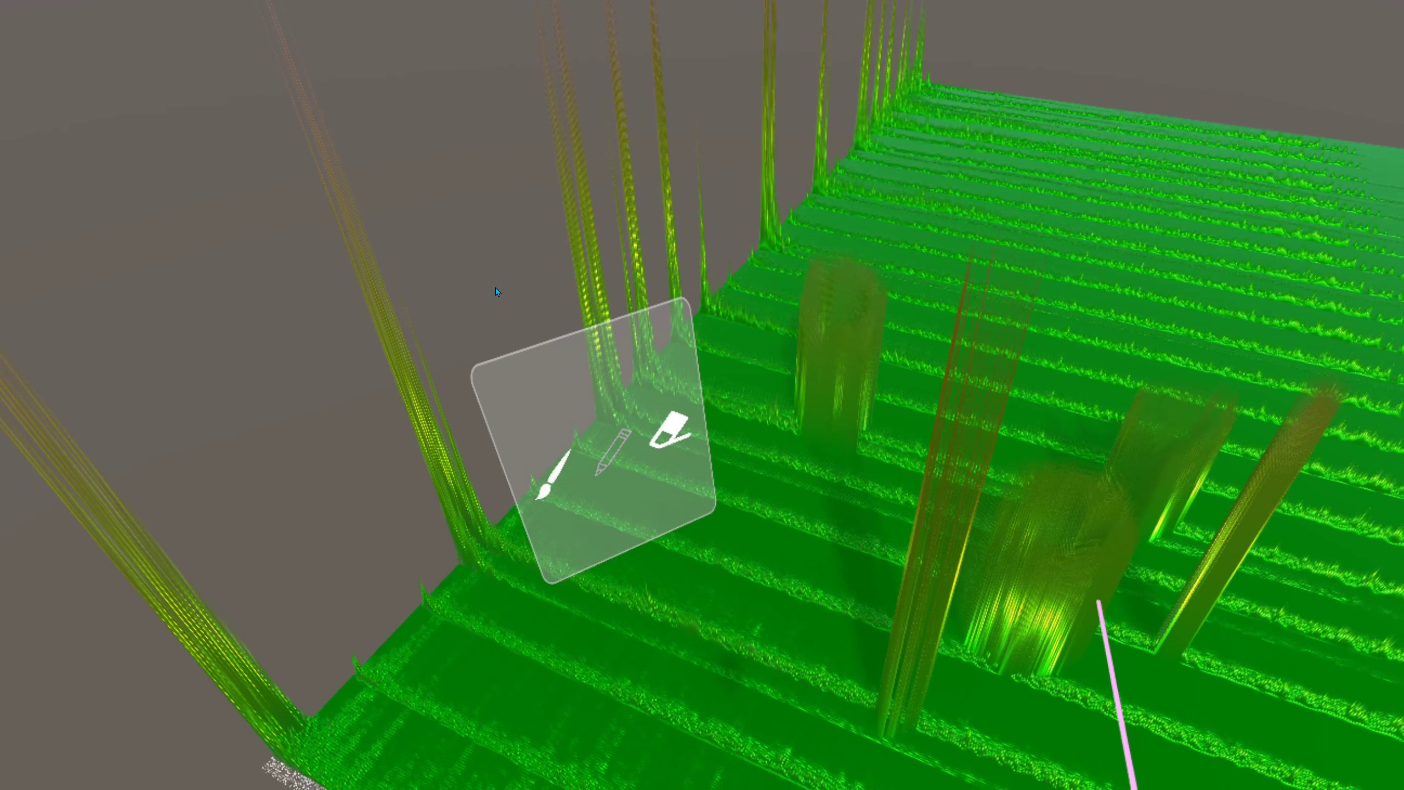
Der Werkzeugradius wird über den linken und rechten Teil des Trackpads der rechten Hand direkt verkleinert und vergrößert. Die Dicke des Laser-Pointers spiegelt den gewählten Radius wider. Über den unteren und oberen Bereich des Trackpads lässt sich der absolute Wert der Amplitude für das Zeichnen im Spektrum festlegen. Dies wird innerhalb des Spektrums mit einer transparenten planen Fläche visualisiert (siehe Abbildung 12).

Abbildung 11 – Wechseln des Werkzeuges – selbst erstellter Screenshot

Beim Pinsel- und Radierwerkzeug kann der Trigger gedrückt gehalten werden, um Veränderungen auf einen größeren Bereich anzuwenden. Dies wird visuell durch eine rote Linie unterstützt (siehe Abbildung 13). Erst danach wird die eigentliche Manipulation der Daten angestoßen.

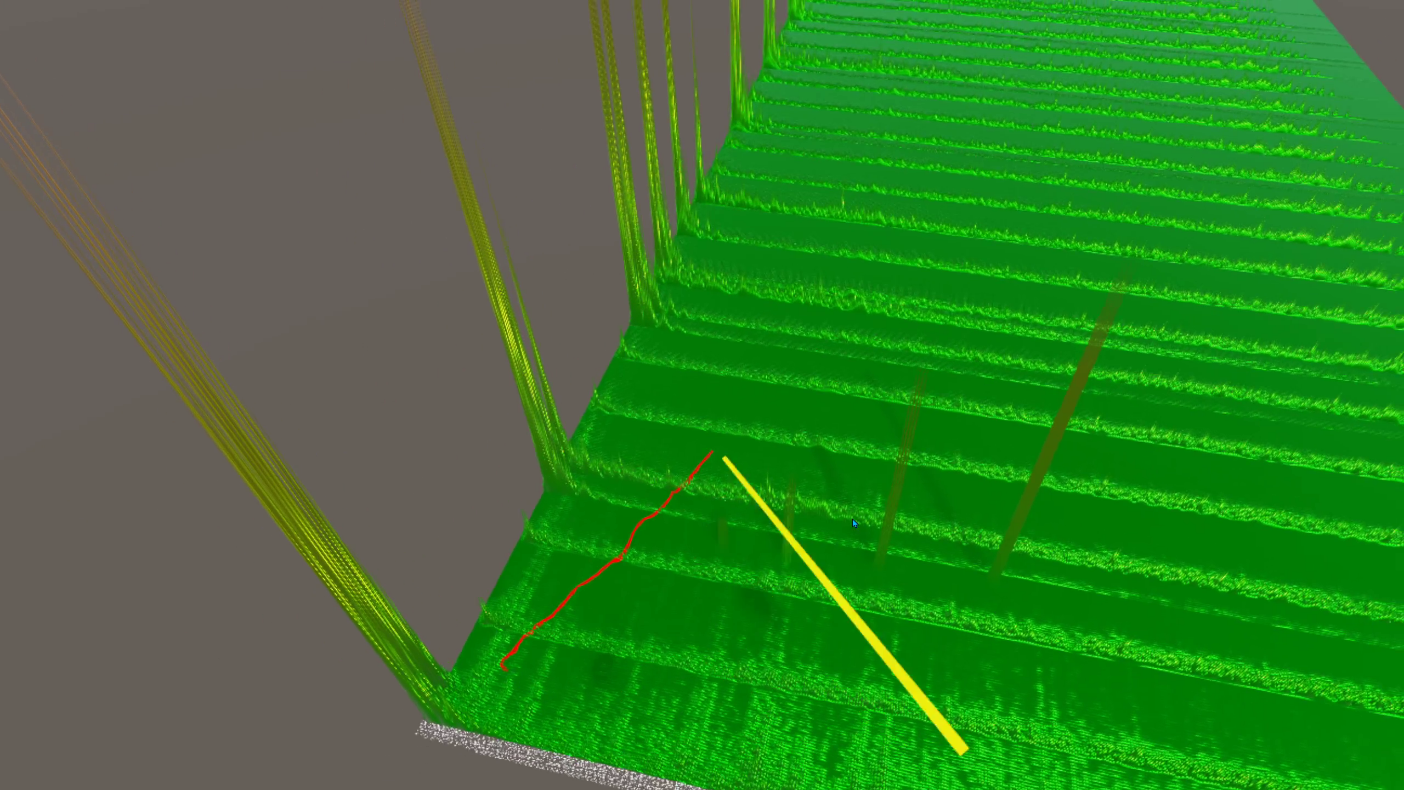
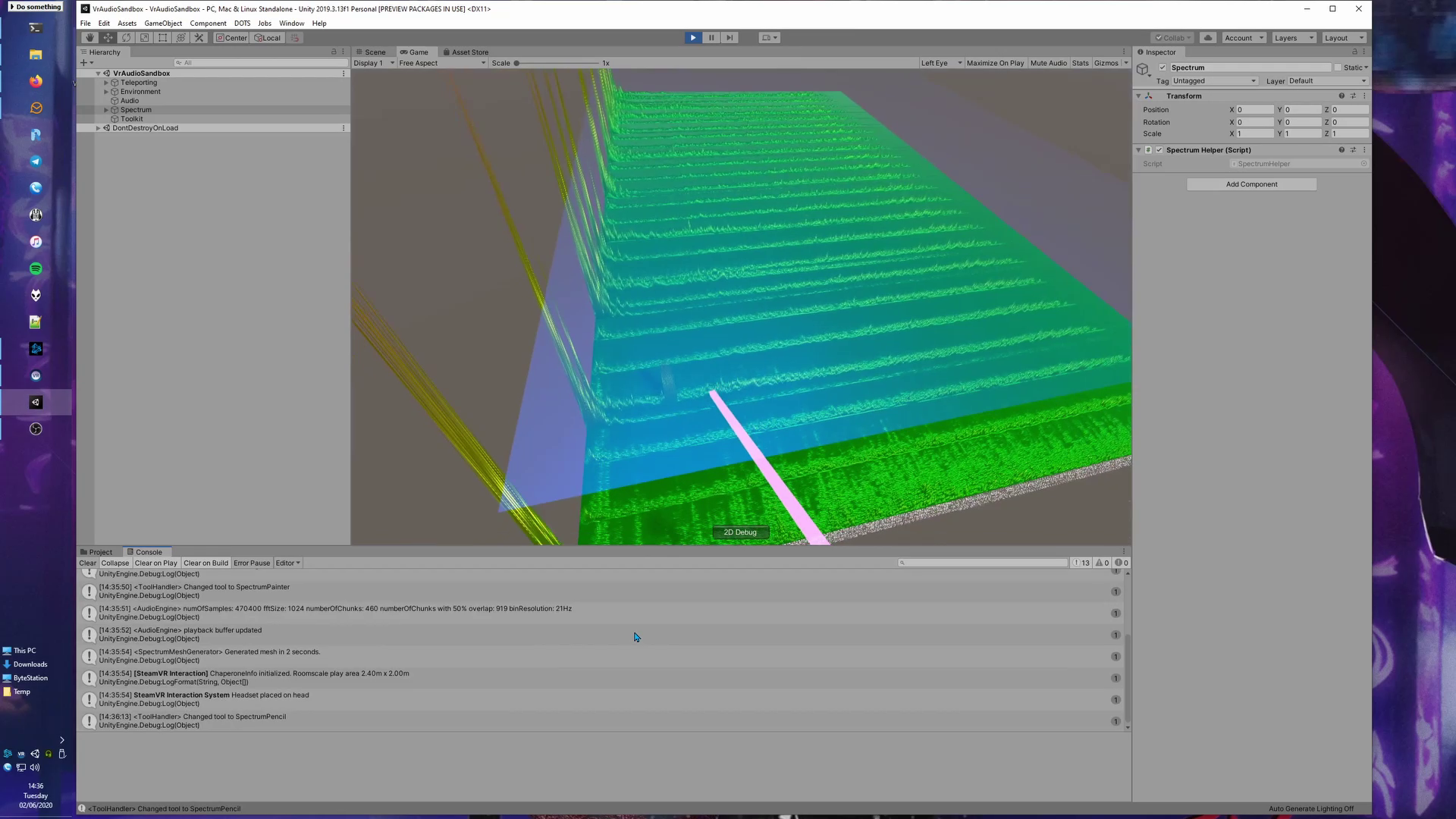


Abbildung 12 –Einstellung der Amplitude – selbst erstellter Screenshot

Abbildung 13 – Visualisierung der gezeichneten Spur – selbst erstellter Screenshot

Über das Hauptmenü kann eine andere Audio-Datei geladen werden, oder ein Export der aktuell vorliegenden Version im Speicher veranlasst werden. Außerdem findet sich dort der Punkt, um das Programm zu beenden. Das Hauptmenü (Abbildung 14) wird bei Bedarf immer mittig im Sichtfeld des Nutzers eingeblendet und steht somit an jeder Position zur Verfügung.

Abbildung – Hauptmenü – selbst erstellter Screenshot

# Schlussbetrachtung

Die praktische Implementation des Konzepts Audio-Daten in VR zu manipulieren ist funktionsfähig und bietet Möglichkeiten, Erkenntnisse bzgl. des Bedienkonzeptes und der Wirkung auf den Nutzer zu sammeln.

Obwohl die Möglichkeiten der Manipulation der Audio-Daten sowohl technisch, als auch künstlerisch in der jetzigen Form noch begrenzt sind, stellt die Art und Weise der Interaktion in VR einen Mehrwert dar. Das Arbeiten in VR ist motivierender und es herrscht weniger Distanz zwischen Nutzer, Daten und den Manipulationswerkzeugen als bei traditioneller Software, die zweidimensional auf einem Bildschirm angezeigt wird. Die Bedienung fühlt sich natürlicher an und tritt dabei stellenweise in den Hintergrund. Insbesondere das Navigieren durch die Audio-Daten, welches traditionell viel feinmotorische Nutzereingaben verlangt (Scrolling durch die Wellenform, Zoom-Level, Selektion einzelner Parts etc.). Die Gestaltung in Form einer Room-Scale-VR-Erfahrung motiviert den Nutzer die Audio-Daten „zu Fuß“ zu erkunden. Die Navigation findet so unbewusst statt und es entsteht ein Raumgefühl für die einzelnen Bestandteile der importierten Audio-Datei – sowohl bezogen auf die verschiedenen Frequenzbereiche, als auch die zeitliche Abfolge der Samples. Der Aspekt der Begehbarkeit vereinfacht das Erfassen aller Klanginformationen und lässt trotzdem den Blick auf Details zu. Durch die Teleportationsmechanik muss nicht zwangsläufig viel Wegstrecke zurückgelegt werden. Das Begehen der Spektrum-Daten geschieht intuitiv punktuell und beim Betrachten oder Manipulieren von Details.

Der Aspekt, dass die Bedienung stellenweise intuitiv und auf natürliche Weise geschieht, vergrößert den Fokus auf die Daten selbst und könnte somit förderlich für das kreative Arbeiten im kompositorischen Kontext oder im Bildungssektor sein. Auch wenn die Komplexität der vorgestellten praktischen Lösung nicht ansatzweise an den Funktionsumfang professioneller Audio-Software heranreicht, scheint das Konzept der Physikalität beim Umgang mit den Audio-Daten eine Faszination und geringe Einstiegshürde darzustellen. Dies ließ sich auch bei (einer nicht repräsentativen Anzahl von) Probanden feststellen, die die Software ausprobiert haben und nicht mit Audio-Bearbeitungswerkzeugen vertraut waren. Zwar musste für das Bedienkonzept – insbesondere die Tastenbelegung – eine kurze Einweisung gegeben werden. Allerdings erfolgten wenige Probleme oder Rückfragen bzgl. der Bedienung.

Die Art der Interaktion ist so angenehm, dass schnell der Wunsch nach möglichen Verbesserungen und Erweiterungen der Funktionalität aufkommt:

Beim Experimentieren wäre eine Funktion zum „Rückgängig-Machen“ der letzten Aktionen hilfreich wie sie bspw. in Text- oder Grafikbearbeitung üblich ist.

Die Unterstützung von Mehrkanal-Audio oder das gleichzeitige Öffnen mehrerer Audio-Dateien zum Vergleich oder Datenaustausch wäre eine mögliche Verbesserung. Wünschenswert wären außerdem noch weitere Bearbeitungswerkzeuge, um das Spektrum der möglichen Manipulationen der Audio-Daten zu erweitern. Denkbar wäre bspw. eine Selektion einzelner Mesh-Bestandteile ähnlich wie bei Fotobearbeitungssoftware um Teile des Spektrums zu kopieren, auszuschneiden und wieder an anderer Stelle einzusetzen. In einer Art Bibliothek könnten diese Bausteine auch für eine spätere Verwendung zwischengespeichert werden.

Eine andere Idee wäre eine netzwerkfähige Mehrbenutzer-Funktionalität, um mit anderen Nutzern Daten und Ideen auszutauschen oder kollaborativ zu arbeiten.

Die Anzeige einiger technischer Daten könnte noch geschickter in das User Interface eingebunden werden – bspw. das Ablesen des Frequenzbereiches eines angepeilten Bereichs im Spektrum.

Eine größere Palette an Fensterfunktionen und die Konfiguration der FFT-Parameter zur Laufzeit könnten je nach Charakteristik der Audio-Daten eine optimalere Visualisierung ergeben.

Die Systemanforderungen sind momentan noch sehr hoch, aber solange man sich auf Audio-Dateien mit kurzer Länge beschränkt, ist es mit Consumer-Hardware möglich flüssig in VR zu arbeiten. Zukünftige – und bereits in der Beta-Version angedeutete – Änderungen in der Architektur des Unity-Job-Systems wären ein guter Ausgangspunkt, um insbesondere die Performance beim Modifizieren der Meshes zu erhöhen. Durch weitere Optimierungen wie z.B. effizientere Datenstrukturen oder eine Datenkompression könnten jedoch Wartezeit und Bildraten weiter verbessert werden.

Das Konzept, im virtuellen Raum mit Audio-Daten zu arbeiten, erscheint zukunftsfähig und das Gefühl von Immersion und Präsenz wirkt motivierend auf die Arbeit. Die praktische Implementation war bzgl. der technischen und theoretischen Herausforderungen spannend und deutet als Machtbarkeitsstudie ein großes Potential für VR-Software an, die abseits vom reinen Konsum von Medieninhalten neue Erfahrungen der Interaktion ermöglicht.

„I am convinced that this is not going to be an extension of cinema or 3-D cinema or video games. It is something new, different, and not experienced yet. The strange thing here is that normally, in the history of culture, we have new stories and narrations and then we start to develop a tool. Or we have visions of wondrous new architecture—like, let’s say, the museum in Bilbao, or the opera house in Sydney—and technology makes it possible to fulfil these dreams. So you have the content first, and then the technology follows suit. In this case, we do have a technology, but we don’t have any clear idea how to fill it with content.”

(Chris Milk, Founder of VRSE)

# Literatur

**Dörner, Ralf; Broll, Wolfgang; Grimm, Paul; Jung, Bernhard**: „Virtual und Augmented Reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität“, Springer-Verlag, Wiesbaden (2019), ISBN 9783662588604

**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

**Heilig, Morton Leonard**: „El Cine del Futuro: The Cinema of the Future” – Presence, Volume 1, Number 3, 1992, S. 279-294

**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

**Preim, Bernhard; Dachselt, Raimund**: „Band 1: Interaktive Systeme – Grundlagen, Graphical User Interfaces, Informationsvisualisierung“, 2. Auflage, Springer, Heidelberg; Dordrecht; London; New York (2010), ISBN 9783642054020

**Preim, Bernhard; Dachselt, Raimund**: „Band 2: Interaktive Systeme – User Interface Engineering, 3D-Interaktion, Natural User Interfaces“, 2. Auflage, Springer, Heidelberg; Dordrecht; London; New York (2010), ISBN 9783642452468

**Slater, Mel**: „Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments”, Philosophical Transactions of the Royal Society (2009), 364, 3549–3557, doi:10.1098/rstb.2009.0138, EVENT Lab, Institute for Brain, Cognition and Behavior(IR3C), ICREA-University of Barcelona, 08035 Barcelona, Spain; Department of Computer Science, University College London, London WCIE 6BT, UK

**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

**Business Insider**: „The incredible story of the 'Virtual Boy' — Nintendo's VR headset from 1995 that failed spectacularly”, 26.03.2018 – https://www.businessinsider.com/nintendo-virtual-boy-reality-3d-video-games-super-mario-2018-3?IR=T, letzter Abruf: 13.07.2020

**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

**InvisionStudio**: „The History of Virtual Reality (VR)”, https://invisionstudio.com/the-history-of-vr-virtual-reality/, letzter Abruf: 13.07.2020

**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

**National Instruments**: „Understanding FFTs and Windowing”, https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf, letzter Abruf: 12.05.2020

**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

**YouTube**: „The Untold Story Of Virtual Reality On The Sega Genesis - The Unreleased The Sega VR Headset”, Wrestling With Gaming – https://www.youtube.com/watch?v=3UN\_pN9ZU8Y, letzter Abruf: 13.07.2020

# Abbildungsverzeichnis

Abbildung 1 – Fotografie „Sensorama“, 1962, Copyright Morton Heilig – http://www.medienkunstnetz.de/assets/img/data/3331/bild.jpg, letzter Abruf: 08.07.2020

Abbildung 2 – Vergleich HUD-basierte Darstellung (oben) von Gesundheit und Munition mit im virtuellen Raum als physische Objekte manifestierter Darstellung (unten)

Abbildung 3 – 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

Abbildung 4 – Eine abstrakte Darstellung der Architektur - Eigene Darstellung

Abbildung 5 – 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

Abbildung 6 – Unity Forums: „Unity Winding Order”, Screenshot von Eric5h5 – https://forum.unity.com/attachments/mesh1-png.244462/, letzter Abruf: 03.07.2020

Abbildung 7 – Tastenbelegung Controller – linke Hand – eigene Darstellung

Abbildung 8 – Tastenbelegung Controller – rechte Hand – eigene Darstellung

Abbildung 9 – Teleportation – selbst erstellter Screenshot

Abbildung 10 – Dateiauswahldialog – selbst erstellter Screenshot

Abbildung 11 – Wechseln des Werkzeuges – selbst erstellter Screenshot

Abbildung 12 –Einstellung der Amplitude – selbst erstellter Screenshot

Abbildung 13 – Visualisierung der gezeichneten Spur – selbst erstellter Screenshot

Abbildung 14 – Hauptmenü – selbst erstellter Screenshot

**Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.**

# Code-Listing-Verzeichnis

[Listing 1 – Generierung der Von-Hann-Koeffizienten 17](#_Toc45808407)

[Listing 2 – Instanzieren des Speicherbereichs, Erstellung fftw-Plan und Ausführung 18](#_Toc45808408)

[Listing 3 – Freigeben der fftw-Speicher-Allokation 18](#_Toc45808409)

[Listing 4 – Definition der Triangles mit korrekter Winding-Order 20](#_Toc45808410)

[Listing 5 – Multi-threaded Burst Code zum Finden der Vertices innerhalb des Kollisionsradius 23](#_Toc45808411)

[Listing 6 – Schwellwert-Logik zur Berechnung der Phaseninformationen 25](#_Toc45808412)

[Listing 7 – Berechnung der FFT-Daten aus Phasen- und Amplituden-Daten für die anschließende IFFT 26](#_Toc45808413)

[Listing 8 – De-Windowing eines Von-Hann-Fensters mit 50% Overlap 27](#_Toc45808414)

**Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.Error! Hyperlink reference not valid.Error! Bookmark not defined.**

# 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. | | | | |
|  | | | | |
|  |  |  |  |  |
| Ort, Datum |  | Vorname Nachname |  | Unterschrift |

# Anhang – Programmcode

**AudioEngine.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using NAudio;

using NAudio.Wave;

using SimpleFileBrowser;

using UnityEngine;

using Valve.VR.Extras;

public class AudioEngine : MonoBehaviour

{

    // just for testing in Unity play mode

    public enum testFiles

    {

        sinesweep1HzTo48000Hz,

        sine4000Hz,

        sine100Hz,

        test,

        drumloop,

        silence

    }

    public testFiles selectAudioTestFile;

    public string filePath;

    [HideInInspector]

    public double[][] fftData;

    public double[][] fftDataMagnitudes;

    public double[][] fftDataPhases;

    public double[][] ifftData;

    private NAudio.Wave.WaveFileReader waveReader;

    private NAudio.Wave.WaveOutEvent waveOut;

    private NAudio.Wave.WaveStream waveProvider;

    public bool fftDataEdited = false;

    private float[] audioData;

    private byte[] playbackBuffer;

    private Stream memoryStream;

    public int importSampleRate;

    public int importBitDepth;

    public double importDurationInMs;

    public int importChannels;

    public int audioNumOfChunks;

    public int fftSize;

    public int fftNumOfChunks;

    public float fftOverlapPercent;

    public int fftOverlapOffset;

    public double[] fftFrequencies;

    public int fftBinCount;

    public bool isPlaying;

    private GameObject fileBrowserVr;

    private SpectrumMeshGenerator spectrum;

    private SteamVR\_LaserPointer laser;

    private Fft fft;

    /// <summary>

    /// Makes sure audio engine is loaded before Start() routines of other GameObjects

    /// </summary>

    void Awake()

    {

        this.fileBrowserVr = GameObject.Find("SimpleFileBrowserCanvas");

        this.fileBrowserVr.SetActive(false);

        this.spectrum = GameObject.Find("SpectrumMesh").GetComponent<SpectrumMeshGenerator>();

        this.laser = GameObject.Find("RightHand").GetComponent<SteamVR\_LaserPointer>();

        if (!Application.isEditor)

        {

            if (!UnityEngine.XR.XRDevice.isPresent)

            {

                // Show 2D file dialog when in standalone mode

                Debug.Log("This application is meant to be used in VR using a HMD. 2D mode fallback only supports basic display of audio information and simple navigation.");

            } else {

                // Show file dialog in Vr world space

                this.fileBrowserVr.SetActive(true);

                FileBrowser.SingleClickMode = true;

                FileBrowser.SetFilters(true, new FileBrowser.Filter("Audio", ".wav", ".aiff", ".mp3", ".m4a", ".ogg"));

                FileBrowser.AddQuickLink("Examples", Application.dataPath + "/Resources/Audio/", null);

                StartCoroutine(WaitForLoadDialog(true));

            }

        } else {

            // Load audio data based on selection in AudioEngine component in Unity editor mode

            switch (this.selectAudioTestFile)

            {

                case testFiles.sinesweep1HzTo48000Hz:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "sinesweep\_1Hz\_48000Hz\_-3dBFS\_30s.wav";

                    break;

                case testFiles.sine4000Hz:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "sinus4000hz-10db.wav";

                    break;

                case testFiles.sine100Hz:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "sinus100hz-10db.wav";

                    break;

                case testFiles.test:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "test.wav";

                    break;

                case testFiles.drumloop:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "drumloop.wav";

                    break;

                case testFiles.silence:

                    this.filePath = Application.dataPath + "/Resources/Audio/" + "silence.wav";

                    break;

            }

            this.LoadAudioData();

        }

    }

    public void OpenDialog()

    {

        // Destroy spectrum meshes GOs

        foreach (Transform child in GameObject.Find("SpectrumMesh").transform.Cast<Transform>().ToArray())

        {

            Destroy(child.gameObject);

        }

        // Destroy frequency legend GOs

        foreach (Transform child in GameObject.Find("FreqLegend").transform.Cast<Transform>().ToArray())

        {

            Destroy(child.gameObject);

        }

        GameObject.Find("UIMainMenu").SetActive(false);

        this.fileBrowserVr.SetActive(true);

        FileBrowser.SingleClickMode = true;

        FileBrowser.SetFilters(true, new FileBrowser.Filter("Audio", ".wav", ".aiff", ".mp3", ".m4a", ".ogg"));

        FileBrowser.AddQuickLink("Examples", Application.dataPath + "/Resources/Audio/", null);

        StartCoroutine(WaitForLoadDialog(false));

    }

    /// <summary>

    /// Shows file browser dialog in VR standalone mode

    /// </summary>

    IEnumerator WaitForLoadDialog(bool ignoreCancel)

    {

        while (!FileBrowser.Success)

        {

            if (ignoreCancel && !this.fileBrowserVr.activeSelf)

                this.fileBrowserVr.SetActive(true);

            yield return null;

        }

        this.filePath = FileBrowser.Result;

        this.LoadAudioData();

    }

    /// <summary>

    /// Loads audio data from external file

    /// </summary>

    public void LoadAudioData()

    {

        // Read in wav file and convert into a float[] of samples

        this.waveReader = new NAudio.Wave.WaveFileReader(this.filePath);

        this.importSampleRate = this.waveReader.WaveFormat.SampleRate;

        this.importBitDepth = this.waveReader.WaveFormat.BitsPerSample;

        this.importChannels = this.waveReader.WaveFormat.Channels;

        this.importDurationInMs = this.waveReader.TotalTime.TotalMilliseconds;

        long numOfSamples = this.waveReader.SampleCount \* this.importChannels;

        this.audioData = new float[numOfSamples];

        float[] sample;

        int pos = 0;

        while ((sample = this.waveReader.ReadNextSampleFrame()) != null)

        {

            if (sample.Length > 1) {

                Debug.Log("<AudioEngine> multi-channel audio files are not supported right now - so only first channel is used, number of channels in the input file: " + this.importChannels.ToString());

                this.importChannels = 1; //override, because only mono supported

            }

            this.audioData[pos] = sample[0];

            pos++;

        }

        this.DoFft();

        this.DoIfft();

        this.fftDataEdited = false;

        this.spectrum.GenerateMeshFromAudioData();

    }

    /// <summary>

    /// Start playback of IFFT audio data

    /// </summary>

    public void Play()

    {

        this.laser.active = false;

        // Do IFFT if there are changes in the spectrum

        if (this.fftDataEdited)

            this.DoIfft();

        Debug.Log("<AudioEngine> Play");

        this.waveProvider = new NAudio.Wave.RawSourceWaveStream(

                this.memoryStream,

                NAudio.Wave.WaveFormat.CreateIeeeFloatWaveFormat(this.importSampleRate, this.importChannels)

                );

        if (this.waveOut == null)

        {

            this.waveOut = new NAudio.Wave.WaveOutEvent();

            this.waveOut.PlaybackStopped += OnPlaybackStopped;

        }

        //this.loopStream = new LoopStream(this.waveProvider);

        this.waveOut.Init(this.waveProvider);

        this.waveOut.Play();

        this.isPlaying = true;

    }

    /// <summary>

    /// Stop playback of IFFT audio data

    /// </summary>

    public void Stop()

    {

        this.laser.active = true;

        this.waveOut.Stop();

        this.isPlaying = false;

        this.spectrum.ResetMeshColors();

        Debug.Log("<AudioEngine> Stop");

    }

    private void OnPlaybackStopped(object sender, System.EventArgs e)

    {

        Debug.Log("<AudioEngine> Playback stopped");

    }

    /// <summary>

    /// Provides current playback position

    /// </summary>

    /// <returns>Playback position in milliseconds</returns>

    public double GetPositionInMs()

    {

        long bytePos = this.waveProvider.Position;

        //long bytePos = this.memoryStream.Position;

        double ms = bytePos / this.importSampleRate / 4 \* 1000.0;

        // Loop

        if (bytePos >= this.memoryStream.Length)

            this.Rewind();

        return ms;

    }

    /// <summary>

    /// Set looping of audio playback

    /// </summary>

    /// <param name="loop">true: enable looping, false: disable looping</param>

    public void SetAudioLooping(bool loop)

    {

       //this.loopStream.EnableLooping = loop;

    }

    /// <summary>

    /// Rewind playback of IFFT audio data

    /// </summary>

    public void Rewind()

    {

        Debug.Log("<AudioEngine> Rewind");

        if (this.waveOut != null)

        {

            this.waveProvider.Position = 0;

            //this.memoryStream.Position = 0;

            this.spectrum.ResetMeshColors();

        }

    }

    /// <summary>

    /// Free audio resources

    /// </summary>

    public void StopAudioEngine()

    {

        if (this.waveOut != null)

        {

            this.waveOut.Dispose();

            this.waveOut = null;

        }

        if (this.waveReader != null)

        {

            this.waveReader.Close();

            this.waveReader.Dispose();

            this.waveReader = null;

        }

        this.isPlaying = false;

    }

    /// <summary>

    /// Create overlapping & windowing and perform FFT

    /// </summary>

    public void DoFft()

    {

        if (this.audioData != null && this.audioData.Length > 0)

        {

            this.fftSize = (int)(this.importSampleRate \* 0.0232199546485261); // magic number taken from 44.100 Hz / 1024

            // Make even if odd

            if (this.fftSize % 2 != 0)

                this.fftSize--;

            this.fftBinCount = fftSize / 2;

            this.fft = new Fft();

            // Calculate number of chunks for the given FFT size...

            this.audioNumOfChunks = (this.audioData.Length + this.fftSize) / this.fftSize; // integer round up

            // ... with overlapping:

            this.fftNumOfChunks = (int)(this.audioNumOfChunks / this.fftOverlapPercent) - 1;

            Debug.Log("<AudioEngine> numOfSamples: " + this.audioData.Length.ToString() +

                " fftSize: " + this.fftSize.ToString() +

                " numberOfChunks: " + this.audioNumOfChunks.ToString() +

                " numberOfChunks with " + (100 \* this.fftOverlapPercent).ToString() + "% overlap: " + this.fftNumOfChunks.ToString() +

                " binResolution: " + (this.importSampleRate / this.fftSize).ToString() + "Hz"

                );

            // Create map of frequencies for the bins

            this.fftFrequencies = new double[this.fftBinCount];

            for (int i = 0; i < this.fftBinCount; i++)

                this.fftFrequencies[i] = (double)(i+1) / this.fftBinCount \* this.importSampleRate / 2;

            // Get window function factors

            double[] window = Fft.MakeWindow(this.fftSize, Fft.WindowType.hann); // Don't change window function, only Von-Hann supported right now

            // Chunk data into overlapping parts, apply window function and run FFT

            double[][] input = new double[this.fftNumOfChunks][];

            double[][] result = new double[this.fftNumOfChunks][];

            double[][] magnitudes = new double[this.fftNumOfChunks][];

            double[][] phases = new double[this.fftNumOfChunks][];

            int idx = 0;

            this.fftOverlapOffset = (int)(this.fftSize \* this.fftOverlapPercent);

            for (int i = 0; i < this.fftNumOfChunks; i++)

            {

                input[i] = new double[this.fftSize];

                result[i] = new double[this.fftSize \* 2];

                magnitudes[i] = new double[this.fftSize];

                phases[i] = new double[this.fftSize];

                for (int j = 0; j < this.fftSize; j++)

                {

                    // last chunk might be smaller than fftSize...

                    if (this.audioData.Length > idx)

                    {

                        input[i][j] = this.audioData[idx] \* window[j];

                    } else {

                        // ... fill up with zeros then

                        input[i][j] = 0;

                    }

                    idx++;

                }

                // Reset index for overlap

                idx -= this.fftOverlapOffset;

                result[i] = this.fft.RunFft(input[i], true);

                magnitudes[i] = Fft.GetMagnitudes(result[i]);

                phases[i] = Fft.GetPhaseInformation(result[i], magnitudes[i]);

                this.fftData = result;

                this.fftDataMagnitudes = magnitudes;

                this.fftDataPhases = phases;

            }

        }

    }

    /// <summary>

    /// Performs IFFT and removes overlapping & windowing

    /// </summary>

    public void DoIfft()

    {

        if (this.fftDataMagnitudes != null && this.fftDataMagnitudes.Length > 0)

        {

            // Do IFFT per chunk

            double[][] result = new double[this.fftDataMagnitudes.Length][];

            for (int i = 0; i < this.fftDataMagnitudes.Length; i++)

            {

                // result of ifft is in interleaved complex format - take only even indexes (the real part)

                result[i] = fft.RunIfft(Fft.GetFftDataFromMagnitudeAndPhase(this.fftDataMagnitudes[i], this.fftDataPhases[i]))

                    .Where((value, index) => index % 2 == 0).ToArray();

            }

            // For debugging

            // this.PrintFftData(fft.RunIfft(this.fftData[1337]), fft.RunIfft(Fft.GetFftDataFromMagnitudeAndPhase(this.fftDataMagnitudes[1337], this.fftDataPhases[1337])), "fftdata.txt");

            // Get window function factors

            double[] window = Fft.MakeWindow(this.fftSize, Fft.WindowType.hann);

            // By using Von-Hann-Window with 50% overlap we can simply sum the values in the overlap region together to revert the windowing

            // and to get the correct sample count again

            this.ifftData = new double[this.audioNumOfChunks][];

            for (int i = 0; i < this.audioNumOfChunks; i++)

            {

                this.ifftData[i] = new double[this.fftSize];

                for (int j = 0; j < this.fftSize; j++)

                {

                    // Copy over and revert window function for the values of the non-overlap region from the first chunk AND...

                    if (i == 0 && j < this.fftSize - this.fftOverlapOffset)

                    {

                        this.ifftData[i][j] = result[i][j] / window[j];

                        continue;

                    }

                    // ...of the non-overlap region at the end

                    if (i >= this.audioNumOfChunks - 1 && j >= this.fftSize - this.fftOverlapOffset)

                    {

                        this.ifftData[i][j] = result[i \* 2][j] / window[j];

                        continue;

                    }

                    // Sum overlapping values (this code only works for 50% overlap)

                    if (j < this.fftSize / 2)

                    {

                        this.ifftData[i][j] = result[i \* 2 - 1][this.fftOverlapOffset + j] + result[i \* 2][j];

                    } else {

                        this.ifftData[i][j] = result[i \* 2][j] + result[i \* 2 + 1][j - this.fftOverlapOffset];

                    }

                }

            }

            this.FillPlaybackBuffer();

        }

    }

    /// <summary>

    /// Converts IFFT data to bytes and fills playback buffer

    /// </summary>

    private void FillPlaybackBuffer()

    {

        this.playbackBuffer = new byte[this.ifftData.Length \* this.fftSize \* 8];

        int pos = 0;

        for (int i = 0; i < this.ifftData.Length; i++)

        {

            // double to float conversion

            float[] values;

            values = System.Array.ConvertAll<double, float>(this.ifftData[i], y => (float)y);

            // float to bytes conversion

            byte[] frames;

            frames = values.SelectMany(value => System.BitConverter.GetBytes(value)).ToArray();

            System.Array.Copy(frames, 0, this.playbackBuffer, pos, frames.Length);

            pos += frames.Length;

        }

        this.memoryStream = new System.IO.MemoryStream(this.playbackBuffer);

        this.memoryStream.Seek(0, System.IO.SeekOrigin.Begin);

        Debug.Log("<AudioEngine> playback buffer updated");

    }

    public void WriteAudioToDisk()

    {

        // Do IFFT if there are changes in the spectrum

        if (this.fftDataEdited)

            this.DoIfft();

        this.waveProvider = new NAudio.Wave.RawSourceWaveStream(

                this.memoryStream,

                NAudio.Wave.WaveFormat.CreateIeeeFloatWaveFormat(this.importSampleRate, this.importChannels)

                );

        string path = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory).ToString() + "/" + "VrAudioSandboxExport-" + DateTime.Now.ToString("yyyy-MM-dd\_HH-mm-ss") + ".wav");

        WaveFileWriter.CreateWaveFile(path, this.waveProvider);

    }

    /// <summary>

    /// Test function for FFT/IFFT routines

    /// </summary>

    private static void FftTest() {

        float[] testInF = { 0.00000002345f, -0.00000045464f, 1.0040346634f, -0.86747333346f };

        double[] testInD = new double[testInF.Length];

        for (int i = 0; i<testInD.Length; i++)

            testInD[i] = testInF[i];

        Fft testFft = new Fft();

        double[] fftResultD = testFft.RunFft(testInD, true);

        double[] ifftResultD = testFft.RunIfft(fftResultD);

        float[] ifftResultF;

        ifftResultF = System.Array.ConvertAll<double, float>(ifftResultD, y => (float)y);

        for (int i = 0; i< testInF.Length; i++)

        {

            Debug.Log("INPUT: " + testInF[i] + " OUPUT: " + ifftResultF[i \* 2]);

        }

    }

    private void PrintFftData(double[] fftData1, double[] fftData2, string fileName)

    {

        List<string> lines = new List<string>();

        lines.Add("fftData-Length: "+fftData.Length.ToString());

        lines.Add("Re1\t\t\t\t\tImg1\t\t\t\t\tRe2\t\t\t\t\tImg2\t\t\t\t\tRe1-Re2\t\t\t\t\tImg1-Img2");

        for (int i = 0; i < fftData.Length / 2; i++)

        {

            lines.Add(

                fftData1[2 \* i].ToString() + "\t\t\t" + fftData1[2 \* i + 1].ToString() + "\t\t\t" +

                fftData2[2 \* i].ToString() + "\t\t\t" + fftData2[2 \* i + 1].ToString() + "\t\t\t" +

                (fftData1[2 \* i] - fftData2[2 \* i]).ToString() + "\t\t\t" + (fftData1[2 \* i + 1] - fftData2[2 \* i + 1]).ToString());

        }

        System.IO.File.WriteAllLines(@"C:\Users\bytecrunch\Desktop\" + fileName, lines);

    }

    /// <summary>

    /// Input handling

    /// </summary>

    private void Update()

    {

        if (Input.GetButtonDown("PlayStop"))

        {

            if (!this.isPlaying)

            {

                this.Play();

            } else {

                this.Stop();

            }

        }

        if (Input.GetButtonDown("Rewind"))

        {

            this.Rewind();

        }

    }

    public void ExitProgram()

    {

        Application.Quit();

    }

    /// <summary>

    /// Exit handler

    /// </summary>

    void OnApplicationQuit()

    {

        Debug.Log("Exit");

        this.StopAudioEngine();

    }

}

**Fft.cs**

using System.Collections;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using FFTWSharp;

// FFT uses FFTW - see http://www.fftw.org/index.html

// C# Wrapper based on the documentation of FFTWSharp - see https://github.com/tszalay/FFTWSharp

public class Fft

{

    System.IntPtr ptr;

    // pointers to FFTW plan objects

    System.IntPtr fplanForward, fplanBackward;

    private int n;

    private double[] window;

    public enum WindowType

    {

        flat,

        hann,

        hamming,

        blackman

    };

    private WindowType windowFunction;

    /// <summary>

    /// Creates array of window function factors for the current FFT size

    /// Windowing taken from https://github.com/101010b/AudioTest/blob/master/fft.cs

    /// </summary>

    public static double[] MakeWindow(int n, Fft.WindowType windowFunction)

    {

        double alpha, a0, a1, a2;

        double[] window = new double[n];

        switch (windowFunction)

        {

            case WindowType.hann:

                for (int i = 0; i < n; i++)

                    window[i] = 0.5 - 0.5 \* System.Math.Cos((double)i \* 2 \* System.Math.PI / (n - 1));

                break;

            case WindowType.hamming:

                for (int i = 0; i < n; i++)

                    window[i] = 0.54 - 0.46 \* System.Math.Cos((double)i \* 2 \* System.Math.PI / (n - 1));

                break;

            case WindowType.blackman:

                alpha = 0.16;

                a0 = (1.0 - alpha) / 2.0;

                a1 = 0.5;

                a2 = alpha / 2;

                for (int i = 0; i < n; i++)

                    window[i] =

                            a0

                        - a1 \* System.Math.Cos((double)i \* 2 \* System.Math.PI / (n - 1))

                        + a2 \* System.Math.Cos((double)i \* 4 \* System.Math.PI / (n - 1));

                break;

            case WindowType.flat:

            default:

                for (int i = 0; i < n; i++)

                    window[i] = 1.0;

                break;

        }

        return window;

    }

    /// <summary>

    /// Runs the FFT with double precision

    /// </summary>

    /// <param name="input">input array</param>

    /// <param name="real">set to true for converting double[] of real numbers to double[] of complex numbers with real and imagenary parts interleaved</param>

    public double[] RunFft(double[] input, bool real)

    {

        if (real)

            input = Fft.RealToComplex(input);

        this.n = input.Length;

        this.ptr = fftw.malloc(this.n \* sizeof(double));

        // (n / 2 because complex numbers are stored as pairs of doubles)

        this.fplanForward = fftw.dft\_1d(this.n / 2, this.ptr, this.ptr, fftw\_direction.Forward, fftw\_flags.Measure);

        Marshal.Copy(input, 0, this.ptr, this.n);

        fftw.execute(this.fplanForward);

        double[] output = new double[this.n];

        Marshal.Copy(this.ptr, output, 0, this.n);

        fftw.free(this.ptr);

        fftw.destroy\_plan(this.fplanForward);

        return output;

    }

    /// <summary>

    /// Runs the IFFT with double precision - RunFft() must be run before for initalization of fft size and window function

    /// </summary>

    /// <param name="input">unnormalized transform ouput of FFT</param>

    /// <param name="output">array to store output of IFFT</param>

    public double[] RunIfft(double[] input)

    {

        this.ptr = fftw.malloc(this.n \* sizeof(double));

        // (n / 2 because complex numbers are stored as pairs of doubles)

        this.fplanBackward = fftw.dft\_1d(this.n / 2, this.ptr, this.ptr, fftw\_direction.Backward, fftw\_flags.Measure);

        Marshal.Copy(input, 0, this.ptr, this.n);

        fftw.execute(this.fplanBackward);

        double[] output = new double[this.n];

        Marshal.Copy(this.ptr, output, 0, this.n);

        /\*

         \* FFTW computes an unnormalized transform, in that there is no coefficient in front of the summation in the DFT.

         \* In other words, applying the forward and then the backward transform will multiply the input by n.

         \* http://www.fftw.org/fftw3\_doc/The-1d-Discrete-Fourier-Transform-\_0028DFT\_0029.html

         \*/

        // Divide by n/2

        for (int i = 0; i < this.n; i++)

        {

            output[i] /= this.n / 2;

        }

        fftw.free(this.ptr);

        fftw.destroy\_plan(this.fplanBackward);

        return output;

    }

    /// <summary>

    /// Converts a real number double[] to complex number double[] with zeros as imagenary part

    /// </summary>

    /// <param name="real">double[] input array of real numbers</param>

    /// <returns>double[] double the size of input with interleaved zeros</returns>

    private static double[] RealToComplex(double[] real)

    {

        int n = real.Length;

        double[] comp = new double[n \* 2];

        for (int i = 0; i < n; i++)

        {

            comp[2 \* i] = real[i];

        }

        return comp;

    }

    /// <summary>

    /// Calculates the absolute values for a double[] of complex numbers.

    /// </summary>

    /// <param name="x">input double[] with real and imagenary parts interleaved</param>

    /// <returns>double[] with absolute values</returns>

    public static double[] GetMagnitudes(double[] x)

    {

        int n = x.Length / 2;

        double[] y = new double[n];

        for (int i = 0; i < n; i++)

        {

            y[i] = System.Math.Sqrt(x[2 \* i] \* x[2 \* i] + x[2 \* i + 1] \* x[2 \* i + 1]);

        }

        return y;

    }

    /// <summary>

    /// Calculates the phase information values for a double[] of complex fft data WITH threshold limit

    /// </summary>

    /// <param name="fftData">input double[] of FFT data with real and imagenary parts interleaved</param>

    /// <param name="magnitudes">corresponding double[] magnitudes (size half of fftData array)</param>

    /// <returns>double[] with phase information</returns>

    public static double[] GetPhaseInformation(double[] fftData, double[] magnitudes)

    {

        int n = fftData.Length / 2;

        double[] y = new double[n];

        double threshold = 0;

        // Use threshold limit if magnitudes were provided

        /\*

            \* "Even a small floating rounding off error will amplify the result and manifest incorrectly

            \* as useful phase information. [...] The solution is to define a tolerance threshold and

            \* ignore all the computed phase values that are below the threshold."

            \* https://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/

            \*/

        // Find abs(maximum) of provided magnitudes data...

        double max = 0;

        for (int i = 0; i < n; i++)

        {

            double m = magnitudes[i];

            if (m > max)

                max = m;

        }

        // ...and use 1/10000th of it as threshold

        threshold = max / 10000;

        // Calculate phase information with above threshold

        for (int i = 0; i < n; i++)

        {

            if (magnitudes[i] > threshold)

            {

                // Calculate phase information

                y[i] = System.Math.Atan2(fftData[2 \* i + 1], fftData[2 \* i]);

            } else {

                // Ignore value

                y[i] = 0;

            }

        }

        return y;

    }

    /// <summary>

    /// Returns FFT complex data for use in IFFT from magnitudes and phase information

    /// </summary>

    /// <param name="magnitudes">double[] with magnitudes</param>

    /// <param name="phases">double[] with phase information</param>

    /// <returns>FFT data as interleaved complex format double[]</returns>

    public static double[] GetFftDataFromMagnitudeAndPhase(double[] magnitudes, double[] phases)

    {

        int n = magnitudes.Length;

        double[] fftData = new double[n \* 2];

        for (int i = 0; i < n; i++)

        {

            fftData[i \* 2] = magnitudes[i] \* System.Math.Cos(phases[i]); // real part

            fftData[i \* 2 + 1] = magnitudes[i] \* System.Math.Sin(phases[i]); // imagenary part

        }

        return fftData;

    }

    /// <summary>

    /// Calculates power in dB

    /// </summary>

    /// <param name="x"></param>

    /// <returns></returns>

    public static double[] GetPowerInDb(double[] x)

    {

        int n = x.Length / 2;

        double[] y = new double[n];

        for (int i = 0; i < n; i++)

        {

            y[i] = 10 \* System.Math.Log(x[2 \* i] \* x[2 \* i] + x[2 \* i + 1] \* x[2 \* i + 1]) / System.Math.Log(10);

        }

        return y;

    }

}

**FloorHelper.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class FloorHelper : MonoBehaviour

{

    void Start()

    {

    }

    void Update()

    {

    }

    public void FitFloorToBounds(Bounds refBounds)

    {

        if (refBounds != null)

        {

            Renderer floorRenderer = (Renderer)GetComponent("MeshRenderer");

            // Scale floor to spectrum dimensions

            Vector3 a = refBounds.size;

            Vector3 b = floorRenderer.bounds.size;

            gameObject.transform.localScale.Set(a.x / b.x, 1, a.z / b.z);

            // Re-position floor

            gameObject.transform.position.Set(refBounds.center.x, 0, refBounds.center.z);

        }

    }

}

**LeftControllerActions.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Valve.VR;

public class LeftControllerActions : MonoBehaviour

{

    public SteamVR\_Action\_Boolean PlayStop;

    public SteamVR\_Action\_Boolean Rewind;

    public SteamVR\_Action\_Boolean ToolMenu;

    public SteamVR\_Action\_Boolean ToggleMainMenu;

    //public SteamVR\_Action\_Vector2 Scroll;

    public SteamVR\_Input\_Sources handType;

    private GameObject leftHand;

    private float yRotation;

    public bool toolMenuActive = false;

    public bool mainMenuActive = false;

    private Vector3 ToolUIworldPos;

    private Vector3 ToolUIlocalPos;

    private Quaternion UIlocalRotation;

    private Vector3 mainMenuworldPos;

    private Vector3 mainMenulocalPos;

    private Quaternion mainMenulocalRotation;

    private AudioEngine audioEngine;

    private SpectrumMeshGenerator spectrum;

    private ToolHandler tool;

    private GameObject uiTool;

    private GameObject uiMain;

    private void Awake()

    {

        this.leftHand = GameObject.Find("LeftHand");

    }

    void Start()

    {

        this.audioEngine = GameObject.Find("Audio").GetComponent<AudioEngine>();

        this.spectrum = GameObject.Find("SpectrumMesh").GetComponent<SpectrumMeshGenerator>();

        this.tool = this.GetComponent<ToolHandler>();

        this.uiTool = GameObject.Find("UITool");

        this.uiTool.SetActive(false);

        this.uiMain = GameObject.Find("UIMainMenu");

        this.uiMain.SetActive(false);

        this.PlayStop.AddOnStateDownListener(this.PlayStopDown, handType);

        this.Rewind.AddOnStateDownListener(this.RewindDown, handType);

        this.ToolMenu.AddOnStateDownListener(this.ToolMenuActivate, handType);

        this.ToolMenu.AddOnStateUpListener(this.ToolMenuDeactivate, handType);

        this.ToggleMainMenu.AddOnStateDownListener(this.MainMenuToggle, handType);

        //this.Scroll.AddOnAxisListener(this.ScaleMesh, handType);

    }

    private void PlayStopDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        if (!this.audioEngine.isPlaying)

        {

            this.audioEngine.Play();

        }

        else

        {

            this.audioEngine.Stop();

        }

    }

    private void RewindDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.audioEngine.Rewind();

    }

    private void ToolMenuActivate(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.toolMenuActive = true;

        this.uiTool.SetActive(true);

        this.ToolUIworldPos = this.uiTool.transform.position;

        this.ToolUIlocalPos = this.uiTool.transform.localPosition;

        this.UIlocalRotation = this.uiTool.transform.localRotation;

        this.uiTool.transform.SetParent(null, true);

        this.uiTool.transform.position = this.ToolUIworldPos;

        this.yRotation = this.leftHand.transform.rotation.y;

    }

    private void ToolMenuDeactivate(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.toolMenuActive = false;

        this.uiTool.SetActive(false);

        this.uiTool.transform.SetParent(GameObject.Find("HoverPointLeft").transform);

        this.uiTool.transform.localPosition = this.ToolUIlocalPos;

        this.uiTool.transform.localRotation = this.UIlocalRotation;

    }

    private void MainMenuToggle(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        if (this.mainMenuActive)

        {

            this.mainMenuActive = false;

            this.uiMain.SetActive(false);

            this.uiMain.transform.SetParent(GameObject.Find("VRCamera").transform, true);

            this.uiMain.transform.localPosition = this.mainMenulocalPos;

            this.uiMain.transform.localRotation = this.mainMenulocalRotation;

        } else {

            this.mainMenuActive = true;

            this.uiMain.SetActive(true);

            this.mainMenuworldPos = this.uiMain.transform.position;

            this.mainMenulocalPos = this.uiMain.transform.localPosition;

            this.mainMenulocalRotation = this.uiMain.transform.localRotation;

            this.uiMain.transform.SetParent(null, true);

            this.uiMain.transform.position = this.mainMenuworldPos;

        }

    }

    /\*public void ScaleMesh(SteamVR\_Action\_Vector2 fromAction, SteamVR\_Input\_Sources fromSource, Vector2 axis, Vector2 delta)

    {

        this.spectrum.ScaleMeshY(delta.y / 10);

    }\*/

    void Update()

    {

        // Handle tool selection menu

        if (this.toolMenuActive)

        {

            if (this.leftHand.transform.rotation.y - this.yRotation > 0.05)

            {

                this.tool.nextTool();

                this.yRotation = this.leftHand.transform.rotation.y;

            }

            else if (this.yRotation - this.leftHand.transform.rotation.y > 0.05) {

                this.tool.prevTool();

                this.yRotation = this.leftHand.transform.rotation.y;

            }

        }

    }

}

RightControllerActions.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using Valve.VR;

using Valve.VR.Extras;

public class RightControllerActions : MonoBehaviour

{

    public SteamVR\_Action\_Boolean DecToolRadius;

    public SteamVR\_Action\_Boolean IncToolRadius;

    public SteamVR\_Action\_Boolean DecToolAbsoluteValue;

    public SteamVR\_Action\_Boolean IncToolAbsoluteValue;

    public SteamVR\_Action\_Boolean hold;

    public SteamVR\_Input\_Sources handType;

    private SteamVR\_LaserPointer laserPointer;

    private AudioEngine audioEngine;

    private SpectrumMeshGenerator spectrum;

    private ToolHandler tool;

    void Start()

    {

        this.audioEngine = GameObject.Find("Audio").GetComponent<AudioEngine>();

        this.spectrum = GameObject.Find("SpectrumMesh").GetComponent<SpectrumMeshGenerator>();

        this.tool = this.GetComponent<ToolHandler>();

        this.laserPointer = GameObject.Find("RightHand").GetComponent<SteamVR\_LaserPointer>();

        this.DecToolRadius.AddOnStateDownListener(this.DecToolRadiusDown, handType);

        this.IncToolRadius.AddOnStateDownListener(this.IncToolRadiusDown, handType);

        this.DecToolAbsoluteValue.AddOnStateDownListener(this.DecToolAbsoluteValueDown, handType);

        this.IncToolAbsoluteValue.AddOnStateDownListener(this.IncToolAbsoluteValueDown, handType);

        this.hold.AddOnStateDownListener(this.TriggerDown, handType);

        this.hold.AddOnStateUpListener(this.TriggerUp, handType);

    }

    private void DecToolRadiusDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.SetToolRadiusWithOffset(-0.005f);

    }

    private void IncToolRadiusDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.SetToolRadiusWithOffset(0.005f);

    }

    private void DecToolAbsoluteValueDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.SetToolAbsoluteValueOffset(-0.05f);

    }

    private void IncToolAbsoluteValueDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.SetToolAbsoluteValueOffset(0.05f);

    }

    private void TriggerDown(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.TriggerDown = true;

    }

    private void TriggerUp(SteamVR\_Action\_Boolean fromAction, SteamVR\_Input\_Sources fromSource)

    {

        this.tool.TriggerDown = false;

        this.tool.TriggerUp();

    }

    void Update()

    {

    }

}

**SpectrumDeformer.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using Unity.Burst;

using Unity.Collections;

using Unity.Jobs;

using UnityEngine;

[RequireComponent(typeof(SpectrumMeshGenerator))]

public class SpectrumDeformer : MonoBehaviour

{

    public Vector3[][] modifiedVertices;

    public List<DeformJob> routineQueue;

    public float deformFactor;

    private SpectrumMeshGenerator spectrum;

    private AudioEngine audioEngine;

    /// <summary>

    /// Clones the mesh vertices and preparation for deforming mesh

    /// </summary>

    public void MeshGenerated()

    {

        this.modifiedVertices = new Vector3[this.spectrum.meshes.Length][];

        for (int i = 0; i < this.spectrum.meshes.Length; i++)

        {

            this.modifiedVertices[i] = new Vector3[this.audioEngine.fftBinCount];

            this.modifiedVertices[i] = this.spectrum.meshes[i].vertices;

        }

    }

    public void DeformMeshPoint(Vector3 point, Vector3 direction, float radius, float absoluteValue = -1)

    {

        List<Vector3> points = new List<Vector3>();

        points.Add(point);

        this.routineQueue.Add(new DeformJob(points, direction, radius, absoluteValue));

    }

    public void DeformMeshMultiplePoints(List<Vector3> points, Vector3 direction, float radius, float absoluteValue = -1)

    {

        this.routineQueue.Add(new DeformJob(points, direction, radius, absoluteValue));

    }

    /// <summary>

    /// Moves FFT vertices to the chosen direction for the given points and radius

    /// </summary>

    /// <param name="point">point for vertices to be affected</param>

    /// <param name="direction">direction of movement</param>

    /// <param name="radius">radius for vertices to be affected</param>

    /// <param name="absoluteValue">(optional) add given absolute value</param>

    private IEnumerator DeformMeshMultiplePointsWorker(DeformJob routine, List<Vector3> points, Vector3 direction, float radius, float absoluteValue)

    {

        routine.isRunning = true;

        int frames = 0;

        // Interpolate additional points if there are gaps

        int i = 0;

        while (i < points.Count - 1)

        {

            float distance = (points[i] - points[i + 1]).magnitude;

            if (distance >= radius)

            {

                int n = (int)(distance / radius);

                for (int j = 1; j <= n; j++)

                {

                    points.Insert(i, Vector3.Lerp(points[i], points[i + 1], (float)1 / n \* j));

                    i++;

                }

            }

            i++;

            if (frames % 20 == 0)

                yield return null;

        }

        // Collect all changes to be made

        NativeArray<Vector3>[] vertices = new NativeArray<Vector3>[this.modifiedVertices.Length];

        NativeQueue<VertexChange> vertexChanges = new NativeQueue<VertexChange>(Allocator.Persistent);

        FindPointsToUpdateJob[] jobData = new FindPointsToUpdateJob[this.modifiedVertices.Length];

        JobHandle[] jobHandles = new JobHandle[this.modifiedVertices.Length];

        for (int j = 0; j < this.modifiedVertices.Length; j++)

        {

            if (j > 0)

                jobHandles[j - 1].Complete(); //Ensure that previous job is complete before preparing next one to prevent access on the NativeQueue while previous job is writing

            // only modify vertices corresponding to fft values, omit raster vertices indexes

            Vector3[] peakVertices = new Vector3[this.modifiedVertices[j].Length - this.spectrum.startIndexOfPeakVertices];

            Array.Copy(this.modifiedVertices[j], this.spectrum.startIndexOfPeakVertices, peakVertices, 0, peakVertices.Length);

            vertices[j] = new NativeArray<Vector3>(peakVertices, Allocator.TempJob);

            jobData[j] = new FindPointsToUpdateJob();

            jobData[j].meshIdx = j;

            jobData[j].vertices = vertices[j];

            jobData[j].points = new NativeArray<Vector3>(points.ToArray(), Allocator.TempJob);

            jobData[j].direction = direction;

            jobData[j].deformFactor = this.deformFactor;

            jobData[j].absoluteValue = absoluteValue;

            jobData[j].radius = radius;

            jobData[j].vertexChanges = vertexChanges.AsParallelWriter();

            if (j < 1)

                jobHandles[j] = jobData[j].Schedule(peakVertices.Length, 256);

            else

                jobHandles[j] = jobData[j].Schedule(peakVertices.Length, 256, jobHandles[j - 1]);

        }

        // wait for last job

        jobHandles[this.modifiedVertices.Length - 1].Complete();

        List<int> meshColorsToUpdate = new List<int>();

        // Update meshes, colliders & fftDataMagnitudes

        while (vertexChanges.TryDequeue(out VertexChange vc))

        {

            // Update vertices

            this.modifiedVertices[vc.meshIdx][vc.vertexIdx + this.spectrum.startIndexOfPeakVertices] = new Vector3(vc.x, vc.y, vc.z);

            this.spectrum.mFilters[vc.meshIdx].mesh.vertices = this.modifiedVertices[vc.meshIdx];

            if (frames % 30 == 0)

                yield return null;

            // Note for color update

            if (!meshColorsToUpdate.Contains(vc.meshIdx))

                meshColorsToUpdate.Add(vc.meshIdx);

            this.spectrum.SetMaxPeakValue(this.modifiedVertices[vc.meshIdx][vc.vertexIdx].y \* this.spectrum.fftScalingFactor);

            // Update colliders

            MeshCollider c = GameObject.Find("FFTData" + vc.meshIdx.ToString()).GetComponent<MeshCollider>();

            c.sharedMesh = this.spectrum.mFilters[vc.meshIdx].mesh;

            // Update fft Data

            this.audioEngine.fftDataMagnitudes[vc.meshIdx][vc.vertexIdx] = (double)vc.y / this.spectrum.fftScalingFactor;

            frames++;

        }

        vertexChanges.Dispose();

        // Update mesh colors

        foreach (int m in meshColorsToUpdate)

        {

            for (int v = 0; v < this.spectrum.colors[m].Length; v++)

            {

                this.spectrum.colors[m][v] = Color.Lerp(Color.green, Color.red, this.modifiedVertices[m][v].y / this.spectrum.maxPeakValue);

            }

            this.spectrum.mFilters[m].mesh.SetColors(this.spectrum.colors[m]);

        }

        this.audioEngine.fftDataEdited = true;

        routine.isFinished = true;

    }

    void Awake()

    {

        this.spectrum = GetComponent<SpectrumMeshGenerator>();

        this.audioEngine = GameObject.Find("Audio").GetComponent<AudioEngine>();

        this.routineQueue = new List<DeformJob>();

    }

    void Update()

    {

        for (int i = 0; i < this.routineQueue.Count; i++)

        {

            if (!this.routineQueue[i].isFinished && !this.routineQueue[i].isRunning)

                StartCoroutine(this.DeformMeshMultiplePointsWorker(this.routineQueue[i], this.routineQueue[i].points, this.routineQueue[i].direction, this.routineQueue[i].radius, this.routineQueue[i].absoluteValue));

            else

                this.routineQueue.RemoveAt(i);

        }

    }

}

public class DeformJob

{

    public List<Vector3> points;

    public Vector3 direction;

    public float radius;

    public float absoluteValue = 0;

    public bool isFinished = false;

    public bool isRunning = false;

    public DeformJob(List<Vector3> points, Vector3 direction, float radius, float absoluteValue)

    {

        this.points = new List<Vector3>(points);

        this.direction = direction;

        this.radius = radius;

        this.absoluteValue = absoluteValue;

    }

}

public struct VertexChange

{

    public int meshIdx;

    public int vertexIdx;

    public float x;

    public float y;

    public float z;

    public VertexChange(int meshIdx, int vertexIdx, float x, float y, float z)

    {

        this.meshIdx = meshIdx;

        this.vertexIdx = vertexIdx;

        this.x = x;

        this.y = y;

        this.z = z;

    }

}

[BurstCompile]

public struct FindPointsToUpdateJob : IJobParallelFor

{

    [ReadOnly] public int meshIdx;

    [DeallocateOnJobCompletion] [ReadOnly] public NativeArray<Vector3> vertices;

    [DeallocateOnJobCompletion] [ReadOnly] public NativeArray<Vector3> points;

    [ReadOnly] public Vector3 direction;

    [ReadOnly] public float deformFactor;

    [ReadOnly] public float absoluteValue;

    [ReadOnly] public float radius;

    public NativeQueue<VertexChange>.ParallelWriter vertexChanges;

    public void Execute(int i)

    {

        for (int p = 0; p < points.Length; p++)

        {

            // Ignore y-axis when determing points to be affected

            var distance = (new Vector3(points[p].x, 0, points[p].z) - new Vector3(vertices[i].x, 0, vertices[i].z)).magnitude;

            if (distance < radius)

            {

                if (absoluteValue == -1)

                    // use factor for changing of y-value

                    vertexChanges.Enqueue(new VertexChange(

                        meshIdx,

                        i,

                        vertices[i].x + direction.x \* deformFactor,

                        (vertices[i].y + direction.y \* deformFactor >= 0) ? vertices[i].y + direction.y \* deformFactor : 0f,

                        vertices[i].z + direction.z \* deformFactor));

                else

                    // use absolute value

                    vertexChanges.Enqueue(new VertexChange(meshIdx, i, vertices[i].x, absoluteValue, vertices[i].z));

            }

        }

    }

}

**SpectrumFreqLegend.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SpectrumFreqLegend : MonoBehaviour

{

    void Start()

    {

    }

    void Update()

    {

    }

    public void SetFreqLegend(double[] freq, float edgeLength)

    {

        int bins = freq.Length;

        Vector3 spectrumPos = GameObject.Find("SpectrumMesh").transform.position;

        GameObject[] textObj = new GameObject[bins];

        MeshRenderer[] textRenderer = new MeshRenderer[bins];

        TextMesh[] textMesh = new TextMesh[bins];

        for (int f = 0; f < bins; f++)

        {

            textObj[f] = new GameObject("Freq" + f.ToString());

            textObj[f].transform.parent = this.transform;

            textObj[f].transform.position = new Vector3(spectrumPos.x + f \* edgeLength, spectrumPos.y, spectrumPos.z);

            textObj[f].transform.localEulerAngles = new Vector3(90f, 90f, 0f);

            textObj[f].transform.localScale = new Vector3(0.001f, 0.001f, 0.001f);

            textRenderer[f] = textObj[f].AddComponent<MeshRenderer>();

            textMesh[f] = textObj[f].AddComponent<TextMesh>();

            textMesh[f].text = System.Math.Round(freq[f], 2).ToString() + " Hz\n";

            textMesh[f].alignment = TextAlignment.Right;

            textMesh[f].fontSize = 70;

            textMesh[f].color = Color.white;

        }

    }

}

**SpectrumHelper.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class SpectrumHelper : MonoBehaviour

{

    private GameObject toolValueIndicator;

    private int toolValueIndicatorFramesToShow;

    private void Awake()

    {

        this.toolValueIndicator = GameObject.Find("ToolValueIndicator");

    }

    private void Start()

    {

        // Set size & position of tool value plane to spectrum dimensions

        Vector3 tScale = this.toolValueIndicator.transform.localScale;

        Bounds tBounds = this.toolValueIndicator.GetComponent<MeshFilter>().mesh.bounds;

        Bounds spectrumBounds = this.GetSpectrumBounds();

        tScale.x = (spectrumBounds.size.x \* tScale.x / tBounds.size.x);

        tScale.z = (spectrumBounds.size.z \* tScale.z / tBounds.size.z);

        this.toolValueIndicator.transform.localScale = tScale;

        this.toolValueIndicator.transform.position = spectrumBounds.center;

        this.toolValueIndicator.SetActive(false);

    }

    private void Update()

    {

        if (transform.hasChanged)

        {

            // dirty hack for now as GameObject.Find() doesn't find inactive objects.

            // Due to the teleportation steam VR script, floor is only enabled during teleportation phase

            FloorHelper floor = Resources.FindObjectsOfTypeAll<FloorHelper>()[0];

            floor.FitFloorToBounds(this.GetSpectrumBounds());

            transform.hasChanged = false;

        }

        // Show indicator plane for tool value change

        if (this.toolValueIndicatorFramesToShow > 0)

        {

            if (this.toolValueIndicatorFramesToShow == 1)

                this.toolValueIndicator.SetActive(false);

            this.toolValueIndicatorFramesToShow--;

        }

    }

    public void ShowToolValueIndicator(int framesCount, float y)

    {

        this.toolValueIndicator.transform.SetPositionAndRotation(new Vector3(this.toolValueIndicator.transform.position.x, y, this.toolValueIndicator.transform.position.z), this.toolValueIndicator.transform.rotation);

        this.toolValueIndicator.SetActive(true);

        this.toolValueIndicatorFramesToShow = framesCount;

    }

    /// <summary>

    /// Calculates combined bounds of all mesh renderers combined

    /// </summary>

    /// <returns>Combined bounds of all mesh renderers</returns>

    private Bounds GetSpectrumBounds()

    {

        Renderer[] renderers = GameObject.Find("SpectrumMesh").GetComponentsInChildren<Renderer>();

        Bounds combinedBounds = new Bounds();

        for (int i = 0; i < renderers.Length; i++)

        {

            combinedBounds.Encapsulate(renderers[i].bounds);

        }

        return combinedBounds;

    }

}

**SpectrumMeshGenerator.cs**

using System.Collections;

using System.Collections.Generic;

using System.Diagnostics;

using UnityEngine;

using Valve.VR.InteractionSystem;

public class SpectrumMeshGenerator : MonoBehaviour

{

    public float edgeLengthOfRaster;

    public float fftScalingFactor;

    private AudioEngine audioEngine;

    private SpectrumDeformer deformer;

    private GameObject[] meshObj;

    public MeshFilter[] mFilters;

    private MeshRenderer[] mRenderers;

    private MeshCollider[] mColliders;

    public Mesh[] meshes;

    public float maxPeakValue;

    public Vector3[][] vertices;

    private int[][] triangles;

    public Color32[][] colors;

    private int countOfRasterVertices;

    public int startIndexOfPeakVertices { get { return this.countOfRasterVertices; } }

    private int countOfPeakVertices;

    public void Awake()

    {

        this.audioEngine = GameObject.Find("Audio").GetComponent<AudioEngine>();

        this.deformer = GameObject.Find("SpectrumMesh").GetComponent<SpectrumDeformer>();

    }

    private void Start()

    {

    }

    private void Update()

    {

        /\*if (Input.GetButtonDown("ScaleMeshYDec"))

        {

            this.ScaleMeshY(-0.05f);

        }

        if (Input.GetButtonUp("ScaleMeshYInc"))

        {

            this.ScaleMeshY(0.05f);

        }\*/

        // Color meshes according to play position

        if (this.audioEngine.isPlaying)

        {

            double msPerChunk = this.audioEngine.importDurationInMs / this.audioEngine.ifftData.Length / 2;

            int posIdx = (int)(this.audioEngine.GetPositionInMs() / msPerChunk);

            for (int i = 0; i < posIdx; i++)

            {

                if (i < this.mRenderers.Length)

                    this.mRenderers[i].material = Resources.Load("Materials/SpectrumMatPlaying") as Material;

            }

            // workaround until loopstream event will work to trigger reset of spectrum material

            for (int i = posIdx; i < this.audioEngine.fftDataMagnitudes.Length; i++)

            {

                this.mRenderers[i].material = Resources.Load("Materials/SpectrumMat") as Material;

            }

        }

    }

    /// <summary>

    /// Generates Meshes from FFT data

    /// </summary>

    public void GenerateMeshFromAudioData()

    {

        this.meshObj = new GameObject[this.audioEngine.fftDataMagnitudes.Length];

        this.mFilters = new MeshFilter[this.audioEngine.fftDataMagnitudes.Length];

        this.mRenderers = new MeshRenderer[this.audioEngine.fftDataMagnitudes.Length];

        this.mColliders = new MeshCollider[this.audioEngine.fftDataMagnitudes.Length];

        this.meshes = new Mesh[this.audioEngine.fftDataMagnitudes.Length];

        this.vertices = new Vector3[this.audioEngine.fftDataMagnitudes.Length][];

        this.triangles = new int[this.audioEngine.fftDataMagnitudes.Length][];

        this.colors = new Color32[this.audioEngine.fftDataMagnitudes.Length][];

        Stopwatch timer = Stopwatch.StartNew();

        this.FindMaxPeakValue();

        for (int i=0; i < this.audioEngine.fftDataMagnitudes.Length; i++)

        {

            // Add GOs, MFs and MRs

            this.meshObj[i] = new GameObject("spectrumMesh" + i.ToString());

            this.meshObj[i].transform.parent = this.transform;

            this.mFilters[i] = meshObj[i].AddComponent<MeshFilter>();

            this.mFilters[i].name = "FFTData" + i.ToString();

            this.mRenderers[i] = meshObj[i].AddComponent<MeshRenderer>();

            this.mRenderers[i].material = Resources.Load("Materials/SpectrumMat") as Material;

            this.meshes[i] = new Mesh();

            // Create Spectrum polygons

            this.meshes[i].MarkDynamic(); // to get better performance when continually updating the Mesh

            this.SetVertices(i);

            this.SetMeshColors(i);

            this.SetTriangles(i);

            this.meshes[i].RecalculateBounds();

            this.meshes[i].RecalculateNormals();

            // It is important not to call Mesh.Optimize() here, because it will re-order the mesh vertices!

            // Only Mesh.OptimizeIndexBuffers is uncritical -

            // Mesh.Optimize() will basically also call Mesh.OptimizeReorderVertexBuffer() which will mess up the order and f\*ck things up I depend on later.

            this.meshes[i].OptimizeIndexBuffers();

            this.mFilters[i].mesh = this.meshes[i];

            // Add mesh colliders

            this.mColliders[i] = this.meshObj[i].AddComponent<MeshCollider>();

            this.mColliders[i].sharedMesh = this.mFilters[i].mesh;

            this.mColliders[i].cookingOptions = MeshColliderCookingOptions.WeldColocatedVertices;

        }

        timer.Stop();

        UnityEngine.Debug.Log("<SpectrumMeshGenerator> Generated mesh in " + (timer.Elapsed.Seconds + timer.Elapsed.Minutes \* 60).ToString() + " seconds.");

        // Generate Frequency legend

        SpectrumFreqLegend freqLegend = GameObject.Find("FreqLegend").GetComponent<SpectrumFreqLegend>();

        freqLegend.SetFreqLegend(this.audioEngine.fftFrequencies, this.edgeLengthOfRaster);

        // Init deformer instance

        this.deformer.MeshGenerated();

    }

    /// <summary>

    /// Sets the vertices of ground raster and FFT data peaks

    /// </summary>

    /// <param name="meshIdx">index of mesh to fill with vertices (= index of corresponding FFT chunk)</param>

    private void SetVertices(int meshIdx)

    {

        this.countOfRasterVertices = 2 \* (this.audioEngine.fftBinCount + 1);

        this.countOfPeakVertices = this.audioEngine.fftBinCount;

        this.vertices[meshIdx] = new Vector3[this.countOfRasterVertices + this.countOfPeakVertices];

        Vector3 center = transform.parent.position;

        // Add vertices

        // first the ground raster vertices, then all fft value peak vertices

        // Add ground raster vertices

        for (int i = 0; i < this.countOfRasterVertices / 2; i++)

        {

            Vector3 r1;

            r1.x = center.x + i \* this.edgeLengthOfRaster;

            r1.y = center.y;

            r1.z = center.z + meshIdx \* this.edgeLengthOfRaster;

            Vector3 r2;

            r2.x = center.x + i \* this.edgeLengthOfRaster;

            r2.y = center.y;

            r2.z = center.z + meshIdx \* this.edgeLengthOfRaster + this.edgeLengthOfRaster;

            this.vertices[meshIdx][i] = r1;

            this.vertices[meshIdx][i + this.audioEngine.fftBinCount + 1] = r2;

        }

        // Add peak vertices

        for (int i = 0; i < this.countOfPeakVertices; i++) {

            float peakValue = (float)this.audioEngine.fftDataMagnitudes[meshIdx][i] \* this.fftScalingFactor;

            Vector3 p;

            p.x = center.x + i \* this.edgeLengthOfRaster + this.edgeLengthOfRaster / 2;

            p.y = center.y + peakValue;

            p.z = center.z + meshIdx \* this.edgeLengthOfRaster + this.edgeLengthOfRaster / 2;

            this.vertices[meshIdx][this.countOfRasterVertices+i] = p;

        }

        this.meshes[meshIdx].vertices = this.vertices[meshIdx];

    }

    /// <summary>

    /// Resets all meshes materials to default

    /// </summary>

    public void ResetMeshColors()

    {

        for (int i = 0; i < this.audioEngine.fftDataMagnitudes.Length; i++)

        {

            this.mRenderers[i].material = Resources.Load("Materials/SpectrumMat") as Material;

        }

    }

    /// <summary>

    /// Find max value for Color lerp

    /// </summary>

    /// <returns>maximum of fftData peak values</returns>

    private void FindMaxPeakValue()

    {

        for (int i = 0; i < this.audioEngine.fftDataMagnitudes.Length; i++)

        {

            for (int j = 0; j < this.audioEngine.fftDataMagnitudes[i].Length; j++)

            {

                float value = (float)this.audioEngine.fftDataMagnitudes[i][j] \* this.fftScalingFactor;

                if (value > this.maxPeakValue)

                    this.maxPeakValue = value;

            }

        }

    }

    /// <summary>

    /// Check if provided peak value is higher than current peak value and sets accordingly

    /// </summary>

    /// <param name="value">peak value to check against</param>

    public void SetMaxPeakValue(float value)

    {

        if (value > this.maxPeakValue)

            this.maxPeakValue = value;

    }

    /// <summary>

    /// Creates color lerp for mesh peaks

    /// </summary>

    /// <param name="meshIdx">index of mesh to colorize (= index of corresponding FFT chunk)</param>

    public void SetMeshColors(int meshIdx)

    {

        this.colors[meshIdx] = new Color32[this.vertices[meshIdx].Length];

        for (int i = 0; i < this.vertices[meshIdx].Length; i++)

            this.colors[meshIdx][i] = Color.Lerp(Color.green, Color.red, this.vertices[meshIdx][i].y / this.maxPeakValue);

        this.meshes[meshIdx].colors32 = this.colors[meshIdx];

    }

    /// <summary>

    /// Creates triangles for the chosen mesh

    /// </summary>

    /// <param name="meshIdx">index of mesh to create triangles for (= index of corresponding FFT chunk)</param>

    public void SetTriangles(int meshIdx)

    {

        // 4 pyramid sides per fft bin (without base), 3 vertices indices per side

        this.triangles[meshIdx] = new int[this.audioEngine.fftBinCount \* 4 \* 3];

        for (int i = 0; i < this.triangles[meshIdx].Length - 12; i += 12)

        {

            this.triangles[meshIdx][i] = this.countOfRasterVertices + i / 12;

            this.triangles[meshIdx][i + 1] = i / 12;

            this.triangles[meshIdx][i + 2] = this.audioEngine.fftBinCount + 1 + i / 12;

            this.triangles[meshIdx][i + 3] = this.countOfRasterVertices + i / 12;

            this.triangles[meshIdx][i + 4] = this.audioEngine.fftBinCount + 2 + i / 12;

            this.triangles[meshIdx][i + 5] = 1 + i / 12;

            this.triangles[meshIdx][i + 6] = this.countOfRasterVertices + i / 12;

            this.triangles[meshIdx][i + 7] = this.audioEngine.fftBinCount + 1 + i / 12;

            this.triangles[meshIdx][i + 8] = this.audioEngine.fftBinCount + 2 + i / 12;

            this.triangles[meshIdx][i + 9] = this.countOfRasterVertices + i / 12;

            this.triangles[meshIdx][i + 10] = 1 + i / 12;

            this.triangles[meshIdx][i + 11] = i / 12 ;

        }

        this.meshes[meshIdx].triangles = this.triangles[meshIdx];

    }

    /// <summary>

    /// Scales the whole mesh

    /// </summary>

    /// <param name="offset">positive or negative offset applied to y-scale</param>

    public void ScaleMeshY(float newY)

    {

        this.transform.localScale = new Vector3(this.transform.localScale.x, newY, this.transform.localScale.z);

    }

    /\*

    // Visualize vertices for Scene view debugging

    // (huge performance killer for lots of vertices, will freeze unity if used for the whole fft data set)

    private void OnDrawGizmos()

    {

        if (this.audioEngine == null)

        {

            return;

        }

        // Draw Spheres for vertices

        var transform = this.transform;

        foreach (var vert in this.meshes[0].vertices)

        {

            Gizmos.color = Color.red;

            Gizmos.DrawSphere(transform.TransformPoint(vert), 0.003f);

        }

    }\*/

}

**ToolHandler.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

using UnityEngine.EventSystems;

using UnityEngine.UI;

using Valve.VR;

using Valve.VR.Extras;

public enum ToolType

{

    SpectrumPainter,

    SpectrumPencil,

    SpectrumEraser

}

public class ToolHandler : MonoBehaviour

{

    public SteamVR\_LaserPointer laserPointer;

    private bool triggerDown;

    private Tool selectedTool;

    private List<Vector3> pointsToDraw;

    private LineRenderer lr;

    public float toolRadius;

    public float toolabsoluteValue;

    public Tool[] tools;

    public ToolType SelectedToolType { get => selectedTool.type; }

    public bool TriggerDown { get => triggerDown; set => triggerDown = value; }

    private SpectrumDeformer deformer;

    private SpectrumHelper helper;

    private void Awake()

    {

        this.lr = GetComponent<LineRenderer>();

        this.helper = GameObject.Find("Spectrum").GetComponent<SpectrumHelper>();

        this.laserPointer.PointerClick += this.PointerClick;

        this.laserPointer.PointerIn += this.PointerIn;

        this.toolRadius = 0.002f;

        this.laserPointer.thickness = this.toolRadius;

        this.lr.startWidth = this.toolRadius;

        this.lr.endWidth = this.toolRadius;

        this.toolabsoluteValue = 0.05f;

        this.helper.ShowToolValueIndicator(1, this.toolabsoluteValue);

        // Init Tools

        this.tools = new Tool[Enum.GetNames(typeof(ToolType)).Length];

        this.tools[(int)ToolType.SpectrumPainter] = new Tool(ToolType.SpectrumPainter, new Color32(20, 20, 255, 100), GameObject.Find("IconSpectrumPainter").GetComponent<Image>());

        this.tools[(int)ToolType.SpectrumPencil] = new Tool(ToolType.SpectrumPencil, new Color32(255, 185, 255, 100), GameObject.Find("IconSpectrumPencil").GetComponent<Image>());

        this.tools[(int)ToolType.SpectrumEraser] = new Tool(ToolType.SpectrumEraser, new Color32(255, 50, 50, 100), GameObject.Find("IconSpectrumEraser").GetComponent<Image>());

        this.changeTool(ToolType.SpectrumPainter);

        this.triggerDown = false;

    }

    private void Start()

    {

        this.deformer = GameObject.Find("SpectrumMesh").GetComponent<SpectrumDeformer>();

        this.pointsToDraw = new List<Vector3>();

    }

    public void changeTool(ToolType tool)

    {

        for (int i=0; i < this.tools.Length; i++)

        {

            if (i == (int)tool)

            {

                this.selectedTool = this.tools[i];

                this.selectedTool.image.color = this.tools[i].color;

                this.laserPointer.color = this.tools[i].color;

                Debug.Log("<ToolHandler> Changed tool to " + Enum.GetName(typeof(ToolType), i));

            } else {

                this.tools[i].image.color = new Color(255f, 255f, 255f);

            }

        }

    }

    public void nextTool()

    {

        if ((int)this.selectedTool.type < this.tools.Length - 1)

            this.changeTool((ToolType)Enum.Parse(typeof(ToolType), Enum.GetName(typeof(ToolType), (int)this.selectedTool.type + 1)));

        else

            this.changeTool((ToolType)Enum.Parse(typeof(ToolType), Enum.GetName(typeof(ToolType), 0)));

    }

    public void prevTool()

    {

        if ((int)this.selectedTool.type > 0)

            this.changeTool((ToolType)Enum.Parse(typeof(ToolType), Enum.GetName(typeof(ToolType), (int)this.selectedTool.type - 1)));

        else

            this.changeTool((ToolType)Enum.Parse(typeof(ToolType), Enum.GetName(typeof(ToolType), this.tools.Length - 1)));

    }

    public void PointerClick(object sender, PointerEventArgs e)

    {

        if (this.SelectedToolType == ToolType.SpectrumPencil && e.target.name.StartsWith("FFTData"))

        {

            deformer.DeformMeshPoint(e.point, Vector3.up, this.toolRadius, this.toolabsoluteValue);

            return;

        }

    }

    public void PointerIn(object sender, PointerEventArgs e)

    {

        if ((this.SelectedToolType == ToolType.SpectrumPainter || this.SelectedToolType == ToolType.SpectrumEraser) && this.triggerDown && e.target.name.StartsWith("FFTData"))

        {

            this.pointsToDraw.Add(e.point);

            // Visualize trail

            this.lr.enabled = true;

            this.lr.positionCount = this.pointsToDraw.Count;

            this.lr.SetPositions(this.pointsToDraw.ToArray());

            return;

        }

    }

    public void TriggerUp()

    {

        if ((this.SelectedToolType == ToolType.SpectrumPainter || this.SelectedToolType == ToolType.SpectrumEraser) && this.pointsToDraw.Count > 0) {

            // Remove duplicates

            this.pointsToDraw = this.pointsToDraw.Distinct().ToList();

            if (this.SelectedToolType == ToolType.SpectrumPainter)

                deformer.DeformMeshMultiplePoints(this.pointsToDraw, Vector3.up, this.toolRadius, this.toolabsoluteValue);

            else if (this.SelectedToolType == ToolType.SpectrumEraser)

                deformer.DeformMeshMultiplePoints(this.pointsToDraw, Vector3.up, this.toolRadius, 0f);

            this.lr.positionCount = 0;

            this.lr.enabled = false;

            this.pointsToDraw.Clear();

            return;

        }

    }

    public void SetToolRadiusWithOffset(float offset)

    {

        if (this.toolRadius + offset >= 0.002f && this.toolRadius + offset < 8f)

        {

            this.toolRadius += offset;

            this.laserPointer.thickness = this.toolRadius;

            this.lr.startWidth = this.toolRadius;

            this.lr.endWidth = this.toolRadius;

        }

    }

    public void SetToolAbsoluteValueOffset(float offset)

    {

        if (this.toolabsoluteValue + offset >= 0 && this.toolabsoluteValue + offset < 6) //TODO check for maximum value according to bit depth

        {

            this.toolabsoluteValue += offset;

            this.helper.ShowToolValueIndicator(120, this.toolabsoluteValue);

        }

    }

}

public class Tool

{

    public ToolType type;

    public Color32 color;

    public Image image;

    public Tool(ToolType type, Color color, Image image)

    {

        this.type = type;

        this.color = color;

        this.image = image;

    }

}

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. Slater, Mel: „Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments”, Philosophical Transactions of the Royal Society (2009), 364, 3549–3557doi:10.1098/rstb.2009.0138, EVENT Lab, Institute for Brain, Cognition and Behavior(IR3C), ICREA-University of Barcelona, 08035 Barcelona, Spain; Department of Computer Science, University College London, London WCIE 6BT, UK, S. 3551ff [↑](#footnote-ref-5)
5. 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-6)
6. Vgl. InvisionStudio: „The History of Virtual Reality (VR)”, <https://invisionstudio.com/the-history-of-vr-virtual-reality/>, letzter Abruf: 13.07.2020 [↑](#footnote-ref-7)
7. Vgl. Heilig, Morton Leonard: „El Cine del Futuro: The Cinema of the Future” – Presence, Volume 1, Number 3, 1992, S. 279-294, S. 289 [↑](#footnote-ref-8)
8. Vgl. Ebd. S. 284ff [↑](#footnote-ref-9)
9. 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-10)
10. Vgl. Ebd. S. 760-761 [↑](#footnote-ref-11)
11. Vgl. Business Insider: „The incredible story of the 'Virtual Boy' — Nintendo's VR headset from 1995 that failed spectacularly”, 26.03.2018 – <https://www.businessinsider.com/nintendo-virtual-boy-reality-3d-video-games-super-mario-2018-3?IR=T>, letzter Abruf: 13.07.2020 [↑](#footnote-ref-12)
12. Vgl. YouTube: „The Untold Story Of Virtual Reality On The Sega Genesis - The Unreleased The Sega VR Headset”, Wrestling With Gaming – <https://www.youtube.com/watch?v=3UN_pN9ZU8Y>, letzter Abruf: 13.07.2020 [↑](#footnote-ref-13)
13. Vgl. Dörner, Ralf; Broll, Wolfgang; Grimm, Paul; Jung, Bernhard: „Virtual und Augmented Reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität“

    Springer-Verlag, Wiesbaden (2019), S. 28ff [↑](#footnote-ref-14)
14. 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-15)
15. Vgl. Ebd. S. 213 [↑](#footnote-ref-16)
16. Vgl. Ebd. S. 183-184 [↑](#footnote-ref-17)
17. Vgl. Ebd. S. 184-185 – Adelstein et al. 2003 & 2006, Jerald 2009, Ellis et al. 1999, 2004. Mania et al. 2004 [↑](#footnote-ref-18)
18. 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-19)
19. 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-20)
20. 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-21)
21. Vgl. Unity Technologies: „Multiplatform | Unity“ – <https://unity.com/features/multiplatform>, letzter Abruf: 24.06.2020 [↑](#footnote-ref-22)
22. *Anmerkung: In der Vergangenheit konnten alterrnativ die Skriptsprachen Boo und UnityScript verwendet werden. Diese sind aber als veraltet klassifiziert.* [↑](#footnote-ref-23)
23. Unity Asset Store: „SteamVR Plugin” – <https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-24)
24. Vgl. Github – NAudio repository – <https://github.com/naudio/NAudio>, letzter Abruf: 23.06.2020 [↑](#footnote-ref-25)
25. 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-26)
26. Vgl. FFTW: „FFT Benchmark Results“ – <http://www.fftw.org/speed/>, letzter Abruf: 13.06.2020 [↑](#footnote-ref-27)
27. 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-28)
28. Vgl. Github: C# wrapper for FFTW – <https://github.com/tszalay/FFTWSharp>, letzter Abruf: 23.06.2020 [↑](#footnote-ref-29)
29. 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-30)
30. 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-31)
31. *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-32)
32. Vgl. National Instruments: *„*Understanding FFTs and Windowing”, <https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf>, letzter Abruf: 12.05.2020 [↑](#footnote-ref-33)
33. 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-34)
34. *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-35)
35. 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-36)
36. Vgl. Unity Documentation: “Coroutine” – <https://docs.unity3d.com/ScriptReference/Coroutine.html>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-37)
37. Vgl. Unity Documentation: „C# Job System Overview“ – <https://docs.unity3d.com/2019.3/Documentation/Manual/JobSystemOverview.html>, letzter Abruf: 07.07.2020 [↑](#footnote-ref-38)
38. 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-39)
39. *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-40)
40. 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-41)
41. Vgl. Ebd. [↑](#footnote-ref-42)
42. Vgl. Ebd. [↑](#footnote-ref-43)
43. 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-44)