Entwicklung eines plattformübergreifenden Audioplugins zur Postproduktion einer Parabolmikrofonaufnahme

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Engineering (M.Eng.)

von

Pascal Escher
33717
Audiovisuelle Medien Master
Schwerpunkt Ton

Erstprüfer:	Prof. Oliver Curdt
Zweitprüfer:	Prof. Dr. Simon Wiest
Praxisbetreuer:	Dr. Helmut Wittek

Hochschule der Medien - Fakultät Elektronische Medien

13.12.2018

Ehrenwörtliche Erkärung

„Hiermit versichere ich, Pascal Escher, ehrenwörtlich, dass ich die vorliegende Masterarbeit mit dem Titel: „Entwicklung eines plattformübergreifenden Audioplugins zur Postproduktion einer Parabolmikrofonaufnahme“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Stuttgart, den 13.12.2018

_____

Pascal Escher

Kurzfassung

Die fortschreitende Digitalisierung im Studioalltag ermöglicht es erfahrenden Tonmeistern, wie auch Einsteigern, zunehmend Einfluss auf Audioaufnahmen in Form der Postproduktion zu nehmen. Dabei kann sowohl die Klangästhetik, wie auch die Klangqualität verändert und verbessert werden. Aus diesem Grund bieten Hersteller zunehmend angepasste Software in Form von Audioplugins zu ihren Produkten an.

Gegenstand der hier vorgestellten Arbeit ist die Konzeptionierung und Entwicklung eines plattformunabhängigen Audioplugins für das Parabolmikrofon der Firma Schoeps. Zu Beginn der Masterarbeit wird ein historischer Rückblick auf die Entstehung des Parabolmikrofons und seinen Einsatzbereich vermittelt. Die grundlegende Technik wird beschrieben und der heutige Einsatzzweck beleuchtet. Zur Verbesserung der Aufnahmequalität von Parabolmikrofonen werden die Vorteile und Möglichkeiten der digitalen Postproduktion in Form eines Audioplugins dargelegt. Darauf aufbauend werden unterschiedliche Pluginformate vorgestellt und die Funktionalität der JUCE-Plattform beschrieben, welche zur Umsetzung eines Audioplugins genutzt werden kann. Aus der Analyse der akustischen Eigenschaften werden die technischen Vor- und Nachteile eines Parabolsystems abgeleitet. Diese Erkenntnisse liefern die zu erfüllenden Anforderungen und Grundvoraussetzungen für ein Audioplugin. Der Kern der Arbeit umfasst die Implementierung eines auf die Anforderungen ausgerichteten Plugins. Die Funktionalität wird daraufhin evaluiert und die Ergebnisse in einer abschließenden Zusammenfassung bewertet. Zuletzt folgt ein Ausblick auf mögliche Erweiterungen und zukünftige Einsatzgebiete.

Schlagwörter: Parabolmikrofon, Hohlspiegelmikrofon, Audioplugin, JUCE, VST, Audio Unit

Abstract

The progressive digitalisation within recording environments allows experienced sound engineers as well as beginners to increase their influence on audio recordings during post production. Changes and improvements can be made focussing on sound aesthetics and sound quality. For this purpose, audio manufacturers increasingly offer customized software, like audio plugins, for their products.

Object of this work is the design and development of a platform-independent audio plugin for the parabolic microphone of the company Schoeps. The first part of the master thesis on hand gives a historical review of the emergence of the parabolic microphone and its range of application. Basic technical aspects and its use nowadays are set out. To improve the recording quality of parabolic microphones the advantages and opportunities of a digital post production in the form of an audio plugin will be presented. Based on that, different plugin formats are introduced and the functionality of the JUCE framework is described, which can be used to implement audio plugins. By analysing the acoustic properties, the resulting technical disadvantages and advantages of a parabolic system are established. The findings allow to define the requirements and basic prerequisite for an audio plugin. The main part of this work focuses on the implementation of a plugin based on these requirements. Subsequently the functionality is evaluated and the results are valued in a final summary. Finally, an outlook on possible extensions and future areas of application is presented.

Keywords: Parabolic microphone, audio plug-in, JUCE, VST, Audio Unit

Inhaltsverzeichnis

1 Einleitung und Motivation **1**

1.1 Ziele . . . 2

1.2 Gliederung . . . 2

2 Grundlagen **3**

2.1 Das Parabolmikrofon und der Hohlspiegel . . . 3

2.1.1 Geschichtliche Entwicklung . . . 3

2.1.2 Technische Funktionsweise . . . 6

2.1.2.1 Der akustische Hohlspiegel . . . 7

2.1.2.2 Das Parabolmikrofon . . . 9

2.1.3 Das Parabolmikrofon der Firma Schoeps . . . 9

2.1.3.1 Klangeigenschaften des verwendeten Mikrofons . . . 10

2.1.3.2 Die Windschutze . . . 12

2.1.3.3 Der Hohlspiegel . . . 12

2.1.3.4 Das Hohlspiegelstativ . . . 13

2.2 Akustische Einflüsse auf die Parabolmikrofonaufnahme . . . 13

2.2.1 Dissipation . . . 13

2.2.2 Windgeräusche . . . 14

2.3 Digitale Postproduktion . . . 15

2.3.1 Vorteile digitaler Filter . . . 16

2.3.2 IIR-/FIR-Filter . . . 16

2.3.3 Pluginformate . . . 17

2.3.3.1 Virtual Studio Technology . . . 17

2.3.3.2 Avid Audio eXtension/Real-Time AudioSuite . . . 18

2.3.3.3 Audio Unit . . . 19

2.3.4 Das Juce-Framework als Implementierungsgrundlage . . . 19

2.3.4.1 Projucer . . . 20

2.3.4.2 Struktur und Funktionsweise . . . 22

2.3.4.3 Das Schoeps-Framework . . . 22

3 Analyse des Parabolsystems **23**
3.1 Technische Messdaten des Parabolmikrofons . . . 23
3.1.1 Frequenzganganalyse . . . 23
3.1.2 Polardiagramm . . . 25
3.1.3 Auswirkung der Windschutze auf die Frequenzantwort . . . 27
3.2 Technische Vor- und Nachteile . . . 28
3.3 Anforderungen an das Audioplugin . . . 29
3.3.1 Auswahl des Pluginformats und des Betriebssystems . . . 29
3.3.2 Digitale Anpassung des Frequenzgangs . . . 33
3.3.3 Hilfreiche Parameter zur Anpassung des Eingangsignals . . . 35
3.3.4 Visualisierung der Frequenzganganpassungen . . . 36
3.3.5 Presets für Audioplugins . . . 37

4 Plugin-Entwicklung mit dem JUCE-Framework **38**
4.1 Prinzipien des Clean Codes . . . 38
4.2 Coding Guidelines . . . 39
4.2.1 Pointer . . . 40
4.2.2 Namenskonventionen . . . 40
4.2.3 Dokumentation . . . 40
4.2.4 Diverses . . . 41
4.3 Plugin-Implementierung . . . 42
4.3.1 Konzeption und Entwicklung der Benutzeroberfläche . . . 42
4.3.1.1 Die Gestaltprinzipien . . . 44
4.3.1.2 Die Schoeps-LooknFeel-Klasse . . . 46
4.3.2 Struktur und Pluginaufbau . . . 46
4.3.2.1 Der ParabolaProcessor . . . 49
4.3.2.2 Der ParabolaEditor . . . 53
4.3.2.3 Die Filterklasse . . . 54
4.3.3 Entwicklung der digitalen Filter . . . 54
4.3.3.1 Parameterwerte der Filter . . . 55
4.4 Umsetzung der plattformübergreifenden Funktionalität . . . 57

5 Evaluation **60**
5.1 Auswertung der ersten Testergebnisse . . . 60
5.2 Auswertung der zweiten Testergebnisse . . . 61
5.3 Auswertung der Testsessions . . . 62

6 Zusammenfassung und Ausblick **64**
6.1 Fazit . . . 64
6.2 Ausblick . . . 65

Abbildungsverzeichnis

2.1 Ausschnitt des Patentantrags von Alfred Marshall Mayer, 1880 4
2.2 Militärische Hohlspiegelmikrofone aus Beton in Dungeness (England) . 5
2.3 Hohlspiegelmikrofon in der Messstrecke eines aeroakustischen Windkanals 6
2.4 Funktionsschema eines paraboloiden und ellipsoiden Hohlspiegelmikrofons 7
2.5 Zusammenhang zwischen Messabstand und Mikrofonposition 8
2.6 Das Parabolmikrofon der Firma Schoeps in der Frontansicht 9
2.7 Kompaktmikrofon CCM 4 10
2.8 Frequenzgang des CCM 4 10
2.9 Hi-Wind Cover der Firma Telinga für den Hohlspiegel 13
2.10 Frequenzabhängige Luftdämpfung in 30 m Abstand bei unterschiedlicher Luftfeuchtigkeit 14
2.11 Übersichtsanzeige in Projucer für ein neues Projekt 20
2.12 Globale Pfadsuche der SDKs für macOS 21

3.1 Frequenzantwort des Parabolasystems mit dem CCM 4 24
3.2 Polardiagramm des Parabolsystems mit CCM 4 26
3.3 Einfluss der verwendeten Windschutze auf die Frequenzantwort 27
3.4 Marktanteile von DAWs nach ask.audio 30
3.5 Marktanteile von DAWs nach songwriter24.de 31
3.6 Frequenzantwort dreier Hochpassfilter 4. Ordnung mit der f_g= 70 Hz . 34
3.7 Pro-Q 2 Filter der Firma FabFilter 36

4.1 Parabola Mockup 42
4.2 Vertikal-Horizontal-Mix 43
4.3 About View 44
4.4 Parab EQ 1.0 Release-Version 45
4.5 Schema des Audioplugins von juce.com 46
4.6 UML-Klassendiagramm des Parab EQ-Plugins 48
4.7 Aktivitätsdiagramm der Audioverarbeitung im Parab EQ-Plugin . . . 51
4.8 Exporter-Übersicht im Projucer 58

5.1 Alte Parameterbezeichnungen 60

5.2 Tooltip für den Distanz-Parameterslider 61
5.3 Info-Buttons und Überschriften der Einstellungsbereiche 62
5.4 Ausschnitt der Testsession in Logic Pro X 63

Listingverzeichnis

4.1 Line-Klasse aus „Der pragmatische Programmierer“ 39
4.2 Dokumentation der getFactorForFrequency-Methode 41
4.3 PrepareToPlay-Methode in der ParabolaProcessor-Klasse 49
4.4 ProcessBlock-Methode 50
4.5 Methoden zur Speicherung des Pluginszustands 53
4.6 parameterChanged-Methode 55

Abkürzungsverzeichnis

Die dargestellten Abkürzungen entsprechen geläufigen Abkürzungen aus dem Audio- oder Informatikbereich.

Abkürzung	Vollständiger Name
AAX	Avid Audio eXtension
AU	Audio Unit
CCIR	Comité Consultatif International des Radiocommunications
CI	Corporate Identity
DAW	Digital Audio Workstation
DLL	Dynamic Link Library
DSP	Digitaler Signalprozessor
FIR	Finite Impulse Response
GUI	Graphical User Interface
HWC	High Wind Cover
I/O	Input/Output
IDE	Integrated Development Environment
IIR	Infinite Impulse Response
ITU-R	International Telecommunication Union des Radiocommunications
JUCE	Jules' Utility Class Extensions
MIDI	Musical Instrument Digital Interface
MVC	Model View Controller
NDK	Native Development Kit
RTAS	Real-Time AudioSuite
SDK	Software Development Kit
THD	Total Harmonic Distortion
UML	Unified Modeling Language
VST	Virtual Studio Technology

Danksagung

An dieser Stelle möchte ich mich besonders bei der Firma Schoeps für die Unterstützung und Betreuung im Rahmen der Thesis bedanken. Ein großes Dankeschön gilt hier vor allem Hannes Dieterle und Helmut Wittek für das Zurverfügungstellen von Equipment und den konstanten Austausch zu vielen technischen und inhaltlichen Schwerpunkten.

Außerdem möchte ich Danke sagen an meine Betreuer Professor Oliver Curdt und Professor Dr. Simon Wiest, wie auch an meine Familie und Freundin Miriam Skowronek für die fachliche und persönliche Unterstützung.

1. Einleitung und Motivation

Bereits vor über 100 Jahren gab es die ersten Versuche zur Ortung und akustischen Verstärkung von Schallquellen. Daraus entwickelten sich über die Jahre die ersten Parabol- beziehungsweise Hohlspiegelmikrofone. Diese fanden in unterschiedlichen Domänen ihre Verwendung. Die Einsatzgebiete veränderten sich im Laufe des 20. Jahrhunderts vom militärischen Gebrauch zur Ortung von feindlichen Fahrzeugen, über Außengeräuschmessungen von Fahrzeugen in der Automobilbranche bis hin zur Anwendung im Privatbereich.

Heutzutage ist das Einsatzgebiet weiterhin vielfältig vorzufinden. Es reicht von der Aufnahme von Tier-/Naturgeräuschen, über Sportaufnahmen, bis hin zu Motorgeräuschen und räumlich distanzierten Sprachaufnahmen. Durch das breite Spektrum an Klangobjekten entstehen unterschiedliche Erwartungen und Vorstellungen beim Anwender bezüglich der Aufnahmequalität. Diese Anforderungen werden durch Umgebungsfaktoren beeinflusst, welche Tonaufnahmen im Außenbereich mit sich bringen. Für den Tonschaffenden sind dadurch bestimmte Bedingungen während der Aufnahme, wie das Verhältnis von Direkt-/Diffusschall und Objektentfernung nicht veränderbar. Diese müssen somit nachträglich im Sinne einer Verbesserung der Audioqualität in der Postproduktion bearbeitet werden.

Durch die zunehmende Digitalisierung in der Tontechnik ist es möglich, diese Anforderungen mit Hilfe einer Softwarelösung zu erfüllen. Moderne Audioplugins können hierbei für spezielle Aufgaben programmiert und angewandt werden. Sie bringen technische Vorteile in Form von Flexibilität, Wiederverwendbarkeit und Zuverlässigkeit.
Die Firma Schoeps GmbH hatte die Intention für ihr Parabolsystem, aus den zuvor beschriebenen Gründen, ein Audioplugin für die Postproduktion einzusetzen. Die vorliegende Arbeit beschäftigt sich aus diesem Grund mit der genauen Analyse der akustischen Eigenschaften des Systems, den Aufnahmeanforderungen im Live-Einsatz und den daraus resultierenden Parametern für die Spezifikation eines Audioplugins, welches in Kooperation mit der Firma Schoeps entwickelt wurde.

1.1 Ziele

Das Ziel der Thesis ist, einen Überblick über die geschichtliche Entstehung und heutige Verwendung eines Parabolmikrofons aufzuzeigen und das Parabola-Produkt der Firma Schoeps vorzustellen. Die akustischen Vor- und Nachteile eines Parabolmikrofons und der damit verbundenen Parabolmikrofonaufnahme sollen vermittelt und die Verbesserungsmöglichkeiten der digitalen Postproduktion, basierend auf einem Audioplugin, demonstriert werden. Die gewonnenen Erkenntnisse werden für die Implementierung eines parametrischen Equalizer-Plugins, zur Postproduktion einer Parabolmikrofonaufnahme, herangezogen und umgesetzt.
Dieses Plugin soll mit vereinfacht bezeichneten Parametern, dem möglicherweise fachfremden Anwender, wie beispielsweise Ornithologen[1] erlauben, eine Parabolmikrofonaufnahme klanglich anzupassen und dadurch zu verbessern. Dazu soll der Nutzer neben voreingestellten Presets für verschiedene Anwendungsfälle auch visuelles Feedback über seine Interaktionen erhalten. Dies wird über eine Frequenzgangkurve aller eingeschalteter Filter im Plugin realisiert.

1.2 Gliederung

Zu Beginn der Thesis erfolgt ein Überblick über die geschichtliche beziehungsweise technische Entwicklung des Parabolmikrofons. Hierbei wird präzise auf den akustischen Hohlspiegel, das verwendete Mikrofon der Firma Schoeps und auf die verfügbaren Windschutze eingegangen. Es werden akustische Einflussfaktoren besprochen, welche frequenzabhängigen Einfluss auf das Aufnahmesignal eines Parabolmikrofons haben. Die Rolle der digitalen Postproduktion und die Vorteile digitaler Filter, wie auch auf Unterschiede bezüglich der Implementierung von selbigen Filtern, wird eingegangen. Es folgt ein Überblick über gängige Pluginformate und die Vorstellung des zur Implementierung verwendeten JUCE-Frameworks. Darauf aufbauend findet eine Analyse aller zur Umsetzung eines Plugins notwendigen Anforderungen statt, die in der konkreten Implementierung eines plattformunabhängigen Plugins umgesetzt werden. Hierbei werden einzelne Aspekte der Entwicklung sowie der zu Grunde liegende Programmieransatz, die Oberflächengestaltung, der strukturelle Aufbau und die Implementierung der Filter erläutert. Anschließend findet eine Auswertung der ersten Testergebnisse und die konkrete Umsetzung der gewonnenen Erkenntnisse statt. Zuletzt folgt eine Zusammenfassung und Auswertung der Implementierung in Bezug auf die Anforderungsanalyse und Evaluationsergebnisse. Abgeschlossen wird die Thesis durch einen Ausblick auf technische Erweiterungen und zukünftige Anwendungsmöglichkeiten.

[1] Vogelkundler

2. Grundlagen

Das folgende Kapitel vermittelt einen Überblick über die zur späteren Implementierung und Analyse notwendigen theoretischen Grundlagen und gibt darüber hinaus Einblicke in die zur Realisierung notwendigen Schritte. Es werden die technischen Voraussetzungen zur Umsetzung eines Audioplugins diskutiert, sowie auf das bereits vorhandene Parabolmikrofonsystem der Firma Schoeps eingegangen.

2.1 Das Parabolmikrofon und der Hohlspiegel

Folgender Abschnitt erläutert die geschichtliche Entwicklung des Parabolmikrofons im Zusammenhang mit der Entwicklung des Hohlspiegels. Dabei wird Bezug auf die ersten Versuche richtungsbezogener Mikrofonierung, der späteren militärischen Verwendung von Hohlspiegeln, bis hin zum heutigen Entwicklungsstand und Einsatzzweck von Parabolspiegelsystemen beziehungsweise Hohlspiegelmikrofonen genommen.

2.1.1 Geschichtliche Entwicklung

Bereits Ende des 19. Jahrhunderts wurden die ersten Versuche zur verbesserten Lokalisation von Schallquellen durchgeführt. Der amerikanische Wissenschaftler Alfred Marshall Mayer ließ sich 1880 zur Lokalisation von Nebelhörner seine verbesserte Form des tragbaren Topophones[1] patentieren. Die Schallquelle wird dabei nicht elektronisch, sondern bauformbedingt innerhalb der Konstruktion durch Mehrfachreflexionen verstärkt. Dazu nehmen zwei Resonatoren, die in einem konstanten Abstand zueinanderstehen, die Schwingungen des Zielobjekts auf und geben diese über einen einzelnen Schlauch an die Ohren des Zuhörers weiter. Steht der Zuhörer nicht im gleichen Abstand der Resonatoren zum Objekt, würden nach Mayer Augmentationen oder Diminutationen der Intensität beziehungsweise Interferenzen[2] entstehen. Diese könnten durch Drehung der Apparatur und letztendlich der Lokalisation der Schallquelle eliminiert werden. Für die erfolgreiche Durchführung mussten die Resonatoren dafür im Voraus auf die entsprechende Tonhöhe des Objekts eingestimmt werden.

[1] Hörrohr-ähnliche Konstruktion, die auf Mehrfachreflexionen basiert.
[2] Amplitudenmodulation bei Überlagerung von zwei oder mehr Wellen.

Dies wurde durch eine horizontale Verschiebung der Resonatoren erreicht. (Vgl. im Englischen S. 2 [May80]).

Folgende Abbildung zeigt die zweite Konstruktionszeichnung des Patentantrags von Alfred Marshall Mayer:

Abbildung 2.1: Ausschnitt des Patentantrags von Alfred Marshall Mayer, 1880
Quelle: S. 1 [May80], abgerufen am 02.07.2018

In Europa wurden Anfang des 20. Jahrhunderts die ersten Systeme der akustischen Lokalisation für militärische Zwecke eingesetzt. Die statischen Hohlspiegelsysteme aus Beton sollten es ermöglichen, feindliche Flugzeuge außerhalb der Sichtweite zu orten. Hohlspiegel mit mehreren Metern Durchmesser wurden aus diesem Grund küstennah angebracht und mit einem zentral vor dem Spiegel positionierten Mikrofon abgenommen. Zusätzlich mussten Personen vor dem Hohlspiegel mittels eines großen Stethoskops auf Fluggeräusche lauschen beziehungsweise den Brennpunkt der Schallwellen ermitteln. Mit Hilfe des lautesten Signals und einer Skala auf dem Hohlspiegel konnte die Richtung ermittelt werden. Auf Grund von Weiterentwicklungen bei der Flugzeugtechnik verkürzte sich die Zeit zur Ortung der Motorengeräusche allerdings auf wenige Minuten. Zusätzlich sorgte die Erfindung des Radars dafür, dass die Hohlspiegel im Krieg nicht zum Einsatz kamen. Grund dafür ist die höhere Geschwindigkeit von elektromagnetischen Wellen im Radiofrequenzbereich (299 792,458 km/s) im Vergleich zur Schallgeschwindigkeit (343,2 m/s bei 20°C) und die damit verbundene höhere Reichweite und Zuverlässigkeit der Ortung. Durch die Reflexion der elektromagnetischen Wellen am Flugzeugkörper konnte zusätzlich die Entfernung der Objekte berechnet und somit die Lokalisation verbessert werden. (Vgl. S. 276-277 [GC10])

Folgende Abbildung zeigt mehrere Hohlspiegelmikrofone, die 1928 in Dungeness erbaut wurden:

Abbildung 2.2: Militärische Hohlspiegelmikrofone aus Beton in Dungeness (England)
Quelle: Photo © Paul Russon (cc-by-sa/2.0), abgerufen am 04.07.2018

Auf der linken Seite ist eine 60 Meter lange Hohlspiegelwand ersichtlich, die ohne Stethoskop genutzt werden konnte und mit 20 Mikrofonen ausgestattet war. Sie wurde verwendet, um typische Flugsignale, die bei ca. 370 Hz liegen, zu erkennen. (Vgl. S. 276 [GC10]).
Weitere Systeme zur Luftüberwachung wurden in Japan mit der japanischen Kriegstuba und der deutschen Wehrmacht mit dem Ringrichtungshörer (RRH) entwickelt. Beide Systeme basieren dabei, ähnlich wie das Topophone von Alfred Mayer, auf Mehrfachreflexionen zur Signalverstärkung.

Ein weiterer Verwendungszweck ist die Anwendung im aeroakustischen Windkanal zur Messung von Windströmungen. Hierbei werden auf offenen Messstrecken die Außengeräusche der Testfahrzeuge unter Windeinfluss gemessen. Für geschlossene Messstrecken ist dies nicht möglich, da das Messsystem hierfür in der Strömung installiert werden müsste (vgl. S. 290 [Gen10]). Durch die Installation des Mikrofons in der Strömung werden negative Aspekte, wie Störgeräusche durch die ankommende

Strömung am Mikrofon, wie durch Strömungswechseldrücke entstehende Verzerrungen der Messmembran, verstärkt (vgl. S. 287 [Gen10]).

Folgende Abbildung zeigt die Installation eines Hohlspiegelmikrofons zur Außengeräuschmessung eines Kraftfahrzeugs:

Abbildung 2.3: Hohlspiegelmikrofon in der Messstrecke eines aeroakustischen Windkanals
Quelle: General Aspects of Vehicle Aeroacoustics, Martin Helfer (vgl. S. 15 [Hel05])

Eine Erweiterung der beschriebenen Technik ist der Array-Hohlspiegel. Dieser ermöglicht es, mehrere Brennpunkte in der Spiegelmitte durch eine Anreihung von Mikrofonen aufzunehmen. Dadurch wird eine bessere akustische Abbildung der untersuchten Oberfläche erreicht (vgl. S. 141 [ZLR05]).

2.1.2 Technische Funktionsweise

Folgender Abschnitt erläutert die Funktionsweise eines akustischen Hohlspiegels und eines Parabolmikrofons. Verschiedene Bauformen des Hohlspiegels werden diskutiert. Dabei werden die Anordnung des Mikrofons und die Auswirkung auf die Klangeigenschaften gegenübergestellt.

2.1.2.1 Der akustische Hohlspiegel

Akustische Hohlspiegel werden generell in drei Bauarten unterschieden: Ellipsoid, Paraboloid und Kugel. Unterschiede ergeben sich neben der Anzahl der Brennpunkte der reflektierten Schallwellen auch in der Anwendung. Das Ellipsoid eignet sich auf Grund seiner zwei Brennpunkte zur Ortung im Nahfeld. Dies ist dadurch begründbar, dass alle reflektierten Wellen nach dem ersten Brennpunkt phasengleich beim zweiten Brennpunkt eintreffen. Es kommt hierbei zu einer Verstärkung der sich zwischen den zwei Punkten bewegenden Signale (vgl. S. 141 [ZLR05]).

Eine Sonderform des Ellipsoids stellt die Kugel dar. Sie verfügt nur über einen Brennpunkt und muss sich aus diesem Grund am selben Punkt wie die Schallquelle befinden. Für akustische Zwecke kann sie somit in vielen Fällen nicht eingesetzt werden. Das Paraboloid ist hingegen für Schallquellen im Fernfeld geeignet, da hierbei der zweite Brennpunkt ins Unendliche verschoben wird und parallele Schallquellen zu Brennpunktstrahlen reflektiert und verstärkt werden (vgl. S. 141 [ZLR05]).

Folgende Abbildung zeigt das Funktionsschema des paraboloiden und des ellipsoiden Hohlspiegelmikrofons:

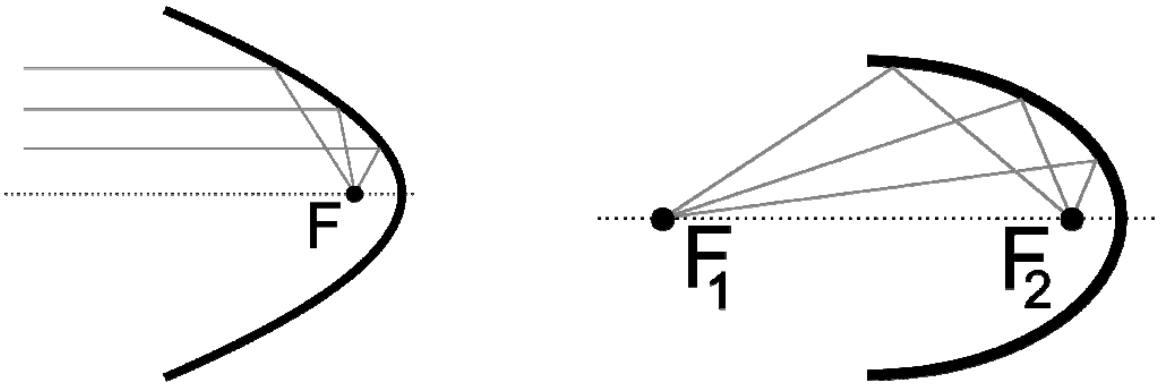

Abbildung 2.4: Funktionsschema eines paraboloiden und ellipsoiden Hohlspiegelmikrofons
Quelle: Hohlspiegelmikrofone. Workshopunterlagen „Mess- und Analysetechnik in der Fahrzeugakustik“, Martin Helfer, 2007

Auf den paraboloiden Hohlspiegel wird im weiteren Verlauf expliziter eingegangen. Zunächst erfolgt die Erklärung der Funktionsweise des ellipsoiden Hohlspiegels.
In der rechten Abbildung sind die zwei Brennpunkte F1 und F2 des ellipsoiden Hohlspiegels eingezeichnet. Zwischen diesen Brennpunkten werden, wie eingangs erwähnt, die Signale verstärkt. F1 entspricht hierbei einer Punktschallquelle.
In der Praxis werden auf Grund der höheren Verfügbarkeit und ihrem Einsatz in der

Nachrichtentechnik, häufig Parabolspiegel verwendet.
In der linken Abbildung sind die parallel eintreffenden Schallwellen zu sehen, die am Spiegel reflektiert und in Punkt F gebündelt werden. Der Punkt F entspricht der idealen Stelle, an welcher das Mikrofon positioniert werden müsste. Dieser kann bei Kenntnis über die zu Grunde liegende Parabelform mittels der folgenden Gleichung ermittelt werden:

$$b = mx^2 + x\tan(\Pi - 2\arctan 2mx - \arctan\frac{a - mx^2}{x})$$

Die genaue Mikrofonposition ist dabei abhängig von der Reflexionsposition im Spiegel (x) und kann im Grunde nicht eindeutig definiert werden. Allerdings erhöht sich die Anzahl der reflektierten Strahlen linear mit steigendem Abstand zum Spiegelmittelpunkt (x=0). Aus diesem Grund kann eine lineare Wichtung der Mikrofonabstände über den Abstand x und somit eine optimale Mikrofonposition ermittelt werden. Die daraus resultierende Kennlinie definiert die idealen Abstände zwischen Mikrofonposition und Scheitelpunkt des Parabolspiegels, wie auch zwischen der Schallquelle und der Spiegeloberfläche. (Vgl. S. 414 [Gen10]).

Folgende Abbildung demonstriert die Abhängigkeiten und die Kennlinie zwischen den genannten Parametern:

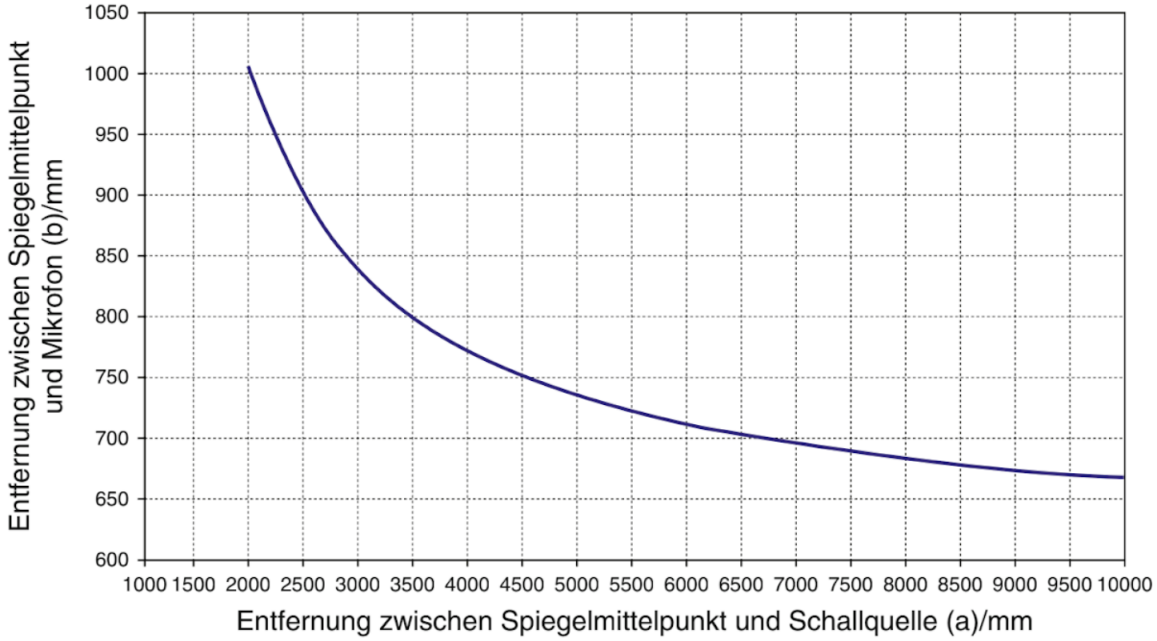

Abbildung 2.5: Zusammenhang zwischen Messabstand und Mikrofonposition
Quelle: Hohlspiegelmikrofone. Workshopunterlagen „Mess- und Analysetechnik in der Fahrzeugakustik“, Martin Helfer, 2007

Die Kennlinie hat hierbei einen sichtbaren logarithmischen Verlauf. Erkennbar ist, dass mit zunehmendem Abstand des Spiegelmittelpunkts zur Schallquelle, der Abstand von Spiegelmittelpunkt zum Mikrofon abnimmt.

2.1.2.2 Das Parabolmikrofon

Das Parabolmikrofon zählt durch die im vorherigen Kapitel beschriebene Schallwellenbündelung zu den gerichteten Mikrofonen. Es besitzt als einziges Mikrofon mit Hilfe des Hohlspiegels die Fähigkeit, akustische Signale ohne elektrische Hilfe zu verstärken. Folglich besitzt das konkrete Signal am Mikrofon einen höheren Pegel, als im ursprünglichen Freifeld. (Vgl. S. 1 [Bac02]). Diese Signalverstärkung erfolgt über ein breites Frequenzspektrum, allerdings nicht gleichmäßig, sondern stark frequenzabhängig. Welche Frequenzen bei der Aufnahme verstärkt oder abgeschwächt werden, wird in der späteren Analyse der akustischen Eigenschaften in Kapitel 3.1.1 genauer beleuchtet.

2.1.3 Das Parabolmikrofon der Firma Schoeps

Folgendes Kapitel beschreibt das für die Thesis zur Verfügung gestellte Parabolmikrofon der Firma Schoeps. Es wird dabei auf den Aufbau des Systems, die Klangeigenschaften des Mikrofons, wie auch auf die verfügbaren Windschutzarten für das System eingegangen.

Folgende Abbildung zeigt die Frontalansicht auf den Windkorb mit einer Aussparung für ein Kompaktmikrofon, einem Stativ mit integriertem XLR-Anschluss und einem Hohlspiegel der Firma Telinga:

Abbildung 2.6: Das Parabolmikrofon der Firma Schoeps in der Frontansicht
Quelle: internes Produktblatt der Firma Schoeps, abgerufen am 02.07.2018

2.1.3.1 Klangeigenschaften des verwendeten Mikrofons

Die Firma Schoeps verwendet für das Parabolmikrofon ein CCM 4L Kleinmembran-Kondensatormikrofon mit Nierencharakteristik. Das Mikrofon ist Teil der Kompaktserie CCM, bei der sich der Vorverstärker und die Mikrofonkapsel im selben Gehäuse befinden. Die Zusatzbezeichnung L steht für den verbauten Lemostecker[1], der im Parabolsystem genutzt wird. Folgende Abbildung zeigt das CCM 4:

Abbildung 2.7: Kompaktmikrofon CCM 4
Quelle: Schoeps.de, https://schoeps.de/produkte/ccm/ccm-mikrofone/nieren.html, abgerufen am 23.07.2018

Der Hersteller verspricht für das CCM 4 weitgehende Frequenzunabhängigkeit bezüglich der Richtwirkung und eine leichte Anhebung von 10 kHz-Frequenzen beim Einsatz im Diffusfeld. Folgende Abbildung des Frequenzgangs des Herstellers zeigt die Eigenschaften grafisch auf:

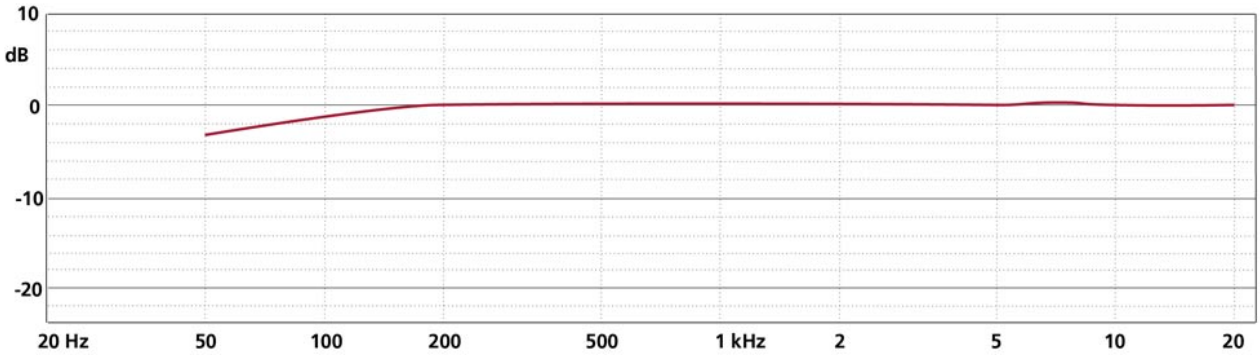

Abbildung 2.8: Frequenzgang des CCM 4
Quelle: Schoeps.de,
https://schoeps.de/produkte/ccm/ccm-mikrofone/nieren/ccm-4.html, abgerufen am 23.07.2018

[1]Ein in der Schweiz ansässiger Hersteller für elektronische und optische Steckverbindungen

Hierbei sind eine leichte Anhebung bei etwa 7,5 kHz und eine Abdämpfung der tiefen Frequenzen bei circa 180 Hz ersichtlich. Die Tiefenabnahme ist dabei ein gewollter Effekt, der den bei Druckgradientenempfänger entstehenden Nahbesprechungseffekt kompensiert (vgl. S. 119-120 [DDHW13]).

Weitere Daten sind in folgender Tabelle zusammengefasst und der Bedienungsanleitung der CCM-Reihe entnommen:

Übertragungsbereich	40 Hz - 26 kHz
Empfindlichkeit	14 mV/Pa
Ersatzgeräuchpegel	24 dB (CCIR) / 15 dB (A)
Geräuschpegelabstand A-bewertet	79 dB
Grenzschalldruck (0,5 % THD)	132 dB

Quelle: Akustische Daten der KompaktMikrofone (vgl. S. 19 [Sch16])

Der Übertragungsbereich erreicht mit 40 Hz nicht ganz die menschliche Hörschwelle von ca. 16 - 20 Hz. Allerdings kann die untere Grenzfrequenz bei gegebenem Nahbesprechungseffekt unter den 40 Hz liegen, siehe [Sch18a]. Dafür reicht die maximale Aufnahmefrequenz über die menschlich wahrnehmbare Hörfrequenz von 20 kHz hinaus. Dies ist im oben zitierten Bedienungsanleitung jedoch nicht vermerkt, da es sich um keinen typischen Anwendungsfall handelt. Durch die obere Grenzfrequenz von 26 kHz ist es möglich, auch Ultraschall von Tieren, wie z.B. Laute von Fledermäusen aufzunehmen. Diese können später durch Tonhöhenänderungen und Pegelanpassungen hörbar gemacht beziehungsweise angeglichen werden.

Die weiteren Daten weisen eine hohe Empfindlichkeit, geringes Eigenrauschen (Ersatzgeräuschpegel) und einen hohen Grenzschalldruck aus. Die beiden erst genannten Eigenschaften sind besonders bei leisen Klangquellen wichtig, welche aus größerer Entfernung aufgenommen werden. Durch das geringe Eigenrauschen hat das Mikrofon nur einen geringen oder nicht hörbaren Einfluss auf die eigentliche Aufnahme und erzeugt somit ein sehr gutes Signal-Rausch-Verhältnis[2]. Die Firma Neumann definiert Studiomikrofone mit Eigenrauschen von 11-15 dB (A) als Mikrofone mit sehr geringen Werten, den die besten Kleinmembranmikrofone erreichen. Die A-Bewertung ist hierbei eine Gewichtung, welche die menschliche Hörempfindlichkeit nachempfindet (vgl. [Neu18]).
Die Gewichtung nach CCIR beziehungsweise ITU-R-Norm erfolgt unter Verwendung eines vorgeschalteten Filters nach ITU-R BS. 468 und sollte Werte um die 25 dB ± 3-4 dB erreichen. Dieser Wert besitzt im Vergleich zur Hörempfindlichkeit gewichteten

[2]im englischen Signal-to-noise ratio (SNR)

Bewertung eine bessere Aussagekraft in der Praxis und sollte bevorzugt verwendet werden (vgl. S. 19 [Sch16] und [Sen14]).
Der Geräuschpegelabstand nach A-bewertet, besitzt einen hohen Bezugswert von 94 dB, der die Ergebnisse hoch ausfallen lässt und in einem gewissen Sinn „schönt“ (vgl. S. 146 [DDHW13]). Der Grenzschalldruck bezieht sich auf die Einhaltung von 0,5% Klirrfaktor[3] bei 1 kHz.
Auf die Auswirkungen der Klangeigenschaften des Mikrofons innerhalb des Parabolsystems wird im späteren Kapitel 3.2 eingegangen und auf Grund dessen nicht weiter erläutert.

2.1.3.2 Die Windschutze

Das Parabolsystem besitzt durch die spezielle CCM Mikrofonhalterung bereits einen standardmäßigen Windschutz. Dieser besitzt eine Länge von 200 mm beziehungsweise 270 mm im kompletten Montagesatz. Die Schaumzelle der Halterung ist ein Kunststoffmaterial aus Bulpren-Polyester. Der Windkorb wird von der englischen Firma Rycote bereitgestellt und kann mit einem Windjammer[4] überzogen werden. Das Gewicht wird mit dem kompletten Montagesatz mit 240 g angegeben.

Der zweite Windschutz ist ein sogenannter Hi-Wind Cover der Firma Telinga für den Hohlspiegel, der die Innenseite des Spiegels vor zu starken Windsignalen schützt. Die Windreduktion kann hierbei bis zu -35 dB erreichen. Er kann bei hohen Windaufkommen über den Spiegel gezogen werden und verhindert somit Windturbulenzen innerhalb des Spiegels. Darüber hinaus kann er den Spiegel vor Störfaktoren, wie Regen und Insekten schützen, welche die Aufnahme beeinträchtigen können (vgl. [Tel]) Folgende Abbildung zeigt den angebrachten Hi-Wind Cover auf dem Hohlspiegel:

2.1.3.3 Der Hohlspiegel

Der Hohlspiegel V2 der Firma Telinga Microphones steht für das Parabolmikrofon in einer faltbaren und einer stationären Version zur Verfügung. In beiden Fällen besteht das Material aus durchsichtigem Kunststoff (Polycarbonat) mit einem abschließenden grauen Rand, wie in Abbildung 2.6 ersichtlich ist. Der Durchmesser beträgt inklusive des Rahmens 585 mm. Der Hohlspiegel erhält durch die Größe und die Krümmung des Spiegels eine Tiefe von 160 mm. Unterschiede der zwei Spiegel liegen in der Dicke des Materials und die daraus resultierende Faltbarkeit. Die stationäre Version besitzt eine Kunststoffdicke von 2 mm, die faltbare Version eine Dicke von 1 mm. Dadurch erhöht sich das Gewicht auf 580 g, im Gegensatz zur faltbaren Variante mit 380 g.

[3] THD = Total Harmonic Distortion
[4] englisch für Windschutz.

Abbildung 2.9: Hi-Wind Cover der Firma Telinga für den Hohlspiegel
Quelle: Produktseite Hi-Wind Cover [Tel], abgerufen am 15.10.2018

2.1.3.4 Das Hohlspiegelstativ

Das Hohlspiegelstativ besteht aus dem Kunststoff Polyamid und besitzt eine Länge von 210 mm. Als Steckerverbindung beziehungsweise -erweiterung ist am Ende des Griffs ein männlicher XLR-Stecker samt Erweiterung verbaut. Dieser Stecker der Firma Lemo kann die benötigte Phantomspeisung von 48 V für das CCM 4 liefern.

2.2 Akustische Einflüsse auf die Parabolmikrofonaufnahme

Das folgende Kapitel beschreibt ungewollte akustische Einflüsse, die bei einer Parabolmikrofonaufnahme auftreten können. Es wird diskutiert, wie sie vermieden beziehungsweise ausgeglichen werden.

2.2.1 Dissipation

Bei den Messungen und Testungen des Parabolsystem der Firma Schoeps wurde die Aufnahmeentfernung zur Schallquelle auf eine Distanz von 12 m - 30 m definiert. Hierbei treten frequenzabhängig unterschiedlich stark ausgeprägte Pegelverluste in Bezug auf das Aufnahmesignal auf, welche von der Luftfeuchtigkeit und der Temperatur beeinflusst werden. Die Dissipation beschreibt in diesem Fall den Verlust von Schall- zu Wärmeenergie bei der Übertragung über ein Medium, in diesem Fall die reine Luftübertragung im Freifeld. Prinzipiell führt eine geringere Luftfeuchtigkeit und eine niedrigere

Temperatur zu einem höheren Pegelverlust. Der reine Pegelverlust über die Entfernung ist allerdings nicht frequenzabhängig und kann mit dem Abstandsgesetz 1/r berechnet werden. Dieser sollte aus diesem Grund unabhängig von der Dissipation berechnet werden.
Folgende Abbildung zeigt die Unterschiede der Dissipation in Relation zur Frequenz, im Abstand von 30 m:

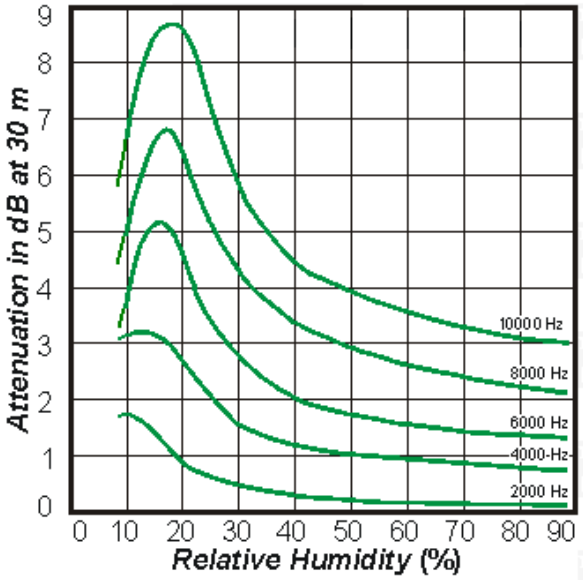

Abbildung 2.10: Frequenzabhängige Luftdämpfung in 30 m Abstand bei unterschiedlicher Luftfeuchtigkeit

Quelle: www.sengpiel.com/Rechner-luft.htm, abgerufen am 15.10.2018

Ersichtlich ist, dass die Dissipation stark frequenzabhängig stattfindet und vor allem Pegelverluste bei hohen Frequenzen erzeugt. Für Frequenzen bei 10 kHz bedeutet dies annähernd eine Abschwächung um 9 dB bei 20% Luftfeuchtigkeit beziehungsweise 1,5-fache Signaldämpfung, während für tiefe Frequenzen bei 2 kHz bei gleicher Luftfeuchtigkeit ein wesentlich geringerer Pegelverlust von 1 dB festzustellen ist.

2.2.2 Windgeräusche

Da das Einsatzgebiet des Parabolmikrofons überwiegend in freier Natur liegt, sind auch Windturbulenzen im Hohlspiegel und am Mikrofon nicht zu vermeiden und müssen berücksichtigt werden. Für gerichtete Druckgradientenempfänger sind die Auswirkungen von Luftverwirbelungen folgenreicher als für ungerichtete Druckempfänger, wie das Kugelmikrofon. Grund dafür ist die erhöhte Empfindlichkeit von Druckgradientenempfängern gegenüber tiefen Frequenzen, welche als Ausgleich zwischen der geringen Druckdifferenz im Bassbereich und dem Wandlerprinzip der Membran fungiert. Auf Grund dessen kann die Verwendung einer Kugel für Aufnahmen mit hohem Windaufkommen sinnvoll sein (vgl. [Wut86])

Für geringes Windaufkommen bieten manche Mikrofonhersteller integrierte, schaltbare Tiefenabsenkungen, welche die angesprochenen Windgeräusche in Teilen herausfiltern. Eine zusätzliche Option ist die Verwendung von Windkörben für das eingesetzte Mikrofon. Allerdings haben diese ebenfalls einen hörbaren Einfluss auf den Frequenzgang. Für Druckgradientenempfänger sind Windschutze aus Schaumstoff hierbei am wirksamsten. Dadurch bedingte Höhendämpfungen können mit Diffusfeldkapseln kompensiert werden. Für Druckgradientenempfänger ist ein Windkorb empfehlenswert, der die Kapselöffnung komplett umschließt (vgl. [Wut86]).

2.3 Digitale Postproduktion

Die digitale Nachbearbeitung (Postproduktion) nimmt heutzutage in modernen Audioproduktionen einen Hauptteil der Arbeit für den Audioschaffenden ein. Durch das zur Verfügung stehen von leistungsfähigen, digitalen Audioworkstations (DAW) kann ein Großteil der früher analogen Misch- beziehungsweise Produktionskette vereinfacht und unter Einschränkungen obsolet gemacht werden. Moderne DAWs übernehmen dabei Aufgaben wie Routing der einzelnen Spuren, Mischverhältnisse der Kanäle, Klangbearbeitung- und gestaltung mit Effekten oder das darauffolgende Mastering. Nach der Analog-Digital-Wandlung können alle Aspekte der Tonaufnahme digital nachbearbeitet werden. Dies macht sich überwiegend vorteilhaft bei niedrigeren Produktionskosten und höherer Flexibilität bemerkbar.

Populäre DAWs wie Pro Tools, Cubase, Logic oder Ableton Live setzen bei der Postproduktion auf verschiedene Betriebssysteme beziehungsweise zugrundeliegende Technologien und Pluginformate, um die angesprochenen Funktionalitäten zur Verfügung zu stellen. Für Pluginentwickler stellt sich hierbei vor der Entwicklung die Frage, auf welcher Plattform das Zielklientel die Postproduktion beziehungsweise das Plugin einsetzen wird. Fehlkalkulationen können hierbei zum finanziellen Misserfolg führen. Verschiedene Firmen bieten aus diesem Grund Frameworks[5] an, um der Gefahr entgegen zu wirken, nicht für die richtige Plattform zu entwickeln. Diese Frameworks übernehmen die Portierung auf die einzelnen Plattformen beziehungsweise Betriebssysteme. Somit können sich die Entwickler um die eigentliche Funktionalität, unabhängig vom Pluginformat, kümmern.

In diesem Zusammenhang beschäftigt sich das folgende Kapitel zu Beginn mit den Vorteilen der digitalen Technik, im Bezug auf die digitale Postproduktion, speziell Filter und beschreibt verschiedene, marktübliche Pluginformate. Darauf folgt die Vor-

[5]Englisch für Rahmenstruktur. Entspricht einem Programmiergerüst.

stellung eines populäres Frameworks zur Portierung eines Audioplugins in diverse Pluginformate.

2.3.1 Vorteile digitaler Filter

Digitale Filter bringen gegenüber analogen Filtern entscheidende Vorteile im Bereich Zuverlässigkeit, Wiederverwendbarkeit und Genauigkeit mit sich, welche im folgenden Verlauf diskutiert werden. Eine Abwägung zwischen Digital- und Analogfilter bedeutet hierbei auch eine Entscheidung zwischen quantisiertem, prozessiertem Signal auf Softwareebene oder einer Signalverarbeitung auf Hardwareebene mit Operationsverstärker und diskreten Bauelementen, wie Kondensatoren und Widerständen.

In analogen Filtern sind bautechnisch bedingte Schwankungen durch Toleranzabweichungen vorgegeben, die nicht eliminiert werden können. Darüber hinaus altern die Bauteile und unterliegen hiermit einer gewissen Unzuverlässigkeit, welche auch von äußeren Faktoren, wie beispielsweise Umgebungstemperaturen verstärkt werden können. Digitale Filter besitzen diese Abhängigkeit nicht und arbeiten immer nahezu gleich. Eine erneute Kalibrierung ist deshalb nicht notwendig. Daraus resultiert ein weiterer Vorteil der Reproduzierbarkeit von digitalen Filtern. Die Zustände oder Voreinstellungen können einfach wiederhergestellt oder das Übertragungsverhalten adaptiv angepasst werden. Dies kann analog nur mit erhöhten Aufwand oder gar nicht bewerkstelligt werden, da es sich um starre Bauelemente handelt.

Durch die Quantisierung der Signale kann digital ein hoher Störabstand erreicht werden, der allerdings ein Quantisierungsrauschen, d.h. Rundungsfehler bei der Signalabtastung mit sich bringt, welches durch die begrenzte Bittiefe verursacht wird. Analoge Systeme weisen dafür ein Grundrauschen auf und sind durch Verzerrungen begrenzt. Manche Autoren vermerken, dass es keine guten Analogfilter gibt, da sie immer einen Kompromiss aus Flankensteilheit, Welligkeit im Durchlassbereich und (nicht-)linearen Phasenverlauf im Durchlassbereich darstellen (vgl. S. 324 [Kar13]). Im Gegensatz dazu können digitale Filter so implementiert werden, dass sie eine lineare Phase besitzen. Dies ist allerdings nicht für alle Anwendungsfälle möglich, da digitale Filter höherer Ordnung ebenfalls eine nichtlineare Phase besitzen können. Das nächste Kapitel legt den Fokus auf Unterschiede in der Implementierung von digitalen Filtern und die dazu gehörigen Vor- und Nachteile.

2.3.2 IIR-/FIR-Filter

Digitale Filter unterscheiden sich ebenso wie analoge Filter durch ihre Übertragungsfunktion oder zeitlich gesehen durch ihre Impulsantwort. Betrachtet man einen digitalen

Filter mit einer zeitlich begrenzten Impulsantwort, spricht man von einem nicht rekursiven, im Englischen „finite impulse response“ Filter (FIR). Mit FIR-Filter lassen sich Filter mit linearen Phasengang realisieren, da sie ein endliches Ausschwingverhalten besitzen. Dies hat zudem den Vorteil, dass sie nicht instabil werden oder eine selbständige Schwingung auslösen können (vgl. S. 627 [DDHW13])
Die Latenz eines FIR-Filter kann dabei je nach Filterfunktion zwischen 1 - 50 ms liegen und benötigt eine hohe Rechenleistung (vgl. S. 461 [Wei09]).
Im Unterschied dazu besitzen die „infinite impulse response“ oder rekursiven Filter (IIR) eine theoretisch unendliche und damit auch die Gefahr, einer um den Signal-Nullpunkt oszillierenden Impulsantwort. Finden wenige beziehungsweise zu vernachlässigbare Rechen- oder Rundungsfehler statt, lassen sich IIR-Filter dafür sehr effizient umsetzen und besitzen im Vergleich eine geringere Zeitverzögerung als FIR-Filter (vgl. S. 628 [DDHW13]).

2.3.3 Pluginformate

Die folgenden drei Kapitel beschäftigen sich mit drei populären Pluginformaten, die auf unterschiedlichen technischen Ansätzen und Betriebssystemen basieren.

2.3.3.1 Virtual Studio Technology

Der Markenname Virtual Studio Technology (VST) ist ein Softwareprotokoll, welches ursprünglich von der Firma Steinberg 1996 für ihren MIDI[6]-Sequenzer und spätere DAW Cubase entwickelt wurde. Mittlerweile zählt das VST-Format zu den führenden Pluginformaten und ist nahezu ein Industriestandard. Die Entwicklung ist mittlerweile bei der Version 3.6.11 (Stand 29.11.2018) angekommen, welche unter dem Namen VST3 bekannt ist und als Software Development Kit (SDK) auf der Steinberg-Webseite zur freien Entwicklung[7] heruntergeladen werden kann. Bei der Benutzung ist es allerdings notwendig, eine Referenz auf Steinberg mit Hilfe des offiziellen VST-Logos in allen Packages, Webseiten und Dokumenten einzusetzen.

Ziel von VST ist die Integration von virtuellen Effektprozessoren und Instrumenten in die DAW beziehungsweise Nachbildung von Hardware-Effektgeräten, Musikinstrumenten, wie auch neuartigen Soundeffekten. Dazu werden die VST-Plugins in den so genannten VST-Mixer eingebunden und können über Bedienelemente im Plugin oder externen Hardware-Controller bedient werden (vgl. [Steb]).
In der praktischen Anwendung verarbeitet das VST-Plugins einen Audiostream, wel-

[6]MIDI = Musical Instrument Digital Interface. Industriestandard für die digitale Datenübertragung von MIDI-fähigen Musikinstrumenten

[7]Entwicklerseite von Steinberg: https://www.steinberg.net/de/company/developer.html

cher von der Hostanwendung zur Verfügung gestellt wird. Der Host bricht diesen Audiostream in eine Serie von Blöcken mit definierter Größe herunter und gibt sie sequenziell an das Plugin weiter. Ein einzelner Block wird im Plugin dann mit Hilfe des Rechnerprozessors mit dem gewünschten Effekt bearbeitet und an den Host zurückgegeben. Die genaue Verarbeitung wird dabei dem Plugin überlassen. Dadurch hat der Host keine Informationen über die genaue Verarbeitung. In diesem Fall wird das Plugins als Blackbox bezeichnet, was die Verwendung von VST-Plugins in verschiedenen DAWs vereinfacht.

Während der Quellcode plattformunabhängig ist, ist das Auslieferungsformat abhängig von der Plattformarchitektur, auf der es später benutzt wird. Dies bedeutet, dass auf Windows-Betriebssystemen ein VST-Plugin einem multi-threading[8] fähigen „Dynamic Link Library (DLL)“[9], entspricht. Auf macOS-Rechnern entspricht das VST-Plugin einem Mach-O Bundle[10]. Auf Linuxsystemen wird das VST-Plugin als Package benutzt. (Vgl. README.md [Stea]).

Ein Merkmal für die Beliebtheit des VST-Formats ist auch die große Anzahl an frei verfügbaren Audioplugins, welche unter anderem auf vst4free.com oder auf delamar.de[11] angeboten werden.

2.3.3.2 Avid Audio eXtension/Real-Time AudioSuite

Das Avid Audio eXtension ist ein proprietäres Plugin-Format der amerikanischen Firma Avid Technology, ehemals Digidesign. Es entspricht dem 64-Bit fähigen Nachfolgeformat von Real-Time AudioSuite desselben Herstellers und wurde 2013 in der 11. Version der DAW Pro Tools als Standard eingeführt. Das AAX-Format wird in zwei Versionen vertrieben. Das AAX DSP-Format ist ausschließlich mit „Pro Tools | HDX“ kompatibel, während AAX Native mit allen Pro Tools-Versionen ab Version 10 kompatibel ist. Das Pro Tools | HDX-Paket enthält dabei neben der Pro Tools-Software eine HDX-Karte mit integriertem DSP[12], der es ermöglicht, bis zu 256 Audiospuren aufzunehmen. Dieser kann auch die Berechnung von Plugins übernehmen. Dagegen ist das AAX Native-Format, ein hostbasiertes Plugin ohne externen DSP. Das AAX-Format besitzt neben der Echtzeitfähigkeit auch die Möglichkeit, innerhalb der zwei vorgestellten Systeme Sessions auszutauschen und weiterhin die verwendeten Plugins benutzen

[8] Die Verwendung von mehreren Threads (Prozessen) kann auf Mehrkernprozessoren ausgelagert und genutzt werden.

[9] Englisch für dynamische Programmbibliothek

[10] Mach-O = Mach object file format. Dateiformat für unter anderem ausführbare Dateien und Objektcode.

[11] https://www.delamar.de/musikproduktion/free-vst-plugins-800/

[12] Digitaler Signalprozessor zur Bearbeitung von Audiosignalen.

zu können. Diese müssen dazu auf dem neuen System nicht installierst sein. (Vgl. S. 2 [Tec16]).
Das Real-Time AudioSuite-Format ist der 32-Bit Vorgänger des AAX-Formats. Es ist nur bis zur 10. Pro Tools-Version kompatibel und wird von Avid nicht weiterentwickelt.

2.3.3.3 Audio Unit

Audio Unit ist ein proprietäres Format der amerikanischen Firma Apple. Audio Units werden primär für die Audioprogramme Logic Pro und GarageBand von Apple entwickelt, welche nur unter dem Betriebssystem macOS verfügbar sind. Das Audio Unit-Format ist Teil der Core Audio Technologie, welche Bestandteil des macOS ist. Durch die direkte Einbindung in das Betriebssystem können die Laufzeiten vergleichsweise gering gehalten werden und die Plugins beziehungsweise die Schnittstellen der Plugins direkt vom System unterstützt werden. Dadurch ist die Verwendung von Audio Units auf anderen Betriebssystemen nicht möglich, wodurch die Einsatzmöglichkeit begrenzt wird. Die Verwendung von VST-Plugins ist unter Logic Pro ab Version 6 nicht mehr möglich. Allerdings können VST-Plugins mittels des Plugins "VST to AU Adapter"der Firma fxpansion[13] portiert werden.

Audio Units übernehmen die Bearbeitung von Signalen in Form von Effekten oder die Portierung von MIDI-Daten in konkrete Audiosignale. Das Audio Unit-Framework ist, wie die VST in der 3. Version erhältlich (Stand 08.08.2018, vgl. [App]).

2.3.4 Das Juce-Framework als Implementierungsgrundlage

JUCE ist ein in Teilen Open-Source basiertes plattformunabhängiges Anwendungsframework, das für die Entwicklung von Desktop- und Mobilanwendungen verwendet wird. Es ist in der Hochsprache C++ geschrieben und wird unter der GNU General Public License vertrieben, welche es ermöglicht, die Software mit zu entwickeln und zu verändern. Der Erfinder Julian Storer, auch Jules genannt, ist Hauptentwickler und Namensgeber der Plattform: **J**ules' **U**tility **C**lass **E**xtensions. Die später beschriebenen Implementierungen basieren auf seiner Grundidee zur Pluginentwicklung beziehungsweise seinen Programmierrichtlinien.

Das JUCE-Framework bietet hierbei die in 2.3 besprochene Funktionalität der Exportierung des Projekts für folgende Zielplattformen: Xcode (macOS und iOS), Visual Studio 2013-2017 (Windows), Linux Makefile, Android Studio, Code::Blocks (Windows und Linux) und CLion (Beta) (Stand 08.08.2018). Auf den Zielplattformen können die Plugins in den folgenden Formaten erstellt werden: VST, VST3, AAX, RTAS, AU, AU

[13] www.fxpansion.com/products/vst-au-adapter/

v3 oder als eigenständiges Programm (Standalone). Die beiden letztgenannten dabei nur auf macOS, wie in 2.3.3.3 bereits angesprochen wurde. Das JUCE-Framework nutzt in der Version 5.3.2 zur Implementierung der Plugins die Programmiersprache C++, standardmäßig in der 14. Version (Stand 08.08.2018).

2.3.4.1 Projucer

Der Projucer ist ein in JUCE integriertes Projektmanagement-Tool zur Erzeugung und Verwaltung von JUCE-Projekten. Er übernimmt Teile der plattformunabhängigen Exportierung eines Projekts, indem er die Erzeugung von Drittanbieter-Dateien übernimmt. Im Rahmen der Thesis wurde ein neues Audioplugin-Projekt erstellt. Die dafür benötigten Drittanbieter-Dateien entsprechen hierbei den Projektdateien der verwendeten IDEs[14]. Der zur Exportierung des Projekts in die 2.3.4 angesprochenen Pluginformate benötigte Quellcode wird ebenfalls automatisch erzeugt. Folgende Abbildung zeigt die Auswahlmöglichkeiten zur Erzeugung eines neuen Projekts mit Hilfe vorgefertigter Templates:

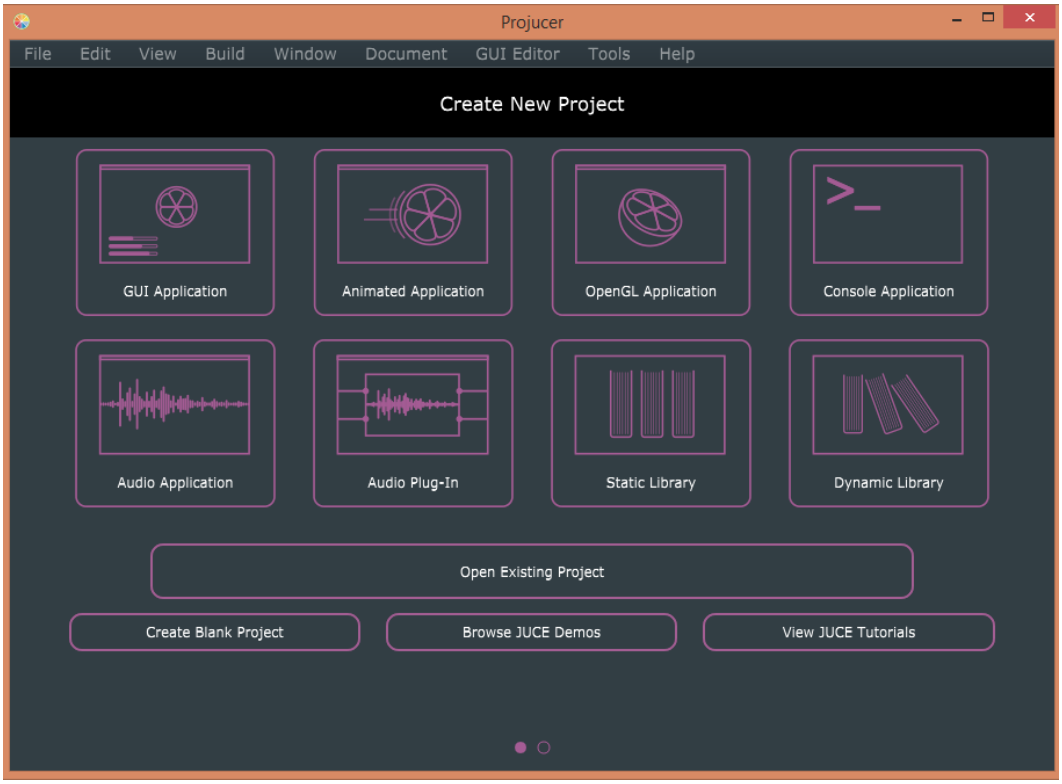

Abbildung 2.11: Übersichtsanzeige in Projucer für ein neues Projekt

Nach der Erzeugung eines Projekts aus den oben ersichtlichen Auswahloptionen ermöglicht der Projucer die Verwaltung der Projekteigenschaften und Einbindung von externen Assets, wie Bilder, Übersetzungsdateien und (externen) Bibliotheken.

[14]IDE = Integrierte Entwicklungsumgebung.

Über den Projucer werden ebenfalls die verschiedenen SDKs der Pluginformate mittels eines globalen Suchpfads für die Zielplattformen manuell eingebunden:

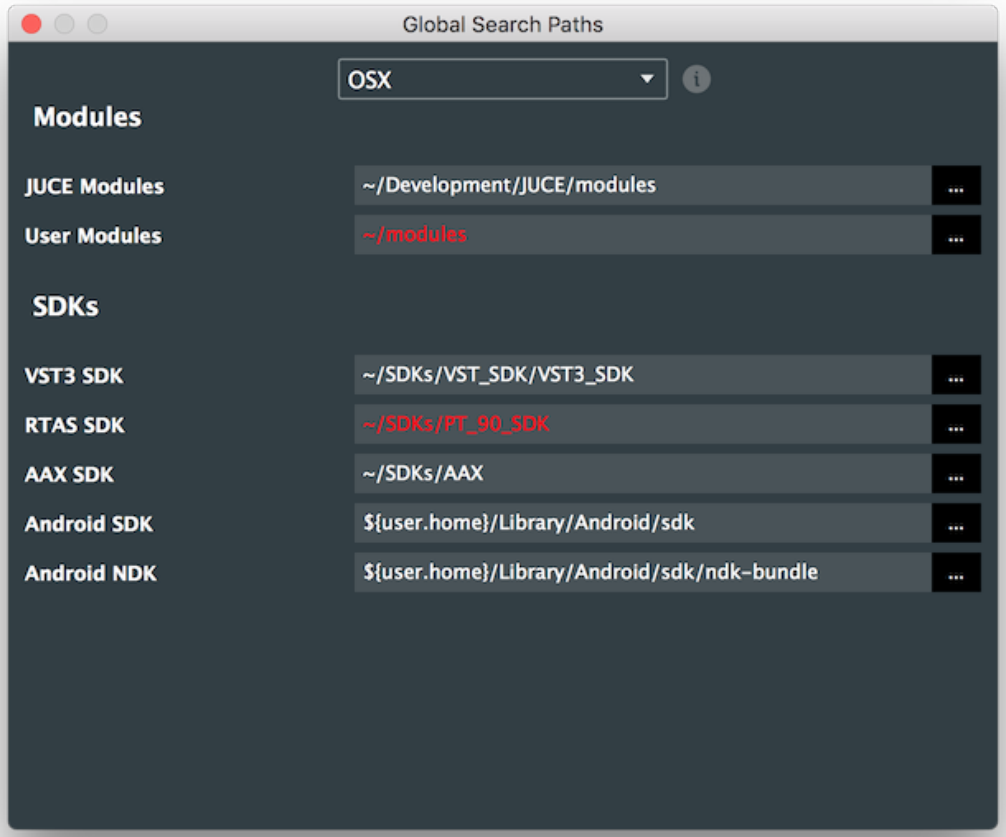

Abbildung 2.12: Globale Pfadsuche der SDKs für macOS

Der Anwender muss hierbei das Bereitstellen der SDKs selbst übernehmen und von der Internetseite des Herstellers beziehen.

2.3.4.2 Struktur und Funktionsweise

Ein JUCE-Projekt besitzt auf der obersten Projektebene eine .jucer-Datei, die alle Projekteigenschaften enthält und mittels des Projucers bearbeitet werden kann. Neben der Projektdatei werden bei der Projektinitialisierung drei Unterordner angelegt. Der Sourceordner enthält den Quelltext der Applikation. Der Build-Ordner beinhaltet Ordner für die ausgewählten Zielplattformen beziehungsweise die Projektdateien der einzelnen IDEs. Der Ordner JuceLibraryCode enthält automatisiert generierte C++-Headerdateien[15], die einzelnen JUCE-Modulen entsprechen und zur Implementierung benötigt werden.

Das Aussehen beziehungsweise die GUI-Elemente des Plugins werden in JUCE mit Hilfe einer zentralen LookAndFeel-Klasse realisiert, die ein einheitliches Aussehen aller Komponenten ermöglicht. Sie bietet auch die Möglichkeit, einzelne Skins (einheitliche Oberflächengestaltung) für die Anwendung anzulegen. Dies wird im späteren Kapitel 4.3.1.2 genauer erläutert.

2.3.4.3 Das Schoeps-Framework

Für das Audioplugin wurde von der Firma Schoeps ein bereits vorhandenes Framework verwendet, welches in Kapitel 4 vorgestellt wird. Es übernimmt die Gestaltung Mit Hilfe einer angepassten Schoeps-LookAndFeel-Klasse und stellt alle benötigten SDKs in einem zentralen Git[16]-Repository[17] zur Verfügung. Ausgangspunkt der Oberflächengestaltung der Anwendung ist aus diesem Grund das Aussehen und Design der bereits vorhandene Plugins der Firma Schoeps.

[15] Textdatei oder Klasse, die Definitionen und Deklarationen enthält beziehungsweise auslagert.
[16] Freie Software zur Verwaltung von Softwareprojekten.
[17] Repository = verwaltbares Verzeichnis zur Archivierung und Speicherung von Quellcode.

3. Analyse des Parabolsystems

Das folgendes Kapitel befasst sich mit der Anforderungsanalyse des Parabolsystems der Firma Schoeps und den daraus resultierende Anforderungen für die Implementierungen in Kapitel 4. Zuerst werden die technischen Messdaten des Mikrofonsystems analysiert, die Stärken und Schwächen des Systems besprochen und die daraus resultierenden Anforderungen an das Plugin vorgestellt. Darüber hinaus werden die Frequenzanpassungen am Eingangssignal besprochen und die Vorteile von Presets für die Postproduktion präsentiert.

3.1 Technische Messdaten des Parabolmikrofons

Wie bereits in Kapitel 2.1.3.1 erwähnt, befassen sich die folgenden Unterpunkte mit den Klangeigenschaften beziehungsweise Stärken und Schwächen des Parabolsystems. Zu Beginn werden die Frequenzganganalyse und das Polardiagramm genauer untersucht, die von der Firma Schoeps im Rahmen der Thesis zur Verfügung gestellt wurden.

3.1.1 Frequenzganganalyse

Für die Analyse wurden mehrere Mikrofone der Firma Schoeps im Abstand von 12 m und 30 m analysiert, um das geeignetste Mikrofon für das Parabolsystem zu ermitteln. Die drei dazu verwendeten Mikrofone sind die Kompaktmikrofone CCM 2 (Kugel), CCM 21 (Breite Niere), wie auch das CCM 4 (Niere). Zur Feststellung der Auswirkung der Mikrofonposition im Windkorb wurde zudem die Position in selbigem variiert und mit in die Analyse einbezogen.

Bei den Messungen wurde ersichtlich, dass das Kugelmikrofon oberhalb von 4 kHz, im Gegensatz zu den Nierenmikrofonen, einen deutlichen Pegelabfall besitzt. Zusätzlich traten beim Kugelmikrofon mehr Kammfiltereffekte auf, als bei den anderen beiden Kompaktmikrofonen. Grund dafür ist, dass das Nierenmikrofon rückwärtsgewandt zur Zielquelle im Spiegel positioniert wird.

Durch die Nierencharakteristik wird der Direktschall mit einer Abdämpfung von 20 - 30 dB nahezu vollständig ausgeblendet und ausschließlich Reflexionen des Spiegels aufgenommen (vgl. [Sch18b]). Dies führt nachweislich zu weniger Kammfiltereffekten. (Vgl. S. 5 [Bac02])

Im Vergleich der beiden Nierenmikrofone CCM 21 und CCM 4 zeigte sich, dass das CCM 21 bei den Messungen wesentlich instabiler, im Vergleich zum CCM 4, bezüglich der Mikrofonposition im Windkorb ist. Hierbei kann es sich allerdings auch um Messungenauigkeiten handeln.
Die späteren Implementierungen beziehen sich aus diesem Grund auf die Verwendung des CCM 4, da für die beiden anderen genannten Mikrofonen noch weitere Messungen vorgenommen werden, um mögliche Messfehler auszuschließen. Diese waren zum Zeitpunkt der Thesis noch nicht abgeschlossen waren.

Die zur Verfügung gestellte Frequenzganganalyse des Parabolsystems mit dem CCM 4 vergleicht die Amplituden des Freifeld und des Diffusfelds von 100 Hz bis 20 kHz. Sichtbar ist hierbei, dass beide im Tieffrequenzbereich nur eine Pegeldifferenz von 2 dB aufweisen. Die Amplitude des Diffusfelds verliert mit ansteigender Frequenz an Pegel, während der Pegel des Direktfelds nahezu linear ansteigt. Bei 8 kHz besitzen die Felder einen maximalen Pegelunterschied von 36 dB. Nach diesem Maximum besitzt die Kurve des Freifelds einen Pegelabfall, übereinstimmend zur Diffusfeldkurve.
Folgende Abbildung zeigt die Frequenzantwort:

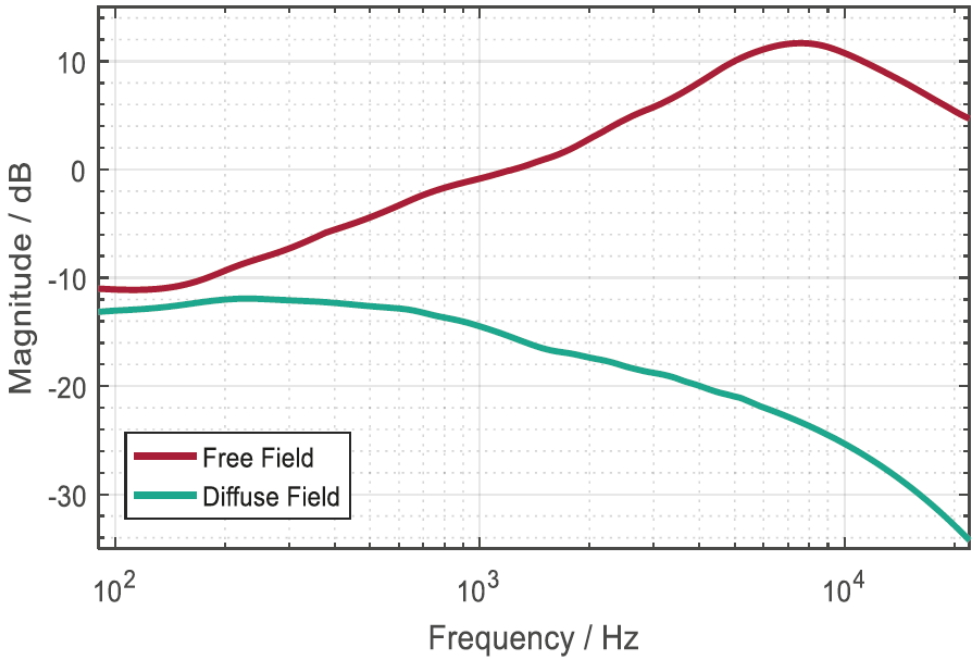

Abbildung 3.1: Frequenzantwort des Parabolasystems mit dem CCM 4
Quelle: Schoeps GmbH

Die Frequenzantwort zeigt die akustischen Zusammenhänge zwischen Frei- und Diffusfeld und mögliche Probleme bei der Postproduktion von aufgenommenen Signalen. Durch die Verstärkung der Mittel- und Höhenanteile im Freifeld und gleichzeitigen Abschwächung von Diffusfeldanteilen, können die beschriebenen Signalanteile einfach verstärkt oder abgeschwächt werden, ohne dass Hintergrundgeräusche, Reflexionen oder mögliche Kammfiltereffekte des Diffusfelds zu präsent in den Vordergrund treten. Im Tiefton- beziehungsweise dem unteren Mittentiefenbereich von 100 Hz bis ca. 500 Hz dagegen, sind die Pegelunterschiede so gering, dass eine Verstärkung auch tendenziell unerwünschte Signale des Diffusfelds verstärken kann. Hierzu zählen neben Griffgeräuschen am Stativ, auch möglicherweise ungewollte Umweltgeräusche wie beispielsweise Blätterrauschen oder Fahrzeuggeräusche. Für die folgende Implementierung muss aus diesem Grund der Mitten-Höhenbereich anders als die unteren Mitten-/Tieftonbereich gefiltert werden.

Des Weiteren ist ersichtlich, dass das System Frequenzabschwächungen/- verstärkungen von -10 dB für tiefe Frequenzen unter 200 Hz und mehr als 10 dB für Frequenzen zwischen 5 kHz - 10 kHz besitzt.

3.1.2 Polardiagramm

Das zur Verfügung gestellte Polardiagramm zeigt ebenfalls die angesprochenen akustischen Eigenschaften. Für Frequenzen unterhalb von 500 Hz verhält sich die Richtcharakteristik des Systems annähernd wie eine Kugel/Breite Niere. Dadurch werden Signalanteile des Diffusfelds in eben diesem Frequenzbereich vermehrt mitaufgenommen. Bei 1000 Hz verändert es sich zu einer Keulencharakteristik, um danach stark richtend für Signale oberhalb von 2 kHz zu wirken. Die Veränderung der Richtwirkung ist in folgender Abbildung ersichtlich:

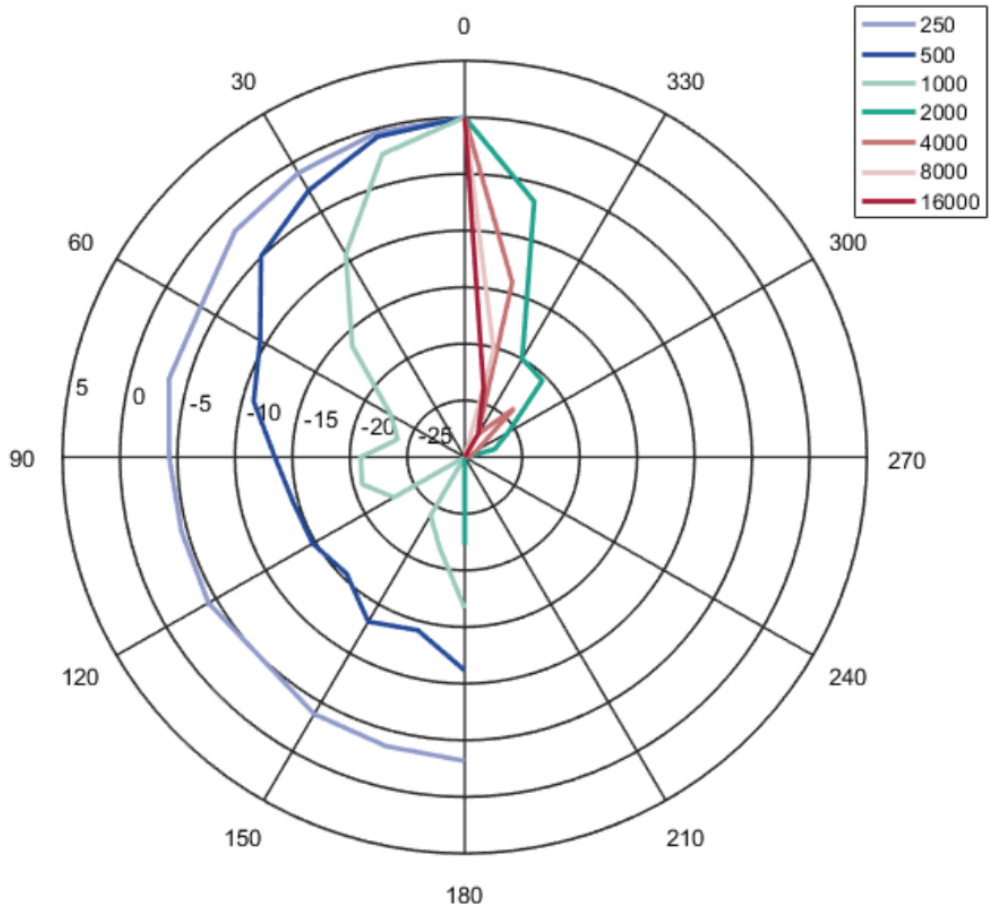

Abbildung 3.2: Polardiagramm des Parabolsystems mit CCM 4
Quelle: Schoeps GmbH

Die beschriebene frequenzabhängige Richtwirkung ist bei Studiomikrofonen mit Nierencharakteristik ebenfalls gegeben (vgl. S. 162 [Gö07]). Diese fällt im Vergleich zum Parabolmikrofon aber wesentlich geringer aus.

3.1.3 Auswirkung der Windschutze auf die Frequenzantwort

Wie bereits in den Grundlagen 2.1.3.2 angesprochen, stehen für Aufnahmen zwei optionale Windschutze zur Verfügung. Diese besitzen sehr unterschiedliche Auswirkungen auf die akustischen Eigenschaften des Systems. Für die Implementierung der Filter wurden auf Grund dessen ebenfalls Messungen zur Auswirkung auf die Frequenzantwort erstellt. Im weiteren Verlauf werden Auffälligkeiten in den Frequenzgangauswirkungen besprochen und mögliche Ausgleichsmöglichkeiten diskutiert.
Folgende Abbildung zeigt die Auswirkungen auf den Frequenzgang des Parabolsystems:

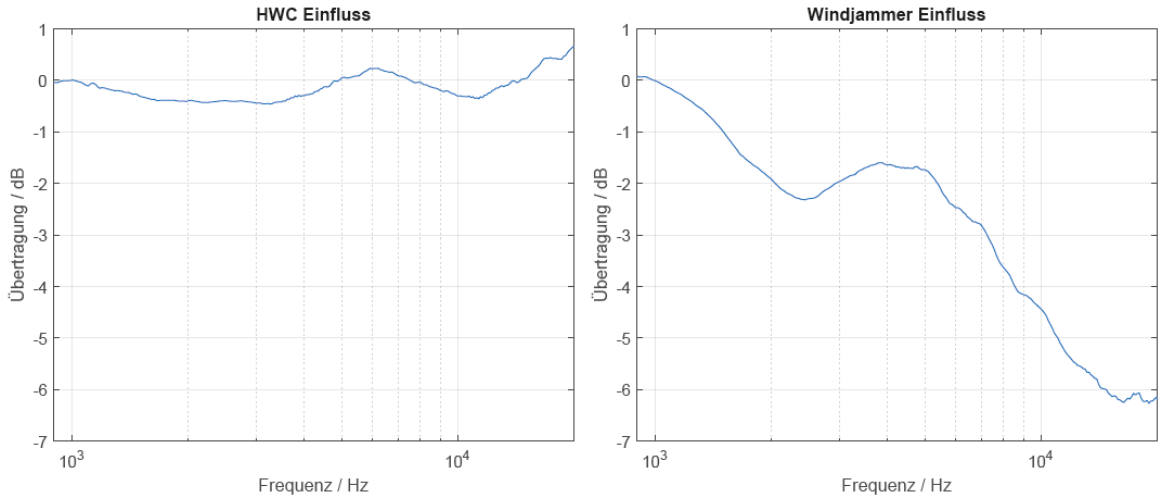

Abbildung 3.3: Einfluss der verwendeten Windschutze auf die Frequenzantwort
Quelle: Schoeps GmbH

Der in 2.1.3.2 angesprochene High Wind Cover besitzt nur eine geringe Auswirkung auf eintreffende Signale. Der Einfluss des Windschutzs für den Hohlspiegel kann hierbei mit einzelnen moderat eingestellten Filtern in den betreffenden Frequenzbereichen neutralisiert werden. Für den Frequenzbereich zwischen 1 kHz und 5 kHz kann ein Glockenfilter mit einer geringen Güte von etwa 0,8 und einer Maximalverstärkung von 0,5 dB verwendet werden. Für den nachfolgenden Bereich bis circa 12 kHz kann ebenfalls ein moderates Filter den Pegelverlust kompensieren.

Der Windjammer hingegen benötigt größere frequenztechnische Anpassungen, die vor allem im Höhenbereich, beispielsweise mit einem Kuhschwanzfilter[1] angepasst werden müssen. Für den Frequenzabfall bei 2,5 kHz kann ein Glockenfilter mit einer ungefähren Güte von 1,5 und 2,25 dB für die Verstärkung veranschlagt werden.
Je nach Anwendungsfall sollte im Plugin zwischen den unterschiedlichen Windschutzarten hin- und her geschaltet werden können und eine entsprechende automatische

[1] im Englischen = high shelf filter

Frequenzanpassung stattfinden.

3.2 Technische Vor- und Nachteile

Die technischen Vor- und Nachteile eines Parabolmikrofons sind vielfältig und werden unter anderem von den bereits angesprochenen akustischen Eigenschaften des Hohlspiegels, wie auch zusätzlich von äußeren Aufnahmebedingungen beziehungsweise der Signalquelle bestimmt. Diese besitzen unterschiedliche Auswirkungen auf die einzelnen Frequenzbereiche, die folglich separiert betrachtet werden.

Ein großer Vorteil des Parabolmikrofons ist seine Richtwirkung und Reichweite, welche die akustische Verstärkung impliziert und es ermöglicht, Signale aus größerer Distanz, im Vergleich zu anderen Mikrofonen, aufzunehmen. Dadurch ist das Einsatzgebiet des Parabolmikrofons vielfältig und bietet eine Vielzahl an Experimentiermöglichkeiten. Mögliche Verwendungsgebiete sind Sportveranstaltungen, wie Fußball oder Football, über Naturaufnahmen von Singvögel, kleinen bis großen Säugetieren, Wasserfällen, bis hin zu Motorengeräuschen von Autos oder Motorrädern. Gleichzeitig steigen mit zunehmenden Einsatzzielen auch die Anforderungen an die Postproduktion und Komplexität der Filterabhängigkeiten in der Postproduktion.

Einige Schwierigkeiten beziehungsweise Nachteile besitzt das System bei tieffrequenten Signalen mit hohem Diffusfeldanteil, wie bereits in 3.1.1 und 3.1.2 angesprochen wurde. Durch die hohe Korrelation der beiden Schallfelder ist das Parabolsystem nicht in der Lage, trotz seiner Richtwirkung eine klanggetreue Wiedergabe der tiefen Frequenzen der Klangquelle zu liefern. Dadurch wird die subjektive Hörqualität verschlechtert und eine spätere Nachbearbeitung erschwert.

Die in den Grundlagen angesprochene akustische Verstärkung gilt, wie in der Frequenzantwort 3.1 ersichtlich, nur für den Mitten- beziehungsweise den Höhenbereich und nicht für den Bassbereich. Dadurch entsteht ein verzerrtes Abbild der Klangquelle, wenn es sich dabei um ein breitbandiges Signal handelt. Für die Postproduktion ist es deswegen essentiell, ein ausgewogenes Verhältnis aus Höhen-, Mitten- und Tiefenanteilen wiederherzustellen beziehungsweise mischen zu können.
Da sich die Klangquellen je nach Einsatzgebiet in einem unterschiedlichen Abstand zum Mikrofon befinden, muss das Plugin die Möglichkeit bieten, Signalanteile entfernungsabhängig verändern zu lassen. Für weit entfernte Objekte muss hierbei die natürliche Dissipation, siehe 2.2.1, berücksichtigt werden, die mit steigender Entfernung zunimmt und vor allem Höhenanteile im Signal reduziert. Für die Postproduktion

ist hierbei eine Höhenverstärkung ein wichtiger Faktor, der berücksichtigt werden muss. Für den tieffrequenten Anteil müssen dagegen gegenteilige Maßnahmen ergriffen werden, obwohl auch hier eine geringfügige Dissipation auftritt. Da durch zunehmende Entfernung auch der prozentuale Anteil an Diffusschall zunimmt, kann kein Tiefenausgleich vorgenommen werden, ohne auch den Diffusschall zu verstärken. Grund hierfür ist, dass der Einfallswinkel des Diffusschalls in Relation zum Mikrofon, mit größerem Abstand abnimmt. Durch den geringen Einfallswinkel werden diese Signalanteile von dem Nierenmikrofon geringer abgedämpft und sind vermehrt auf der Aufnahme zu hören.

Durch die manuelle Ausrichtung des Parabolmikrofons von Hand kann es zu Erschütterungen am Mikrofonstativ und Klopfgeräuschen kommen, welche auf der Aufnahme hörbar werden. Eine vorsichtige Handhabung ist bei der Aufnahme notwendig, die allerdings die Bewegungsfreiheit einschränkt und es erschwert, schnelle Objekte präzise zu erfassen. Aus diesem Grund ist für die spätere Verwendung ein Filter notwendig, der die angesprochenen Störsignale herausfiltern beziehungsweise abschwächen kann. Da dieser Störschall frequenztechnisch betrachtet leicht im Tiefenbereich einzuordnen ist, kann hierfür ein steiles Hochpassfilter eingesetzt werden. (vgl. S. 339-340 [DDHW13])

3.3 Anforderungen an das Audioplugin

Die Anforderungen an ein Audioplugin sind vielfältig und beginnen bereits bei der Auswahl der zu Grunde liegenden Plattform (Betriebssystem) und der Plugintechnologie, siehe 2.3.3. Um sicher zu stellen, dass keine Fehlentscheidungen diesbezüglich getroffen werden, muss entweder auf eine weit verbreitende Technologie zurückgegriffen oder ein Hybrid entwickelt werden, welches mehrere Formate beziehungsweise Technologien unterstützt. Auf Grund dessen folgt in den weiteren Unterkapiteln eine Evaluierung gängiger DAWs und ihre jeweilige Pluginunterstützung beziehungsweise technische Anforderungen an die Postproduktions-Möglichkeiten des später entwickelten Plugins.

3.3.1 Auswahl des Pluginformats und des Betriebssystems

Um die Marktanteile und Verbreitung von verschiedenen DAWs genauer eingrenzen zu können, gibt es immer wieder Umfragen von Onlineseiten wie ask.audio, audioskills.com[2] oder songwriter24.de[3]. Diese Umfragen beschränken sich hierbei meistens auf einen lokalen Markt, wie den amerikanischen oder deutschsprachigen Raum. Aus diesem Grund können die Umfragen nur eine Tendenz wiedergeben, aber keine weltweit

[2]https://audioskills.com/data/daws/

[3]songwriter24.de/blog/studie-beste-beliebteste-daw-ergebnisse/

einheitliche Aussage. Zur Analyse dieser Tendenz wurde eine Umfrage der Internetseite ask.audio aus dem Jahr 2018 verwendet. Diese zeigt eine Befragung unter mehr als 30 600 Musikern und Produzenten nach ihrer bevorzugten DAW:

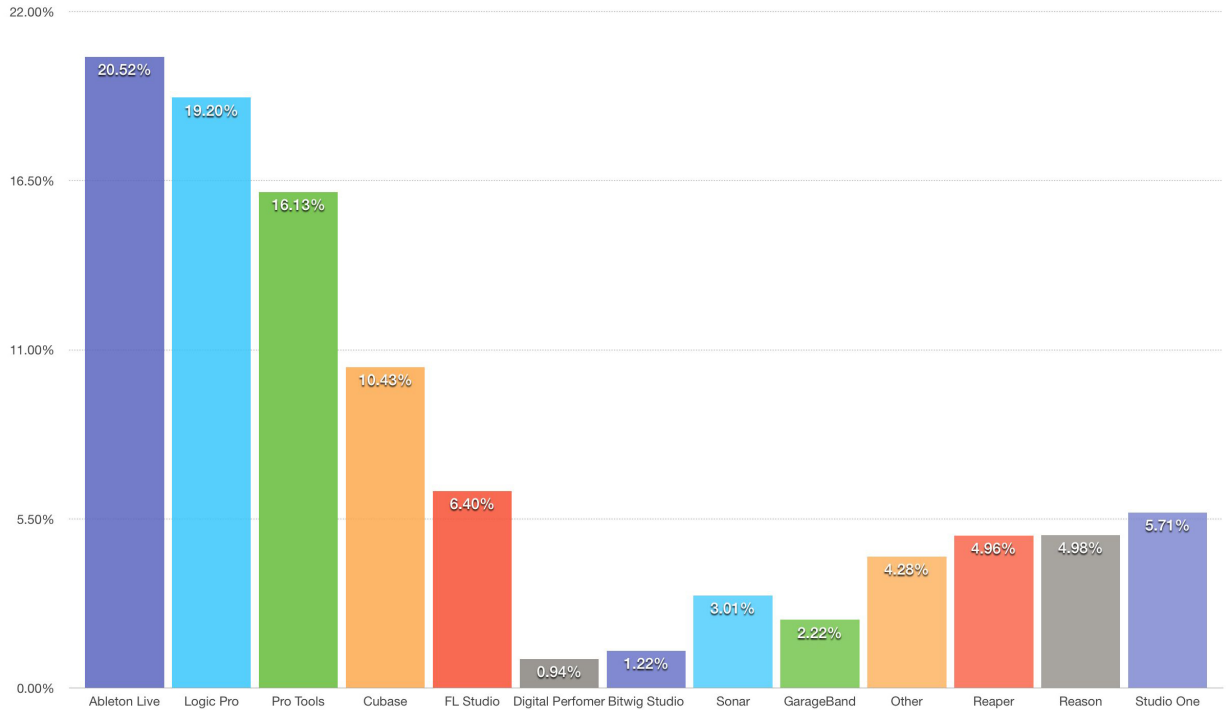

Abbildung 3.4: Marktanteile von DAWs nach ask.audio
Quelle: https://ask.audio/articles/top-12-most-popular-daws-you-voted-for, abgerufen am 20.10.2018

Auf Platz 1 wird Ableton Live der Berliner Softwarefirma Ableton AG angeführt, dicht gefolgt von Logic Pro und Pro Tools mit wenigen Prozentpunkten Unterschied. Auf den Plätzen 4 und 5 befinden sich die Software Cubase der Firma Steinberg und FL Studio, ehemals Fruity Loops, der Firma image-line. Deutlich wird, dass es keine Marktdominanz einer einzelnen DAW gibt, sondern dass die ersten 5 DAWs fast 75% der gesamten DAWs ausmachen.
Um die genannten Ergebnisse vergleichen zu können, zeigt die nachfolgende Abbildung eine Auswertung von deutschsprachigen DAW-Benutzern, welche die Webseite songwriter24.de erstellt hat. Die Teilnehmeranzahl ist hierbei allerdings wesentlich kleiner und auf 630 Teilnehmer beschränkt. Befragt wurden DAW-Nutzer des Recording.de-Forums (68%), Facebook- und Twitterseite von Sound & Recording (27%) und 5% in diversen Foren (guitarworld.de, musikertalk.de, tonstudio-forum.de). Laut Autor wurde besonders darauf geachtet, nicht in spezifischen DAW-Foren oder Unterforen zu befragen. (vgl. [Oet])

Abbildung 3.5: Marktanteile von DAWs nach songwriter24.de
Quelle: https://www.songwriter24.de/blog/studie-beste-beliebteste-daw-ergebnisse/, abgerufen am 22.10.2018

Auffällig ist, dass die ersten 4 DAWs mit der Umfrage von ask.audio deckungsgleich sind, wenn auch nicht in gleicher Reihenfolge auftreten. In der Umfrage von Bernd Oettinger ist die führende DAW Cubase mit deutlichem Abstand vor der zweitplatzierten oder den übrigen DAWs genannt. Dies kann mit kulturellen Unterschieden, da Steinberg eine in Deutschland gegründete Firma ist, oder mit anderen beruflichen Ansprüchen beziehungsweise Zielgruppen erklärt werden. Auf Grund der geringen Teilnehmerzahl kann hierbei auch eine gewisse Ungenauigkeit nicht ausgeschlossen werden.

Um mit dem entwickelten Plugin eine möglichst große Anzahl an DAWs zu unterstützen, wird im weiteren Verlauf die genaue Technologie hinter den in beiden Umfragen vorkommenden DAWs aufgelistet. Aus Gründen der Übersichtlichkeit wurde die Auflistung von 32 beziehungsweise 64-bit Kompatibilität nur dann erwähnt, wenn nur eine der beiden Bitversionen unterstützt wird. Manche Hersteller geben auch keine genaue Angabe auf ihrer Produktseite an, um welche Kompatibilität es sich in Bezug auf die Pluginformat-Version handelt. Deshalb wird im diesem Fall nur der Name des Formats, beispielhaft VST oder AU verwendet. Die Tabelle wurde um die DAW Sequoia

14 erweitert, die nicht genannt wurde, aber an der Hochschule der Medien eine Rolle spielt. Auf Grund der Ähnlichkeit und Überschneidungen bezüglich der Funktionalität und Kompatibilität wurden die DAWs Cubase und Nuendo des Herstellers Steinberg nicht separat aufgelistet.
Folgende Auflistung zeigt die Kompatibilität der einzelnen DAWs:

DAW-Name	**Pluginformat**	**Betriebssystem**
Ableton Live 9-10 [1]	VST2, AU2	macOS, Windows
Logic Pro X[2]	AU, AU2, AU3	macOS
Pro Tools 11-12[3]	AAX (64-bit)	macOS, Windows
Cubase 9[4]	VST2, VST3	macOS, Windows
FL Studio 20[5]	VST2, AU (64-bit)	macOS, Windows
Digital Performer 9[6]	VST, AU	macOS, Windows
Bitwig Studio[7]	VST2, VST3	macOS, Windows, Linux
Sonar X3[8]	VST2, VST3	macOS (Prototyp), Windows
GarageBand[9]	AU, AU2, AU3	macOS
Reaper 5[10]	VST2, VST3, AU	macOS, Windows
Reason 10[11]	VST2	macOS, Windows
Studio One 4 Professional[12]	VST2, VST3, AU	macOS, Windows
Sequoia 14[13]	VST2, VST3	Windows
Samplitude Pro X3[14]	VST2, VST3	Windows
Music Maker 2019 PE[15]	VST2, VST3	Windows

[1] help.ableton.com/hc/en-us/articles/209769405-Supported-Plug-in-formats
[2] support.apple.com/kb/PH26318?viewlocale=de_DE&locale=de_DE
[3] avid.force.com/pkb/articles/compatibility/en343311
[4] www.steinberg.net/en/products/cubase/resources.html
[5] www.image-line.com/support/flstudio_online_manual/html/basics_externalplugins.htm#AU
[6] s3.amazonaws.com/motu-www-data/manuals/software/dp/v951/Digital+Performer+User+Guide.pdf
[7] www.bitwig.com/en/support/faq.html
[8] www.cakewalk.com/Documentation?product=SONAR%20X3&language=1&help=Dialogs2.064.html
[9] support.apple.com/kb/PH25047?viewlocale=de_DE&locale=de_DE
[10] www.reaper.fm/

Betrachtet man die Kompatibilität der einzelnen DAWs mit verschiedenen Pluginformaten, wird auffällig, dass mit dem Pluginformat VST nahezu alle DAWs abgedeckt werden können. Die Unterstützung für AudioUnit ist bedingt durch die macOS-Betriebssystem-Voraussetzung, siehe 2.3.3.3, stärker limitiert und grenzt die Verwendung ein. Das AAX-Format wird nur von Pro Tools unterstützt und hat aus diesem Grund die niedrigste Priorität hinsichtlich der Auswahl des Pluginformats.
Die Unterstützung mehrerer Betriebssysteme spielt eine wichtige Rolle, da viele DAWs für beide Betriebssysteme Windows und macOS zur Verfügung stehen und somit theoretisch Verwendung findet. Eine Entwicklung für Linux kann unter diesen Gesichtspunkten vernachlässigt werden, da es nur eine direkte Portierung auf das genannte Betriebssystem unter den aufgelisteten DAWs gibt (Stand 22.10.2018).

Zusammenfassend lässt sich sagen, dass das entwickelte Plugin in den beiden Formaten VST und AudioUnit auf jeweils beiden Betriebssystemen Windows und macOS implementiert werden sollte. Optional kann auch noch das Format AAX unterstützt werden.

3.3.2 Digitale Anpassung des Frequenzgangs

Das folgende Kapitel beschäftigt sich mit der Auswahl geeigneter Filter, welche zur Postproduktion des Parabolmikrofonsignals benötigt werden.
Um die in 3.2 angesprochenen Griffgeräusche zu reduzieren, die bei Anwendung zwischen Hand des Anwenders und dem Mikrofonstativ entstehen, wird ein Hochpassfilter benötigt. Dieser kann sich bei den Parametern an vorhandenen Tiefenabsenkungen bei Mikrofonen der Firma Schoeps orientieren. Als Mikrofonreferenz wurde hierbei das MiniCMIT ausgewählt, welches ein festes Hochpassfilter 4. Ordnung bei einer Grenzfrequenz(f_g) von 70 Hz besitzt. Zur Umsetzung dieses Hochpasses kann beispielsweise ein Butterworth-, Tschebyscheff, Bessel- oder Cauer- (elliptischer) Hochpass verwendet werden. Diese unterscheiden sich neben ihrer Phasenlinearität auch in ihrer Übergangscharakteristik (engl. = roll off) und Dämpfung pro Oktave. Hierbei wird

[11] www.propellerheads.com/fr/reason/new
[12] www.presonus.com/produkte/de/Studio-One/version-auswaehlen
[13] www.thomann.de/de/magix_sequoia_14.htm
[14] www.magix.com/de/musik/samplitude/technische-daten/#c710442
[15] magazine.magix.com/de/vst-instrumente-teil-1/

bei einer stärkeren Übergangscharakteristik auch ein höheres Überschwingen in Nähe der Grenzfrequenz und teilweise im Sperrbereich erzeugt. Dies ist bei Tschebyscheff- und Cauer-Filtern, wie auch bei Butterworth-Filter mit höherer Ordnung der Fall. Die Cauer-Filter besitzen hierbei auch im Sperrbereich eine gewisse Welligkeit. Zusätzlich erzeugen die genannten Filter eine nicht lineare Phase im Übergangsbereich. (Vgl. im Englischen [Cir]).
Folgende Abbildung zeigt die Unterschiede bei der Verwendung eines Hochpasses mit den angesprochenen Filtertopologien. Ein Cauer-Filter wurde auf Grund seiner starken Welligkeit im Sperr-, wie im Durchlassbereich im Vergleich zu den anderen Filtern nicht in Betracht gezogen.

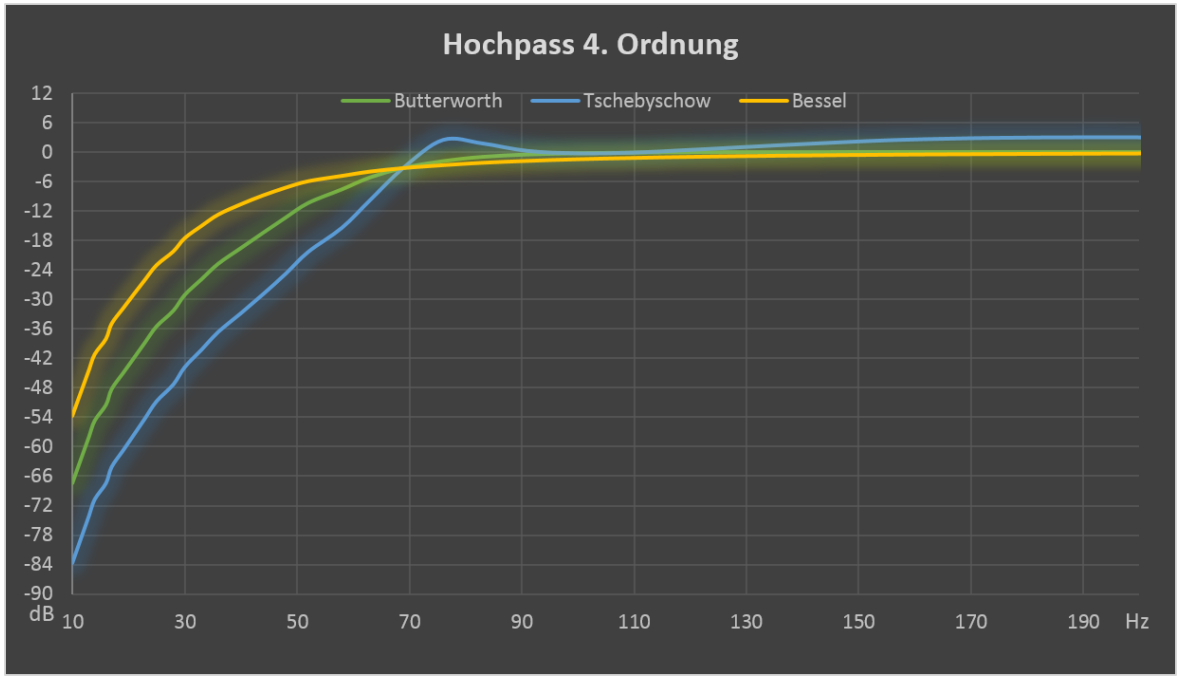

Abbildung 3.6: Frequenzantwort dreier Hochpassfilter 4. Ordnung mit der f_g= 70 Hz
Quelle: Eigene Abbildung

Ein Bessel-Filter besitzt im Vergleich den längsten Übergangsbereich und eine durchgehende lineare Phase im Durchlassbereich. Dadurch wird die Übertragungscharakteristik nahezu ideal, die Flankensteilheit aber am geringsten im Vergleich zu den anderen Filtern. Das Tschebyscheff-Filter besitzt eine starke Überhöhung in der Nähe der Grenzfrequenz, die wie bereits erwähnt der Flankensteilheit geschuldet ist. Das Butterworth-Filter besitzt einen fast linearen Phasenverlauf im Durchlass beziehungsweise Sperrbereich. Er entspricht einem Kompromiss aus fast-linearer Phase und Steilheit im Übergangsbereich.
Aus diesem Grund wurde für den Hochpassfilter in der späteren Implementierung auf einen Butterworth-Filter zurückgegriffen. Um den Hochpass des MiniCMIT zu imitieren, können zwei Butterworth-Filter 2. Ordnung, in Serie geschaltet werden.

Da Griffgeräusche während der Aufnahme nur ein möglicher Störfaktor sind, kann das Hochpassfilter ein zuschaltbarer Parameter sein, welcher je nach Anwenderwunsch ausgeschaltet werden kann.
Dem Anwender soll es später möglich sein, die Mikrofonkapseln auszuwechseln und somit die Richtcharakteristik ändern zu können. Diesbezüglich muss das Plugin die Filterkurven für einzelne Mikrofone gegebenenfalls anpassen und die Möglichkeit zur Auswahl des Mikrofons bieten. Bei der Verwendung einer Kugelcharakteristik kann der eingangs erwähnte Hochpass gegebenenfalls mit einer niedrigen Ordnung verwendet werden, da wie in 2.2.2 erwähnt, die Windgeräusche einen geringeren Einfluss besitzen.

Die in Kapitel 2.1.3.2 angesprochenen Windschutze sollten ebenfalls im Plugin auswählbar sein und die in 3.3 aufgezeigten Signalabdämpfungen kompensieren. Hierbei kann es ausreichend sein, bereits implementierte Filter um die benötigten Verstärkungen zu erweitern, sofern sich die Frequenzbereiche überschneiden.

3.3.3 Hilfreiche Parameter zur Anpassung des Eingangsignals

In Rücksprache mit der Firma Schoeps sollen zur Postproduktion vordefinierte Parameter zur Verfügung gestellt werden, die es auch unerfahrenen Benutzern ermöglicht, frequenztechnische Anpassungen am Ausgangssignal vorzunehmen. Um die Handhabung dabei intuitiv und verständlich zu gestalten, wurden aus diesem Grund übergeordnete Begriffe für die Parameter verwendet, welche das Verständnis erleichtern. Ziel dabei war, dass der Anwender sich überwiegend auf das Hören konzentriert und die Parameter nicht nur nach Werten einstellt, welche durch die Aufnahmesituation definiert sind. Schlussendlich wurde sich dabei auf die folgenden Parameter geeinigt:

Parameter-Name	**Minimum**	**Maximum**
Distanz	12 m Abstand	30 m Abstand
Zielungenauigkeit	100 % auf Parabolmikrofonachse	0 % auf Parabolmikrofonachse
Diffus-/Freifeld	Pures Diffusfeld	Pures Freifeld

Der Minimum- beziehungsweise Maximumwert soll dabei die Einstellungen des Parabolmikrofons während der Aufnahme wiedergeben. Dies bedeutet für den Nutzer die Übertragung der ungefähren Distanz von Mikrofon zu Zielobjekt mit dem Distanzparameter. Die Einschätzung der Zielgenauigkeit beziehungsweise punktgenaue Ausrichtung des Mikrofons und Hohlspiegels, erfolgt mit dem Parameter Zielungenauigkeit. Hierbei ist die Besonderheit, dass nur eine Filteranpassung eintritt, wenn der Anwender der Meinung ist, das Zielobjekt nicht genau getroffen zu haben. Mit dem letzten Parameter Diffus-/Freifeld soll das Verhältnis von Direktschall zu Diffusschall eingestellt werden können. Mit dem Minimum soll das pure Diffusfeld mit Reflexionen, Umgebungsge-

räuschen und Kammfiltereffekte zu hören sein. Bei purem Freifeld soll theoretisch reiner Direktschall von der Signalquelle aufgenommen werden.

3.3.4 Visualisierung der Frequenzganganpassungen

Um dem Anwender eine visuelle Rückmeldung über frequenztechnische Anpassungen zu geben, die er mit den in 3.3.2 und 3.3.3 beschriebenen Parametern einstellen kann, sollte das Plugin den Frequenzgang aller Filter abbilden. Die nachfolgende Abbildung zeigt vergleichsweise den Aufbau des parametrischen Pro-Q 2-Equalizers der Firma FabFilter.

Abbildung 3.7: Pro-Q 2 Filter der Firma FabFilter
Quelle: https://fabfilter.com/help/pro-q/, abgerufen am 21.11.2018

Der Pro-Q 2 enthält die für Equalizer im Audiobereich angewandten, typischen Einheiten von 0 Hz bis 20 kHz auf der horizontal Achse und eine einstellbare maximal Verstärkung von 12 dB,s auf der Vertikalachse. Das zu implementierte Plugin sollte ebenfalls den gleichen Frequenzbereich abdecken und als Maximalpegel einen auf die Parameter abgestimmte Skalierung besitzen. Darüber hinaus sollte diese Skalierung durch einen kleinen Headroom[4] erweitert werden, der dem Anwender einen gewissen Spielraum ermöglicht.

[4]englisch für Aussteuerungsreserve. Entspricht Differenz von Nenn- zu Maximalpegel.

Für die Farbe der Frequenzgangkurve wurde vorgesehen, analog zu den prägnanten Farben des Pro-Q2 eine typische Farbe der Firma Schoeps zu verwenden. Dies erleichtert die Lesbarkeit und sorgt für die Einhaltung des CI[5] im Plugin. Eine direkte Veränderung der Filter durch eine grafische Interaktion mit dem Filterkurven, wie es im Pro-Q2 möglich ist, wurde von der Firma Schoeps nicht gefordert.

3.3.5 Presets für Audioplugins

Um den Einstieg für Benutzer zu vereinfachen, soll es möglich sein, innerhalb kurzer Zeit eine Einstellung für eine Aufnahme zu finden, welche der Aufnahmesituation des Anwenders entspricht. Aus diesem Grund kann es sinnvoll sein, dem Benutzer mögliche Aufnahmesetups vorzugeben, die er als Grundeinstellung verwenden kann. Presets erfüllen diese Anforderungen, in dem sie die Aufnahmesituation mit ihrem Namen beschreiben und der Benutzer zwischen eigenen Einstellungen und vorgefertigten Presets wechseln kann. Auf Grund dessen sollte das Plugin fertig eingestellte Grundeinstellungen besitzen, mit dem Ziel, den Workflow zu vereinfachen und zu beschleunigen.

[5]Corporate Identity = einheitliches Erscheinungsbild einer Firma

4. Plugin-Entwicklung mit dem JUCE-Framework

Das folgende Kapitel liefert einen Einblick in die Konzeption und Implementierung des Audioplugins Parab EQ mit dem JUCE-Framework. Der Name des Plugins entspricht zum Zeitpunkt der Thesis einem Arbeitstitel und wird gegebenenfalls zur Veröffentlichung verändert. Zur Umsetzung der Implementierung wurde das JUCE-Framework in der Version 5.3.2 mit dem C++ 14. Standard verwendet.

Zu Beginn des Kapitels wird die Idee des Clean Codings vorgestellt und die Coding Guidelines erläutert, welche im gesamten Projekt mit einzelnen Einschränkungen eingehalten werden. Darauf folgend wird die Struktur des Plugins vorgestellt und die Vorgehensweise zur Gestaltung der Pluginoberfläche präsentiert. Im Kapitel 4.3.3 werden darauf aufbauend sukzessiv die Schritte zur konkreten Implementierung der Filterfunktionalität und die Umsetzung der Plattformunabhängigkeit aufgezeigt.

4.1 Prinzipien des Clean Codes

Der Name Clean Code wurde von Robert Cecil Martin mit seinem gleichnamigen Buch geprägt. Als sauberen Code wird hierbei Quellcode zusammengefasst, der unter anderem übersichtlich, leicht zu verstehen beziehungsweise zu lesen, wartbar und offen für Erweiterungen ist. Diese Ideen wurden bereits teilweise in den Prinzipien „Don't repeat yourself (DRY)“ und „Keep it simple [and] stupid (KISS)“ aufgefasst. Das DRY-Prinzip von Dave Thomas und Andy Hunt verlangt vom Softwareentwickler die Vermeidung von redundanten oder doppelten Codezeilen. Diese bergen bei Veränderungen potenzielle Gefahren- beziehungsweise Fehlerquellen und dürfen bei der Softwarewartung nicht vergessen werden. Für die Umsetzung dieses Prinzips ist es erforderlich, dass bestimmte Funktionen oder Klassen ausgelagert werden, sofern deren Funktionalität an mehreren Stellen benötigt wird. Des Weiteren ist es sinnvoll, keinerlei redundante Informationen in Datenstrukturen/Klassen zu erzeugen.

Folgendes Beispiel zeigt eine redundante Information in einer primitiven C++-Klasse aus dem von Thomas und Hunt verfassten Buch „Der pragmatische Programmierer“:

```
class Line {
        public:
                Point start;
                Point end;
                double length;
};
```

Listing 4.1: Line-Klasse aus „Der pragmatische Programmierer“

Die Information der Länge zwischen beiden Punkten kann dynamisch aus den Koordinaten der Punkte berechnet werden. Aus diesem Grund handelt es sich um eine redundante Variable, die bei Änderungen am Start- oder Endpunkt nicht automatisch aktualisiert wird. Sie beinhaltet somit eine potenzielle Fehlerquelle und sollte entfernt werden. Eine Möglichkeit, diese Problematik zu umgehen, besteht darin, aus der Variable eine Methode zu machen, welche die Länge dynamisch berechnet und zurückgibt. (Vgl. S. 26 [HT03])

Das eingangs erwähnte KISS-Prinzip erfordert, dass der Entwickler seine Implementierungen auf das Nötigste reduziert, um eine Funktionalität umzusetzen. Mögliche zusätzliche Funktionen, die möglicherweise in zukünftigen Anforderungen benötigt werden, dürfen aus diesem Grund nicht eingebaut werden.
Das nächste Kapitel beschäftigt sich mit Coding Guidelines und der Umsetzung der genannten Prinzipien an konkreten Objekte und Datenstrukturen.

4.2 Coding Guidelines

Im gesamtes Projekt werden die Coding Guidelines eingehalten, die von JUCE-Gründer und Softwarearchitekten Julian Storer, in dem auf der JUCE-Seite veröffentlichten Artikel Coding Standards[1], festgehalten wurden. Ziel dabei ist die höhere Wartbarkeit, bessere Verständlichkeit und die Vermeidung von unnötig komplexen Datenstrukturen in geschriebenen Klassen und Dateien. Um dies zu verdeutlichen, werden im folgenden Verlauf einige Beispiele aufgelistet, welche die genannten Prinzipien aufgreifen und ihre mögliche Umsetzung aufzeigen.

[1] https://juce.com/discover/stories/coding-standards

4.2.1 Pointer

Die Programmiersprache C++ verwendet zur Ressourcenfreigabe Destruktoren, um nicht mehr benötigte Objekte zu zerstören. Dies bedeutet, dass es keine standardmäßige, automatische Speicherfreigabe durch sogenannte Garbage Collectoren, wie in anderen Hochsprachen wie C# oder Java gibt. Diesbezüglich ist es empfehlenswert, bei der Verwendung von Pointern auf intelligente Zeiger, wie ScopedPointer oder seit dem C++ 11-Standard auf „UniquePointer (std::unique_ptr)“ zurückzugreifen. Denn diese überwachen die Lebensdauer von Pointern und können somit nicht mehr benötigte Ressourcen freigeben. Der Programmierer wird somit vor den Gefahren von sogenannten Memory Leaks[2] bewahrt, die dazu führen, dass belegter Speicher im Arbeitsspeicher nicht mehr korrekt freigegeben wird.
Für viele Entwickler stellen Pointer eine effektive, aber risikobehaftete Arbeitsweise dar, weswegen sie empfehlen, Pointer nur dann einzusetzen, wenn es unbedingt notwendig ist. In der fertigen Anwendung werden float-Pointer verwendet, die auf Filterparameter zeigen. Diese Filterparameter werden in einem zentralen AudioProcessorValueTreeState gespeichert und können somit effektiv ausgelesen werden. Die Funktionsweise des AudioProcessorValueTreeState-Objekts wird später in 4.3.2.1 genauer erklärt.

4.2.2 Namenskonventionen

Im ganzen Projekt wurden die folgenden Namenskonventionen eingehalten, um die Lesbarkeit zu verbessern. Als Implementierungssprache wurde Englisch gewählt. Variablennamen werden nach dem Kamelprinzip[3] geschrieben: myVariableName. Die Verwendung von Unterstrichen ist nur zur Unterteilung von langen Variablennamen erlaubt und nicht zu Beginn der Variable, da diese Form bereits in Standardbibliotheken verwendet wird. Klassennamen verwenden das Kamelprinzip ebenso, allerdings mit führendem Großbuchstaben: MyClassName. Die Ungarische Notation[4] wird ebenfalls nicht verwendet. Diese impliziert den Typ der Variable in den Variablennamen zu übernehmen, wie beispielsweise floatSampleRate. Hierbei steigt unter anderem die Gefahr, bei Typkonventierungen[5] den Variablennamen nicht zu aktualisieren und somit Fehler, wie beispielsweise Rundungsfehler zu verursachen.

4.2.3 Dokumentation

Die Quellcode-Dokumentation ist für viele Programmierer eine langwierige und scheinbar überflüssige Arbeit nach der Implementierung der Funktion. Allerdings ermöglicht

[2] engl. für Speicherleck im Arbeitsspeicher.
[3] im Englischen „Camel Case“
[4] im Englischen „Hungarian notation“
[5] im Englischen „type cast“

sie bei korrekter Anwendung eine aussagekräftige Beschreibung und Erklärung der Implementierung. Dies kann für nachfolgende Programmierer oder bei Anpassungen am Quellcode nach längerer Zeit genutzt werden und erleichtert das Zurechtfinden und Verständnis von Programmlogik. Dadurch wird die Wartbarkeit erhöht und Korrekturen oder Änderungen können effizienter durchgeführt werden.
Aus diesem Grund wird die Erstellung von Quellcode-Kommentaren und Dokumentation von Methoden und Klassen im ganzen Projekt eingehalten. Die Art der Dokumentation sollte sich dabei an den Stil von Javadoc, dem Dokumentationswerkzeug von Java-Quellcode halten (vgl. [Par18]).
Dieser definiert unter anderem, dass mehrzeilige Dokumentationsblöcke mit „/**“ beginnen und mit „*/“ beendet werden. Für einzeilige Kommentare kann folgende Notation verwendet werden „///“. Des Weiteren werden Methodenparameter mit der vorangehenden Auszeichnung „@param“ deklariert. Für Methodenrückgaben gilt die selbe Notation mit „@return“. Folgendes Listing zeigt die verwendete Codedokumentation an Hand der im Projekt verwendeten getFactorForFrequency-Methode in der ParabolaAudioProcessorEditor-Klasse:

```
/**
        Calculates the factor for a given frequency to draw the plot units

        @param freq is the current frequency in the filter plot
        @return factor for a frequency
*/
double ParabolaAudioProcessorEditor::getFactorForFrequency(const float freq)
{
 return ...
}
```

Listing 4.2: Dokumentation der getFactorForFrequency-Methode

4.2.4 Diverses

Bei der Verwendung von Schleifen sollte die Präinkrementierung vor der Postinkrementierung bevorzugt werden, da sie laut Julian Storer in komplexen Datenstrukturen effizienter arbeiten kann (vgl. [Juc]).
Für Schleifen mit einem primitiven Schleifenzähler, beispielsweise Datentyp int, muss hierbei bei der Präinkrementierung keine temporäre Variable mit dem alten Wert angelegt werden, wie es bei Postinkrementierung der Fall ist. Allerdings kann dieser Umstand auch von dem jeweiligen Compiler ohne Laufzeitunterschiede optimiert werden. Da es keinen zusätzlichen Programmieraufwand bedeutet, wurde im Projekt die Präinkrementierung eingesetzt, um eine mögliche Performanceverbesserung zu bewirken.

4.3 Plugin-Implementierung

Zum Umsetzung des Audioplugins wurde ein neues Projekt auf Grundlage des Audioplugin-Template erzeugt, welches zuvor bereits in 2.11 dargestellt wurde. Durch die Templateverwendung wird die Projektstruktur an gewissen Stellen bereits vorgegeben. Zur Umsetzung der in 3 angesprochenen Anforderungen, wie beispielsweise Filter oder Audioprozessierung, wurde zusätzlich das DSP-Modul von JUCE eingesetzt. Dies ist in den folgenden Kapiteln durch die Verwendung des Namensraum[6] vor Datentypen, wie „dsp::IIR::Filter<float>“ ersichtlich. Des Weiteren wurden sechs Filter implementiert, welche die angesprochene Anforderungen in 3.3.2 und 3.3.3 erfüllen. Die genaue Beschreibung erfolgt in Kapitel 4.3.3
Folgende Kapitel beschäftigen sich mit der Gestaltung der Pluginoberfläche, Projektstruktur und Implementierung des Parab EQ-Plugins.

4.3.1 Konzeption und Entwicklung der Benutzeroberfläche

Zur Umsetzung der Oberfläche wurde von der Firma Schoeps zu Beginn ein Mockup[7] zur Verfügung gestellt. Die Anordnung der Elemente war hierbei optional und sollte nur einer groben Orientierung dienen und alle für das Plugin erforderlichen Elemente beziehungsweise Parameter enthalten. Folgende Abbildung zeigt das Mockup und die ersten Bezeichnungen für die späteren namentlich angepassten Parameter:

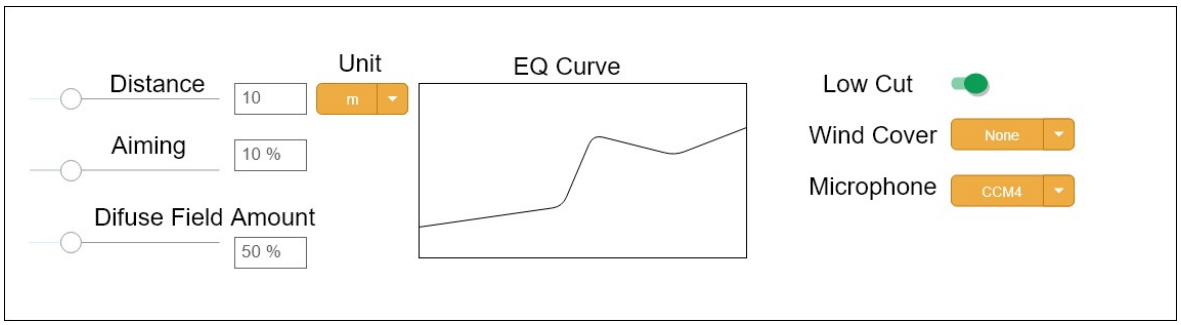

Abbildung 4.1: Parabola Mockup
Quelle: Schoeps GmbH

Aus diesem grafischen Prototypen entstand eine eigene Version, welche vor allem die starke horizontale Ausrichtung des Plugins vermeidet und eine Mischung aus den ersten stark vertikal ausgerichteten Plugin-Ideen des Autors und Schoeps ist. Dazu wurde die Frequenzantwort der Filter unterhalb der Parameter verlagert und die Anordnung selbiger verändert. Statische Parameter, welche die Einstellung des Mikrofons, des Windschutzes oder den Hochpass betreffen, wurden auf die linke Seite verlagert.

[6] engl. namespace
[7] englisch für Vorführmodell

Die Auswahl des Mikrofons beinhaltet hierbei das CCM 4 und das CCM 21. Letzteres wurde für zukünftige Entwicklungen mit eingebaut, aber noch nicht auswählbar gemacht, siehe 3.1.1.
Die dynamischen Parameter, welche in 3.3.3 angesprochen wurden, befinden sich ausnahmslos auf der rechten Seite. Dies geschah mit Rücksicht auf die Aspekte der Gestaltprinzipien, auf die später in Kapitel 4.3.1.1 eingegangen wird. Für die Farbgebung war zum Großteil die später in 4.3.1.2 angesprochene LookNFeel-Klasse von Schoeps verantwortlich.
Folgende Abbildung zeigt den genannten Prototypen, der nahezu alle grafische Komponente der fertigen Anwendung beinhaltet:

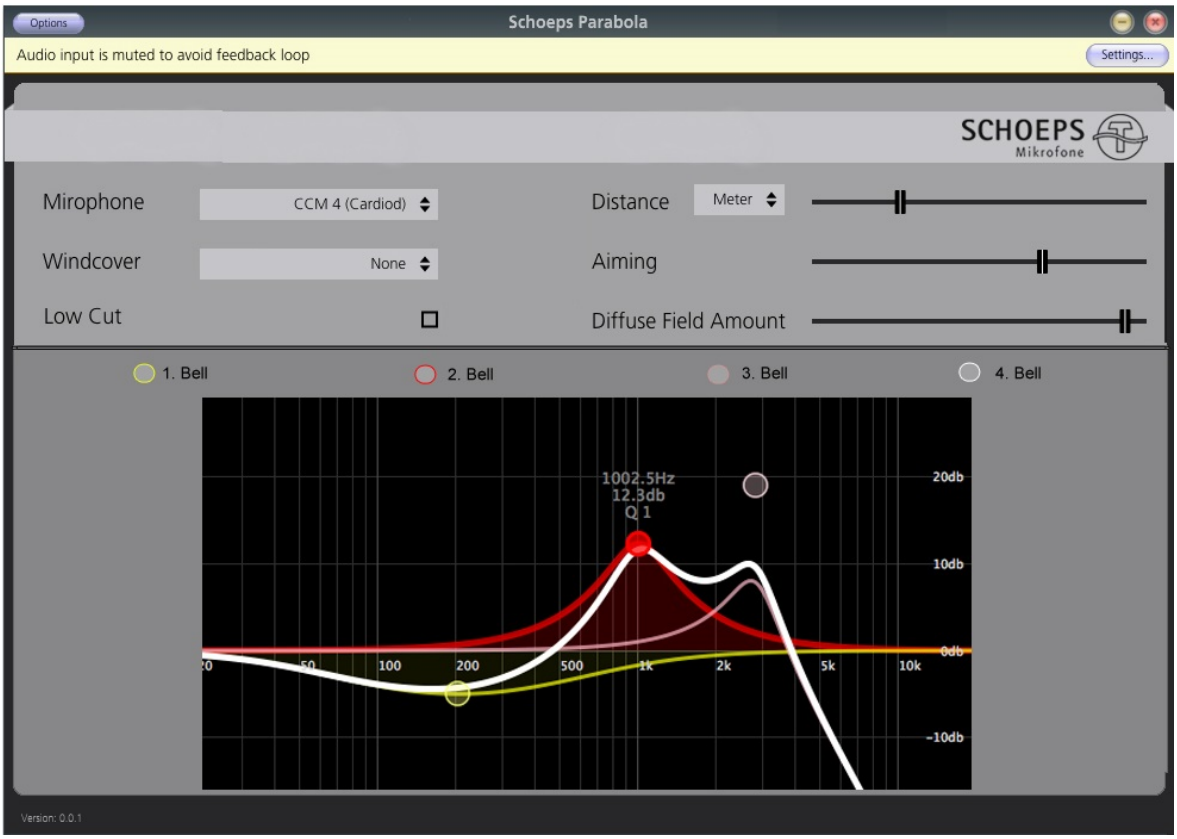

Abbildung 4.2: Vertikal-Horizontal-Mix
Quelle: Eigene Abbildung

Nach Rücksprache mit der Firma Schoeps sollten alle Filter dauerhaft Einfluss auf das Audiosignal haben und nicht wie im Prototypen sichtbar einzeln abschaltbar sein. Aus diesem Grund wurde diese Option in der späteren Anwendung wieder entfernt.

Die Funktionalität des Schoeps-Framework beinhaltet ebenfalls eine „About-View“, die Auskunft über Programmversion, Autor des Plugins und der durch Steinberg vorgeschriebene Verweis auf die VST, siehe 2.3.3.1, enthält. Diese View wird mit einem Mausklick auf das Schoeps-Logo aktiviert und sieht wie folgt aus:

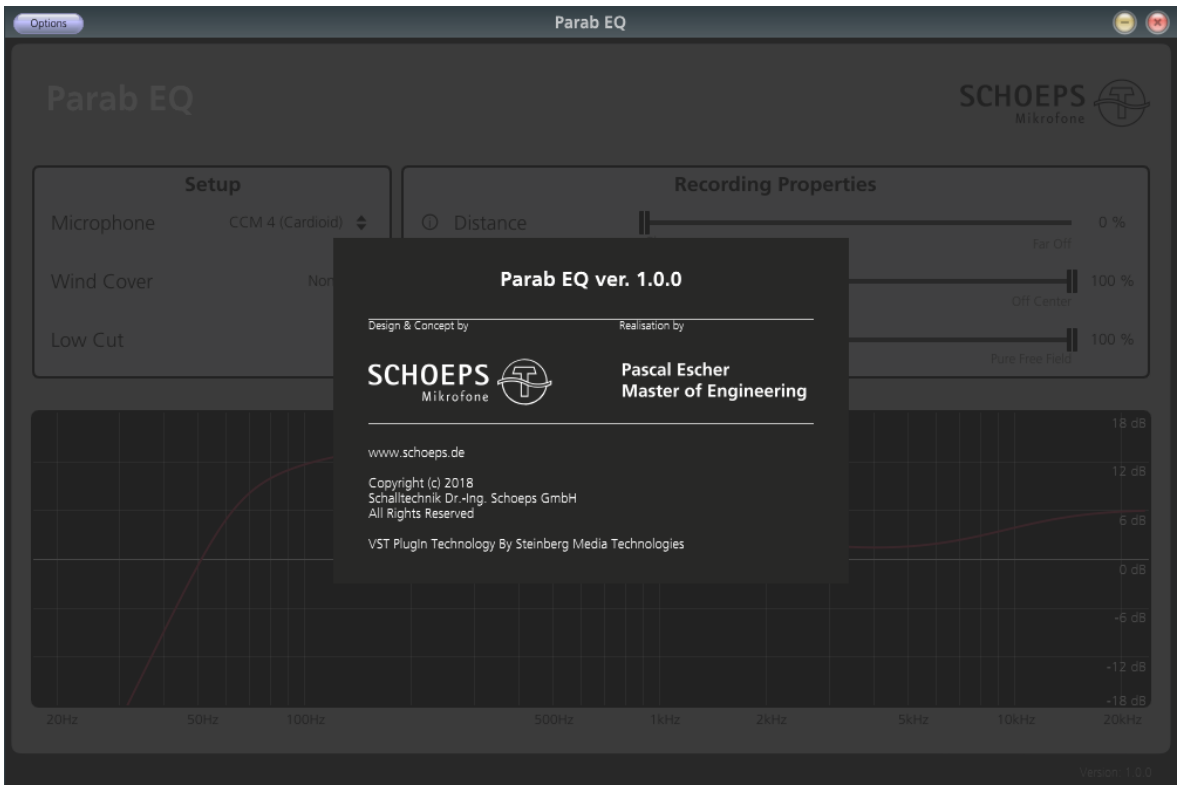

Abbildung 4.3: About View
Quelle: Eigene Abbildung

4.3.1.1 Die Gestaltprinzipien

Aus Gründen der Usability[8] und um eine möglichst kurze Einarbeitungsphase zu ermöglichen, wurden für die Oberflächengestaltung einige Gestaltprinzipien eingehalten. Diese werden im Folgenden an Hand der fertigen Pluginoberfläche erklärt. Der Begriff Usability wird vom Deutschen Institut für Normung hierbei in Norm ISO 9241-11 wie folgt festgehalten: „Usability ist die Effektivität, Effizienz und das Ausmaß der Zufriedenheit, mit denen bestimmte Benutzer spezifizierte Ziele in vorgegebener Umgebung erreichen“ (vgl. S. 2 [Har08]).

Um das Prinzip der Geschlossenheit einzuhalten, wurde für den Setup- beziehungsweise den Recordingbereich ein umschließender Rahmen verwendet. Dies führt dazu, dass die menschliche Wahrnehmung diese als Gruppe beziehungsweise zusammenhängend interpretiert.
Durch das Prinzip der Größe werden die Bereiche hinsichtlich ihrer Priorität unterschiedlich gewichtet. Da der Recording Properties-Bereich im Verhältnis mehr als doppelt so groß ist als der Setup-Bereich, wird ihm mehr Bedeutung zugeteilt. Mit Fokus auf die spätere Anwendung ein gewollter Effekt. Für die einstellbaren Parameter

[8]Englisch für Gebrauchstauglichkeit. Auch als Software-Ergonomie bezeichnet.

gilt das Prinzip der Fortsetzung. Dies bedeutet, dass auf jeden Parameternamen auf gleicher Höhe das interaktive GUI-Element, wie ComboBox, ToggleButton oder Slider folgt. Diese symmetrische Darstellung soll dem Nutzer eine Struktur bieten und für eine einfache Orientierung sorgen.
Die drei Parameter auf der rechten Seite werden nach dem Prinzip der Ähnlichkeit vom Betrachter ebenfalls als Gruppe wahrgenommen, da sie alle die gleichen Dimensionen und Anordnung besitzen. Dadurch bekommen alle Regler eine gleiche Gewichtung in der Anwendung. (Vgl. S. 102-104 [Gol14]).
Folgende Abbildung zeigt das endgültige Plugin in seiner release-fähigen Version als Standalone-Anwendung:

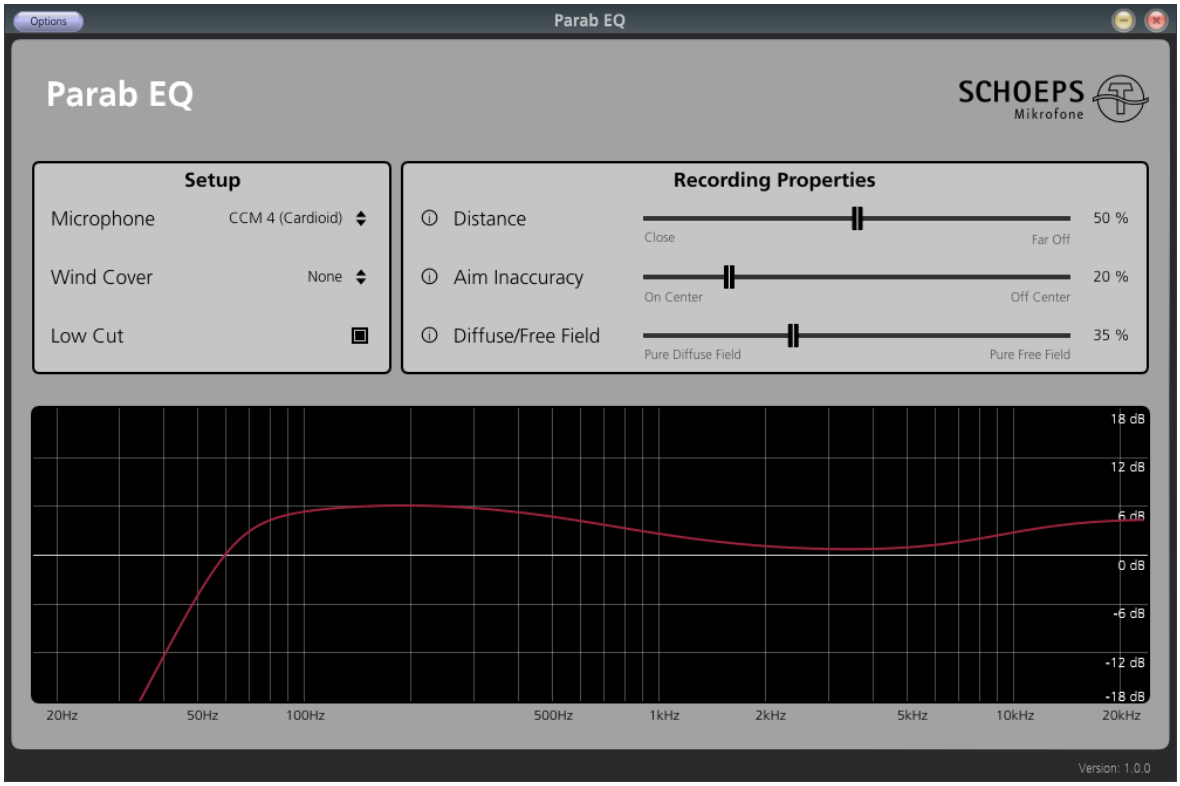

Abbildung 4.4: Parab EQ 1.0 Release-Version
Quelle: Eigene Abbildung

Der Frequenzgangplot beinhaltet passive und neutrale Farben, wie schwarz und weiß, welche den Hintergrund definieren und eine Signalfarbe für die Frequenzgangkurve. Hierbei wurde, wie bereits erwähnt, das spezifische Schoeps-Rot verwendet. Die Größe des Plots ist hierbei zum einen an die Größe der Einstellungsbereiche angelehnt und soll visuell eindrücklich jeder Parameteränderung gerecht werden. Jede Veränderung eines interaktiven GUI-Element soll hierbei direkt durch visuelles Feedback dem Anwender zurückgemeldet werden. Für den Filterfrequenzgang-Plot greift ebenfalls das Prinzip der Größe, da es sich um das größte Element und somit um die wichtigste Komponente für den Betrachter handelt.

4.3.1.2 Die Schoeps-LooknFeel-Klasse

Um ein einheitliches Design aller Elemente im Sinne des CI von Schoeps herzustellen, wurde auf eine modifizierte LooknFeel-Klasse von Schoeps zurückgegriffen. Die LooknFeel-Klasse ist eine von JUCE bereit gestellte, individuell anpassbare GUI-Gestaltungs-Klasse, die es mittlerweile in der 4. Version gibt (Stand 20.11.2018). Wird das LookAndFeel-Objekt einer Komponente zugewiesen, überschreibt sie das Aussehen selbiger und derer Kindskomponente, wenn diese nicht explizit überschrieben wird. Durch die Entkopplung von Stil-Elementen in das genannte Objekt kann auf schnelle Weise der grafische Skin eines Plugins verändert werden.

4.3.2 Struktur und Pluginaufbau

Bei der Erzeugung eines Audioplugins erstellt JUCE eine AudioProcessor- und eine AudioProcessorEditor-Klasse mit dem jeweiligen Namen des Projekts: ParabolaProcessor und ParabolaEditor. Der Arbeitstitel des Plugins Parab EQ wurde hierbei erst später festgelegt und aus diesem Grund an dieser Stelle noch nicht verwendet.
Der ParabolaProcessor übernimmt das Prozessieren von Audiosignalen und entspricht einer Instanz eines geladenen Audioplugins. Hinsichtlich der verschiedenen Pluginformate wird er von jeder einzelnen Technologie gewrappt[9] und kann somit technologieunabhängig benutzt werden. Der ParabolaEditor wird von der JUCE eigenen Componentenklasse abgeleitet und beinhaltet die GUI[10]-Komponenten und Funktionalitäten. Die nachfolgende Abbildung zeigt eine vereinfachte Darstellung der beiden Klassen:

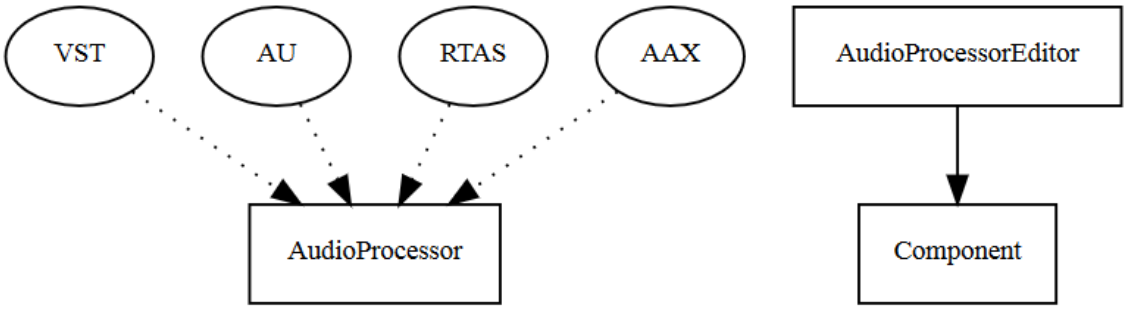

Abbildung 4.5: Schema des Audioplugins von juce.com
Quelle: https://docs.juce.com/master/tutorial_choosing_projucer_template.html, abgerufen am 07.11.2018

Der Aufbau eines Audioplugins hat damit gewisse Ähnlichkeiten zu dem bekannten Modell-Präsentation-Steuerung-Muster[11], welches häufig in der Softwareentwicklung

[9] Englisch für Umhüllung. Fungiert ähnlich einer Schnittstelle zwischen zwei Komponenenten.
[10] Graphical User Interface = Grafische Benutzeroberfäche/-schnittstelle
[11] im Englischen „Model View Controller (MVC)-Pattern“

eingesetzt wird. Das MVC-Muster soll eine möglichst große Trennung von verschiedenen Schichten, in diesem Fall der (Oberflächen-)Präsentation, der Programmsteuerung und des Datenmodells ermöglichen. Dadurch wird eine größere Flexibilität und Wiederverwendbarkeit erreicht, die das Austauschen einer einzelnen Schicht ermöglicht. (Vgl. S. 5-6 [GJHV11]).

Das JUCE-Framework an sich unterstützt dieses Entwurfsmuster nach Aussage von Julian Storer eher nicht, da dieser das MVC-Muster als mühsam und weniger hilfreich zur Erstellung von Audiosoftware empfindet, wie er im JUCE-Forum[12] zum Ausdruck bringt. Aus diesem Grund übernehmen sowohl ParabolaProcessor als auch der ParabolaEditor Aufgaben des Controllers. Dies wird bereits von Elternklassen übernommen und kann auf Grund dessen nur mit größerem Aufwand geändert werden.

Zur Umsetzung der benötigten Filter wurde eine zusätzliche, gleichnamige Klasse implementiert, welche die Filterparameter beinhaltet. Der ParabolaProcessor kann somit verschiedene Instanzen eines Filters erzeugen und verwenden. Die Struktur des Parabola-Projekts kann somit auch als eine abgeänderte Form des MVC-Musters betrachtet werden, in der die Filterklasse die Rolle des Models übernimmt. Dafür müsste die Klasse allerdings weitgehende Funktionalität und die Verarbeitung der Audiodaten übernehmen, was wie bereits erwähnt, einige Anpassungen an vorgegeben Klassen in Anspruch nehmen würde.
Einen Überblick über die drei Klassen und somit das Projekt liefert das nachfolgende Parab EQ-Klassendiagramm, welches nach Spezifikation der Unified Modeling Language (UML) geschrieben wurde. Die Klassen wurden an manchen Stellen gekürzt und vereinfacht, um die Übersichtlichkeit zu gewährleisten und wichtige Funktionen und Variablen hervorzuheben. Fehlende Methoden oder Variablen werden in den Elternklassen, von denen die ParabolaProcessor- und die ParabolaEditor-Klasse erben, mit entsprechenden Verweis „...“ deklariert. Die nachfolgenden Unterkapitel beschreiben die einzelnen Abhängigkeiten, wie auch Sichtbarkeiten, sowie Methoden der verschiedenen Klassen, die im UML-Klassendiagramm sichtbar sind.

[12] https://forum.juce.com/t/general-question-regarding-the-idea-for-juce-model-classes/10955, abgerufen am 08.11.2018

Abbildung 4.6: UML-Klassendiagramm des Parab EQ-Plugins

4.3.2.1 Der ParabolaProcessor

Der ParabolaProcessor erbt von den drei Klassen AudioProcessor, AudioProcessorValueTreeState::Listener und ChangeBroadcaster. Die AudioProcessor-Klasse ist für die Vorbereitung und Verarbeitung von Audiosignalen zuständig. Sie liefert die Grundstruktur für das eigentliche Pluginformat, wie in 4.3.2 bereits erwähnt.
Die AudioProcessor-Klasse vererbt die zwei Methoden processBlock und prepareToPlay an den ParabolaProcessor. Über die Methode prepareToPlay wird die Abtastrate (engl. Sample rate) und die maximale Anzahl an Samples in einer Blockgröße definiert. Dies geschieht einmalig vor der Bearbeitung von Audiodaten. Die Blockgröße wird für die spätere Verarbeitung der einzelnen Blöcke in der processBlock-Methode benötigt. Die Methode wird im späteren Verlauf genauer vorgestellt. Die im folgenden Listing ersichtlichen „static_casts“ werden zur korrekten Kompilierung unter macOS benötigt.

```
void ParabolaAudioProcessor::prepareToPlay(double sampleRate, int samplesPerBlock)
{
        lastSampleRate = sampleRate;
        dsp::ProcessSpec processSpec{
                lastSampleRate, static_cast<uint32>(samplesPerBlock),
                static_cast<uint32>(getTotalNumOutputChannels()) };
        processorChain.get<6>().setGainLinear(1.0f);
        //init filters
        parameterChanged(windCoverParameterID, *windCoverParameter);
        parameterChanged(lowCutParameterID, *lowCutParameter);
        parameterChanged(distanceParameterID, *distanceParameter);
        parameterChanged(aimingParameterID, *aimingInaccuracyParameter);
        parameterChanged(freeFieldParameterID, *freeFieldParameter);
        processorChain.prepare(processSpec);
}
```

Listing 4.3: PrepareToPlay-Methode in der ParabolaProcessor-Klasse

Zu Beginn der Methode wird die zuletzt gesetzte Samplerate mit der neuen Samplerate überschrieben. Danach wird dem ProcessSpec-Objekt diese aktuelle Abtastrate, die Anzahl an Samples pro Block und die maximale Anzahl an Outputkanälen übergeben. Diese Prozessorspezifikation wird der ProcessorChain in ihrer Prepare-Methode übergeben, bevor die eigentliche Filterprozessierung beginnt. Die Funktionsweise des ProcessorChain-Objekts wird mit dem nächsten Codebeispiel erklärt.
Die parameterChanged-Methode wird für jedes einzelne Filter aufgerufen, um jeden verwendeten Parameter zu Beginn zu initialisieren. Dies geschieht zum einen mit einer zuvor definierten ParameterID und einen konkreten Default-Wert, der einer voreingestellten Session entspricht.

Das folgende Listing zeigt die bereits genannte Methode processBlock:

```
void ParabolaAudioProcessor::processBlock(
        AudioBuffer<float>& buffer, MidiBuffer& midiMessages)
{
        dsp::AudioBlock<float> audioBlock(buffer);
        dsp::ProcessContextReplacing<float> context(audioBlock);
        processorChain.process(context);
}
```

Listing 4.4: ProcessBlock-Methode

Diese übernimmt die konkrete Prozessierung der einzelnen Audioblöcke. Dazu wird der per Parameter übergebene Buffer zur Initialisierung eines neuen AudioBlocks verwendet. Mit Hilfe des erstellten AudioBlock kann ein ProcessContextReplacing-Objekt erzeugt werden. Dieses Kontextobjekt impliziert, dass die bearbeiteten Samples inplace bearbeitet werden und keine neuen Samples für den Output generiert werden. Zuletzt bekommt das ProcessorChain-Objekt den neuen Kontext und kann die zuvor eingestellten Filter sukzessive auf den AudioBlock anwenden.
Die Abbildung auf der nächsten Seite zeigt die Verarbeitung von Audioblöcken innerhalb der Anwendung mit Hilfe eines Aktivitätsdiagramms und sogenannten „Schwimmbahnen“[13]. Diese beschreiben die Aufgaben der einzelnen Klassen (vgl. S. 17-18 [Kle18]).
Der Start- und Endpunkt des Aktivitätsdiagramms wird durch die beiden schwarzen Kreise visualisiert. Ein- und Ausgaben werden mittels Parallelogrammen, Verzweigungen mittels Rauten dargestellt. Die einzelnen Prozessschritte finden sich in abgerundeten Rechtecken wieder.

[13] im Englischen „swim lanes“.

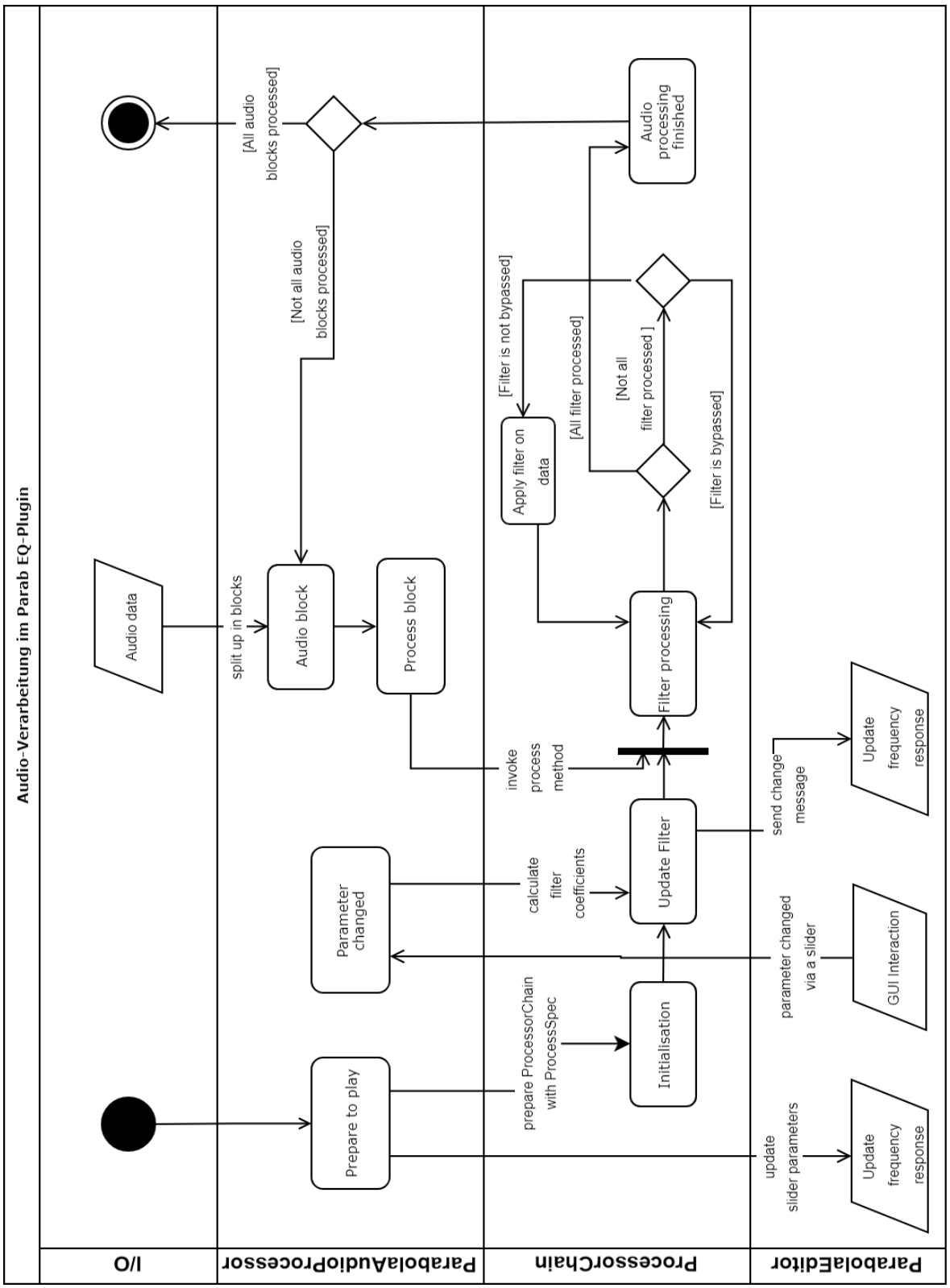

Abbildung 4.7: Aktivitätsdiagramm der Audioverarbeitung im Parab EQ-Plugin
Quelle: Eigene Abbildung

Die Methoden der ChangeBroadcaster-Klasse ermöglicht es allen erbenden Klassen, eine vereinfachte Implementierung des Beobachter-Musters, siehe S. 287 [GJHV11]. Dazu registriert sich jede Klasse, die über Änderungen des Processors informiert werden will, über die addChangeListener-Methode als Listener beim ParabolaProcessor. Der Processor kann dann nach eigenen Kriterien entscheiden, wann er eine ChangeMessage an alle registrierten Listener abschickt. Im konkreten Programm bedeutet dies, dass Veränderungen an den Filterparameter an den ParabolaEditor als Änderungen per „ChangeMessage“ gemeldet werden. Dieser kann daraufhin bestimmte Komponente der Pluginoberfläche aktualisieren. Diese Schnittstellenlösung ermöglicht eine effiziente und lose Kopplung der beiden Klassen, ohne die Funktionalität einzuschränken oder zu spezifisch werden zu lassen.

Die Variable parameters ist eine Instanz der AudioProcessorValueTreeState-Klasse. Diese Klasse beinhaltet einen Wertebaum[14], der den Zustand des AudioProcessors abbildet. Innerhalb des AudioProcessorValueTreeState werden diese Werte in Parameterobjekten gespeichert und können über eine eindeutige ID identifiziert werden. Diese ID wurde bereits in der Methode parameterChanged verwendet, um die einzelnen Werte des ValueTrees auszulesen. Eine weitere Funktionalität ist die Verknüpfung von GUI-Elementen mit Werten innerhalb des ValueTrees. Dazu werden einzelne GUI-Komponente, wie Slider mit Attachments[15] versehen, die Interaktionen des Benutzers mit konkreten Parametern verknüpfen. Dies führt zu einer stetigen Aktualisierung der Werte.
Mittels des parameters-Objekt kann der Plugin-Zustand auch über die Lebensdauer des Plugins hinweg gespeichert und wiederhergestellt werden. Dazu wird vor dem Beenden des Plugins der Zustand des parameters-Objekt in einen XML-Objekt konvertiert und anschließend binär gespeichert. Dies Vorgang findet in der getStateInformation-Methode statt:

[14] im Englischen „ValueTree“
[15] Englisch für Anhänge.

```
void ParabolaAudioProcessor::getStateInformation(MemoryBlock& destData)
{
        auto state = parameters.copyState();
        std::unique_ptr<XmlElement> xml(state.createXml());
        copyXmlToBinary(*xml, destData);
}

void ParabolaAudioProcessor::setStateInformation(const void* data, int sizeInBytes)
{
        std::unique_ptr<XmlElement> xmlState(getXmlFromBinary(data, sizeInBytes));
        if (xmlState.get() != nullptr && xmlState
                ->hasTagName(parameters.state.getType()))
        parameters.replaceState(ValueTree::fromXml(*xmlState));
}
```

Listing 4.5: Methoden zur Speicherung des Pluginszustands

Beim Starten der Anwendung wird die setStateInformation-Methode aufgerufen und das parameters-Objekt mit den im XML-Objekt gespeicherten Informationen wiederhergestellt.
Die nachher in der Anwendung zur Verfügung gestellten Presets erhalten mittels des StringArray programNames ihren in der Presetauswahl der DAW ersichtlichen Namen. Die Methode setCurrentProgram mappt die Auswahl des Benutzers in der DAW auf einen der sechs vorhanden Presets. Dadurch wird die in der Analyse 3.3.5 definierte Anforderung erfüllt.

4.3.2.2 Der ParabolaEditor

Der ParabolaEditor übernimmt wie eingangs erwähnt die komplette Gestaltung und Aktualisierung der Pluginoberfläche. Dazu werden im Konstruktor des ParabolaEditor alle GUI-Elemente initialisiert beziehungsweise sichtbar gemacht, Schriftarten eingebunden und Farben definiert. Um Veränderungen bezüglich Anordnungen und Positionen einzelner Elemente zu erleichtern, wurde hierbei X- und Y-Positionen mit zuvor definierten Offset kombiniert, die bei Bedarf zentral für alle Elemente angepasst werden können.

Durch die bereits im vorherigen Kapitel 4.3.2.1 erwähnte Registrierung des ParabolaEditors beim ParabolaProcessor als ChangeListener, wird der Editor bei Filteränderungen informiert. Wird ein ChangeBroadcast versendet, kann der ParabolaEditor mittels der changeListenerCallback-Methode auf die neuen Ereignisse reagieren. Über den „source“-Parameter der Methode kann hierbei ermittelt werden, um welches Objekt es sich explizit handelt. Da es sich bei Filteränderungen immer um eine den gesamten Frequenzgang betreffende Änderung handelt, wird der gesamte Plot zurück-

gesetzt und neu gezeichnet. Die genannte Methode wird vom ChangeListener geerbt.

Für jede Komponente, welche einem Filterparameter entspricht, wird wie bereits im vorherigen Kapitel angesprochen, ein zugehöriges Attachment erzeugt. Dieses ist für die Kopplung von Parameterwert und GUI-Komponente verantwortlich und sorgt für die Übertragung von Presetwerten auf die entsprechenden GUI-Komponenten.

4.3.2.3 Die Filterklasse

Die Filterklasse enthält alle für die spätere Filterkoeffizienten-Berechnung erforderlichen Parameter. Dazu gehören neben Verstärkung (Gain), der Gütefaktor (Q-Faktor), die (Grenz-)Frequenz, der Bypass-Parameter und die Aussteuerungen (Magnitudes). Die zuletzt genannten können für die Filterplotdarstellung genutzt werden, in dem sie mit dem jeweiligen Faktor der db-Skalierung multipliziert werden.

4.3.3 Entwicklung der digitalen Filter

Im folgenden Verlauf wird die Entwicklung der Filter exemplarisch an der Implementierung des Hochpassfilters verdeutlicht und die Methode vorgestellt, welche die Filteraktualisierungen übernimmt.
Wie bereits zu Beginn des Kapitels erwähnt, wurden für das Parab EQ-Plugin sechs IIR-Filter verwendet. IIR-Filter wurden aus Gründen der Effizienz und besseren Echtzeitfähigkeit ausgewählt. Zwei der sechs IIR-Filter wurden als Hochpassfilter zur Reduzierung der Windgeräusche und Griffgeräusche eingesetzt. Drei Glockenfilter für den Tiefen- und tiefe Mitten-Bereich im Bezug auf die Distanzanpassungen und Windschutzkompensierungen und ein HighShelf-Filter zur Verstärkung verlorener hochfrequenter Anteile durch Dissipation und Objektentfernung.
Die bereits zur Initialisierung der Filter verwendete Methode parameterChanged wird von der AudioProcessorValueTreeState::Listener-Klasse an den AudioProcessor vererbt. Als Methodenparameter werden der Methode die ID des sich verändernden Parameters und der neue Wert des Parameters übergeben. Innerhalb des Methodenrumpfs wird in Abhängigkeit zur hereingebenen ID die dazugehörige Neuberechnung von Filterkoeffizienten und Übertragungsfunktion übernommen.

Folgendes Listing zeigt die Implementierung der Berechnung der Filterkoeffizienten für den verwendeten Hochpass mit Hilfe des DSP-Moduls von JUCE:

```
if (parameter == lowCutParameterID)
{
if (newValue == 1.0f) {
processorChain.setBypassed<0>(false);
processorChain.setBypassed<1>(false);

auto coefficientsArray =
dsp::FilterDesign<float>::designIIRHighpassHighOrderButterworthMethod(...);
*processorChain.get<0>().coefficients = coefficientsArray[0];
processorChain.get<0>().coefficients->getMagnitudeForFrequencyArray(...);
// Put another highPass in series
*processorChain.get<1>().coefficients = coefficientsArray[1];
processorChain.get<1>().coefficients->getMagnitudeForFrequencyArray(...);
}
else
{
...
}
calcHighShelf();
}
```

Listing 4.6: parameterChanged-Methode

Für den in 3.3.2 angesprochenen Hochpass wurden wie beschrieben zwei Butterworthfilter 2. Ordnung in Serie geschaltet. Dies wurde im Quellcode durch einen Kommentar verdeutlicht. Zu Beginn der Filterimplementierung wird mit Hilfe der JUCE-Bibliothek eine Koeffizientenreihe berechnet, die für die beiden Filter verwendet werden kann. Die Koeffizienten werden danach auf den ersten Positionen im bereits erwähnten ProcessorChain eingefügt und die Aussteuerungen für die Koeffizienten neu berechnet.
Am Ende des Beispiels wird die Methode calcHighShelf aufgerufen, welche die Neuberechnung des verwendeten HighShelf-Filters handhabt. Sie wurde ausgelagert, um diverse Abhängigkeiten von Parametern auf die hochfrequenten Anteile zentral zu handhaben. Die Implementierungsweise ist hierbei analog zu dem des Hochpasses und wird deshalb nicht explizit ausgeführt.

4.3.3.1 Parameterwerte der Filter

Die für die Filter verwendeten Werte wurden teilweise aus vorherigen Testungen von Seitens der Firma Schoeps übernommen oder entsprechend angepasst. Folgende Tabelle fasst die Filterwerte für die verwendeten Filter zusammen:

Bassanhebung	Gain-Faktor	dB-Verstärkung	Min/Max	Distanz	Zielungenauigkeit	Diffus-/Freifeld	Windschutz
	∼2,37	7,5 dB	-	0%	0%	0%	-
	1	0 dB	Min	100%	0%	0%	-
	∼4,75	13,53 dB	Max	0%	0%	100%	-
	∼2,37	7,5 dB	-	100%	0%	100%	-

Höhenanhebung	Gain-Faktor	dB-Verstärkung	Min/Max	Distanz	Zielungenauigkeit	Diffus-/Freifeld	Windschutz
	1,25	∼1,94 dB	Min	0%	0%	0%	-
	1,75	∼4,86 dB	-	100%	0%	0%	-
	2	∼6 dB	-	0%	100%	0%	-
	2,5	∼8 dB	-	100%	100%	0%	-
	2,56	∼8,16 dB	-	100%	100%	0%	HWC
	4,5	∼13 dB	Max	100%	100%	0%	Windjammer

Die Funktionalität der in der Tabelle ersichtlichen Parameter wurden bereits in Kapitel 3.3.3 festgehalten und beschrieben. Die Parametertabelle zeigt für fast jede veränderbare GUI-Komponente den korrespondierenden Gain-Faktor beziehungsweise die dB-Verstärkung an. Davon ausgenommen sind die Mikrofonauswahl und der Hochpass, da erstere zum Zeitpunkt der Thesis noch keine weitere Auswahl besitzt und der Hochpass bewusst keinen Einfluss auf die Bassanhebung hat.
Zusätzlich werden diejenigen Einstellungen markiert, welche für eine maximale beziehungsweise minimale Verstärkung oder Abdämpfung in einem Frequenzbereich sorgen. Daran ergänzend werden auch die Einflüsse des jeweiligen Windschutzes in der letzten Spalte der Tabelle berücksichtigt. Mit Hilfe der Tabelle können einzelne Anpassungen am Gain-Faktor der Filter festgehalten und die Grenzfälle berechnet werden. Basierend auf diesen Werten wurde die dB-Skalierung des Frequenzgangplots auf 18 dB Vollaussteuerung mit 6 dB Abständen festgelegt, siehe 4.4.

4.4 Umsetzung der plattformübergreifenden Funktionalität

Für die Umsetzung der plattformübergreifenden Funktionalität wurde das Plugin sowohl auf Windows 8.1 in Visual Studio 2015, wie auch auf macOS Sierra in Xcode 9.2 und Linux Mint in Code::Blocks importiert und kompiliert. Dies wurde durch die Erstellung spezifischer Exporter für die jeweilige Plattform innerhalb des Projucers vollzogen, siehe 2.3.4.1. Durch die Trennung von Quellcode in den „Source“-Ordner und plattformspezifischer Buildumgebungen in den „Builds‘’-Ordner ist es jederzeit möglich, neue Exporter hinzuzufügen oder zu entfernen. Beim Hinzufügen eines neuen Export-Ziels erzeugt der Projucer eine entsprechende „build configuration“, die zur Kompilierung und Erstellung eines Plugins in der IDE benutzt wird. (Vgl. [juc18]).
Möchte der Anwender mehrere Versionen eines Pluginformats, wie VST2.4 und VST 3 unterstützen, ist dies ebenfalls möglich. Durch die Auswahl des jeweiligen Pluginformats in den Projekteinstellungen des Projucers werden separate „build configuration“ für die jeweiligen Versionen erstellt. Diese beinhalten die bereits in Abbildung 4.5 ersichtlichen, individuell auf die Pluginformate angepassten Pluginwrapper.

Die folgende Abbildung zeigt die Übersicht aller Exporter und die Auswahl der Pluginformate im Projekt:

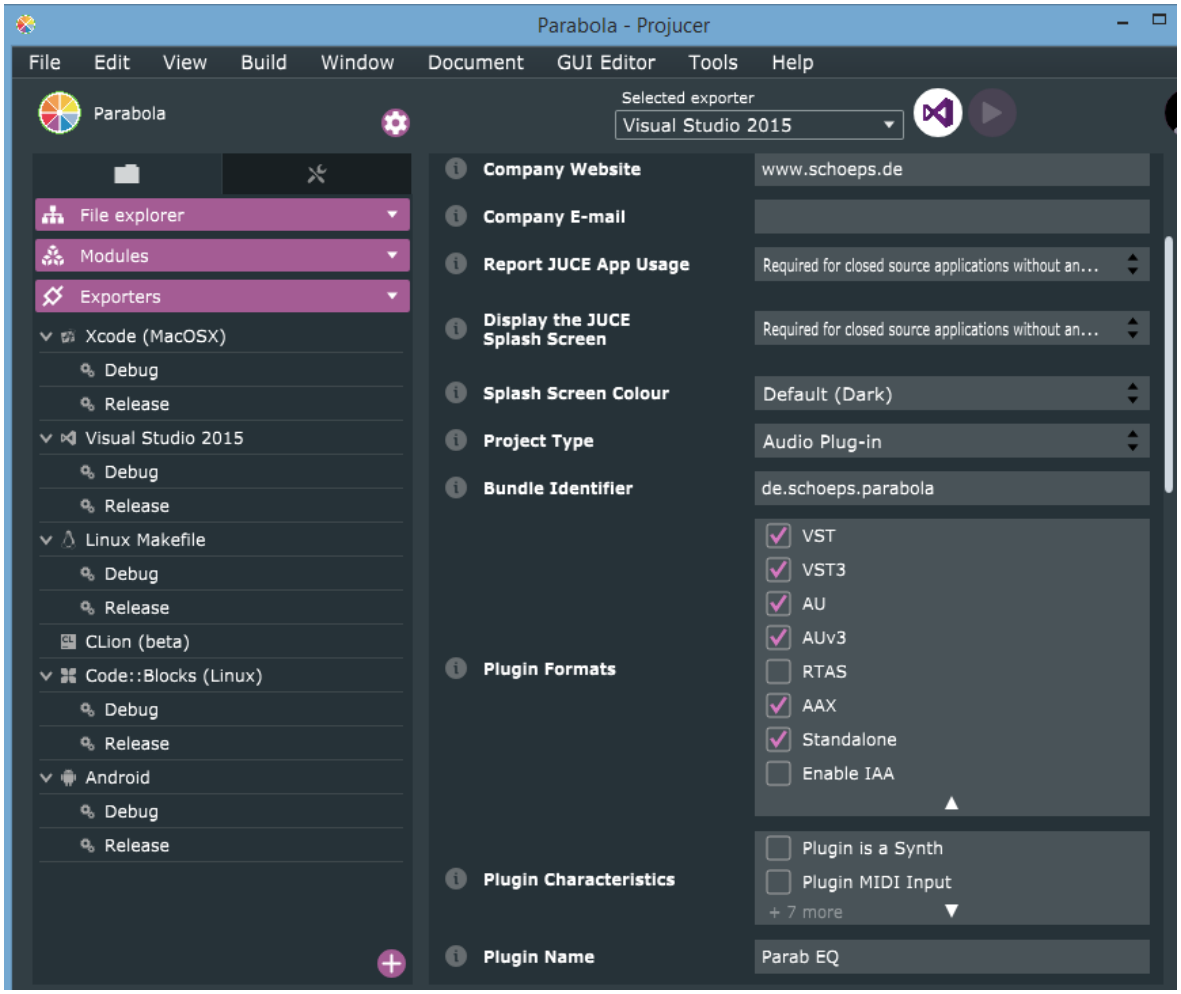

Abbildung 4.8: Exporter-Übersicht im Projucer
Quelle: Eigene Abbildung

Es wurden folgende Pluginformate/-programme exportiert: AudioUnit v2/v3, VST 2.4/3, AAX und eine host-unabhängige Standalone-Version auf Windows, macOS und Linux. Somit konnten alle in 3.3.1 geforderten Pluginformate bedient werden.
Alle Versionen waren hierbei lauf- und funktionsfähig, wenn auch die Linuxversion eine nicht angepasste Schriftart benutzte. Da keine in den Anforderungen untersuchte DAW eine Linuxkompatiblität erfordert, wurde hierfür keine weitere Arbeit in die korrekte Darstellung investiert. Eine Exportierung für Smartphones mit Android-Betriebsystem wäre theoretisch ebenfalls möglich, da JUCE zur Exportierung das Android-NDK[16] beziehungsweise SDK verwenden kann. Dies war im Rahmen der Thesis allerdings nicht gefordert.

[16]Das NDK-Toolset ermöglicht die App-Implementierung mittels in C/C++ geschriebener Klassen.

Für die Exportierung auf macOS wurden für einzelne Codezeilen Anpassungen gemacht, die für die Kompilierung notwendig sind. Eine Änderung wurde bereits in Kapitel 4.3.2.1 angesprochen und betrifft die Typkonvertierung von übergebenen Argumenten in Konstruktoren und Methoden. Dies war ebenfalls für die Maßeinheiten des Frequenzgangplots notwendig und wurde deswegen im Quellcode vermerkt.

5. Evaluation

Das folgende Kapitel beschäftigt sich mit der Auswertung und Validierung der erfolgten Testungen des Audioplugins. Zu Beginn wurde die Grundfunktionalität des Plugins entwickelt und in Zusammenarbeit mit der Firma Schoeps in einer ersten Testung evaluiert und verbessert. Nach dieser ersten Testphase wurde eine externe Testperson eingebunden, die weiteres Feedback zur Oberflächenstruktur und Usability gab. Die daraus resultierenden Erkenntnisse wurden implementiert und in einer zweiten Testung bei Schoeps mit weiteren Testpersonen evaluiert.

5.1 Auswertung der ersten Testergebnisse

Bei der ersten Testung in Kooperation mit Schoeps wurde eine Inkonsistenz bei den Filterparametern festgestellt. Dieser bezog sich auf den Höhenausgleich bei unzureichender Fokussierung, bzw. Zielungenauigkeit der Schallquelle und den damit verbundenen Höhenverlust.
Zu Beginn wurde dieser Parameter noch als „Aiming Accuracy“ betitelt und besaß bei einer Einstellung von 100% keinen Einfluss auf die Filter innerhalb des Plugins:

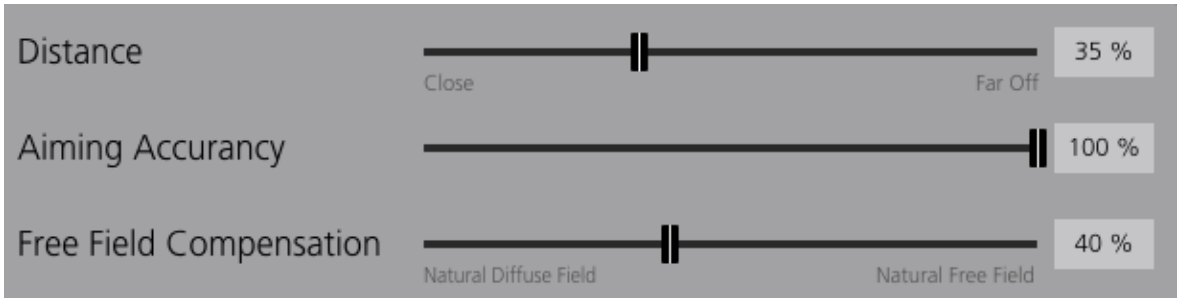

Abbildung 5.1: Alte Parameterbezeichnungen
Quelle: Eigene Abbildung

Damit wurde das sogenannte Prinzip des gemeinsamen Schicksals, bzw. das Prinzip der gemeinsamen Bewegung verletzt (vgl. S. 104 [Gol14]). Grund dafür ist, dass dieser Parameter die gleiche Bewegungsrichtung wie die beiden übrigen Parameter enthielt, allerdings eine gegenteilige Auswirkung, bzw. Abdämpfung der Signalanteile innerhalb des Plugins verursachte. Aus diesem Grund wurde die Bezeichnung auf „Aiming Inac-

curacy“ geändert und die Funktionalität invertiert. Dies bedeutet bei 100% „Aiming Inaccuracy“ eine maximale Höhenkompensierung.

Eine weitere Änderung bezog sich auf den dritten Parameter hier „Free Field Compensation“. Die in Abbildung 3.1 erwähnte geringe Pegeldifferenz im Tiefenbereich zwischen Frei-/Diffusfeld sorgt dafür, dass eine Verstärkung der tiefen Frequenzen nicht ohne eine Anhebung der diffusen Signalanteile erfolgen kann. Aus diesem Grund sollte keine Tiefenverstärkung durchgeführt werden, wenn der Slider vermehrt in Richtung Diffusfeld eingestellt wird. Dies wurde bei der Implementierung der Sliderfunktionalität zuerst nicht beachtet und somit nachgezogen.

Weitere Änderungen betrafen die Oberflächengestaltung. Für jeden Slider wurde ein Tooltip[1] eingeführt, um die eigentliche Funktionalität bei Bedarf genauer zu erklären, wie in der nachfolgenden Abbildung ersichtlich ist:

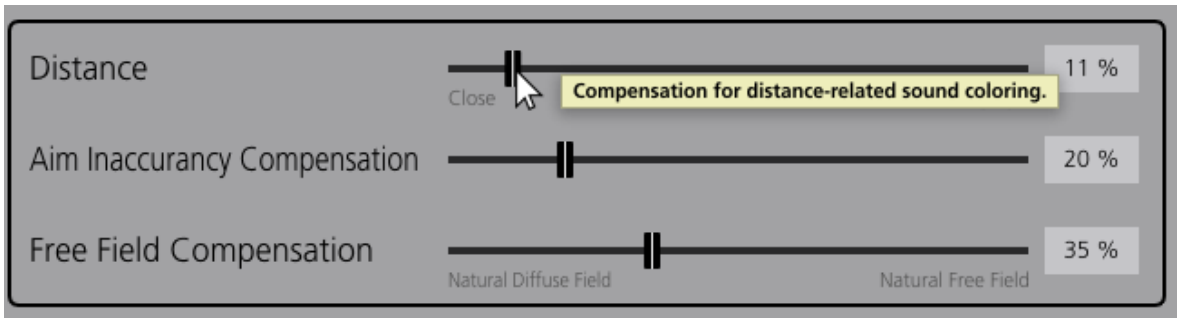

Abbildung 5.2: Tooltip für den Distanz-Parameterslider
Quelle: Eigene Abbildung

5.2 Auswertung der zweiten Testergebnisse

Vor der zweiten Testrunde bei Schoeps wurde das Plugin von einer unbeteiligten dritten Person auf Usability getestet. Hierbei wurde deutlich, dass die Testperson nicht intuitiv erfassen konnte, ob die Parameter für die Nachstellung der ursprünglichen Aufnahmesituation oder für die Einstellung des Klangbildes gedacht sind. Um zu verdeutlichen, dass der Anwender die Parameter zur Abbildung der Aufnahmesituation verwenden soll, wurden neue Texte für die Tooltips geschrieben und zusätzliche Informationsbuttons vor jeden Slider implementiert. Diese Buttons übernahmen die alten Tooltip-Texte. Zudem wurden für die zwei Einstellungsbereiche Überschriften eingeführt, welche die Aufgaben der beinhalteten Parameter beschreibt:

[1]Hilfetext einer Komponente, der beim Pausieren der Maus erscheint.

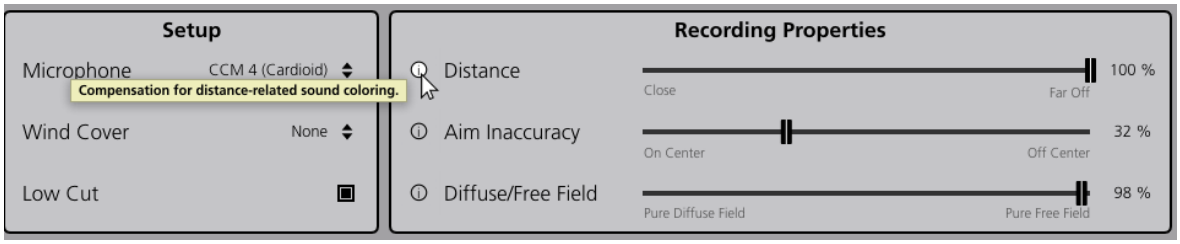

Abbildung 5.3: Info-Buttons und Überschriften der Einstellungsbereiche
Quelle: Eigene Abbildung

Des Weiteren wurden die Namen für die Parameter überarbeitet und das Wort „Compensation“ entfernt. Grund dafür ist, dass das Wort keinen Mehrwert hinsichtlich der Verständlichkeit besitzt und ebenso die Funktionalität nicht treffend beschreibt. Aus diesem Grund wurde der letzte Parameter in „Diffuse-/Free Field“ umbenannt. Für den „Aim Inaccuracy“-Slider wurden zudem Hilfstexte unterhalb des Sliders angebracht, um die Extremwerte und die Funktionalität zu beschreiben.

Ebenfalls angepasst wurden die Maximalwerte der Filter und die dadurch bedingte größere Skalierung des Frequenzgangsplot im Plugin. Dies geschah unter dem Aspekt, dem Nutzer mehr Freiheiten in Bezug auf die individuelle Klangbearbeitung zu geben.

5.3 Auswertung der Testsessions

Zur Testung der Pluginfunktionalität wurde sowohl bei Schoeps eine Testsession in Nuendo 8, wie auch aus eigenen Testzwecken eine Testsession in Logic Pro X erstellt. In diesen Testsessions wurden Kanäle mit Testaufnahmen angelegt, die jeweils einen möglichen Anwendungsfall darstellen. Die Aufnahmen stammen aus zwei Messungen, welche Schoeps vor Beginn der Thesis erstellt hat. Diese unterscheiden sich in der Aufnahmedistanz von 12 m oder 30 m und beim Aufnahmeobjekt, wie beispielsweise Sprecher, Wasserfall, Hund und Vogelgeräusche. Mit Hilfe der Messbeispiele konnten die in der Analyse erwähnten Presets erstellt und getestet werden, siehe 3.3.5. Folgende Abbildung zeigt die Testsession in Logic Pro X unter Verwendung des Parab EQ-Plugins. Im Plugin wurde passend zur Aufnahme das Preset Singbird (Close) ausgewählt:

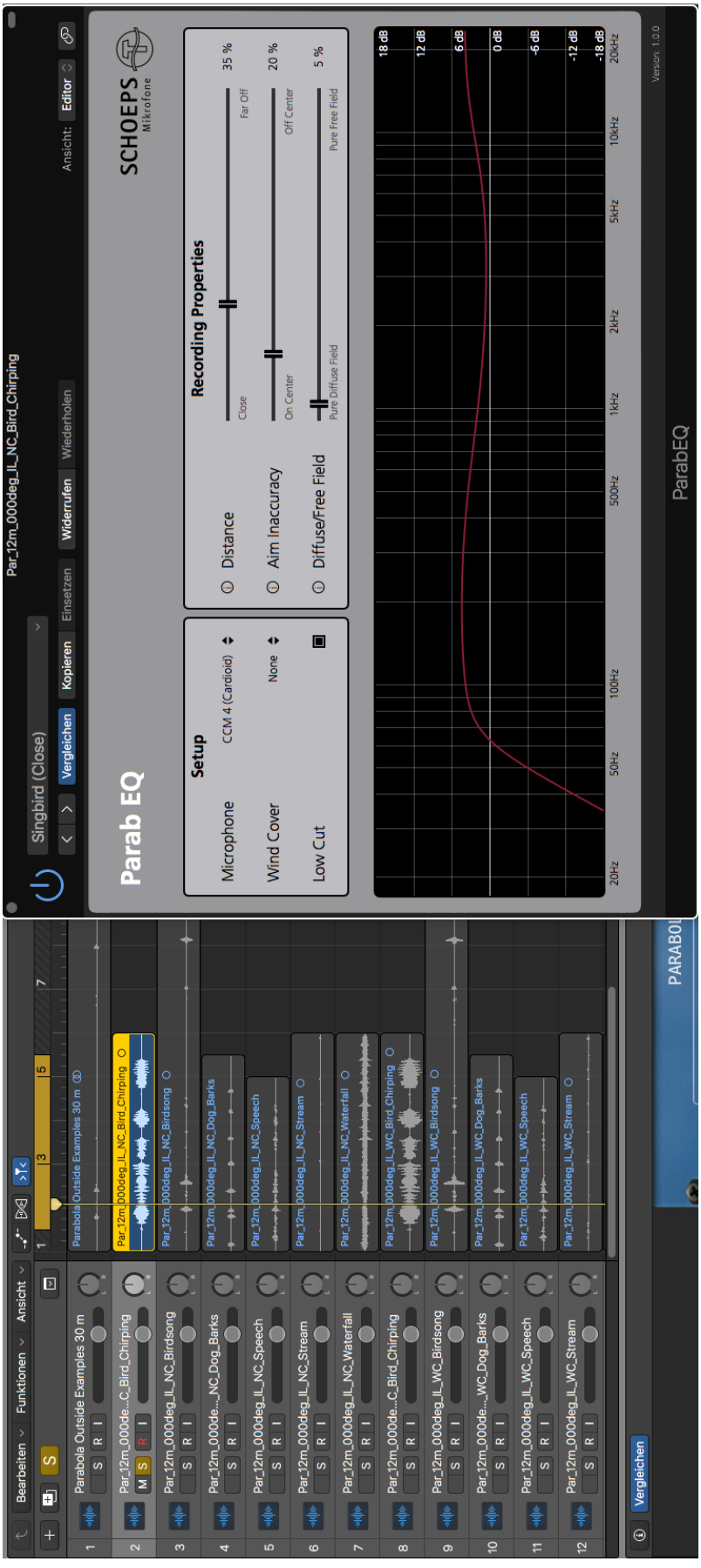

Abbildung 5.4: Ausschnitt der Testsession in Logic Pro X
Quelle: Eigene Abbildung

6. Zusammenfassung und Ausblick

Im abschießenden Kapitel wird das zu Beginn der Thesis definierte Ziel an den erreichten Ergebnissen gemessen. Es werden Rückschlüsse bezüglich der Verwendbarkeit des Plugins gezogen und Vorteile der digitalen Postproduktion besprochen. Der Abschnitt „Ausblick“ bildet den Abschluss der Thesis und stellt zukünftige Möglichkeiten der Weiterentwicklung, sowie weitere optionale Anwendungsgebiete vor.

6.1 Fazit

Die in den Grundlagen beschriebene geschichtliche Entwicklung bietet einen Überblick über die technische und historische Entwicklung eines Parabolmikrofons und die damit verbundenen akustischen Vor- und Nachteile bezüglich Parabolmikrofon-Aufnahmen.

Mittels der Analyse des Parabolasystems der Firma Schoeps und den daraus generierten Anforderungen wurde ein plattformübergreifendes Audioplugin für Windows beziehungsweise macOS konzipiert und realisiert.
Dazu wurde das JUCE-Framework verwendet, welches es ermöglicht, auch für kommende Technologien oder zukünftige Pluginformate neue Exporter zu erstellen und zu implementieren. Bei der Oberflächengestaltung wurden verschiedene Gestaltprinzipien eingehalten, welche die Usability und Einarbeitungszeit verkürzen und mit zusätzlichen Hilfestellungen, wie Tooltips und Info-Buttons die Handhabung für fachfremde Anwender erleichtern. Das Audioplugin wurde in mehreren Testsessions auf seine Gebrauchstauglichkeit getestet und mittels der gewonnenen Erkenntnisse verbessert. Darüber hinaus wurden verschiedene Presets implementiert, die eine Nachbearbeitung vereinfachen und beschleunigen. Durch die Einhaltung der Coding Standards von Julian Storer und die Ideen des Clean Codings wurde zudem Wert auf die Wartbarkeit, Lesbarkeit und Robustheit des geschriebenen Quellcodes gelegt. Die zu Beginn gestellten Anforderungen an die Masterthesis konnten somit vollständig umgesetzt und eingehalten werden.

6.2 Ausblick

In diesem Abschnitt werden Erweiterungen und technische Features vorgestellt, die in zukünftige Entwicklungen miteinbezogen werden können und die Arbeits-/Funktionsweise der erstellten Anwendung erweitern.

Durch die Weiterentwicklung und Zunahme von leistungsfähigen Tablets ist es möglich, Audioplugins für den Live-Einsatz in mobile Ableger bekannter DAWs, wie Cubasis 2 von Steinberg, FL Studio Mobile 3 von Image Line oder GarageBand von Apple zu exportieren. Dadurch könnte der Anwender bereits während der Aufnahme, Teile der Postproduktion durchführen, was insbesondere bei Live-Übertragungen ohne Übertragungswagen wichtig sein könnte. Dies wurde im Rahmen der Thesis nicht getestet.

Durch die Auswahlmöglichkeit des Aufnahmemikrofons kann der Anwender die Postproduktion weiter seinen Aufnahmekriterien anpassen und die Arbeitsweise, insbesondere Filterfunktionen des Plugins verfeinern. Da die Auswahlmöglichkeit eines anderen Mikrofons, wie dem CCM 21, bereits implementiert wurde, müssten nur geringfügige Änderungen an der Filterfunktionsweise nachgezogen werden.

Unter Berücksichtigung der Entwicklung des NGA[1]-Formats MPEG-H Audio könnten Parabolmikrofone auch weiterhin an Bedeutung gewinnen. Der vom Fraunhofer IIS mitentwickelte Audio-Codec umfasst die Codierung und Decodierung von interaktiven und personalisiertem (3D-)Klang für den Fernseher und Virtual Reality[2]. Durch die Möglichkeit der personalisierten Klangeinstellung können auch viele einzelne Signale, wie Geräusche auf einem Spielfeld, dem Anwender zur Verfügung gestellt werden. Hierbei könnte vermehrt auf die Funktionalität eines Parabolmikrofon zurückgegriffen werden.

[1] **N**ext **G**eneration **A**udio.

[2] siehe https://iis.fraunhofer.de/de/ff/amm/prod/digirundfunk/digirundf/tvaudio.html

Quellenverzeichnis

[App] Apple, *Audio unit documentation*, https://developer.apple.com/documentation/audiounit#overview, aufgerufen am 08.08.2018.

[Bac02] Juha Backman, *Parabolic reflector microphones*, Audio Engineering Society Convention 112, Apr 2002, aufgerufen am 02.08.2018.

[Cir] CircuitsToday.com, *Active filter types*, https://circuitstoday.com/active-filter-types, aufgerufen am 06.11.2018.

[DDHW13] M. Dickreiter, V. Dittel, W. Hoeg, and M. Wöhr, *Handbuch der Tonstudiotechnik*, 8 ed., De Gruyter Saur, 2013, aufgerufen am 08.08.2018.

[Gö07] T. Görne, *Mikrofone in Theorie und Praxis*, 8 ed., Elektor-Verlag, 2007.

[GC10] J. Graham-Cumming, *Der Geek-Atlas: 128 Orte auf der Welt, um Wissenschaft und Technik zu erleben*, 1 ed., O'Reilly Verlag, 2010, aufgerufen am 24.07.2018.

[Gen10] K. Genuit, *Sound-Engineering im Automobilbereich: Methoden zur Messung und Auswertung von Geräuschen und Schwingungen*, Springer Berlin Heidelberg, 2010, 29.07.2018.

[GJHV11] E. Gamma, R. Johnson, R. Helm, and J. Vlissides, *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, Programmer's choice, Pearson Deutschland, 2011.

[Gol14] E. Bruce Goldstein, *Wahrnehmungspsychologie: Der Grundkurs*, Springer Berlin Heidelberg, 2014.

[Har08] M. Hartmann, *Usability Untersuchung Eines Internetauftrittes Nach Din En Iso 9241*, Diplomica-Verlag, 2008.

[Hel05] M. Helfer, *General aspects of vehicle aeroacoustics*, General Aspects of Vehicle Aeroacoustics, 06 2005.

[HT03] A. Hunt and D. Thomas, *Der Pragmatische Programmierer*, b-Agile, Hanser, 2003.

[Juc] Juce.com, *Coding Standards - Latest JUCE coding standards*, aufgerufen am 06.11.2018.

[juc18] juce.com, *Tutorial: Projucer Part 2: Manage your Projucer projects* , https://docs.juce.com/master/tutorial_manage_projucer_project.html, 2018, aufgerufen am 23.11.2018.

[Kar13] U. Karrenberg, *Signale - Prozesse - Systeme: Eine multimediale und interaktive Einführung in die Signalverarbeitung*, Springer Berlin Heidelberg, 2013, aufgerufen am 07.08.2018.

[Kle18] Stephan Kleuker, *Grundkurs software-engineering mit uml: Der pragmatische weg zu erfolgreichen softwareprojekten*, Springer Fachmedien Wiesbaden, 2018.

[May80] Alfred Marshall Mayer, *Topophone letter patent*, https://patentimages.storage.googleapis.com/71/3f/93/102be20ae729c9/US224199.pdf, Februar 1880, aufgerufen am 05.07.2018.

[Neu18] Neumann, *Was ist Eigenrauschen (Ersatzgeräuschpegel)?*, https://neumann.com/homestudio/de/was-ist-eigenrauschen-bzw-ersatzgeraeuschpegel, 2018, aufgerufen am 15.10.2018.

[Oet] Bernd Oettinger, *Marktanteile von DAW-Software*, www.songwriter24.de/blog/studie-beste-beliebteste-daw-ergebnisse/, aufgerufen am 22.10.2018.

[Par18] Edward Parrish, *How To Document and Organize Your C++ Code*, http://www.edparrish.net/common/cppdoc.html, 2018, aufgerufen am 19.11.2018.

[Sch16] Schoeps, *Schoeps Bedienungsanleitung CCM*, 2016, aufgerufen am 25.07.2018.

[Sch18a] ———, *Übertragungsbereich*, https://schoeps.de/wissen/knowledge-base/technische-grundlagen/uebertragungsbereich.html, 2018, aufgerufen am 26.07.2018.

[Sch18b] ———, *Beschreibung des CCM 4*, https://schoeps.de/produkte/ccm/ccm-mikrofone/nieren/ccm-4.html, 2018, aufgerufen am 26.07.2018.

[Sen14] Eberhard Sengpiel, *Mikrofon-Empfindlichkeit (sensitivity)*, http://www.sengpielaudio.com/Rechner-sensitivity.htm, 2014, aufgerufen am 19.11.2018.

[Stea] Steinberg, *About VST Plug-ins in general*, https://github.com/steinbergmedia/vst3sdk, aufgerufen am 06.08.2018.

[Steb] ———, *VST: Standard zur Integration virtueller Instrumente und Effektgeräte*, https://www.steinberg.net/de/company/technologies.html, aufgerufen am 06.08.2018.

[Tec16] Avid Technology, *Audio-Plug-Ins-Handbuch*, 2016, aufgerufen am 08.08.2018.

[Tel] Telinga, *Hi-Wind Cover*, www.telinga.com/products/accessories/hi-wind-cover, aufgerufen am 15.10.2018.

[Wei09] S. Weinzierl, *Handbuch der Audiotechnik*, VDI-Buch, Springer Berlin Heidelberg, 2009.

[Wut86] Jörg Wuttke, *10 Betriebsverhältnisse bei Wind und Popp*, www.ingwu.de/mikrofontechnik/mikrofonaufsaetze/24-10-betriebsverhaeltnisse-bei-wind-und-popp.html, 1986, aufgerufen am 23.10.2018.

[ZLR05] A. Zeibig, M. Lippmann, and D. Richter, *Akustisches Hohlspiegelmesssystem mit Mehrmikrofonanordnung*, Akustisches Hohlspiegelmesssystem mit Mehrmikrofonanordnung, 2005, abgerufen am 27.07.2018.

Anhang

Der Anhang enthält eine Dokumenten-CD, die alle während der Thesis erstellten Programme oder Dokumente enthält. Folgende Auflistung gibt einen Überblick über die enthaltenen Dateien.

CD-Inhalt:

- VST 2.4/3.0-Plugin (Windows/macOS)
- AU v2/v3-Plugin (macOS)
- AAX-Plugin (Windows/macOS)
- Standalone-Programm (Windows/macOS/Linux)
- Quellcode des Parabola-Projekts
- Gespeicherte Internetquellen
- Thesisdokument