

Hochschule der Medien Stuttgart
Fakultät Electronic Media
Studiengang Audiovisuelle Medien

Untersuchung von Lösungsansätzen zur dynamischen
Anpassung der Verzögerungszeit von Audiosignalen
für die Wiedergabe zeitvarianter Schallfelder

Masterthesis

Zur Erlangung des akademischen Grads eines Master of Engineering

vorgelegt von

Sören Hauser

Matrikelnummer: 33735
Trautäckerstraße 23, 70567 Stuttgart
soerenhauser@gmail.com

08.10.2018

Erstprüfer: Prof. Oliver Curdt (Hochschule der Medien)
Zweitprüferin: Dr. rer. Nat. Elena Shabalina (d&b audiotechnik GmbH)
Praxisbetreuer: Felix Einsiedel (d&b audiotechnik GmbH)

Eigenständigkeitserklärung

Hiermit versichere ich, Sören Hauser, ehrenwörtlich, dass ich die vorliegende Masterarbeit mit dem Titel „Untersuchung von Lösungsansätzen zur dynamischen Anpassung der Verzögerungszeit von Audiosignalen für die Wiedergabe zeitvarianter Schallfelder“ selbständig 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 (§ 23 Abs. 2 Master-SPO der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Stuttgart, 08.10.2018

Sören Hauser Matrikelnummer: 33735

Sperrvermerk

Diese Masterarbeit enthält vertrauliche Informationen. Sie darf nur vom Erst- und Zweitprüfer sowie berechtigten Mitgliedern des Prüfungsausschusses eingesehen werden. Eine Vervielfältigung und Veröffentlichung der Arbeit – auch in Auszügen – ist nicht gestattet. Dritten darf diese Masterarbeit nur mit der ausdrücklichen Genehmigung des Verfassers und der d&b audiotechnik GmbH zugänglich gemacht werden.

Die Masterarbeit, exklusive des Titelblattes und Abstracts, ist für die Dauer von zwei Jahren, beginnend ab dem Abgabedatum, für eine Veröffentlichung gesperrt.

Ende des Sperrvermerks: **8. Oktober 2020**

Kurzbeschreibung

Diese Arbeit behandelt verschiedene Algorithmen zur dynamischen Anpassung der Verzögerung von Lautsprechersignalen während der Wiedergabe. Dies ist erforderlich um die Position von virtuellen Klangquellen in einem Mehrkanalsystem zur Schallfeldsynthese zu ändern. Bei der Betrachtung liegt der Fokus auf den akustischen Artefakten, die durch die Algorithmen hervorgerufen werden.

Hierfür werden zunächst physikalische und psychoakustische Grundlagen zur akustischen Lokalisation, dem Maskierungseffekt, dem Kammfiltereffekt und dem akustischen Dopplereffekt zusammengefasst. Daneben wird grundlegendes Basiswissen der Entwicklungsumgebungen *Max 7* und *MATLAB R2018a* dargelegt, mit denen die Algorithmen implementiert wurden.

Es werden drei Algorithmen vorgestellt, die auf Überblendung, Tonhöhenverschiebung und Interpolation beruhen. Dafür werden jeweils zunächst allgemein die Funktionsweise, eine Implementierung mit *Max 7*, die durch diesen Ansatz hervorgerufenen Artefakte und schließlich Möglichkeiten zur Optimierung erläutert. Daneben werden Kombinationen der Algorithmen betrachtet.

Dem folgt die Beschreibung eines *MATLAB* Prototyps der *d&b Soundscape EnScene* Rendering Software, in den zwei der Algorithmen eingebaut wurden.

Schließlich wird ein Hörversuch erläutert, für dessen Durchführung und die Erfassung der Daten ein weiteres Programm in *Max 7* implementiert wurde.

Das Fazit legt dar, welcher Algorithmus nach den Untersuchungen dieser Arbeit für eine praktische Anwendung empfohlen wird.

Abstract

This paper discusses different algorithms for dynamically adjusting the delay of speaker signals during playback. This is necessary to change the position of virtual sound sources in a multichannel sound field synthesis system. The focus of attention is on the acoustic artifacts caused by the algorithm.

First, physical and psychoacoustic basics of acoustic localization, the masking effect, the comb filter effect and the acoustic Doppler effect are summarized. In addition, basic knowledge of the development environments Max 7 and MATLAB R2018a, which have been used to implement the algorithms, is presented.

Three algorithms are presented which are based on crossfade, pitch shift and interpolation. First of all, the general functionality, an implementation with Max 7, the artifacts caused by this approach and finally possibilities for optimization are explained. In addition, combinations of the algorithms are considered.

This is followed by the description of a MATLAB prototype of the d&b Soundscape En-Scene Rendering Software in which two of the algorithms have been integrated.

Finally, a listening test is explained, which was carried out with the help of a Max program specially developed for testing the two algorithms.

The conclusion is that which algorithm is recommended for practical use according to the research in this paper.

Inhaltsverzeichnis

1 Einleitung.....	1
2 Grundlagen	5
2.1 Akustische Lokalisation	5
2.1.1 Interaurale Laufzeitdifferenzen	5
2.1.2 Frequenzabhängige interaurale Pegeldifferenzen	6
2.1.3 Präzedenzeffekt	7
2.2 Maskierungseffekt	9
2.3 Kammfiltereffekt	10
2.4 Akustischer Dopplereffekt	11
2.5 Verwendete Entwicklungsumgebungen	13
2.5.1 Max 7	13
2.5.1.1 Ringpuffer zur Signalverzögerung mit Max 7	16
2.5.2 MATLAB R2018a	18
2.5.2.1 Ringpuffer zur Signalverzögerung mit MATLAB R2018a	18
3 Lösungsansätze.....	21
3.1 Blende	22
3.1.1 Implementierung des Blendalgorithmus mit <i>Max 7</i>	23
3.1.2 Die Artefakte des Blendalgorithmus	27
3.1.3 Optimierung des Blendalgorithmus	30
3.1.3.1 Ramp and Hold	30
3.1.3.2 Rampendauer	30
3.1.3.3 Rampenform.....	33
3.2 Sprung-Interpolation	37
3.2.1 Implementierung des Interpolationsalgorithmus mit Max 7	38
3.2.2 Artefakte der Sprung-Interpolation	41

3.3 Slide	43
3.3.1 Implementierung des Slidealgorithmus mit Max 7	44
3.3.2 Artefakte des Slidealgorithmus	47
3.3.2.1 Frequenzverschiebung	47
3.3.2.2 Verzerrungsartige Artefakte	51
3.3.3 Optimierung des Slidealgorithmus	52
3.3.3.1 Mittelwertglättung	52
3.3.3.2 Blendglättung	54
3.4 Kombinationen der Ansätze	56
3.4.1 Parallelansatz.....	56
3.4.2 Multi-Short-Blend	57
3.4.3 Grenzwertansatz	58
3.5 Fazit der Lösungsansätze	60
4 MATLAB Prototyp.....	61
4.1 Benutzeroberfläche	61
4.1.1 Main-Window	61
4.1.2 Graphic-Window	64
4.2 Implementierung	66
4.2.1 Berechnungsschleife.....	66
4.2.2 Implementierung des Blendalgorithmus.....	68
4.2.3 Implementierung des Slidealgorithmus.....	69
4.2.3.1 Slide ohne Artefaktoptimierung.....	69
4.2.3.2 Slide mit Blendglättung	72
5 Hörversuche	73
5.1 Aufbau und Vorbereitung der Tests	73
5.1.1 Versuchsanordnung	74
5.1.2 Max-Patch des Tests	75
5.1.3 Bewertungskriterien	77

5.1.4 Klangbeispiele	78
5.2 Auswertung	81
6 Fazit und Ausblick	84
Literaturverzeichnis	87
Anhang	89

Abbildungsverzeichnis

Abbildung 1: Lokalisation im Lautsprecheresetup	8
Abbildung 2: Kammfilter von identischen Signalen mit zeitlichem Versatz. 10	
Abbildung 3: Dopplereffekt	11
Abbildung 4: Ringpuffer zur Signalverzögerung in Max 7	17
Abbildung 5: Blendalgorithmus.....	22
Abbildung 6: Blende mit MAX 7: blender-Subpatch	24
Abbildung 7: Seitenbänder bei Amplitudenmodulation	28
Abbildung 8: Sonogramm Rampendauern; weißes Rauschen	31
Abbildung 9: Sonogramm Rampendauern; Sinus 800 Hz	32
Abbildung 10: Rampenfunktionen	33
Abbildung 11: Sonogramm Rampenfunktionen; weißes Rauschen	34
Abbildung 12: Sonogramm Rampenfunktionen; Sinus 800Hz.....	35
Abbildung 13: Sprung-Interpolationsansatz	37
Abbildung 14: Interpolation mit zu großem n führt zur Übersteuerung	41
Abbildung 15: Interpolation mit kleinem n.....	42
Abbildung 16: Slidealgorithmus	43
Abbildung 17: der Zentrale Silde-SubPatch <i>gen~ slideTimeG</i>	45
Abbildung 18: Knicke des Slideansatzes.....	51
Abbildung 19: Mittelwertglättung des Slidealgorithmus.....	53
Abbildung 20: Sonogramm Slideoptimierung	55
Abbildung 21: Prototyp; Main-Window.....	62
Abbildung 22: Prototyp; Graphic-Window	64
Abbildung 23: Berechnung Blendpuffer	69
Abbildung 24: Übertragungsprinzip beim Slide	70
Abbildung 25: Anordnung Hörversuch.....	74
Abbildung 26: Hörversuch; Max-Patch.....	76
Abbildung 27: Hörversuch; Flugkurve.....	79
Abbildung 28: Auswertung Hörversuch; Beurteilung des Klangs	81

Tabellen

Tabelle 1: Häufige Sonderobjekte	15
Tabelle 2: Geschwindigkeit eines Dopplereffekts mit der gleichen Frequenzverschiebung wie beim Slide mit j	49

Formeln

Formel 1: Frequenzverschiebung Dopplereffekt	12
Formel 2: Rampenfunktionen	33
Formel 3: Polynom der Sprung-Interpolation	37
Formel 4: relative Bewegungsgeschwindigkeit eines Slides.....	48

Abkürzungen

AM	Amplitudenmodulation
VZ	Verzögerungszeit
PA	Public Adress (Beschallungssystem für großes Publikum)

1 Einleitung

Mehrkanalige Wiedergabesysteme, vor allem die unterschiedlichen Verfahren zur Schallfeldsynthese, spielen heutzutage in der Audiotechnik eine immer größere Rolle. Ihr Vorteil besteht im Gegensatz zu kanalbasierten Surround- oder 3D-Audio-Systemen darin, dass sie eine präzise Lokalisation von virtuellen Klangquellen ermöglichen, die nicht nur von einer bestimmten Hörerposition (Sweetspot) abhängt, sondern von jeder Position eines größeren Beschallungsbereichs aus wahrgenommen werden kann.

Die Lokalisation akustischer Ereignisse beruht nicht nur auf Pegel-, sondern auch und vor allem auf interauralen Laufzeitdifferenzen. Sobald ein Wiedergabesystem eine breite Fläche beschallt, ist deshalb die virtuelle Positionierung einzelner Klangereignisse mit einem Verfahren das ausschließlich *Amplitude-Pannings* betreibt nicht sinnvoll. Die Ansätze des *Amplitude Pannings* erzeugen anhand von Pegelunterschieden in den Lautsprecherkanälen Phantomschallquellen. Wenn die Lautsprecher um nur eine Hörerposition herum verteilt werden, führt das meist zu guten Ergebnissen. Bei einem System, das große Flächen beschallt, wirken eine Vielzahl an Lautsprechern für die Positionierung einer Quelle zusammen. Dabei kann es aufgrund des Präzedenzeffekts an manchen Hörerpositionen zu Problemen mit der korrekten Lokalisation kommen.

Das lässt sich vermeiden indem neben den Pegeln auch Verzögerungszeiten zur Platzierung von virtuellen Schallereignissen berücksichtigt werden. Das bedeutet, dass das Signal einer Klangquelle je nach Lautsprecherposition unterschiedlich verzögert werden muss, um die Laufzeit des Schalls von der virtuellen Quelle zum Lautsprecher zu kompensieren. Je weiter sich ein Lautsprecher von dem Punkt entfernt befindet, an dem ein Klangobjekt lokalisiert werden soll, desto mehr muss das Signal verzögert werden.

Es ergibt sich ein Problem, wenn Klangobjekte keine statischen Positionen haben, sondern bewegt werden. Dabei ändern sich die nötigen Verzögerungszeiten, die für eine korrekte Objektpositionierung nötig sind. Die meisten gängigen Systeme reproduzieren keine fließenden Bewegungen. Stattdessen werden die statischen Positionen der Klangobjekte in regelmäßigen Intervallen aktualisiert.¹ Wenn sich die Position einer Quelle während der Wiedergabe ändert, treten bei jeder Aktualisierung akustische Artefakte auf. Eine Änderung in Form eines plötzlichen Umschaltens beispielsweise, führt zu Phasensprüngen und somit zu Störgeräuschen. Zur Reduzierung dieser Artefakte gibt es verschiedene Ansätze zur Übergangsgestaltung von einem Signal zum selben Signal mit anderer Verzögerungszeit.

Einen Ansatz, bei dem keine Artefakte auftreten, existiert gegenwärtig nicht. Die Ansätze zur Übergangsgestaltung versuchen die Artefakte zu verringern. Ihr Ziel ist die Reduzierung auf eine schwache, im Idealfall gar nicht mehr als störend wahrgenommene Dimension.

Da die Laufzeitberechnung zu einem erhöhten Bedarf an Rechenressourcen führt, der durch eine mögliche aufwendige Übergangsgestaltung zusätzlich gesteigert wird und das reproduzierte Schallfeld zum Zeitpunkt einer Positionsänderung je nach Übergang instabil werden kann, verzichten die meisten Wiedergabesysteme gänzlich auf die individuelle Verzögerung der Lautsprechersignale. Das stellt bei einem System, das sich auf einen einzigen Punkt einen sog. Sweetspot konzentriert, kein Problem dar. Sobald ein System, mit dem virtuelle Quellen positioniert werden sollen, eine größere Fläche beschallt, ist die Verzögerungsberechnung notwendig, um ein natürliches Klangbild zu reproduzieren.

¹ (Peters Nils, 2014)

Diese Arbeit untersucht Ansätze der Laufzeitänderung von Audiosignalen im Hinblick das Auftreten störender Artefakte, um die Platzierung von virtuellen Klangquellen mit dynamischen Positionen in einem laufzeitbasierten Mehrkanalwiedergabesystem zu ermöglichen.

2 Grundlagen

Im folgenden Kapitel werden für die Arbeit relevante physikalische und psychoakustische Grundlagen und Sachverhalte zusammengefasst. Zudem wird grundlegendes Basiswissen der Entwicklungsumgebungen *Max7* und *MATLAB* dargelegt, mit denen die Lösungsansätze umgesetzt wurden.

2.1 Akustische Lokalisation

Beim Richtungshören unterscheidet die Literatur zwischen Lokalisation in der horizontalen und in der vertikalen Ebene, der sog. Medianebene. Während in der Medianebene keine Unterschiede zwischen den Signalen an beiden Ohren auftreten, wird die Lokalisation in der horizontalen Ebene in erster Linie durch Unterschiede der beiden Ohrsignale ermöglicht.

Da bei stereofonen Wiedergabesystemen hauptsächlich von der Richtungswahrnehmung in der horizontalen Ebene Gebrauch gemacht wird,² wird die Lokalisation in der Medianebene hier nicht näher thematisiert.

Die Unterschiede der Ohrsignale setzen sich aus frequenzabhängigen interauralen Pegeldifferenzen und interauralen Laufzeitdifferenzen zusammen.

2.1.1 Interaurale Laufzeitdifferenzen

Die interauralen Laufzeitdifferenzen gelten beim natürlichen Hören als die wichtigsten Merkmale der Ohrsignale für die Lokalisation. Ein Schallereignis, das nicht genau auf der mittleren Achse zwischen beiden Ohren passiert, kommt unweigerlich an einem Ohr später an als am anderen. Dabei ist das

² (Dickreiter Michael, 2014)

menschliche Gehör in der Lage schon geringe Abweichungen von drei bis fünf Grad, also Laufzeitunterschieden von etwa 0,03 ms wahrzunehmen.³

Dabei orientiert sich das Gehör sowohl am Schwingungsverlauf als auch an der Einhüllenden der Schwingung. Bei tiefen Frequenzen unter 800 Hz können dem Schwingungsverlauf direkt Zeit- und Phasendifferenzen entnommen werden, weshalb auch unmodulierte Sinusschwingungen in diesem Frequenzbereich geortet werden können. Bei höheren Frequenzen ist das Gehör lediglich in der Lage aus den Hüllkurven einen zeitlichen Versatz abzuleiten. Übliche natürliche Schallereignisse wie Sprache zeichnen sich durch ein breites Frequenzspektrum und durch starke Hüllkurven aus. Sie sind deshalb besonders präzise und leicht zu lokalisieren, wohingegen Sinussignale schwerer zu orten sind.

2.1.2 Frequenzabhängige interaurale Pegeldifferenzen

Im Vergleich zu Laufzeitdifferenzen sind die frequenzabhängigen Pegeldifferenzen der Ohrsignale weniger ausschlaggebend für die Lokalisation von Schallereignissen. Die Ursache für Pegeldifferenzen in den Ohrsignalen sind die Abschattung und Dämpfung durch den Kopf und die Ohrmuscheln. Das Gehör ist in der Lage Pegelunterschiede im gesamten Hörspektrum wahrzunehmen. Jedoch sind diese in den tiefen Frequenzen unterhalb von etwa 300 Hz für die Lokalisation nicht mehr von Bedeutung, da sich Schallwellen mit entsprechend großer Wellenlänge um den Kopf beugen und praktisch ungedämpft an beiden Ohren ankommen. Mit zunehmender Frequenz werden die Pegeldifferenzen größer. Jedoch ist die Zunahme äußerst unregelmäßig statt linear. Deshalb wird davon ausgegangen, dass zwischen Pegeldifferenz und Schallrichtung keine feste Zuordnung für jede Signalart besteht. Sie hängt von dem

³ (Dickreiter Michael, 2014)

Frequenzspektrum des Signals ab. Je breitbandiger ein Signal ist, desto einfacher und übersichtlicher werden die Zusammenhänge von Pegeldifferenz und Schalleinfallrichtung. Es treten nicht bewusst wahrgenommene Klangfarbenunterschiede sog. interaurale Spektraldifferenzen auf, die eine entscheidende Rolle für die Lokalisation spielen.

Im Unterschied zu den Laufzeitunterschieden muss die Ortung durch Spektraldifferenzen erst erlernt werden, da ein Vergleich zu dem Spektrum des Signals gezogen wird, das vorliegt wenn das Schallereignis frontal (auf der Mittelachse) passiert.

Wegen der Komplexität dieser Zusammenhänge gilt die Ortung einer Schallquelle, die ausschließlich durch interaurale Pegelunterschiede passiert, unter bestimmten Umständen als nur beschränkt möglich.⁴

2.1.3 Präzedenzeffekt

Der Präzedenzeffekt, der auch oft als das Gesetz der ersten Wellenfront bezeichnet wird, ist die Ursache dafür, dass der Mensch in Lage ist in geschlossenen Räumen Klangereignisse zu lokalisieren.

Wenn es in einen Raum zu einem Schallereignis kommt, wenn beispielsweise jemand spricht, dann hören wir ihn oder sie im Regelfall aus der Richtung an der er oder sie sich befindet. Die Reflexionen von Wand, Decke und Boden scheinen bei der Ausbildung der Hörereignisrichtung keine Rolle zu spielen. Erst wenn Reflexionen sehr viel später ankommen als das Direktsignal werden diese anders lokalisiert. Sie werden dann als Echos, also als eigene Schallereignisse wahrgenommen. Die Echoschwelle, die den Grenzwert markiert ab

⁴ (Dickreiter Michael, 2014)

dem eine eintreffende Reflexion als neues Schallereignis wahrgenommen wird, hängt von verschiedenen Faktoren ab und liegt meist zwischen 1 ms und 80 ms.⁵

Alle hinreichend ähnlichen Signale, die zwischen dem Direktschall und der Echoschwelle am Ohr eintreffen, sind für die Schallereignisortung nicht von Bedeutung und steigern in der Wahrnehmung lediglich den Pegel. Ihr Pegel kann sogar je nach Umstand bis zu 10 dB höher sein als der des Primärschalls, ohne dass die Lokalisation beeinträchtigt wird.⁶

Dieser Effekt wird in der Beschallungstechnik vielfach ausgenutzt und ist gerade bei der Platzierung von virtuellen Schallquellen für eine breite Beschallungsfläche von großer Bedeutung. Wie in Abbildung 1 dargestellt, muss ein Objekt je nach Zuhörerposition zwischen unterschiedlichen Lautsprechern geortet werden. Dazu ist es notwendig, dass das Signal der Primärquelle aus allen abgebildeten Lautsprechern wiedergeben wird.

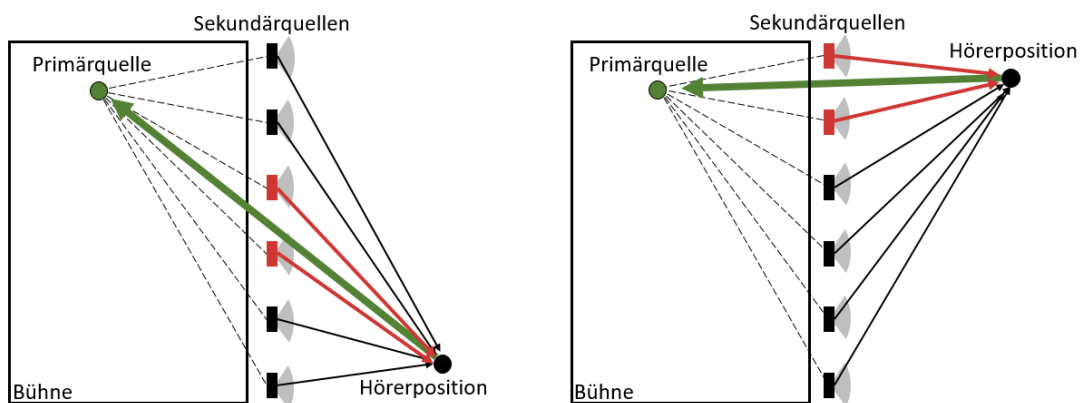


Abbildung 1: Lokalisation im Lautsprechersystem

⁵ (Blauert Jens, 2008)

⁶ (Blauert Jens, 2008)

Damit die Quelle wie in der linken Zeichnung nicht durch den Präzidenzeffekt zwischen den Lautsprechern mit dem geringsten Abstand zur Hörerposition lokalisiert wird, ist die individuelle Verzögerung der jeweiligen Lautsprecherkanäle notwendig.⁷

Deshalb ist bei einem immersive Audio Wiedergabesystem für großen Flächen, bei dem eine Vielzahl an Sekundärquellen in horizontaler Ebene zusammenwirken, eine Laufzeitberechnung für die einzelnen Lautsprecherkanäle nötig. Bei einem rein pegelbasierten System ist in diesem Fall nicht sichergestellt, dass die Position einer virtuellen Schallquelle von jeder Hörerposition aus exakt wahrgenommen wird.

2.2 Maskierungseffekt

Das menschliche Gehör ist in der Lage seine Schallempfindlichkeit einem herrschenden Pegel anzupassen. Dies führt u. a. zu einem Phänomen, das als Maskierung bezeichnet wird. Ein Reiz der auf das Gehör einwirkt; senkt die Empfindlichkeit für manche anderen Reize. Dies kann dazu führen, dass Schallereignisse nicht wahrgenommen werden, obwohl sie innerhalb des hörbaren Frequenzspektrums und über dem Pegel der „normalen“ Hörschwelle liegen, weil sie durch andere intensivere verdeckt werden. Denn die Hörschwelle verschiebt sich beim Einwirken eines Reizes auf das Gehör frequenzabhängig nach oben.

Ob ein Reiz maskiert wird, hängt von seinem Pegel und den enthaltenen Frequenzen ab, wie weit die Frequenzen von denen des Maskierers entfernt liegen und welche Frequenzanteile der Maskierer aufweist. Generell gilt, dass Frequenzen, die in unmittelbarer Nähe zu dem Maskierer werden stärker verdeckt als weiter entfernte. Rauschen besitzt ein starkes Maskierungsverhalten.

⁷ (d&b audiotechnik GmbH, 2018)

Je größer der Rauschanteil eines Signals, desto höher ist die Maskierungsschwelle. Signale mit eher tonalem Charakter weisen ein schwächeres Maskierungsverhalten auf.⁸

2.3 Kammfiltereffekt

Als Kammfilter wird eine partielle Auslöschung mehrerer Frequenzen bezeichnet, die durch die Summierung zweier identischer oder hinreichend ähnlicher Signale mit zeitlichem Versatz entsteht. Abhängig von der relativen Phasenlage zueinander treten konstruktive und destruktive Interferenzen auf.

Konstruktive Interferenzen entstehen bei allen Frequenzen, bei denen der Versatz der beiden Signale einem Vielfachen der Periodendauer entspricht. An diesen Stellen im Spektrum kommt es zu sog. Peaks, zur vollständigen Addition der Pegel. Bei identischen Signalen bedeutet ein Peak eine Steigerung um +6 dB. Bei identischen Signalen bedeutet ein Peak eine Steigerung um +6 dB.

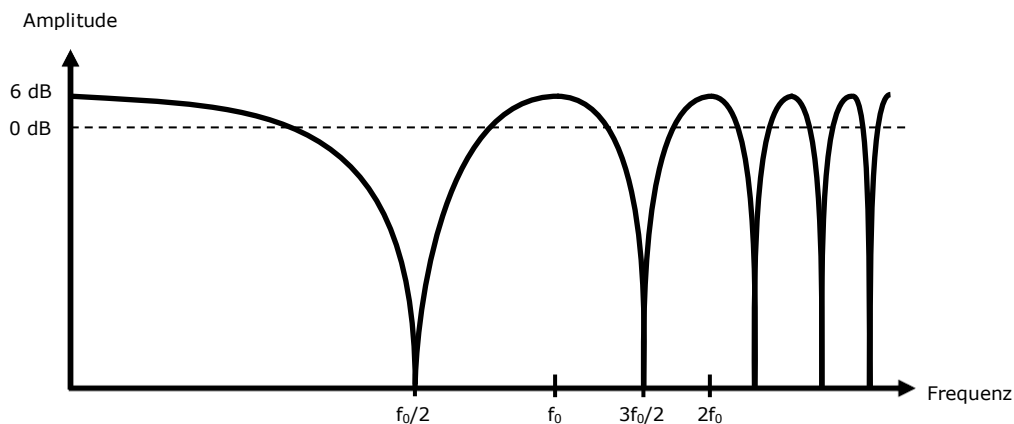


Abbildung 2: Kammfilter von identischen Signalen mit zeitlichem Versatz

⁸ (Bernhard Grill, 2014)

Destruktive Interferenzen treten bei allen Frequenzen auf, bei denen der Versatz der Signale zu einer relativen Phasenlage von 180° führt. Diese Frequenzen löschen sich bei gleichem Pegel vollständig aus.⁹

In dieser Arbeit werden vor allem Kammfilter thematisiert die durch die Summierung von zwei identischen zeitlich versetzten Signalen entstehen. Das bedeutet, dass ein Peak zu einer Verdoppelung des Pegels (+6 dB) und eine destruktive Interferenz zu einer vollständigen Auslöschung führt.

2.4 Akustischer Dopplereffekt

Der akustische Dopplereffekt beschreibt das Phänomen, dass das Signal einer Quelle bei einem zu ihr relativ bewegten Empfänger in der Tonhöhe verändert empfangen wird. Bewegt sich eine Quelle auf einen Beobachter zu, so wird die Tonhöhe ihres Signals nach oben verschoben empfangen, wenn sie sich entfernt nach unten. Das liegt daran, dass der Schall sich mit einer festen Geschwindigkeit ausbreitet, die von der Bewegung einer Quelle nicht beeinflusst wird. Dies führt dazu, dass die Wellen gestaucht bzw. gedehnt werden (siehe Abbildung 3).

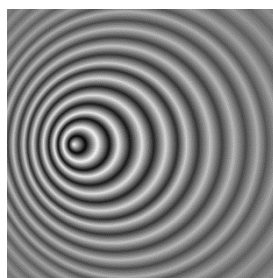


Abbildung 3: Dopplereffekt

Quelle: https://de.wikipedia.org/wiki/Doppler-Effekt#/media/File:Doppler_effect.svg

⁹ (Maier, 2008)

Berechnen lässt sich die empfangene Frequenz wie folgt:

Empfänger ruht, Sender bewegt sich:

$$f_E = f_S \frac{1}{1 \pm \frac{v_S}{c}}$$

Empfänger bewegt sich, Sender ruht:

$$f_E = f_S \left(1 \pm \frac{v_E}{c}\right)$$

Formel 1: Frequenzverschiebung Dopplereffekt

mit f_E = Frequenz beim Empfänger, f_S = Frequenz am Sender, v_E = Geschwindigkeit des Empfängers, v_S = Geschwindigkeit des Senders und c = Schallgeschwindigkeit.

Mit einem PA-Mehrkanalwiedergabesetup wie *d&b Soundscape* ist es nicht möglich einen natürlichen Dopplereffekt für virtuelle Schallquellen zu erzeugen. Denn ausschlaggebend für die Tonhöhenverschiebung ist die relative Geschwindigkeit und somit die Entfernungsänderung zwischen Quelle und Empfänger.

Diese unterscheidet sich je nach Höherposition in der Beschallungsfläche erheblich. Wenn ein virtuelles Klangobjekt beispielsweise von links nach rechts über die Bühne bewegt wird, müsste die Tonhöhenverschiebung bei einer Person vorne stärker ausfallen als bei einer Person weiter hinten. Dazu müssten die Beobachter im Beschallungsbereich zeitgleich aus denselben Lautsprechern verschiedene Signale empfangen.

Deshalb ist es nur sehr begrenzt möglich, mit Lautsprechern einen Dopplereffekt realitätsnah zu simulieren.

2.5 Verwendete Entwicklungsumgebungen

Der folgende Abschnitt fasst die für das Verständnis der in dieser Arbeit thematisierten Implementationen notwendigen Grundlagen in *Max 7* und *MATLAB R2018a* zusammen.

2.5.1 Max 7

Max ist eine graphische Entwicklungssoftware der Firma *Cycling '74*, die vor allem bei Projekten für Musik und Medienkunst angewandt wird. Sie basiert auf dem Prinzip der Objektorientierung und ist zudem modular und datenstromorientiert aufgebaut.

Die mit Max programmierten Projekte werden „Patches“ genannt. Sie bestehen aus Objekten, die mit Datenkanälen verbunden sind. Objekte sind kleine Funktionen, die anhand empfangener Signale oder Nachrichten neue Nachrichten oder Signale berechnen und ausgeben. Sie haben an ihrer Oberseite Ein- und an der Unterseite Ausgänge, an denen die Datenkanäle angeschlossen werden.

Es wird zwischen zwei Arten von Objekten unterschieden, den Max- und den MSP-Objekten. MSP steht für *MaxSignalProcessing*. Während die Max-Objekte Nachrichten (Zahlen, Symbole, Listen, usw.) empfangen, verarbeiten und senden, sind MSP-Objekte für Signale bzw. Signalströme wie beispielsweise Audiosignale zuständig. Um welchen Typ es sich bei einem Objekt handelt, wird durch den Namen festgelegt. Ein MSP-Objekt hat am Ende seines Namens ein „~“ Zeichen.




Die Übertragung einer Nachricht findet nur statt, wenn ein Max-Objekt die Nachricht über den Ausgang schickt, was bei den meisten Objekten bedeutet, dass ihre Berechnung auf irgendeine Weise ausgelöst wurde. Die Updaterate beträgt dabei je nach Systemeinstellung meist ca. 20 ms. Ein MSP-Objekt hingegen sendet ununterbrochen ein Signal.

Die Signale eines MSP-Objekts werden nicht in einzelnen Samples übertragen, sondern in Blöcken. Die Größe dieser Blöcke und die Samplerate mit der das System arbeitet, können im Vorfeld festgelegt werden.

Ob ein Datenkanal Max-Nachrichten oder MSP-Signale überträgt, lässt sich visuell unterscheiden. Nachrichtenkanäle werden als durchgezogene einfarbige, Signalkanäle als zweifarbig gestreifte Linie dargestellt.

Bei Eingängen wird bei Max-Objekten zwischen sog. *hot-* und *cold-inlets* unterschieden. Daten die an einem *cold-inlet* ankommen werden lediglich gespeichert, während Daten die an einem *hot-inlet* ankommen gleichzeitig das Objekt ausführen, also eine Ausgabe im Ausgang hervorrufen.

Ein klassisches Objekt hat einen intuitiven Namen, aus dem hervorgeht, was das Objekt berechnet. Es gibt daneben jedoch auch einige Sonderformen, die meist für die Interaktion konzipiert sind. In Tabelle 1 sind die in den Programmen dieser Arbeit verwendeten Sonderobjekte aufgelistet.

<p>number</p> 	<p>Bekommt durch den Eingang oder Benutzerinteraktion eine Zahl und gibt diese sobald sie sich ändert aus.</p>
<p>number~</p> 	<p>Hat zwei Zustände:</p> <ol style="list-style-type: none"> 1. Zeigt die aktuellen Werte eines anliegenden Signals 2. Ähnlich wie <i>number</i>, mit dem Unterschied, dass die Zahl jederzeit als Signal mit gleichbleibendem Wert gesendet wird. <p>Kann nicht die ankommende Zahl anzeigen und als Signal senden</p>
<p>button</p> 	<p>Übergibt bei jeder ankommenden Nachricht und bei Benutzerinteraktion ein sog. <i>BANG</i>. Das ist eine Nachricht die ein Objekt ausführt ohne neue Zahlen zu übergeben</p>





toggle 	Schalter mit zwei Zuständen „0“ und „1“; gibt bei Zustandsänderung den neuen Zustand weiter. Ankommende Nachricht beeinflusst Zustand: „0“ setzt auf „0“ alles andere auf „1“. Ankommendes <i>BANG</i> ändert den Zustand.
message 	Sobald eine Nachricht am ersten Eingang ankommt wird der Inhalt des Objekts als Nachricht gesendet. Nachrichten am zweite Eingang überschreiben den Inhalt
slider 	Schnelles Eingeben einer Zahl durch Benutzerinteraktion, die am Ausgang gesendet wird.
gswitch 	Nachricht am ersten Eingang bestimmt, ob an den Ausgang Nachrichten des zweiten oder des dritten Eingangs übergeben werden.

Tabelle 1: Häufige Sonderobjekte

Neben dem Verwenden von klassischen und Sonderobjekten ist es auch möglich, eigene Objekte zu definieren. Ein Beispiel dafür sind die sog. Subpatches, also Unterprogramme. Subpatches werden mit dem Label „patch“ oder „p“ gekennzeichnet. Sie beinhalten ihrerseits eine Verkettung von Objekten und dienen bei großen Projekten vor allem der Übersichtlichkeit. Eine Besonderheit unter den Subpatches stellen die „gen~“-Objekte dar.

Bei *gen~*-Subpatches handelt es sich um MSP Objekte, in denen jedes Sample einzeln verarbeitet wird. Es ist nicht möglich die Max- oder MSP-Objekte in einem solchen Subpatch zu verwenden. Stattdessen gibt es eigene *gen~*-Objekte, die in diesen Patches verwendet werden können. Wie durch das Tildezeichen angedeutet, besitzt *gen~* nur MSP-Ein- und Ausgänge. Zur Steuerung ist es aber auch möglich, Nachrichten an diese Form der Subpatches zu schicken.

Diese Nachrichten werden Parameter genannt. Sie müssen aus einer Liste von zwei Elementen bestehen und können an jeden beliebigen Eingang angeschlossen sein. Zum Empfangen der Nachricht muss im *gen~*-Patch ein „*param*“-Objekt angelegt sein. Ein *param*-Objekt hat einen Namen, der gleichzeitig die Empfangsadresse für Nachrichten definiert. Es empfängt jede Nachricht, deren erstes Element seinem Namen entspricht. Das Objekt sendet dann ununterbrochen das zweite Element der Nachricht, die zuletzt mit seiner Adresse empfangen wurde.

2.5.1.1 Ringpuffer zur Signalverzögerung mit Max 7

Wie bei den meisten Entwicklungsumgebungen gibt es auch bei Max 7 viele Möglichkeiten und Wege um zu einem Ergebnis zu kommen. Ein Beispiel hierfür ist das Verzögern eines Audiosignals. Dafür gibt es ein Objekt namens „*delay~*“. Am ersten Eingang kommt ein Signal an, das zeitlich verzögert ausgegeben wird. An einem zweiten Eingang, einem *cold-inlet*, kann die Verzögerungszeit (VZ) in Samples festgelegt werden.

Dieses Objekt eignete sich für die Implementierung der Ansätze zur Übergangsgestaltung bei VZ-Änderung nicht. Denn im *delay~*-Objekt ist bereits eine Form der Übergangsgestaltung, die der Blende, integriert. Außerdem ist die Steuerung zeitlich nicht präzise genug, da sie über Nachrichtenkanäle erfolgt.

Die Max-Patches, in denen die Lösungsansätze aus Kapitel 3 implementiert wurden, benutzen zur Signalverzögerung daher eine Art Ring-Puffer.

In diesem Puffer mit fester Länge werden die Samples des ankommenden Signals gespeichert. Das passiert mit einem Objekt namens „*poke~*“. Dieses bekommt am ersten Eingang das Signal und am zweiten die Position, an der das aktuelle Sample gespeichert werden soll. Am zweiten Eingang liegt das Signal eines MSP-Zählerobjekts an. Dieses zählt in Sampleschritten aufwärts.

2.5.2 MATLAB R2018a

MATLAB ist eine codebasierte Entwicklungsumgebung von MathWorks, die vor allem von Ingenieuren und Wissenschaftlern zur Lösung mathematischer Probleme genutzt wird.

Der Name setzt sich aus den Abkürzungen *MAT* für Matrix und *LAB* für Laboratory zusammen. Denn MATLAB ist vor allem für numerische Berechnungen von Matrizen ausgelegt. Effektiv ist es vor allem dann, wenn viele Zahlen einer Matrix oder eines Vektors auf einmal übertragen oder verarbeitet werden sollen. Anders als bei vielen anderen Sprachen, wie z. B. C, muss bei der Übertragung einer Vielzahl an Werten aus einem Vektor, keine Schleife verwendet werden. Stattdessen kann eine beliebige Anzahl an Werten gleichzeitig übertragen werden.

MATLAB verfügt über die Möglichkeit sog. *Toolboxen* einzubinden, die die Sprache um zusätzliche Befehle erweitern. Für den Prototyp aus Kapitel 4 wurden folgende Toolboxen verwendet:

- Signal Processing Toolbox
- DSP System Toolbox
- Audio System Toolbox

2.5.2.1 Ringpuffer zur Signalverzögerung mit MATLAB R2018a

Auch beim MATLAB-Prototyp wurde ein Ringpuffer zur steuerbaren Verzögerung des Eingangssignals eingesetzt.

Ähnlich wie auch bei Max 7 verarbeitet das MATLAB Programm Blöcke aus mehreren Samples auf einmal. Der Ringpuffer des Eingangssignals funktioniert jedoch anders als der in Max. Die Werte des Maxpuffers werden an eine festen Position geschrieben und Lese- und Schreibposition durchlaufen ihn

ringförmig. Beim MATLAB Ringpuffer, wie er in Kapitel 4 verwendet wird, wird stets an letzter Position geschrieben und alle anderen Werte wandern nach vorne durch. Dies passiert in zwei Schritten. Zunächst werden die alten Werte mit dem Befehl „circshift“ um die Blockgröße nach vorne geschoben. Danach können die Werte des aktuellen Blocks an die hintersten Stellen geschrieben werden.

Die neuen Werte als Block hinten anzufügen, ist in diesem Fall sinnvoll, da beim Einlesen eines Blockes ein Puffer so befüllt wird, dass der erste Wert an vorderste Stelle geschrieben wird und der letzte Wert (also der zuletzt gelesene) an hinterster Stelle.

3 Lösungsansätze

Dieses Kapitel behandelt die untersuchten Lösungsansätze zur VZ-Änderung, die mit der Entwicklungssoftware *Max7* implementiert wurden. Dabei wird zunächst das Prinzip des jeweiligen Algorithmus erläutert, welche Artefakte bei der Anwendung des Ansatzes auftreten und wie sich der Algorithmus im Hinblick auf diese optimieren lässt.

Die Max-Patches mit denen die Ansätze umgesetzt und getestet wurden, ermöglichen die VZ-Änderung eines einzigen Signalkanals. Sie dienen der Voruntersuchung und Vorauswahl für die Entwicklung des immersive Audio Renderingprototyps, der in Kapitel 4 thematisiert wird.

Neben den Verfahrensweisen thematisiert das Unterkapitel 3.4 einige Versuche Kombinationen aus diesen Verfahrensweisen zu bilden.

Zu Erläuterung der Verfahren wird der Vergleich zu einer Bandmaschine herangezogen. Der Schreibkopf der Maschine entspricht dem ankommenden Signal. Ein Lesekopf, der sich weiter hinten am Tonband befindet, liest das Wiedergabesignal aus. Der Abstand von Lese- und Schreibkopf entspricht der Verzögerungszeit.

3.1 Blende

Ein Ansatz, der bei *d&b Soundscape* und anderen Systemen¹⁰ Anwendung findet, ist der der Blende. Das Prinzip entspricht dem einer Bandmaschine mit zwei Leseköpfen. Während ein Lesekopf das Signal ausliest und wiedergibt, kann der andere an eine neue gewünschte Position angebracht werden. Anschließend werden die Signale der Leseköpfe mit einem Crossfade überblendet.

Abbildung 5 zeigt eine beispielhafte Blende. Oben sind die beiden Lesekopfsignale abgebildet, unten das daraus resultierende Wiedergabesignal.

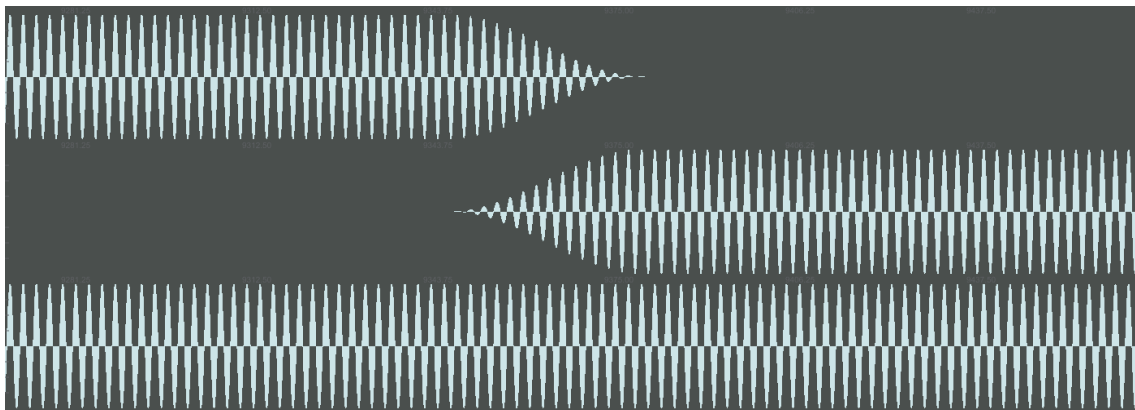


Abbildung 5: Blendalgorithmus

Für das Überblenden wird ein zusätzlicher passiver Signalkanal benötigt. Während das Signal des aktiven Kanals wiedergegeben wird, kann die Verzögerungszeit des Signals im passiven Kanal angepasst werden, ohne dass es am Signalausgang zu Phasensprüngen kommt.

¹⁰ z. B. (Peters Nils, 2014)

Wenn die VZ des passiven Kanals der Ziel-VZ entspricht, tauschen die beiden Kanäle mit einer Überblendung ihre Funktion, d. h. passiv wird zu aktiv und umgekehrt.

3.1.1 Implementierung des Blendalgorithmus mit *Max 7*

Der folgende Abschnitt thematisiert die Umsetzung des Blendalgorithmus mit *Max 7*. Das ankommende Signal wird zunächst in einem Ringpuffer gespeichert (siehe 2.5.1.1). Das Auslesen aus diesem Puffer mit dem *index~*-Objekt erfolgt zweimal parallel.

Zum Ändern der VZ ist ein *slider* vorgesehen, bei dem durch Klicken die Zeit einmalig geändert, oder mit gehaltener Maus eine fließende Bewegung der VZ simuliert werden kann. Die VZ die vom *slider* geschickt wird, durchwandert einen Subpatch namens „*p select*“. Dieser Subpatch gibt die ankommende Zahl im Wechsel an seinen rechten und linken Output aus. An den Ausgängen liegen die Subtraktionen des „Lesekopfzähler-MSP-Signals“ an, das mit dem Objekt „*index~*“ das verzögerte Signal ausließt. Damit wird die VZ immer nur in einem der beiden Ausleseprozesse geändert, was dem Prinzip des aktiven und passiven Kanals entspricht.

Beide verzögerten Signale werden in einem Subpatch namens „*p blender*“ verarbeitet. In ihm findet die Überblendung statt, für die eine aus sieben Rampenfunktionen ausgewählt werden muss. Dieser Subpatch bekommt neben den Signalen noch fünf weitere Nachrichten zur Steuerung. Diese sind:

1. Eine Zahl, die festlegt welche Rampenfunktion verwendet wird.
2. Eine Zahl, die den Grad der Butterworth Approximation festlegt, falls diese als Rampenfunktion gewählt wurde.
3. Eine Zahl, die die Rampendauer in Millisekunden festlegt.
4. Auslöser zum Starten und Beenden einer Blende.

5. Eine Zahl (0 oder 1), die festlegt, welches der Signale aktiv und welches passiv ist.

Abbildung 6 zeigt eine vereinfachte Form des *p blender*-Subpatches, die auf die logarithmische Rampenfunktion reduziert wurde. Davon abgesehen, dass keine andere Rampenfunktion gewählt werden kann, unterscheidet sich die Funktionsweise aber nicht.¹¹

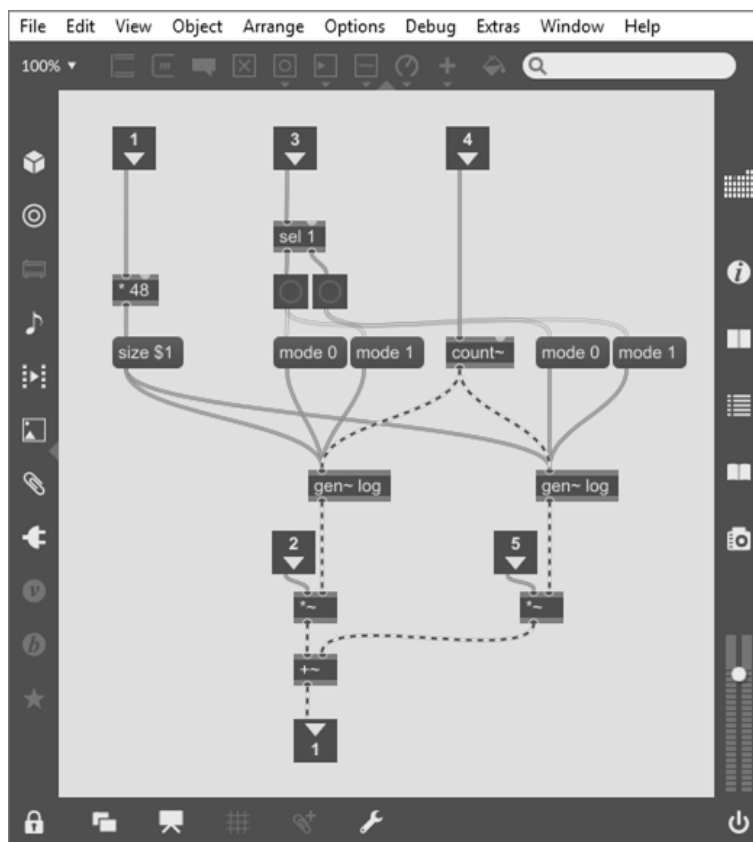


Abbildung 6: Blende mit MAX 7: blender-Subpatch

¹¹ siehe Anhang für Abbildung des vollständigen Blend-Max-Patches

Durch die Eingänge 2 und 5 (beim vollständigen Subpatch 6 und 7) kommen die beiden Signale (aktiv und passiv) an. Sie durchlaufen jeweils ein Multiplikationsobjekt ($*\sim$), werden danach zu einem Signal addiert ($+\sim$) und aus dem Patch ausgegeben. Die „ $*\sim$ “-Objekte modulieren die Signale mit der Rampe (zum Ein-/Ausblenden), wenn keine VZ-Änderung stattfindet, wird das aktive Signal mit 1, das passive mit 0 multipliziert. Ein MSP-Kanal in dem eine 0 gesendet werden soll, überträgt zur besseren Effizienz standardmäßig nichts.

Die Zahl mit der ein Signal multipliziert wird, die 0, 1 oder der Rampenwert dazwischen, kommt aus einem $gen\sim$ -Subpatch. In diesem Fall heißen die $gen\sim$ -Patches „log“. Das steht für die logarithmische Form der Blende.

Das Objekt „ $gen\sim\ log$ “ hat nur einen Eingang. An diesem liegen der Ausgang eines Zählerobjekts namens „ $count\sim$ “ und Nachrichtenboxen für die Parameter „ $size$ “ und „ $mode$ “ an. Die Zählvariable von $count\sim$ stellt das „ x “ der mathematischen Form der Rampenfunktion dar, außerdem wird es für die Steuerung genutzt. Wenn das Ende der Rampendauer erreicht wird, muss keine weitere Berechnung mehr stattfinden. Denn dann wird je nachdem ob das Signal die Rolle des aktiven oder des passiven innehat, nur die Zahl 0 oder 1 ausgegeben.

Der Parameter „ $mode$ “ legt durch die Zahl 0 oder 1 fest, ob es sich um die Rampe eines aktiven oder eines passiven Signals handelt. Welche Rampe den aktiven und welche den passiven Modus berechnet, wird von außerhalb des $p\ blender$ -Subpatches durch „ $p\ sliderToValue$ “ gesteuert.

Der Subpatch $p\ sliderToValue$ übergibt die im $slider$ gewählte VZ. Ein $slider$ -Objekt sendet bei jeder Änderung die neue Zahl. Damit keine VZ-Änderung während einem Blendvorgang stattfinden kann, überprüft $p\ sliderToValue$ in einem zeitlichen Raster, ob vom $slider$ ein neuer Wert gesendet wurde.

Damit kann, ähnlich wie bei einem Renderer zur Schallfeldreproduktion, eine Rate eingestellt werden, die festlegt in welchen Intervallen die VZ aktualisiert wird. Liegt seit der letzten Aktualisierung eine VZ-Änderung vor, wird die neue Zeit als Nachricht gesendet.

Diese Nachricht gelangt in den *p select*-Subpatch und löst über zwei Buttons *BANG* Nachrichten aus. Das erste *BANG* startet den Blendvorgang, in dem es in *p blender* das *count~*-Objekt aktiviert. Das zweite schaltet ein *toggle* um. Dieses *toggle* legt fest, welches der beiden *index~*-Signale das aktive und welches das passive ist. 0 bedeutet der linke Kanal ist passiv; 1 bedeutet der rechte Kanal ist passiv. Der *toggle*-Ausgang ist mit *p select* verbunden und legt dort fest in welchem Signal die VZ geändert wird (0 = im linken, 1 = im rechten Kanal). Außerdem regelt er in *p blender* welcher Kanal ein- und welcher ausgeblendet wird.

Am oberen Rand des Patches befinden sich links neben dem Slider zu Änderung der VZ zwei Slider in Form von Potentionmetern. Mit dem linken wird die Rampendauer in Millisekunden festgelegt, mit dem rechten eine Haltezeit in der nicht geblendet wird. Die Summe aus beiden Zeiten legt das Intervall fest, in dem die VZ-Aktualisierung in *p sliderToValue* stattfindet. Zudem wird die Rampendauer an *p blender* übergeben, wo die Rampenfunktionen berechnet werden.

Der Subpatch *p blender* hat einen Ausgang, der das resultierende Signal ausgibt. Von dort wird das Signal zu einem Objekt names „*dac~*“ geleitet. Der Objektname *dac~* steht für *digital-analog-converter*. Das ist die Audioschnittstelle von MAX 7 Patches.

3.1.2 Die Artefakte des Blendalgorithmus

Der Einsatz der Blende hat neben spektralen Seitenbandeffekten vor allem Kammfiltereffekte zur Folge, die bei jeder Änderung der VZ auftreten.

Vermeiden lassen sich die auftretenden Artefakte nicht. Jedoch sind sie von mehreren Faktoren abhängig, durch deren Optimierung sie reduziert werden können. Diese Faktoren sind die Länge der Blende und ihre Form.

Bei dem Kammfilter handelt es sich um ein Artefakt, das vor allem bei Signalen mit breitem Spektrum hörbar wird. Je schmalbandiger das Frequenzspektrum des Inputsignals ist, desto wahrscheinlicher ist es, dass keine der Frequenzen vollständig ausgelöscht wird. Deshalb wird ein Kammfilter bei diesen Signalen meist weniger stark wahrgenommen, solange die vorhandenen Frequenzen nicht genau in einem Bereich des Peaks oder der Auslöschung liegen.

Wie stark der Kammfilter auftritt hängt davon ab, wie ähnlich sich die Pegel der überlagerten Signale sind. Wenn beide Signale genau den gleichen Pegel haben, bedeuten konstruktive Überlagerungen die Verdopplung der Magnitude (+6 dB) und destruktive Überlagerungen die vollständige Auslöschung der jeweiligen Frequenz.

Deshalb sollte der Moment, an dem die Pegel nahe bei einander liegen, von möglichst kurzer Dauer sein. Das ist dann der Fall, wenn die Rampe insgesamt von kurzer Dauer und die Steigung im Bereich der ähnlichen Pegel hoch ist. Eine zu kurze und zu steile Rampe oder ein nicht stetiger Verlauf führen jedoch zu spektralen Seitenbandartefakten.

Die Blende kann als Amplitudenmodulation (AM) von Wiedergabesignal und Rampenfunktion verstanden werden, bei der das Wiedergabesignal dem Trägersignal entspricht.

Bei AM entstehen im Spektrum um die Frequenz des Trägersignals nach oben und unten gespiegelte Abbildungen, das untere und das obere Seitenband, des Niederfrequenzsignals (siehe Abbildung 7).

Da das Trägersignal in diesem Fall das Wiedergabesignal ist, sich also aus mehreren Frequenzen des Hörspektrums zusammensetzt, liegen auch die Seitenbänder im hörbaren Bereich.

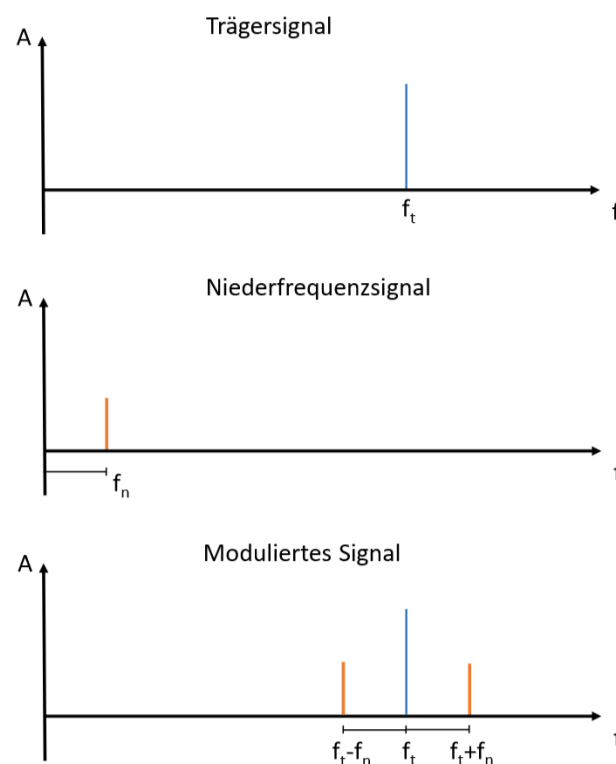


Abbildung 7: Seitenbänder bei Amplitudenmodulation

Eine theoretische Rampe mit der Dauer von null Samples, also ein sofortiges Umschalten, entspräche einer Rampe mit Form der Heaviside-Funktion, der Sprungfunktion, in der unendlich viele Frequenzen enthalten sind. Dies führt zu Seitenbändern, die sich über das gesamte Frequenzspektrum erstrecken. Eine Rampe mit größerer Dauer entspricht einer tiefpassgefilterten Sprungfunktion. Je länger die Rampe ist, desto tiefer ist die Filterfrequenz der entsprechenden tiefpassgefilterten Sprungfunktion und desto schmaler sind

die Seitenbänder. Diese Artefakte werden also durch eine möglichst große Rampendauer verringert.

Bei Wiedergabesignalen mit breitem Frequenzspektrum können die Seitenbänder bei ausreichendem Pegel durch das Phänomen der Maskierung überdeckt werden. Sie werden dann nur gering oder gar nicht wahrgenommen. Je weniger Frequenzen im Wiedergabesignal und je mehr in der Rampenfunktion enthalten sind, desto deutlicher wahrnehmbar sind die Seitenbandeffekte. Die Verwendung einer Cosinus-Rampe reduziert sie am stärksten, da die Cosinusfunktion nach Fourier nur eine Frequenz enthält.

Zusammengefasst entstehen bei dem Ansatz der Blende Kammfilterartefakte, die bei zunehmender und Seitenbänder, die bei abnehmender Rampendauer deutlicher hörbar werden. Deshalb stellt die Optimierung der Dauer einen Kompromiss zwischen dem Auftreten und der Reduktion beider Artefakte dar.

Bei einer fließenden Bewegung erinnert ein immer wiederkehrender kurzzeitig auftretender Kammfilter an eine sehr schnelle Schwebung oder einen Tremoloeffekt. Es löschen sich zwar nie alle Frequenzen aus, aber gerade bei Kammfiltern hoher Ordnung werden Frequenzen des gesamten Hörspektrums in ihrem Pegel beeinflusst.

3.1.3 Optimierung des Blendalgorithmus

Da bei zu kurzen und zu steilen Rampen Seitenbandeffekte deutlich in Erscheinung treten und bei zu langen oder zu flachen Rampen die Kammfilter deutlicher zu hören sind, gilt es bei der Optimierung einen Mittelweg zu finden bei dem sich beide Artefakte möglichst in geringem Ausmaß ausbilden.

3.1.3.1 Ramp and Hold

Der erste Schritt der Optimierung erfolgt durch das sog. „Ramp and Hold“ Verfahren. Dabei wird die VZ nach jeder Blende für eine gewisse Dauer gehalten, d. h. für einen bestimmten Zeitraum wird keine Blende durchgeführt. Damit wird vermieden, dass bei einer fließenden Bewegung das Signal ununterbrochen verzerrt wird. Wenn die Haltedauer länger als die Rampendauer ist, wird mehr unverzerrtes Signal wiedergegeben als verzerrtes.

3.1.3.2 Rampendauer

Um eine praxistaugliche Rampendauer zu ermitteln, wurden in Tests die Artefakte von Cosinus-Rampen unterschiedlicher Dauer verglichen. Die VZ-Änderung entspricht dabei der Bewegung einer Quelle die sich hinter der Sekundärquelle auf einer Kreisbahn befindet. In Abbildung 8 sind die Sonogramme eines Test mit weißem Rauschen als Eingangssignal zu sehen.

Es ist deutlich erkennbar, dass eine große Rampendauer die Kammfilter verstärkt in Erscheinung treten lässt. Bei einer Dauer von 10 ms sind die Kammfilter auditiv kaum wahrnehmbar.

Die selbe VZ-Änderung mit einem Sinus-Signal als Input zeigt die Spektralen Seitenbänder, die bei der kurzen Rampendauer von 10 ms sehr stark ausgeprägt sind. Schon bei 35 ms sind sie so schwach, dass sie bei Signalen mit

breiterem Spektrum kaum hörbar sind. Ein Gleichgewicht, bei dem beide Artefakte nicht wahrgenommen werden gibt es jedoch nicht.

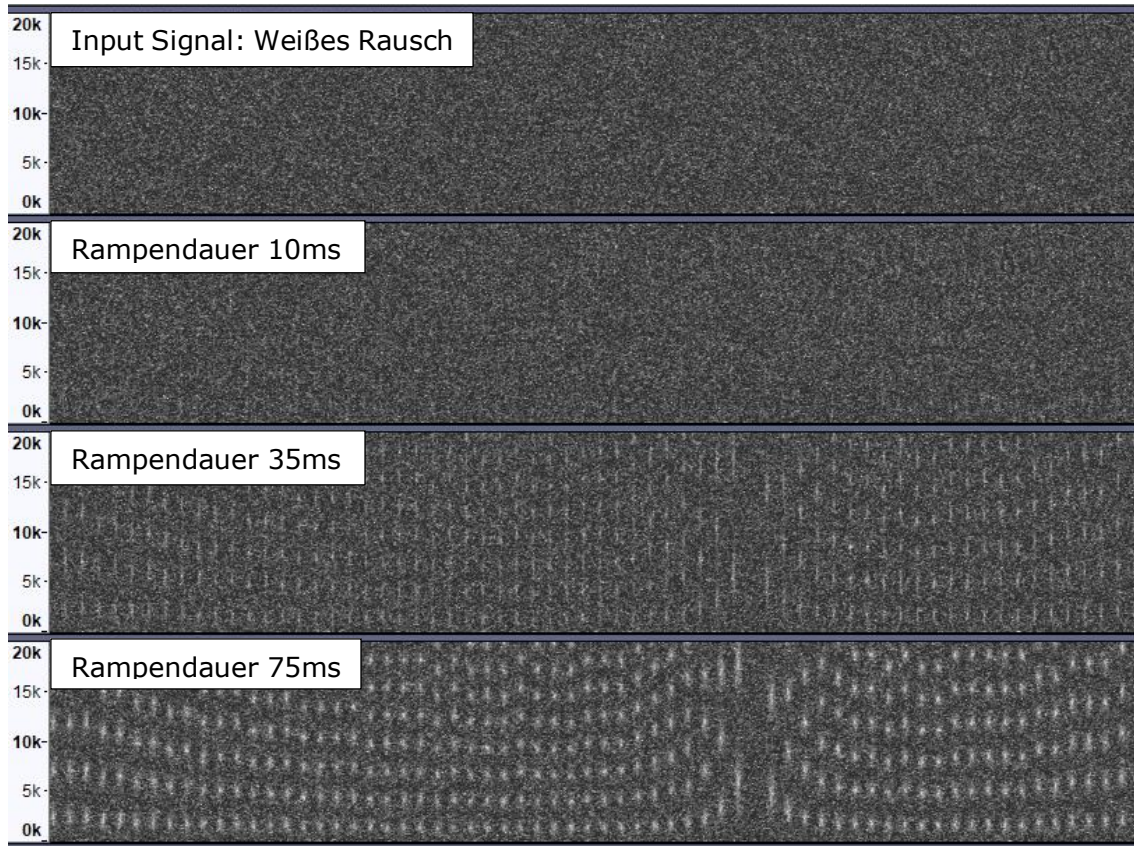


Abbildung 8: Sonogramm Rampendauern; weißes Rauschen

Tests mit Musik ergaben, dass geringe Seitenbandeffekte deutlicher zu hören sind als kurzzeitig auftretende Kammfilter.

Eine Rampendauer von 35 ms liefert den subjektiv als am besten bewerteten Kompromiss aus Kammfilter- und Seitenbandeffekten.

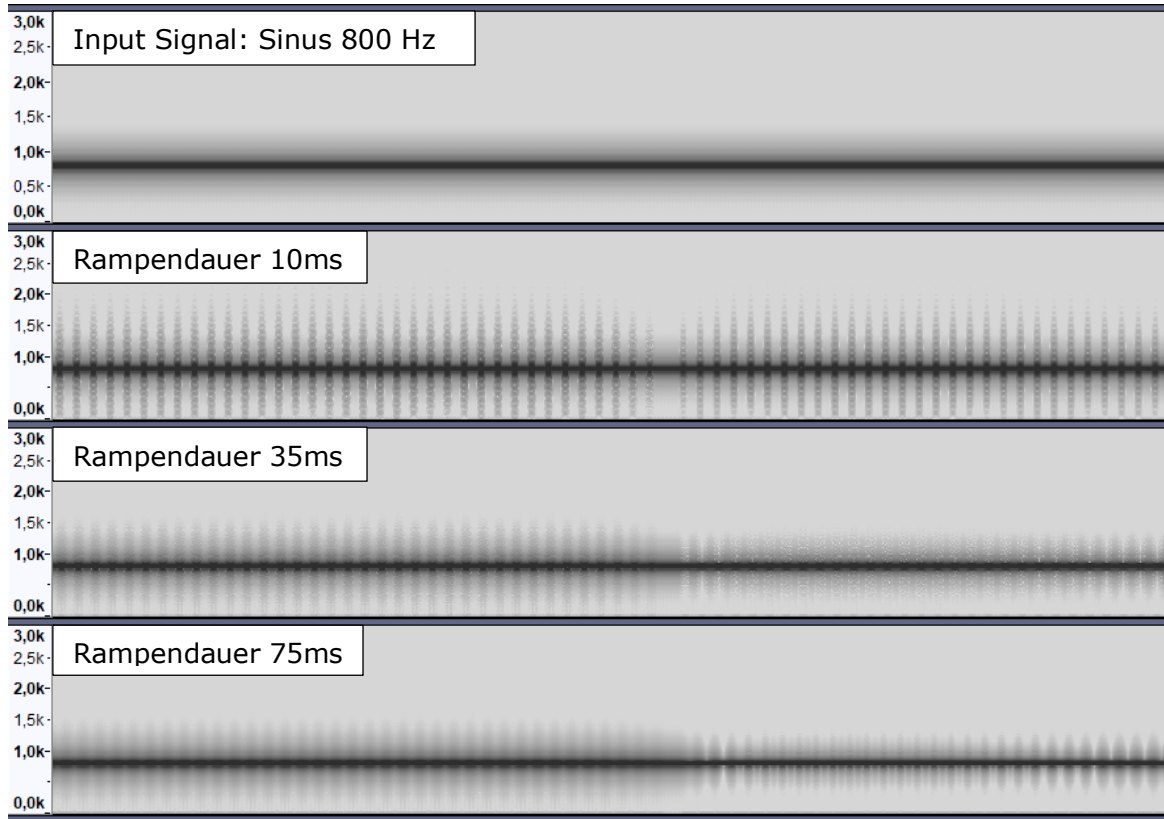


Abbildung 9: Sonogramm Rampendauern; Sinus 800 Hz

3.1.3.3 Rampenform

Ähnlich wie für die Rampendauer wurden auch Vergleiche unterschiedlicher Rampenfunktionen angestellt. Die Rampendauer lag hierfür bei allen Rampen bei 35 ms. Konkret wurden vier Rampenfunktionen verglichen (siehe Abbildung 10).

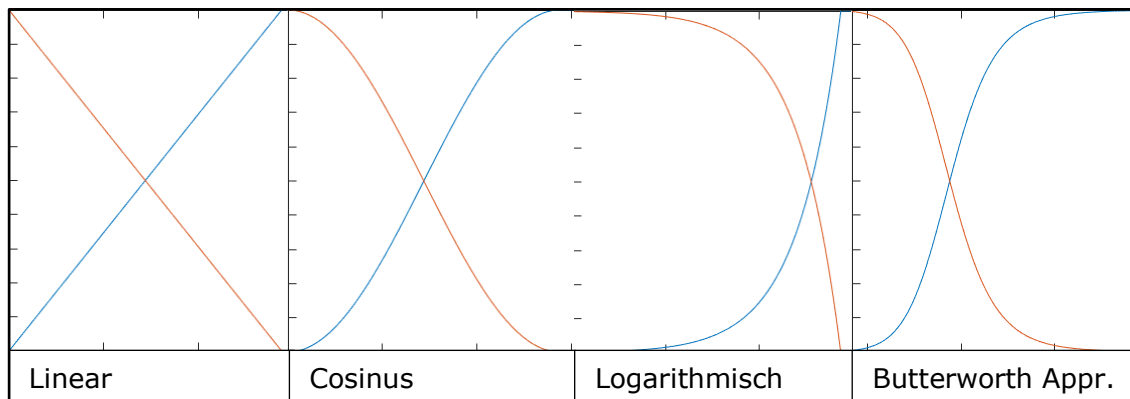


Abbildung 10: Rampenfunktionen

Mathematisch sind die Kurven wie folgt definiert, mit n als Rampendauer:

Linear: $f(x) = \frac{x}{n}$

Cosinus: $f(x) = -0.5 * \cos\left(\frac{\pi * x}{n}\right) + 0.5$

Logarithmisch: $f(x) = \frac{e^{\frac{n-x}{n}}}{n}$

Butterworth Approximation¹² (6.Grades): $f(x) = 1 - \frac{1}{1 + \frac{x^{12}}{2 * n}}$

Formel 2: Rampenfunktionen

¹² Wegen der asymptotischen Form dieser Funktion wurde die Rampendauer in der Formel verlängert und dafür die *Hold*-Zeit entsprechend verkürzt.

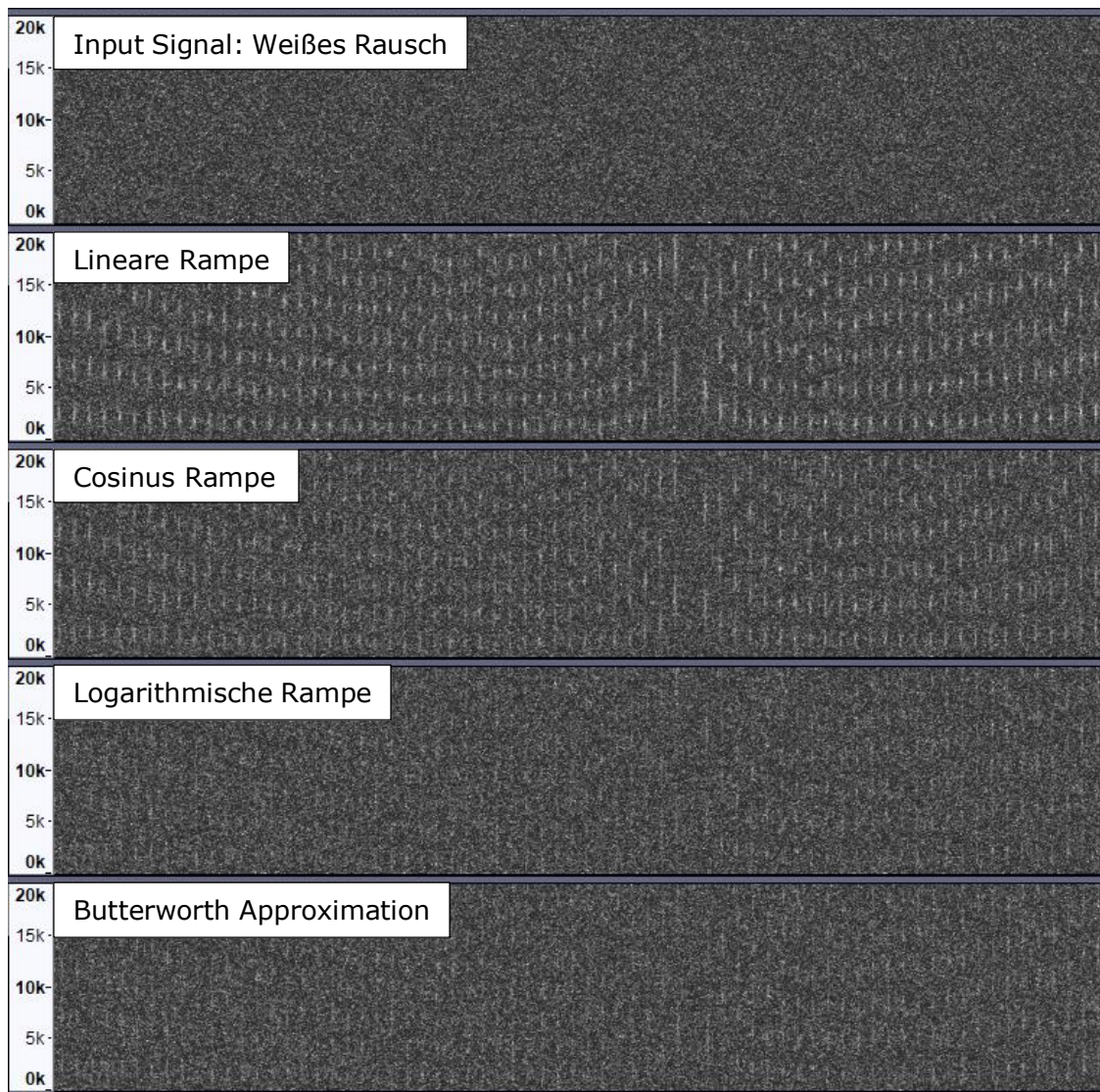


Abbildung 11: Sonogramm Rampenfunktionen; weißes Rauschen

Bei den Rampenformen, bei denen die Steigung in dem Bereich sehr hoch ist, in dem die Pegel beider Signale ähnlich sind, ist die Dauer der Kammfilter deutlich kürzer als bei Formen mit vergleichsweise niedriger Steigung. Sie sind dann nicht so deutlich hörbar. Dies wird bei dem Test mit weißem Rauschen deutlich sichtbar (siehe Abbildung 11). Die geringste Steigung hat die lineare Rampe. Bei ihr sind die Kammfilter am deutlichsten zu sehen. Bei der

logarithmischen Funktion und Butterworth-Approximation sind die Kammfilter nur von sehr kurzer Dauer. Jedoch zeigt der Test mit einem Sinussignal als Eingangssignal (siehe Abbildung 12), dass der Verlauf der logarithmischen Funktion sehr starke Seitenbänder hervorruft. Das liegt am nicht stetigen Verlauf der logarithmischen Rampe.

Die mathematische Funktion ist zwar stetig, jedoch endet die Rampe mit einer hohen Steigung. Ab dem Ende der Rampe wird die Einhüllende plötzlich linear zu $f(x) = 1$.

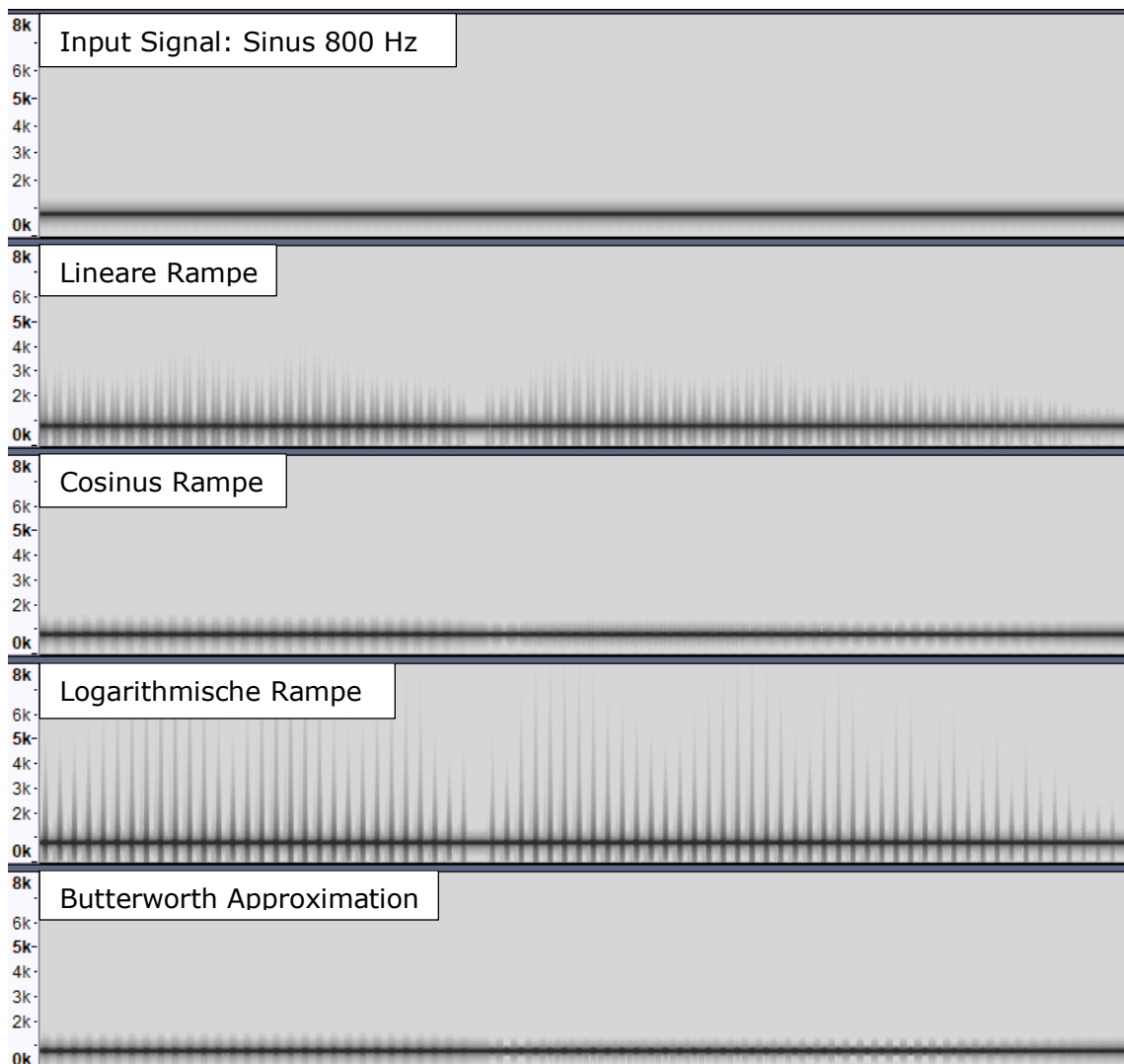


Abbildung 12: Sonogramm Rampenfunktionen; Sinus 800Hz

Der Übergang von Rampe zu Linearanteil der Einhüllenden ist nicht stetig, genau wie bei der linearen Rampe. Nur Rampenformen die mit der Steigung von 0 beginnen und enden, ergeben einen stetigen Verlauf der Einhüllenden.

Die beiden Rampenformen mit den geringsten Artefakten sind die Butterworth-Approximation und die Cosinusfunktion. Aus den beiden Sonogrammen Abbildung 11 und Abbildung 12 geht die Butterworth-Rampe als Favorit hervor. Eine direkte Gegenüberstellung mit einigen Testhörern ergab jedoch, dass die Unterschiede so gering sind, dass die Rampenformen akustisch nicht unterschieden werden können.

3.2 Sprung-Interpolation

Der Ansatz der Sprung-Interpolation sieht ein Umschalten der VZ vor, bei dem das Eingangssignal in dem Bereich des Phasensprungs durch ein interpoliertes stetiges Signal ersetzt wird.



Abbildung 13: Sprung-Interpolationsansatz

Dafür werden vom Ursprungssignals vier Werte ausgelesen, von denen jeweils zwei direkt aufeinander folgen. Das eine Wertepaar liegt vor, das andere hinter der Stelle im Signal, an der der Phasensprung auftritt. Beide stellen die Übergangsbereiche von Originalsignal und Interpolation dar, da sie in beiden Kurven gleich sind.

Die vier Punkte werden in die allgemeine Polynomgleichung

$$y = ax^3 + bx^2 + cx + d$$

Formel 3: Polynom der Sprung-Interpolation

eingesetzt und zu einer 4x4-Matrix zusammengefasst.

Aus dieser werden mithilfe des Gaußschen Eliminationsverfahrens die Variablen a , b , c und d ermittelt und daraus eine Kurve berechnet, die zum Zeitpunkt des Phasensprungs anstelle des Ursprungssignals wiedergegeben wird.

Für die Umsetzung dieses Ansatzes müssen alle Eingangssignale verzögert werden. Und zwar um die maximal einstellbare VZ und die Länge der errechneten Kurve. Diese Verzögerung ist nötig, da die oben genannten Samplewerte vor der Berechnung ausgelesen werden.

Wird die VZ geändert, werden die Samples ausgelesen. Sie bilden die y -Werte der in Formel 3 einzufügenden Punkte. Die x -Werte werden vorab generiert. Sie haben die Werte „-2“ und „-1“ beim ersten Wertepaar und „ $n+1$ “ und „ $n+2$ “ beim zweiten, wobei n die Länge der Interpolation in Samples ist. Es wird dann ein Puffer der Länge n mit der interpolierten Funktion befüllt.

3.2.1 Implementierung des Interpolationsalgorithmus mit Max 7

Wie auch beim Blendalgorithmus liegt dem Max-Patch der Sprung-Interpolation ein Ringpuffer zugrunde (siehe 2.5.1.1). Zusätzlich zu diesem Ringpuffer für das Eingangssignal wird ein weiterer Puffer für das interpolierte Signal erzeugt.

Die Realisierung der Sprung-Interpolation erfolgt mit zwei Ausgangskanälen, zwischen denen hart umgeschaltet wird. Der erste gibt das ankommende Signal, das im Ringpuffer gespeichert ist, wieder, der zweite das errechnete Interpolationssignal. Während Kanal eins das Signal mit der aktuellen VZ wiedergibt, kann eine Interpolation berechnet werden. Ist das erfolgt, kann zu dem zweiten Kanal umgeschaltet werden. Während der Wiedergabe des zweiten Kanals, kann im ersten die VZ angepasst werden. Nach der Wiedergabe des interpolierten Signals kann zum ersten Kanal umgeschaltet werden, der dann das Signal mit der Ziel-VZ wiedergibt.

Um das zu ermöglichen, muss das Signal im ersten Kanal zusätzlich zur VZ der Schallfeldreproduktion verzögert werden. Schließlich muss das Sample des Umschaltzeitpunkts, das den Anfang der Interpolation bildet, zur Berechnung abgegriffen werden, bevor es wiedergegeben wird. Ähnlich verhält es

sich mit dem Sample, das am Ende des Interpolationssignals steht. Es soll der Ziel-VZ entsprechen, muss aber zusätzlich zu der Verzögerung des Anfangssamples um die Länge des Interpolationssignals verschoben ausgelesen werden. Welche Samples zu den Zeitpunkten des Umschaltens am Kanal einliegen, wird zu Beginn berechnet. Sie werden direkt aus dem Ringpuffer ausgelesen. Dafür sind die beiden Variablen „*n*“ und „*compTime*“ nötig.

Die Länge des interpolierten Signals hängt von *n* ab. *compTime* kompensiert die Zeit, die für die Berechnung des Interpolationssignals benötigt wird.

Zur Signalverzögerung, wird vom Schreibindex des Ringpuffers die Summe aus den Variablen *n* und *compTime* und der maximalen VZ subtrahiert. Der neue Indexwert kommt in ein Subpatch names "*gen~ outpotato*". Dieser Patch hat vier Ausgänge.

Der erste sendet Steuerdaten an ein „*selector~*“-Objekt. Dieses Objekt funktioniert ähnlich zu *gswitch* (siehe 2.5.1) nur mit Signalen, d. h. am ersten Signaleingang kommt ein Steuersignal an, das festlegt welches Signal der anderen Eingänge an den Ausgang übergeben wird. Die anliegenden Signale, zwischen denen der *selector~* wechselt, sind das mit *index~* ausgelesene Ringpuffersignal und das Interpolationssignal.

Die übrigen drei Ausgänge von *gen~ outpotato* geben Indexsignale aus. Ausgang zwei sendet den Index der aktuellen VZ, Ausgang drei den der Ziel-VZ und der letzte den Index zum Auslesen aus dem Interpolationspuffer.

Die beiden Indexsignale aus den Ausgängen zwei und drei, sowie einige Nachrichtenkanäle, die die Variablen *n*, *compTime*, die Länge des Ringpuffers und die ZielVZ übergeben, münden in dem Subpatch, in dem die Berechnung des Interpolationssignals stattfindet. Dieser heißt „*p interpolation*“.

In diesem Interpolationssubpatch wird das Objekt „*snapshot~*“ verwendet. Dieses bekommt ein Signal und übergibt bei einer ankommenden *BANG*-Nachricht, den Samplewert des Zeitpunkts der Nachricht.

Damit werden der Index des Signals mit aktueller VZ und der Index der Ziel-VZ aus den Indexsignalen von *gen~ outpotato* ausgelesen. Der Index der Ziel-VZ wird dabei so gewählt, dass er auf das Sample im Signal zeigt, das um die Ziel-VZ und zusätzlich um die Länge des Interpolationssignals verzögert ist. Mit diesen Indexwerten und der Länge des Interpolationssignals werden die beiden Samplewerte vor und nach dem gesuchten Interpolationssignal aus dem Ringpuffer ausgelesen und zu vier Listen zusammengefasst. Diese Listen entsprechen den Reihen der 4x4-Matrix.

Ein Subpatch namens „*p lgs-Gauss-2*“ berechnet aus diesen Listen die vier Variablen der Formel 3, mit denen ein weiter Subpatch namens „*p poker*“ das Interpolationssignal berechnet. Das Signal wird in *p poker* mit dem Objekt „*poke~*“ in den Interpolationsspeicher geschrieben.

Wenn die VZ geändert wird, schaltet *gen~ outpotato* nach Ablauf der *compTime* durch das *selector~* Objekt den Signalausgang vom Ringpuffersignal zum Interpolationssignal um. In dem Zeitraum in dem das Interpolationssignal wiedergegeben wird, ändert sich die VZ des Index, der das Signal mit der aktuellen VZ aus dem Puffer ausliest. Sobald das interpolierte Signal wiedergegeben wurde, schaltet der Patch wieder auf das Ausgeben des Ringpuffersignals zurück, dessen VZ dann zur Ziel-VZ umgeschaltet wurde.

3.2.2 Artefakte der Sprung-Interpolation

Die bei diesem Ansatz hervorgerufenen Artefakte hängen von der Länge n der Interpolationsfunktion und den zum Zeitpunkt enthaltenen Frequenzen des Originalsignals ab. Generell ist festzustellen, dass ein Teil des Originalsignals geht verloren und wird durch ein Signal ersetzt, das ein anderes Frequenzspektrum aufweist. Je länger n ist, desto tiefer ist die höchste enthaltene Frequenz. Bei einem Signal mit hohen Frequenzanteilen erklingt ein tiefes „Pop“-Geräusch, wenn n zu groß gewählt wurde. Außerdem nimmt der Pegel des errechneten Signals mit zunehmendem n zu und es steigt die Wahrscheinlichkeit, dass das Signal übersteuert.

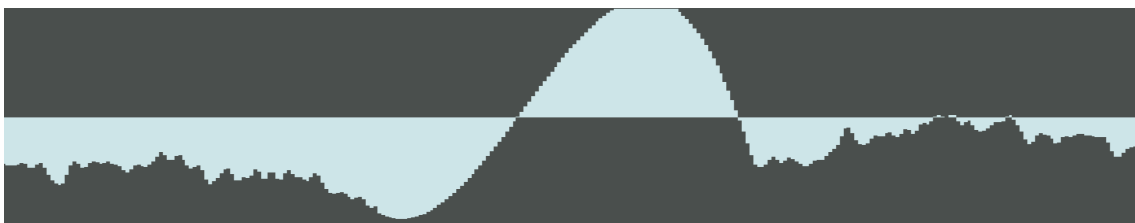


Abbildung 14: Interpolation mit zu großem n führt zur Übersteuerung

Bei Signalen mit ausschließlich niedrigen Frequenzanteilen werden die Artefakte mit abnehmendem n deutlicher wahrnehmbar, da dann hohe Frequenzanteile zu hören sind, die vorher nicht im Signal enthalten waren. Der durchschnittliche Pegel des hinzugefügten Signals nimmt mit abnehmendem n ab, sodass diese hohen Frequenzen in den meisten Fällen nicht so deutlich wahrgenommen werden, wie die oben genannten „Pop“-Geräusche bei zu großem n .

In dem Fall, in dem die Samplewerte der beiden Wertepaare weit auseinanderliegen und somit ein großer Phasensprung überbrückt wird, ruft eine Interpolation mit kleinem n wiederum deutliche Artefakte hervor. Denn dann wirkt sich eine Interpolation ähnlich wie ein Phasensprung aus.

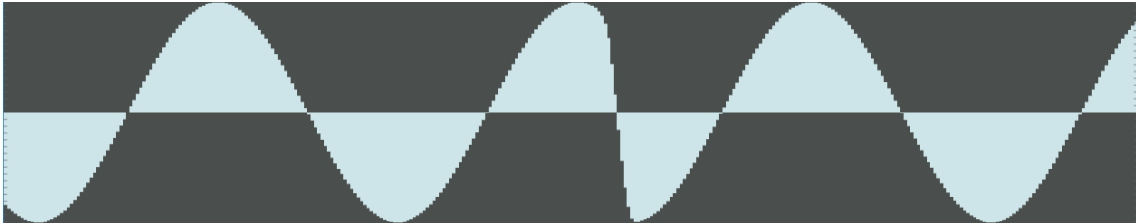


Abbildung 15: Interpolation mit kleinem n

Die zum Ursprungssignal hinzugefügten spektralen Anteile werden geringer, je ähnlicher das interpolierte Signal der Form einer Blende (siehe 0) wird. Das bedeutet eine Optimierung der Sprung-Interpolation würde eine Annäherung an den Ansatz der Blende bedeuten. Da der Ansatz der Interpolation jedoch mit deutlich höherem Rechenaufwand verbunden ist als die Blende und da er eine zusätzliche Latenz mit sich bringt ist, eignet er sich nicht für einen Einsatz in der Praxis.

Aus diesem Grund wurde dieser Algorithmus im Prototyp, mit dem die Hörversuche durchgeführt wurden, nicht umgesetzt.

3.3 Slide

Dieses Kapitel beschreibt einen neuen Algorithmus, der erst im Rahmen dieser Arbeit entstand und der sich als praxistauglich entpuppte.

Das Prinzip des Sildes entspricht dem einer Bandmaschine mit einem beweglichen Lesekopf, der bei jeder Änderung der VZ während der Wiedergabe zur neuen Position über das Band gleitet.

Abbildung 16 zeigt eine Visualisierung des Slidealgorithmus und eines Phasensprungs der selben VZ-Änderung.

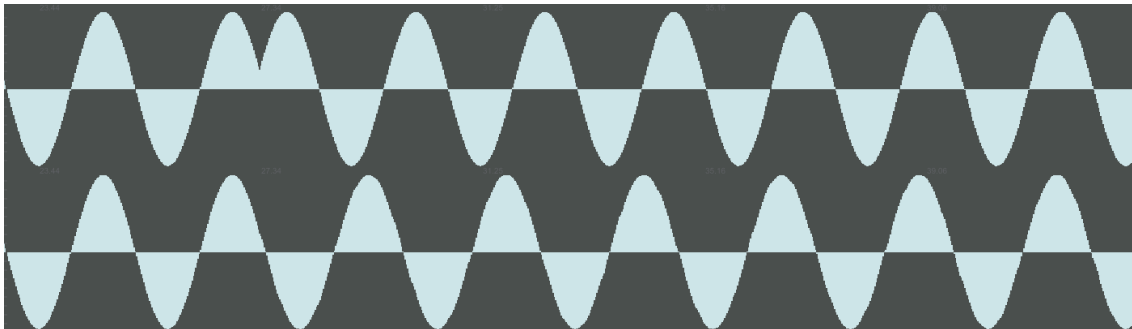


Abbildung 16: Slidealgorithmus

In einem analogen System wie dem der Bandmaschine würde dieser Ansatz zu einer Frequenzverschiebung ähnlich der eines Dopplereffekts führen. Wenn sich der Lesekopf in Richtung des Schreibkopfes hinbewegt, entsteht eine Frequenzverschiebung nach oben, wenn er sich entfernt eine nach unten.

In einem digitalen System ist aufgrund der Quantisierung des Signals in Samples, eine exakte Nachbildung dieses Bandmaschinenprinzips nicht möglich. Die in Kapitel 3.3.3 erläuterten Optimierungsansätze dienen der Minderung der Artefakte, die durch die Quantisierung entstehen.

In diesem Abschnitt wird zunächst der Slide-Algorithmus ohne Optimierung vorgestellt, der mit verhältnismäßig wenig Rechenressourcen auskommt.

Bis die VZ eines Signals ihren gewünschten Wert erreicht, wird sie in regelmäßigen Abständen schrittweise angepasst, indem einzelne Samples entweder übersprungen oder wiederholt werden.

3.3.1 Implementierung des Slidealgorithmus mit Max 7

Die Grundlage der Implementierung bildet auch in diesem Fall der Ringpuffer aus Kapitel 2.5.1.1. Ebenso sind der *slider* zur VZ-Änderung und der angrenzende *p sliderToValue* Subpatch identisch zur Implementierung der Sprung-Interpolation.

Das Prinzip des Algorithmus ist es, durch eine schrittweise Anpassung des Index, mit dem aus dem Ringpuffer ausgelesen wird, einzelne Samples zu überspringen oder zu wiederholen.

Der Kern des Patches ist ein Subpatch namens „gen~ slideTimeG“. Mit ihm wird der Index zum Auslesen aus dem Ringpuffers berechnet. Er hat einen Eingang, an dem das Signal mit dem Schreibindex anliegt. Zusätzlich bekommt er über Parameternachrichten die Länge des Ringpuffers für die Modulo-Berechnung vor dem Auslesen, die Ziel-VZ und eine Variable „j“ übermittelt. *j* stellt den Intervallabstand der Slideschritte dar, also die Anzahl an Samples die wiedergegeben wird, bis eines übersprungen oder wiederholt werden soll.

Das ankommende Signal mit den Indexwerten zum Auslesen aus dem Ringpuffer durchwandert ein Subtraktionsobjekt („-“) und ein Moduloobjekt („mod“) bevor es wieder ausgegeben wird. Der Modulowert ist die Länge des Ringpuffers, die als Parameternachricht an den Subpatch übermittelt wird.

Der Wert, der vom Index subtrahiert wird, kommt aus einer kurzgeschlossenen Schleife des Subpatches. Solche Schleifen werden mit einem *gen~* Objekt namens „*history*“ ermöglicht. Dieses verzögert ein ankommendes Signal

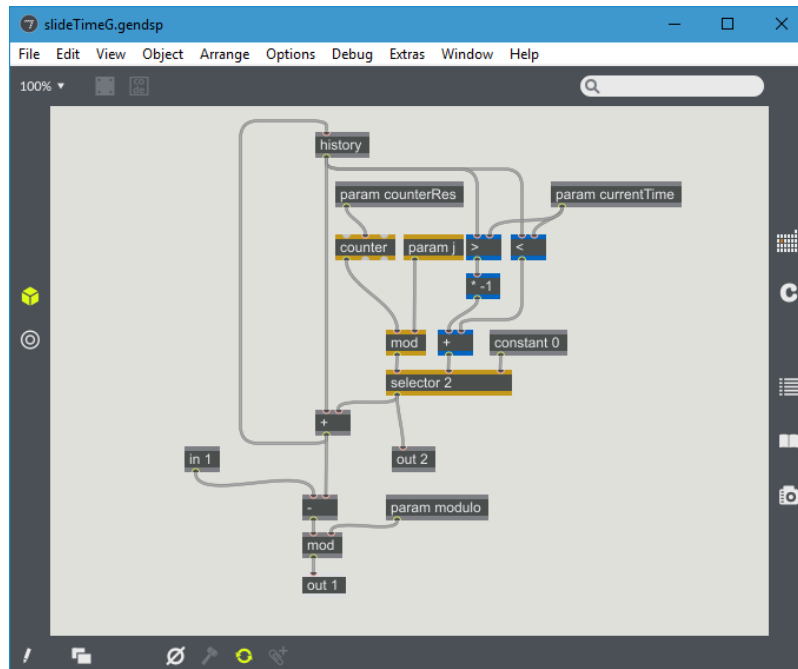


Abbildung 17: der zentrale Silde-SubPatch *gen~ slideTimeG*

um die Dauer von einem Sample. Damit kann das Sample, das als Ergebnis aus einer Rechnung hervorgeht, bei der Berechnung des nachfolgenden Samples mitverwendet werden.

Durch dieses kurzgeschlossene *history*-Objekt läuft das Ergebnis, der letzten Berechnung. Dabei handelt es sich um die VZ des Signals am Ausgang des Programms, die bei Programmstart null beträgt. Sie durchläuft Objekte mit logischen Operationen durch die abgefragt wird, ob sie größer oder kleiner als die Ziel-VZ ist. Das Ergebnis dieser Abfragen ist „0“, wenn aktuelle VZ und Ziel-VZ identisch sind, „1“ wenn die Ziel-VZ größer und „-1“ wenn die aktuelle VZ größer ist. Es wird an den zweiten Eingang eines *selector*-Objekt geschickt.

Dieses *gen~*-Objekt funktioniert genau wie das MSP-Objekt *selector~*; ankommende Werte des ersten Eingangs steuern, welches der an den anderen Eingängen anliegenden Signalen an den Ausgang übergeben wird. Neben dem Ergebnis der Logikabfragen kann der *selector* in diesem Fall nur die Zahl 0 übergeben.

Gesteuert wird das *selector*-Objekt vom Ausgangssignal eines Zählerobjekts („*counter*“), das durch ein Moduloobjekt („*mod*“) manipuliert wird. Sein Berechnungswert ist *j*. Durch die Modulorechnung nimmt das Signal des Zählers aufsteigende ganzzahlige Werte von null bis *j* an. Sobald der *selector* eine 1 bekommt, gibt er das Ergebnis der Logikabfrage aus, andernfalls eine 0.

Der Ausgang aus dem *selector*-Objekt wird mit der aktuellen VZ der Kurzschlusschleife addiert. Somit nähert sich die aktuelle VZ in Intervallen von *j* Samples der Ziel-VZ an.

Diese aktuelle VZ der Schleife wird von jedem Wert des Schreibindex des Ringpuffers subtrahiert. Damit wird der Index schrittweise auf seine neuen Ziel-VZ-Werte hinbewegt, was beim Auslesen aus dem Puffer ein Überspringen bzw. ein Wiederholen einzelner Samples zur Folge hat.

Zusätzlich zum Leseindex gibt *gen~ slideTimeG* in einem zweiten Ausgang die Ausgabe des *selector*-Objekts aus, die zu der aktuellen VZ addiert wird. Dieser Ausgang stellt ein Steuersignal da, aus dem hervorgeht, ob und wann ein Slideschritt passiert und in welche Richtung, also ob die VZ durch Wiederholung eines Samples verlängert oder durch Überspringen verkürzt wurde. Dieses Signal wird für einen Subpatch benötigt, der eine Optimierung durch einen Glättungsansatz betreibt (siehe 3.3.3 Optimierung des Slidealgorithmus). Für die Glättung ist es notwendig, dass der Zeitpunkt des Slideschrittes übermittelt wird.

3.3.2 Artefakte des Slidealgorithmus

Die Anwendung des Slides führt zu unterschiedlichen Artefakten. Zum einen kommt es wie beim analogen Beispiel zu einer Frequenzverschiebung. Sie hängt davon ab, wie schnell die VZ angeglichen wird, bzw. wie häufig Samples wiederholt oder übersprungen werden. Daneben treten bei jedem Anpassungsschritt verzerrungsartige Artefakte auf. Sie hängen vorrangig von der Steigung der Signalkurve zum Zeitpunkt der Wiederholung bzw. des Überspringens einzelner Samples ab und sind mit einem Quantisierungsrauschen vergleichbar.

3.3.2.1 Frequenzverschiebung

Wie weit die Tonhöhe verändert wird, hängt von der Differenz zwischen der aktuellen und Ziel-VZ ab und somit von der Geschwindigkeit, mit der sich ein Objekt bewegt. Sie lässt sich aus dem Intervall zwischen den übersprungenen oder wiederholten Samples berechnen. Daraus kann abgeleitet werden, wie schnell sich eine reale Schallquelle bewegen müsste, damit ihr Signal bei einem natürlichen Dopplereffekt dieselbe Tonhöhenverschiebung erfahren würde.

Ein Sinussignal dessen Periodendauer genau dem Intervallabstand j entspricht, würde beim Überspringen eine Verschiebung zu der Frequenz des Signals der Periodendauer $j-1$ erfahren. Diese Frequenzen sind

$$f_0 = \frac{r}{j}$$

$$f_{new} = \frac{r}{j-1}$$

mit j =Intervallabstand in Samples und r = Samplerate in Hz.

Die allgemeine Formel des Dopplereffekts eines bewegten Senders und stehenden Empfängers lautet

$$\frac{c}{f_e} = \frac{c}{f_s} - \frac{v}{f_s}$$

mit f_e als Frequenz beim Schallempfänger und f_s als Frequenz beim Sender.

Nach der Geschwindigkeit umgeformt ergibt dies

$$v = \left(\frac{c}{f_s} - \frac{c}{f_e} \right) * f_s$$

mit $f_s = f_0$ und $f_e = f_{new}$

$$v = \left(\frac{c}{f} - \frac{c}{f_{j-1}} \right) * \frac{r}{j} ,$$

was sich zu

$$v = \frac{c}{j}$$

vereinfachen lässt.

Formel 4: reaktive Bewegungsgeschwindigkeit eines Slides

Mit dieser Formel lässt sich die Geschwindigkeit berechnen, mit der sich eine reale Quelle auf ihren Empfänger zubewegen müsste, damit ihr Signal durch den Dopplereffekt dieselbe Frequenzverschiebung erfährt wie durch den Slide. Damit kann ein Bezug hergestellt werden, welche reale Geschwindigkeit dem Slide entspricht.

Eine Erkenntnis durch diese Formel ist, dass es nicht zielführend ist, das Intervall j mit der Größe des Audiopuffers des Renderingcomputers gleichzusetzen. Wenn z. B. bei einem Puffer der Länge 16 Samples, bei jedem Auslesen ein Sample übersprungen wird, bis die Ziel-VZ erreicht ist, entspräche das dem Dopplereffekt einer Quelle, die sich mit ca. 21,4 m/s (ca. 77 km/h) auf den Zuhörer zubewegt.

j	16	32	64	128	256	512
v in m/s	21,4375	10,71875	5,359375	2,6796875	1,33984375	0,66992188
v in km/h	77,175	38,5875	19,29375	9,646875	4,8234375	2,41171875

Tabelle 2: Geschwindigkeit eines Dopplereffekts mit der gleichen Frequenzverschiebung wie beim Slide mit j

Folglich, sollte die Geschwindigkeit, mit der sich die aktuelle VZ der Ziel-VZ annähert, nicht fest vorgegeben sein, sondern abhängig von der Objektgeschwindigkeit bei jeder Positionsaktualisierung neu berechnet werden. Dies kann am einfachsten stattfinden, wenn bei einem festen zeitlichen Aktualisierungsraster die Geschwindigkeit so gewählt wird, dass die Ziel-VZ immer zum Zeitpunkt der nächsten Aktualisierung erreicht wird. Damit ist gesichert, dass stets die geringstmögliche Frequenzverschiebung auftritt und die VZ nicht wie bei einer zu geringen Geschwindigkeit verspätet ihr Ziel erreicht.

Außerdem ist das Einfügen einer Haltezeit, anders als bei der Blende, beim Slideansatz nicht ratsam. Denn dann ändert sich die Tonhöhenverschiebung während einer längeren Bewegung andauernd, was einem sehr schnellen Vibratoeffekt gleichkommt. Stattdessen wird beim Slidealgorithmus die Geschwindigkeit, so festgelegt, dass die Ziel-VZ genau dann erreicht wird, wenn eine neue Position und somit eine neue Ziel-VZ feststeht. Die Geschwindigkeit der VZ-Änderung ($v_{\Delta t}$) ist demnach

$$v_{\Delta t} = \frac{\Delta VZ}{t_a}$$

mit t_a als Zeitintervall der Positionsaktualisierung.

Das führt dazu, dass die Lokalisationsschärfe steigt. Denn im Unterschied zum Ansatz der Blende „springt“ das Klangobjekt nicht von einer Position zur nächsten. Stattdessen kommt es auf Grund des „auf den Zielwert zu Bewegens“ zu einer Art linearen Positionsinterpolation und damit zu einer „flüssigeren“ Bewegung.

Ein Problem mit der Tonhöhenverschiebung im Verhältnis zum natürlichen Dopplereffekt tritt dann auf, wenn sich ein Objekt nicht auf den Zuhörer zu oder von ihm wegbewegt, sondern sich auf einer horizontalen Bahn z. B. auf einer Bühne von links nach rechts befindet. Denn bei einem natürlichen Dopplereffekt hängt die wahrgenommene Frequenzverschiebung von der relativen Bewegung und somit vom Abstand des Empfängers vom Sender ab. Bei einem Wiedergabesystem wäre die wahrgenommene Verschiebung bei jedem Zuhörer unabhängig von seiner Entfernung zur virtuellen Quelle gleich.

Der Vorteil der Frequenzverschiebung liegt vor allem im Unterschied zum Kammfilter der Blende daran, dass sie von der Bewegungsgeschwindigkeit abhängt und darum bei langsamen Bewegungen sehr gering ist. Bei einem Objekt, das sich so langsam bewegt wie ein Sprecher der auf einer Bühne geht, sind sie kaum wahrnehmbar. Erst bei schnelleren Bewegungen wird die Frequenzverschiebungen zum deutlichen Artefakt.

Dieses Verfahren eignet sich aus einem weiteren Grund besonders beim Fallbeispiel des Sprechers. Im Unterschied zu tonalen Signalen ist das Obertonspektrum der gesprochenen Sprache sehr unregelmäßig. Durch das ständige Ändern der spektralen Zusammensetzung der Foneme, fällt es schwer geringe Tonhöhenverschiebungen überhaupt wahrzunehmen. Musikalische Signale, bei denen eine größere Regelmäßigkeit im Signal festzustellen ist, sind anfälliger für solche Artefakte. Denn bei solchen Signalen wird beim Hören eine Konstanz der Obertöne erwartet.

3.3.2.2 Verzerrungsartige Artefakte

Neben der Tonhöhenverschiebung kommt es beim Slidealgorithmus noch zu verzerrungsartigen Artefakten. An der Stelle des Signals, an der das Sample eingefügt oder übersprungen wird, entsteht ein kleiner Knick in der Wellenform, wie in Abbildung 18 zu sehen.

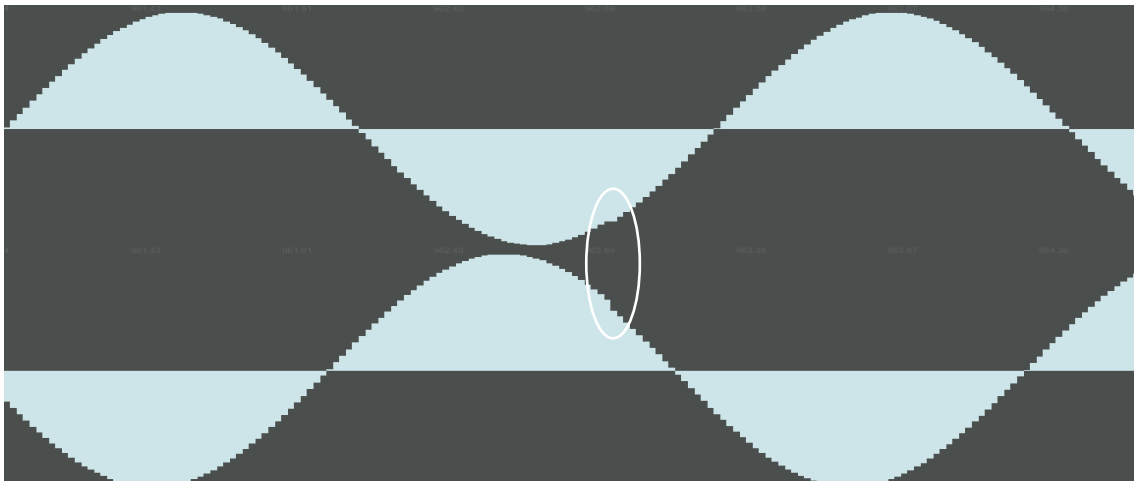


Abbildung 18: Knicke des Slideansatzes

Wie deutlich dieser hörbar ist, ist abhängig von der Steigung der Signalkurve zu diesem Zeitpunkt. Die Steigung hängt vom Pegel und dem Verhältnis aus Frequenzanteilen und Samplerate ab. Bei einer geringen Steigung, z. B. bei einem tieffrequenten Signal, einem Signal mit geringer Amplitude oder einer hohen Samplerate, sind diese Artefakte weniger stark wahrnehmbar als bei Signalen mit großer Steigung. Je regelmäßiger ein Signal ist, je weniger verschiedene Frequenzen es enthält und je höher die Frequenzanteile im Verhältnis zur Samplerate sind, desto deutlicher fallen die Störeffekte auf. Bei Signalen mit einem Grundrauschen, wie es bei Mikrofonsignalen häufig der Fall ist, sind diese Knick-Artefakte schwächer wahrnehmbar, da sie vom Rauschen überdeckt werden.

3.3.3 Optimierung des Slidealgorithmus

Dieser Abschnitt behandelt Ansätze zur Minderung der Artefakte aus Kapitel 3.3.2.2. Ein Verfahren mit dem die Frequenzverschiebung verringert werden kann, ist derzeit nicht bekannt.

Im Rahmen dieser Arbeit wurden zwei Optimierungsansätze untersucht, die sich in Rechenaufwand und Grad der Artefaktminimierung unterscheiden. Sie zielen beide darauf ab, die Knicke in der Signalkurve, die bei jedem Slide-schritt entstehen, zu glätten. Sie heißen Mittelwertglättung und Blendglättung.

3.3.3.1 Mittelwertglättung

Die Mittelwertglättung ist ein äußerst einfaches Verfahren, dessen zusätzlicher Rechenaufwand sehr gering ist. Es führt zu einer zusätzlichen Latenz, die aber nur wenige Samples beträgt und vernachlässigt werden kann.

Ziel der Optimierung ist es, die gleichen Samples einer Wiederholung auseinander zu bewegen und die weiter voneinander entfernten Samples eines Überspringens anzunähern. Da sich der Slideansatz bereits auf Sampleebene abspielt, ist ein zeitliches Entfernen oder Annähern nicht möglich. Die Werte werden stattdessen durch eine Mittelwertberechnung nach oben oder unten angepasst. Dabei werden sie als Mittelwert zwischen dem vergangenen und dem zukünftigen Sample neu berechnet (siehe Abbildung 19).

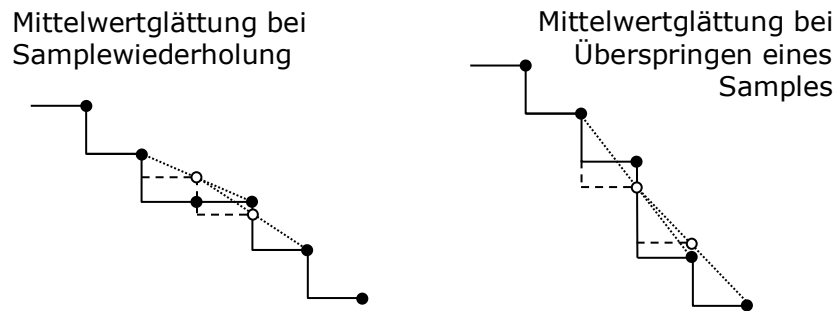


Abbildung 19: Mittelwertglättung des Slidealgorithmus

Dazu muss das Signal um zwei Samples verzögert werden, damit der Wert der zukünftigen Samples bei der Berechnung bekannt ist. Diese zusätzliche Latenz ist jedoch so gering, dass sie vernachlässigt werden kann.

Die zusätzlich benötigten Rechenressourcen fallen ebenfalls äußerst gering aus, da der Slidealgorithmus nur um zwei Mittelwertberechnungen erweitert wird.

Die Mittelwertglättung wirkt sich auf die verzerrungsartigen Artefakte wie ein Tiefpassfilter aus (siehe Abbildung 20). Durch das Herausfiltern der hohen Frequenzanteile werden die Artefakte weniger stark wahrgenommen. Bei Eingangssignalen mit breitem Frequenzspektrum, vor allem bei Signalen mit starkem Grundrauschen, kann die Mittelwertoptimierung ausreichen um diesen Teil der Artefakte auf ein nicht wahrnehmbares Niveau abzusenken.

Je schmalbandiger das Eingangssignal ist und je höher die enthaltenen Frequenzanteile sind, desto deutlicher werden die Artefakte. Bei sehr hohen Frequenzen, bei denen eine Halbwelle nur wenige Samples lang ist, wirkt sich die Mittelwertberechnung zudem nicht mehr als Glättung aus. Ein 12 kHz Sinuston hat beispielsweise eine Halbwelle mit der Länge von zwei Samples. Wenn dort zwei Mittelwerte berechnet werden, wird die Form des Signals grundlegend verändert.

Zusammengefasst ist die Mittelwertglättung ein mögliches Verfahren, mit dem die Artefakte durch minimalen zusätzlichen Rechenaufwand geringfügig verringert werden können.

Dem gegenüber steht mit der Blendglättung eine Optimierungsmethode, mit der die verzerrungsartigen Artefakte aus Kapitel 3.3.2.2 auf ein äußerst geringes Maß reduzieren werden können, sodass sie praktisch nicht mehr wahrgenommen werden. Der Rechenaufwand dieser Optimierung ist jedoch ungleich größer.

3.3.3.2 Blendglättung

Bei der Blendglättung handelt es sich um ein Glättungsverfahren, welches den VZ-Änderungsansatz der Blende aufgreift. An jedem Slideschritt, also immer dann wenn ein Sample übersprungen oder wiederholt werden soll, wird eine Blende mit der Dauer von 24 Samples durchgeführt. Dies führt zu einer fast vollständigen Glättung der Kurve, also einer starken Reduzierung der spektralen Artefakte (siehe Abbildung 20).

Die Kammfilter die durch das Überblenden entstehen, werden praktisch nicht wahrgenommen und können vernachlässigt werden. Das liegt daran, dass immer Signale mit der Differenz von einem Sample überblendet werden. Bei einer Samplerate von 48 kHz bedeutet das, dass die erste Auslöschung bei 24 kHz also außerhalb des Hörspektrums liegt. Damit wirkt sich der Filter im hörbaren Bereich wie ein Hi-Shelf-Filter mit geringer Güte aus.

Verglichen mit der Mittelwertberechnung sind Rechenaufwand und Speicherbedarf hierbei deutlich höher. Zudem wächst beides mit der VZ-Änderung, die der Ansatz in einem Aktualisierungsintervall vollzieht. Denn je weiter die Ziel-VZ von der aktuellen entfernt ist, desto häufiger müssen Blenden eingesetzt werden.

Dennoch wird der Slidealgorithmus erst durch dieses Optimierungsverfahren praxistauglich. Deshalb ist diese Version des Slides die, die beim Hörversuch getestet wurde.

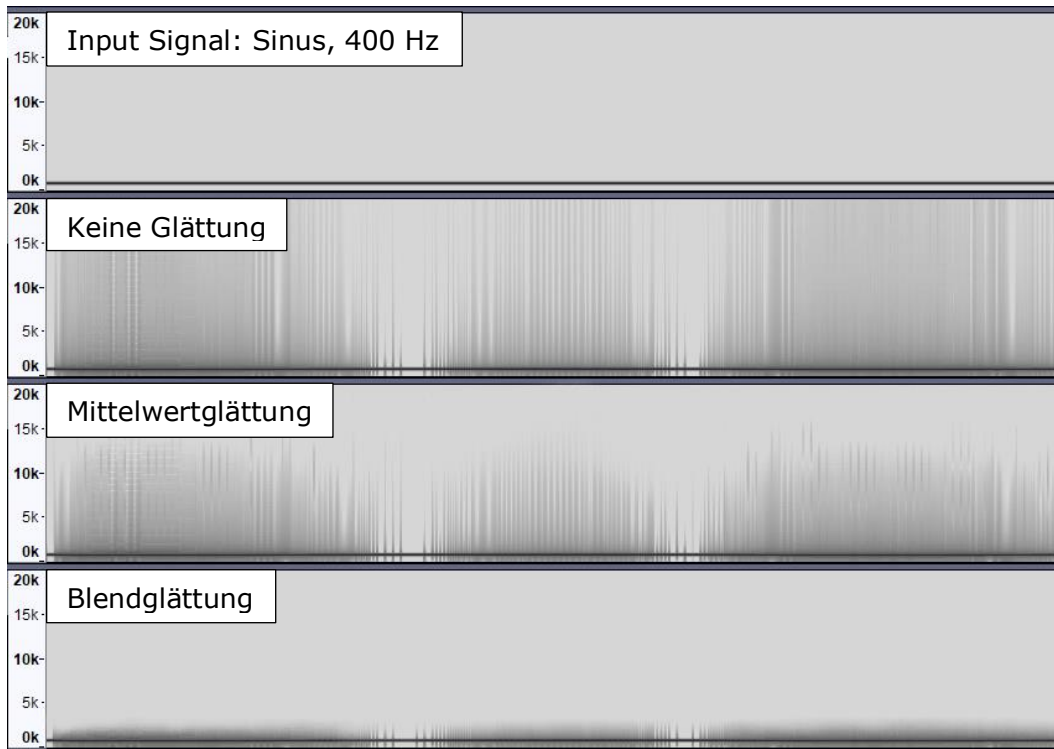


Abbildung 20: Sonogramm Slideoptimierung

3.4 Kombinationen der Ansätze

Neben der Optimierung des Slideansatzes durch das Verwenden von kurzen Blenden, wurden auch weitere Kombinationsmöglichkeiten beider Ansätze untersucht. Jedoch stellen diese oft keine Verbesserung dar. Der folgende Abschnitt diskutiert Kombinationen aus Slide- und Blendalgorithmus. Dabei werden zunächst zwei Überlegungen thematisiert, die sich als nicht zielführend entpuppten und schließlich wird eine Verfahrensweise präsentiert, die für die praktische Anwendung zur Anpassung der VZ empfohlen wird.

3.4.1 Parallelansatz

Beim Parallelansatz werden zeitgleich Slide und Blende durchgeführt, indem eines oder beide Blendsignale einen Slide vollziehen. Da der zeitliche Versatz der beiden Signalanteile dabei nicht konstant ist, entsteht kein Kammfilter wie er in Kapitel 3.1.1 thematisiert wird. Auch die Frequenzverschiebung kann geringer ausfallen, da die Slidebewegung nicht von der aktuellen zur Ziel-VZ vollzogen werden muss. Wenn beispielsweise das eingeblendete Signal den Slide vollzieht, könnte es bei der Hälfte oder beim letzten Drittel der VZ-Differenz starten.

Dieser Ansatz führt jedoch zu deutlichen Artefakten. Eine praktische Umsetzung ist daher nicht zielführend. Denn jeder der beiden verwendeten Algorithmen ruft alleine weniger Störeffekte hervor. Diese Form der Kombination bedeutet somit eine Verschlechterung.

Durch die Tonhöhenverschiebung kommt es beim Überblenden statt zu statischen Kammfiltern zu Schwebungen.

Eine Schwebung tritt immer dann auf, wenn zwei Schwingungen sumiert werden, deren Frequenz sich gering unterscheidet. Dann kommt es durch das Verschieben der relativen Phasenlage zu einer periodischen Zu- und Abnahme

der Amplitude.¹³ Das Ergebnis kommt einer Modulation des Signals gleich, zu der die Frequenzverschiebung als zusätzliches Artefakt hinzukommt.

Wenn die Differenz der Frequenzen groß genug ist, wird zwar keine Schwebung mehr wahrgenommen, dann kommt es aber zu einem Choruseffekt.

3.4.2 Multi-Short-Blend

Dem Multi-Short-Blend-Ansatz liegt die Idee zu Grunde, eine Mitte zwischen Slide und Blende zu finden. Die Blende (siehe Kapitel 3.1) stellt auf der einen Seite das einmalige Ändern der VZ pro Aktualisierung der Position bzw. der Ziel-VZ dar und führt zu Kammfiltern ohne Frequenzverschiebung. Auf der anderen Seite handelt es sich beim Slide (siehe Kapitel 3.3) um ein häufiges Blenden für jede Aktualisierung ohne Kammfilter aber mit Frequenzverschiebung.

Das Ziel ist ein Bereich zwischen beiden Ansätzen, in dem die beiden Artefakttypen Frequenzverschiebung und Kammfilter auftreten, dies jedoch nur in geringem Ausmaß.

Ein Ansatz zwischen beiden Seiten ist in diesem Fall ein Algorithmus, der ähnlich wie der Slide zwischen der Positionsaktualisierung in regelmäßigen Abständen die VZ schrittweise anpasst, pro Anpassungsschritt aber mehr als ein Sample überbrückt. Dies geschieht indem jedes Mal eine kurze Blende durchgeführt wird. Dadurch kann die Anzahl der Anpassungsschritte reduziert

¹³ (Meschede, 2004)

und damit der Abstand zwischen den Schritten vergrößert werden. Gleichzeitig ist der zeitliche Versatz der Blendsignale geringer und somit die Ordnung der Kammfilter niedriger.

Dennoch ist auch diese Form der Kombination nicht zielführend. Auch wenn die Ordnung der Kammfilter kleiner ist als bei einer einmaligen Blende, ist sie groß genug, um als Artefakt deutlich hörbar zu sein. Die Problematik besteht in diesem Fall darin, dass sie viel häufiger auftreten und die klangliche Stabilisierung durch das „Ramp and Hold“ Verfahren (siehe Kapitel 3.1.3.1) entfällt. Außerdem können durch die kurzen Rampendauern spektrale Seitenbandeffekte auftreten.

Eine klangliche Verbesserung entsteht nur, wenn die Schrittgröße der einzelnen Slideschritte, also der Versatz der Blendsignale, bei einer Samplerate von 48 kHz auf ein Sample reduziert wird. Bei der niedrigsten Auslöschungsfrequenz von 24 kHz, ist die Ordnung des Kammfilters klein genug. Das Ergebnis liegt dann aber nicht zwischen beiden Ansätzen. Stattdessen ergibt sich ein Slide, der äußerst wenige verzerrungsartige Artefakte hervorruft. Das entspricht genau dem Optimierungsansatz der Blendglättung (siehe Kapitel 3.3.3.2).

3.4.3 Grenzwertansatz

Abgesehen von der Optimierung durch die Blendglättung sollte, wie aus den zuvor diskutierten Kombinationen hervorgeht, zum selben Zeitpunkt immer nur entweder ein Slide oder eine Blende vollzogen werden. Dennoch ist eine Kombinationslösung sinnvoll.

Denn die Artefakte des Slides, die von der Differenz zwischen aktueller VZ und Ziel-VZ abhängen, bilden sich bei langsam bewegten Objekten weniger, bei schnell bewegten Objekten deutlich stärker aus, als die der Blende. Darum wird beim Grenzwertansatz für das Verwenden des Slides ein Maximalwert

der Objektgeschwindigkeit festgelegt. Unterhalb dieses Wertes wird die VZ-Änderung mit dem Slideansatz und oberhalb mit dem Ansatz der Blende vollzogen.

Dabei ist der Grenzwert nicht direkt an die Geschwindigkeit gebunden mit der die virtuelle Schallquelle bewegt wird. Stattdessen wird ein Maximalwert für die Differenz der VZ festgelegt. Denn sie alleine ist ausschlaggebend für das Maß in dem die Tonhöhe verschoben wird. Wenn eine virtuelle Schallquelle beispielsweise an einer Sekundärquelle vorbeibewegt wird, ist die VZ-Änderung bei gleicher Geschwindigkeit größer, wenn der Abstand von Primär- und Sekundärquelle klein ist, als bei einem großen Abstand.

Der Vorteil der Verwendung einer Blende bei großen Positionsänderungen entsteht dann, wenn auch eine Pegelberechnung stattfindet. Denn dann blendet ein Signal mit geringem und kaum wahrnehmbarem Pegel in ein Signal mit hohem Pegel über.

Die Positionsinterpolation, die der Blende gegenüber dem Slide fehlt, stellt bei großen Bewegungen kein Problem dar. Da unser Gehör mit der Lokalisation von Quellen mit großen Geschwindigkeiten ohnehin Probleme hat, nimmt es keinen Unterschied zwischen einer fließenden und einer springenden Bewegung wahr.

Den Slideansatz bei kleinen Positionsänderungen zu verwenden macht neben der Positionsinterpolation auch deshalb Sinn, weil die Frequenzverschiebung solange sie gering genug ist, nur sehr schwach manchmal sogar gar nicht wahrgenommen wird.

3.5 Fazit der Lösungsansätze

In diesem Kapitel wurden drei Lösungsansätze diskutiert. Es zeigte sich, dass ein Ansatz nicht praxistauglich ist. Dies ist der Ansatz der Sprung-Interpolation. Dagegen erweisen sich die Ansätze des Slides und der Blende als vielversprechend.

Die Blende, die bei einigen Systemen, wie beispielsweise *d&b Soundscape En-Scene*, Anwendung findet, überblendet bei jeder Positionsaktualisierung mit einem Crossfade vom Signal mit der bisherigen zum Signal mit der neuen VZ. Dieser Ansatz hat Kammfilterartefakte und spektrale Seitenbandeffekte zur Folge. Diese sind bei einer Cosinus- oder Butterworthrampe der Dauer von 35 ms und einer anschließenden „*Holdphase*“ von 50 ms am geringsten wahrnehmbar.

Neben diesem entstand im Rahmen dieser Arbeit ein weiterer Ansatz mit dem Namen Slide. Dieser liefert vor allem bei langsamen Bewegungen, wie z. B. denen eines Sprechers auf der Bühne, gute Ergebnisse. Beim Slide wird die VZ nicht einmalig pro Positionsaktualisierung angepasst. Stattdessen nähert sich die aktuelle VZ in vielen „ein-Sample-Schritten“ an die Ziel-VZ an. Dies führt ebenfalls zu seitenbandartigen Verzerrungsartefakten und zu einer Tonhöhenverschiebung, die entfernt mit einem Dopplereffekt vergleichbar ist. Der Ansatz ruft dann am wenigsten Artefakte hervor, wenn jeder Schritt mit einer kurzen Blende geglättet wird. Die Frequenzverschiebung wird dadurch jedoch nicht gemindert.

Da die Tonhöhenverschiebung von der Höhe der VZ-Änderung abhängt und bei hohen Bewegungsgeschwindigkeiten eine starke Beeinträchtigung darstellt, ist eine Kombination aus Blende und Slide sinnvoll. Diese funktioniert mit einer Obergrenze der VZ-Änderungen für das Verwenden des Slidealgorithmus. Bei Werten über dieser Grenze ist der Blendansatz empfehlenswert.

4 MATLAB Prototyp

Für das Testen und Vergleichen der Algorithmen wurde ein *MATLAB*-Prototyp der *d&b Soundscape En-Scene* Software dahingehend erweitert, dass er über die Ansätze Blende und Sildes verfügt. Mit ihm wurden Hörversuche durchgeführt. Dieses Kapitel beschreibt den Aufbau des Prototyps und seine Implementierung.

Die Erläuterung der Implementierung beschränkt sich dabei auf die Ansätze zur VZ-Änderung. Der Prototyp umfasst auch einige weitere Algorithmen, beispielsweise zur Pegelberechnung der Lautsprecherkanäle, die nicht im Rahmen dieser Arbeit entstanden, sondern von Mitarbeitern der *d&b audiotechnik GmbH* entwickelt wurden. Sie sind für die hier thematisierten Ansätze der VZ-Änderung nur indirekt von Bedeutung und wurden lediglich mitverwendet um die Algorithmen in eine realistische Umgebung einzubetten.

4.1 Benutzeroberfläche

Die Benutzeroberfläche umfasst zwei Fenster, das Main- und das Graphic-Window. Im Main-Window werden Einstellungen zur Berechnung vorgenommen und das Processing gestartet. Im Graphic-Window werden die Positionen der Primär- und Sekundärquellen in einem Koordinatensystem dargestellt.

4.1.1 Main-Window

Das Main-Window dient in erster Linie dazu, Einstellungen vorzunehmen. Es ist in vier Bereiche unterteilt:

- General Settings
- Input Device
- Output Device
- Channel Matrix

In den Bereichen *Input* und *Output Device* werden Audio-Treiber und Eingangs- bzw. Ausgangsgeräte eingestellt und festgelegt, welche der vorhandenen Kanäle benutzt werden.

General Settings umfasst allgemeine Einstellungen, wie unter anderem Anzahl an Ein- und Ausgängen und Samplerate. Außerdem gibt es einen „Play“-Button für das Starten und Einstellungen und Steuerelemente für die Regulierung der VZ-Änderung (siehe Abbildung 21).

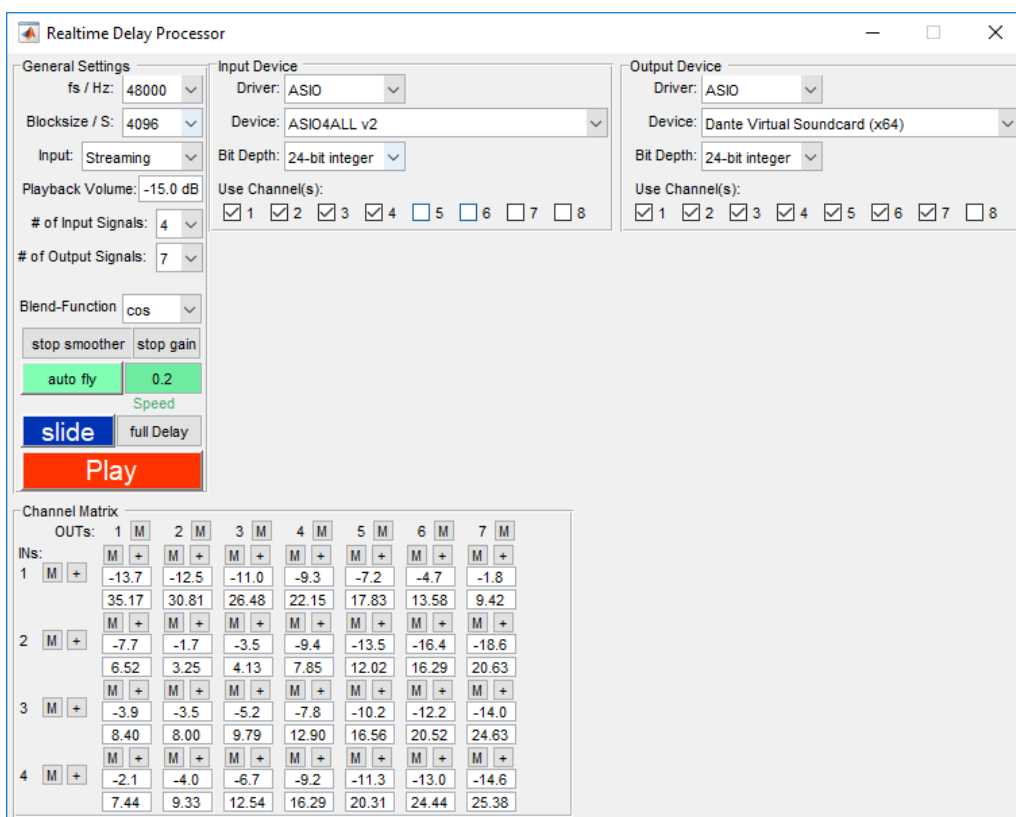


Abbildung 21: Prototyp; Main-Window

Die Einstellungen für die VZ-Änderungsalgorithmen umfassen:

- VZ-Änderungs-Algorithmus (slide, blend, auto)
 - „slide“ und „blend“ bezieht sich auf die Ansätze des Slide und der Blende aus Kapitel 3

- „auto“ bedeutet, dass das Programm sich abhängig von der Geschwindigkeit der virtuellen Quelle automatisch für einen der beiden anderen Algorithmen entscheidet.
- Delaymodus (full, tight)
 - Genau wie *En-Scene*, unterscheidet auch der Prototyp zwischen einem „full“ und einem „tight“ Delaymodus. Im *full*-Modus entspricht die VZ eines Lautsprechersignals genau der Laufzeit, die der Schall einer realen Quelle zur jeweiligen Sekundärquelle benötigen würde. Im *tight*-Modus wird diese Zeit um die kleinste VZ verkürzt. Das verhindert eine gefühlte Latenz, mit der ein Signal verzögert wiedergegeben wird, und mindert gleichzeitig die durch die VZ bedingten Artefakte. Das ist möglich, da die VZ-Differenzen zwischen den einzelnen Lautsprechern bestehen bleiben. Jedoch ist das Verwenden eines *tight*-Modus nicht sinnvoll, wenn das Signal beispielsweise das Mikrofonsignal eines Instruments oder Sängers ist, dessen Direktschall im Publikum zu hören ist. Für diesen Fall ist der *full*-Modus geeignet, da der Direktschall dann nicht später am Hörer eintrifft, als die Lautsprechersignale.
- Pegelberechnung (an, aus)
- Bewegungsmodus (mouse move, auto fly)
 - Diese Einstellung beeinflusst die Bewegungssteuerung des ersten Inputkanals. Bei „mouse move“ kann das Objekt wie alle anderen mit gehaltener Maustaste bewegt werden; bei „auto fly“ bewegt es sich selbstständig auf einer Flugbahn, die vorher im Code definiert wurde.
- Objektgeschwindigkeit
 - Die Geschwindigkeit, mit der sich das erste Objekt (Kanal 1) im „auto fly“-Modus bewegt.
- Slide-Smooth (an, aus)

- Damit kann die Artefaktoptimierung des Slides an und ausgeschaltet werden.
- Blend-Rampenfunktion (cos, log, lin, but)

Der Bereich *Channel Matrix* zeigt die Matrix, die dem Programm zu Grunde liegt. Vertikal sind die Eingänge und horizontal die Ausgänge aufgereiht. An jedem Kreuzungspunkt können Pegel- und VZ-Werte gesetzt, das Signal phaseninvertiert und stumm geschaltet werden. Bei der Berechnung von Objektbewegungen werden diese Werte automatisch angepasst und im *Matrix-Channel* Bereich angezeigt.

4.1.2 Graphic-Window

Das Graphic-Window wird mit dem Start des Audioprocessings, durch Klicken des *Play-Buttons*, geöffnet. Es zeigt ein Koordinatensystem, in dem die Pri-

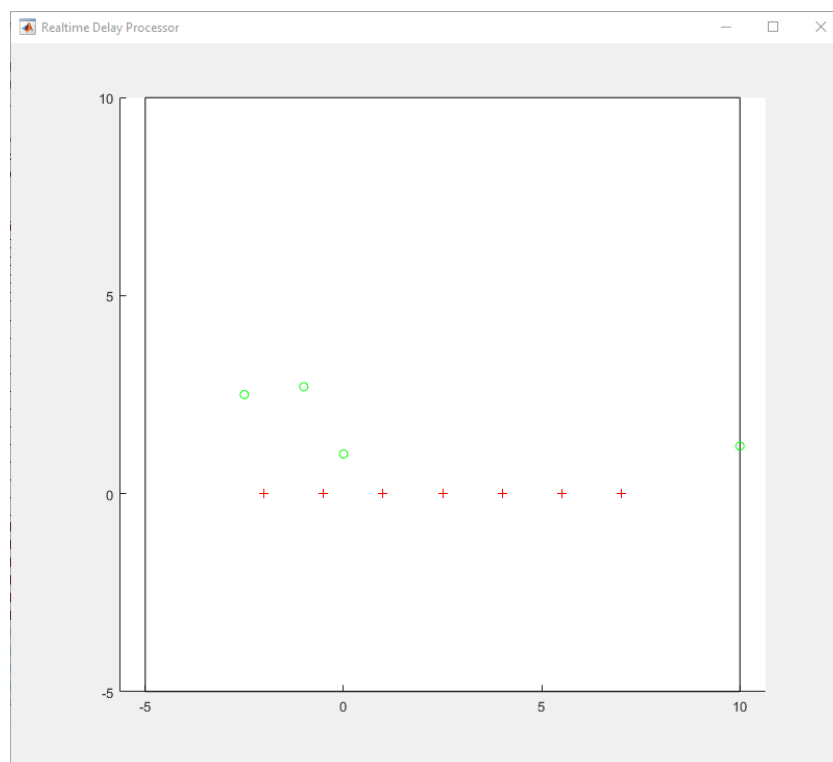


Abbildung 22: Prototyp; Graphic-Window

mär- und Sekundärquellen dargestellt sind. Die Primärquellen, also die virtuellen Schallquellen, werden als grüne Kreise, die Sekundärquellen, also die Lautsprecher, werden als rote Kreuze angezeigt. In erster Linie dient dieses Fenster als Visualisierung der Objektpositionen und Bewegungen, die auch mit der Maus manipuliert werden können.

4.2 Implementierung

Ähnlich wie *En-Scene* liegt dem Programm eine Matrix zu Grunde, in der jeder Eingangskanal jedem Ausgangskanal zugeordnet ist. An jedem Kreuzungspunkt in der Matrix kann ein individueller Pegel und ein Delaywert eingestellt werden.

Bei der Bewegung von Objekten werden die Pegel und VZ-Werte von Algorithmen berechnet. Sie können aber auch manuell geändert werden.

Mit dem Prototyp sollen Audiosignale als virtuelle Quelle platziert werden, die beispielsweise mit einem Audioplayer abgespielt werden. Eine Livesituation, also das Verwenden von Mikrofonsignalen, ist nicht vorgesehen. Ob oder in welchem Maße Latenzen entstehen, ist daher von nachrangiger Bedeutung. Wichtiger als die zeitliche Verzögerung des Systems ist die Stabilität, d.h. das Programm sollte nicht ins Stocken geraten oder gar abstürzen.

Deshalb arbeitet das Programm in großen Datenblöcken. Standardmäßig werden 4096 Samples in einem Block verarbeitet. Auch beim Hörtest wurde diese Einstellung so verwendet.

Ein Block mit 4096 Samples hat bei einer Samplerate von 48 kHz etwa eine Dauer von 85 ms. Da das mit einem „Ramp and Hold“-Verhältnis (siehe 3.1.3.1 Ramp and Hold) von 35 ms zu 50 ms kompatibel ist, wurde das Aktualisierungsintervall der Quellposition mit der Blockgröße gleichgesetzt.

4.2.1 Berechnungsschleife

Mit dem Klicken des *Play*-Buttons im Main-Window öffnet sich das Graphic-Window und es startet eine „while“-Schleife, in der die Audioverarbeitung stattfindet.

Zu Beginn dieser Schleife wird aus jedem Eingangskanal ein Block an Samples (standardmäßig 4096) eingelesen.

Dann wird ein Eingangspuffer aktualisiert, indem der Eingangsblock am Ende dieses Puffers angefügt wird. Die Länge dieser zweidimensionalen Puffermatrix für jeden Eingang ist die Länge eines Blocks plus die maximale VZ in Samples. Damit umfasst sie alle Samples, die ausgegeben werden können. Bei jedem Schleifendurchlauf rotiert sie um eine Blocklänge nach vorne, d. h. die vorderste Blocklänge an Samples wird gelöscht und die hinteren Werte rücken nach vorne nach. Anschließend wird der aktuelle Eingangsblock hinten angefügt.

Nach der Aktualisierung des Eingangspuffers wird die Position der Primärquellen aktualisiert und die entsprechenden Pegel und VZ-Werte in einer Matrix der Größe Anzahl der Eingänge mal Anzahl der Ausgänge, gespeichert.

Dann wird der aktuelle Ausgangsblock berechnet. Er enthält für jeden Ausgangskanal einen Block an Samples. Wenn sich die VZ seit dem letzten Schleifendurchlauf nicht geändert hat, wird aus dem Eingangspuffer ein Block an der Position ausgelesen, die der aktuellen VZ entspricht. Andernfalls wird der Block mit einem VZ-Änderungsalgorithmus errechnet. Wie hängt davon ab, welcher VZ-Änderungsalgorithmus gerade aktiv ist. Dieser kann entweder manuell festgelegt sein oder in Abhängigkeit von der Geschwindigkeit der VZ-Änderung vom Algorithmus bestimmt werden. Letzteres stellt die Umsetzung des Grenzwertansatzes dar.

Die Algorithmen zur VZ-Änderung berechnen einen Puffer für jeden Matrixkreuzungspunkt. Alle Puffer eines Ausgangs werden am Ende zu einem Ausgangskanal addiert. Die dabei entstehende zweidimensionale Matrix, aus Ausgangskanälen mit je einer Blocklänge an Samples, ist der Ausgangsblock.

Sobald der Ausgangsblock berechnet ist, wird er an die Audioausgänge, die im Main-Window festgelegt wurden, geschickt und zusätzlich in eine Audio-datei geschrieben. Danach beginnt die Schleife von vorne.

4.2.2 Implementierung des Blendalgorithmus

Wie in Kapitel 3.1 erläutert, gibt es zwei Puffer für jeden Matrixkreuzpunkt, einen für den aktiven und einen für den passiven Kanal. Der aktive ist dabei nur so lange wie die Rampenfunktion.

Aus dem Eingangspuffer werden die Samples in beide Puffer übertragen. Die Position des Auslesens aus dem Eingangspuffer entspricht dabei der jeweiligen VZ des Matrixkreuzpunkts. Diese ist beim aktiven die aktuelle und beim passiven die Ziel-VZ.

Im Anschluss werden die Rampenfunktionen angepasst. Für jeden Blendvorgang gibt es eine Funktion zum Ein- und eine zum Ausblenden. Die Einblendfunktion steigt standardmäßig von 0 bis 1 an und bleibt bis zum Ende der Blocklänge bei 1. Sie wird mit dem neuen Zielpegel, der durch den Pegelalgorithmus errechnet wurde, multipliziert. Die Ausblendfunktion, die standardmäßig von 1 auf 0 absinkt, wird mit dem aktuellen Pegel multipliziert.

Nun können die beiden Puffer mit den Rampenfunktionen multipliziert und anschließend addiert werden. Der passive wird mit der Funktion zum Einblenden und der aktive mit der zum Ausblenden verrechnet. Wegen der unterschiedlichen Länge der beiden Puffer wird der aktive auf den Anfangsteil des passiven addiert, der Rest des passiven bleibt, von der Multiplikation mit dem neuen Pegelwert abgesehen unverändert (siehe Abbildung 23).

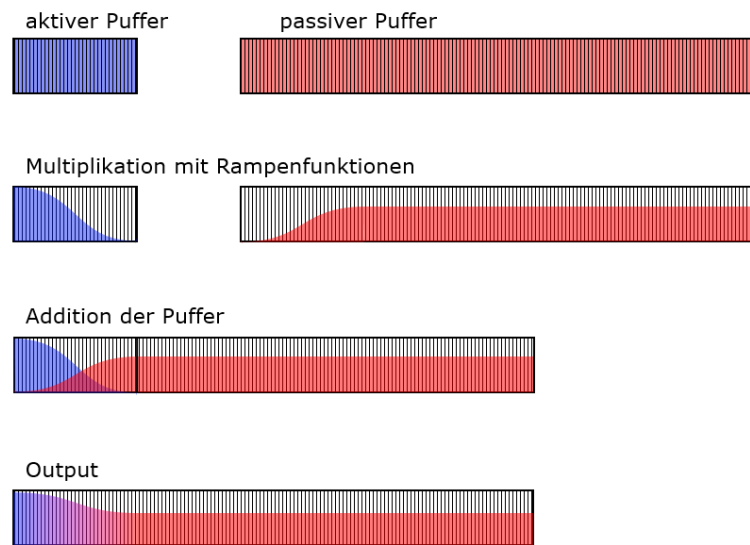


Abbildung 23: Berechnung Blendpuffer

4.2.3 Implementierung des Slidealgorithmus

Der Slidealgorithmus wurde auf zwei verschiedene Weisen umgesetzt. Einmal ohne Optimierung, also indem Samples wiederholt oder übersprungen werden und einmal in der Optimierungsversion, bei der jeder Slideschritt mit einer kurzen Blende vollzogen wird (siehe Kapitel 3.3.3). Letzere Version führt zu einem höheren Rechenaufwand, ruft aber weniger Artefakte hervor.

4.2.3.1 Slide ohne Artefaktoptimierung

Der Slidealgorithmus ermittelt zunächst, wie groß die Differenz zwischen der aktuellen VZ und der Ziel-VZ ist. Anhand dieser Differenz wird der Intervallabstand j der Slideschritte ermittelt.

Die weitere Berechnung wird in einer Schleife realisiert, in der ein Index bis zur VZ-Differenz ansteigt. In jedem Schleifendurchlauf wird eine Liste der

Länge j mit Pufferindizes befüllt. Dazu bekommt sie eine Folge an natürlichen Zahlen.

Mithilfe dieser Liste können alle Samples zwischen zwei Slideschritten auf einmal übertragen werden. Zunächst wird dazu ihre Anfangsposition im Eingangspuffer berechnet. Dazu wird vom hintersten Index des Eingangspuffers die Summe aus der aktuellen VZ und der Blockgröße subtrahiert. Jeder Index aus der Liste wird dann mit dieser Anfangsposition addiert.

Schließlich können die Werte, die an den eben berechneten Positionen im Eingangspuffer stehen, in den Matrixkreuzungspuffer übertragen werden. Die Positionsindizes in diesem Puffer sind die Indizes aus der Liste (also ohne die Addition mit der Anfangsposition). Die Übertragung der Werte mit der Liste an Indizes ist in Abbildung 24 dargestellt.

L = Liste mit Natürlichen Zahlen als Indizes

1, 2, ... j

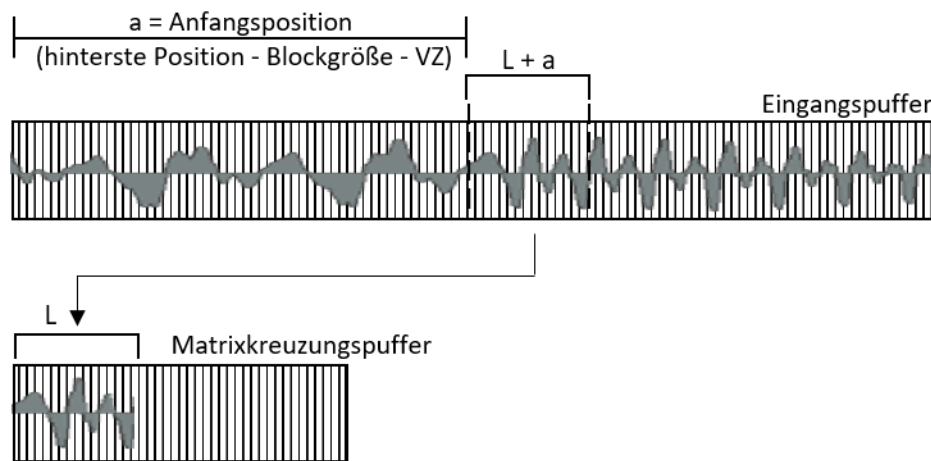


Abbildung 24: Übertragungsprinzip beim Slide

Nach einer solchen Übertragung von Werten in den Matrixkreuzungspuffer, wird die Liste der Indizes angepasst, indem jede der natürlichen Zahlen mit j addiert wird. Würde die Übertragung nun erneut stattfinden, würde das Sig-

nal exakt so im Matrixkreuzungspuffer gespeichert werden, wie es im Eingangskanal ankommt. Um dies zu verhindern, wird zusätzlich die aktuelle VZ angepasst. Ist sie größer als die Ziel-VZ, wird ihr Betrag um eins verringert; wenn sie kleiner ist, dann steigt ihr Wert um eins.

Nun kann die nächste Übertragung stattfinden. Wurde die aktuelle VZ vergrößert, ist die Anfangsposition für das Auslesen aus dem Eingangspuffer kleiner, als beim Durchlauf davor und alle Werte werden um eins nach vorne versetzt ausgelesen.

Das führt dazu, dass der erste Index aus der Liste denselben Wert ausliest, den bei der Übertragung zuvor der letzte Index übergeben hatte. Es entsteht die Wiederholung eines Samples, durch die die VZ um einen Schritt vergrößert wurde.

Umgekehrt wird ein Sample nicht ausgelesen, wenn die VZ verkürzt wird, was zu einem Überspringen führt.

Dieser Ablauf aus Übertragen und Anpassung von Indexliste und VZ, wiederholt sich solange, bis der Matrixkreuzungspuffer vollständig gefüllt ist.

Ist die Blocklänge, die auch die Länge des Matrixkreuzungspuffers ist, kein ganzzahliges Vielfaches von j , führt das dazu, dass die Indizes immer entweder über das Ende des Puffers hinausreichen oder nicht bis zum Ende kommen. Um das zu verhindern wird die Anzahl der Listen vor dem Prozess aufgerundet und die letzte Liste verkürzt, sodass ihr höchster Index genau auf das Ende des Puffers zeigt.

Sobald die Puffer aller Matrixkreuzungen befüllt wurden, wird für jeden eine Pegelkurve berechnet. Sie startet beim bisherigen Pegel und steigt bzw. sinkt mit einer Cosinusrampe zum neuen Ziel-Pegel. Nachdem diese Pegelanpassung auf jeden Puffer übertragen wurde, können die Matrixkreuzungspuffer

zum Ausgangsblock addiert und dieser zur Wiedergabe an die Audioschnittstelle gesendet werden.

4.2.3.2 Slide mit Blendglättung

Der Slide mit Blendglättung funktioniert sehr ähnlich zu dem Slide ohne Glättung. Der einzige Unterschied besteht im Übertragungsprinzip. Die Übertragung wird in zwei Schritte aufgeteilt: In die Übertragung des Blendteils und die Übertragung des Restteils. Der Blendteil umfasst alle Samples, die für die Blende verwendet werden. Der Restteil beinhaltet die übrigen, die unverändert übertragen werden.

Für die Unterteilung werden zwei Indexlisten erzeugt. Eine reicht von der Anfangsposition bis zum Ende des Blendvorgangs, die andere vom ersten Sample nach der Blende bis j . Bei einer Blendendauer 24 Samples reicht der Blendteil von 1 bis 24 und der Restteil von 25 bis j .

Das Auslesen der Werte durch die Indizes des Restteils erfolgt auf die gleiche Weise, wie in Abschnitt 4.2.3.1 beschrieben. Beim Befehl des Auslesens des Rampenteils wird zeitgleich von der Position der vorherigen und der aktuellen VZ ausgelesen. Die Samples der vorherigen VZ werden mit einer Rampe zum Ausblenden, die der aktuellen mit einer zum Einblenden multipliziert.

In MATLAB ist es möglich, mehrere Werte eines Vektors oder einer Matrix zeitgleich zu übertragen oder zu verechnen. Deshalb werden die Vektoren mit Samples, die auf einmal ausgelesen werden, direkt mit je einem Vektor multipliziert, der die Rampenfunktion beinhaltet. Damit wird in einem Schritt die Blendglättung vollzogen, während die Werte ausgelesen werden ohne, dass sie zusätzlich zwischengespeichert werden müssen.

5 Hörversuche

Am 23. und 24.08.2018 wurden zur Beurteilung der beiden im Prototyp umgesetzten Algorithmen Hörversuche mit 17 Probanden durchgeführt. Die Tests sollten zeigen, inwiefern die durch die Anpassung der Verzögerungszeiten hervorgerufenen Artefakte zum störenden Effekt werden und welcher Algorithmus sich demnach für den Einsatz eines Systems wie *d&b soundscape En-Scene* eignet.

Bei der Auswahl der Probanden wurde darauf geachtet, dass es sich um Personen mit Hörerfahrung handelt, bei denen davon ausgegangen werden kann, dass zumindest ein Großteil in der Lage ist einen Kammfilter o.ä. zu erkennen. Die Mehrheit der Testpersonen sind Mitarbeiter der d&b audiotechnik GmbH aus den Abteilungen Forschung und Entwicklung, Education/ Application Support und Produktmanagement. Daneben beteiligten sich auch Angehörige der Hochschule der Medien Stuttgart.

Das folgende Kapitel behandelt die Vorbereitung, Durchführung und Auswertung der Tests.

5.1 Aufbau und Vorbereitung der Tests

Bei dem Test befindet sich je ein Hörproband vor einer horizontalen Anordnung an Lautsprechern, die der realen Frontbeschallung mit einem *d&b soundscape*-System entspricht. In einem Blindversuch werden virtuelle Schallereignisse durch das Verwenden unterschiedlicher Algorithmen zur Anpassung der VZ hinter den Lautsprechern bewegt. Die Probanden sollen bewerten, wie sehr das Klangerlebnis durch störende Artefakte beeinträchtigt wird.

5.1.1 Versuchsanordnung

Das horizontale Lautsprecherarray besteht aus sieben *T10* Lautsprechern, die im Abstand von 1,5 Metern angeordnet sind. Ein Tisch mit einem Laptop befindet sich 3,5 Meter vor der Lautsprecheranordnung. Da das System auch außerhalb eines Sweet Spots funktionieren soll, hat die Hörerposition einen Meter Abstand zur Mitte der Anordnung (siehe Abbildung 25).

Beim Test sitzen die Probanden vor dem Laptop, mit dem sie Klangbeispiele starten und Beurteilungen abgeben. Dabei sind die Klangbeispiele in Fallbeispielen zusammengefasst. Pro Beispiel bewegen sich virtuelle Schallereignisse mit dem selben Eingangssignal und derselben Bewegungsgeschwindigkeit auf der selben Flugbahn. Sie unterscheiden sich einzig durch den verwendeten Algorithmus zur Anpassung der VZ und somit in den klanglichen Artefakten.

Damit sichergestellt ist, dass alle Probanden identische Beispiele hören, werden die Bewegungen nicht in Echtzeit berechnet. Stattdessen liegen die

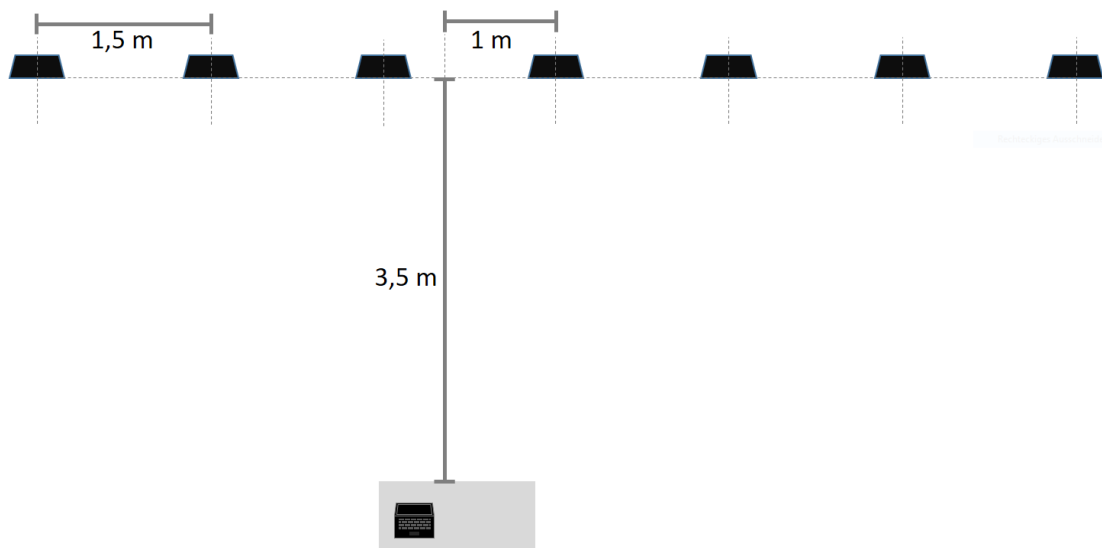


Abbildung 25: Anordnung Hörversuch

Klangbeispiele als Audiodateien (.wav) vor. Im Vorfeld wurden mit dem MATLAB-Prototyp aus Kapitel 4 Bewegungen berechnet und die Lautsprecher-sig-nale aufgezeichnet. Diese Aufzeichnungen werden mit einem Max-Patch wie-dergegeben, der von der Testperson bedient wird.

5.1.2 Max-Patch des Tests

Der Aufbau des Testpatches ist an den MUSHRA-Test¹⁴ angelehnt. Pro Fall-beispiel liegen dem Proband vier Klangbeispiele stellvertretend für vier Ver-fahrensweisen vor, zu denen beim Hören Beurteilungen abgegeben werden sol-len. Dabei kann die Testperson für einen direkten Vergleich eigenständig zwi-schen den Beispielen hin und herschalten und jedes beliebig oft abspielen.

Hierfür ist der Patch in Form einer Tabelle aufgebaut. Links in der ersten Spalte sind untereinander vier nummerierte Knöpfe abgebildet, mit denen die Wiedergabe der Klangbeispiele gestartet wird. Rechts neben diesen Knöpfen soll mit *slider*-Objekten die Beurteilung der jeweiligen Beispiele stattfinden.

Um die Bewertung möglichst intuitiv zu gestalten, sollen die Probanden Schul-noten vergeben, d. h. in einer Skala von 1 bis 6 bewertern, wobei 1 die beste Bewertung ist. Dabei können nur Bewertungen in ganzen Zahlen abgegeben werden. Das verhindert eine neutrale Beurteilung zwischen bester und schlechtester Bewertung, was dazu führt, dass jede mittlere Bewertung eine Tendenz, entweder zum Guten oder zum Schlechten hat. Zur Veranschauli-chung für die Probanden sind die *slider*-Objekte in einen grünen und einen roten Bereich unterteilt. Je schlechter eine Bewertung ausfällt, desto größer

¹⁴ (International Telecommunication Union (ITU), 2014)

ist der rote Anteil. Je besser die Beurteilung ist, desto breiter wird der grüne Bereich dargestellt. (siehe Abbildung 26).

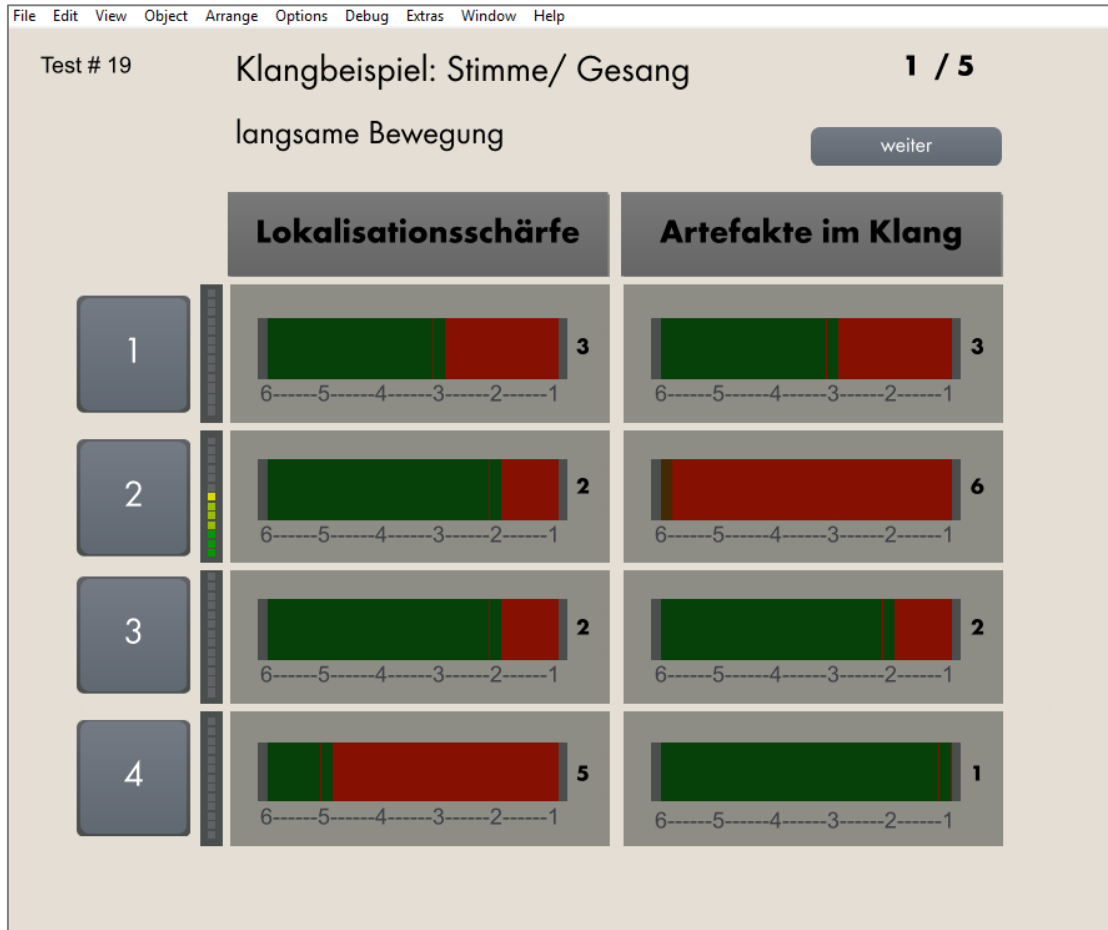


Abbildung 26: Hörversuch; Max-Patch

Sobald eine Testperson die Bewertung aller vier Beispiele abgeschlossen hat, gelangt sie über einen „weiter“-Knopf am oberen rechten Rand zum nächsten Fallbeispiel. Sobald dieser Knopf gedrückt wird, werden die Bewertungen gespeichert und können ab dann nicht mehr korrigiert werden. Wenn die Daten aller Fallbeispiele gespeichert wurden, öffnet sich ein Pop-up-Fenster, das den Probanden darauf hinweist, dass der Test beendet ist. Gleichzeitig steigt ein Zähler um eins an, unter dem die Daten des nächsten Tests gespeichert werden können.

Das Programm speichert alle eingegebenen Daten als Tabelle in einem Speicherobjekt namens „col/“. Diese kann später als Textdatei exportiert und schließlich zur Auswertung mit Excel, oder einer vergleichbaren Software, importiert werden.

5.1.3 Bewertungskriterien

Die Bewertungen, die von den Probanden abgegeben werden, sind in zwei Kategorien eingeteilt, die „Lokalisationschärfe“ und die „Artefakte im Klang“.

Zunächst soll eingeschätzt werden, wie scharf die Quelle lokalisierbar ist, d. h. wie sehr sich eine gleichmäßige Flugbahn abzeichnet und wie präzise sich eine Position bestimmen lässt. Eine schlechte Schärfe würde in dem Fall bedeuten, dass die Bahn nicht stetig verläuft, dass das Objekt von einer zur nächsten Sekundärquelle springt oder nur eine grobe Richtung ausgemacht werden kann, aus der der Schall eintrifft.

Dies soll in erster Linie den Fokus der Hörer weg vom Lautsprecher hin zur platzierten virtuellen Primärquelle lenken, d.h. die virtuelle Schallquelle soll als Quelle betrachtet und auf ihre Natürlichkeit hin bewertet werden. Denn ein ideales Verfahren würde keine Unterscheidbarkeit zwischen virtuellem und realem Schallereignis zulassen.

Der eigentliche Kern des Tests ist jedoch die Bewertung der klanglichen Artefakte. Dabei würde die Beurteilung „1“ bedeuten, dass die Testperson findet, dass der Klang nicht durch das Auftreten von Artefakten beeinträchtigt wird. Eine „6“ hingegen entspricht einer Beeinträchtigung durch Artefakte, die so stark ausfällt, dass das Verfahren nach Ansicht des Probanden nicht für eine Anwendung in der Praxis eingesetzt werden kann.

5.1.4 Klangbeispiele

Die abgespielten Klangbeispiele sind in fünf Fallbeispielen sortiert. Pro Fallbeispiel gibt es vier Klangbeispiele, deren Wiedergabe von der Testperson beliebig oft gestartet werden kann.

Hinter jedem der vier Beispiele versteckt sich ein anderer Ansatz. Das sind neben der Blende (35 ms Rampendauer und 50 ms „Hold“) und dem Slide (mit Blendglättung mit 24 Samples Rampendauer) zwei weitere Beispiele, die der Skalierung der Ergebnisse dienen. Diese sind ein manipulierter Slide, der mit zusätzlichen künstlichen Kammfiltern und verzerrungsartigen Artefakten versetzt wurde und ein rein pegelbasiertes Verfahren, bei dem die Artefakte durch die VZ-Änderung nicht auftreten. Der Test soll zeigen wie Slide und Blende bei einem Blindversuch in Relation zu diesen eingeordnet werden.

Die fünf Fallbeispiele sollen zeigen in welchen Fällen die Algorithmen gut oder weniger gut geeignet sind. Dazu werden drei verschiedene Bewegungsgeschwindigkeiten mit dem selben Eingangssignal und drei Eingangssignale bei der selben Bewegungsgeschwindigkeit untersucht. Die Flugbahn ist bei allen Fallbeispielen identisch. Es handelt sich um eine Ellipse, auf der die Quelle im Uhrzeigersinn bewegt wird (siehe Abbildung 27).

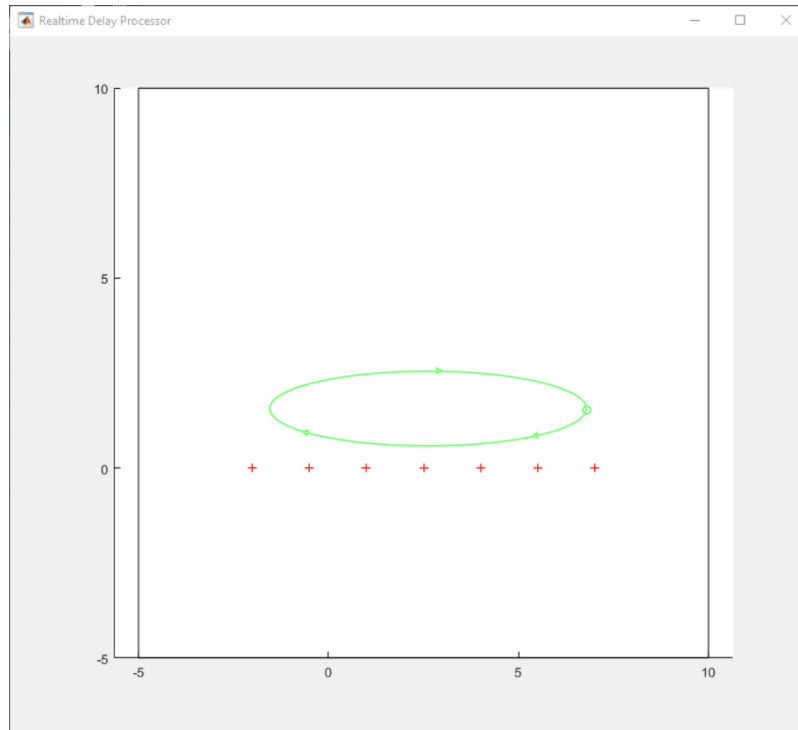


Abbildung 27: Hörversuch; Flugkurve

Unser Gehör ist auf die menschliche Stimme ausgelegt und es reagiert deshalb am empfindlichsten auf Signale, die im Frequenzbereich der Sprache liegen. Deshalb ist eines der Fallbeispiele ein männlicher Sprecher. Seine Bewegungsgeschwindigkeit ist „langsam“, d. h. sie entspricht dem Tempo, mit dem ein Moderator sich auf der Bühne bewegt (0.1 m/s bis 0.5 m/s).

Das ist eine der Hauptanwendungen für Immersive Audio Systeme im Beschallungsbereich. Mit sog. „Trackingsystemen“ wird die Position von Musical-Darstellern, Bandmitgliedern, Moderatoren usw. ermittelt und an den Renderer übergeben.

Artefakte der VZ-Änderung wie spektrale Seitenbänder, Kammfilter und Frequenzverschiebung werden bei regelmäßigen tonalen Signalen besonders

deutlich wahrgenommen. Deshalb sind die anderen beiden Fallbeispiele eine akustische Gitarre und eine weibliche Sängerin.

Um auch hier die Bühnensituation zu rekonstruieren in der eine Person getrackt wird, werden beide Beispiele mit der selben Geschwindigkeit bewegt wie der Sprecher.

Ein weiterer realistischer Anwendungsfall, neben dem Tracking von Menschen und Objekten, ist das Platzieren und Bewegen von virtuellen Klangquellen aus künstlerischen und ästhetischen Gründen. Wenn ein Künstler beispielsweise Teile seiner Musik wild durch den Raum bewegen möchte, spielt der Einfluss der Geschwindigkeit auf die Artefakte eine große Rolle. Um diesen Einfluss zu ermitteln, gibt es drei Fallbeispiele, bei denen dasselbe Eingangssignal mit unterschiedlichen Geschwindigkeiten bewegt wird. Das verwendete Signal ist das der akustischen Gitarre. Zusätzlich zu dem „langsamen“ Beispiel wird dieses auch einmal „mittelschnell“ (0,5 m/s bis 1 m/s) und einmal „äußerst schnell“ (1.0 m/s bis 8.5 m/s) bewegt. Dabei wird jedes Mal die selbe Flugkurve (siehe Abbildung 27) verwendet.

5.2 Auswertung

Der Test sollte zeigen, ob und wie sich die Ansätze Slide und Blende für eine praktische Anwendung eignen. Das Ergebnis deckt sich mit dem Fazit aus der Ansatzoptimierung in Kapitel 3.5. Das Diagramm in Abbildung 28 zeigt die Beurteilung des Klangs der vier Ansatzbeispiele.

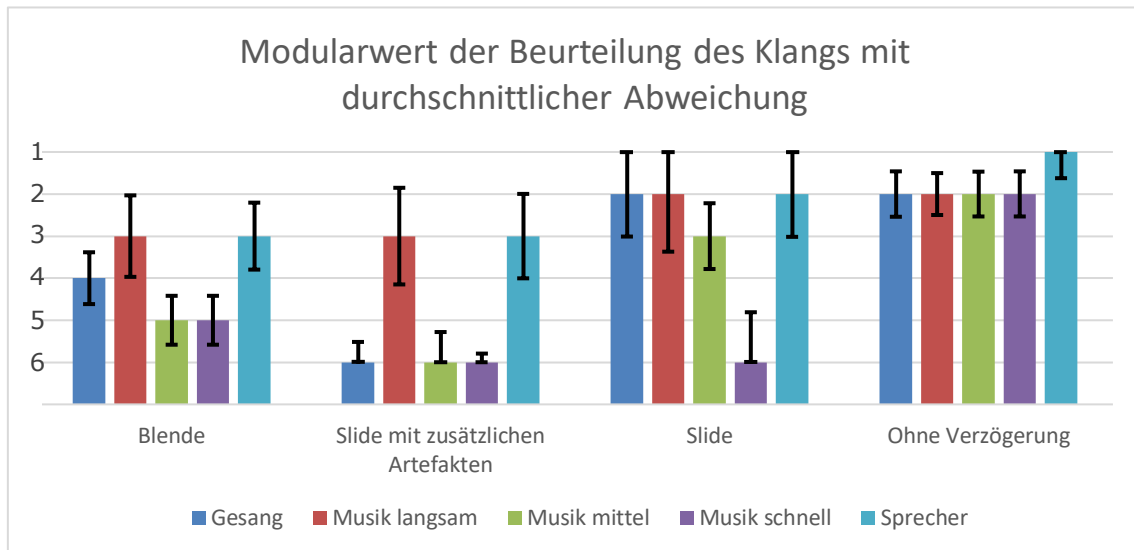


Abbildung 28: Auswertung Hörversuch; Beurteilung des Klangs

Der Ansatz des Slides liegt bei langsamen Bewegungen auf einem vergleichbaren Niveau wie reines *Amplitude Panning*, bei dem keine Artefakte der VZ-Änderung auftreten. Vor allem bei der Sängerin wird dieser Algorithmus deutlich besser bewertet, als die Blende. Bei schneller Bewegung schneidet dieses Verfahren jedoch äußerst schlecht ab.

Die Unterschiede zwischen schneller und langsamer Bewegung sind bei der Blende geringer. Sie wird insgesamt zwar schlechter bewertet, bei hoher Bewegungsgeschwindigkeit erzielt sie jedoch eine bessere Beurteilung als der Slideansatz.

Generell wird die schnelle Bewegung, außer beim rein pegelbasierten Verfahren, als sehr artefaktbehaftet und damit als schlecht beurteilt. Auffällig ist,

dass die gesprochene Sprache überall vergleichsweise gut abschneidet und folglich nicht so anfällig für diese Form der Artefakte ist.

Das Ergebnis des Hörversuchs zeigt, dass das Verwenden des Slidealgorithmus mit einer Obergrenze der Bewegungsgeschwindigkeit sinnvoll ist, wie es in Kapitel 3.4.3 erläutert wird.

Auch wenn ein möglicher Rückschluss sein könnte, dass oberhalb dieser Grenze keine Signalverzögerung stattfinden sollte, ist dies dennoch nicht zu empfehlen. Die Blende schneidet zwar bei der sehr schnellen Bewegung nicht gut ab, obwohl die Kammfilterartefakte dann, wie in Abschnitt 3.4.3 thematisiert, nicht so deutlich hörbar sind. Das liegt jedoch daran, dass sich die Primärquelle beim Hörversuch auf einer sehr kleinen Ellipse bewegt. Dieser Fall ist deswegen wenig aussagekräftig, weil in der Praxis eine hohe Geschwindigkeit fast immer mit großen Distanzen einhergeht. Eine große Pegeländerung, die die Kammfilter reduziert, bleibt also beim Klangbeispiel des Hörversuchs aus, in den meisten realen Anwendungen jedoch nicht.

Zudem bliebe beim Umschalten auf ein rein pegelbasiertes Verfahren die Frage offen, mit welcher VZ die Signale dann wiedergegeben werden. Schließlich ist jedes Lautsprechersignal entsprechend der bisherigen Position der Primärquelle individuell verzögert. Da sich die wahrgenommene Schallereignisrichtung jedoch ändern muss, können die VZ nicht bei den Werten bleiben, die sie beim Erreichen des Grenzwertes haben. Den Slide zu verwenden um die VZ aller Kanäle gleichzusetzen ist deshalb nicht ratsam, weil es dann passieren kann, dass der Grenzwert dabei überschritten wird. Außerdem würde eine Bewegung, die häufig für kurzen Zeitraum den Grenzwert überschreitet, andauernd eine Frequenzverschiebung hervorrufen, die nicht zur Bewegung passt. Das „Ausschalten“ der Verzögerung sollte daher mit einer Blende vollzogen werden.

Da der Einsatz der Blende beim Erreichen des Grenzwertes ohnehin notwendig ist, ist es auch ratsam die VZ oberhalb des Grenzwertes anzupassen und dafür den Ansatz der Blende zu verwenden. Denn dann bleibt die klare Lokalisation bestehen. Die Möglichkeit, bei bestimmten Objekten, bei denen im Voraus bekannt ist, dass sie mit sehr hohen Geschwindigkeiten bewegt werden, die Signalverzögerung von vornherein auszuschalten, wäre dazu eine gute Ergänzung.

6 Fazit und Ausblick

In dieser Arbeit wurden Ansätze zur Änderung der VZ von Audiosignalen untersucht. Ziel war es einen Algorithmus zu finden oder zu entwickeln, durch den das Bewegen von virtuellen Schallquellen mit einem Mehrkanalsystem zur Schallfeldreproduktion nach Möglichkeit wenige bis gar keine hörbaren Artefakte hervorruft.

Dabei wurde ein neuer Algorithmus namens „Slide“ entwickelt, der bei Bewegungen mit geringer bis mittlerer Bewegungsgeschwindigkeit eine Verbesserung zu bestehenden Systemen, wie dem sog. „Blendansatz“ darstellt. Die Funktionsweise des Slides entspricht der einer Bandmaschine mit beweglichem Lesekopf. Der Abstand zwischen Lese- und Schreibkopf entspricht der VZ. Wenn nun die VZ geändert werden soll, gleitet der Lesekopf zu seiner neuen Position. Das ruft Frequenzverschiebungen hervor, die dem akustischen Dopplereffekt entfernt ähneln. Diese Frequenzverschiebung ist bei Bewegungen mit langsamer bis mittlerer Geschwindigkeit kaum wahrnehmbar. Bei hohen Geschwindigkeiten tritt dieses Artefakt jedoch äußerst stark in Erscheinung.

Deshalb ist das Fazit dieser Arbeit die Empfehlung, den Slidealgorithmus bei einer Aktualisierungsrate der Objektposition von 85 ms mit einer Blendglättung von 24 Samples Rampendauer und einer Obergrenze zu verwenden. Diese Obergrenze sollte bei einer VZ-Änderung von 40 Samples pro Aktualisierungsschritt liegen. Bei VZ-Änderungen über diesem Wert ist das Verwenden des Blendansatzes empfehlenswert. Dabei sollten Blenden mit einer Rampendauer von 35 ms und einer Cosinusfunktion als Rampenfunktion oder einer Butterworth-Approximation sechsten Grades durchgeführt werden. Um das Klangbild zu stabilisieren, ist nach jedem Blendvorgang eine „Hold“-Zeit von 50 ms sinnvoll, in der keine Anpassung der VZ passiert.

Bei der Weiterentwicklung der Wiedergabesysteme für immersive Audio ist es sinnvoll, auch das hier beschriebene Verfahren zur Änderung der Verzögerungszeit weiter zu verbessern oder gänzlich neue Ansätze in Betracht zu ziehen. So wäre es beispielsweise denkbar, dass die Kammfilter des Blendansatzes nicht so stark wahrgenommen werden, wenn die Blenden nicht in jedem Lautsprechersignal zum selben Zeitpunkt passieren. Zudem könnte das Arbeiten mit einer höheren Samplerate möglicherweise die aufwendige Optimierung des Slidealgorithmus obsolet machen.

Literaturverzeichnis

About Max [Online] / Verf. Cycling '74 // CYCLING '74: Tools for sound, graphics and interactivity. - Cycling '74, 2018. - 06.. 09. 2018. - <https://cycling74.com/products/max>.

Audiocodierung [Buchabschnitt] / Verf. Bernhard Grill Wolfgang Hoeg // Handbuch der Tonstudioteknik, 8. Auflage; Band 2 / Buchverf. Wöhr Martin (Hrsg.) Dickreiter Michael, Dittel Volker, Hoeg Wolfgang,. - Berlin : De Gruyter Saur, 2014.

Beschallung [Buchabschnitt] / Verf. Bernhard Schullan Ralf Zuleeg, Wolfgang Hoeg // Handbuch der Tonstudioteknik / Buchverf. Matrin Wöhr (Hrsg.) Michael Dickreiter Volker Dittel, Wolfgang, Hoeg,. - Berlin : De Gruyter Saur, 2014.

BS.1534 : Method for the subjective assessment of intermediate quality levels of coding systems / Verf. International Telecommunication Union (ITU). - Genf : ITU-R; Radiocommunication Sector of ITU, 18. 08 2014.

Christian Doppler [Online] / Verf. Christian Doppler Fonds. - Universität Salzburg, Naturwissenschaftliche Fakultät, 2018. - 28. 08 2018. - christian-doppler.net/dopplereffekt.

Entdecken Sie Matlab [Online] / Verf. MathWorks // Homepage of MathWorks. - MathWorks, 2018. - 06.. 09. 2018. - <https://de.mathworks.com/products/matlab.html>.

Gersthen Physik, 22. Auflage [Buch] / Verf. Meschede Dieter. - Berlin : Springer-Verlag, 2004.

Räumliches Hören [Buchabschnitt] / Verf. Blauert Jens Braasch Jonas // Handbuch der Audiotechnik / Buchverf. Weinzierl Stefan. - Berlin Heidelberg : Springer-Verlag, 2008.

Schallwahrnehmung [Buchabschnitt] / Verf. Dickreiter Michael Goeres-Petri Jürgen // Handbuch der Tonstudioteknik, 8. Auflage; Band 1 / Buchverf. Wöhr Martin (Hrsg.) Dickreiter Michael, Dittel Volker, Hoeg Wolfgang,. - Berlin : De Gruyter Saur, 2014.

Spatial Sound Rendering in MAX/MSP with ViMiC [Konferenz] / Verf. Peters Nils Matthews Tristan, Braasch Jonas, McAdams Stephen. - McGill University, Montréal, CA & School of Architecture, Troy, US : CIRMMT - Centre for Interdisciplinary Research in Music Media and Technology, 2014.

TI 501 d&b Soundscape - System design and operation [Bericht] / Verf. d&b audiotechnik GmbH. - Backnang : d&b audiotechnik GmbH, 2018.

Tonaufnahme und Tonwiedergabe [Buchabschnitt] / Verf. Günther Theile Michael Dickreiter, Wolfram Graul, Florian Camerer, Gerhard Spikofski // Handbuch der Tonstudioteknik / Buchverf. Martin Wöhr (Hrsg.) Michael Dickreiter, Volker Dittel, Wolfgang Hoeg,. - Berlin : De Gruyter Saur, 2014.

Wellentheoretische Raumakustik [Buchabschnitt] / Verf. Maier P. // Handbuch der Audiotechnik / Buchverf. Weinzierl Stefan. - Berlin : Springer-Verlag, 2008.

Anhang

Übersicht

1 Max7 Patches.....90

Blende

Sprung-Interpolation

Slidealgorithmus

2 MATLAB Code.....98

Slide und Blende

Slide mit Blendglättung

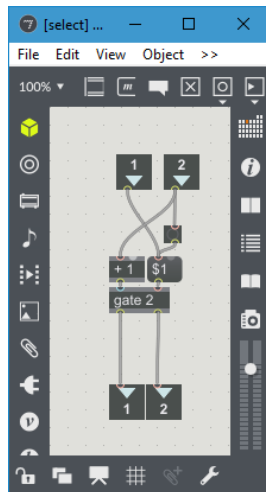
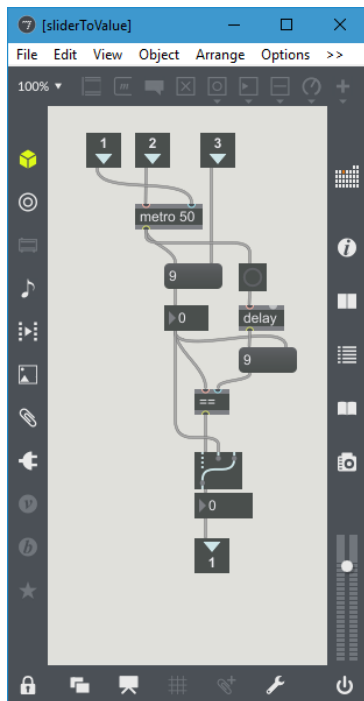
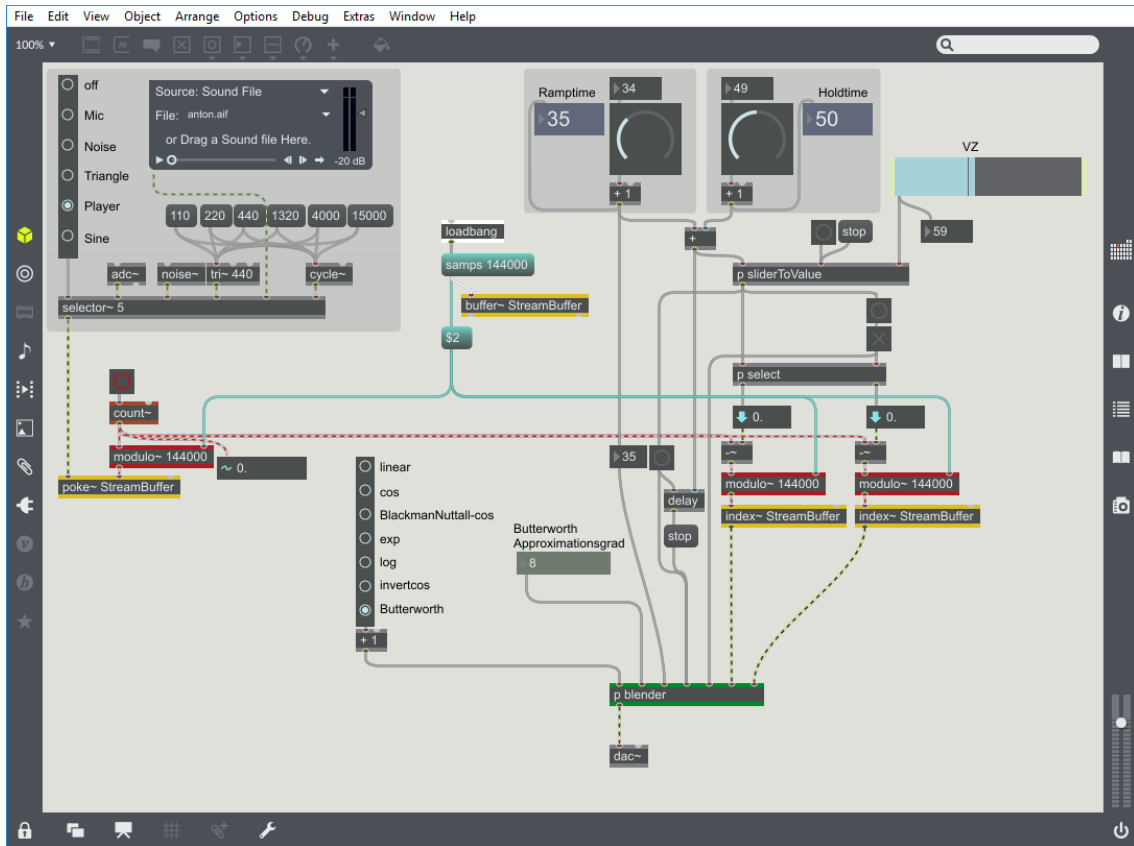
3 Hörversuche102

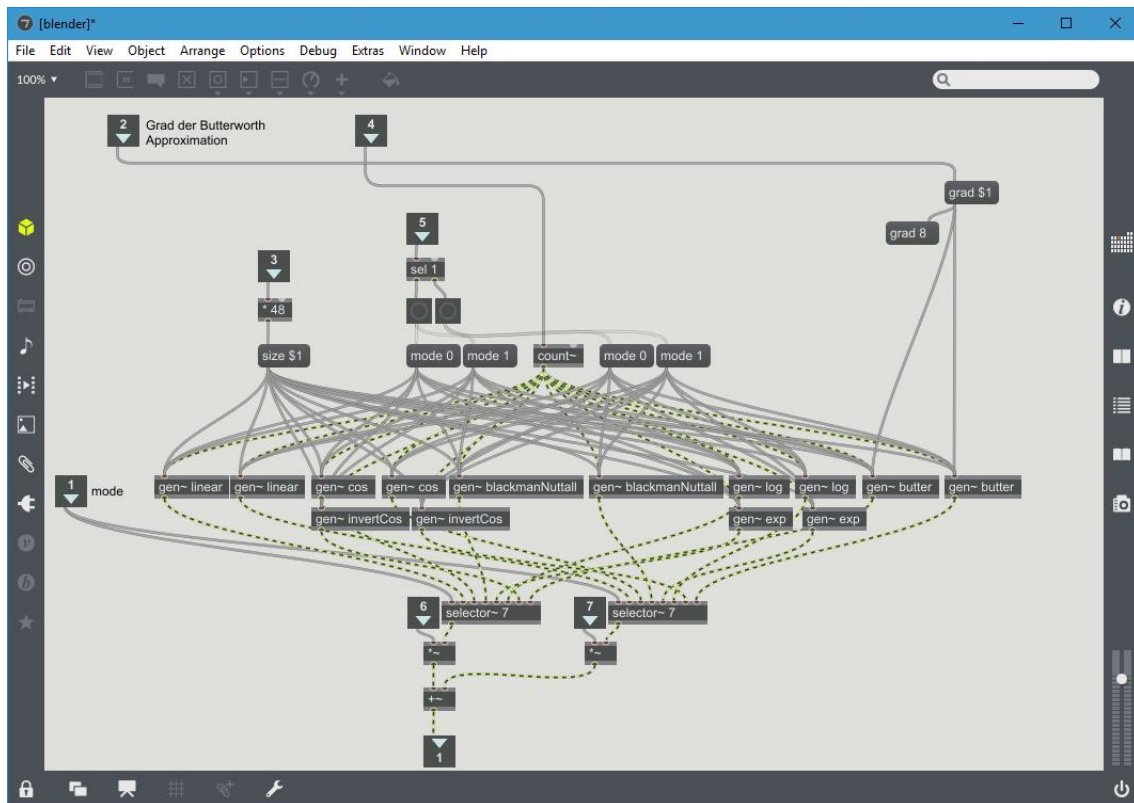
Foto der Versuchsanordnung

Auswertung

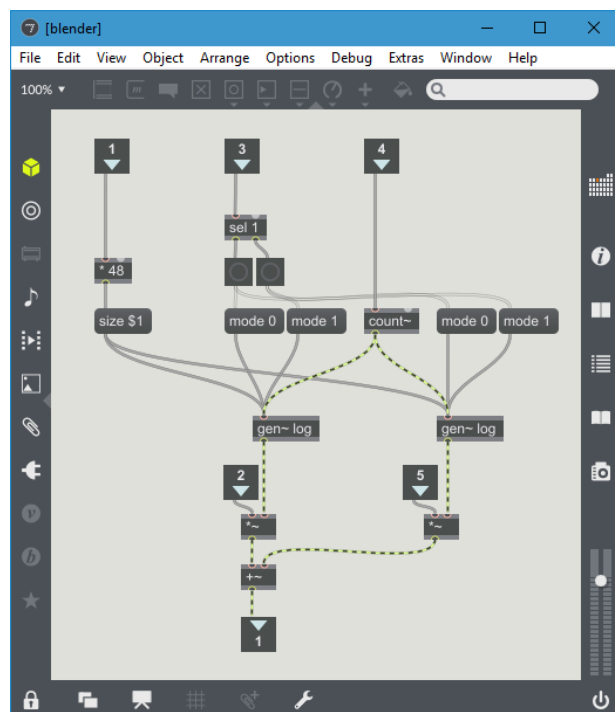
1. Max 7 Patches

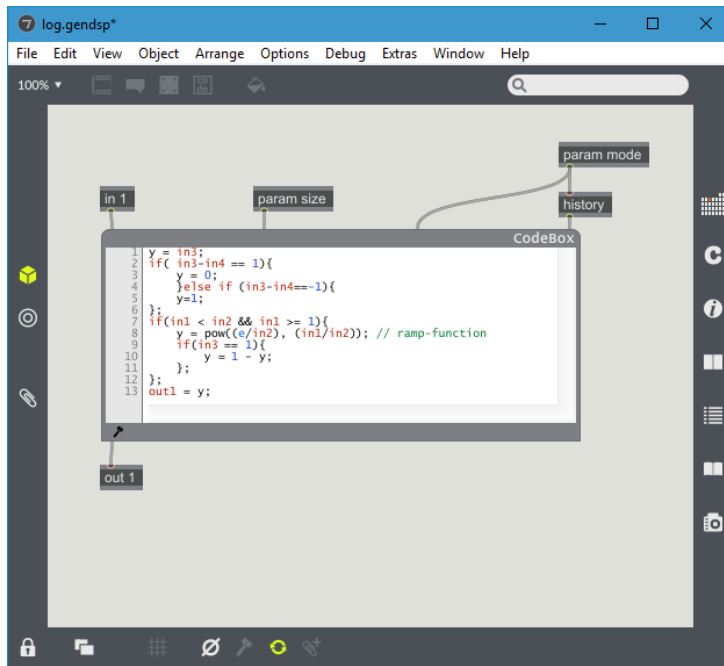
Blendalgorithmus



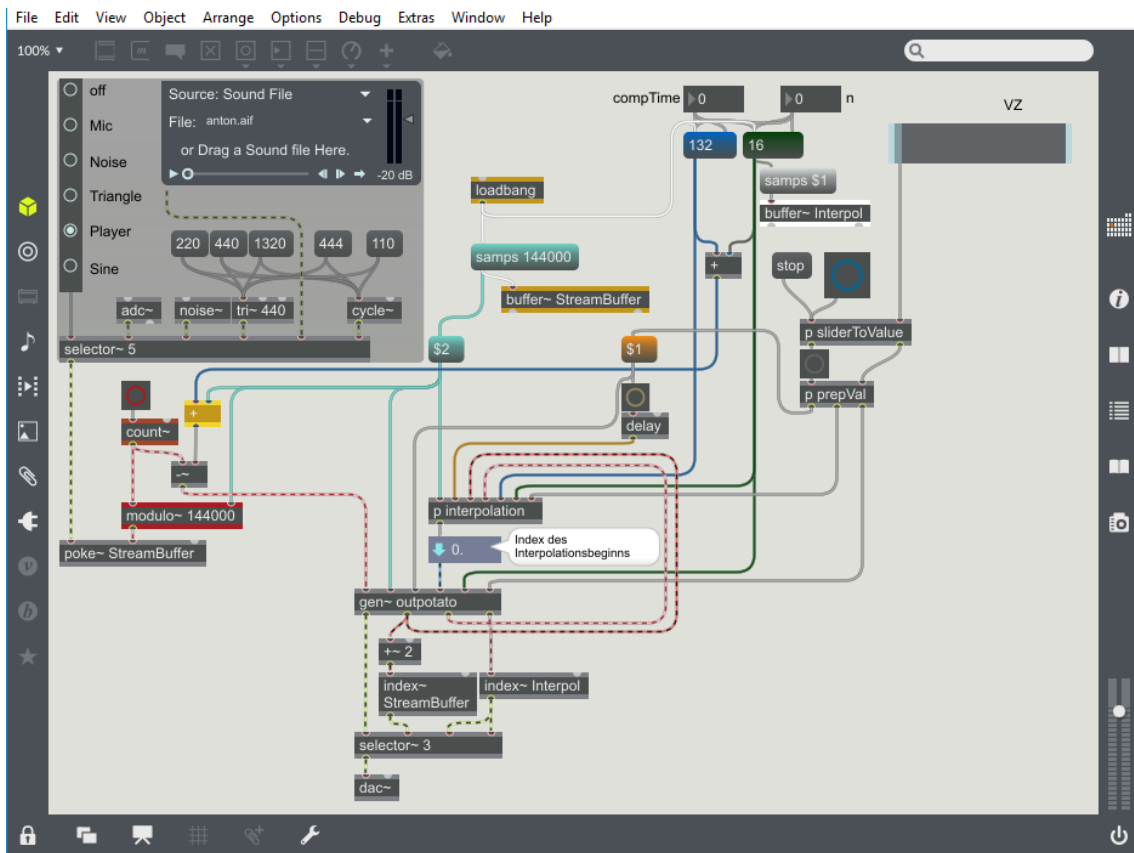


Da der blender-Subpatch durch die hohe Anzahl an Rampenfunktionen sehr unübersichtlich ist, ist er rechts auf eine Rampenfunktion (log) reduziert abgebildet.



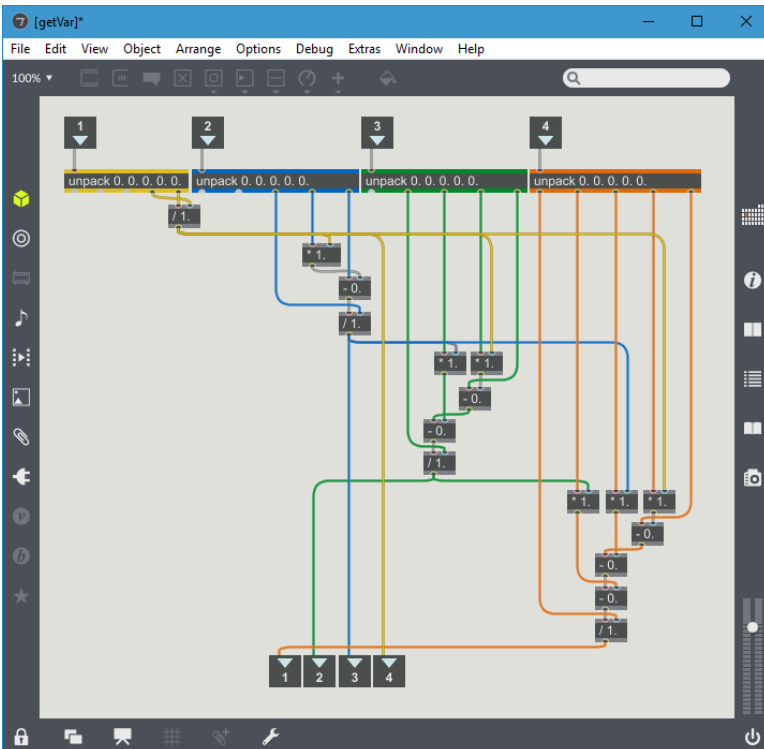
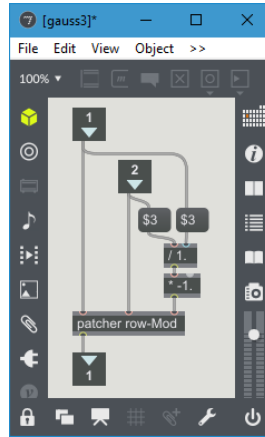
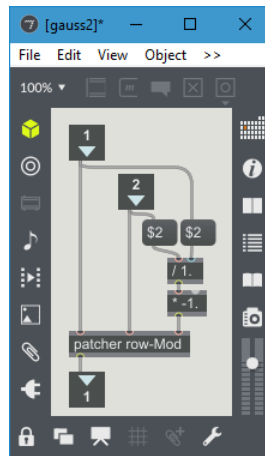
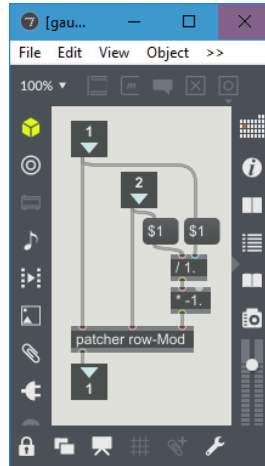
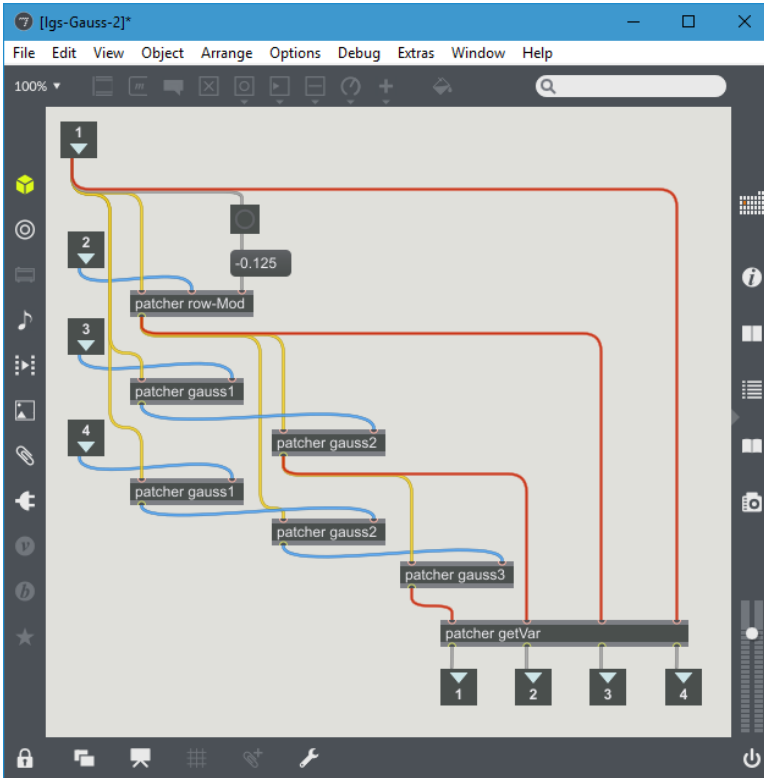


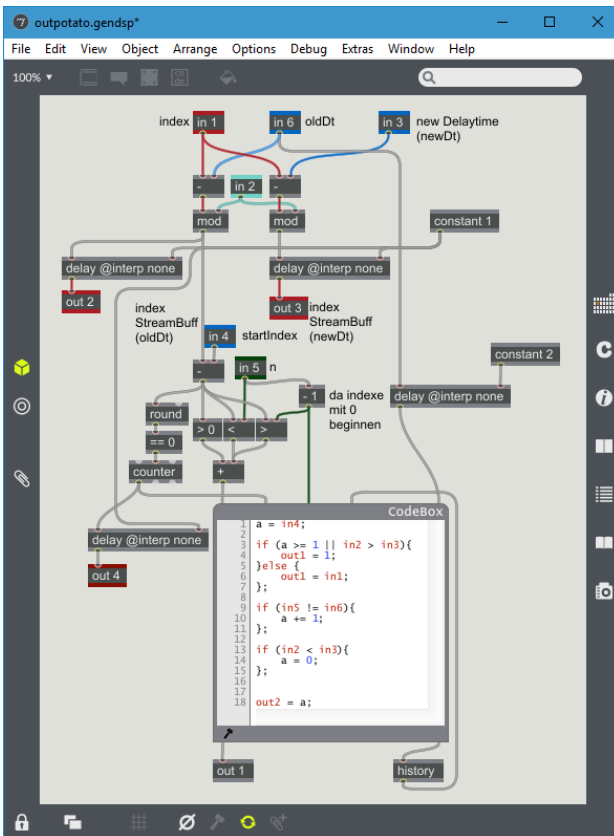
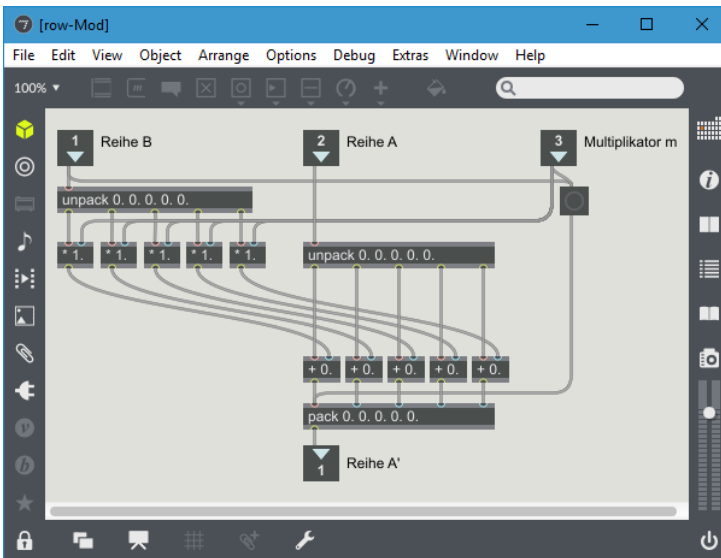
Sprung-Interpolation



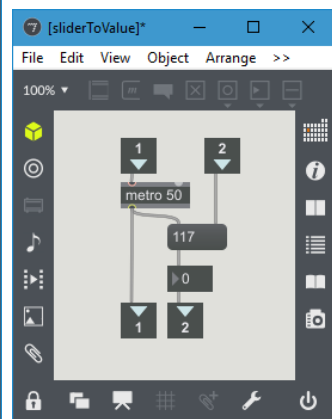
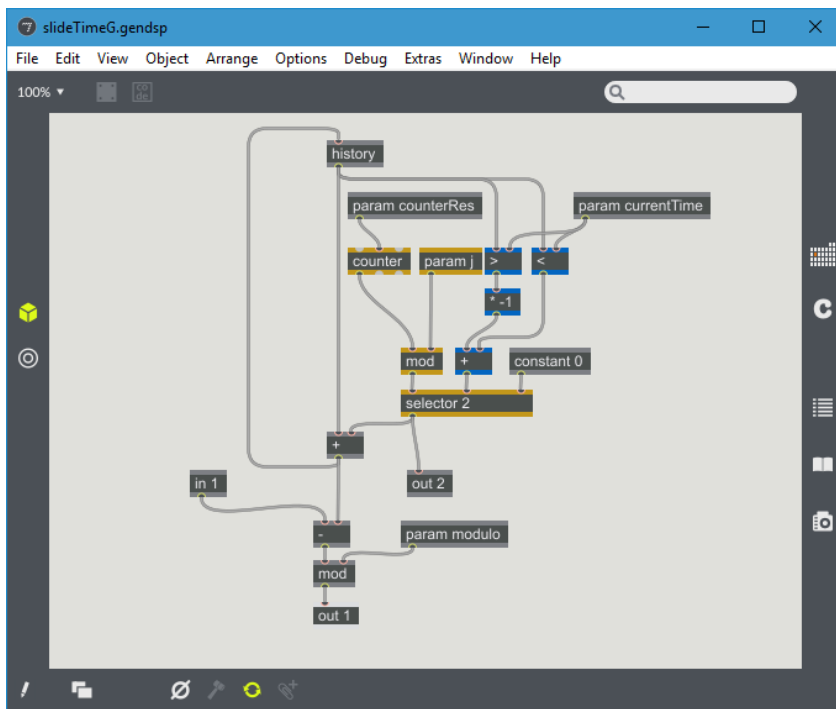
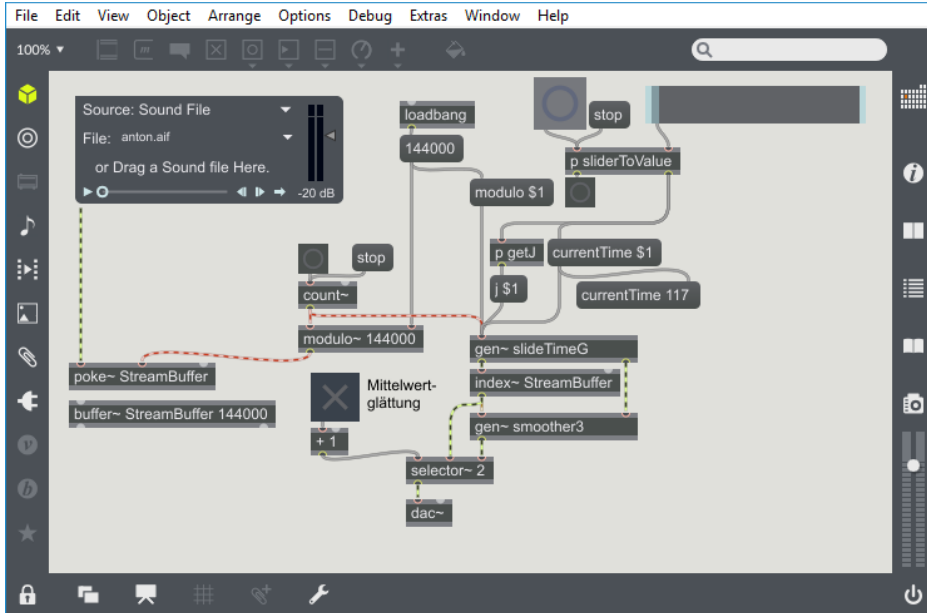
The image displays five Pure Data patch windows, each illustrating a different audio processing or data manipulation technique:

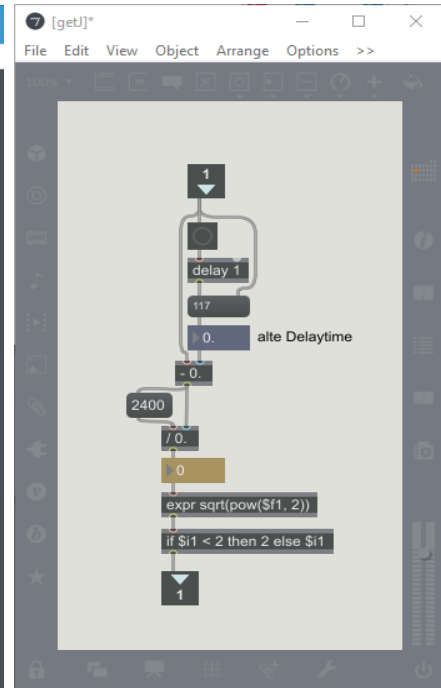
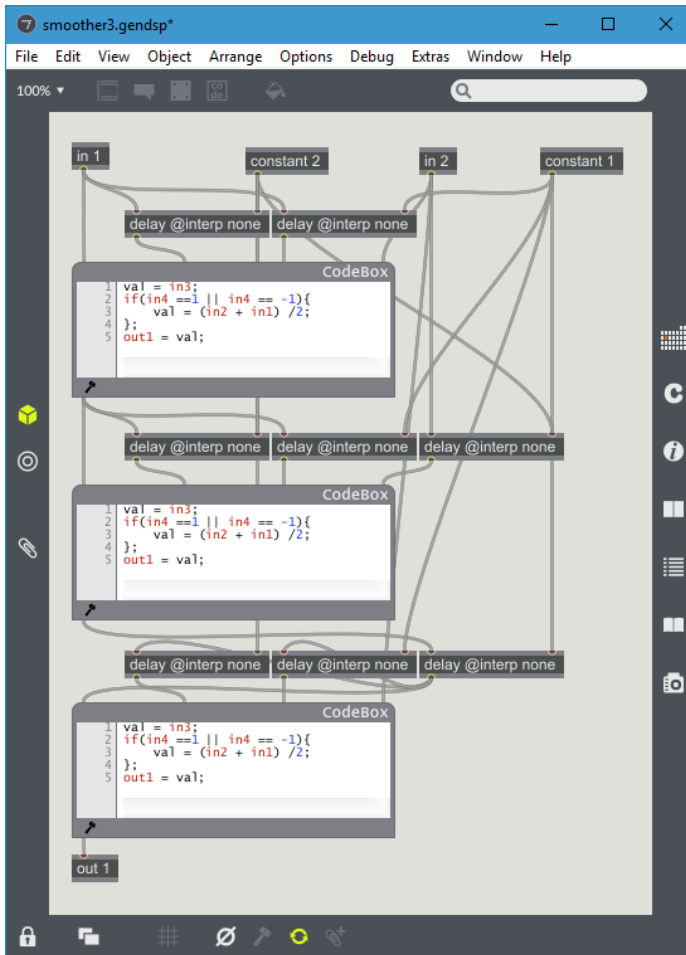
- [sliderToValue]***: A simple patch with two inlets labeled 1 and 2, connected to a 'metro 50' object. The outputs of 'metro 50' are connected to two outlets labeled 1 and 2.
- [poker]**: A patch featuring a 'Uzi' object, a complex exponential expression $\text{exp}(\text{pow}(\$f1, 3) * \$f2) + (\text{pow}(\$f1, 2) * \$f3) - (\$f1 * \$f4) + \$f5$, and a 'poke~ Interpol' object.
- [prepVal]***: A patch with a 'float' object, a '-0.' object, a 'round' object, and a 'delay' object with a 'delaytime' of 5 ms.
- [interpolation]***: A complex patch involving a 'delaytime' of 1 ms, 'Uzi 6' objects, 'snapshot~' objects, 'inter Time' objects, and multiple 'peek~ StreamBuffer' objects. It also includes a 'patcher makeMatrixRows' and 'p lgs-Gauss-2' object.
- [makeMatrixRows]***: A patch with five inlets labeled y(-2), y(-1), y(n+1), y(n+2), and n. It uses 'pack' objects to create four rows of data: 'pack -8. 4 -2. 1. 0. Zeile (a, b, c, d, | y)', 'pack -1. -1. -1. 1. 0. I. Zeile (a, b, c, d, | y)', 'pack 1. 1. 1. 1. 0. III. Zeile (a, b, c, d, | y)', and 'pack 1. -1. 1. 1. 0. IV. Zeile (a, b, c, d, | y)'. The patch uses various signal processing objects like 'float', 'pack', and 'unpack' to process these rows.





Slidealgorithmus





2. MATLAB Code

Da der MATLAB-Prototyp auch Code beinhaltet, der nicht im Rahmen dieser Arbeit entstand, wird hier nur die "while"-Schleife der Audiodberechnung gezeigt.

Slide und Blende:

```

while 1
[currentInputBlockAllInputChannels, overrun] = step(aDR);

if ~strcmp(autoFlyPushButton.String, 'auto fly')
switch flightCurve
case 1
xdT = cos(moveCounter/(2*pi))*3.5+.5;
ydT = -sin(moveCounter/(2*pi))*2-0.3;
moveCounter = moveCounter+speed;
xT(1)=xdT*1.2+2.;
yT(1)=ydT*0.5+1.7;
case 2
if xT(1) < -3
xT(1) = 8;
else
xT(1) = xT(1) - speed;
end
case 3
if xT(1) < 1.5
case3speed = speed*-1;
elseif xT(1) > 8
case3speed = speed;
end
xT(1) = xT(1) - case3speed;
case 4
if moveCounter > 30
moveCounter = 1;
if xT(1) < 0
xT(1) = xT(1)+7;
else
xT(1) = xT(1)-7;
end
end
moveCounter = moveCounter+speed;
case 5
switch speed
case 0.05
if moveCounter > 900*speed
ydT = 0;
elseif moveCounter > 3*speed
ydT = ydT - speed/5;
else
ydT = 4.5;
end
case 0.2
if moveCounter > 500*speed
ydT = 0;
elseif moveCounter > 3*speed
ydT = ydT - speed/5;
else
ydT = 8.5;
end
case 2
if moveCounter > 400*speed
ydT = 0;
elseif moveCounter > 3*speed
ydT = ydT - speed/5;
else
ydT = 6.5;
end
otherwise
if moveCounter > 500*speed
ydT = 0;
elseif moveCounter > 3*speed
ydT = ydT - speed/5;
else
ydT = 8.5;
end
end

xdT(1) = cos(moveCounter/(2*pi))*3.5+.5;
moveCounter = moveCounter+speed;
xT(1)=xdT*1.2+2.5;
yT(1)=max(ydT,1);
end
end
plotHTarget.XData = xT;
plotHTarget.YData = yT;
calcDelays(1)

```



```

newDel = config.delMatrix;
moveSpeed = prevDel - newDel;
showStatusField2.String = string(moveSpeed);
prevDel = newDel;
if uneven == 1
    uneven = 0;
else
    uneven = 1;
end
smooth = strcmp(smoothPushButton.String, 'stop smoother');

currentIndices = currentIndices + interval;
curGain = config.gainMatrix;
processingFrameCount = processingFrameCount + 1;

currentMaxValsLevel = max(abs(currentInputBlockAllInputChannels));
for k = 1:anzIn
    currentMaxLevel = max(1, min(size(colors, 1), -round(20*log10(currentMaxValsLevel(k)))));
    matrixPanel.UserData.inNumberText(k).BackgroundColor = colors(currentMaxLevel, :);
end

linearOutputChannelProcessing = zeros(1, size(config.muteMatrix, 2), size(config.muteMatrix, 1));
linearOutputChannelProcessing(1, :, :) = (config.muteMatrix.*config.polMatrix)';
InputStore = circshift(InputStore, -interval);
InputStore(end-(interval-1):end, :) = currentInputBlockAllInputChannels;

if strcmp(changeModePushButton.String, 'auto')
    doSlide(:) = max(max(sqrt(moveSpeed.^2))) < maxSlideDif;
    if doSlide
        showStatusField.String = 'Slide ';
    else
        showStatusField.String = 'Blend ';
    end
end

for n = 1:anzIn
    switch doSlide(n)##### Slide #####
    case 1
        delDif = currDel - newDel;
        for k = 1:anzOut
            if delDif(k,n) %~= 0
                jumpSize(k,n) = sqrt(ceil(interval/delDif(k,n).^2));
            else
                jumpSize(k,n) = 0;
            end
            if currDel(k,n) == newDel(k,n)
                delayBuffer(:, k,n) = InputStore(end-(interval+currDel(k,n))+1:end-currDel(k,n),n);
            else
                for jump = 1:ceil(sqrt(delDif(k,n).^2))+1
                    jStep = min(max((jump-1)*jumpSize(k,n), 1), length(delayBuffer)):min(jump*jumpSize(k,n), length(delayBuffer));

                    if jump == ceil(sqrt(delDif(k,n).^2))+1 && jStep(end)<interval
                        jStep = jStep:interval;
                    end
                    delayBuffer(jStep,k,n) = InputStore(end-(length(delayBuffer) + currDel(k,n) + jStep,n);
                    if currDel(k,n) < newDel(k,n)
                        currDel(k,n) = currDel(k,n) + 1;
                        if processingFrameCount > 1 && smooth
                            stepAdd1 = min(jStep(1) + 1, interval);
                            stepAdd2 = min(jStep(1) + 2, interval);
                            delayBuffer(stepAdd1, k, n) = (delayBuffer(jStep(1), k, n) + delayBuffer(stepAdd2, k, n))/2;
                            if jStep(1) == 1
                                delayBuffer(jStep(1), k, n) = (oldBufferEnd(k,n) + delayBuffer(stepAdd1, k, n))/2;
                            else
                                delayBuffer(jStep(1), k, n) = (delayBuffer(jStep(1)-1, k, n) + delayBuffer(stepAdd1, k, n))/2;
                            end
                            delayBuffer(stepAdd1, k, n) = (delayBuffer(jStep(1), k, n) + delayBuffer(stepAdd2, k, n))/2;
                        end
                    elseif currDel(k,n) > newDel(k,n)
                        currDel(k,n) = currDel(k,n) - 1;
                        if processingFrameCount > 1 && smooth
                            delayBuffer(min(jStep(end), interval), k, n) = (delayBuffer(jStep(end)-1, k, n) + InputStore(end-(length(delayBuffer) + currDel(k,n) + jStep(end), n))/2);
                            delayBuffer(min(jStep(end)-1, interval), k, n) = (delayBuffer(jStep(end)-2, k, n) + delayBuffer(jStep(end), k, n))/2;
                        end
                    end
                end
            end
        end
        ChProcessingBuffer = delayBuffer();
        oldBufferEnd(:, n) = delayBuffer((end-1), :, n);
        for k = 1:anzOut
            startGain = oldGain(n,k);
            endGain = curGain(n,k);
            inRamp = ones(config.B, 1);
            inRamp(:) = (endGain-startGain)*slideGainRamp(:)+startGain;
            ChProcessingBuffer(:, k,n) = ChProcessingBuffer(:, k,n) .* inRamp;
        end
        oldDel = currDel;
    case 0 ##### Blend #####

        for k = 1:anzOut
            blendBuffer(:, k, n, active(n)) = InputStore(end -(interval + oldDel(k, n))+1:end-oldDel(k, n), n);
            blendBuffer(:, k, n, passive(n)) = InputStore(end-(interval + newDel(k, n))+1:end-newDel(k, n), n);
        end
        for k = 1:anzOut
            endGain = curGain(n,k);
            startGain = oldGain(n,k);
            outRamp = startGain-blendInRamp*startGain;
            inRamp = blendInRamp*endGain;
            ChProcessingBuffer(:, k, n) = (blendBuffer(:, k, n, active(n)).*outRamp) + (blendBuffer(:, k, n, passive(n)).*inRamp);
            oldDel(k, n) = newDel(k, n);
        end
    end
end

```

```

        active(n) = 3-active(n);
        passive(n) = 3-active(n);
        currDel = oldDel;
    end
end
ChProcessingBuffer = ChProcessingBuffer.*repmat(linearOutputChannelProcessing,intervall,1);
currentOutputBlock = sum(ChProcessingBuffer,3);
if ~strcmp(processPushButton.String,'Stop')
    fadeOutFactors = (max(0,linspace(1-fadeOutCount/(fadeOutFrames),1-(fadeOutCount+1)/(fadeOutFrames),config.B)).^2);
    fadeOutCount = fadeOutCount + 1;
    if fadeOutCount < fadeOutFrames
        currentOutputBlock = currentOutputBlock .* repmat(fadeOutFactors,1,anzOut);
    else
        for k = 1:10; step(aDW,zeros(config.B,anzOut)); end; break
    end
end
underrun = step(aDW,currentOutputBlock*10^(config.outputVolume/20));

if underrun > 0
    fprintf('Audio writer queue was underrun by %d samples.\n',underrun);
end
if overrun > 0
    fprintf('Audio reader queue was overrun by %d samples.\n',overrun);
end

drawnow limitrate
oldGain = curGain;
step(fileWriter,currentOutputBlock);
end

```

Slide mit Blendglättung:

```

while 1
%AudioDatenInput
[currentInputBlockAllInputChannels,overrun] = step(aDR);
InputStore = circshift(InputStore,-intervall);
InputStore(end-(intervall-1):end,:) = currentInputBlockAllInputChannels;

%Flugkurven
if ~strcmp(autoFlyPushButton.String,'auto fly')
    switch flightCurve
    case 1
        xdT = cos(moveCounter/(2*pi))*3.5+.5;
        ydT = -sin(moveCounter/(2*pi))*2-0.3;
        moveCounter = moveCounter+speed;

        xT(1)=xDT*1.2+2.;
        yT(1)=ydT*0.5+1.7;
    case 2
        if xT(1) < -3
            xT(1) = 8;
        else
            xT(1) = xT(1)-speed;
        end
    case 3
        if xT(1) < 1.5
            case3speed = speed*-1;
        elseif xT(1) > 8
            case3speed = speed;
        end
        xT(1) = xT(1) - case3speed;
    case 4
        if moveCounter > 30
            moveCounter = 1;
            if xT(1) < 0
                xT(1) = xT(1)+7;
            else
                xT(1) = xT(1)-7;
            end
        end
        moveCounter = moveCounter+speed;
    case 5
        switch speed
        case 0.05
            if moveCounter > 900*speed
                ydT = 0;
            elseif moveCounter > 300*speed
                ydT = ydT-speed/5;
            else
                ydT = 6.5;
            end
        case 0.2
            if moveCounter > 500*speed
                ydT = 0;
            elseif moveCounter > 300*speed
                ydT = ydT-speed/5;
            else
                ydT = 8.5;
            end
        case 2
            if moveCounter > 400*speed
                ydT = 0;
            elseif moveCounter > 200*speed
                ydT = ydT-speed/5;
            else

```

```

        ydT = 6.5;
    end
    otherwise
        if moveCounter > 500*speed
            ydT = 0;
        elseif moveCounter > 300*speed
            ydT = ydT-speed/5;
        else
            ydT = 8.5;
        end
    end

    xdT(1) = cos(moveCounter/(2*pi))*3.5+.5;
    moveCounter = moveCounter+speed;
    xT(1)=xdT*1.2+2.5;
    yT(1)=max(ydT,1);
end
else
end
plotHTarget.XData = xT;
plotHTarget.YData = yT;
calcDelays(1)

%inputPegel-Anzeige
currentMaxValsLevel = max(abs(currentInputBlockAllInputChannels));
for k = 1:anzIn
    currentMaxLevel = max(1,min(size(colors,1),-round(20*log10(currentMaxValsLevel(k)))));
    matrixPanel.UserData.inNumberText(k).BackgroundColor = colors(currentMaxLevel,:);
end

%Del, Gain und co aktualisieren
testCounter = testCounter+1;
newDel =config.delMatrix;
delDif = currDel - newDel;
currentIndices = currentIndices + intervall;
currGain = config.gainMatrix;
processingFrameCount = processingFrameCount + 1;
linearOutputChannelProcessing(1,,:) = (config.muteMatrix.*config.polMatrix)';

%Algo-Processing Multishortblend-Slide
for n = 1:anzIn
    for k = 1:anzOut

        if delDif(k,n) % ~= 0 %JumpSize
            jumpSize(k,n) = sqrt(ceil(intervall/delDif(k,n))^2);
        else
            jumpSize(k,n) = 0;
        end

        if currDel(k,n) == newDel(k,n)
            delayBuffer(:, k,n) = InputStore(end-(intervall+blendDel(k,n))+1:end-blendDel(k,n),n);
        else
            for jump = 1:ceil(sqrt(delDif(k,n).^2))+1
                jStep = min(max((jump-1)*jumpSize(k,n),1),length(delayBuffer)):min(jump*jumpSize(k,n),length(delayBuffer));
                if jump == ceil(sqrt(delDif(k,n).^2))+1 && jStep(end)<intervall
                    jStep = jStep:intervall;
                end
                jStepRamp = min(rampCount,length(jStep));
                jStepRest = min(rampCount+1,length(jStep));
                if jStepRamp == rampCount
                    delayBuffer(jStep(1:jStepRamp),k,n) = InputStore(end-intervall-currDel(k,n)+jStep(1:jStepRamp), n).* blendIn-
                    blendDel(k, n)+jStep(1:jStepRamp), n).*outRamp;
                    blendDel(k,n) = currDel(k,n);
                    delayBuffer(jStep(jStepRest:end),k,n) = InputStore(end-intervall-blendDel(k,n)+jStep(jStepRest:end), n);
                else
                    if currDel(k,n) < newDel(k,n)
                        currDel(k,n) = min(currDel(k,n) + 1, newDel(k,n));
                    elseif currDel(k,n) > newDel(k,n)
                        currDel(k,n) = max(currDel(k,n) - 1, newDel(k,n));
                    end
                end
                delayBuffer(jStep,k,n) = InputStore(end-intervall-blendDel(k,n)+jStep, n);%*endGain (ist hier 1)
            end
        end
    end
end
end
ChProcessingBuffer = delayBuffer();
%% % GainRamping
for k = 1:anzOut
    startGain = oldGain(n,k);
    endGain = currGain(n,k);
    inRamp = ones(config.B,1);
    inRamp(:) = (endGain-startGain)*slideGainRamp(:)+startGain;
    ChProcessingBuffer(:,k,n) = ChProcessingBuffer(:,k,n) .* inRamp;
end
ChProcessingBuffer = ChProcessingBuffer.*repmat(linearOutputChannelProcessing,intervall,1);

%% % Output
currentOutputBlock = sum(ChProcessingBuffer,3);
if ~strcmp(processPushButton.String,'Stop')
    fadeOutFactors = (max(0,linspace(1-fadeOutCount/(fadeOutFrames),1-(fadeOutCount+1)/(fadeOutFrames),config.B))').^2;
    fadeOutCount = fadeOutCount + 1;
    if fadeOutCount < fadeOutFrames
        currentOutputBlock = currentOutputBlock .* repmat(fadeOutFactors,1,anzOut);
    else
        for k = 1:10; step(aDW,zeros(config.B,anzOut)); end; break
    end
end

underrun = step(aDW,currentOutputBlock*10^(config.outputVolume/20));

```

```
if underrun > 0
    fprintf('Audio writer queue was underrun by %d samples.\n',underrun);
end
if overrun > 0
    fprintf('Audio reader queue was overrun by %d samples.\n',overrun);
end
drawnow limitrate
    oldGain = currGain;
    step(fileWriter,currentOutputBlock);
end
```

3. Hörversuche

Foto des Versuchaufbaus (die Line-Arrays an der Decke wurden für den Hörversuch nicht mitverwendet)

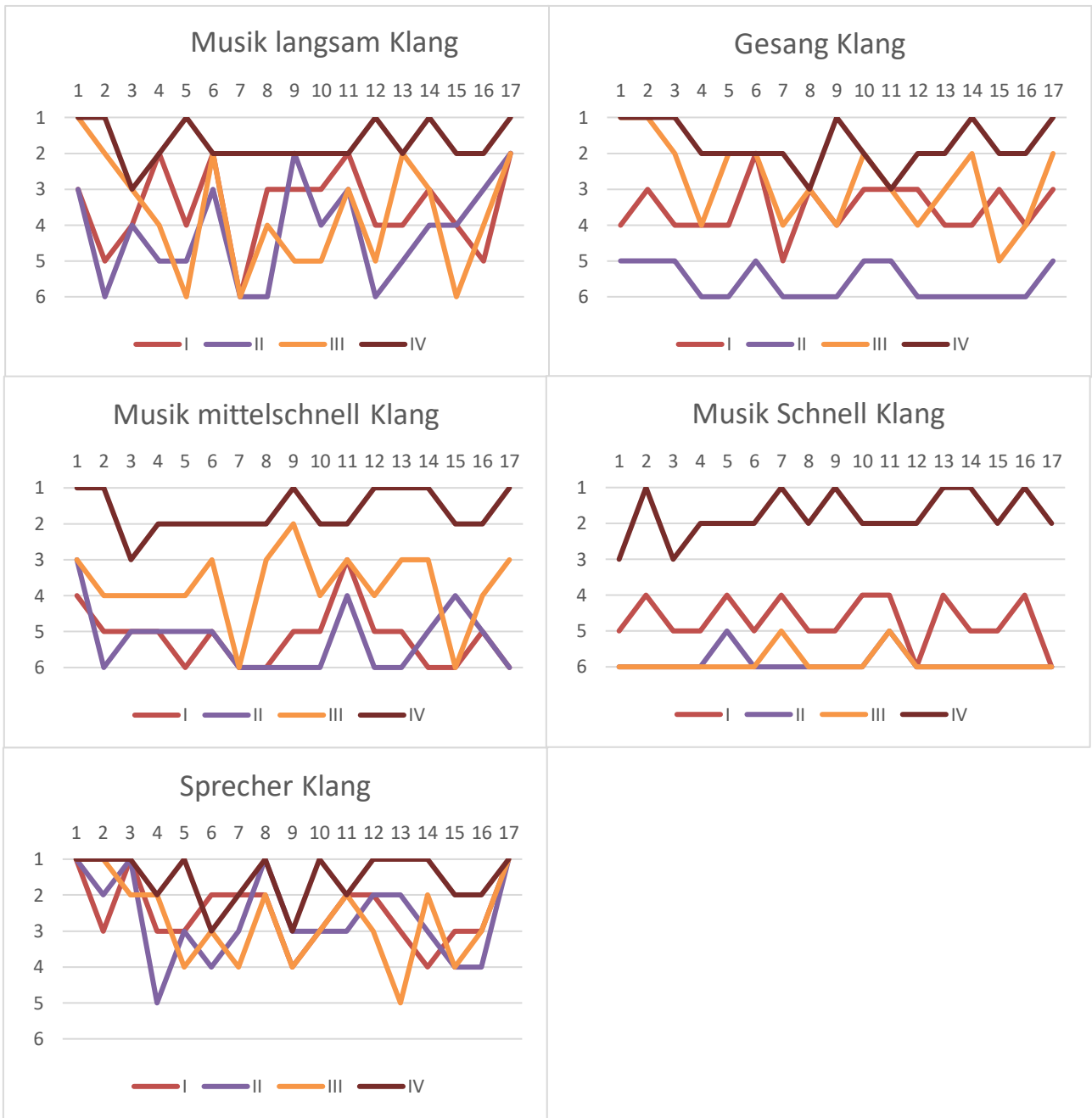


Auswertung

		Lokali- sation	Klang		Lokali- sation	Klang		Lokali- sation	Klang		Lokali- sation	Klang
Testperson 1												
1.Gesang	I	1	4	II	1	5	III	1	1	IV	1	1
1.Musik_langsam	I	1	3	II	1	3	III	2	1	IV	2	1
1.Musik_mittel	I	1	4	II	1	3	III	2	3	IV	2	1
1.Musik_schnell	I	3	5	II	3	6	III	3	6	IV	4	3
1.Sprache	I	1	1	II	2	1	III	2	1	IV	3	1
Testperson 2												
2.Gesang	I	3	3	II	5	5	III	4	1	IV	2	1
2.Musik_langsam	I	2	5	II	4	6	III	3	2	IV	5	1
2.Musik_mittel	I	2	5	II	4	6	III	3	4	IV	5	1
2.Musik_schnell	I	1	4	II	4	6	III	4	6	IV	5	1
2.Sprache	I	1	3	II	1	2	III	3	1	IV	1	1
Testperson 3												
3.Gesang	I	2	4	II	3	5	III	3	2	IV	2	1
3.Musik_langsam	I	3	4	II	2	4	III	3	3	IV	2	3
3.Musik_mittel	I	2	5	II	2	5	III	2	4	IV	2	3
3.Musik_schnell	I	2	5	II	3	6	III	3	6	IV	4	3
3.Sprache	I	2	1	II	2	1	III	2	2	IV	2	1
Testperson 4												
4.Gesang	I	2	4	II	2	6	III	3	4	IV	4	2
4.Musik_langsam	I	2	2	II	3	5	III	2	4	IV	2	2
4.Musik_mittel	I	3	5	II	4	5	III	4	4	IV	5	2
4.Musik_schnell	I	3	5	II	2	6	III	3	6	IV	5	2
4.Sprache	I	3	3	II	4	5	III	2	2	IV	3	2
Testperson 5												
5.Gesang	I	3	4	II	2	6	III	4	2	IV	4	2
5.Musik_langsam	I	5	4	II	5	5	III	6	6	IV	2	1
5.Musik_mittel	I	2	6	II	2	5	III	5	4	IV	3	2
5.Musik_schnell	I	5	4	II	4	5	III	4	6	IV	5	2
5.Sprache	I	4	3	II	3	3	III	3	4	IV	5	1
Testperson 6												
6.Gesang	I	1	2	II	2	5	III	2	2	IV	3	2
6.Musik_langsam	I	2	2	II	2	3	III	1	2	IV	3	2
6.Musik_mittel	I	2	5	II	2	5	III	2	3	IV	4	2
6.Musik_schnell	I	2	5	II	2	6	III	2	6	IV	4	2
6.Sprache	I	2	2	II	2	4	III	2	3	IV	3	3

Testperson 7												
7.Gesang	I	4	5	II	4	6	III	4	4	IV	2	2
7.Musik_langsam	I	4	6	II	4	6	III	5	6	IV	3	2
7.Musik_mittel	I	3	6	II	3	6	III	3	6	IV	4	2
7.Musik_schnell	I	2	4	II	4	6	III	5	5	IV	3	1
7.Sprache	I	4	2	II	3	3	III	2	4	IV	4	2
Testperson 8												
8.Gesang	I	2	3	II	2	6	III	3	3	IV	5	3
8.Musik_langsam	I	3	3	II	4	6	III	4	4	IV	5	2
8.Musik_mittel	I	2	6	II	3	6	III	3	3	IV	4	2
8.Musik_schnell	I	3	5	II	2	6	III	3	6	IV	5	2
8.Sprache	I	2	2	II	2	1	III	3	2	IV	4	1
Testperson 9												
9.Gesang	I	2	4	II	3	6	III	2	4	IV	3	1
9.Musik_langsam	I	3	3	II	3	2	III	3	5	IV	4	2
9.Musik_mittel	I	2	5	II	2	6	III	2	2	IV	3	1
9.Musik_schnell	I	3	5	II	3	6	III	4	6	IV	3	1
9.Sprache	I	3	4	II	3	3	III	3	4	IV	3	3
Testperson 10												
10.Gesang	I	2	3	II	2	5	III	2	2	IV	2	2
10.Musik_langsam	I	2	3	II	3	4	III	2	5	IV	3	2
10.Musik_mittel	I	2	5	II	3	6	III	3	4	IV	4	2
10.Musik_schnell	I	4	4	II	4	6	III	4	6	IV	5	2
10.Sprache	I	2	3	II	2	3	III	2	3	IV	3	1
Testperson 11												
11.Gesang	I	3	3	II	2	5	III	2	3	IV	4	3
11.Musik_langsam	I	2	2	II	2	3	III	3	3	IV	5	2
11.Musik_mittel	I	2	3	II	2	4	III	2	3	IV	3	2
11.Musik_schnell	I	2	4	II	2	5	III	2	5	IV	4	2
11.Sprache	I	3	2	II	2	3	III	3	2	IV	4	2
Testperson 12												
12.Gesang	I	1	3	II	1	6	III	2	4	IV	3	2
12.Musik_langsam	I	1	4	II	2	6	III	2	5	IV	3	1
12.Musik_mittel	I	2	5	II	2	6	III	1	4	IV	3	1
12.Musik_schnell	I	2	6	II	1	6	III	3	6	IV	4	2
12.Sprache	I	1	2	II	1	2	III	1	3	IV	2	1
Testperson 13												
13.Gesang	I	4	4	II	3	6	III	3	3	IV	2	2
13.Musik_langsam	I	4	4	II	4	5	III	3	2	IV	2	2

13.Musik_mittel	I	3	5	II	4	6	III	2	3	IV	2	1
13.Musik_schnell	I	3	4	II	4	6	III	2	6	IV	3	1
13.Sprache	I	2	3	II	3	2	III	4	5	IV	2	1
Testperson 14												
14.Gesang	I	1	4	II	1	6	III	1	2	IV	6	1
14.Musik_langsam	I	1	3	II	1	4	III	2	3	IV	5	1
14.Musik_mittel	I	2	6	II	1	5	III	2	3	IV	6	1
14.Musik_schnell	I	2	5	II	2	6	III	2	6	IV	6	1
14.Sprache	I	1	4	II	1	3	III	1	2	IV	4	1
Testperson 15												
15.Gesang	I	2	3	II	4	6	III	5	5	IV	2	2
15.Musik_langsam	I	4	4	II	4	4	III	5	6	IV	2	2
15.Musik_mittel	I	4	6	II	5	4	III	5	6	IV	2	2
15.Musik_schnell	I	5	5	II	6	6	III	6	6	IV	2	2
15.Sprache	I	1	3	II	3	4	III	4	4	IV	2	2
Testperson 16												
16.Gesang	I	2	4	II	3	6	III	3	4	IV	3	2
16.Musik_langsam	I	2	5	II	3	3	III	3	4	IV	4	2
16.Musik_mittel	I	3	5	II	4	5	III	2	4	IV	3	2
16.Musik_schnell	I	5	4	II	3	6	III	3	6	IV	2	1
16.Sprache	I	2	3	II	3	4	III	2	3	IV	3	2
Testperson 17												
17.Gesang	I	1	3	II	2	5	III	3	2	IV	4	1
17.Musik_langsam	I	1	2	II	1	2	III	2	2	IV	4	1
17.Musik_mittel	I	3	6	II	4	6	III	2	3	IV	5	1
17.Musik_schnell	I	2	6	II	4	6	III	2	6	IV	3	2
17.Sprache	I	1	1	II	1	1	III	2	1	IV	3	1



I: Blende, II: künstlich verzerrtes Signal, III: Slide, VI: ohne Verzögerung