Ambisonics Binaural Dekoder Implementation als Audio Plug-in mit Headtracking zur Schallfeldrotation

Matthias Kronlachner

BACHELORARBEIT

eingereicht am

Institut für Elektronische Musik und Akustik
Universität für Musik und darstellende Kunst Graz

Elektrotechnik-Toningenieur

in Graz

im September 2012

Institut für Elektronische Musik und Akustik

Abstract

This thesis deals with the surround sound technique Ambisonics and the possibility of decoding Ambisonics Signals for binaural headphone playback.

A binaural decoder audio plug-in has been developed within the programming framework JUCE. Therefore it supports various plug-in standards (AU, VST, RTAS, LV2) on different operating systems (Windows, Mac OS X, Linux). The decoder is developed for 3rd order 3d Ambisonics. So far no 3rd order 3d encoders and rotators have been available as audio plug-in except for Linux (LADSPA). Therefore an encoder and rotator have been developed to create a test environment for the decoder.

A Microsoft Kinect™camera is used to measure the listeners head rotation. The estimated angles are used to rotate the Ambisonics soundfield in the opposite direction. This allows to keep the position of the sound sources stable in relation to the listener. Therefore the localization capability for the listener can be improved.

Kurzfassung

In dieser Arbeit wird die Spatialisierungsmethode Ambisonics genauer untersucht und auf die Dekodierung von Ambisonics Signalen für binaurale Kopfhörerwiedergabe eingegangen.

Ein Ambisonics Binaural Dekoder wurde als Audio Plug-in entwickelt. Dafür wurde die Programmbibliothek JUCE verwendet, welche die gängigsten Plug-in Formate (AU, VST, RTAS, LV2) und Betriebssysteme (Windows, Mac OS X, Linux) unterstützt und damit entwickelte Software universell einsetzbar macht. Der Dekoder ist für die 3. Ambisonics Ordnung 3D entwickelt. Um eine Testumgebung zu schaffen, wurde ein Enkoder und ein Schallfeldrotator für die 3. Ordnung 3D entwickelt, welche bisher nur für das Betriebssystem Linux verfügbar waren.

Eine Microsoft Kinect™Kamera wird zum Messen der Kopfdrehung verwendet. Adäquat zu den Kopfbewegungen wird das Schallfeld in der Ambisonics Domäne in Gegenrichtung gedreht, und stabil in Relation zur Umwelt gehalten. Dadurch kann die Lokalisationsfähigkeit des Zuhörers mittels Peilbewegungen verbessert werden.

Es wird auf die Implementation des Ambisonics Systems eingegangen sowie deren Anwendung in der Praxis.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Problemstellungen

- Was ist Ambisonics und wie funktioniert es?
- Wie kann ich ein Ambisonics Schallfeld enkodieren, rotieren und dekodieren?
- Wie kann ich ein Ambisonics Signal über Kopfhörer wiedergeben?
- Welche Möglichkeiten gibt es um die Kopfposition des Hörers zu bestimmen?
- Wie implementiere ich diese Algorithmen als Audio Plug-in?

1.2 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll ein Audio Plug-in entwickelt werden, welches sich mit gängigen Digitalen Audioworkstations (DAW) verwenden lässt um Ambisonicssignale 3. Ordnung 3D auf Kopfhörerwiedergabe zu dekodieren.

Das virtuelle Schallfeld sollte sich stabil gegenüber Kopfbewegungen des Hörers verhalten, es soll sich nicht mit dem Kopf des Hörers drehen. Dies kann durch Messung der Kopfposition und einer gegenläufigen Drehung in der Ambisonics Domäne passieren und führt zu einer gesteigerten Lokalisationsfähigkeit.

Diese praxisnahe Lösung sollte Musikproduzenten, Komponisten, Klangkünstler sowie Konsumenten eine Möglichkeit geben, Ambisonics Aufnahmen zu hören oder zu erstellen, ohne eine große Anzahl von Lautsprechern zur Verfügung zu haben.

Die im Rahmen der Arbeit entwickelte Software wird in einem Projektarchiv zur Verfügung gestellt [22].

1.3 Struktur der Arbeit

Zu Beginn der Arbeit werden die für die Richtungswahrnehmung relevanten psychoakustischen Phänomene vorgestellt [Kap. 5] sowie die Möglichkeiten diese zu simulieren.

In Kapitel 3 wird der ursprüngliche Ambisonics Ansatz sowie die Erweiterung zu *Higher Order Ambisonics* erläutert. Es werden Implementierungsprobleme durch unterschiedliche Konventionen bzgl. Kanalordnung und Normalisierung aufgezeigt.

Kapitel 4 geht auf die Arbeitsumgebung (Digitale Audioworkstation) für die Plug-ins ein und stellt exemplarisch den VST Standard vor. Weiters wird auf die Entwicklung von Audio Plug-ins mit Hilfe der JUCE Programmbibliothek eingegangen.

Kapitel 5 beschreibt den Signalverlauf für den Ambisonics Binaural Decoder sowie die Implementation des gesamten System bestehend aus Enkoder, Rotator und Dekoder. Die Testumgebung in der Software Reaper sowie die Plug-in Bedienelemente werden vorgestellt.

Kapitel 6 stellt unterschiedliche Möglichkeiten zur Bestimmung von Kopfpositionen des Hörers vor.

Zum Schluss [Kap. 7] wird noch auf die möglichen Verbesserungen und Erweiterungen eingegangen.

1.4 Surround Sound über Kopfhörer

Es ist auffällig, dass Systeme zur Erschaffung von visuellen virtuellen Realitäten von der Industrie bisher sehr viel mehr Aufmerksamkeit im Hinblick erfahren haben, als Verfahren zur akustischen Realitätsnachbildung. Für grafisches Rendering haben sich zumindest 2 Systeme am Computer etabliert: OpenGL [1] und DirectX[2]. Von OpenGL abgeleitet entwickelte sich OpenAL (Open Audio Library) welches jedoch noch nicht so eine Verbreitung wie OpenGL für Grafik erzielen konnte.

Die folgende Aufzählung legt keinen Wert auf Vollständigkeit aber soll einige zur Zeit erhältliche Systeme für Surround Sound über Kopfhörer vorstellen.

Headphone Surround 3D

Die Firma *New Audio Technology GmbH* bietet mit ihrem Audio Plug-in *Spatial Audio Designer*[30] (Windows, Mac OS X) die Möglichkeit, Lautsprechersignale virtuell im Raum zu platzieren und für Kopfhörerwiedergabe

[1] Open Graphics Library (OpenGL) ist eine plattformunabhängige Programmierschnittstelle zur Entwicklung von 2D und 3D Computergrafik

[2] DirectX ist eine von Microsoft für Windows entwickelte Programmierschnittstelle für Multimedia Anwendungen

zu optimieren.

WaveArts Panorama 5 [40]

Von der Firma *Wave Arts* ist das Audio Plug-in *Panorama 5* (Windows, Mac OS X) erhältlich. Es wurde entwickelt um Mono oder Stereo Schallquellen räumlich zu platzieren. Dabei besteht die Möglichkeit einen variablen virtuellen Raum mit Erstreflexionen und diffusem Nachhall zu simulieren. Zusätzlich kann für Bewegungen der Schallquelle eine Simulation des Dopplereffekts[3] hinzugeschaltet werden.

Die Ausgabe kann für Kopfhörer- oder Lautsprecherwiedergabe mittels crosstalk cancellation[4] optimiert werden.

Abbildung 1.1: WaveArts Panorama 5 [40]

Harpex Ambisonics Decoder [18]

Harpex (High Angular Resolution Planewave Expansion) ist ein parametrischer Ambisonics Decoder. Basierend auf der Technologie beschrieben in [4] kann ein Ambisonics Signal erster Ordnung hinsichtlich seiner Lokalisationsschärfe verbessert werden. Die als Plug-in oder Standalone erhältliche Software ist vor allem für die Verarbeitung von Aufnahmen mit einem Soundfield Mikrofon entwickelt worden (siehe Kap. 3.2.2). Als Ausgabeformat kann unter anderem Binaurale Wiedergabe gewählt werden.

[3] Der Dopplereffekt beschreibt die Veränderung der wahrgenommenen Frequenz durch Annäherung oder Entfernung der Schallquelle in Relation zum Hörer.

[4] Crosstalk canellation versucht das Übersprechen von den jeweils vom Ohr aus gegenüberliegenden Lautsprechern zu minimieren um eine zur Kopfhörerwiedergabe ähnliche Situation zu erreichen.

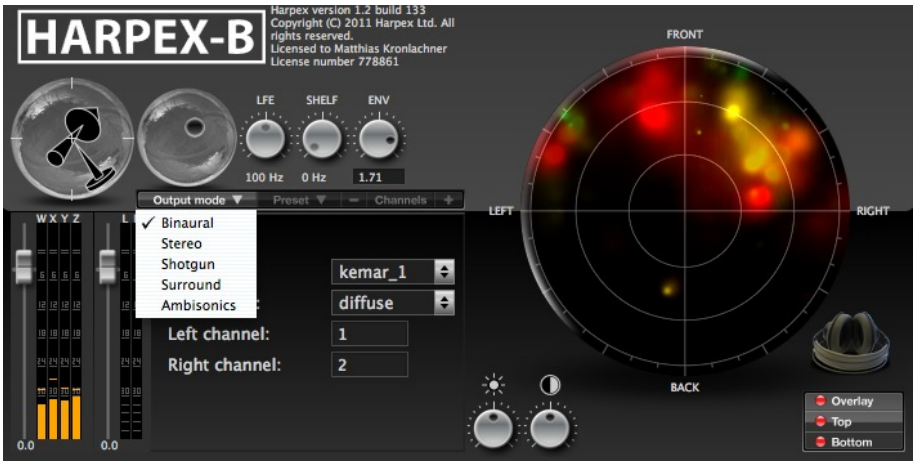

Abbildung 1.2: Harpex Ambisonics Decoder [18]

Dolby Headphone [13]

Dolby Headphone ist konzipiert um 5.1 oder 7.1 Surround Inhalte über Kopfhörer wiederzugeben. Es findet in Software DVD-Playern, HiFi Systemen, Notebooks und Headsets Verwendung. Neben den Außenohrübertragungsfunktionen wird auch ein vom Anwender in Presets veränderbarer Raumklang (Erstreflexionen, Nachhall) simuliert. Headtracking ist bei diesem System zurzeit nicht vorgesehen.

SRS Headphone [34]

SRS Labs hat ähnlich wie *Dolby* Algorithmen im Angebot, um die Kopfhörerwiedergabe räumlicher zu machen. Es unterstützt dabei kodiertes Stereo Material (SRS Circle Surround) welches wieder auf Front/Rear dekodiert werden kann.

Yamaha Silent Cinema [44]

Die Firma *Yamaha* setzt die *Silent Cinema Technology* in ihren HiFi Verstärkern ein, um ein Surround Erlebnis über Kopfhörer hörbar zu machen.

Beyerdynamic Headzone [5]

Das einzige erhältliche Surround-Sound System für Kopfhörer mit Headtracking bietet *Beyerdynamic* mit ihrer *Headzone* Produktlinie. Für die Headtracking Möglichkeit muss man jedoch auf die Professionelle Variante zurückgreifen, die Gamer - also Consumer - Variante bietet keine unterstützung für Headtracking.

Das *Headzone* System ist zum Professionellen Monitoring von 5.1 Audiomaterial gedacht und bietet die Möglichkeit die Lautsprecher im virtuellen Raum beliebig zu positionieren und den Abstand vom Zuhörer zu den Lautsprechern zu verändern. Der simulierte Raum kann in Größe und Beschaffenheit angepasst werden. Das Headtracking funktioniert mittels Ultraschall.

Abbildung 1.3: Beyerdynamic Headzone [5]

1.5 Experimentelle Software

Binaural-Ambisonic 4.Ordnung 3D-Raumsimulationsmodell mit ortsvarianten Quellen und Hörerin bzw. Hörer für PD [27]

Diese von Wolfgang Musil in Pure Data[5] programmierte Software ermöglicht es, mehrere Schallquellen sowie den Zuhörer beliebig im Raum zu platzieren. Ein Raummodell mit Erstreflexionen und Nachhall kann verändert werden und unterstützt die Lokalisation. Diese Software implementiert ein 3D Ambisonics System 4. Ordnung und basiert auf den vom selben Author entwickelten Pure Data Bibliotheken *iem_ambi* und *iem_bin_ambi*.

[5]Pure Data ist eine visuelle Programmierumgebung für interaktive Computermusik und Multimedia Anwendungen.

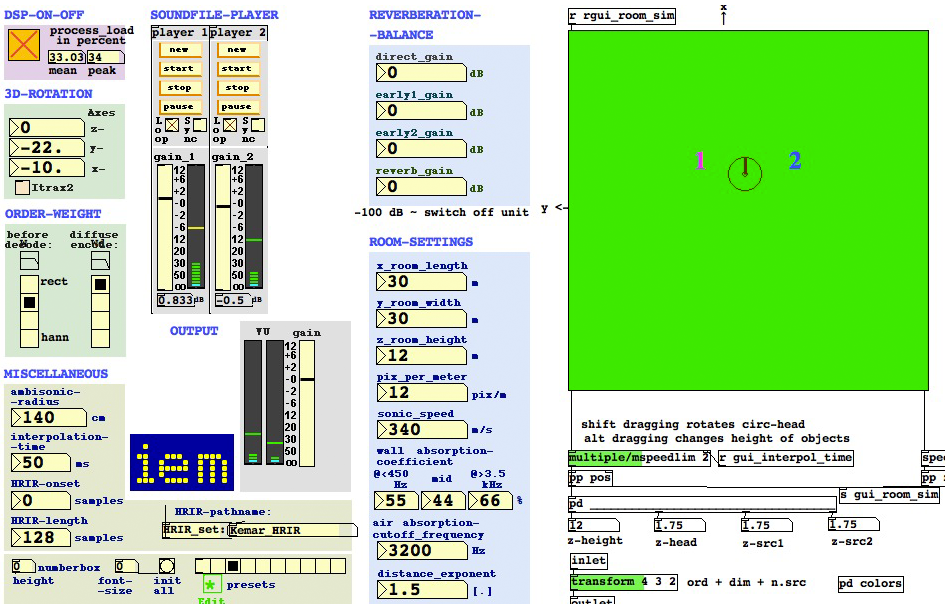

Abbildung 1.4: Binaural-Ambisonics Simulation mit Raummodell in Pure Data [27]

Ambisonic Bidules [12]

Aristotel Digenis stellt auf seiner Homepage eine Serie von Ambisonics Plug-ins für die kommerzielle, modulare Audio Software Bidule[6] zur Verfügung. Diese beinhaltet unter anderem einen Binaural Dekoder für Ambisonics 1. Ordnung 3D.
Diese Plug-ins wurden basierend auf einer Ambisonics C++ Bibliothek implementiert, welche ebenfalls auf der Homepage des Entwicklers verfügbar ist.

[6] Bidule http://www.plogue.com

Kapitel 2

Richtungshören

Das Wahrnehmen der Position einer Schallquelle beruht auf psychoakustischen und neurophysiologischen Modellen. Im folgenden soll kurz auf die psychoakustischen Phänomene eingegangen werden.

2.1 Lokalisation

Lokalisation ist das Zuordnen von Azimuth, Elevation und Entfernung bezüglich einer Schallquelle. Die Wahrnehmung dieser Parameter gliedert sich in Monaurale und Binaurale Faktoren.

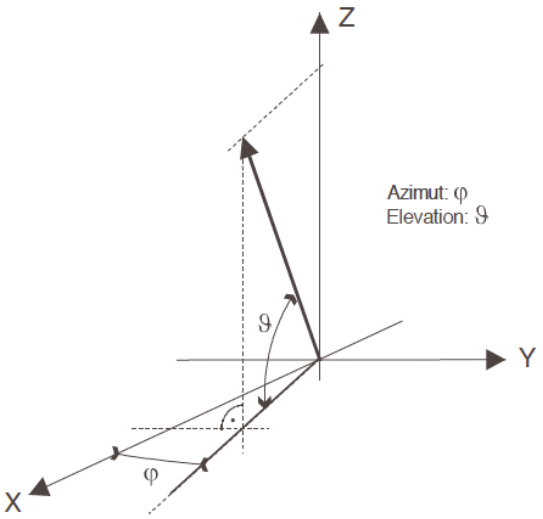

Abbildung 2.1: Winkelbezeichnungen [25]

2.1.1 Monaurale Faktoren

Monaurale (mon(o) - griech. allein, einzig, ein) Faktoren werden von beiden Ohren gleich wahrgenommen. Wenn sich eine Schallquelle von dem Hörer entfernt wird diese durch Luftdämpfung vor allem im höheren Frequenzbereich gedämpft. Diese Klangverfärbung und die Pegelabnahme über das ganze Frequenzband mit der Entfernung setzt ein bekanntes Quellspektrum mit genügend hohen Frequenzanteil voraus.

2.1.2 Binaurale Faktoren

Binaurale Faktoren beschreiben die unterschiedliche Wahrnehmung von dem zur Schallquelle zugewandten und abgewandten Ohr.

Interaurale Zeitdifferenzen (ITD *Interaural Time Difference*)

Laufzeitunterschiede zwischen den Ohren ergeben sich durch Beugung der Schallwellen um den Kopf. Dies ist für Frequenzen unterhalb 1,5 kHz wirksam, wo die Wellenlänge größer als der Kopfdurchmesser ist. ($\lambda > d_{Kopf}$) [25](Seite 6)

Interaurale Pegeldifferenzen (ILD *Interaural Level Difference*)

Frequenzen über 1,5kHz erfahren eine Abschattung durch den Kopf. ($\lambda < d_{Kopf}$) Dadurch entstehen Pegelunterschiede zwischen dem der Quelle zugewandten (*ipsilateralem*) und dem der Quelle abgewandten (*kontralateralem*) Ohr.

Cone of confusion

Durch die Mehrdeutigkeit der ITDs entsteht eine Unsicherheit bezüglich Vorne-Hinten Lokalisation. Eine Verbesserung der vorne-hinten Lokalisation kann durch kleine Kopfbewegungen erreicht werden. Diese müssen von der Person selbst kontrolliert werden, Bewegungen der Quelle haben laut einer Studie keinen Einfluss. [43]

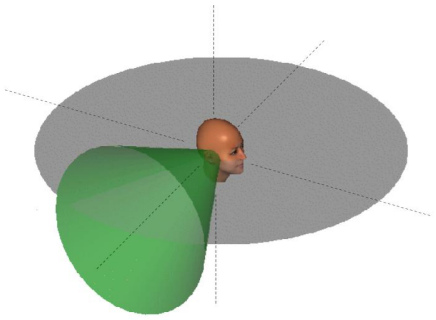

Abbildung 2.2: Cone of confusion [24]

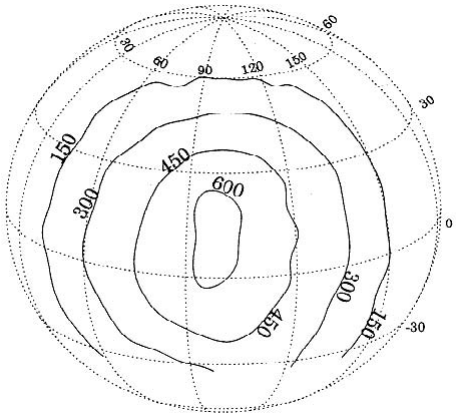

Abbildung 2.3: Kurven gleicher Interaurale Zeitdifferenzen (ITD) (in μs) [16]

2.2 HRTF - Head Releated Transfer Function

Die oben beschriebenen Faktoren können mit Hilfe eines LTI System[1] beschrieben werden. Im Frequenzbereich wird dieser Filter *Head Releated Transfer Function*[2] (HRTF) - im Zeitbereich *Head Releated Impulse Response*[3] (HRIR) genannt.

Diese Übertragungsfunktionen werden mit Mikrofonen in beiden Ohren einer Versuchsperson oder eines Kunstkopfes gemessen. Die Forschungsgruppe *equipe acoustique des salles* am IRCAM stellt auf ihrer Website eine Sammlung von gemessenen HRIRs zur Verfügung. [39] Diese sind in ei-

[1] LTI - Linear und Zeitinvariant (linear time-invariant)
[2] dt. Außenohr Übertragungsfunktion
[3] dt. Außenohr Impulsantwort

ner Auflösung von 15° in Azimuth und Elevation sowie ca. 1 Meter Entfernung zur Schallquelle in einem reflexionsarmen Raum vermessen worden und sind als Stereo Wave-Files oder im MAT Format zum Import für Matlab™erhältlich.

Durch Faltung eines beliebigen Quellsignals mit einem Satz von HRIRs für einen bestimmten Punkt kann somit eine Schallquellenposition über Kopfhörerwiedergabe simuliert werden.

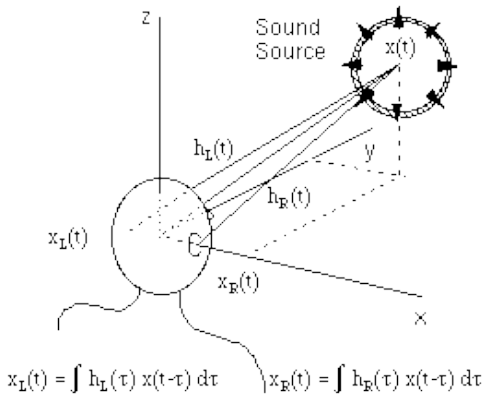

Abbildung 2.4: Lokalisation als Faltung des Quellsignals mit einem Satz von HRIRs

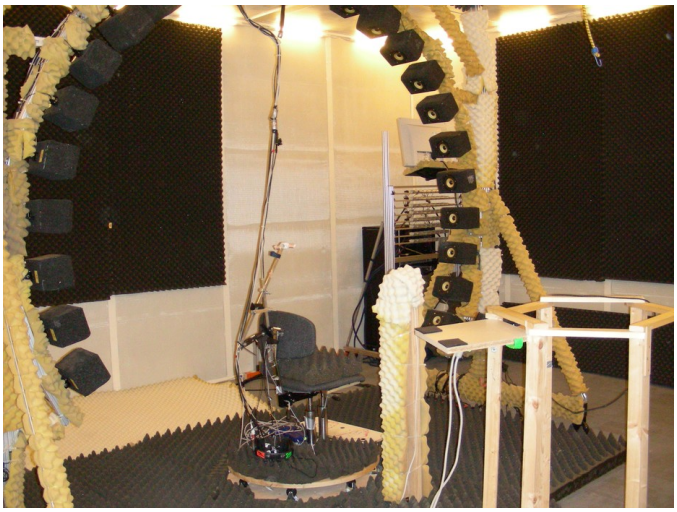

Abbildung 2.5: Anlage zum Messen von HRIRs am Institut für Schallforschung, Wien [24]

Kapitel 3

Ambisonics

3.1 Allgemeines

Ambisonics ist ein Aufnahme- und Wiedergabeverfahren zur Reproduktion von zwei- bzw. dreidimensionalen Schallfeldern über eine variable Lautsprecheranordnung. Bei Ambisonics tragen alle Lautsprecher zur Synthetisierung des Schallfeldes bei. Im Gegensatz dazu wird bei Vector Based Amplitude Panning (VBAP) ein Quellsignal zwischen drei Lautsprecher aufgeteilt.

Die Anfänge von Ambisonics reichen in die frühen 1970er Jahren zurück, wo Michael Gerzon am Mathematischen Institut der Universität Oxford an Surround Verfahren forschte und experimentiere.

Der Vorteil von Ambisonics gegenüber anderen Surround Technologien ist die Einführung eines von der Lautsprecherkonfiguration und der Anzahl an Quellsignalen unabhängigen Zwischenformats. Die Qualität der Schallfeldreproduktion hängt von der Ambisonics Ordnung und der Anordnung und Anzahl an Lautsprechern ab. Die Lautsprecherpositionen des Wiedergabesetups sind von der Produktion des Ambisonics Schallfeldes unabhängig (Abb. 3.1).

Der grundlegende Ambisonics Ansatz basiert auf dem Vergleich einer aufgenommen Schallwelle (Referenzwelle) mit der durch einen Lautsprecher synthetisierten Schallwelle. Es wird davon ausgegangen, dass sowohl die aufgenommene Schallwelle als auch die abgestrahlte Schallwelle vom Lautsprecher einer ebenen Welle entspricht. Dies trifft dann zu, wenn die aufgenommene Schallquelle sowie die wiedergebenden Lautsprecher weit genug von der Abhörposition entfernt sind (Freifeld Bedingung).

Ambisonics Enkodierung, Manipulation und Dekodierung wird durch eine Reihe von einfachen Matrixmultiplikationen und Additionen ermöglicht.

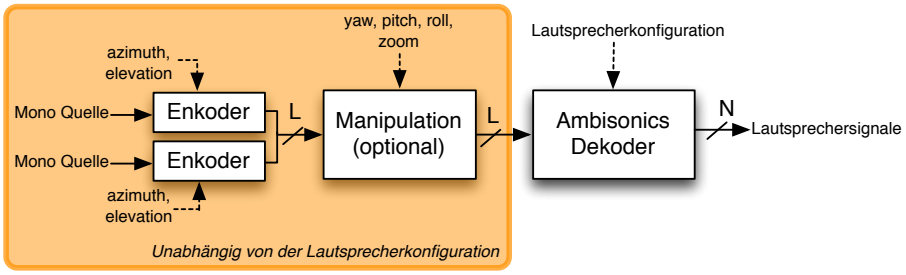

Abbildung 3.1: Ambisonics Blockdiagramm

3.2 Ambisonics Erster Ordnung

3.2.1 Enkodierung - B-Format

Die Theorie zu Ambisonics erster Ordnung wurde von Michael A. Gerzon im Jahr 1970 entwickelt. Es wird ein 3-dimensionales Schallfeld in 4 Kanäle (W, X, Y und Z) kodiert. Diese Kanäle bezeichnet man als B-Format. Mit folgender Enkodiervorschrift können Monoquellen im B-Format kodiert (positioniert) werden.

$$W = \frac{1}{k}\sum_{i=1}^{k} s_i \left[\frac{1}{\sqrt{2}}\right] \qquad \text{Omnidirektionaler Anteil} \qquad (3.1)$$

$$X = \frac{1}{k}\sum_{i=1}^{k} s_i \left[\cos(\phi_i)\cos(\theta_i)\right] \qquad \text{nach X gerichteter Anteil} \qquad (3.2)$$

$$Y = \frac{1}{k}\sum_{i=1}^{k} s_i \left[\sin(\phi_i)\cos(\theta_i)\right] \qquad \text{nach Y gerichteter Anteil} \qquad (3.3)$$

$$Z = \frac{1}{k}\sum_{i=1}^{k} s_i \left[\sin(\theta_i)\right] \qquad \text{nach Z gerichteter Anteil} \qquad (3.4)$$

Tabelle 3.1: Enkodiervorschrift Ambisonics 1. Ordnung

Dabei beschreibt s_i die einzelnen Monoquellen welche wir an der gegebenen Position mit dem horizontalen Winkel ϕ_i (Azimuth) und dem vertikalen Winkel θ_i (Elevation) enkodieren wollen.

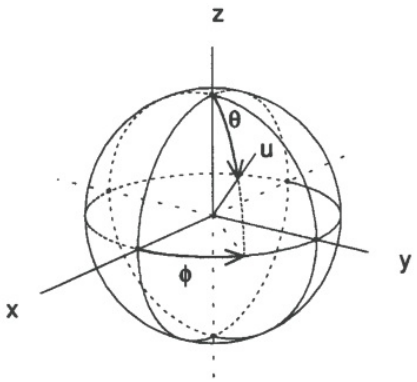

Abbildung 3.2: Sphärisches Koordinatensystem zur Berechnung eines Ambisonics System [25]

Grafisch lassen sich die vier B-Format Kanäle folgendermaßen darstellen:

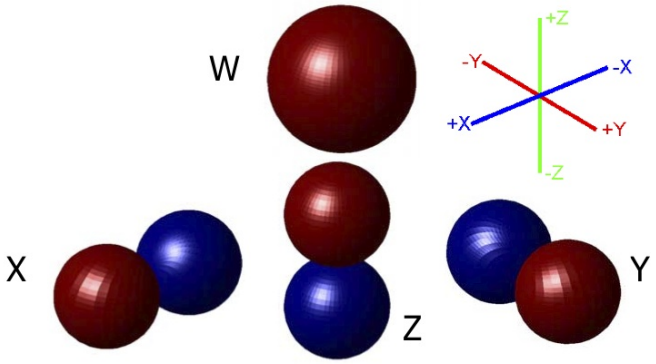

Abbildung 3.3: W, X, Y, Z Kanal - erste Ordnung Ambisonics, rot ist der In-Phase Anteil [41]

Aus der Grafik kann man erkennen, dass diese Signale von einem Omnidirektionalen Mikrofon für den W-Kanal und 3 Mikrofonen mit der Richtcharakteristik Acht für die X, Y, Z Kanäle in einem realen Schallfeld (z.B. Konzertsituation) aufgenommen werden können. Idealerweise sollten diese vier Mikrofone gemeinsam im Ursprung des Koordinatensystems platziert sein, was sich aber praktisch nicht realisieren lässt.

3.2.2 A-Format

In der Praxis werden Ambisonics Aufnahmen erster Ordnung im sogenannten A-Format aufgenommen. Das A-Format setzt sich aus 4 tetraedisch angeordneten Nierenmikrofonen zusammen. In der Literatur wird dafür häufig der Markenname *Soundfield Mikrofon* verwendet.

Abbildung 3.4: Core Sound Tetramic für A-Format Aufnahmen

Kapsel	Azimut	Elevation
A	45°	35°
B	135°	-35°
C	-45°	-35°
D	-135°	35°

Tabelle 3.2: Anordnung der Nierenkapseln bei einem A-Format Ambisonics Mikrofon.

Durch Summen und Differenzenbildung kann das A-Format in das B-Format übergeführt werden [41]:

$$W = 0.5 \cdot (A + B + C + D) \tag{3.5}$$
$$X = (A + C) - (B + D) \tag{3.6}$$
$$Y = (A + B) - (C + D) \tag{3.7}$$
$$Z = (A + D) - (B + C) \tag{3.8}$$

Die Mikrofonkapseln müssten idealer weise koinzident angeordnet sein. Dies ist in der Praxis aber nicht möglich. Gerzon beschreibt in seinem Paper[15] eine Möglichkeit die Klangverfärbungen, welche durch den räumlichen Abstand der Kapseln verursacht werden, zu minimieren.

3.2.3 Rotation - Drehung und Richtungsdominanz beeinflussen

Ein Ambisonics Signal im B-Format kann sehr einfach um seine Achsen rotiert und in der Richtungsdominanz (Zoom) verändert werden. Als Beispiel sollen hier die Vorschriften für die Rotation um die Z- und die X-Achse sowie der Zoom in die X-Richtung angeführt sein. Dabei ist ϕ der Rotationswinkel und d der Dominanzfaktor von -1 bis 1, $d = 0$ entspricht dabei keiner Veränderung der Richtungsdominanz. [41]

$$W' = W \tag{3.9a}$$
$$X' = X \cdot \cos\phi + Y \cdot \sin\phi \tag{3.9b}$$
$$Y' = Y \cdot \cos\phi - X \cdot \sin\phi \tag{3.9c}$$
$$Z' = Z \tag{3.9d}$$

Tabelle 3.3: Rotation um den Winkel ϕ bezüglich der Z-Achse

$$W' = W \tag{3.10a}$$
$$X' = X \tag{3.10b}$$
$$Y' = Y \cdot \cos\phi - Z \cdot \sin\phi \tag{3.10c}$$
$$Z' = Z \cdot \cos\phi + Y \cdot \sin\phi \tag{3.10d}$$

Tabelle 3.4: Rotation um den Winkel ϕ bezüglich der X-Achse

$$W' = W + \frac{1}{\sqrt{2}} \cdot d \cdot X \tag{3.11a}$$
$$X' = X + \sqrt{2} \cdot d \cdot W \tag{3.11b}$$
$$Y' = \sqrt{1 - d^2} \cdot Y \tag{3.11c}$$
$$Z' = \sqrt{1 - d^2} \cdot Z \tag{3.11d}$$

Tabelle 3.5: Zoom in Richtung X-Achse mit Parameter d $\{-1...1\}$

3.2.4 Dekodierung

Bei einem einfachen Ambisonics Dekoder erhält jeder Lautsprecher eine gewichtete Summe aus allen Ambisonics Kanälen. Es werden mindestens 3 Lausprecher für 2-dimensionale und 4 Lautsprecher für 3-dimensionale Wiedergabe benötigt[1]. Bessere Ergebnisse werden jedoch erzielt, wenn die Anzahl der Lautsprecher größer als das Minimum ist. Für reguläre Lautsprecheraufstellungen, zum Beispiel 8 kreisförmig angeordnete Lautsprecher mit gleichen

[1] minimale Anzahl von Laustprechern(N) bei Ordnung (M): 2D: $N = (2M + 1)$; 3D: $N = (M + 1)^2$

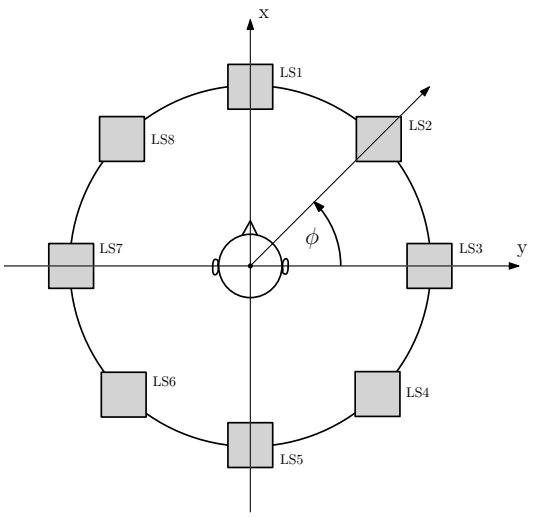

Abbildung 3.5: reguläre Anordnung von 8 Lautsprechern für 2-dimensionale Wiedergabe

Winkelabständen für 2D oder eine Anordnung anhand der fünf platonischen Körper für 3D, ergibt sich folgende Enkodierungsvorschrift:

$$p_j = \frac{1}{L}\left[W \cdot (\frac{1}{\sqrt{2}}) + X \cdot (\cos\phi_j \cos\theta_j) + Y \cdot (\sin\phi_j \cos\theta_j) + Z \cdot (\sin\phi_j)\right] \tag{3.12}$$

Dabei ist (ϕ_j, θ_j) die Position des j-ten Lautsprechers. L stellt die Anzahl der Ambisonics Kanäle dar. Für eine 2-dimensionale Wiedergabe kann der Z-Kanal vernachlässigt werden.

3.3 Higher Order Ambisonics (HOA)

Ambisonics erster Ordnung bietet nur eine beschränke Genauigkeit in der Abbildung und Reproduktion von Schallfeldern. Anfang der 1990er Jahre wurde gezeigt, dass der Ambisonics Ansatz auf höhere Ordnungen erweitert werden kann. Dies führt zu einer Verbesserung in der Richtungsabbildung und zu einer Vergrößerung des Sweetspots[2].

Eine Herleitung für den Higher Order Ambisonics Ansatz, welcher in einer Reihenentwicklung in Sphärische Harmonische (Abb. 3.6) resultiert, kann in [10], [33], [25] und [45] gefunden werden.

[2]Der Sweetspot bezeichnet einen Raumbereich, indem das Schallfeld optimal reproduziert werden kann und somit die Richtungsabbildung am besten funktioniert.

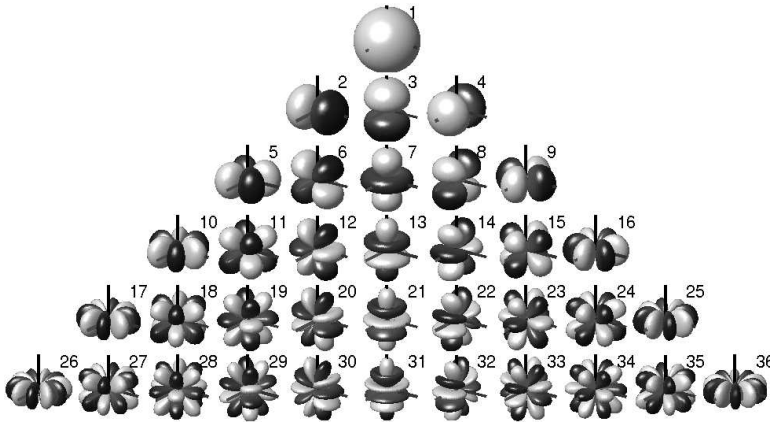

Abbildung 3.6: Sphärische Harmonische [28]

3.4 Standards bzgl. Normalisierung und Kanalreihenfolge

In der Literatur zu Ambisonics findet man viele unterschiedliche Konventionen bezüglich Normalisierung und Reihenfolge der Sphärischen Harmonischen. Sphärische Harmonische werden mit Y_l^m bezeichnet, wobei l die Ordnung und m den Index bezeichnet. Die Bezeichnung der Ordnung und Indizes variiert ebenso.

$$l \geq 0 \tag{3.13}$$

$$-l \leq m \leq l \tag{3.14}$$

In Ambisonics Systemen bis zur 3. Ordnung hat sich die Furse-Malham Kanalreihenfolge und die N3D Normalisierung durchgesetzt. Für Systeme höherer Ordnung wurde die ACN Ordnung und N3D [7] bzw. SN3D [28] vorgeschlagen.

Um die Kompatibilität bezüglich verfügbarer Ambisonics Aufnahmen bis zur 3. Ordnung und anderen Plug-ins zu gewährleisten, wurde bei der entwickelten Software die Furse-Malham Kanalreihenfolge und N3D Normalisierung als Standard gewählt. Es werden jedoch alle in der Literatur üblichen Konventionen zu Kanalreihenfolge und Normalisierung unterstützt.

3.4.1 Normalisierung

N3D (normalized, three dimensions) [6]

$$\sum_{m=-l}^{l} (Y_l^m)^2 = 2k + 1 \tag{3.15}$$

SN3D (semi-normalized, three dimensions) [6]

$$\sum_{m=-l}^{l} (Y_l^m)^2 = 1 \tag{3.16}$$

Furse-Malham Gewichtsfaktoren (weighting factors)

Die Furse-Malham Gewichtsfaktoren sind bis zur 3. Ordnung definiert und werden auf die *SN3D* Darstellung der Sphärischen Harmonischen angewendet um einen Maximalwert aller Komponenten von |1| zu gewährleisten [26].

Siehe Tabelle 3.7 für Konvertierungsfaktoren ausgehend von *N3D*.

3.4.2 Kanalreihenfolge

Tabelle 3.8 soll einen Überblick über die Kanalanordnungen bis zur 5. Ordnung geben. Die Furse-Malham Beschriftung reicht nur bis zur 3. Ordnung. In dieser Arbeit wurde die 4. Ordnung mit dem Buchstaben A beginnend ergänzt.

FuMa-Sym	Furse-Malham symbolisch
FuMa-Num	Furse-Malham numerisch
ACN	Ambisonics Channel Numbering $l^2 + l + m$
Daniel	Jerome Daniel[10] und IEM Ambi Werkzeuge

Tabelle 3.6: Begriffserklärung zu Tab. 3.8 und Tab. 3.7

Y_l^m	**FuMa Sym**	**FuMa Num**	**FuMa→N3D**	**SN3D→N3D**
Y_0^0	W	0	$\sqrt{2}$	1
Y_1^1	X	1	$\sqrt{3}$	$\sqrt{3}$
Y_1^{-1}	Y	2	$\sqrt{3}$	$\sqrt{3}$
Y_1^0	Z	3	$\sqrt{3}$	$\sqrt{3}$
Y_2^0	R	4	$\sqrt{5}$	$\sqrt{5}$
Y_2^1	S	5	$\frac{\sqrt{15}}{2}$	$\sqrt{5}$
Y_2^{-1}	T	6	$\frac{\sqrt{15}}{2}$	$\sqrt{5}$
Y_2^2	U	7	$\frac{\sqrt{15}}{2}$	$\sqrt{5}$
Y_2^{-2}	V	8	$\frac{\sqrt{15}}{2}$	$\sqrt{5}$
Y_3^0	K	9	$\sqrt{7}$	$\sqrt{7}$
Y_3^1	L	10	$\sqrt{\frac{224}{45}}$	$\sqrt{7}$
Y_3^{-1}	M	11	$\sqrt{\frac{224}{45}}$	$\sqrt{7}$
Y_3^2	N	12	$\frac{\sqrt{35}}{3}$	$\sqrt{7}$
Y_3^{-2}	O	13	$\frac{\sqrt{35}}{3}$	$\sqrt{7}$
Y_3^3	P	14	$\sqrt{\frac{35}{8}}$	$\sqrt{7}$
Y_3^{-3}	Q	15	$\sqrt{\frac{35}{8}}$	$\sqrt{7}$
Y_4^0	A[a]	16	-	3
Y_4^1	B	17	-	3
Y_4^{-1}	C	18	-	3
Y_4^2	D	19	-	3
Y_4^{-2}	E	20	-	3
Y_4^3	F	21	-	3
Y_4^{-3}	G	22	-	3
Y_4^4	H	23	-	3
Y_4^{-4}	I	24	-	3
Y_5^0		25	-	$\sqrt{11}$
Y_5^1		26	-	$\sqrt{11}$
Y_5^{-1}		27	-	$\sqrt{11}$
Y_5^2		28	-	$\sqrt{11}$
Y_5^{-2}		29	-	$\sqrt{11}$
Y_5^3		30	-	$\sqrt{11}$
Y_5^{-3}		31	-	$\sqrt{11}$
Y_5^4		32	-	$\sqrt{11}$
Y_5^{-4}		33	-	$\sqrt{11}$
Y_5^5		34	-	$\sqrt{11}$
Y_5^{-5}		35	-	$\sqrt{11}$

[a] FuMa Symbole für 4. Ordnung wurden ergänzt

Tabelle 3.7: Ambisonics Normalisierung Konvertierungstabelle [37]

Y_l^m	**FuMa-Sym**	**FuMa-Num**	**ACN**	**Daniel**
Y_0^0	W	0	0	0
Y_1^1	X	1	3	1
Y_1^{-1}	Y	2	1	2
Y_1^0	Z	3	2	3
Y_2^0	R	4	6	8
Y_2^1	S	5	7	6
Y_2^{-1}	T	6	5	7
Y_2^2	U	7	8	4
Y_2^{-2}	V	8	4	5
Y_3^0	K	9	12	15
Y_3^1	L	10	13	13
Y_3^{-1}	M	11	11	14
Y_3^2	N	12	14	11
Y_3^{-2}	O	13	10	12
Y_3^3	P	14	15	9
Y_3^{-3}	Q	15	9	10
Y_4^0	A[a]	16	20	24
Y_4^1	B	17	21	22
Y_4^{-1}	C	18	19	23
Y_4^2	D	19	22	20
Y_4^{-2}	E	20	18	21
Y_4^3	F	21	23	18
Y_4^{-3}	G	22	17	19
Y_4^4	H	23	24	16
Y_4^{-4}	I	24	16	17
Y_5^0		25	30	35
Y_5^1		26	31	33
Y_5^{-1}		27	29	34
Y_5^2		28	32	31
Y_5^{-2}		29	28	32
Y_5^3		30	33	29
Y_5^{-3}		31	27	30
Y_5^4		32	34	27
Y_5^{-4}		33	26	28
Y_5^5		34	35	25
Y_5^{-5}		35	25	26

[a] FuMa Symbole für 4. Ordnung wurden ergänzt

Tabelle 3.8: Ambisonics Kanalreihenfolge

Kapitel 4

Digitale Audioworkstation (DAW)

Bevor Computer zum Aufnehmen und Abspielen von Audiomaterial verfügbar waren, wurden diese Aufgaben von Bandmaschinen erfüllt. Die ersten Versuche Digitale Audio Workstations in den 1970er und 80er zu entwickeln wurden vor allem durch hohe Preise für Speichermedien, geringe Rechenleistung und langsamen Speicherzugriff begrenzt.

Inzwischen bieten Standardcomputer genug Speicher und Rechenleistung zum Bearbeiten und Produzieren von Mehrkanal Audio, so dass spezielle Hardwarelösungen für das Abspielen und Aufnehmen von Audiosignalen verdrängt wurden. Die Berechnung von Effekten (Filter, Hall, Delay, ...) erfolgt für die meisten Anwendungen auch durch den Hauptprozessor des Computers. Es gibt aber auch dezidierte Systeme die nur für die Effektberechnung verwendet werden und somit den Hauptprozessor der Computers entlasten.

4.1 VST Plug-ins

Im folgenden soll VST als Beispiel eines Plug-in Standards vorgestellt werden. Auch wenn im Rahmen dieser Arbeit ein Architekur übergreifender Ansatz gewählt wurde, treffen die meisten Eigenschaften für sämtliche Plug-in Standards zu.

VST steht für Virtual Studio Technology und wurde von der Firma Steinberg 1996 vorgestellt.

Ein Plug-in beschreibt eine Black Box, welche von einer Host-Software (Digitale Audio Workstation) mit einer beliebigen Anzahl von Eingangssignalen beschickt wird. Diese Signale werden von dem Plug-in verarbeitet und eine beliebige Anzahl von Ausgangssignalen kann an die DAW zurückgegeben werden. Steuerungsparameter für die Signalverarbeitung innerhalb der Black Box können der Host Software bzw. dem Benutzer zugänglich gemacht werden.

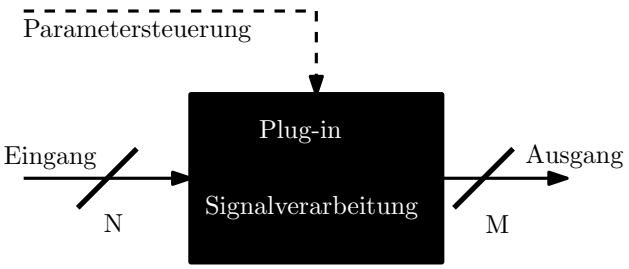

Abbildung 4.1: Plug-in als Blackbox mit N Eingangssignalen, M Ausgangssignalen und Parametersteuerung

Die Einbindung in den Signalfluss übernimmt die Host Software. VST beschreibt dabei einen Standard wie die Digitale Audioworkstation mit dem Plug-in Audio und Kontrolldaten austauscht.

Es gibt unzählige Softwareprogramme zur Aufnahme und Bearbeitung von Musik welche als VST Host fungieren können. Ich möchte hier ein paar Beispiele nennen:

- Steinberg Cubase, Wavelab (Windows, Mac OS)
- Apple Logic (Mac OS)
- Digidesign ProTools (Windows, Mac OS)
- Adobe Audition (Windows, Mac OS)
- Reaper (Windows, Mac OS)
- Ardour (Open Source; Linux, Mac OS)
- Audacity (Open Source; Linux, Mac OS, Windows)
- VSTHost (Freeware, reiner VST Host; Windows)

In den Spezifizierungen zur VST Architektur gibt es drei verschiedene Arten von VST Plug-ins:

- VST Instrumente

  generieren Audiosignale. Meist werden sie durch MIDI[1] Noten gesteuert. z.B. Synthesizer, Sampler,...
- VST Effekte

  bekommen Audiosignale von der Host Software, verändern diese und geben sie am Ausgang wieder zurück. z.B.: Hall, Delay, Phaser,...
- VST MIDI Effekte

[1]Musical Instrument Digital Interface (MIDI) ist ein Standard zum Austausch von Steuerdaten zwischen elektronischen Musikinstrumenten.

bekommen MIDI Daten von der Host Software, bearbeiten diese und geben sie am Ausgang wieder aus. z.B.: transponieren, Arpeggiator,...

Mit dem erscheinen der Digitalen Audio Workstation Cubase Version 3.02 im Jahr 1996 wurde der VST Standard zum ersten mal implementiert.

Seit dem haben sich neben Steinbergs' VST System auch andere Plug-in Formate entwickelt:

- AU - Audio Unit von Apple für ihr CoreAudio System, nur MacOS
- DirectX - Microsoft, nur auf Windows Basis
- LADSPA - Linux Audio Developers Simple Plugin API
- LV2 - (LADSPA version 2) Nachfolger von LADSPA
- RTAS - (Real-Time AudioSuite) von Digidesign für ihr ProTools System
- AAX - Nachfolger von RTAS als ProTools Plug-in Format

Auch wenn VST Version 2 inzwischen nicht mehr das vielseitigste Plug-in Format darstellt, ist es im Gegensatz zu anderen Standards mit den meisten Audioprogrammen auf verschiedenen Betriebssystemen (Windows, Mac OS, Linux) kompatibel und somit am universellsten einsetzbar. Einzig die proprietäre Lizenz macht es im Umgang mit Open Source Software problematischer, weshalb man die Software Ardour zum Beispiel für VST Support selber kompilieren muss.

4.1.1 Implementierung eines VST-Plug-in

Das Steinberg VST SDK (Software Developement Kit) ist eine Sammlung von C++ Klassen, basierend auf einer C API (Application Programming Interface).

Der Source Code von einem VST-Plug-in ist Plattform unabhängig, so lange in der Signalverarbeitung bzw. der Benutzeroberfläche keine plattformabhängigen Programmteile implementiert werden.
Das Plug-in muss jedoch für jede Plattform extra kompiliert werden. Unter Windows ist ein VST Plug-in eine multi-threaded Dynamic Link Library (Dateiendung .dll), normalerweise in dem Standardordner *Programme\Steinberg\VstPlugins*.
Unter MacOS ist ein VST Plug-in ein Mach-O Bundle (Dateiendung .dylib) meist im Ordner */Library/Audio/Plug-Ins/VST*. Unter UNIX Systemen (z.B. Linux) eine shared Library (Dateiendung .so). [35]

Ein VST Plug-in muss zumindest aus einem Signalverarbeitungsteil bestehen. Mit Hilfe von Benutzer Parametern kann der implementierte Algorithmus modifiziert werden. Um dem Benutzer bzw. der Host Software die Möglichkeit zu geben diese User Paramter zu beeinflussen, kann eine Grafische Benutzeroberfläche (GUI)[2] implementiert werden. Eine Möglich diese zu entwickeln wird mit dem Open Source Projekt VSTGUI[1] zur Verfügung gestellt.

Die Implementation einer eigenen GUI ist aber nicht zwingend notwendig. Falls keine GUI für das VST Plug-in vorhanden ist, wird das Plug-in in einer Standardoberfläche, abhängig vom Host Programm, dargestellt.

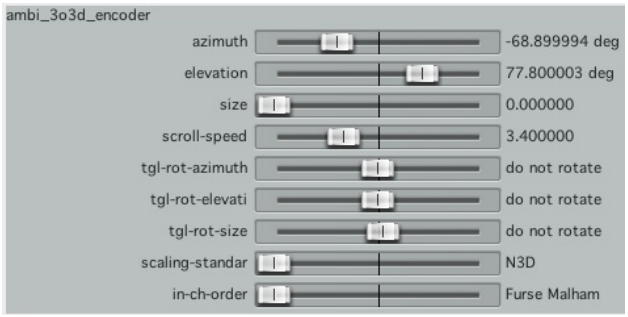

Abbildung 4.2: Plug-in Standardoberfläche der Software Reaper

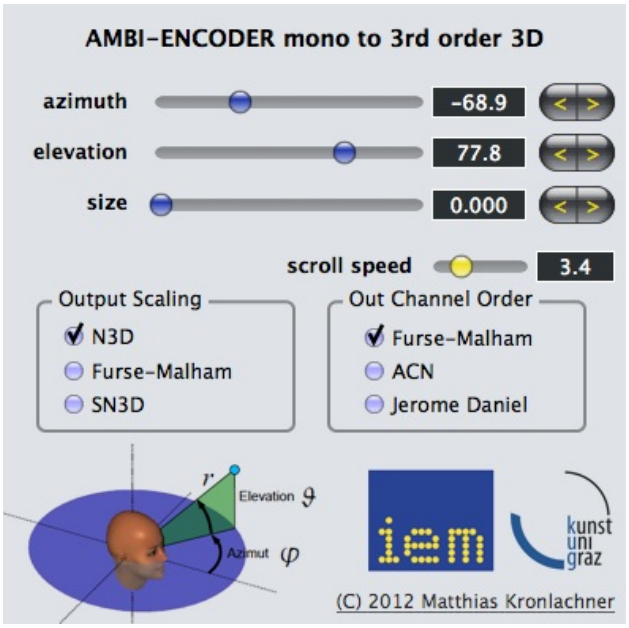

Abbildung 4.3: Plug-in eigene Benutzeroberfläche

[2]Graphical Userinterface (GUI)

Will man ein VST Plugin implementieren muss man von der Basisklasse AEffect (VST 1.0 Standard) oder AEffectX (VST 2.0 Standard) ableiten. Die wichtigste und obligate Funktion ist *processReplacing()* welche mit 32 bit (single precision) Gleitkommadarstellung (floating-point) Audiosamples arbeitet. Optional kann auch die Funktion *processDoubleReplacing()*, welche mit 64 bit (double precision) Gleitkommazahl Samples arbeitet, implementiert werden.

Beide Funktionen erhalten die Eingangssignale in Blöcken, die Blockgröße wird dabei von der Hostsoftware gesteuert, und schreiben in den Ausgangsbuffer. Audiosamples können im Bereich von -1.0 bis 1.0 liegen. (1.0 entspricht dabei 0dBFS[3], 0.5 -6dBFS und 0.0 -∞dBFS)

Die Benutzer Parameter werden als 32 bit Gleitkomma Zahlen im Bereich von 0.0 bis 1.0 verarbeitet. Der Bereich ist fixiert, egal für welcher Wert dieser im Algorithmus oder in der GUI steht. Der Wertebereich muss intern in den gewünschten Bereich transformiert werden.

Der Wert, das Label und die Einheit von User Parametern können als ASCII String ausgegeben werden. [35] Damit ist es möglich, die Intern von 0.0 bis 1.0 reichenden Parameter in jeden Wertebereich, oder auch als Text, auszugeben.

VST Versionen

Zur Zeit existiert schon Version 3.5 des VST Standards. Er bringt einige Verbesserungen im Vergleich zur Version 2 wie zum Beispiel:

- Silence Flag
  Um Rechenleistung zu sparen, können VST3 Plug-ins dynamisch ein und ausgeschaltet werden, je nachdem ob ein Signal anliegt oder nicht.
- Dynamische Ein/Ausgänge
  Ein VST3 Plug-in kann seine Ein/Ausgangskonfiguration dynamisch verändern. Dieses Feature ist für Ambisonics interessant. Ein Plug-in kann verschiedenen Ordnungen implementieren und je nach Ordnung die Ein- und Ausgänge variieren. Im VST2 Standard muss man für jede Ordnung eigene Plug-ins zu Verfügung stellen.
- Separation von User Interface und Recheneinheit
  Die benötigte Rechenleistung eines Plug-ins kann durch die Separation von User Interface und Recheneinheit einfacher auf einen anderen

[3]Dezibel Full Scale (dBFS) beschreibt die logarithmische Dynamikskala in digitalen Systemen. 0 dBFS entspricht einer Vollaussteuerung der Amplitude.

Prozessor/Computer ausgelagert werden.

Dadurch, dass VST3 nicht abwärts kompatibel ist und noch nicht von allen Host Programmen unterstützt wird, sind die im Rahmen dieser Arbeit entstandenen Plug-ins im VST2.4 Standard.

4.1.2 Entwicklungsmethoden für Audio Plug-ins

Entwicklungsumgebungen für VST Plug-ins

Unter Mac OS X bietet sich die Apple eigene Entwicklungsumgebung (IDE - Integrated Developement Environment) *Xcode* an. Sie ist frei verfügbar und vereint unter anderem verschiedene Frameworks, Bibliotheken, Compiler, Linker und Versionsverwaltungen.

Auf der Homepage von Teragon Audio [36] findet sich eine Dokumentation über die Einrichtung von Xcode für die Entwicklung eines VST Plug-ins.

Unter Windows bietet sich die Entwicklungsumgebung Visual Studio von Microsoft an. Die im Umfang für die VST Entwicklung ausreichende *Visual C++ Express-Edition* ist kostenlos von Microsoft zu beziehen. Teragon Audio[36] bietet auch für die Entwicklung unter Windows einen detailierten Leitfaden auf ihrer Homepage an.

In jedem Fall wird zum Kompilieren des Plug-ins das VST-Software Developement Kit von Steinberg benötigt. Es muss dabei aus lizenzrechtlichen Gründen direkt von Steinberg bezogen werden. Eine Anmeldung als Entwickler auf der Homepage von Steinberg[35] ist dazu notwendig. Es fallen jedoch keine Kosten an.

Durch die Vielfalt an Plug-in Formaten wurden allgemeingültigere Open Source Lösungen entwickelt, welche das kompilieren in verschiedene Formate ermöglicht.

Diese bieten den Vorteil, dass man den Code nur einmal schreiben muss und damit eine breitere Palette an Plattformen abdecken kann.

FAUST

FAUST (Functional AUdio STream) ist eine Programmiersprache um effizient C++ Audio Plug-ins bzw. selbstständige Programme zu entwickeln. FAUST Code beschreibt Block Diagramme bzw. Signalflussgraphen. Diese werden vor dem Kompilieren in ihrer mathematischen Bedeutung analysiert und erst dann in C++ Code umgewandelt. Dieses Verfahren führt zu sehr effizientem C++ Code. Als Zielarchitektur kann unter vielen anderen VST und L2V Plug-ins gewählt werden. [17]

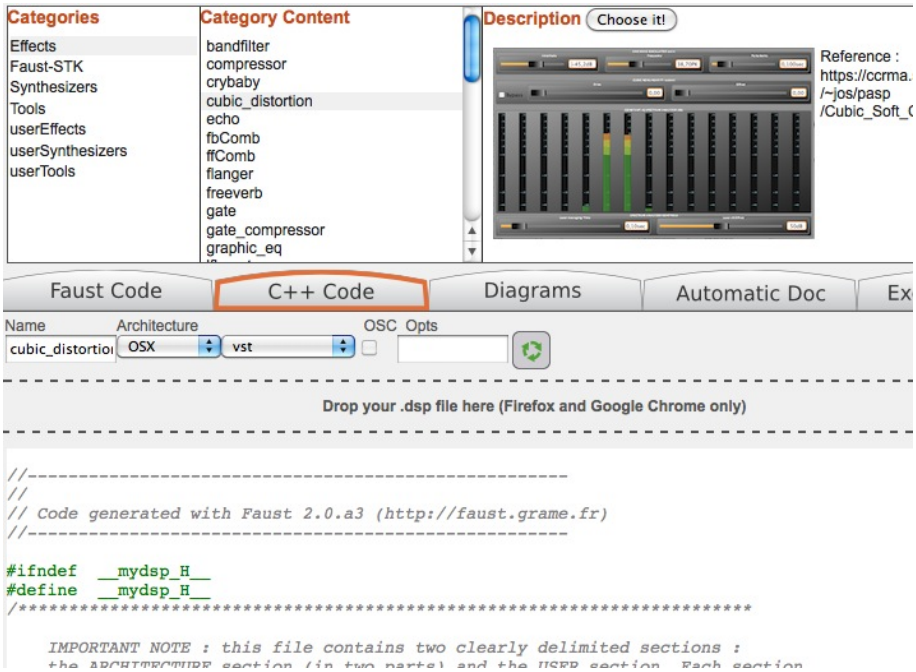

Abbildung 4.4: FAUST Online Compiler

4.2 Audio Plug-in Entwicklung mit JUCE

Im Laufe der ersten Prototypen für diese Arbeit wurde die primäre Entwicklung für die VST Plattform verworfen und auf Grund der Portabilität und der einfach zu gestaltenden Benutzeroberflächen auf Juce umgestellt.

JUCE (Jules' Utility Class Extensions) ist eine C++ Programmbibliothek zum plattformübergreifenden Entwickeln. Es unterstützt unter anderem die Erstellung von Grafischen Benutzeroberflächen und eignet sich auch für

Audioanwendungen.

Es unterstützt Windows, Mac OS X, Linux, iOS und Android. JUCE ist auch für die Audio Plug-in Entwicklung geeignet. In JUCE entwickelte Plug-ins können als VST, AU, RTAS, AAX, LADSPA und LV2 Plug-in mit GUI kompiliert werden. Die Unterstützung von LV2 Plug-ins ist derzeit noch nicht direkt im JUCE Projekt integriert und kann in dem separaten Projekt Distroh[8] gefunden werden.

Juce ist Open Source und darf für Open Source Projekte kostenfrei verwendet werden(GNU GPL Lizenz)[4]. Für eine kommerzielle Nutzung muss eine Lizenz erworben werden.

Einige namhafte Audio Plug-in Hersteller verwenden JUCE, was eine gewisse Absicherung bzgl. zukünftiger Weiterentwicklungen bedeuten kann.

Im Juce Paket ist eine Sammlung umfangreicher Beispielprojekten wie einem Audio Plug-in, einem Audio Plug-in Host, einem Browser Plug-in und einem selbständigen Programm mit Benutzeroberfläche für Audio, Video inkl. OpenGL Unterstützung, enthalten.

Weiters werden Werkzeuge mitgeliefert, welche die Erstellung Projekten erheblich vereinfachen.

4.2.1 Introjucer

Mit Hilfe des Werkzeugs *Introjucer* kann ein Juce Projekt für verschiedene Entwicklungsumgebungen wie Microsoft Visual Studios (Windows), Apple XCode (Mac OSX) oder Linux GNU Makefile eingerichtet werden. Es wird dabei das Grundgerüst des C++ Codes und eine Compiler Konfigurationen erstellt. Zum erfolgreichen Kompilieren eines VST Plug-ins wird das VST SDK von Steinberg benötigt [Kap. 4.1.2].

[4]General Public License (GPL) bedeutet, dass alle Entwicklungen, die Teile einer Software mit GPL verwenden, selber auch unter der GPL weitergegeben werden müssen (Copyleft).

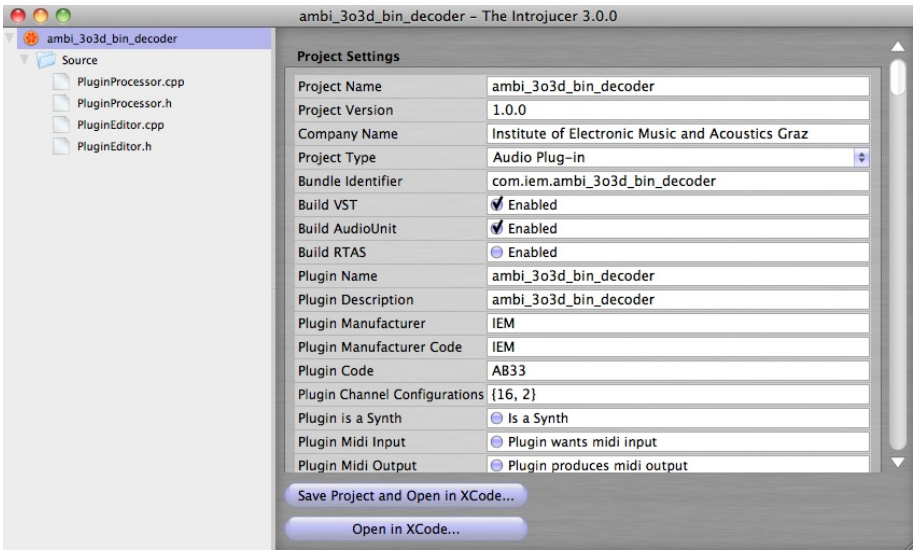

Abbildung 4.5: Introjucer hilft beim Einrichten des Grundgerüstes

Introjucer erstellt zwei .cpp Dateien. `PluginEditor.cpp` enthält die Ableitung der `AudioProcessorEditor` Klasse. Diese kann für die Erstellung einer grafischen Benutzeroberfläche verwendet werden. `PluginProcessor.cpp` enthält eine Ableitung der `AudioProcessor` Klasse und implementiert den DSP Teil des Plug-ins sowie die Verwaltung der Benutzerparameter.

4.2.2 Jucer

Für die einfache Erstellung einer Benutzeroberfläche ist in Juce das Werkzeug *Jucer* enthalten. Mit ihm lässt sich die Datei `PluginEditor.cpp` editieren oder erstellen. Es bietet einen WYSIWYG[5] Editor zum Erstellen von Fenstern mit verschiedensten grafischen Elementen und Bedienoberflächen. Dabei generiert es C++ Code der in ein Juce Projekt eingebunden werden kann. Grafiken werden dabei ebenfalls in den Code eingebunden und brauchen somit keine externen Verlinkungen.

[5]What You See Is What You Get

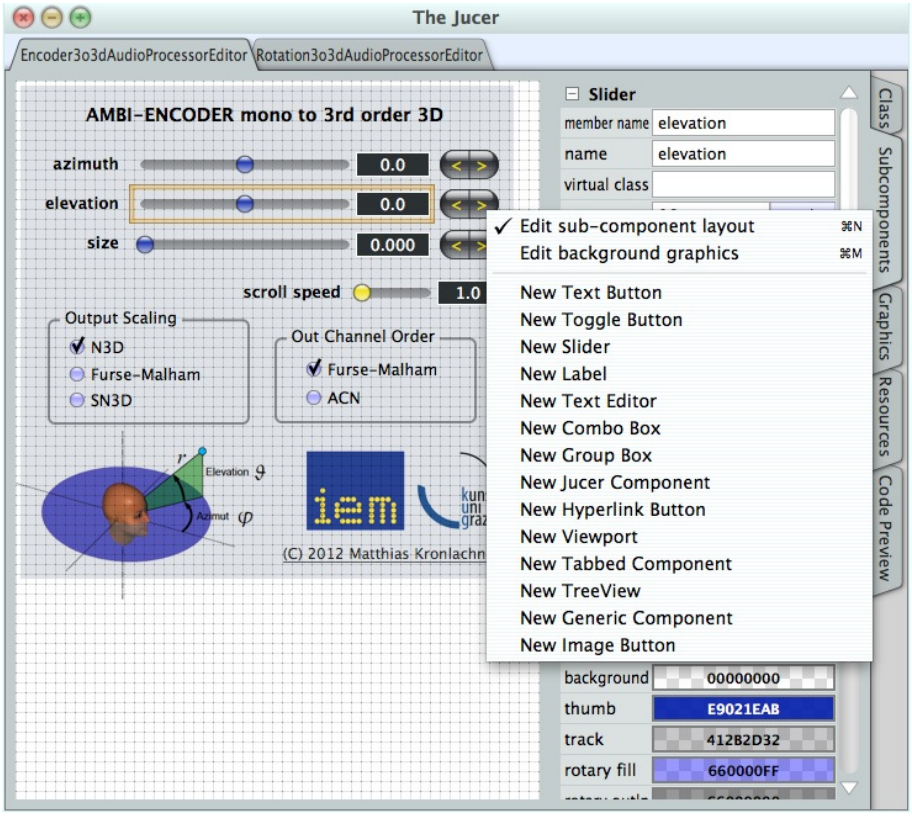

Abbildung 4.6: Jucer zum Entwerfen einer grafischen Benutzeroberfläche

4.2.3 Wichtige Funktionen in Juce

Die Hauptfunktion der Klasse `AudioProcessor` ist der DSP Block `processBlock`. Dieser Funktion wird ein 32 bit Gleitkommazahl Audio Buffer übergeben, welcher sämtliche Eingangskanäle beinhaltet. Derzeit sind für Audio Buffer keine 64 bit Gleitkommazahlen (double precision) vorgesehen, diese Funktion kann aber bei Bedarf für VST Plug-ins hinzugefügt werden.

Zu Beachten ist, dass kein getrennter Eingangs- und Ausgangsbuffer an die Funktion übergeben wird. Es gibt nur einen Buffer, aus dem das Eingangssignal gelesen, und die Ausgabe zurückgeschrieben wird. Falls das Plug-in eine ungleiche Anzahl an Ein- und Ausgängen besitzt, werden die nicht verwendeten Kanäle verworfen.

Folgendes Beispiel für die `processBlock` Funktion skaliert alle Eingangssignale um den Faktor `gain`.

```
1 void TestjuceAudioProcessor::processBlock (AudioSampleBuffer& buffer,
      MidiBuffer& midiMessages)
2 {
3     int numSamples = buffer.getNumSamples();
4     int numChannels = buffer.getNumChannels();
5
6     for (int channel = 0; channel < numChannels; ++channel) // iterate
      over channels
7     {
8       float* channelData = buffer.getSampleData (channel); // pointer to
       array of samples
9
10      for (int i = 0; i < numSamples; ++i) // iterate over samples
11      {
12        (*channelData++) *= gain;   // scale every sample
13      }
14    }
15 }
```

Listing 4.1: Skaliere alle Eingangssignale um den Faktor gain

Eine praktische Eigenschaft ist, dass sämtliche Parameter eines Plug-ins von der Host Software verwaltet werden können. Diese werden mit einem Projekt gespeichert. Die Host Software kann auch Presets für Plug-ins speichern und verwalten. Dazu muss der Programmierer allerdings sicherstellen, dass alle Parameter über die dafür vorgesehenen Funktionen zugänglich und definiert sind.

```
1 enum Parameters
2 {
3   gainParam = 0,
4   totalNumParams
5 };
6
7 void TestjuceAudioProcessor::setParameter (int index, float newValue)
8 {
9   switch (index)
10  {
11    case gainParam:     gain = newValue;  break;
12    default:            break;
13  }
14 }
15
16 float TestjuceAudioProcessor::getParameter (int index)
17 {
18   switch (index)
19   {
20     case gainParam:     return gain;
21     default:            return 0.0f;
22   }
23 }
24
```

```
25 const String TestjuceAudioProcessor::getParameterName (int index)
26 {
27   switch (index)
28   {
29     case gainParam:     return "Gain";
30     default:            break;
31   }
32   return String::empty;
33 }
34
35 const String TestjuceAudioProcessor::getParameterText (int index)
36 {
37   String text;
38   switch (index)
39   {
40     case gainParam:
41     {
42       float val = 20.f*log10(gain); // convert factor to dB
43       text.formatted("%f dB", val); // display eg. 0.5 as -6 dB
44       break;
45     }
46     default:            break;
47   }
48   return text;
49 }
```

Listing 4.2: Verwaltung der Benutzer Parameter in Juce

Bei Verwendung einer eigenen grafischen Oberfläche muss zudem sichergestellt werden, dass die Regler regelmäßig an den internen Status der Parameter angepasst werden, um mögliche Automatisierungen seitens der Host Software zu erfassen. Dies kann durch eine Timer Funktion realisiert werden.

Kapitel 5

Binaural Dekoder Implementation

5.1 Allgemeines

Der Dekoder berechnet aus den Ambisonics Signalen die für eine Resynthese notwendigen Lautsprechersignale einer spezifischen Lautsprecheranordnung. Die Position eines Lautsprechers kann durch Faltung mit einem Set von HRIRs [vgl. 2.2] über Kopfhörerwiedergabe simuliert werden. Für einen Binauraldekoder werden dafür bei einer Anzahl von N Lautsprechern $2N$ Faltungen benötigt.

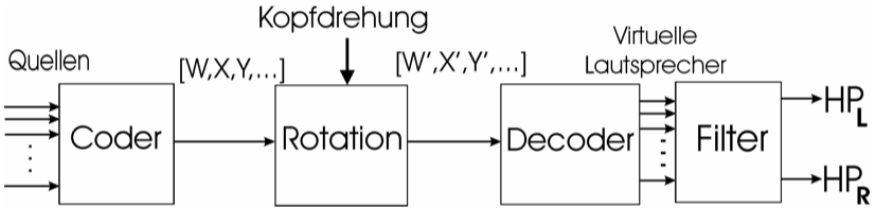

Abbildung 5.1: Signalflussdiagram eines Binauralen Ambisonics Wiedergabesystems [33]

Für die binaurale Simulation von Schallquellen könnte auch ein direkter Ansatz gewählt werden[25], indem das Quellsignal mit dem positionsabhängigen Paar von Impulsantworten gefaltet wird. Durch die nur in diskreten Abständen verfügbaren HRIRs wäre aber eine Interpolation zwischen HRIRs

sowie die Implementation eines zeitvarianten Filters notwendig. Bei einem Ambisonics Lösungsansatz bleiben durch die fixe Position der Lautsprecher die Filter invariant. Die Drehung des Schallfeldes sowie die Bewegung von Schallquellen erfolgt in der Ambisonics Domäne.

Für diese Arbeit wurden die diffusfeld entzerrten[1] MIT Kemar HRIRs verwendet. Aristoteles Digenis hat die C Bibliothek `mit_hrtf_lib`[11] geschrieben, welche die Einbindung der MIT HRIRs in Programme erheblich vereinfachen. Leider stehen die Impulsantworten in dieser Bibliothek nur in der Länge von 128 Samples zur Verfügung. Laut [25] bewirkt die Verwendung von HRIRs mit weniger als 256 Samples eine Glättung der Frequenzantwort. Dadurch können wichtige Details zur Lokalisation verloren gehen.

Sontacchi[33] beschreibt die Herleitung eines reduzierten HRIR Satzes. Durch die Anwendung von Symmetriebedingungen der Kopfbezogenen Impulsantworten sowie der Ambisonics Dekodermatrix bei symmetrischer Lautsprecheranordnung kann die Anzahl der Faltungsoperationen reduziert werden. Dieser Ansatz ist in dem Pure Data External `bin_ambi_reduced_decode_fir`, Teil der Bibliothek `iem_bin_ambi`, implementiert.

Dabei wird die Anzahl der Faltungsoperationen unabhängig von der Anzahl an Lautsprechern. Es muss nur noch jeder Ambisonicskanal mit einer abgeleiteten Impulsantwort gefaltet werden. Das Linke und Rechte Ohrsignal setzt sich aus einer Linearkombination der gefalteten Ambisonicskanälen zusammen [Abb. 5.2].

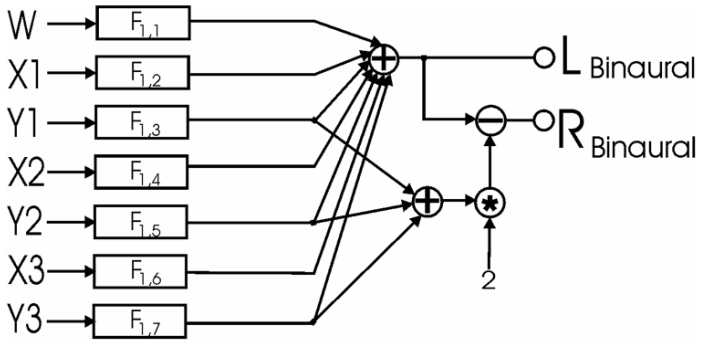

Abbildung 5.2: Reduziertes Set von HRIRs [33]

[1] Die Einflüsse der nicht idealen Frequenzantwort von Mikrofon und Lautsprecher werden damit kompensiert.

5.2 Implementierung

Zur Zeit der Erstellung dieser Arbeit waren keine Ambisonics Enkoder und Rotatoren für 3. Ordnung 3D als plattformunabhängige Audio Plug-ins erhältlich. Deshalb mussten diese erst adaptiert werden. Für Linux ist eine umfangreiche LADSPA Plug-in Sammlung von Fons Adriaensen and Joern Nettingsmeier[3] erhältlich. Diese wurden teilweise als Vorlage für die im Rahmen der Arbeit entwickelten plattformunabhängigen Audio Plug-ins genommen.

5.2.1 Ambisonics Enkoder

Der Signalverarbeitungsteil für den 3. Ordnung 3D Enkoder wurde teilweise von den Open Source AMB-plugins[3] für Linux übernommen. Zusätzlich wurde ein Size Parameter implementiert. Dieser bewirkt eine Skalierung der höheren Ambisonics Ordnungen ($M > 1$) und führt bei Werten $size > 0$ zu einer Ausweitung der Quellenbreite. Das Plug-in hat einen Mono Signaleingang und gibt 16 Ambisonicskanäle aus.

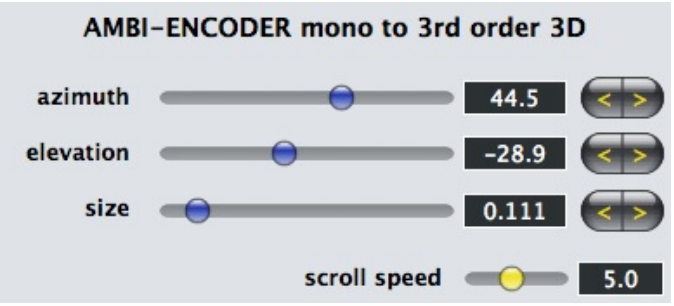

Abbildung 5.3: Ambisonics Enkoder 3. Ordnung 3D mit size Parameter

5.2.2 Ambisonics Rotator

Für die optimale Kompensation von Kopfbewegungen des Hörers ist ein Ambisonics Rotator nötig, der eine Rotation in allen drei Achsen ermöglicht. Es konnte kein Audio Plug-in gefunden werden, welches diese Möglichkeit bis zur 3. Ordnung 3D ermöglicht. Es wurden daher die Rotationsmatrizen von Zmölnig[45] übernommen und in einem Audio Plug-in realisiert.

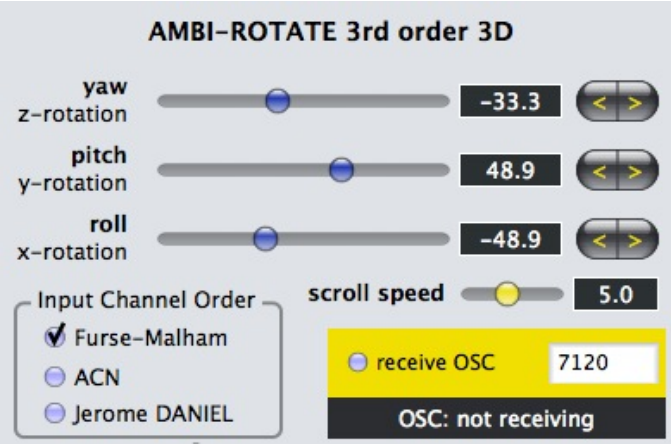

Abbildung 5.4: Ambisonics Rotator 3. Ordnung 3D mit OSC

OSC Einbindung

Um eine optimale Einbindung des Headtrackings zu ermöglichen, wurde der Rotator mit OSC Funktionalität ausgestattet. Dafür wurde die Bibliothek `liblo`[19] eingebunden. Der Empfang von OSC Nachrichten sowie der Port kann über die grafische Benutzeroberfläche eigestellt werden. Der Syntax für die OSC Nachricht richtet sich dabei nach dem Programm `head_pose_estimation_demo` (siehe Kap. 6.2.2).

Grundsätzlich sollte bei einem Plug-in die Host Software nicht bei der Kontrolle von Parametern übergangen werden. Die Implementation von OSC im Plug-in selber ist daher als etwas problematisch anzusehen. Auf Grund des Umstands, dass keine DAW außer *Reaper* zur Zeit Unterstützung für OSC anbietet, wurde diese Steuerungsmöglichkeit inkludiert. Des weiteren sollten diese Steuerdaten für die Kopfdrehung nicht von der DAW aufgezeichnet werden sondern in Echtzeit von dem Headtracking System kommen.

```
1 String osc_in_port = "7120"; // set in port
2 if ((st = lo_server_thread_new(osc_in_port.toUTF8(), error))) // start
      thread
3 {
4   lo_server_thread_add_method(st, "/head_pose", "fffffff",
      headpose_handler, this); // add callback
5   lo_server_thread_start(st);  // start server
6 }
7
8 int headpose_handler(const char *path, const char *types, lo_arg **argv,
       int argc,
9                       void *data, void *user_data) // head_pose [User_ID]
       [x] [y] [z] [pitch] [yaw] [roll]
```

```
10 {
11   Rotation3o3dAudioProcessor *me = (Rotation3o3dAudioProcessor*)
       user_data;
12
13   // internal angles are between 0.0 and 1.0
14   // headtracking sends -180 to 180 deg, so invert and scale them
15   me->setParameterNotifyingHost(Rotation3o3dAudioProcessor::rotxParam,
       ((argv[6]->f)*(-1.f))/360.f+0.5f);
16   me->setParameterNotifyingHost(Rotation3o3dAudioProcessor::rotyParam,
       ((argv[4]->f)*(-1.f))/360.f+0.5f);
17   me->setParameterNotifyingHost(Rotation3o3dAudioProcessor::rotzParam,
       ((argv[5]->f)*(-1.f))/360.f+0.5f);
18
19   return 0;
20 }
```

Listing 5.1: Empfang von OSC Nachrichten mit liblo

5.2.3 Ambisonics Binaural Dekoder

Der Binauraldekoder `ambi_3o3d_bin_decoder` basiert auf einer Dekodermatrix von Fons Adriaensen und seinem Ambisonics Dekoder AmbDec[2], welcher für Linux und Mac OS X erhältlich ist und mittels Jack[2] in ein Audiosystem eingebunden werden kann. Die Matrix wurde aus dem Preset `icosahedron-3h3v.ambdec` übernommen. Es kodiert Ambisonics 3. Ordnung 3D auf 20 Lautsprecher in der Anordnung eines Ikosaeders.

Als CPU sparende Variante wurde der Binauraldekoder `ambi_3o3d_bin_decoder_red` mit reduziertem HRIR Set entwickelt. Dieser basiert auf den Pure Data Externals `bin_ambi_reduced_decode_fir` von Thomas Musil (siehe Kap. 5.1) und dem daraus abgeleiteten HRIR Set. Die Anordnung der 32 virtuellen Lautsprecher entspricht einem abgestumpften Ikosaeder (siehe [27]).

[2]Jack ist ein System zur Verwaltung von Echtzeit Audio und Midi und bietet flexible Routingmöglichkeit zwischen Programmen und physikalischen Ein/Ausgängen.

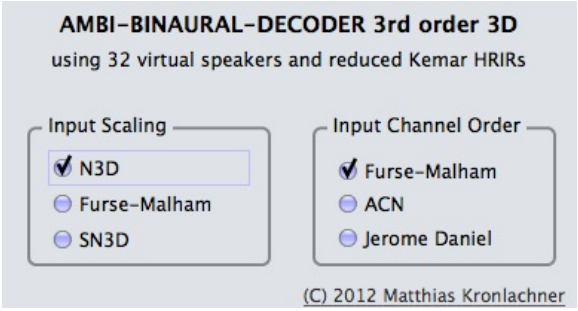

Abbildung 5.5: Bedienoberfläche für den Ambisonics Binaural Dekoder

Faltung

Die notwendigen Faltungsoperationen werden bei den Plug-ins in dieser Arbeit in der Zeitdomäne durchgeführt. Hier wäre eine Optimierung hinsichtlich einer Operation im Frequenzbereich möglich.

```
1  float* conv::process(float* input)
2  {
3    output_buffer[0] = 0.0f; // zero output
4    output_buffer[1] = 0.0f;
5    input_buffer[pos] = *input; // write inputsample into ringbuffer
6
7    float* hrir_l = pLeft_f; // pointer to ir
8    float* hrir_r = pRight_f;
9
10   int cur_pos=pos;
11   for (int i=0; i < conv_length; i++)
12   {
13     output_buffer[0] += input_buffer[cur_pos] * (*hrir_l++); // compute
       convolution sum
14     output_buffer[1] += input_buffer[cur_pos] * (*hrir_r++);
15
16     if (--cur_pos < 0)
17     {
18       cur_pos=conv_length-1;  // ringbuffer -> wrap index for ir
19     }
20   }
21   if (++pos >= conv_length)
22   {
23     pos=0; // ringbuffer - wrap index for inputsample
24   }
25   return output_buffer; // return convolved stereo signal
26 }
```

Listing 5.2: Die Faltungsfunktion erzeugt aus einem (mono) Lautsprechersignal ein Binaurales (stereo) Signal

5.3 Bedienelemente

5.3.1 Kontinuierliche Kreisbewegungen

Das Problem einer Darstellung in Polarkoordinaten ist der Sprung von $-180°$ auf $180°$. Was in der Automationskurve des Sequenzers wie eine Sprunghafte Bewegung aussieht[Abb. 5.6], ist in Wirklichkeit eine kontinuierliche Kreisbewegung.

In dieser Arbeit wurde versucht, das Problem der Unstetigkeit in der Winkeldarstellung durch zusätzliche Funktionalität auszugleichen.

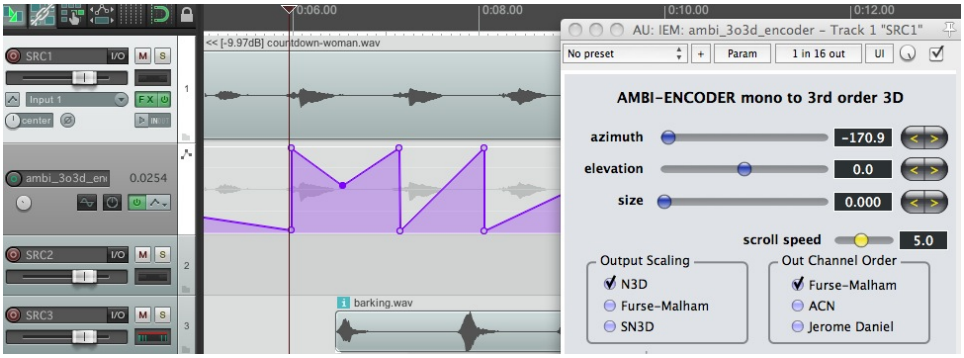

Abbildung 5.6: Automationskurve der Rotation - Sprung zwischen $-180°$ und $+180°$

Durch die Einbindung von Toggle Schaltern für eine automatische positive bzw. negative Drehung kann eine kontinuierliche Bewegung ohne Rücksicht auf den Sprung in der Winkeldarstellung realisiert werden. Diese Schalter lassen sich genauso Automatisieren wie die Geschwindigkeit der Kreisbewegung [Abb. 5.8].

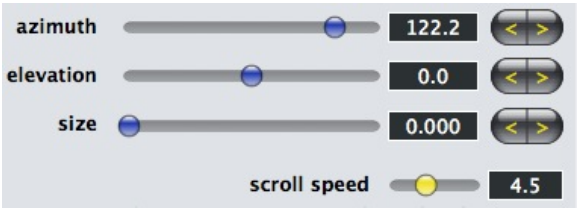

Abbildung 5.7: Schalter für kontinuierliche Bewegungen

5.3.2 Skalierung und Kanalordnung

Auf Grund der unterschiedlichen existierenden Normierungsfaktoren der Sphärischen Harmonischen (siehe Kap. 3.4) sowie deren unterschiedliche Reihenfolge wurden alle Plug-ins mit Konvertierungsfunktionen ausgestattet. Somit können mit dem Plug-in Set verschiedene Kombinationen bzgl. Normierungen und Kanalreihenfolge bedient werden. Als Standardkonfiguration wurde die N3D Skalierung sowie die Furse-Malham Kanalordnung gewählt.

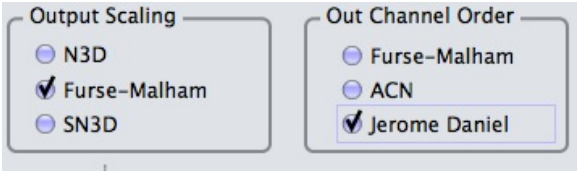

Abbildung 5.8: Schaltflächen zur Auswahl der Skalierung und Kanalordnung

5.4 Testumgebung

Für die Produktion von Ambisonics Inhalten ist ein flexibles Routing seitens der Digitalen Audioworkstation nötig. Viele kommerzielle Programme unterstützen nur die Standard Lautsprecherkonfigurationen wie Stereo, Quadrophonie, 5.1-7.1 Surround. Für Ambisonics sind aber im Falle von 2D $(2N+1)$[3] bzw. im 3D Fall $(N+1)^2$ Kanäle notwendig. Die Software Reaper (Windows und Mac OS X) ist mit sehr flexiblen Routingmöglichkeiten ausgestattet. Jede Spur (Kanalzug) kann aus bis zu 64 Kanäle bestehen, was einer theoretische Grenze von 7. Ordnung in 3D (64 Kanäle) bzw. 31. Ordnung 2D (63 Kanäle) entspricht. Diese Grenze wird praktisch durch die Rechenleistung begrenzt. Eine uneingeschränkte Testversion von Reaper kann auf der Homepage[4] heruntergeladen werden.

Als Open-Source Alternative kann Ardour[5] genannt werden, welches für Linux und Mac OS X angeboten wird. Eine Ausführliche Anleitung über die Verwendung von Ambisonics in Ardour wurde von Jörn Nettingsmeier verfasst[29].

[3] N bezeichnet dabei die Ambisonics Ordnung
[4] Reaper - Digital Audio Workstation: http://www.reaper.fm
[5] Ardour - Digital Audio Workstation: http://www.ardour.org

5.4.1 Ambisonics in Reaper

Für eine detaillierte Beschreibung zur Konfiguration von Reaper für Ambisonics soll hier auf die Video Tutorials von Bruce Wiggins[42] verwiesen werden. Diese beschreiben zwar nur die Konfiguration eines Ambisonics Systems 1. Ordnung, doch kann diese sehr einfach auf höhere Ordnungen erweitert werden.

In der Testumgebung für die erstellten Binaural Dekoder und den Rotator wurden mehrere Wiedergabespuren mit jeweils 16 Kanälen angelegt. Auf diese Spuren wurden Mono Klangbeispiele gelegt. Jede Spur nutzt den Ambisonics 3. Ordnung 3D Enkoder, welcher im Rahmen der Arbeit entwickelt wurde, als Insert-Effekt (Panning Plug-in). Die Mono Signale wurden mit dem Enkoder räumlich verteilt und die resultierenden 16 Ambisonicssignale pro Spur an den Ambisonics Masterbus geroutet und summiert.

Der Ambisonics Masterbus erhält als Insert-Effekt den 3. Ordnung 3D Rotator und einen 3. Ordnung 3D Binauraldekoder.

Der Rotator wird durch die Kopfdrehungen des Hörers, welche von dem Kinect Sensor und dem `head_pose_estimation_demo` [Kap. 6.2.2] Programm detektiert werden, gesteuert.

Der Stereo Ausgang des Ambisonics Master Bus entspricht der Kopfhörersimulation einer in der Realität nur sehr aufwendig zu realisierenden Abhörsituation.

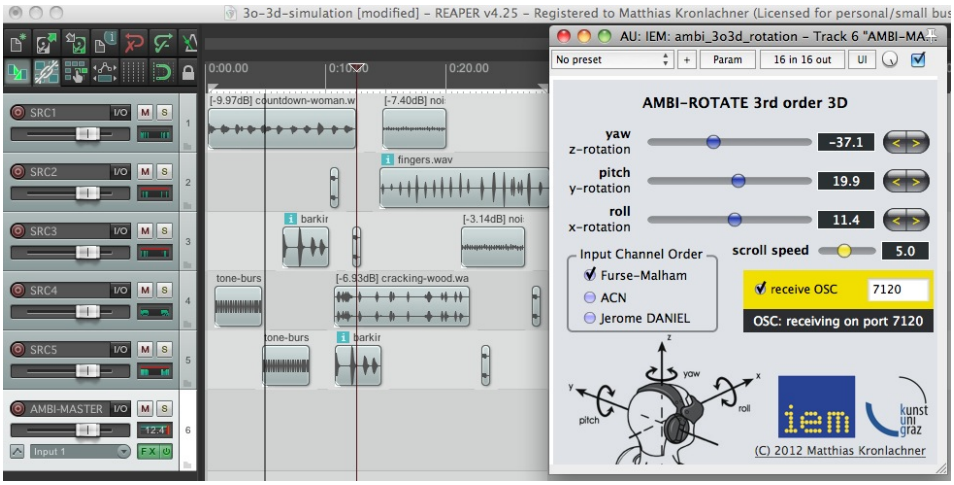

Abbildung 5.9: Reaper Testumgebung

Kapitel 6

Headtracking

6.1 Allgemeines

Seit den 60er Jahren wird an Systemen für virtuelle Realitäten geforscht, das Militär und die Raumfahrt waren dabei der primäre Motor für frühe Entwicklungen. Inzwischen sind Virtual Reality Systeme aus Bereichen wie Piloten-Training, Medizin, Architektur, Computerspiele kaum mehr wegzudenken. Die Anfänge der Virtuellen Realität liegen aber schon viel früher wie zum Beispiel die Zentralperspektive im 15. Jahrhundert. Dabei werden vom Projektionszentrum ausgehend alle Objekte an geraden Linien, welche die Sehstrahlen simulieren, verbunden. [32]

Abbildung 6.1: Perugino: Schlüsselübergabe (Sixtinische Kapelle)

Weitere Beispiele waren die im 18. Jahrhundert sehr populären Rundgemälde. Ein gut erhaltenes Exemplar ist in Innsbruck zu sehen, ein Panoramabild der Schlacht am Bergisel.

Mit der Entwicklung der Computertechnologie wurde es möglich, den Betrachter in eine virtuellen Welt mit Head-Mounted-Display und Kopfhörer zu stellen und ihm darin die Freiheit der Bewegung zu geben. Sowohl für die optische als auch für die akustische räumliche Wahrnehmung ist es dabei von entscheidender Bedeutung wie eine virtuelle Welt auf die Bewegungen des Betrachters reagiert. Sitzt der Betrachter auf einem Sessel und bleibt in dieser Position reicht es, die Position und Drehung seines Kopfs zu bestimmen.

Der Kopf des Betrachters stellt einen Körper im 3-dimensionalen Raum dar welcher 6 Freiheitsgrade[1] besitzt. Diese bestehen aus den drei Euler Winkel $[pitch, yaw, roll]$[2] und der Position $[x, y, z]$.

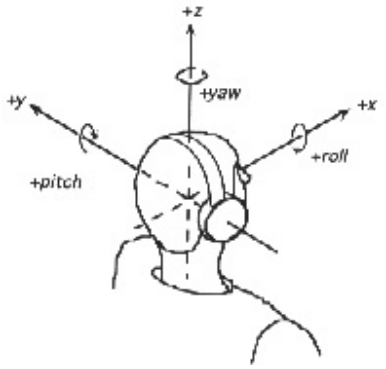

Abbildung 6.2: Euler Winkel - yaw (rotation), pitch (tumble), roll (tilt)

Um diese 6 Freiheitsgrade zu bestimmen gibt es verschiedene Lösungsansätze. Viele davon sind mit kommerziellen Produkten verbunden, einige aber auch als Open-Source Projekte frei verfügbar.

In dieser Arbeit wurde versucht, ein stabiles Head Tracking System welches frei verfügbar ist zu entwickeln bzw. zu verwenden. Dabei kann auf günstig verfügbare Technologie aus der Computerspiele Industrie zurückgegriffen werden. Im folgenden werden zwei im Rahmen der Arbeit adaptierte Systeme basierend auf der WiiRemote von Nintendo und ein System basierend auf dem Tiefensensor Kinect für die Microsoft Xbox vorgestellt [23].

[1] (DOF - degrees of freedom)

[2] Diese Winkel werden in macher Literatur auch $[tilt, rotation, tumble]$ genannt.

6.2 Head Tracking Systeme

6.2.1 Nintendo WII Remote

Die Nintendo WiiRemote bietet sich mit 2 unterschiedlichen Features für Headtracking an. Zum einen ist dies die eingebaute Infrarot Kamera mit einer Auflösung von 128x96 Pixel. Ein integrierter Prozessor kann 4 Infrarot Punkte getrennt erkennen und mit Hilfe einer 8-fachen Subpixel Analyse in einer Auflösung von 1024x768 Punkten als x und y Koordinaten und einem Parameter für die Größe ausgeben.

Zum anderen verfügen neuere Versionen der WiiRemote über die MotionPlus Erweiterung, welche einen zweiachsigen Gyrometer (pitch und roll) und einen einachsigen Gyrometer (yaw) eingebaut hat. Damit können 6 Freiheitsgrade über eine kurze Zeit bestimmt werden. Nur über kurze Zeit deswegen, weil durch die Integration um von den Beschleunigungen zur Position zu kommen, die Fehler addiert werden und nach kurzer Zeit einen Drift verursachen. Die WiiRemote müsste für diese Anwendung auf den Kopf des Hörers befestigt werden.

In dieser Arbeit wurden 3 Infrarot Leuchtdioden in einem gleichschenkligen Dreieck auf einen Funkkopfhörer angeordnet. Die Position dieser Leuchtdioden wird durch die überkopf montierte WiiRemote Infrarot Kamera verfolgt und über Bluetooth an einen Pure Data[3] Patch übermittelt. Dieser errechnet sich aus den Koordinaten der drei Punkte die Drehung des Kopfes und sendet diese mittels OSC[4] Nachricht an das Ambisonics Rotation Plug-in.

$$\vec{P_M} = \frac{\vec{P_2} - \vec{P_1}}{2} \qquad P_M \text{ berechnen} \tag{6.1}$$

$$\vec{D} = \vec{P_3} - \vec{P_M} \qquad \vec{D} \text{ gibt Blickvektor an} \tag{6.2}$$

$$\varphi = atan2(\frac{\vec{D_y}}{\vec{D_x}}) \tag{6.3}$$

Tabelle 6.1: Berechnung des Rotationswinkels (yaw)

[3]Pure Data ist eine visuelle Programmierumgebung für interaktive Computermusik und Multimedia Anwendungen.

[4]Open Sound Control (OSC) ist ein nachrichtenbasiertes Kommunikationsprotokoll für die Kontrolle von Multimedia Systemen (elektronische Musikinstrumente, Mischpulte, ...) Lokal oder im Netzwerk

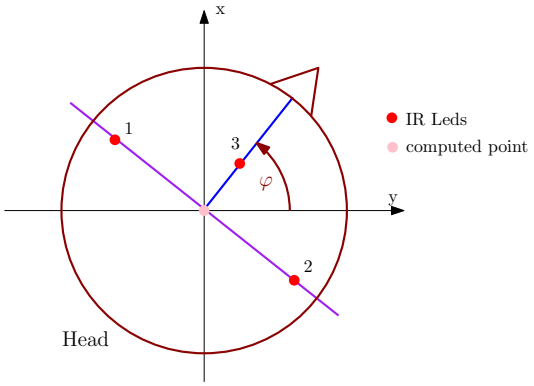

Abbildung 6.3: Bestimmen der Rotation mit 3 IR LEDs und WiiRemote

Eine weitere Methode, welche die Bestimmung aller drei Eulerwinkel mit Hilfe der WII Remote ermöglicht ist in [38] beschrieben. Hierbei wird eine kreisförmige Anordnung von Infrarot Dioden am Kopf verwendet.

6.2.2 Microsoft Xbox Kinect

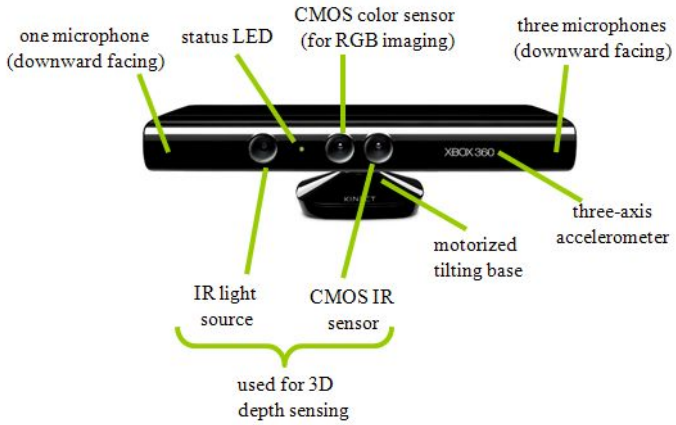

Abbildung 6.4: Microsoft Xbox Kinect 3D Sensor

Der von der Firma Primesense für die Microsoft Xbox Spielekonsole entwickelte Kinect Sensor verfügt über einen Infrarot Emitter, eine Infrarot Kamera, einem Mikrofonarray und eine Farbkamera. Der Infrarot Laser projiziert ein spezielles Muster auf die Oberfächen vor dem Kinect Sensor. Der eingebaute Signalprozessor errechnet mit Hilfe des Infrarot Bildes der Ka-

mera und einem vergleich mit gespeicherten Referenzmustern ein Tiefenbild. In jedem Pixel ist dadurch die Distanz zum Sensor kodiert.

Abbildung 6.5: IR Kamerabild mit Muster, extrahiertes Tiefenbild

Mit Hilfe der Arbeit von *G. Fanelli - Real Time Head Pose Estimation from Consumer Depth Cameras*[14] kann die Kopfposition in allen 6 Freiheitsgraden (Euler Winkel $[pitch, yaw, roll]$ und Position $[x, y, z]$) ermittelt werden.

Der Algorithmus funktioniert nach dem Schema des Random Regression Forest. Dafür wurden mehrere Probanden von einem Meter Entfernung mit Hilfe des Kinect Sensors gefilmt. Die Testpersonen wurden angewiesen frei ihren Kopf in alle möglichen Richtungen zu bewegen, ohne dabei den ganzen Körper zu drehen. Aus diesem Material wurde mit Hilfe der Software faceshift[5] eine Datenbank erstellt, welche einzelne Tiefenbilder und die dazugehörigen Winkel und Kopfpositionen der abgebildeten Probanden enthält.

Der Algorithmus von G. Fanelli detektiert zuerst den Kopf das gegebene Tiefenbildes, vergleicht anschließend diesen Bildbereich mit der Datenbank und versucht so die Kopfposition und die Eulerwinkel zu ermitteln.
Bei dem System wird jedes Bild einzeln analysiert, es benötigt keine Kalibrierung und läuft in Echtzeit. Mit dem derzeit inkludierten Datensatz von über 15.000 Bildern (14 Männer, 6 Frauen, 4 davon mit Brille) können Winkel von ca. $\pm 75°$ yaw, $\pm 60°$ pitch und $\pm 50°$ roll ermittelt werden.[14]

[5]Faceshift ist einen nicht frei erhältliche Software zum Berechnen von Gesichtsgesten und Kopfbewegungen aus Tiefenbildern http://www.faceshift.com

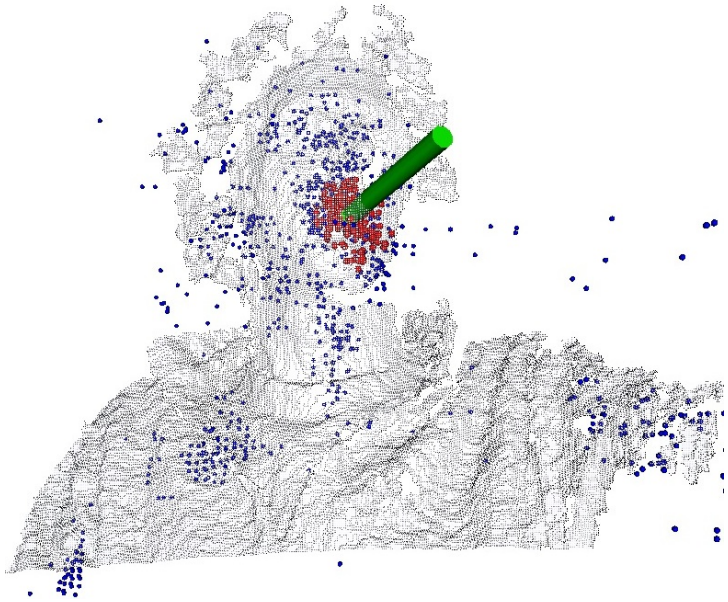

Abbildung 6.6: Visualisierung der Head Pose Estimation von G. Fanelli

Im Rahmen dieser Arbeit wurde der Source Code von G. Fanelli erweitert und das Programm mit OSC Funktionalität ausgestattet. Der Quellcode des Programms ist unter [23] zu finden.
Es sendet zu einer frei konfigurierbaren IP Adresse und UDP[6]-Port folgende OSC Nachricht

```
1 /head_pose [User_ID] [x] [y] [z] [pitch] [yaw] [roll]
```

wobei die Winkel in Grad übergeben werden.
Das Ambisonics 3D Rotations Plug-in kann diese OSC Nachrichten direkt empfangen und dreht anhand der Winkelinformation das Schallfeld in Gegenrichtung.

Mit folgendem Befehl wird das head pose estimation Demonstrationsprogramm gestartet

```
1 ./head_pose_estimation_demo config.txt <show visual 0 or 1> <send osc 0
      or 1> <osc-ip> <osc-port>
```

Falls keine IP-Adresse oder Port übergeben werden ist dieser standardmäßig auf 127.0.0.1:7120 gesetzt.

[6]User Datagram Protocol (UDP) ist ein verbindungsloses Internetprotokoll, welches keine Übertragungssicherheit zu Gunsten einer kleinen Latenz bereitstellt.

6.3 Alternative Systeme

Neben den für diese Arbeit verwendeten Verfahren gibt es noch sehr viel mehr Möglichkeiten zur Bestimmung der Kopfposition.
Folgende Aufzählung legt keinen Wert auf Vollständigkeit.

FreeTrack

FreeTrack[20] ist ein Open-Source Programm für das Windows Betriebssystem, welches mit Hilfe einer normalen Webcam und Markern [Abb. 6.8] die Kopfposition und Winkel bestimmen kann. Für die Erfassung von sechs Freiheitsgraden werden drei Marker in Form von Leuchtdioden am Kopf benötigt.

Abbildung 6.7: Leuchtdioden auf Kappe als Marker für die Software FreeTrack

InterSense InertiaCube BT

Der *InertiaCube BT* der Firma InterSense bietet die Möglichtkeit der kabellosen Detektion der drei Eulerwinkel in 360°.
Er erreicht eine Genauigkeit von ±1° yaw, ±0.5° pitch und ±0.5° roll[21]. Durch die Verwendung der drahtlosen Bluetooth Technologie kann der Sensor bis zu 30 Meter Distanz zum Empfänger überbrücken.

Abbildung 6.8: InterSense InertiaCube BT

PNI Space Point 9-axis

Die Firma PNI Corp. ist spezialisiert auf Sensor Technologie. Der *Space Point 9-axis*[31] Sensor verbindet einen 3-Achsen geomagnetischen Sensor[7] mit einen 3-Achsen Gyrometer[8] und 3-Achsen Beschleunigungssensor. Die mitgelieferten Algorithmen sollen laut Hersteller eine exakte Positionsbestimmung über längere Zeit ohne Drift ermöglichen.
Die Künstlergruppe CREW verwendet bei ihrer interaktiven Theateraufführung *Terra Nova*[9] Sensoren von PNI Corp. für das erfassen der Kopfbewegung des Besuchers. Jeder Besucher ist mit Kopfhörer und Bildschirmbrillen ausgestattet. Anhand der Kopfbewegung wird das Blickfeld und das Schallfeld der virtuellen Realität gedreht.

[7] Geomagnetische Sensor können wie ein Kompass die Orientierung anhand des Erdmagnetfeldes bestimmen.
[8] Ein Gyrometer misst Drehbewegungen.

Kapitel 7

Ausblick

Im Rahmen der Arbeit wurden Ambisonics Audio Plug-ins für die praktische Anwendung in Digitalen Audio Workstations auf verschiedenen Betriebssystemen entwickelt. Die JUCE Programmbibliothek hat sich als stabile Lösung für plattformübergreifende Audio Plug-in Programmierung herausgestellt.

Recherchen haben gezeigt, dass die Verfügbarkeit von Ambisonics Audio Plug-ins höherer Ordnung noch nicht gegeben ist. Diese Arbeit könnte ein Anfang für die Entwicklung einer solchen Sammlung sein.

Die Wiedergabe von 3D Ambisonics Aufnahmen mit Kopfhörern kann einen guten Eindruck über das kodierte Schallfeld vermitteln, ersetzt jedoch nicht ein Abhören über eine reale Lautsprecheranordnung.

Im folgenden sollen einige Optimierungsvorschläge angeführt werden:

- Zur Zeit benötigen die im Rahmen der Arbeit entstandenen Plug-ins noch zu viel Rechenleistung. Der Quellcode müsste optimiert werden. Eine Möglichkeit wäre die Durchführung der Faltungsoperationen in der Frequenzdomäne mittels Fast Fourier Transformation (FFT).
- Hinzufügen von positionsabhängigen Erstreflexionen könnte die Richtungswahrnehmung der Schallquellen steigern. Dies könnte im Enkoder implementiert werden, bevor die Schallquellen in das Ambisonics Format kodiert werden. Eine andere Möglichkeit wäre die Bestimmung der Einfallsrichtung und Trennung der Quellsignale in der Ambisonics Domäne. Diese könnten anschließend um Erstreflexionen ergänzt werden.
- Dekoder mit verschiedenen virtuellen Lautsprecher Aufstellungen sowie austauschbare HRTF Sets würden einen Klangvergleich ermöglichen.

- Enkoder, Rotatoren und Dekoder für höhere und gemischte Ordnungen würde die Flexibiliät erweitern.
- Die Darstellung der Position von Schallquellen beim Enkoder könnte mittels 3d Grafik visualisiert werden.
- Weiterentwicklungen der Software können im Projektarchiv verfolgt werden [22].

Literaturverzeichnis

[1] *VST GUI*, 2012. http://sourceforge.net/projects/vstgui/.

[2] Adriaensen, F.: *AmbDec - Jack Ambisonics Dekoder für Linux und Mac OS X″*, 2005. http://kokkinizita.linuxaudio.org/linuxaudio/downloads/index.html.

[3] Adriaensen, F. und J. Nettingsmeier: *Ladspa Ambisonic Plugins*, Nov. 2011. http://kokkinizita.linuxaudio.org/linuxaudio/.

[4] Berge, S. und N. Barrett: *High Angular Resolution Planewave Expansion.* In: *Proc. of the 2nd International Symposium on Ambisonics and Spherical Acoustics*, May 2010.

[5] beyerdynamic: *Headzone Technology.* http://www.beyerdynamic.de/headphones-headsets/headphones-headsets/headzone/headzone-technology.html, besucht: 2011-03-31.

[6] Chapman, M. und P. Cotterell: *Towards A Comprehensive Account Of Valid Ambisonic Transformations.* In: *Ambisonics Symposium 2009, Graz*, June 2009.

[7] Chapman, M., W. Ritsch, T. Musil, I. Zmölnig, H. Pomberger, F. Zotter und A. Sontacchi: *A standard for interchange of Ambisonic signal sets including a file standard with metadata.* In: *Ambisonics Symposium 2009, Graz*, June 2009.

[8] Coelho, F. und A. Saraiva: *distroh - Cross-Platform Audio Plugins*, 2012. http://distrho.sourceforge.net.

[9] Crew: *Terra Nova*, 2011. http://www.crewonline.org/art/project/138.

[10] Daniel, J.: *Representation de champs acoustiques, application a la transmission et a la reproduction de scenes sonores complexes dans un contexte multimedia.* Dissertation, Universite Paris 6, 2000.

[11] Digenis, A.: *MIT Kemar HRTF C Library*, 2007. http://www.digenis.co.uk/mit_hrtf_lib/.

[12] Digenis, A.: *Ambisonic bidule plugins*, März 2011. http://www.digenis.co.uk.

[13] Dolby Laboratories: *Dolby headphone*, März 2011. http://www.dolby.com/consumer/understand/enhancement/dolby-headphone.html.

[14] Fanelli, G., T. Weise, J. Gall und L. V. Gool: *Real Time Head Pose Estimation from Consumer Depth Cameras*. In: *33rd Annual Symposium of the German Association for Pattern Recognition (DAGM'11)*, September 2011.

[15] Gerzon, M.: *The Design of Precisely Coincident Microphone Arrays for Stereo and Surround Sound*, 1975.

[16] Gilkey, R. und T. Anderson: *Binaural and spatial hearing in real and virtual environments*. Lawrence Erlbaum Assoc., New Jersey, 1997.

[17] grame: *FAUST, Functional Audio Stream*, Okt. 2011. http://faust.grame.fr.

[18] Harpex Ltd.: *Harpex Ambisonics Decoder*, 2012. http://harpex.net/.

[19] Harris, S. und S. Sinclair: *LibLO - Lightweight OSC implementation*, 2007. http://liblo.sourceforge.net.

[20] InterSense: *Free-Track*, 2012. http://www.free-track.net.

[21] InterSense: *InertiaCube BT*, 2012. http://www.intersense.com/pages/18/60/.

[22] Kronlachner, M.: *Ambisonics Audio Plug-ins - Projektarchiv*, Sep. 2012. https://svn.iem.at/svnroot/iem/ambi-plug.

[23] Kronlachner, M.: *Kinect head-pose-estimation - Projektarchiv*, März 2012. http://github.com/kronihias/head-pose-estimation.

[24] Majdak, P.: *Algorithmen in Akustik und Computermusik*. Vorlesungsskript, 2010. http://piotr.majdak.com/alg/VO/spatial1.pdf.

[25] Majdak, P. und M. Noisternig: *Implementation kopfpositionsbezogener Binauraltechnik*. Diplomarbeit, Institut für elektronische Musik - Universität für Musik und darstellende Kunst Graz, Graz, Austria, 2002.

[26] Malham, D.: *Space in Music - Music in Space*. Dissertation, University of York, 2003.

[27] Musil, Sontacchi, Noisternig und Höldrich: *IEM Report 38/07 - Binaural-Ambisonic 4.Ordnung 3D-Raumsimulationsmodell mit ortsvarianten Quellen und Hörerin bzw. Hörer für PD.* Techn. Ber., IEM, 2007. http://old.iem.at/projekte/publications/iem_report/report38_07/report38_07.

[28] Nachbar, C., F. Zotter, E. Deleflie und A. Sontacchi: *Ambix - A Suggested Ambisonics Format.* In: *Ambisonics Symposium 2011, Lexington*, June 2011.

[29] Nettingsmeier, J.: *Ardour and Ambisonics*, 2009. http://cec.sonus.ca/econtact/11_3/nettingsmeier_ambisonics.html.

[30] New Audio Technology GmbH: *HEADPHONE SURROUND 3D*, 2012. http://www.headphone-surround3d.de.

[31] PNI Corp.: *SpacePoint 9-axis sensor*, 2012. http://www.pnicorp.com/products/spacepoint-9-axis-sensor-technology.

[32] Schwarzbauer, C.: *Grundlagen Virtual Reality*, 2005. http://www.dma.ufg.ac.at/app/link/Grundlagen%3AAllgemeine/module/13975.

[33] Sontacchi, A.: *Dreidimensionale Schallfeldreproduktion für Lautsprecher- und Kopfhöreranwendungen.* Dissertation, Technische Universität Graz, 2003.

[34] SRS Labs: *Srs headphone*, März 2011. http://www.srslabs.com/content.aspx?id=397.

[35] Steinberg GmbH: *Steinberg VST PlugIns SDK*, 2.4 Aufl., January 2006. http://www.steinberg.net.

[36] Teragon Audio: *How to make a VST plugin with Xcode or Visual Studio*, Okt. 2011. http://www.teragonaudio.com.

[37] The Ambisonics Association: *Ambisonic Standards*, 2008. http://ambisonics.ch.

[38] Ubilla, M., D. Mery und R. Cadiz: *Head Tracking for 3D Audio using the Nintendo WII Remote.* In: *ICMC 2010, New York*, June 2010.

[39] Warusfel, O.: *Listen hrtf database*, März 2011. http://recherche.ircam.fr/equipes/salles/listen/index.html.

[40] Wave Arts: *Panorama 5*, 2012. http://wavearts.com/products/plugins/panorama/.

[41] Wiggins, B.: *An Investigation into the Real-Time Manipulation and Control of Three- Dimensional Sound Fields.* Dissertation, University of Derby, 2004.

[42] Wiggins, B.: *How to use Ambisonic in Reaper*, Nov. 2011. http://www.brucewiggins.co.uk.

[43] Wightman, F. und D. Kistler: *Resolution of front-back ambiguity in spatial hearing by listener.* J. Acoust. Soc. Am., 105, 1999.

[44] Yamaha Corporation: *Silent cinema technology*, März 2011. http://www.yamaha.co.jp/english/product/av/guide/technologies/bsr/bsr1.html.

[45] Zmölnig, J.: *Entwurf und Implementierung einer Mehrkanal- Beschallungsanlage.* Diplomarbeit, Technische Universität Graz, Kunstuniversität Graz, Graz, Austria, 2002.