Master's Thesis

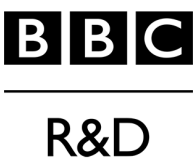
Comparison of Amplitude Panning Approaches on ITU-R BS.2051 Loudspeaker Layouts with Height

Michael Romanov

Supervision: Dr. Matthias Frank (IEM), Dr. Franz Zotter (IEM), Chris Pike (BBC R&D), O.Univ.Prof. Dr. Robert Höldrich (IEM)

Institut für Elektronische Musik und Akustik
BBC Research & Development

Graz, June 11, 2018



Abstract

This thesis describes rendering methods required by the sound objects of the audio definition model (ADM) issued within the ITU. After outlining several methods to do so via amplitude panning and other distance- and reverberation-based effects, the implementation part of this work focuses on amplitude panning methods for direction rendering of sound objects and their realization on standard ITU loudspeaker layouts. The second part ot this thesis focuses on experimental evaluation of amplitude-panning direction rendering. In order to spot differences in the continuum of temporal and directional parameters of the different rendering techniques, technical quality measures were used to obtain an overview of notable technical differences between the various rendering techniques. This technical analysis gives hints on panning directions with pronounced distinction in estimated perceived direction, perceived width, etc. and together with informal listening sessions at BBC R&D and IEM, a listening experiment was designed to deal with perceived quantities. In particular, perceived differences in extent for static sound sources, and in colouration and continuity for moving sound sources were found to deliver relevant distinctive impressions as attributes to be tested with specific directions or trajectories in speed and path. The thesis concludes with the perceptual advantages and disadvantages of every direction renderer and gives implementation examples in python.

Zusammenfassung

Diese Arbeit beschreibt Rendering-Methoden, die von Klangobjekten des Audio Definition Model (ADM) benötigt werden und von der ITU definiert sind. Nach der Beschreibung verschiedener Methoden zur Realisieung von Amplituden-Panning und anderen distanz- und nachhallbasierten Effekten, fokussiert sich der Implementierungsteil dieser Arbeit auf Amplituden-Panning-Methoden zur Richtungsaufbereitung von Klangobjekten und deren Wiedergabe auf Standard ITU Lautsprecher Layouts. Der zweite Teil dieser Arbeit konzentriert sich auf die experimentelle Auswertung verschiedener Amplituden-Panning-Richtungen. Um Unterschiede in der Kontinuität der zeitlichen und direktionalen Parameter der verschiedenen Renderingtechniken zu auszumachen und einen Überblick über relevante technische Unterschiede zwischen den verschiedenen Renderingtechniken zu erhalten, wurden technische Qualitätsmaße verwendet. Diese technische Analyse gibt Hinweise zu Panning-Richtungen mit ausgeprägter Unterscheidung in geschätzter wahrgenommene Richtung und wahrgenommene Quellbreite. Auf Basis dieser technischen Analyse und Hörsitzungen bei BBC R&D und dem IEM, wurde eine Hörversuchsreihe zum Vergleich solcher Rendering Methoden entwickelt. Insbesondere wurden wahrgenommene Unterschiede in der Ausdehnung für statische Schallquellen sowie der Färbung und Kontinuität für bewegte Schallquellen als relevant erachtet und markante Eindrücke als zu testende Attribute mit bestimmten Richtungen oder Trajektorien in Geschwindigkeit und Weg definiert. Die Arbeit schließt mit einer Zusammenfassung der perzeptiven Vor- und Nachteilen jedes Renderers und gibt Beispiele für deren die Implementierung in Python.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline and KUGonline is identical to the present master's thesis.

Date	Signature

Acknowledgements

Writing this thesis would not have been possible without the support and help of many people.

First of all i would like to express my special thanks to Dr. Matthias Frank and Dr. Franz Zotter who have always been available for very fruitful discussions about implementations, listening test design and statistics.

My special thanks goes to the audio team at the BBC R&D. I always felt welcome, valued, and part of the team from the first day of my internship in Manchester. It helped me a lot to improve my English.

Thanks to Dr. Frank Melchior who always critically questioned all my development steps, listening test designs and results.

Also i want like to thank Chris Pike for all the critical discussions and the supervision of this work from the side of the BBC and for all the corporations that have raised after my internship in Manchester.

Another special thanks goes to Tom Nixon for patiently answering all my questions about python and coding in general.

Furthermore, I would like to thank all the participants of the listening tests, colleagues and friends that spent their spare time on these experiments.

Last but not least i want to thank my parents Galina and Victor Romanov and my whole family for always supporting me during the last eight years of studies and always believing in my decisions.

Contents

1 Introduction

In the age of virtual reality, the trend is towards immersive 360° playback of audiovisual content and interactivity. This content can be played back in virtual reality headset glasses, 360° videos, dome-to-loudspeaker concert halls, home multimedia systems, studios, and vehicles. Of particular relevance is the fact that Ambisonics, a method for the spatial recording and playback of a sound field, has recently been used uniformly in the 360° video formats of Youtube and Facebook, thus opening this area of technology to the mass market.

This raises the question of how 3D content for such systems can be generated, stored and transmitted, which leads to new demands on the media formats. In contrast to classical stereo recordings, these media formats must, in addition to the pure audio information, additionally also contain metadata about the room-acoustic setting, such as e.g. position of an instrument or the reverberation included. This should make it possible to design the content format independently of the concrete playback system. The ITU (International Telecommunication Union) defines a special format (ADM - Audio Definition Model) [1], which unites audio and metadata in a uniform manner. This can then be stored independently of the end user system, transmitted and finally reproduced individually and sound optimized.

Although the media format ADM is now standardized, the international experience of recent years shows that in professional studio production of 3D audio content, the sound quality and spatial aspects of the playback are extremely different from the monitoring system used in a particular studio.

A scientifically sound determination of these spatial distortion effects is not available to date and is the subject of this thesis. Their primary goal is therefore the psychoacoustic evaluation of several professional reproduction systems in terms of localization, colouration, and smoothness of movements of different sound content. By means of extensive listening tests with trained listeners, the weak points of the various systems and their potential for improvement should be identified. As a result, the work should provide information about a generically usable, studio-compatible speaker configuration with excellent surround sound rendering and the necessary optimized control methods.

This will give studio designers and sound engineers in the area of movies, broadcast, gaming and music a tool to ensure a uniform and reliable production workflow in the booming and diverse market for 3D AV content.

Rendering in general describes the process of generating a final digital product from a specific type of input data. The term usually applies to graphics and video, but can also refer to audio. For spatial audio this means generating a channel-based audio mix to a specific loudspeaker layout out of multichannel audio files and attached meta data that describes attributes of these audio files such as position in space or even just volume to name a few possible parameters. The combination of audio source and meta data independent from an aimed reproduction systems are called audio-objects. So the renderer should produce should be able to deal with such audio-objects and interpret meta data in a suitable way to first create a scene-based mix of all audio-objects and finally generate the channel-based mix for a specific loudspeaker layout. A scene-based audio mix does not contain the meta data any more but is not reproduced to a specific loudspeaker layout as e.g. an Ambisonics mix.

1.1 Audio Definition Model

The new standard by the Radiocommunication Sector of the ITU [1], called Audio Definition Model, and specifies how to store and broadcast spatial audio content. It is also part of the MPEG-H standard immersive TV audio system [2]. Along with the audio data, it contains meta data such as position, extent, or diffuseness for every virtual sound source. Position of the source is defined in spherical coordinates (angles and distance) or cartesian coordinates. It hereby allows to store content independently of the audio and rendering system. It also causes the need for any renderer to be capable of rendering this information to standard loudspeaker layouts. This thesis presents several common rendering methods and a perceptual evaluation of basic performance measures in listening experiments.

1.2 ITU Standard Loudspeaker Layouts

Beside the traditional 2.0 stereo and 5.1 surround loudspeaker layouts, ITU-R BS.2051 [1] standardized a variety of layouts that also include loudspeakers above and below the horizontal plane, i.e. the ear height of the listeners. The number of loudspeakers in each of the three height layers is denoted in the notation of each layout, e.g. the 4+5+0 layout comprises 4 loudspeakers in above the listener, 5 around the listener, and none below. A reference listening room for 3D audio research containing all of these layouts can be found at the BBC R&D in Manchester [3].

In general, ADM should allow rendering on any loudspeaker layout. Nevertheless, reproducible configurations are required and defined by ITU's eight standard loudspeaker layouts [4]. They contain standard two-channel stereophony 0+2+0 (A), standard 5.1 or 5.0 as 0+5+0 (B), and the layouts 2+5+0 (C), 4+5+0 (D) that are extensions of the 5.0 layout by additional height loudspeakers. The layout 4+5+1 (E) is moreover extended by an additional floor loudspeaker in front. 3+7+0 (F) and 4+9+0 (G) are a bit more uniformly spaced, and 9+10+3 (H) denotes the standard 22.0 layout. Any ADM-capable rendering methods should at least be able to render audio to every of those eight layouts. For exact coordinates and tolerances cf. [4]. The following tables represent the azimuth and elevation coordinates for layouts C-H whereby the loudspeakers are assumed to be on a sphere. When this is not the case they should preferably be time aligned (at the central listening position) with an accuracy of $100\mu s$.

Table 1 – Loudspeaker positions of the 2+5+0 (C) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-30	0
3	30	0
4	-110	0
5	110	0
6	-30	30
7	30	30

Table 2 – Loudspeaker positions of the 4+5+0 (D) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-30	0
3	30	0
4	-110	0
5	110	0
6	-30	30
7	30	30
8	-110	30
9	110	30

Table 3 – Loudspeaker positions of the 4+5+1 (E) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-30	0
3	30	0
4	-110	0
5	110	0
6	-30	30
7	30	30
8	-110	30
9	110	30
10	0	-30

Table 4 – Loudspeaker positions of the 3+7+0 (F) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-30	0
3	30	0
4	-90	0
5	90	0
6	-135	0
7	135	0
8	-45	30
9	45	30
10	180	45

Table 5 – Loudspeaker positions of the 4+9+0 (G) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-20	0
3	20	0
4	-30	0
5	30	0
6	-90	0
7	90	0
8	-135	0
9	135	0
10	-45	30
11	45	30
12	-110	30
13	110	30

Table 6 – Loudspeaker positions of the 9+10+3 (H) layout.

lspk.idx	elevation / °	azimuth / °
1	0	0
2	-30	0
3	30	0
4	-60	0
5	60	0
6	-90	0
7	90	0
8	-135	0
9	135	0
10	180	0
11	0	30
12	-45	30
13	45	30
14	-90	30
15	90	30
16	-135	30
17	135	30
18	180	30
19	0	90
20	0	-30
21	-45	-30
22	45	-30

2 Ambisonics Renderer Implementation

In this thesis, among other typical amplitude panning techniques, an Ambisonic renderer was implemented in python together with colleagues at BBC R&D in order to facilitate comparative evaluation.

A library introduced here is called “ambipy”. It contains all basic rendering functionalities as well as some extended rendering functionalities that are presented in this chapter.

Special functionalities and fundamentals are explained in this chapter and the whole “ambipy” library can be found in the appendix of this thesis.

2.1 Fundamentals of Ambisonics

2.1.1 Spherical Coordinates

We begin with the description of the common used coordinate systems in the work with spatial audio rendering systems. To describe linear movements in the space it is common to use cartesian coordinates. To describe circular movements it is common to use spherical coordinates. In this work we will mostly use spherical coordinate systems because so far the most rendering systems are only developed as a point source panning systems along the unit circle.

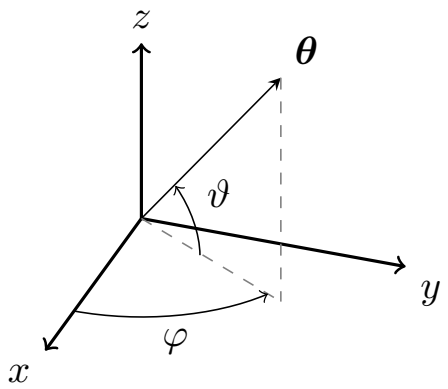


Figure 1 – Cartesian and spherical coordinate systems.

φ describes the azimuth and ϑ the elevation angle. r represents the radius that can be understood as distance from the origin.

The transformation from cartesian coordinates to spherical is performed as described in

$$r = \sqrt{x^2 + y^2 + z^2}, \tag{1}$$

$$\varphi = \arctan \frac{y}{x}, \tag{2}$$

$$\vartheta = \arcsin \frac{z}{\sqrt{x^2 + y^2 + z^2}} = \arcsin \frac{z}{r}, \tag{3}$$

The other way round the computation is described as followed:

$$x = r \cos \vartheta \cos \varphi, \tag{4}$$

$$y = r \cos \vartheta \sin \varphi, \tag{5}$$

$$z = r \sin \vartheta. \tag{6}$$

2.1.2 Spherical Harmonics

To get an understanding of how Ambisonics works is to get familiar with the spherical harmonic representation directivity patterns. A proposed representation has been published in [5]

$$Y_n^m(\varphi, \vartheta) = N_n^{|m|} P_n^{|m|}(\sin \vartheta) \begin{cases} \sin(|m|\varphi), \ for\ m < 0 \\ \cos(|m|\varphi), \ for\ m \geq 0 \end{cases} . \tag{7}$$

Hereby n is the order and m the degree. $P_n^{|m|}$ are the Legendre functions and $N_n^{|m|}$ represents the normalization function and for the trigonometrical functions and for the Legendre functions. Fig. 2 visualizes the spherical harmonics.

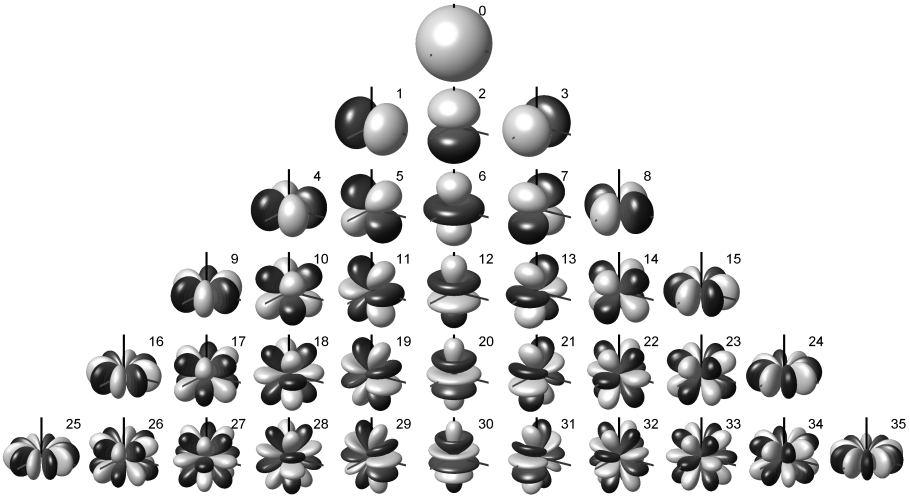


Figure 2 – The spherical harmonics for 5$^{\text{th}}$-order Ambisonics (Source: Zotter).

Every Ambisonics channel has a number that can be calculated like shown in Eq. 8

$$ACN = n^2 + n + m. \tag{8}$$

For $N_n^{|m|}$ in Ambix is defined with the SN3D normalization in

$$N_n^{|m|} = \sqrt{\frac{2 - \delta_m}{4\pi} \frac{(n - |m|!)}{n + |m|!}}. \tag{9}$$

The number of channels in a system is defined by

$$k = (n + 1)^2. \tag{10}$$

whereby n is the Ambisonics order. The zeroth Ambisonics channel (the 0$^{\text{th}}$-order channel) is the omnidirectional channel. All other spherical harmonics are combination of spherical sine and cosine functions.

2.1.3 Encoding

We can represent a sound source as a directivity pattern composed by a input signal weighed with the sampled spherical harmonics at a specific direction. It is very similar to the beamforming with microphone signals. If you overlay a omnidirectional microphone with a figure-of-eight (dipole) microphone you end up with a cardioid directivity pattern. Same happens in the spherical harmonics domain. Over layering the omnidirectional di-

rectivity pattern with the spherical eight functions result in a spherical cardoid directivity pattern. Increasing the Ambisonics order leads the directivity pattern to be more focused like a super cardioid pattern. The focus of the directivity pattern increases with the Ambisonics order n.

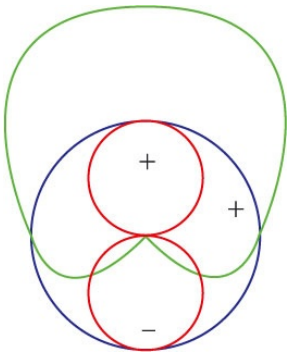


Figure 3 – Superposition of a omnidirectional and a figure-of-eight pattern that result in a cardioid pattern in two dimensions.

To bring a mono signal $s(t)$ in the Ambisonics domain we just calculate the weights of every spherical harmonic for the chosen Ambisonics order and a given direction (φ, ϑ), multiply the input signal with the weight for every Ambisonics channel yields $k = (n+1)^2$ Ambisonic channels

$$\chi_n^m(t) = Y_n^m(\varphi, \vartheta)s(t). \tag{11}$$

2.1.4 The theoretically direction-continuous Ambisonic signal

With the set of Ambisonic signals $\chi_n^m(t)$, a theoretically direction-continuous surround signal can be calculated

$$x(\varphi, \vartheta, t) = \sum_{n=0}^{N} \sum_{m=-n}^{n} (2n+1) Y_n^m(\varphi, \vartheta)\chi_n^m(t). \tag{12}$$

It serves as an intermediate representation that rendering to loudspeakers aims to reproduce. Suitable discratization (=decoding) methods of its direction-continuous nature permits play back Ambisonic content to a somewhat variable loudspeaker layout by a suitable Ambisonic decoder.

The basic idea behind the Ambisonics decoding is sampling the spherical harmonic representation of the Ambisonics signal at the loudspeaker positions. The decoding process

as itself is also quite simple. It is the multiplication of the Ambisonics signals with a decoding matrix D with the dimension k channels to L loudspeakers

$$\mathbf{x}(t) = \mathbf{D}\ \chi(t). \tag{13}$$

So this looks very simple but the main effort is the design of such a decoder matrix. The decoder design is explained in a further chapter of this thesis with different approaches.

2.2 Decoder Design

The design of an Ambisonics decoder can be done by various solutions. In the next chapter we will introduce two simple decoders, the sampling decoder and a decoder using the inverse matrix. These decoders do work only in special cases. An better approach is the so called AllRAD Decoder [6] that combines the classic Ambisonics approach with Vector Base Amplitude Panning. VBAP and AllRAD will be explained in the following section.

2.2.1 Sampling decoder

The sampling decoder is the simplest Ambisonics decoder approach. It simply samples the Ambisonics directivity pattern in loudspeaker positions and applies a normalization $\frac{4\pi}{L}$ like described in Eq. 14

$$\mathbf{D} = \frac{4\pi}{L}\mathbf{Y}_N^T \mathrm{diag}\{[2n+1]_n^m\}, \tag{14}$$

whereby $[2n+1]_n^m$ = [1 3 3 3 5 5 5 5 5 ...]. This is simply achieved by transposing the matrix of sampled spherical harmonics in the loudspeaker positions.

The problem here is that in areas covered by a lot loudspeakers it will we louder because more loudspeakers are sampling the main lobe of the Ambisonics directivity pattern. Also it can be possible that loudspeakers will not play if a zero crossing is sampled. This all will result in not equal loudness distributions.

2.2.2 Inverse Decoder

A more advanced but not ideal way to achieve a Ambisonics decoding matrix is to sample the spherical harmonics at the loudspeaker position coordinates and take the inverse matrix that leads us to the decoder matrix. This approach was published by Jerome Daniel [7] also called “mode-matching decoder”

$$\boldsymbol{\chi}(t) = \mathbf{Y}\boldsymbol{x}(t). \tag{15}$$

So $\mathbf{Y}$ is called the re-encoding matrix. It represents values of spherical harmonics at the loudspeaker position directions. $\boldsymbol{\chi}(t)$ is the Ambisonics signal vector. By inverting this matrix we get the decoding matrix

$$\boldsymbol{x}(t) = \mathbf{Y}^{-1}\boldsymbol{\chi}(t), \tag{16}$$

$$\mathbf{D} = \mathbf{Y}^{-1}. \tag{17}$$

In terms of linear algebra it is clear that this method works only if this matrix is a square matrix. That means that there a as many loudspeaker as Ambisonics channels.

This is the case in a full sphere layout with equidistantly spread loudspeakers.

If this is not the case then it is only possible to compute the pseudo inverse matrix. We will only get good results if the matrix $\mathbf{Y}$ is well conditioned and this is only the case if we have a equidistant and equally spread loudspeaker distribution. So the pseudo inverse and the resulting gain weights are computed as a right inverse

$$\begin{aligned} pinv(\mathbf{Y}) &= \mathbf{Y}^T(\mathbf{Y}\mathbf{Y}^T)^{-1} \\ \boldsymbol{x(t)} = pinv(\mathbf{Y})\boldsymbol{\chi}(t) &= \mathbf{Y}^T(\mathbf{Y}\mathbf{Y}^T)^{-1}\boldsymbol{\chi}(t). \end{aligned} \tag{18}$$

In the BS.2051 loudspeaker layouts we will find mostly non ideal loudspeaker distributions so this method will fail in the most cases.

2.2.3 VBAP

A common panning rule for multiple loudspeakers called Vector Base Amplitude Panning has been published by Ville Pulkki in 1997 [8]. In general for the three dimensional case a sound source can be reproduced as a phantom source using three loudspeakers within a loudspeaker triangle. A triangulation of a loudspeaker layout can be achieved by calculating the convex hull [9] of a layout.

In python it is very easy to compute the convex hull by one command using the spatial part of the scipy library, whereby the function expects a list of point positions in cartesian coordinates and returns a list of point triples that represent the triangulation. The origin must lie within the convex hull then the implementation goes as follows:

```
from scipy import spatial
triangulation = spatial.ConvexHull(speakerpos_cart).simplices
```

If we have found a triangulation we figure out within which triangle the sound source needs to be reproduced and then reproduce the sound source within this triangle using the

three loudspeakers.

The gain funtion for one loudspeaker triangle is given as:

$$\mathbf{p} = g_1\mathbf{l_1} + g_2\mathbf{l_2} + g_3\mathbf{l_3}, \tag{19}$$

$$\mathbf{p}^T = \mathbf{g}\mathbf{L_{123}}. \tag{20}$$

Whereby l_l is the representation of the normalized loudspeaker positions and $\mathbf{p}$ is the three dimensional unit vector of the source position and g are the gain weights for the three loudspeakers within this triangle. $\mathbf{L_{123}}$ represents the vector base. We can compute the gain weighs for this triangle as shown in Eq. 22

$$\mathbf{g} = \mathbf{p}^T{\mathbf{L_{123}}}^{-1} = \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33}. \end{bmatrix}^{-1}. \tag{21}$$

As a last step the gains values need to be normalized as followed

$$\mathbf{g}_{scaled} = \frac{\mathbf{g}}{\sqrt{g_1^2 + g_2^2 + g_3^2}}. \tag{22}$$

As this was the representation of one triangle we can define more triangles as calculated with the convex hull as defined in [9]. One loudspeaker can be part of multiple triangles. So to select the active triangle we simply can mute all triangles that contain negative gain values. To avoid numerical issues we need to add a offset to the values at $-\infty$ with $\varepsilon = 10^{-11}$.

So a rendering system needs to take a list with the given triangulation to compute the vector bases, a routine that mutes the non-active triangles and a normalization. All the triangles are calculated in parallel before the decision of muting.

There is no native implementation of VBAP in python but a solution as a python object with a rendering function using the numpy library in presented here:

```
import numpy as np

class VBAP(object):
  """VBAP implementation with manual triangulation."""

  def __init__(self, speaker_positions, triangles):
  """Initialise and compute basises using loudspeaker positions and a
    list
  of triangles which indexes into the speaker positions."""
    self.num_speakers = len(speaker_positions)
    self.basises = np.array([np.linalg.inv(speaker_positions[triangle].
   T) for triangle in triangles])
    self.triangles = triangles

  def render(self, position):
  """Calculate panning values for a source position using source
    position and
  returns normalised panning values for each speaker position"""
    for triangle, basis in zip(self.triangles, self.basises):
    g = np.dot(basis, position)

    # If position is within this triangle, all the panning values
    # need to be positive. To avoid numerical issues on triangle edges
    # slightly negative.
    eps = -1e-11
    if g[0] >= eps and g[1] >= eps and g[2] >= eps:
      g /= np.linalg.norm(g)
      g.clip(0, 1, out=g)  # check if all gains are positive
      g_all = np.zeros(self.num_speakers) # channel order remapping
      g_all[triangle] = g

    return g_all
```

This basic VBAP Method has one feature that can cause big variations in the sound colour. If the source position vector faces a single loudspeaker then only this speaker will play. If it faces the middle of a speaker triangle then three loudspeakers will play at once, what will cause some comb filter effects as well as a minor change in source with between on and off speaker positions.

However, VBAP is a quite simple approach that needs low computational resources and works quite well if we have a equidistant loudspeaker configuration where there is only

one possible solution for the convex hull [9] that minimizes the maximal distances between the loudspeakers. In loudspeaker layouts as we can find them in the BS.2051 we find areas, where it is possible to calculate the unambiguously what can lead to a non symmetrical triangulation. Some approaches are discussed in the following section.

To avoid silent areas and inconstancy for gain values in opening angles with more than 90 degrees it is helpful to insert virtual loudspeakers. This leads to two options: Just calculate the gains for all loudspeakers and just skip the virtual loudspeaker or spread the matrix row that represents the decoding for the surrounding real loudspeakers in equal parts. Therefore an ideal position for the virtual loudspeakers has to be found. Approaches to achieve that will be presented in a later chapter of this thesis

2.2.4 AllRAD Ambisonics Decoder

An approach to optimize the Ambisonics decoding matrix in a elegant way was published by Franz Zotter and Matthias Frank in 2012 called the All Round Ambisonics Decoder [6]. This approach is based on a combination of the Ambisonics and the VBAP approach. In a first step a sound source in the Ambisonics domain is sampled in a high resolution (about 5200 sample points) virtual t-design loudspeaker layout that has a optimal geometry for reproducing sound sources. Afterwards the signals that reach the virtual loudspeakers are rendered down within a VBAP triangle to the surrounding real loudspeakers in the given loudspeaker layout. Compared to the classic VBAP approach all loudspeakers are always playing because the vitual t-design covers the whole sphere.

In a first step we use a simple sampling decoder $\mathbf{D_{virt}}$ at all virtual loudspeaker positions of the t-design. Using a virtual t-design results in equal loudness and source width distribution.

As the second step we now can apply VBAP to render down the virtual loudspeakers to the real loudspeakers. It is enough to do this once because the virtual speaker are not moving so one matrix $\mathbf{G}$ represents all the VBAP rendering.

We apply the VBAP rendering function $\mathbf{G}$ to the transpose re-encoding and end up with the final AllRAD decoding matrix.

$$\mathbf{D_{allrad}} = \mathbf{G}\mathbf{D_{virt}}. \tag{23}$$

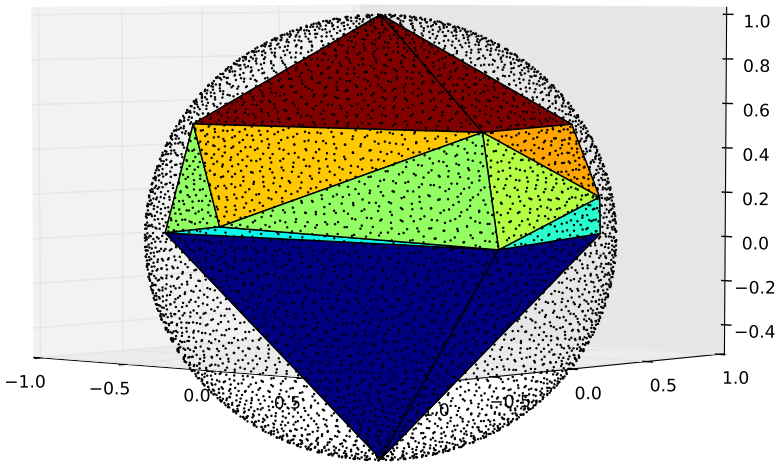


Figure 4 – AllRAD decoder visualisation of an exemplary triangulation on loudspeaker layout D and a sphere of 5200 sampling points.

The implementation of the AllRAD decoder is pretty simple. There is only a need of the VBAP object, a list of 5200 cirtual loudspeaker positions (data) and a simple Ambisonics panner (sample decoder) and the computation can be performed as follows:

```
import numpy as np
import ambipy
import scipy as sp
import data


def design_allrad(render_func, nchannels, order):

  points = data.points

  V2R = np.zeros((len(points), nchannels))
  for i in xrange(len(points)):
    V2R[i] = render_func(points[i])

  az = np.arctan2(points[:,0], points[:,1])
  el = np.arctan2(points[:,2], np.hypot(points[:,0], points[:,1]))
  K = ambipy.Ambi_Panner(order).ambi_pan(az, el)

  D = np.dot(V2R.T, np.linalg.pinv(K).conj().T)
```

2.2.5 max-rE-weighting

There is a way to focus the directivity pattern in the Ambisonics domain to get a max-rE energy distribution that can be found in [6]. In general this algorithm is damping the side lobes of the Ambisonics directivity pattern and you end up with more focused sound source. The weighting is explained in Eq. 24

$$a_n = P_n(\cos(\frac{137.9^\circ}{N + 1.51})). \tag{24}$$

This computation can be applied during the decoder design. In this case the computation has only to be performed once compared to the calculation during the rendering process at itself where it needs to be applied during every rendering iteration.

A computation of max-rE weighting is presented as follows:

```
for i in range(0,D.shape[1]):
  n = sp.floor(sp.sqrt(i))
  w = np.polyval(sp.special.legendre(n),sp.cos(137.9*np.pi/180/(order
    +1.51)))
  D.T[i] *= w
```

2.3 Extended Rendering Functionalities

2.3.1 Distance Coding

The distance coding is split into two sections. The first section describes the exterior behaviour outside the unit circle and the second section describes a approach for the interior behaviour within the unit circle.

2.3.2 Exterior Behaviour

For the exterior behaviour we assume the inverse disctance law $1/r$ from theoretical acoustics for point sources in a free field. The calculation can be performed directly to the speaker gains at the end of the rendering chain. If a normalization of loudspeaker gains is done at the end of the rendering chain then the exterior distance coding has to be applied afterwards. Otherwise the normalisation would efface the effect. We use Eq. 25 to realize exterior behaviour

$$g_d = \begin{cases} \frac{1}{r} g_l, & \text{for } r > 1 \\ g_l, & \text{for } r = 1 \end{cases} . \tag{25}$$

2.3.3 Interior Behaviour

The interior behaviour can be achieved by a simple manipulation in the Ambisonics domain. As mentioned earlier the first Ambisonics channel contains the omnidirectional information. If we gain this to 1 and damp all other channels to 0 then all loudspeaker will play with the same gain. So if we assume the constraints that for $r = 1$ the gain for the 0^{th}-order channel is 0.28 and for $r = 0$ the gain is 1. For the other channels we just multiply the gain with r that leads to 0 gain for all Ambisonics channels for $r = 0$. So solving the equation using the mentioned constraints leads us to following context:

$$\hat{\mathbf{y}}_{\mathbf{ACN}} = \begin{cases} 0.28(3.57 - 2.57r)\mathbf{y}_{\mathbf{ACN}}, & \text{for } ACN = 0 \\ \mathbf{y}_{\mathbf{ACN}} r, & \text{for } ACN \geq 1 \end{cases} . \tag{26}$$

This function realizes a smooth balance between the omnidirectional and directional information. For $r = 0$ this causes a phantom source within the head if the listener sits in the sweet spot. One disadvantage of this method is that the ceiling speaker also start

to play when the radius decreases under one. But for layouts without the voice of god speaker this effect is not that pronounced.

2.3.4 Spatial Lowpass Filter (Order Reduction)

To switch between the Ambisonics orders in a smooth way there is one possible solution. The Ambisonics channels belonging to an order can be faded out stepwise.

We let the omnidirectional 0^{th}-order channel be 1 in every case

$$g_0 = 1. \tag{27}$$

For the other channels we apply a linear gain function to all channels belonings to a Ambisonics order. If we define a number i that goes from 0 to the maximum order n of the rendering system we can apply following function to get a liner Ambisonics order fading system.

$$g_n = \begin{cases} i - n + 1 \text{ for } n - 1 < i < n \\ 0 \text{ for } i < n - 1 \\ 1 \text{ for } i > n \end{cases} . \tag{28}$$

In this context we have a similarity to a low pass filter system. If you imagine the lower Ambisonics orders such as lower base frequencies and the higher Ambisonics order signals as high frequencies. Increasing the Ambisonics order yields to a more focused directivity pattern. So i f you mute the higher order signals you increase your source width. This procedure need to be applied in the Ambisonics domain.

2.3.5 Rotation

As we need the Ambisonic rotation for the implementation of the ambisonic widening and diffusion approach in [10] we implemented a efficient rotation function based on the proposal by Franz Zotter in [11]. The z-axis is efficient rotation is easy to implement. To save computational power we can explain they-rotation in terms of a $z{-}y(\pi/2){-}z{-}y(\pi/2){-}z$ rotation containing only constant rotations around y. The $y(\pi/2)$ rotation is one fixed matrix that does not need to be calculated for every rotation operation. Pre stored matrices provided by Franz Zotter (`https://ambisonics.iem.at/xchange/fileformat/docs/spherical-harmonics-rotation`) have been used. A visualisation of this ap-

proach is presented in Fig. 6

$$\begin{aligned}\boldsymbol{r}' &= \boldsymbol{Q}(\alpha,\beta,\gamma)\,\boldsymbol{r}\\ &= \boldsymbol{Q}_z(\alpha)\,\boldsymbol{Q}_y(\beta)\,\boldsymbol{Q}_z(\gamma)\,\boldsymbol{r}\\ &= \underbrace{\boldsymbol{Q}_z(\alpha+\pi/2)\,\boldsymbol{Q}_{y\frac{\pi}{2}}\,\boldsymbol{Q}_z(\beta+\pi)\,\boldsymbol{Q}_{y\frac{\pi}{2}}\,\boldsymbol{Q}_z(\gamma+\pi/2)}_{\boldsymbol{Q}_{z\text{-}y\frac{\pi}{2}\text{-}z\text{-}y\frac{\pi}{2}\text{-}z}(\alpha,\beta,\gamma)}\,\boldsymbol{r}.\end{aligned}$$

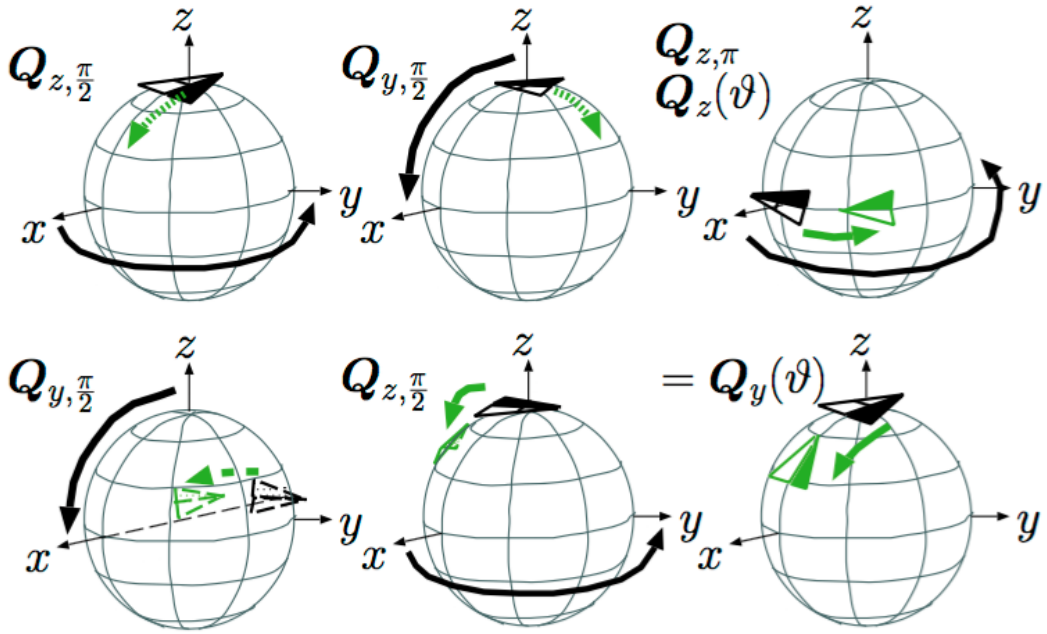


Figure 5 – Rotation computing [11]

The implementation can be found in the ambipy/diffusion library in the annex of this thesis.

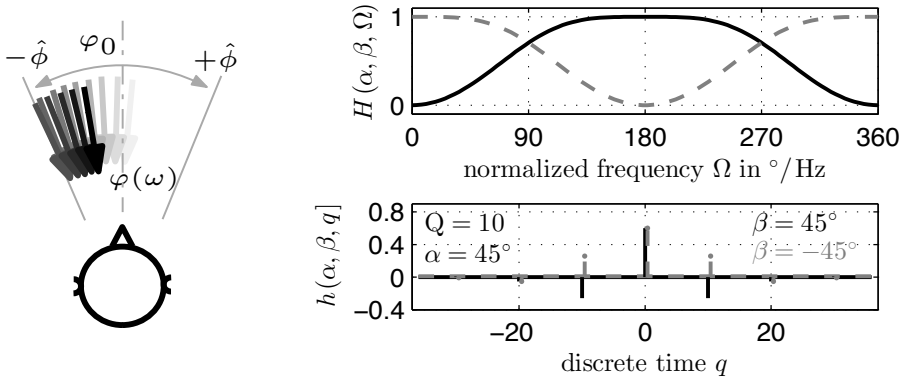
2.3.6 Widening/Diffusion

As a proposal to deal with the diffusion and widening parameter in the ADM there is a approach of Matthias Frank and Franz Zotter [10] that is also used in the Ambix Plugins by Matthias Kronlachner. It can be used for widening and for diffusion depending on the input parameters of the implemented functions.

In general the idea is to similar to a frequency dependant stereo widening approach where we can use two different filters for right and left channels with an whose two frequency

responses in sum yield to a liner frequency response. The transfer function in this case is a cosine (sine) in the frequency domain along the frequency axis for left and a phase shifted version of $90°$ at the right channel.

This context is illustrated in Fig.6.



(a) Basic idea. (b) $H(\alpha, \beta, \Omega)$ has a sparse time response.

Figure 6 – Stereo widening as basis for Ambisonics widening [10].

In time domain this represents a $sinc$ shaped series of delayed impulses. This pattern can be convolved with the input signal for every output channel. It can also be implemented as a series of weighted time delays.

However this approach can be used for Ambisonics widening as well. Instead of Two channels as in the stereo version we here have a lot of symmetrical pairs of spherical harmonics channels. We increase the frequency of the cosine shape of the frequency response for every spherical harmonic pair in the frequency domain and end up with a set of impulse responses that are $sinc$-shaped and differ in the time delay values for every pair. Using the approach in the [10], we get a spread function that is dependant of a widening angle and also performs a rotation as described in the rotation chapter of this thesis.

To get a better sounding result is to use only the causal part of the impulse response. That means so truncate the non causal side of the impulse response and move the highest point to zero position. This approach only sound good for a spread angle less than $65°$. For greater angles it produces bad sounding artefacts. The causal and the non causal approach have been implemented in the ambipy/widening and ambipy/widening_causal whereby the causal version is proposed to use within a renderer for better sounding results [10].

The normal widener can be used to renderer the z widening parameter in the ADM. For the y-widening a combination of the z-widening and the y-rotation can be used to achieve

widening along the y-axis.

As in this work we investigate only the behaviour of the point source panner and also the BBC internal rendering system does not support multichannel-bus effects within the rendering chain at this point. However the implementations can be used for research and further works on the Ambisonics rendering system. A detailed explanation of this algorithm can be found in [10].

2.3.7 Feedback Delay Network

As an different approach to achieve diffuseness in a rendering system we implemented a feedback delay network based on a matlab script from Udo Zölzers “DAFX" [12]. The idea here is to implement an Ambisonic feedback delay network that instead of summing up all the delayed signals at one point and feed to the system output this approach feeds the calculated delays in to the Ambisonics channels.

Hereby the calculated delays are chosen in prime number time steps to achieve a maximum decorrelation. The output of summing points need to be limited to 1. If the time delays do not exceed the psychoacoustic echo threshold of 30-50 ms, listeners should perceive a highly diffuse reverberation.

We implemented a python version of this combined approach. It works in its basic idea but the sound is very metallic. As an approach to reach a realistic and not metallic sounding reverberation is modulating the feedback matrix with a oscillator as it is done by chorus effects [12].

As in this work we investigate only the behaviour of the point source panning system and also the BBC internal rendering system does not support multichannel-bus effects within the rendering chain at this point, we will not further optimize the algorithm. But this idea can be kept for further works on the Ambisonics rendering system.

3 Manipulations Improving Amplitude Panning in Case of Loudspeaker Quadrilaterals or Higher Polygons

Text parts and figures of this chapter were published at the Tonmeistertagung 2016 as a scientific paper called “Manipulations improving amplitude panning on small standard loudspeaker arrangements for surround with height” [13].

In particular, loudspeakers are sometimes arranged such that the convex hull delivers a planar conglomerate of triangles that could be gathered to (nearly) plane quadrilaterals, pentagons, hexagons, or other l-gons with $l > 3$. In such cases, special strategies are required and a set of tools to permit a technical quality of ampltude panning. Whenever the convex triangulation of the surrounding loudspeakers results in triangles that are coplanar, the geometric decomposition into the particular choice of triangles can appear somewhat arbitrary. It might appear that using quadrilaterals or pentagons would be the better choice, however, amplitude panning is typically easier to define based on triangles. Within, e.g., a quadrilateral of 4 loudspeakers {1,2,3,4}, one could often choose between either the triangulation {[1,3,2], [1,3,4]} with the diagonal $\{1, 3\}$ or its alternative {[2,4,1], [2,4,3]} with the diagonal $\{2, 4\}$. While for quadrilaterals, it might be a remedy to superimpose panning gains of the two alternatives, the triangulation of pentagons or hexagons have many more alternatives to be discussed.

3.1 Manipulation Strategies

As a remedy of most problems, there are three general approaches:

- Imaginary loudspeaker insertion (and *downmix* to available loudspeakers) to "fill the gaps" [6].
- Imaginary loudspeaker insertion (and *downmix* to available loudspeakers) to enforce symmetries [14].
- Idealisation of the physical loudspeaker layout.

3.2 Quality Prediction Metrics

To analyse the quality of the panning strategy with regard to amplitude preservation of signals across the panning directions, we typically regard the squared norm of the amplitude-panning gains g_l for every loudspeaker l in the number of loudspeakers L for a certain layout,

$$E(\boldsymbol{\theta}) = \sum_{l=1}^{L} g_l^2(\boldsymbol{\theta}) \tag{29}$$

or called $E(\boldsymbol{\theta})$ measure by Gerzon [18].

Gerzon's $\boldsymbol{r}_\mathrm{E}$ vector measure models the perceived direction of a virtual source and its length models the perceived source width [18]. It can also be used visualize asymmetries in source width caused by asymmetrical triangulation for certain directions. The $\boldsymbol{r}_\mathrm{E}$ vector uses the loudspeaker directions $\boldsymbol{\Theta}_l = [\cos\varphi_l \cos\vartheta_l,\, \sin\varphi_l \cos\vartheta_l,\, \sin\vartheta_l]^\mathrm{T}$ superimposed with the squared panning gains depending on the variable panning direction $\boldsymbol{\theta} = [\cos\varphi\cos\vartheta,\, \sin\varphi\cos\vartheta,\, \sin\vartheta]^\mathrm{T}$; φ, ϑ are the azimuth and elevation angle, respectively,

$$\boldsymbol{r}_\mathrm{E}(\boldsymbol{\theta}) = \frac{\sum_{l=1}^{L} g_l^2(\boldsymbol{\theta})\, \boldsymbol{\Theta}_l}{\sum_{l=1}^{L} g_l^2(\boldsymbol{\theta})}. \tag{30}$$

Taking the norm $\|\boldsymbol{r}_\mathrm{E}\|$ of the $\boldsymbol{r}_\mathrm{E}$ vector yields a value expressing the directional extent, i.e., source spread of the reproduced sound. If the panning direction is aligned with a loudspeaker, the loudspeaker will exclusively be activated yielding an $\boldsymbol{r}_\mathrm{E}$ length 1. If the signal is evenly spread all around, the length of the $\boldsymbol{r}_\mathrm{E}$ vector is 0. We re-express the source width estimation of the $\boldsymbol{r}_\mathrm{E}$ vector by the angular spread

$$\sigma_\mathrm{E}(\boldsymbol{\theta}) = 2\arccos(\|r_\mathrm{E}(\boldsymbol{\theta})\|). \tag{31}$$

3.3 Example Case: 4+5+0 Layout

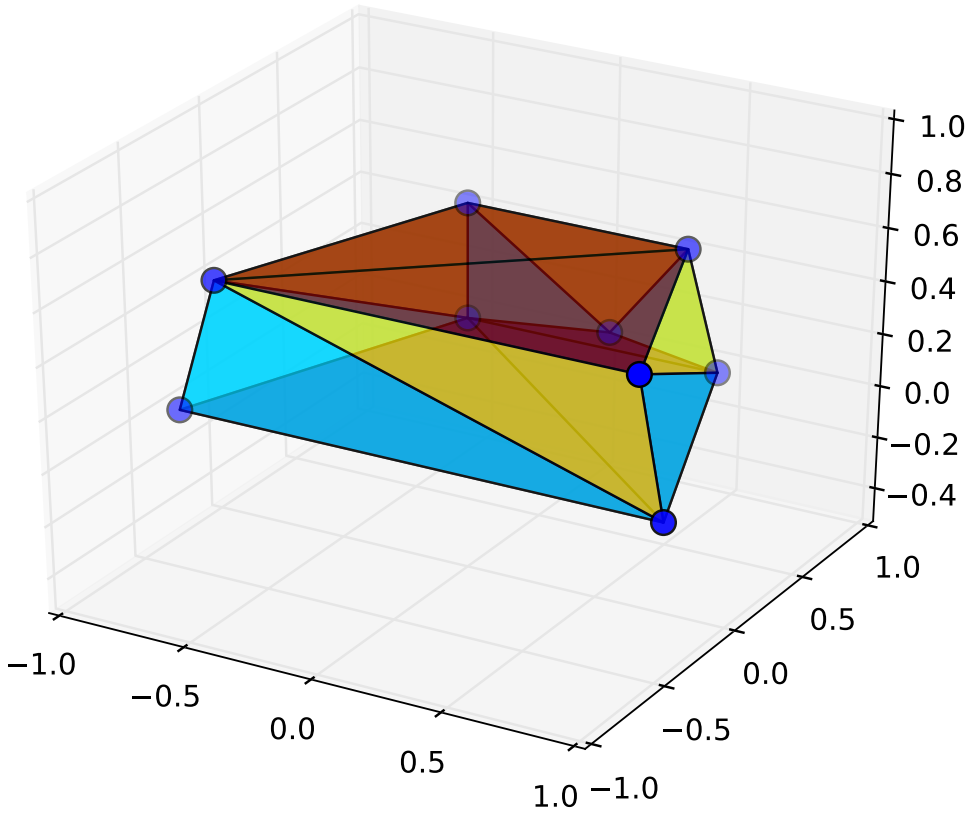


Figure 7 – Triangulation of the 4+5+0 layout using the convex hull.

The analysis of the classical VBAP technique as presented in [8] using the convex hull algorithm [9] for the 4+5+0 setup yields a triangulation shown in Fig. 7.

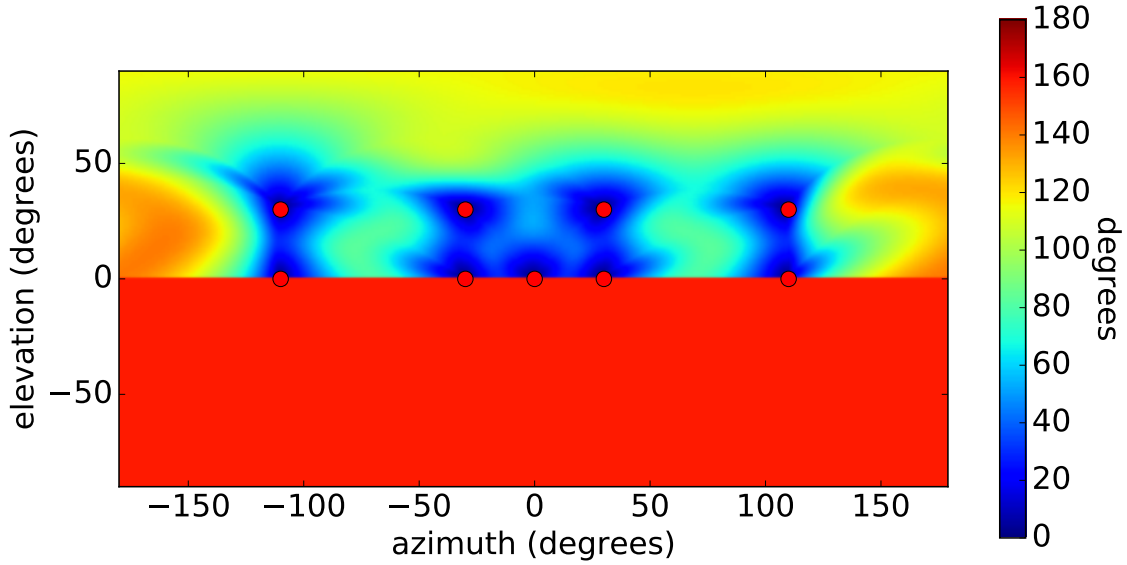


Figure 8 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for classic VBAP with convex hull triangulation on loudspeaker layout 4+5+0.

The spread measure illustrates that the the way the signal is distributed to loudspeakers is left-right asymmetric for panning in the rear and top directions of the setup. Lateral quadrilaterals are left-right symmetric, however, panning will be different between going from, e.g., $\varphi = 30°$, $\vartheta = 0°$ to $\varphi = 110°$, $\vartheta = 30°$ and going from $\varphi = 110°$, $\vartheta = 0°$ to $\varphi = 30°$, $\vartheta = 30°$, which appears counter-intuitive.

It also can be observed that the area with negative elevation yields to a maximum energy spread and abrupt change of the spread at the border because panning values below the horizon are not defined.

3.4 Imaginary Loudspeakers and Downmix

The insertion of imaginary loudspeakers as shown in Tab. 7 leads to a unambiguous and symmetric triangulation and can be observed in Fig. 9.

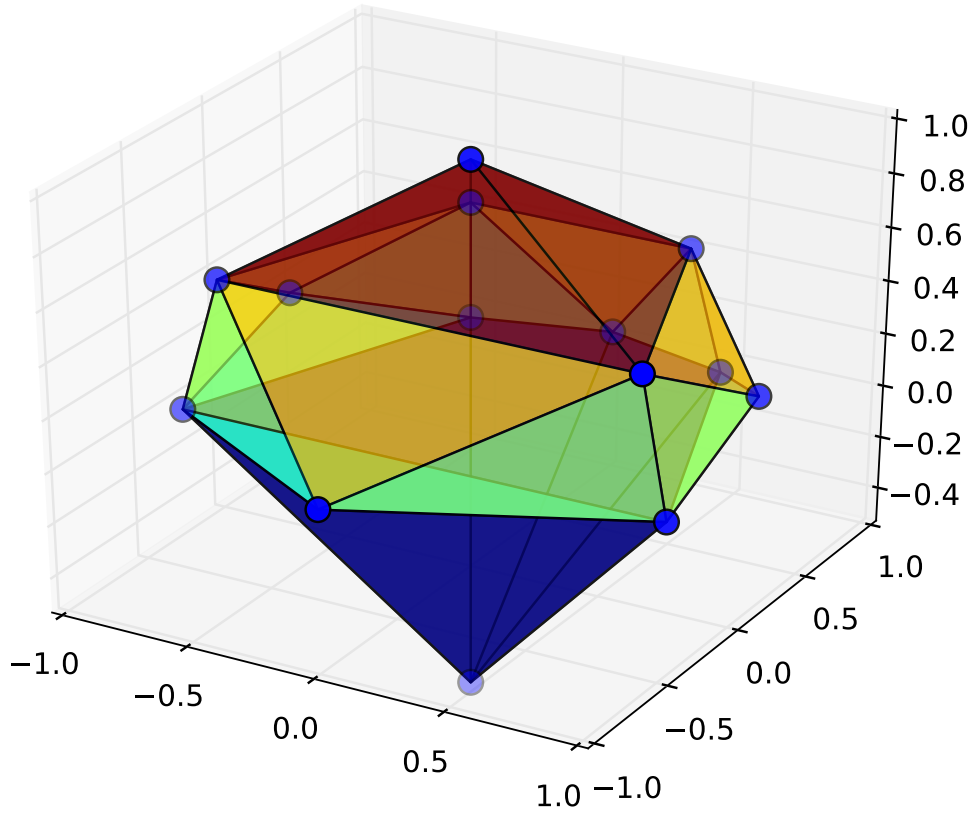


Figure 9 – Triangulation of the 4+5+0 with inserted imaginary loudspeakers layout using the convex hull.

Fig. 9 shows the triangulation of the 4+5+0 layout with additional imaginary loudspeakers in Tab. 7 that help to symmetrize the signal distribution and improve how signals below the horizontal plane are dealt with.

The additional imaginary loudspeakers inserted using the mean vectors of the big loudspeaker quadrilaterals at the bottom, top, left, right, and back are listed in Tab. 7.

lspk.idx	elevation / °	azimuth / °
10	0	−65
11	0	65
12	−70	19
12	70	19
13	180	35

Table 7 – Imaginary loudspeakers (rounded to 5°) inserted at mean positions of the quadrilaterals of the 4+5+0 layout.

Fig. 10 shows the $E(\boldsymbol{\theta})$ measure of the *downmix* of each imaginary loudspeaker to its N neighbours with the gain of $1/\sqrt{N}$. At the direction of the imaginary loudspeaker we get an $E(\boldsymbol{\theta})$ measure of 1, which means that the factor $1/\sqrt{N}$ is optimal there for normalized. For panning directions between the imaginary loudspeaker and its neighbours, downmixing causes an increased $E(\boldsymbol{\theta})$ loudness measure.

An amplitude fluctuation can be entirely avoided by normalization with $1/\sqrt{E(\boldsymbol{\theta})}$. However, this is not an option if the imaginary loudspeaker signals are chosen to be attenuated.

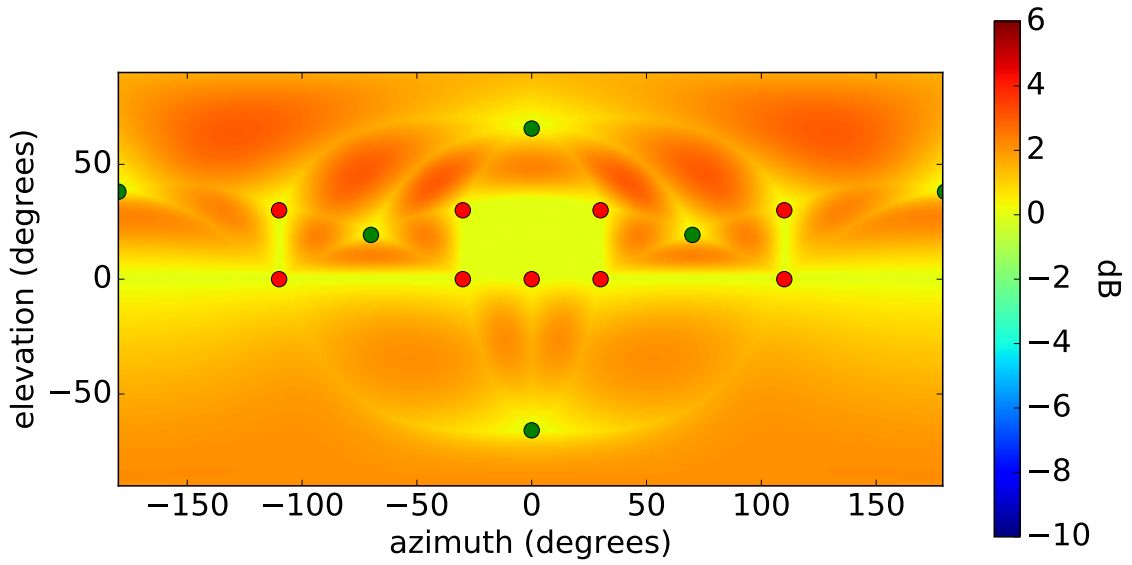


Figure 10 – Loudness measure $E(\boldsymbol{\theta})$ for downmix of imaginary loudspeakers of the 4+5+0 layout using a mixing factor of $1/\sqrt{N}$ to distribute its signal to neighbouring loudspeakers.

After applying the $1/\sqrt{N}$ *downmix* technique the analysis of the measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ results in Fig. 11. The result indicates a symmetric distribution of the loudness measure $E(\boldsymbol{\theta})$ in the area, when compared to the classic VBAP approach in Fig. 8. Within the quadrilaterals

we also observe a more symmetric distribution. The problem in the area with negative elevation is also solved this way.

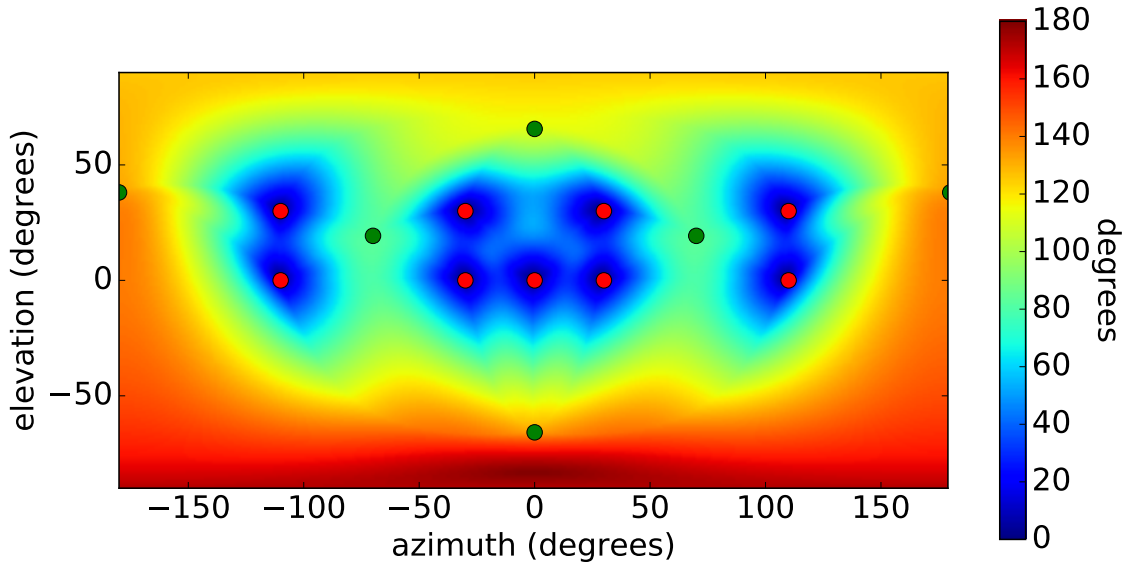


Figure 11 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for the $1/\sqrt{N}$ *downmix* technique on loudspeaker layout 4+5+0.

For the case of the disposal of the imaginary loudspeaker signal, the spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ in Fig. 13 yields directions for which only one physical loudspeaker is activated, because the resulting triangles can contain two imaginary loudspeakers, see top in Fig. 13. Here we get a spread σ_{E} of 0° in these areas, which is not helpful to obtain a smooth panning behaviour. At the bottom and sides, this is not a problem, and smooth panning around the omitted directions is possible.

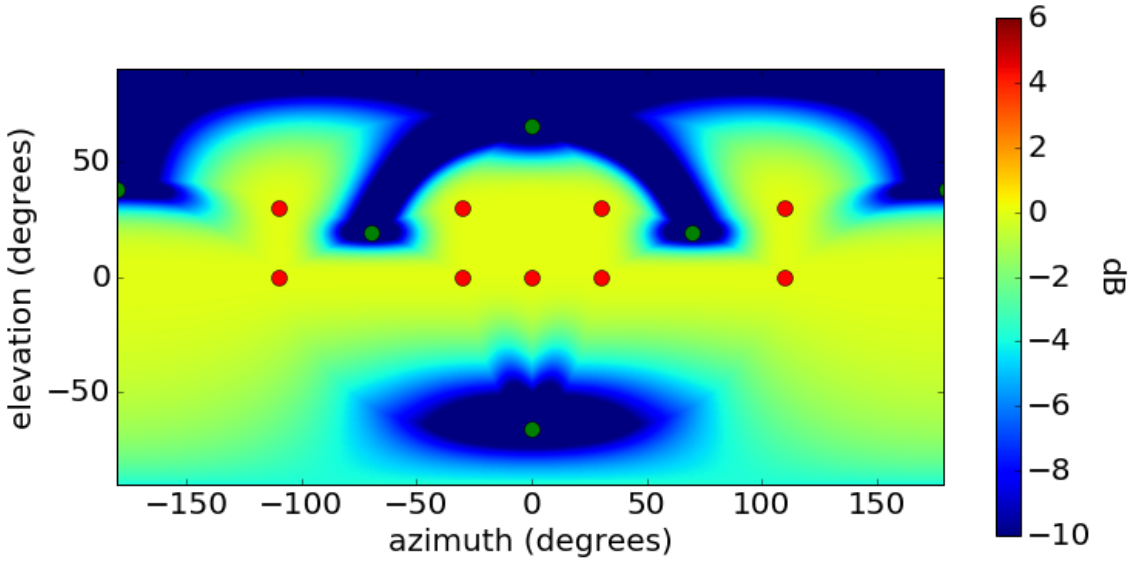


Figure 12 – Loudness measure $E(\boldsymbol{\theta})$ for disposed imaginary loudspeaker (Tab. 7) signals on the 4+5+0 loudspeaker layout

By muting the imaginary loudspeakers entirely (without downmix), we get silence in closely aligned directions, cf. Fig. 12. It can be observed that a panning towards a muted imaginary loudspeaker leads to a fade down in loudness.

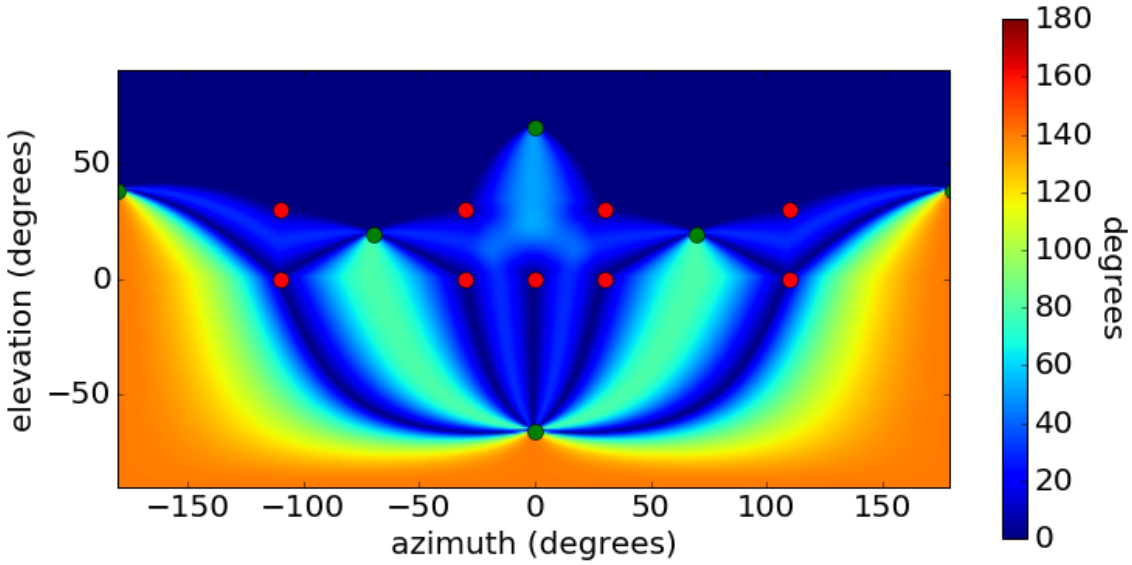


Figure 13 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for disposed imaginary loudspeaker signals on loudspeaker layout 4+5+0.

Clearly, if we do not have real loudspeakers at the bottom we won't be able to produce a virtual source from below the listener. It is therefore thinkable to combine the two

techniques, $1/\sqrt{N}$ *downmix* for $\vartheta \geq 0°$ and the disposal technique for $\vartheta < 0°$. Using this combines approach we can suggest the listener a source moving one floor below the listener during panning in $\vartheta < 0°$. Using a sub-woofer for a panning-independent playback of low-frequency components enhances this effect.

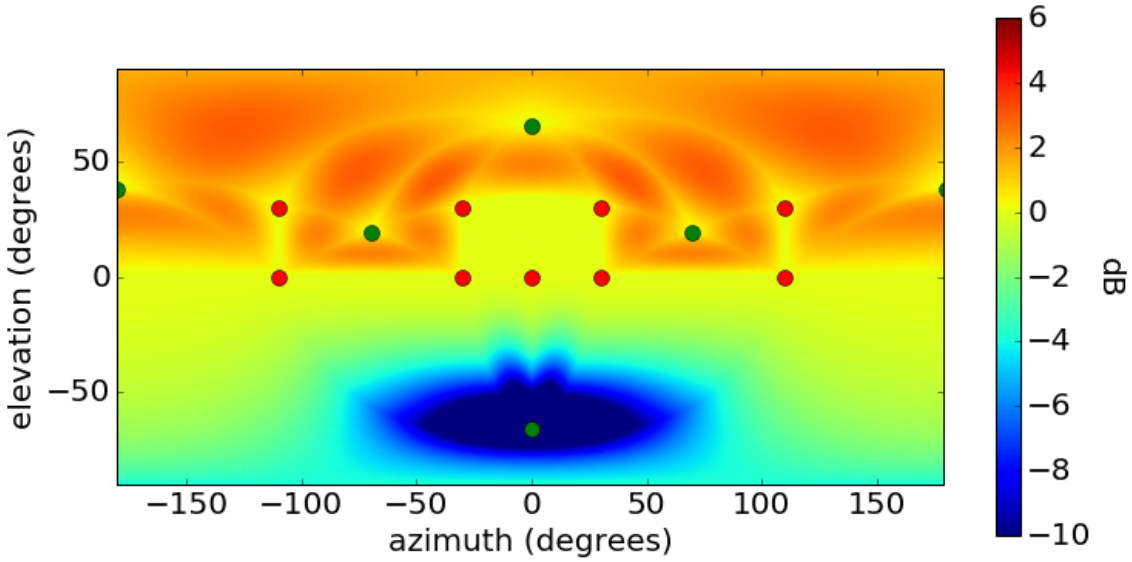


Figure 14 – Loudness measure $E(\boldsymbol{\theta})$ for the combination of disposal and $1/\sqrt{N}$ *downmix* technique on loudspeaker layout 4+5+0.

The $E(\boldsymbol{\theta})$ measure in Fig. 14 shows the downmix above and the fadedown below the horizon, representing the combined approach. Fig. 15 shows the resulting angular spread. We can observe that for $\vartheta \geq 0°$ horizon we now have a left right symmetric spread over the hemisphere. $\vartheta < 0°$ yields two dimensional VBAP between the loudspeaker pairs on the horizontal plane.

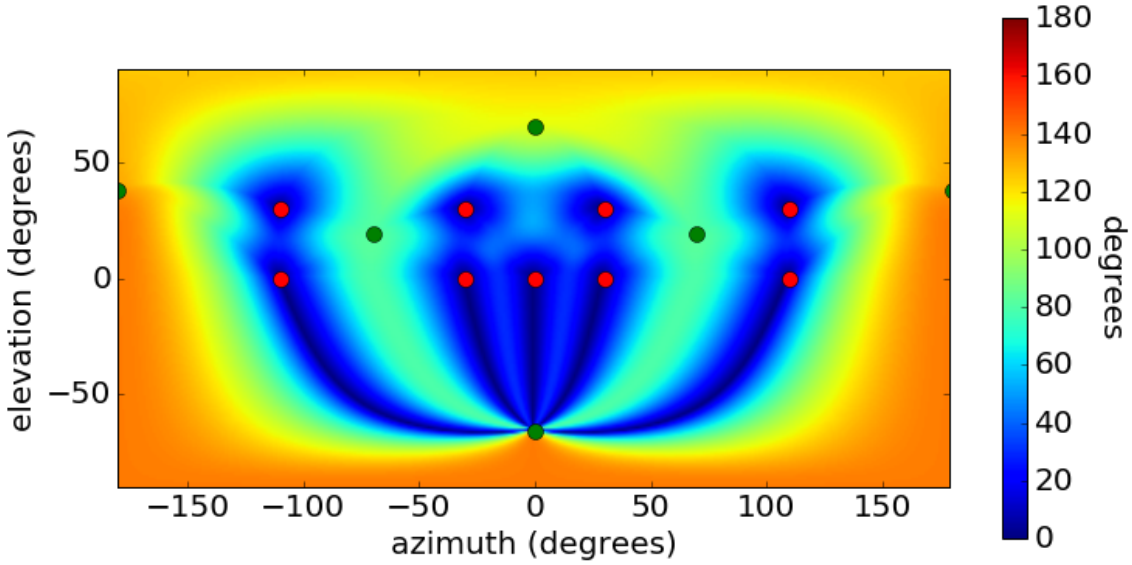


Figure 15 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for the combination of disposal and $1/\sqrt{N}$ *downmix* technique on loudspeaker layout 4+5+0.

The combined approach is a suitable solution for three-dimensional rendering on the upper hemisphere covered with loudspeakers, when a direction-independent subwoofer signal suggests there being a virtual source moving below the floor for rendering on the lower hemisphere.

3.5 Distributing Imaginary Loudspeaker Signals

3.5.1 Disposal of imaginary loudspeaker signal and direction

One option to deal with the imaginary loudspeaker is to dispose the associated signal entirely. This would mute any virtual sound source entirely that is exactly aligned with the direction of the imaginary loudspeaker. By contrast, the signal will never be entirely muted when using panning methods activating a multitude of virtual sources around the panning direction, as with All-Round Ambisonic Decoding [6] or multi-direction amplitude panning [15], for instance.

3.5.2 Downmix with preserved loudness

Another option to deal with the imaginary loudspeaker signal is to distribute (*downmix*) it to the N neighboring physical loudspeakers by the factor $g/\sqrt{N}$, which maintains the squared loudness measure E, Eq. (29), of a signal that exclusively feeds the imaginary

loudspeaker, when the factor $g = 1$. Note that this strategy yields a tendency of a slightly increased E if not only the imaginary loudspeaker but also its neighbors receive a signal.

By using a weights $g < 1$, one can freely adjust how much of the signal should be preserved, and hence how the renderer should deal with signals whose directions are only poorly covered by loudspeakers.

3.5.3 Gain vector normalization

If the signal should not only be fully preserved $g = 1$ but perfectly normalized in loudness $E(\boldsymbol{\theta}) = 1$ for any any panning direction $\boldsymbol{\theta}$, one can normalize the loudspeaker gain vector $\boldsymbol{g}(\boldsymbol{\theta})$. This can be done for channel-based panning of a single virtual source by employing the factor $1/\sqrt{E(\boldsymbol{\theta})}$ as a normalization.
While the same way to normalize is still feasible for gain vectors calculated by Ambisonic panning, Ambisonic decoding or scene-based rendering has no access anymore to the loudspeaker gain vector belonging to every single virtual source. As the AllRAD decoder design technique relies on vector-base amplitude panning, however, it allows to use the above-mentioned gain vector normalization to obtain a decoder design of nearly constant loudness measure $E(\boldsymbol{\theta})$.

3.5.4 Downmix with preserved direction

The pre-requisite to obtain a suitable localisation from a $g/\sqrt{N}$ downmix of an imaginary loudspeaker is to define an adequately chosen direction to insert the imaginary loudspeaker. Accepting the direction of the $\boldsymbol{r}_\mathrm{E}$ vector as a model of the perceived direction, see Eq. (30) and [16], the imaginary loudspeaker needs to be aligned with the mean direction of its N neighboring physical loudspeakers. Fig. 9 shows the convex-hull triangulation of the 4+5+0 layout with additional imaginary loudspeakers that help to symmetrize the signal distribution and solve the signal-loss problem below the horizontal plane. The imaginary loudspeakers above the horizon are inserted at the mean directions of the quadrilaterals, the one below is a mirrored version of the top loudspeaker. The *downmix* of the imaginary loudspeaker to the neighbouring loudspeakers leads to a symmetrical distribution within the quadrilaterals, as Fig. 11 illustrates.

Other similar techniques for such downmixing and loudspeaker insertion can be found in [17] and [14].

3.6 Idealisation of the Loudspeaker Layout

In some cases it might be beneficial to provide the renderer with a more equidistant, idealised set of loudspeaker coordinates than the coordinates of the actual loudspeaker setup. This is, for instance, relevant in Ambisonics, which is based on a strictly isotropic angular resolution. As isotropical angular resolution does not correspond with the typical loudspeaker density increase of common loudspeaker layouts towards the frontal direction, it can be beneficial for the calculation of an Ambisonic decoder to shift the frontal loudspeakers apart to achieve a more equidistant directional coverage by loudspeakers. The goal this elegantly achieved is a more constant loudness, which is typically measured by the the $E(\boldsymbol{\theta})$ measure of Eq. (29).

While such kind of idealization is a suitable means in Ambisonic decoding to fix panning-dependent loudness variation, however, it will cause angular distortion and hereby entail the need to encode virtual sources using a pre-distortion. Therefore idealising the loudspeaker layout is not suitable in productions that aren't specifically designed for one particular loudspeaker layout.

3.7 AllRAD with downmix technique

As we know that the AllRAD approach [6] is a combination of VBAP [8] and classic Ambisonics, the same technique can be used to optimise Ambisonic rendering using the same approaches. We simply calculate the symmetric triangulation with imaginary loudspeakers and sample the Ambisonic directivity pattern at a high number of virtual loudspeakers (about 5200 virtual speakers) and render them down to the real loudspeakers using the VBAP approach. Doing this yields a symmetric Ambisonic decoding matrix that can be used for Ambisonic rendering on non optimal loudspeaker layouts as the 4+5+0 layout. This behaviour can also be analysed using the presented methods.

Fig. 16 shows the spread σ_{E} for AllRAD using the triangulation in Fig. 7. We can observe similarities to Fig. 8.

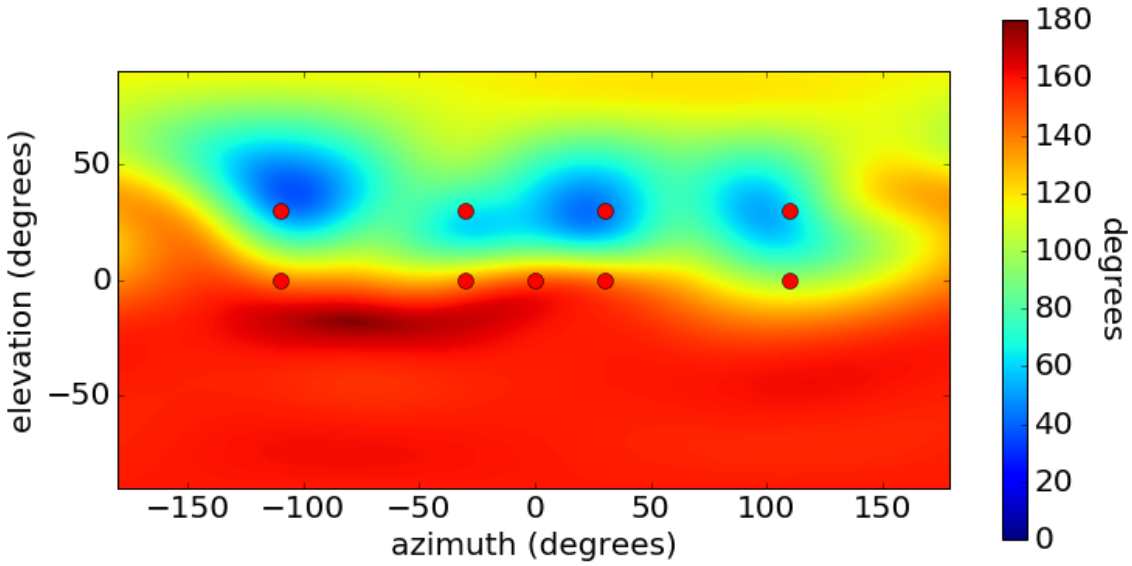


Figure 16 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for the AllRAD with classic VBAP Rendering System on loudspeaker layout 4+5+0.

The rear section is distorted and at points where multiple lines of the triangulation end in one point we can observe a narrower spread. Using triangulation method with imaginary loudspeakers and *downmix* as explained above yields to a symmetric distribution as shown in Fig. 18.

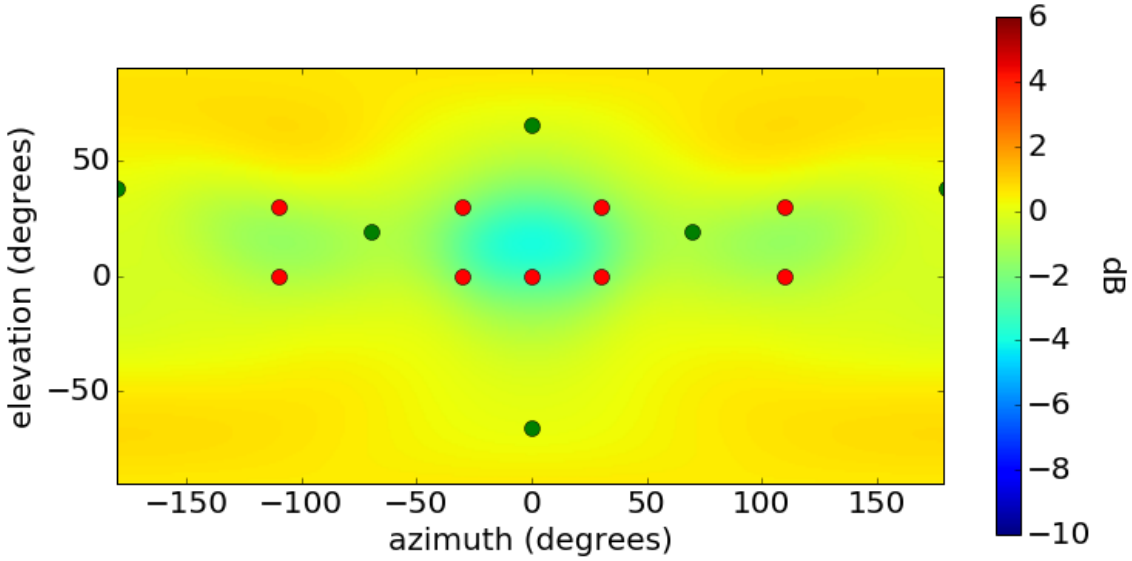


Figure 17 – Loudness measure $E(\boldsymbol{\theta})$ for the Ambisonic 5th Order Rendering System on loudspeaker layout 4+5+0.

Hereby we obtain a decrease of gains in the front section of about 3dB what becomes obvious in Fig. 17.

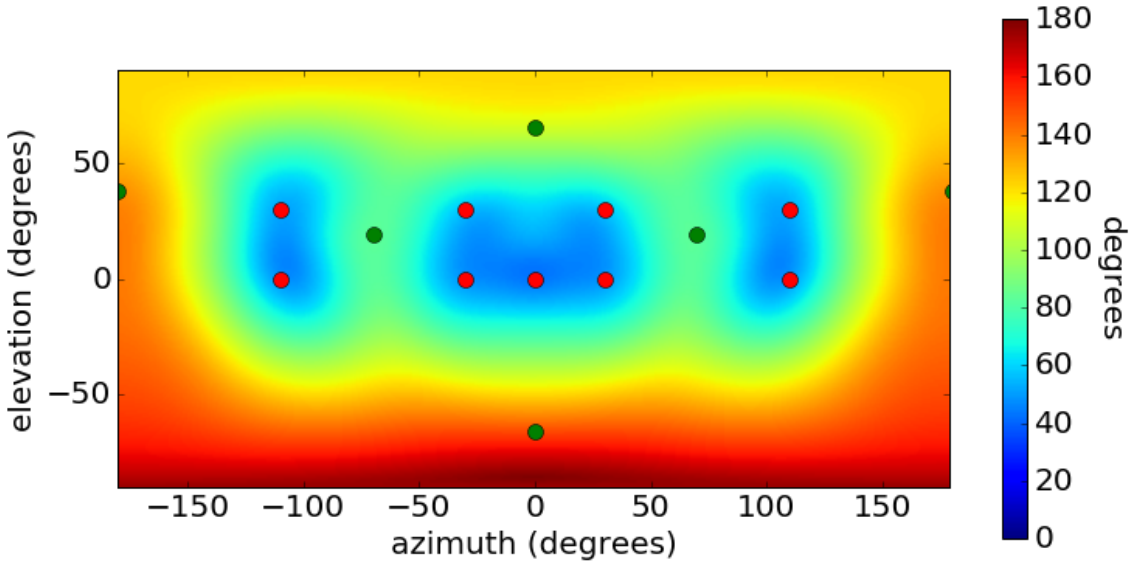


Figure 18 – Spread measure $\sigma_{\mathrm{E}}(\boldsymbol{\theta})$ for the Ambisonic 5th Order Rendering System on loudspeaker layout 4+5+0.

Of course also for Ambisonics, it is thinkable to omit imaginary loudspeakers at the bottom. To equalize the loudness for frontal rendering, it is moreover thinkable to increase the decoder lines corresponding to frontal loudspeakers by a scalar factor.

4 Evaluation

Some text parts and figures of this chapter were published at the International Conference on Spatial Audio 2017 as a scientific paper called “Comparison of spatial audio rendering methods for ITU-R BS.2051 standard loudspeaker layouts” [19].

4.1 Evaluated Rendering Methods

Point source rendering in 3D by amplitude panning is achievable in multiple ways. A selection of common fundamental methods is explained and evaluated, here.

4.1.1 VBAP

The most common and most cited panning method for multiple loudspeakers is Vector Base Amplitude Panning (VBAP), cf. [8]. The three-dimensional repdoduction of a sound source is achieved in terms of an amplitude-panned phantom source within a loudspeaker triangle. A decomposition of all vertices of the loudspeaker layout into triangles is typically achieved by a convex hull algorithm [9]. Within the triangulation of this hull, the triangle is activated that allows rendering of the virtual source with all-positive gains. A full explanation and implementation was already given in section 3.2.

4.1.2 Optimized VBAP with downmix

Amplitude panning methods, such as VBAP, usually rely on the decomposition of the loudspeaker vertices into a convex hull. At positions with small loudspeaker densities, the triangulation can lead to nearly coplanar triangles (typical for the top layer in 2+X+X and 4+X+X). Moreover, angular distances greater than 90° can deliver unstable results (between the rear loudspeakers in X+5+X), or negative gains and numerical instability for angular distances close to 180° or more.

Coplanar triangles can yield perceptual impairment with regard to loudness, location, colouration, and extent. The insertion of imaginary loudspeakers within coplanar triangles can solve this problem. The signals of the imaginary loudspeakers are then downmixed (distributed) to the neighbouring real loudspeakers. An approach for imaginary loudspeaker insertion and downmixing is presented in [13], symbolized as "TMT" later on and was already discussed in section 3.6.

4.1.3 Ambisonics: AllRAD

This approach was already discussed in section 3.7. Here the optimized VBAP with downmix inside the AllRAD decoder is used. In this evaluation, 5th order Ambisonics is implemented with max-rE weighting.

Compared to the optimized VBAP and downmix approach, Ambisonics is of strictly limited resolution. Therefore it distributes the signal to the loudspeakers at a more or less uniform spreading, without the ability of activating single loudspeakers only.

4.1.4 L1 Norm Rendering

Another possible solution for amplitude panning is to solve the L1-norm optimization problem, as outlined in [20]. In general, considering amplitude panning as a global optimization problem, the optimizer generates the gain vector without any explicit loudspeaker selection step. Subset selection is achieved implicitly by the sparsity-enforcing L1-norm optimization under a non-negativity constraint

$$\begin{aligned} &\underset{\mathbf{g}}{\operatorname{argmin}} \, ||\mathbf{g}||_1 \\ \text{subject to } &\mathbf{Lg} = \mathbf{p} \\ &\mathbf{g} \geq 1. \end{aligned} \tag{32}$$

The L1 and VBAP solutions are equal if there are no close-to-coplanar triangles. As a disadvantage of this solution, one has to be aware that L1-norm optimization may not be useful in real-time computation. It is worth to pre-compute a gain matrix with all panning directions and store it in memory.

4.1.5 BBC Rendering System

The performance evaluation in this thesis also involves the BBC Baseline renderer. Despite the internals of this renderer are not disclosed here, its gain matrix as a black-box result was used.

4.1.6 DBAP

Another panning method that is fundamentally different to the rest is Distance-based amplitude panning (DBAP) cf. [21]. DBAP does not assume a specific layout of the loudspeaker array or the position of the listener and is quite flexible.

Basically, DBAP computes the gains by the reciprocal distance of the virtual source to all loudspeakers, up to a given roll-off factor. Here, this roll off is 20 dB.

4.2 Unified Implementation

The gains resulting from the different rendering methods were sampled in terms of the panning direction. One degree resolution was considered to be fine enough to account for the smallest (frontal) JND of directional hearing [22]. By this, we obtain the matrix

$$G_l^{(i)} = \begin{bmatrix} g_l^{(i)}(-180°, 90°) & \dots & g_l^{(i)}(180°, 90°) \\ \vdots & \ddots & \vdots \\ g_l^{(i)}(-180°, 0°) & \dots & g_l^{(i)}(180°, 0°) \end{bmatrix} \tag{33}$$

for each renderer i, containing l^{th} loudspeaker's gain for any pair of panning coordinates (azimuth and elevation). This pre-calculated matrix is stored in memory and used to perform the subsequent evaluation. Moreover each gain is normalized to achieve panning-invariant loudness,

$$g_l^{(i)}(\varphi, \vartheta) \leftarrow \frac{g_l^{(i)}(\varphi, \vartheta)}{\sqrt{\sum_l [g_l^{(i)}(\varphi, \vartheta)]^2}}. \tag{34}$$

4.3 Technical Analysis of Differences

To give a map of the differences between the different renderers, a median gain

$$g_l^{(\text{med})}(\varphi, \vartheta) = \underset{i}{\text{median}}\, g_l^{(i)}(\varphi, \vartheta) \tag{35}$$

is defined for any panning direction and loudspeaker. This allows us to plot the root-mean-square deviation of all N renderers to a *median renderer*

$$\sigma(\varphi, \vartheta) = \frac{1}{N} \sqrt{\sum_{i=1}^{N} \sum_{l=1}^{L} \left[g_l^{(i)}(\varphi, \vartheta) - g_l^{(\text{med})}(\varphi, \vartheta) \right]^2}. \tag{36}$$

For the N renderers under test yields the plot of σ presented in Fig. 19 for layout 4+5+0 and Fig. 20 for layout 3+7+0.

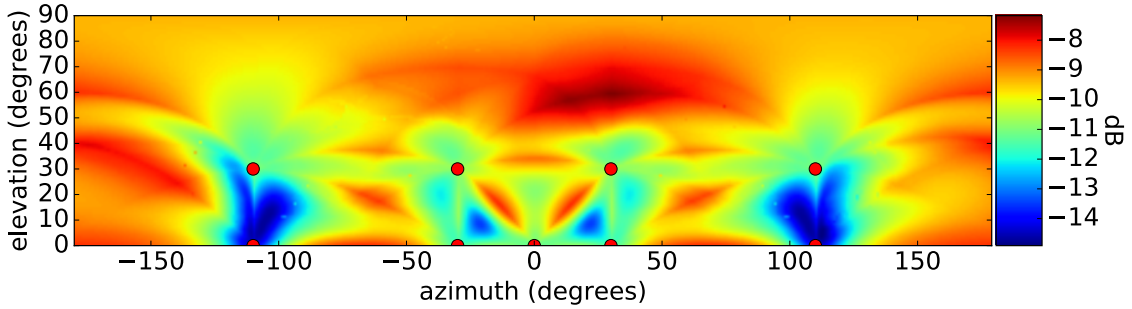


Figure 19 – Renderer differences for layout 4+5+0.

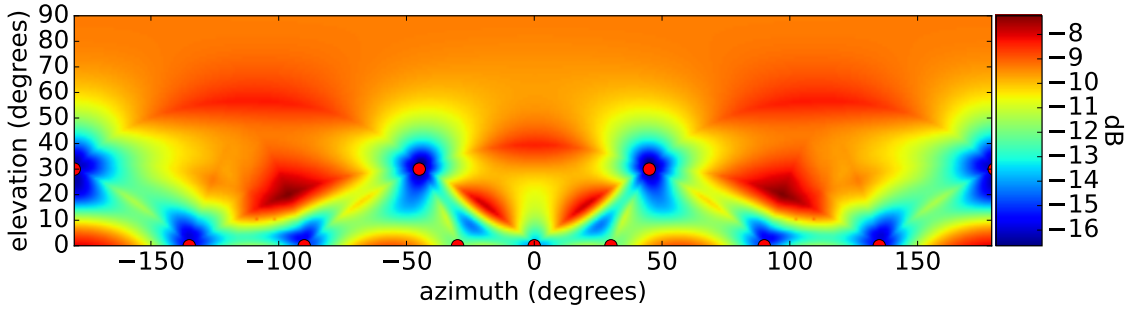


Figure 20 – Renderer differences for layout 3+7+0.

The red and green areas indicate panning directions with the greatest differences, whereas blue areas indicate small differences. The perceptual evaluation focuses on static and dynamic panning through areas of substantial difference.

Moreover, an analysis of differences of every renderer on layout 4+5+0 and 3+7+0 compared to the median renderer was performed and results in single plots per renderer. The plots can be found in section B of the appendix. The differences are squared and normalized whereby 1 represents the maximum difference to the median renderer and 0 represents no difference. It can be observed that the subset selecting renderers apart from VBAP behave pretty similar. Ambi and DBAP perform substantially different. Also for layout 4+5+0 it can be clearly seen that classic VBAP renders asymmetrically.

4.4 rE-Analysis of all Renderers

A full rE-Analysis of all renderers as explained in section 3 has been performed on layout 4+5+0 and 3+7+0 and the resulting plots are attached in section A of the appendix.

4.5 Perceptual Evaluation

Important perceptual qualities such as colouration, extent, and continuity of moving sources are best studied based on listening experiments. The experiments below deal with the 6 above-mentioned rendering methods applied on the 4+5+0 and 3+7+0 loudspeaker layouts.

For the experiment we used 16 Genelec 8020 loudspeakers driven by a RME ADI-648 connected to Behringer Ultragain ADAT interfaces.

All loudspeakers were positioned as close to the standart layouts as possible on the rail system that can be seen in Fig. 21, gain and time aligned to 1 sample precision and equalized with parametric filters (left and right height speakers required to do so).

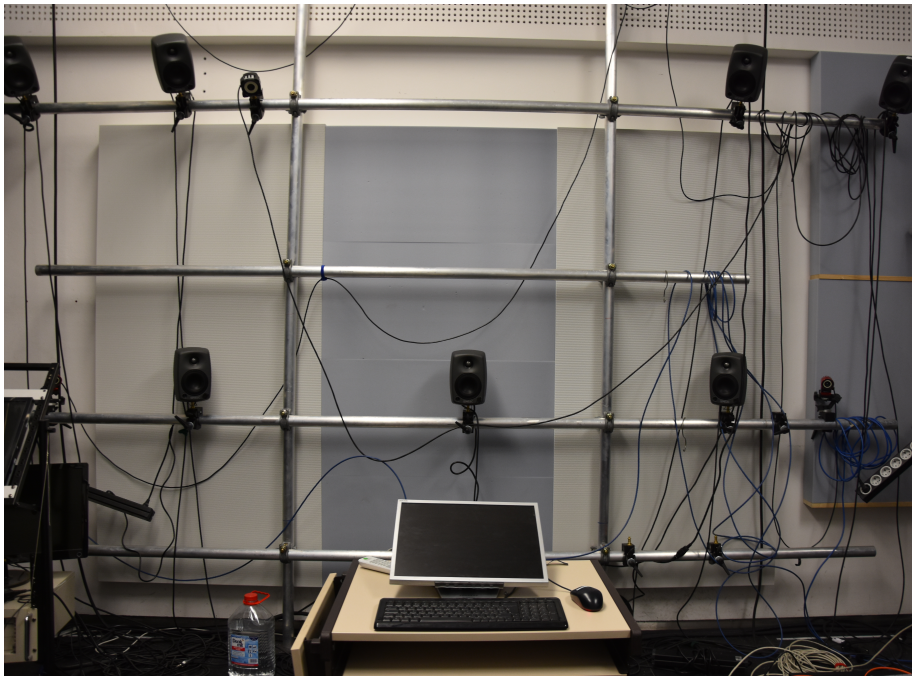


Figure 21 – Listening experiment setup at the "Experimentalstudio" at IEM.

The participants were sitting in the sweet spot and were surrounded by an acoustically transparent curtain to avoid bias by seeing the loudspeaker arrays (Fig.22). Listeners were sitting in front of small desk where a PC mouse and monitor were positioned. The user interface contains 12 sliders and buttons to select and rate the renderers and layouts (6 renderers $\times$ 2 layouts) and can be seen in Fig. 23.

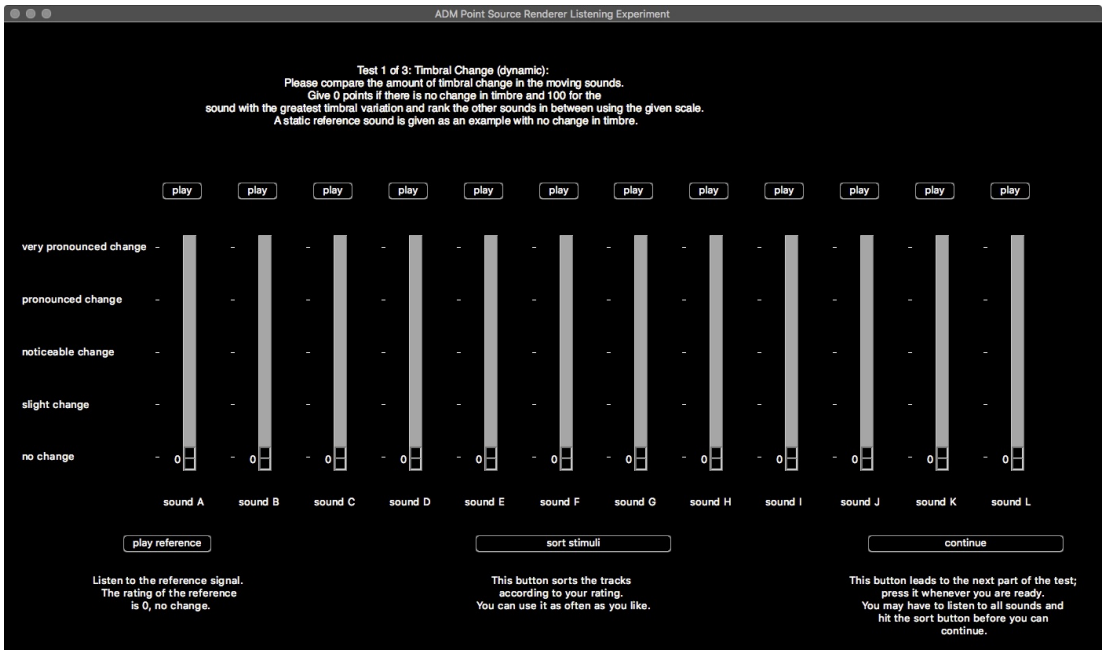


Figure 22 – GUI of listening experiments.

Colours of the GUI were chosen black with white letters to produce as little light distraction as possible. For each trial and participant, the 12 stimuli were presented in an individual random order. A participant could switch, sort, and rate the presented stimuli at will to give satisfactory ratings.

Figure 23 – Listening experiment setup at the "Experimentalstudio" at IEM.

Three experiments were defined (extent, colouration, continuity), each containing three directions/trajectories. One experiment took about 25 min; most participants took breaks between them. Three trials per experiment were performed with one repetition for each experiment. The stimuli (dynamic/static) are live-rendered by a pure-data (Pd) patch (shown in Fig. 24) reading and interpolating the pre-calculated gain matrices (interpolation 10ms). This unified way of rendering keeps the computational load independent of renderer and layout.

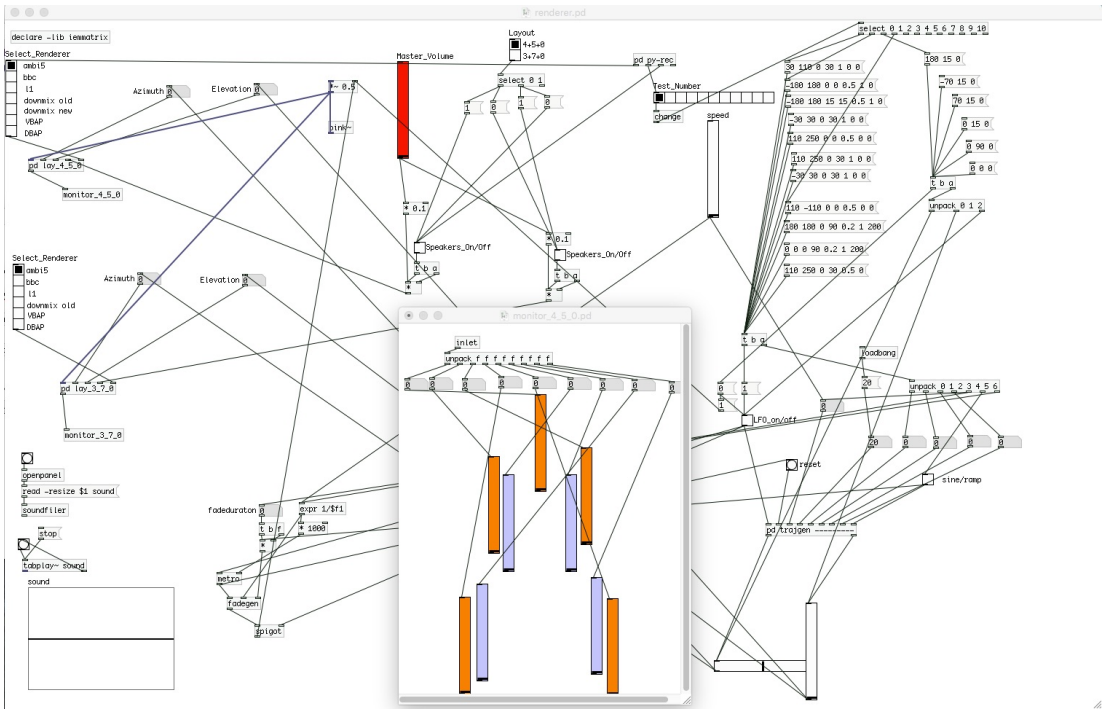


Figure 24 – Pure Data implementation of the listening experiment.

18 experiences listeners took part, most of whom being students at the university and employees of IEM. Two inexperienced listeners also took part, whereby the comparison of repeated experiments shows a higher spread compared to the experienced listeners.

4.6 Extent

A static pink noise sound is positioned in front ($\varphi = 0°$ and $\vartheta = 0°$), rear ($\varphi = 180°$ and $\vartheta = 15°$), and the top ($\varphi = 0°$ and $\vartheta = 90°$) direction.

Participants were asked to compare the perceived size of the sound sources and to give $+100$ points if a source appears as widest source, -100 points if a source appears as narrowest source and to rank other sounds proportionally along the given scale.

4.6.1 Extent Ratings

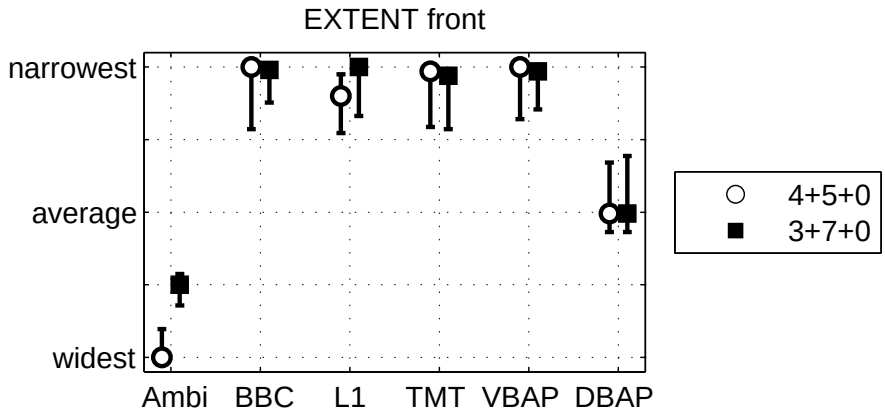


Figure 25 – Extent (medians and 95%-confidence intervals) for panning to front.

The plot in Fig. 25 shows the extent rating for panning to the front. Ambi obviously produces the widest perceived extent ($p << 0.001$) on layout 4+5+0 and a bit narrower one for the 3+7+0 layout. Renderers 2-5 have been perceived narrowest for both layouts and are not distinguishable ($p \geq 0.64$). In between we find the DBAP rendering method with no differences between the layouts.

The VBAP-based renders are not distinguishable ($p \geq 0.64$). DABP lies between Ambi and the VBAP-based renderers. The same relations hold for the 3+7+0 layout. Using Ambi, width decreases significantly for the larger 3+7+0 in comparison to the smaller 4+5+0 layout ($p = 0.0013$). The width of all other renderers is not depending on the loudspeaker layout ($p > 0.60$).

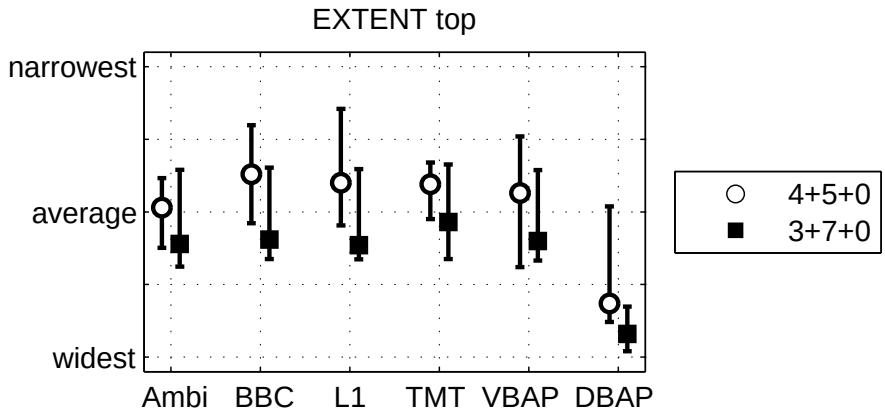


Figure 26 – Extent (medians and 95%-confidence intervals) for panning to top.

The plot in Fig. 26 shows the extent rating for panning to the top. DBAP produces the widest perceived sources for the 3+7+0 layout ($p = 0.0002$). Renderers 1-5 produce narrower sources than DBAP for 3+7+0. For the 4+5+0 layout, the extent was narrower than for the 3+7+0 layout for renderers 1-5 but not significantly different. For the 4+5+0 layout the renderers have no significant difference. All values show that $p > 0.0512$. There is no significant difference between the layouts ($p > 0.0757$).

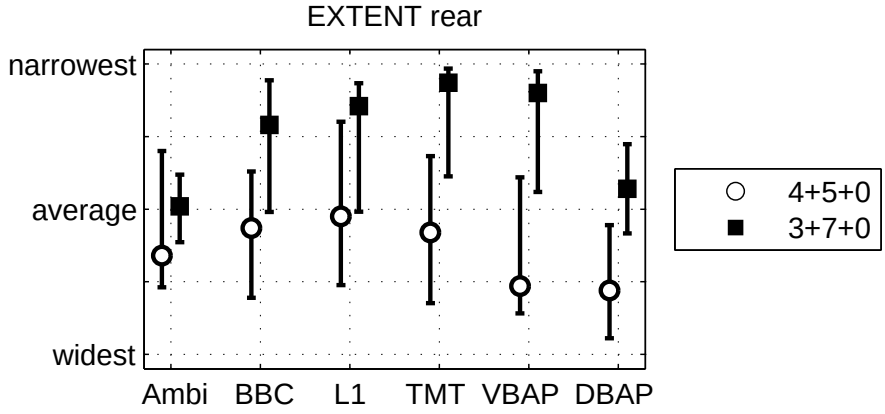


Figure 27 – Extent (medians and 95%-confidence intervals) for panning to rear.

For panning to the rear, the plot in Fig. 27 shows the extent ratings. It is remarkable that here the variance is much larger than for other directions. VBAP and DBAP produce the largest extent on the 4+5+0 layout. Renderers 2-5 produce the narrowest sources on the 3+7+0 layout. In between, we find the Ambi renderer with no significant differences between the layouts. The BBC, TMT and VBAP renderers produce a largely different

extent on both layouts. On the 4+5+0 layout, there is only a significant difference between DBAP and the L1 approach ($p = 0.0418$)

For the 3+7+0 layout, the significance analysis shows that Ambi differs significantly to all renderers apart from DBAP ($p > 0.4082$). DBAP differs significantly only from the TMT approach ($p = 0.0436$).

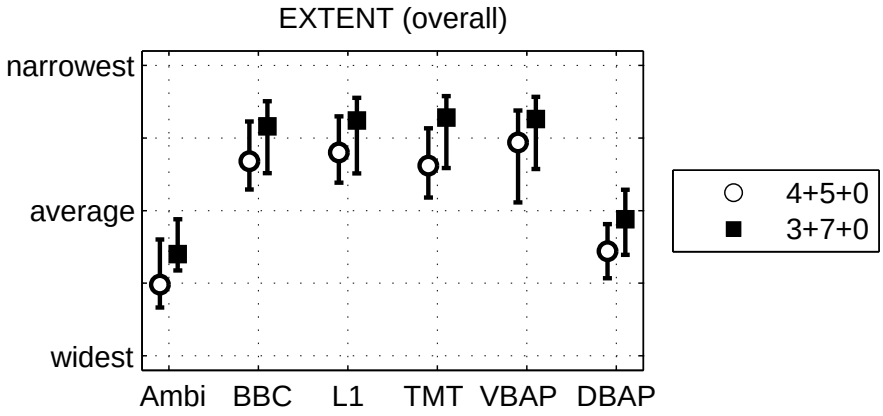


Figure 28 – Extent (medians and 95%-confidence intervals) for panning to front, top, and rear altogether.

In the overall comparison, the plot in Fig. 28, Ambi produces the widest extent followed by DBAP, at no significant difference to each other ($p =$). Renderers 2-5 differ significantly from Ambi ($p = 0.0001$) and DBAP ($p = 0.0021$) but are of similar extent among each other ($p > 0.4247$). Tendentially, all renderers produce narrower sources on the 3+7+0 layout but not significantly so. There is no significant difference between the layouts ($p > 0.0992$).

4.7 Colouration

The colouration experiment employed a pink noise sound moving along the trajectories shown in Fig. 29. The orange points represent the front trajectory, green represents the side trajectory, and blue the circular trajectory. The back-and-forth and circular motion trajectories took 2s.

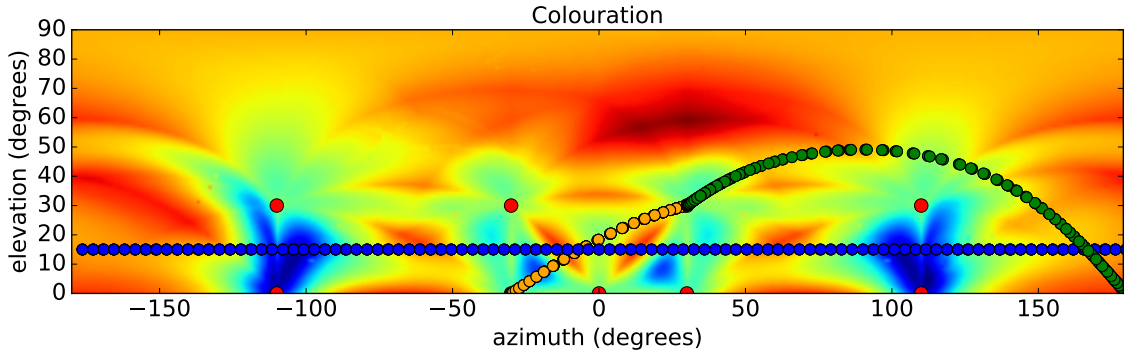


Figure 29 – Trajectories in front (orange), right (green) and circular (blue) colouration listening experiment.

Participants were asked to compare the amount of timbral fluctuation in the moving sounds, to give 0 points if there is no change in timbre, 100 points for the sound with the greatest timbral variation, and rank the other sounds proportionally along the given scale. A static reference sound was given as an example of neutral colouration.

4.7.1 Colouration ratings

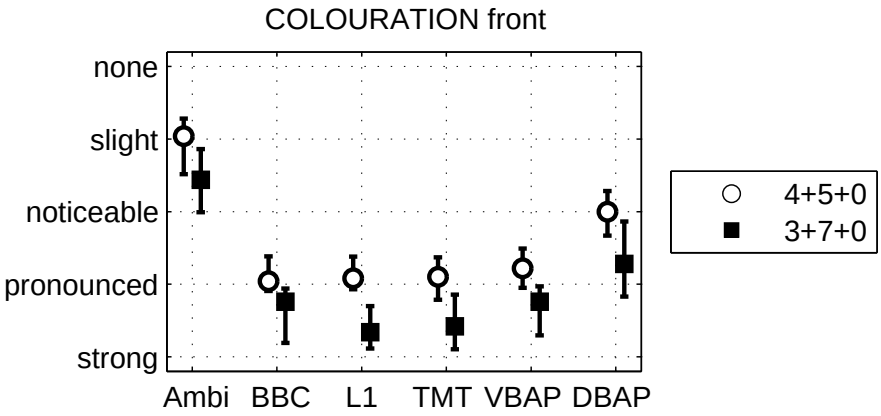


Figure 30 – Colouration (medians and 95%-confidence intervals) for frontal trajectory.

The plot in Fig. 30 shows the colouration ratings for the front trajectory. Ambi rendering distinctly performs best and is significantly different to all renderer on layout 4+5+0 ($p = 0.0044$ to DBAP and $p < 0.001$ for the rest). DBAP is also significantly different to the rest as $p < 0.0001$. The renderers 2-5 seem to exhibit quite pronounced colouration fluctuations for the 3+7+0 layout and distinctly less on the 4+5+0 layout. For the front trajectory, differences between the loudspeaker layouts are significant for all loudspeaker subset selecting approaches ($p < 0.0254$) but not for Ambi and DBAP ($p > 0.0638$), with the 4+5+0 layout introducing less colouration all renderers except Ambi.

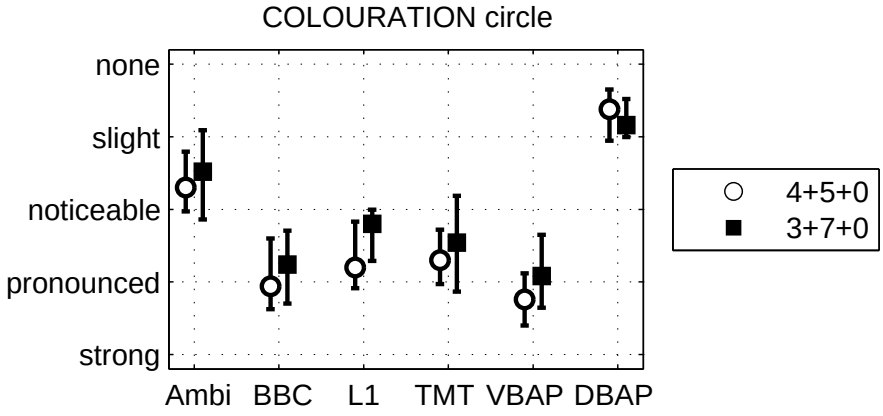


Figure 31 – Colouration (medians and 95%-confidence intervals) for frontal circular trajectory.

The plot on Fig. 31 represents the results of the colouration experiment for the circular trajectory. One can observe that DBAP performs best followed by Ambi. Ambi and DBAP differ significantly from the rest and also from each other ($p < 0.02$). The other renderers perform worse. In general, results show that there is less colouration fluctuation on layout 4+5+0 compared to 3+7+0 but not significantly as $p > 0.1672$ for all renderers.

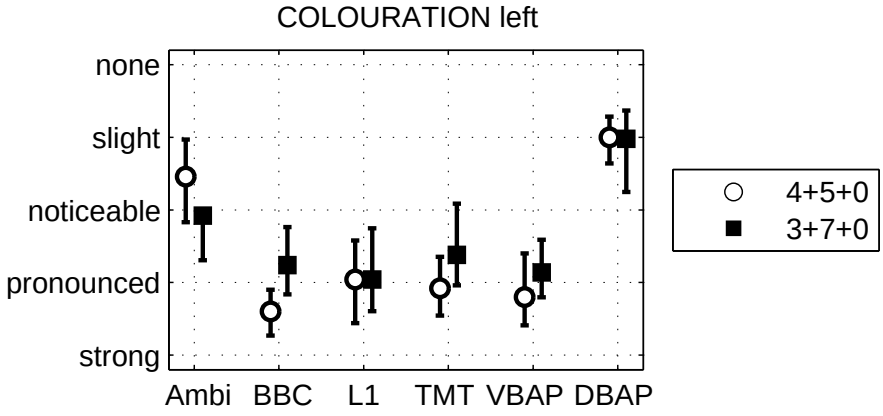


Figure 32 – Colouration (medians and 95%-confidence intervals) for frontal left trajectory.

For the right side trajectory, ratings are shown in the plot in Fig. 32. DBAP produces the least, but slight colouration fluctuations on both layouts ($p < 0.01$). Ambi is only significantly different from DBAP on layout 3+7+0 ($p = 0.002$). For layout 4+5+0 Ambi is significantly differents to all renderers apart from DBAP ($p = 0.106$ for DBAP and $p < 0.2539$) for the rest. Ambi introduces more noticeable changes for 4+5+0 and definitely noticeable ones for the 3+7+0 layout. Again, the other renderers perform similarly and yield pronounced colouration fluctuations. The BBC rendering method yields very pronounced fluctuations for the 4+5+0 layout and pronounced ones for 3+7+0. The colouration of renderers BBC and TMT significantly different for the two loudspeaker layouts ($p = 0.011$ for BBC and $p = 0.048$ for TMT). The other ones have no significant difference on the loudspeaker layouts ($p > 0.05$).

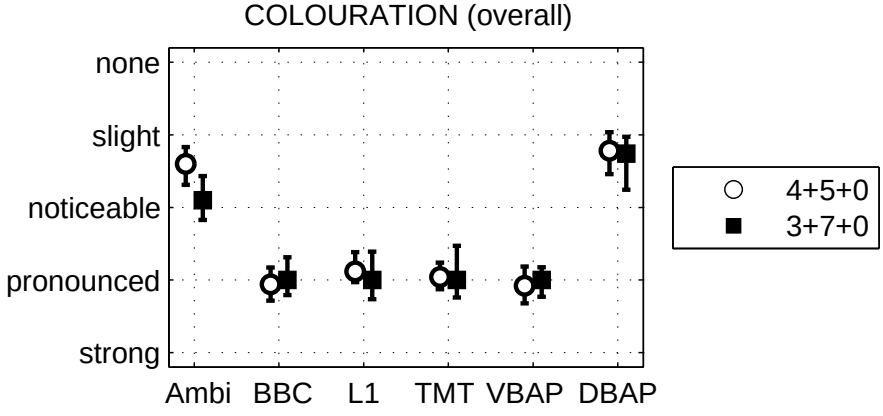


Figure 33 – Colouration (medians and 95%-confidence intervals) for vertical trajectoriy.

The plot on Fig. 33 shows the overall colouration rating. In total the layouts do not differ significantly as $p > 0.06$ for all renderers. Renderers 2-5 yield have pronounced colouration during rendering. DBAP and Ambi introduce less pronounced colouration that range between slight (Ambi on 4+5+0, DBAP on both layouts) and noticeable (Ambi on 3+7+0). Ambi and DBAP differ significantly from the rest ($p < 0.001$) but not within each other ($p > 0.125$ on both layouts).

4.8 Continuity

To investigate the performance on continuity, a pink noise sound is moving along two trajectories upwards from the horizon and along a circle around the listener as shown in Fig. 34. The trajectories took about 4s; front and rear trajectories were automatically re-started.

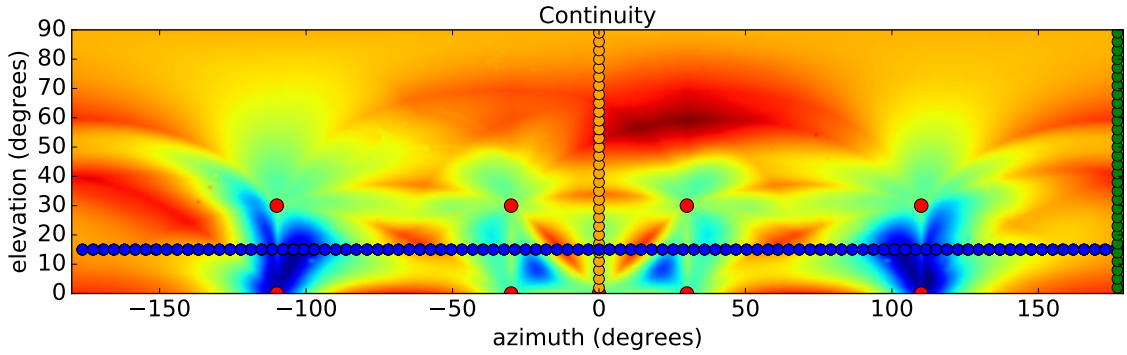


Figure 34 – Trajectories in front (orange), rear (green) and circular (blue) continuity listening experiment.

For one trajectory per multi-stimulus trial, participants were asked to rate if the reproduced source appears to move continuously. They were asked to compare the amount of continuity between the moving sound sources and give 100 points if the source does appear to move perfectly continuous and smooth, 0 points for the sources with the worst continuous (unsteady) movement and rank other sounds proportionally along the given scale.

4.8.1 Continuity ratings

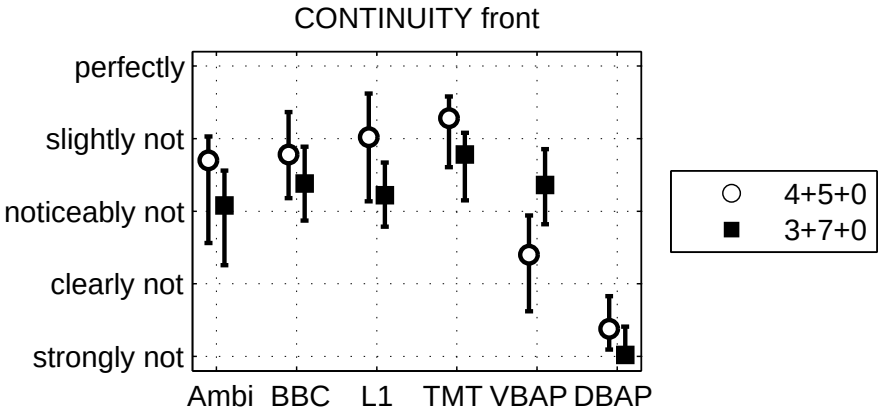


Figure 35 – Continuity (medians and 95%-confidence intervals) for frontal trajectory.

The plot in Fig. 35 shows the continuity ratings for the front trajectory. DBAP yields the motion perceived as least (strongly not) continuous for both layouts. The coplanar top triangles of 4+5+0 cause the clear incontinuity of VBAP. With this exception, renderers 1-5 produce noticeable unsteadiness on the 3+7+0 layouts and slight unsteadiness on the 4+5+0 layout, however without significant distinction. It seems that except VBAP, all renderers perform better on the 4+5+0 layout for the front trajectory. Here DBAP differs significantly from the rest with $p < 0.001$ on both layouts. Very interesting is that VBAP differs significantly from all the other renderers on layout 4+5+0 ($p < 0.01$). This clearly shows that without imaginary loudspeaker insertion and downmix VBAP, leads to audible differences. For layout 3+7+0 only DBAP differs significantly from the rest ($p < 0.001$ for DBAP and $p > 0.3866$ compared to the rest).

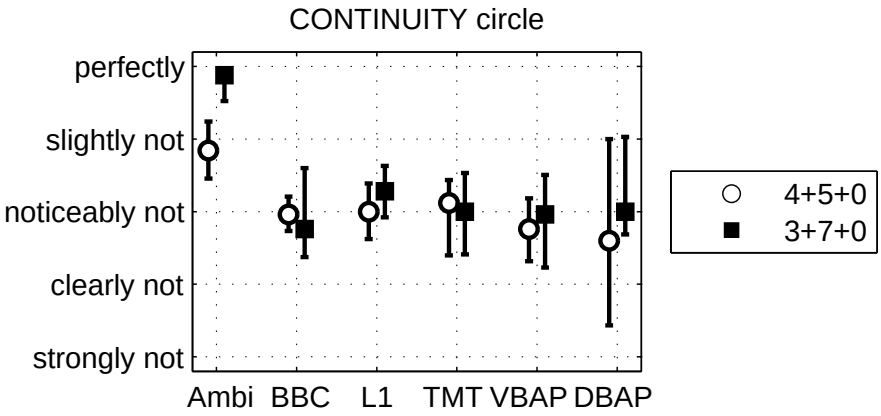


Figure 36 – Continuity (medians and 95%-confidence intervals) for circular trajectory.

The plot in Fig. 36 shows the continuity ratings for the circular trajectory. Ambi produces a perfectly continuous circular trajectory on the 3+7+0 layout and a significantly different slightly unsteady one for 4+5+0 ($p = 0.001$); moreover the continuity of Ambi for circular motion outperforms the other renderers significantly ($p < 0.001$). Renderers 2-6 produce noticeable incontinuous motion for both layouts, with ratings for DBAP largely spread for 4+5+0: participants reported the difficulty of rating the continuity of motion that does not seem to move at all. For renderers 2-6 there is no difference between the renderers and also not inbetween the layouts ($p > 0.1$).

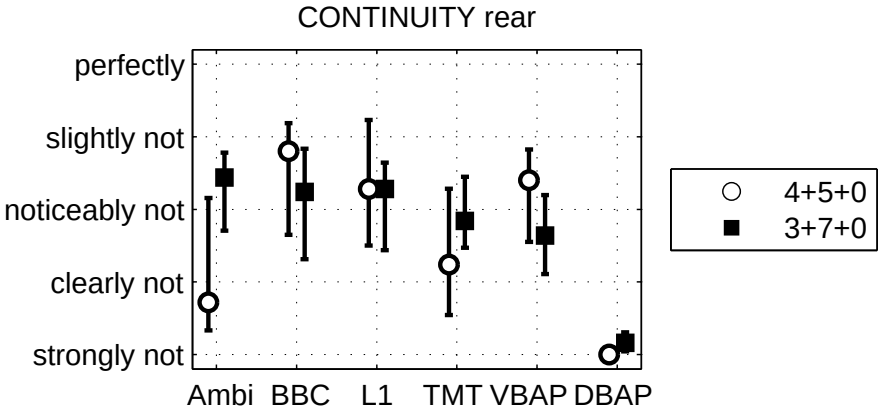


Figure 37 – Continuity (medians and 95%-confidence intervals) for rear trajectory.

The plot in Fig. 37 shows continuity for the rear trajectory. Again, DBAP produces pronounced strongly incontinuous motion on both layouts with a significant difference to the other renderers ($p < 0.001$). DBAP is followed by Ambi on the 4+5+0 layout. Ambi and TMT have no significant difference on layout 4+5+0 ($p = 0.53$) probably caused by the same imaginary loudspeaker insertion and downmix approach. Also BBC and TMT renderers introduce significant differences on the 4+5+0 layout ($p = 0.41$). Renderers 2-5 produce noticeable incontinuous motion on both layouts, and ratings for renderers 1-5 are not significantly different for layout 3+7+0 ($p > 0.165$). DBAP and Ambi introduce significant differences between the two layouts ($p < 0.02$).

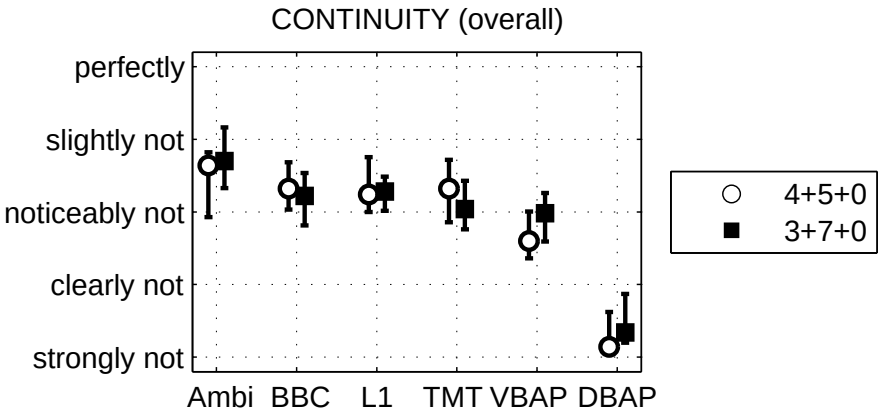


Figure 38 – Continuity (medians and 95%-confidence intervals) for vertical trajectory.

Overall, DBAP introduces the largest incontinuity in trajectories, see fourth plot in Fig. 38. Ambi seems to perform best, but is overall not significantly different to renderers 2-4 on both layouts. The VBAP renderer tends to perform worse on the 4+5+0 layout. Ambi is significantly different to VBAP and DBAP for layout 4+5+0 ($p < 0.049$). DBAP differs significantly to the rest ($p < 0.001$) on both layouts. However, in between the layouts the renderers appear to be not significantly different for the continuity task ($p > 0.08$).

4.9 Summary

VBAP, optimized VBAP with imaginary loudspeaker insertion and downmix (TMT), the L1 approach, and the BBC renderer yield the narrowest extent for rendered sounds and tend to behave similarly. They are related to the same kind of solution implying the use of the tightest-possible loudspeaker subset. Such loudspeaker-subset-related methods yields the narrowest extent compared to Ambisonics and DBAP as a benefit.

By contrast, Ambisonics yields slightly better continuity for moving sounds compared to the rest, and it is distinctly outperforms other renderers when presenting a circular motion, by nature, as the assumption of a limited angular resolution yields a relatively constant extent. The problem of coplanar triangles of VBAP are solved by the BBC, TMT, and L1 methods (see vertical frontal trajectory).

DBAP produces the least colouration fluctuation with moving sounds in the trajectories of the experiment, however, it also maps continuous motion the worst. In colouration fluctuation, it is only competing with Ambisonics, while the loudspeaker-subset-related methods introduce pronounced colouration changes.

5 Conclusion

In this thesis an Ambisonics based approach to deal with all rendering parameters of the ITU Audio Definition Model was proposed. It includes a basic "point-source-panning" to render sound sources on the unit sphere and also some extended rendering functionalities for the parameters source width, diffusion, rotation and distance coding for interior and exterior behaviour. The implementation includes an optimized version of the AllRAD approach whereby symmetry problems caused by the triangulation of the underlying VBAP technique that appear on layouts like the ITU-R BS.2051 4+5+0 layout method are solved by imaginary loudspeaker insertion and downmix to real loudspeakers. This results in optimal rendering.

Another part of this thesis is the evaluation of several rendering systems (VBAP, VBAP with imaginary loudspeaker insertion and downmix, Ambisonics with AllRAD and imaginary loudspeaker insertion and downmix, L1-norm rendering, the BBC rendering system and DBAP) in regard to extent, colouration and smoothness of moving sources. The evaluation first was performed mathematically using a proposed technical analysis that has been presented in this thesis.

This technical analysis was used as a base for perceptual evaluation. The goal was to find positions where differences within the rendering systems could be expected and afterwards evaluated perceptually. Moreover, a collection of methods to efficiently analyse surround with height point source rendering methods on different loudspeaker layouts could be presented. With the settings and trajectories for the listening experiments, relatively clear and effective statements can be made about which features (extent, colouration, continuity) are best represented by which rendering method. It could be shown that the imaginary loudspeaker insertion and downmix is audible by human listeners and also that loudspeaker subset selecting rendering methods behave pretty similar within each other (apart from classic VBAP) compared to Ambisonics and DBAP.

In this way, it could be clarified and shown that the distinct advantages and disadvantages that one gets with the most common amplitude-panning methods for surround with height, on the example of the ITU 4+5+0 and 3+7+0 loudspeaker layouts.

References

[1] Radiocommunication Sector of ITU, "Recommendation ITU-R BS.2076-0 Audio Definition Model," June 2015.

[2] J. Herre, J. Hilpert, A. Kuntz and J. Plogsties, "MPEG-H Audio - The New Standard for Universal Spatial / 3D Audio Coding," 2014.

[3] T. Nixon, A. Bonney, F. Melchior, "A Reference Listening Room for 3D Audio Research," August 2015.

[4] Radiocommunication Sector of ITU, "Recommendation ITU-R BS.2051-0 Advanced sound system for programme production," February 2014.

[5] C. Nachbar, F. Zotter, E. Deleflie, A. Sontacchi, "AMBIX - A Suggested Ambisonics Format," June 2011.

[6] F. Zotter, M. Frank, "All-Round Ambisonic Panning and Decoding," October 2012.

[7] J. Daniel, "ReprÃ©sentation de champs acoustiques, application Ã la transmission et Ã la reproduction de scÃ©nes sonores complexes dans un contexte multimÃ©dia," 2001.

[8] V. Pulkki, "Virtual Sound Source Positioning Using Vector Base Amplitude Panning," June 1997.

[9] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa, "The Quickhull Algorithm for Convex Hulls," 1996.

[10] F. Zotter, M. Frank, "Efficient Phantom Source Widening," October 2012.

[11] F. Zotter, *Analysis and Synthesis of Sound-Radiation with Spherical Arrays*. PhD thesis, December 2009.

[12] U. Zölzer, *DAFX: Digital Audio Effects, Second Edition*. 2014.

[13] M. Romanov, M. Frank, F. Zotter, T. Nixon, "Manipulations improving amplitude panning on small standard loudspeaker arrangements for surround with height," November 2016.

[14] C. Borss, "A Polygon-Based Panning Method for 3D Loudspeaker Setups," 2014.

[15] V. Pulkki, "Uniform spreading of amplitude panned virtual sources," 1999.

[16] M. Frank, "Phantom Sources using Multiple Loudspeakers in the Horizontal Plane," June 2013.

[17] Fraunhofer-Gesellschaft, "Apparatus and method for generating a plurality of audio channels," 2015.

[18] M. A. Gerzon, “General Metatheory of Auditory Localization,” 19925.

[19] M. Romanov, M. Frank, F. Zotter, C. Pike, “Comparison of spatial audio rendering methods for ITU-R BS.2051 standard loudspeaker layouts,” September 2017.

[20] A. Franck, W. Wang, F.M. Fazi, “Sparse, L1-Optimal Multi-Loudspeaker Panning and its Relation to Vector Base Amplitude Panning,” February 2017.

[21] T. Lossius, P. Baltazar, T. de la Hogue, “DBAP - Distance-based amplitude panning,” February 2009.

[22] R. David, D.R. Perrott and K. Saberi, “Minimum audible angle thresholds for sources varying in both elevation and azimuth,” 1990.

A rE-Analysis of all Renderers

A.1 4+5+0 Layout

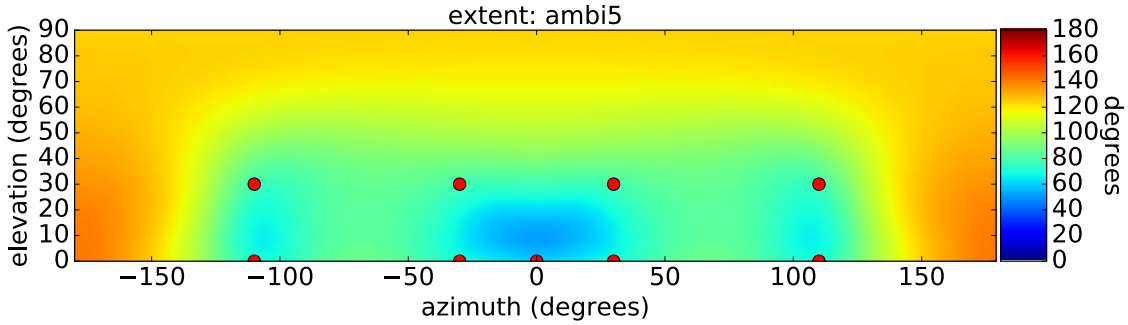


Figure 39 – rE-vector angle analysis on 4+5+0 layout with 5th order Ambisonics renderer.

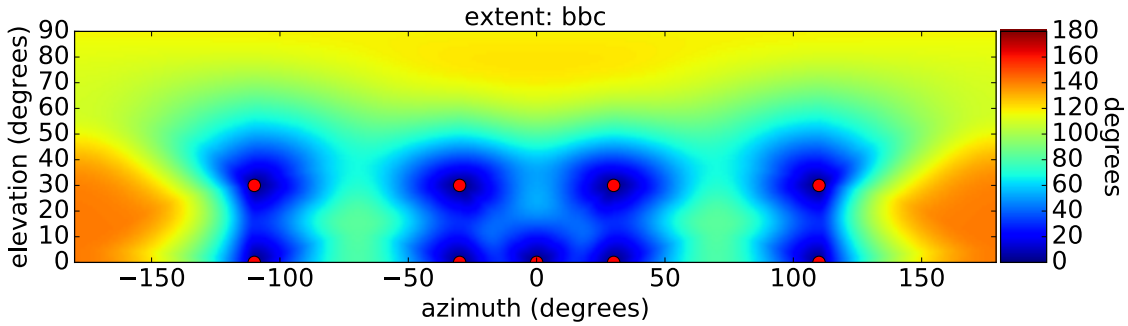


Figure 40 – rE-vector angle analysis on 4+5+0 layout with BBC renderer.

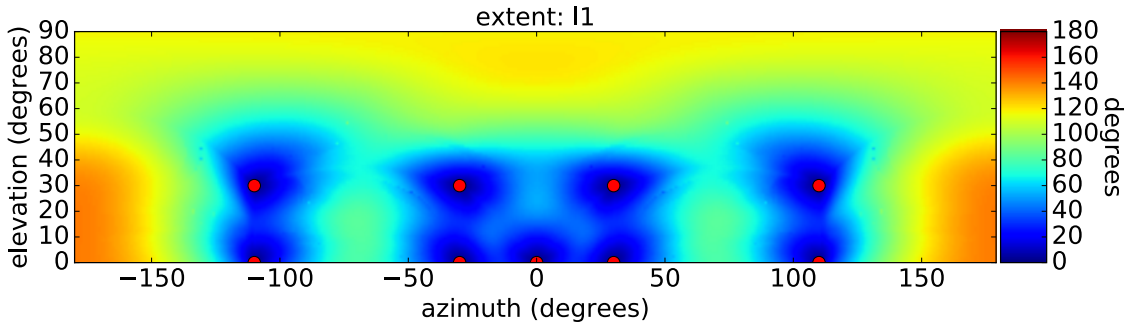


Figure 41 – rE-vector angle analysis on 4+5+0 layout with L1-norm renderer.

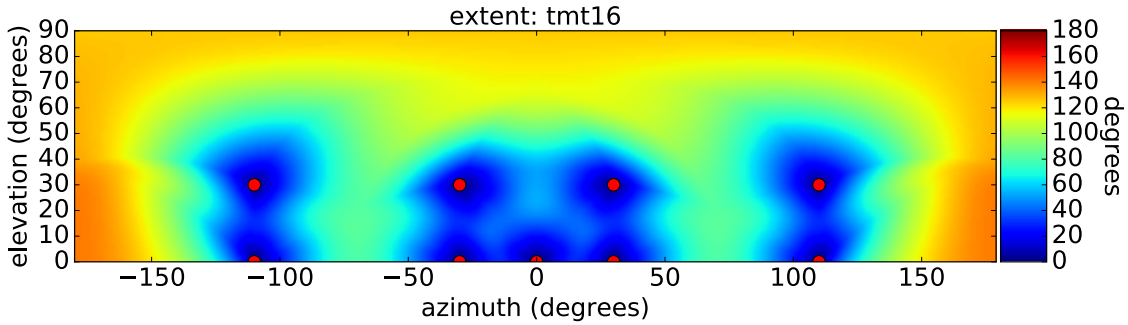


Figure 42 – rE-vector angle analysis on 4+5+0 layout with TMT renderer.

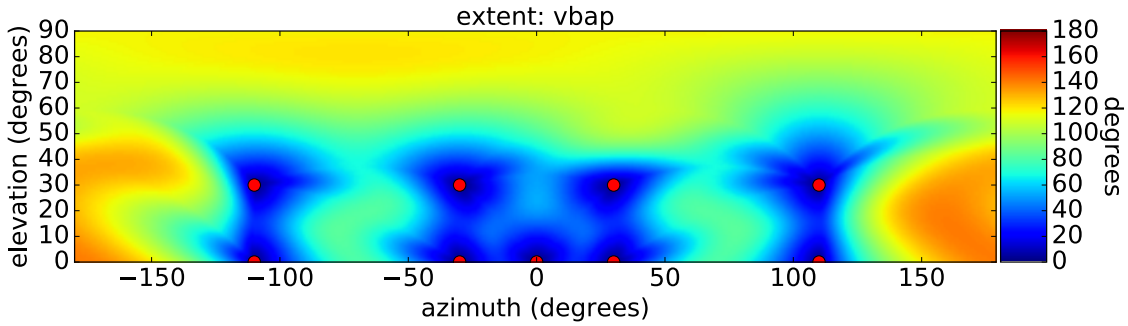


Figure 43 – rE-vector angle analysis on 4+5+0 layout with VBAP renderer.

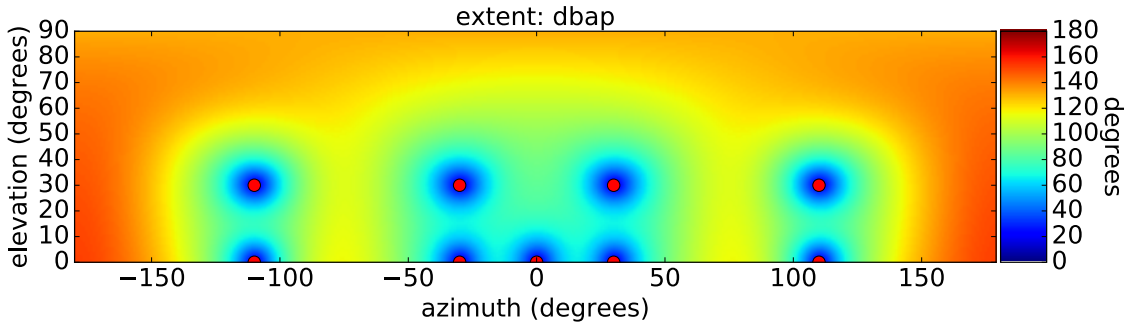


Figure 44 – rE-vector angle analysis on 4+5+0 layout with DBAP renderer.

A.2 3+7+0 Layout

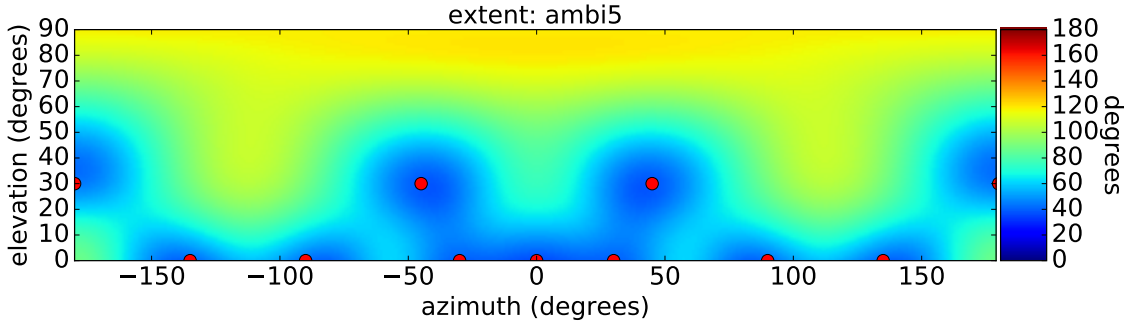


Figure 45 – rE-vector angle analysis on 3+7+0 layout with 5th order Ambisonics renderer.

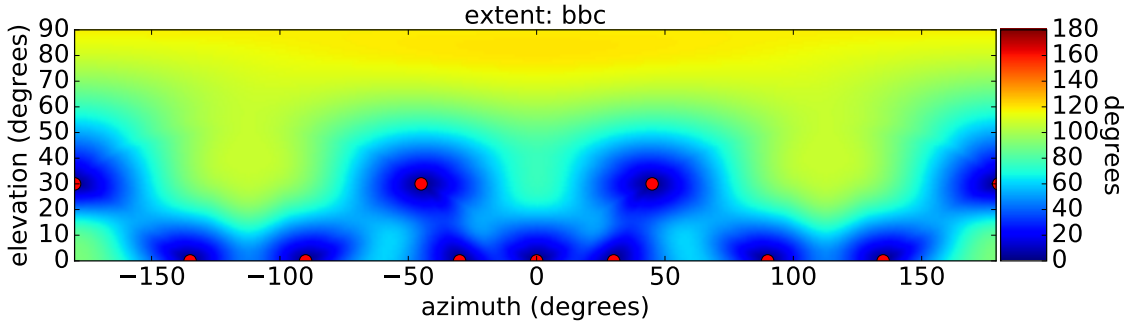


Figure 46 – rE-vector angle analysis on 3+7+0 layout with BBC renderer.

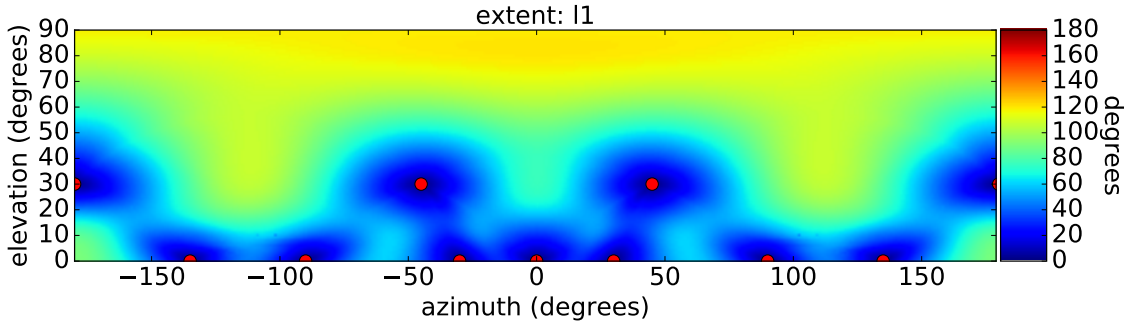


Figure 47 – rE-vector angle analysis on 3+7+0layout with L1-norm renderer.

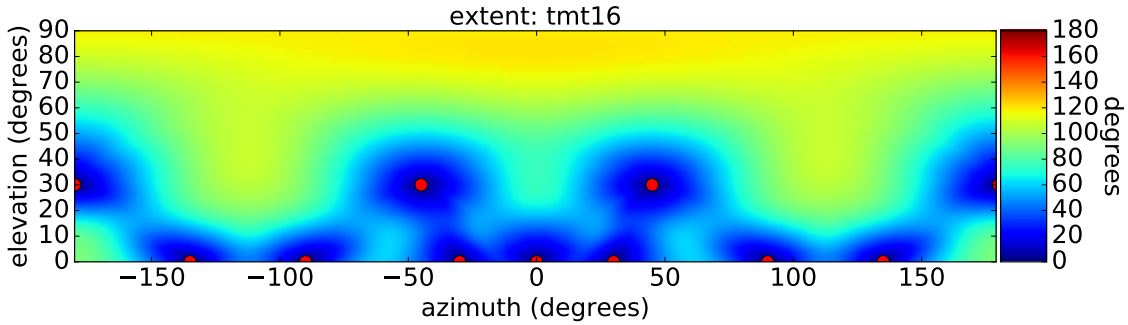


Figure 48 – rE-vector angle analysis on 3+7+0 layout with TMT renderer.

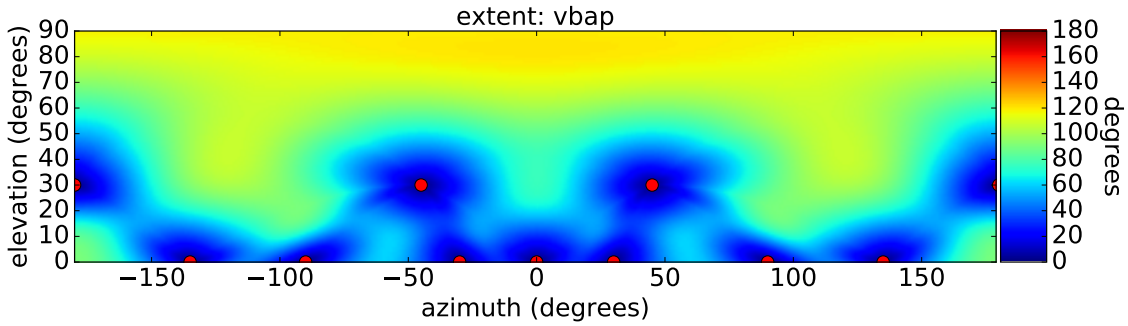


Figure 49 – rE-vector angle analysis on 3+7+0 layout with VBAP renderer.

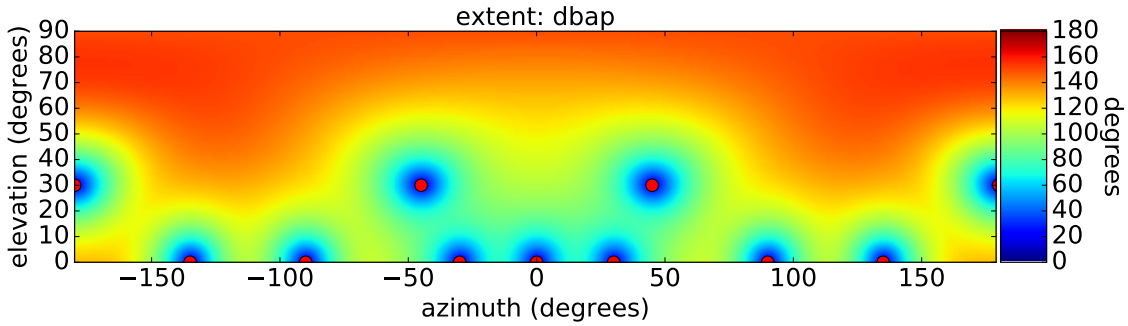


Figure 50 – rE-vector angle analysis on 3+7+0 layout with DBAP renderer.

B Median Renderer Differences of all Renderers

B.1 4+5+0 Layout

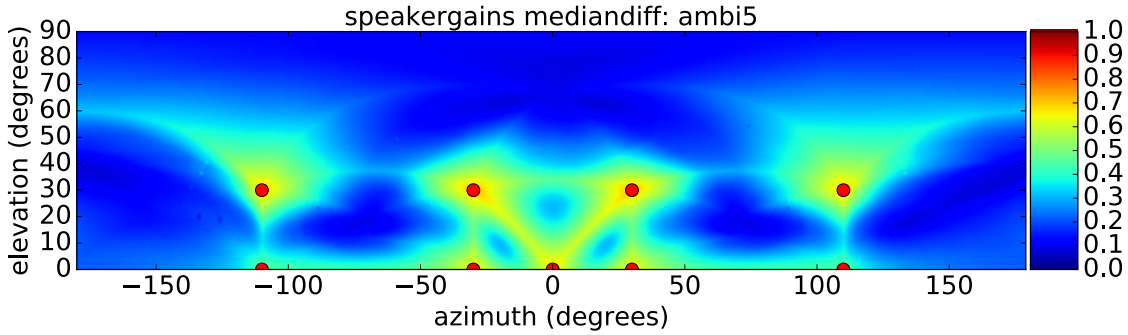


Figure 51 – Median renderer differences on 4+5+0 layout with 5th order Ambisonics renderer.

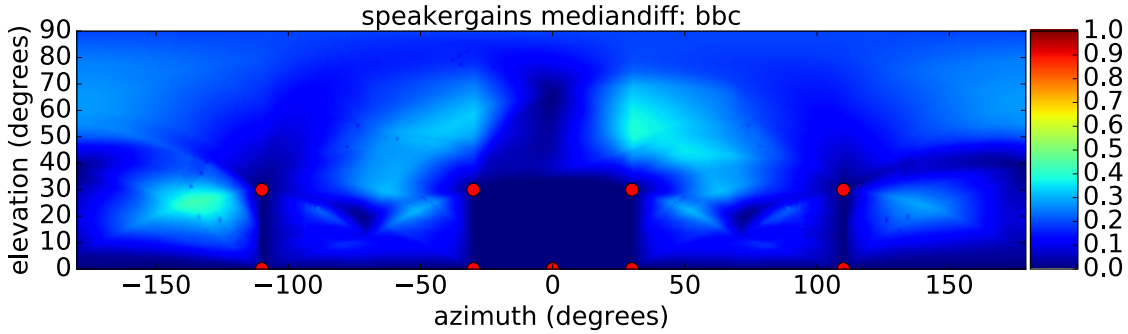


Figure 52 – Median renderer differences on 4+5+0 layout with BBC renderer.

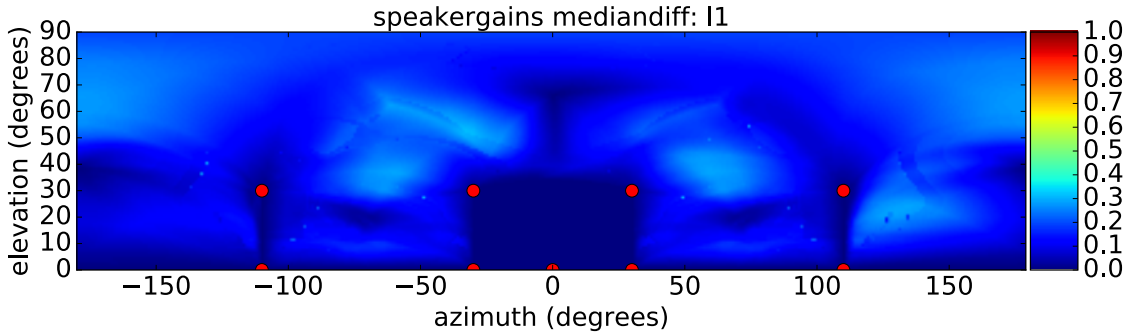


Figure 53 – Median renderer differences on 4+5+0 layout with L1-norm renderer.

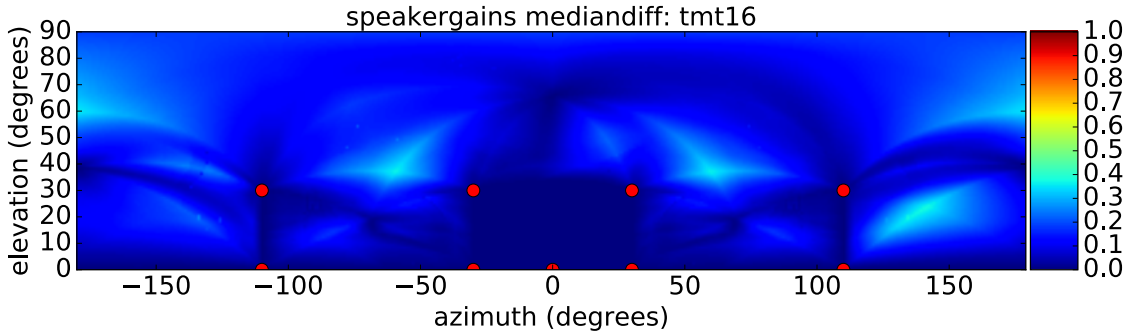


Figure 54 – Median renderer differences on 4+5+0 layout with TMT renderer.

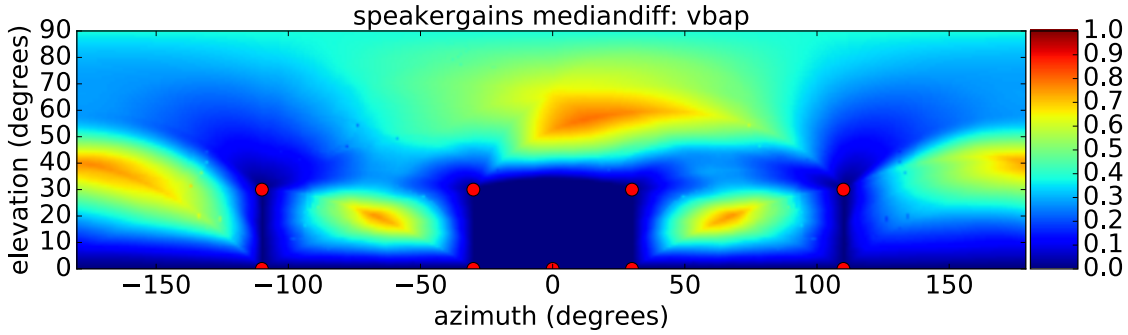


Figure 55 – Median renderer differences on 4+5+0 layout with VBAP renderer.

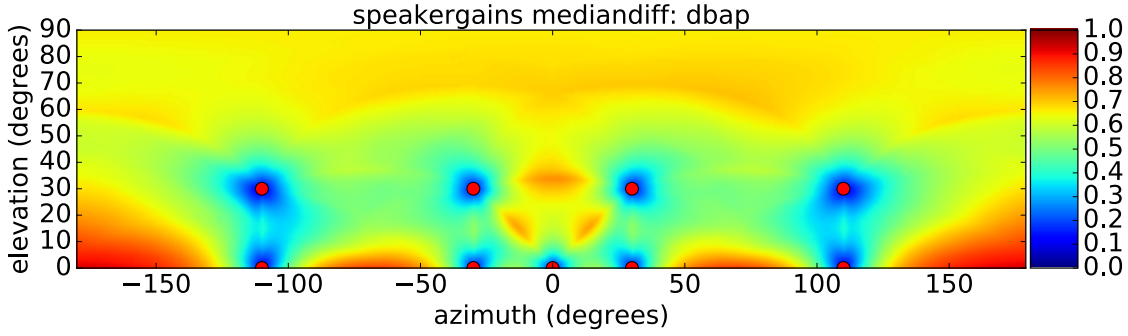


Figure 56 – Median renderer differences on 4+5+0 layout with DBAP renderer.

B.2 3+7+0 Layout

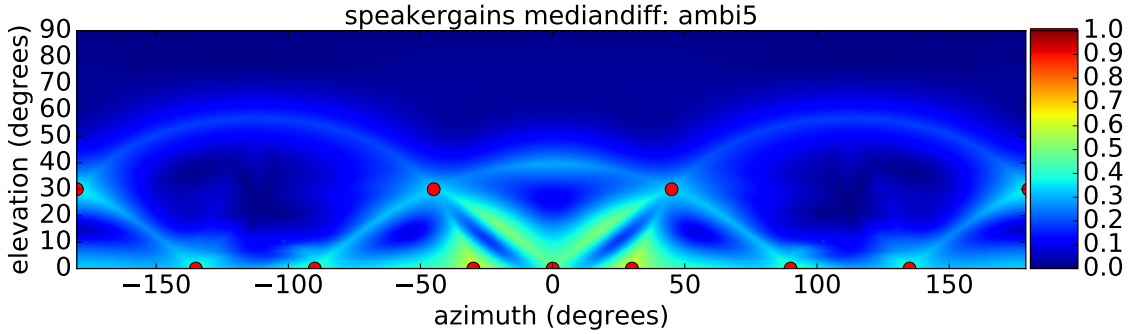


Figure 57 – Median renderer differences on 3+7+0 layout with 5th order Ambisonics renderer.

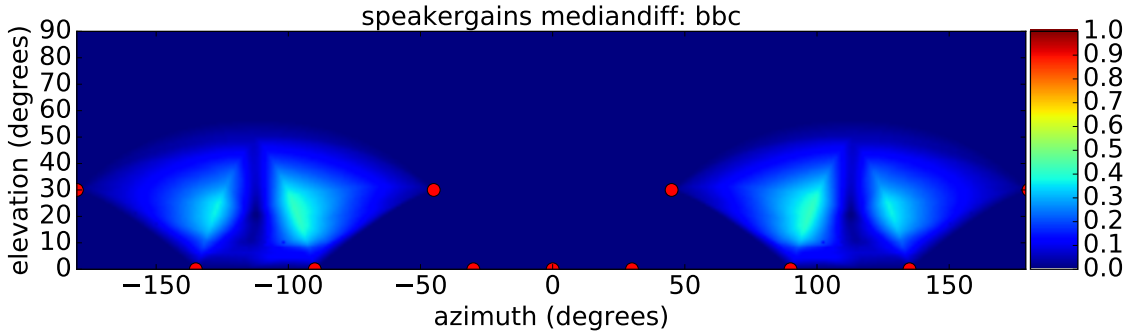


Figure 58 – Median renderer differences on 3+7+0 layout with BBC renderer.

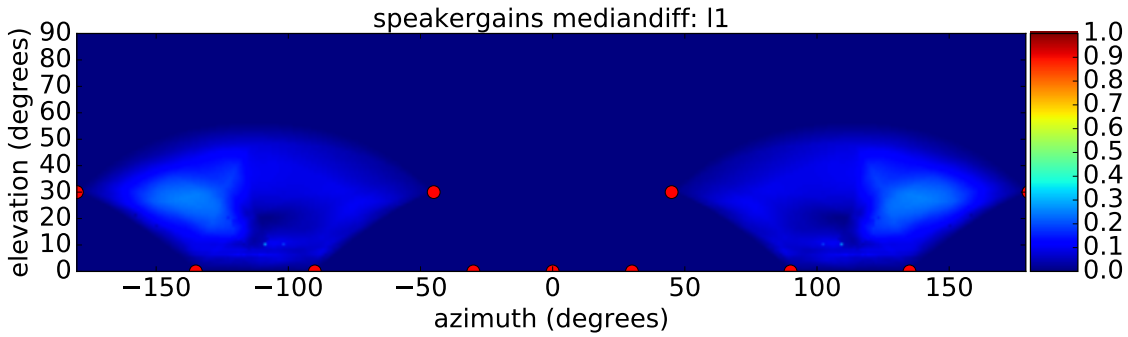


Figure 59 – Median renderer differences on 3+7+0 layout with L1-norm renderer.

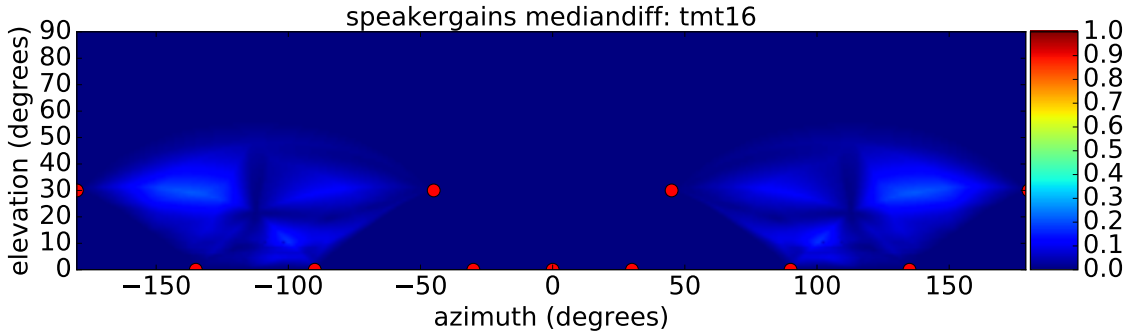


Figure 60 – Median renderer differences on 3+7+0 layout with TMT renderer.

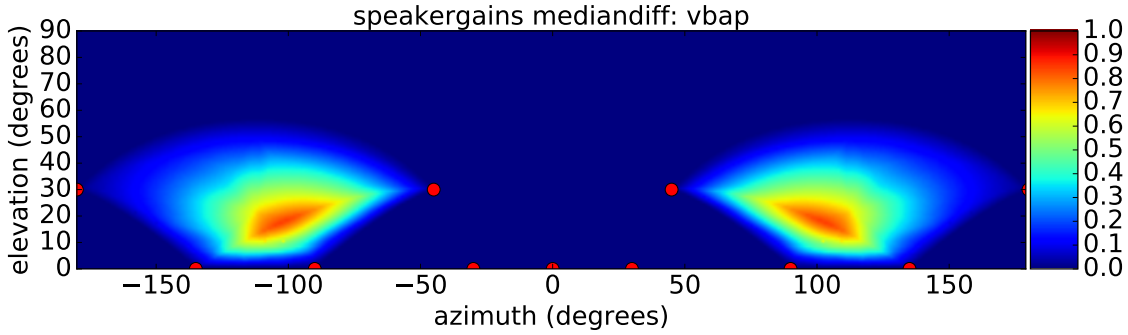


Figure 61 – Median renderer differences on 3+7+0 layout with VBAP renderer.

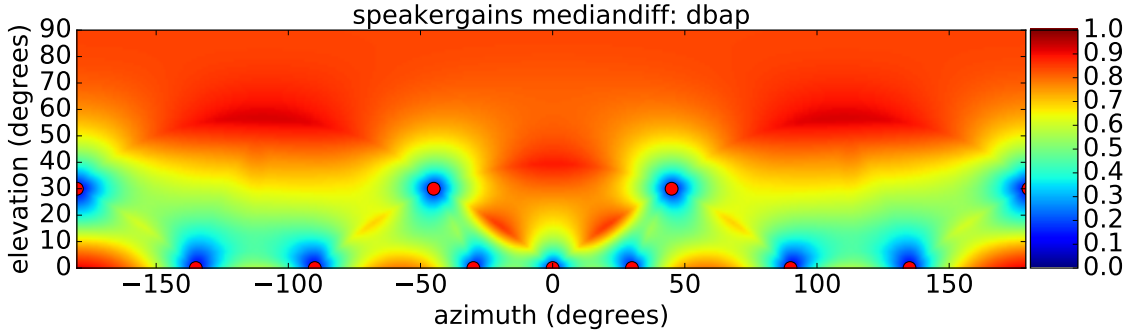


Figure 62 – Median renderer differences on 3+7+0 layout with DBAP renderer.

C "ambipy" Python Library

C.1 General Ambisonics

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 11 15:52:46 2016

@author: michaelr
"""

from scipy import special
from scipy import misc
import numpy as np




def fact(n):
    """Factorial function."""
    return misc.factorial(n, exact=True)

# bizarely, factorial is not natively vectorised with exact=True
fact_vec = np.vectorize(fact)


def sph_harm(n, m, azi, ele):
    """Implement spherical harmonics with SN3D normalisation according
    to section 2.1 of:

        Nachbar, Christian, et al. "Ambix-a suggested ambisonics format
    ." 3rd
        Ambisonics Symposium, Lexington, KY. 2011

    Parameters:
        n (int): HOA order 0 <= n
        m (int): HOA degree -n <= m <= n
        azi (float): azimuth in radians, as per bs.2076
        ele (float): elevation in radians, as per bs.2076
    Returns:
        gain (float): gain for this HOA channel
    """
```

```
37      azi = -azi # convert to counter-clockwise azimuth
38
39      delta_m = 1 if m == 0 else 0
40      norm = np.sqrt(((2.0 - delta_m) / (4 * np.pi)) * (float(fact(n - np
        .abs(m))) / fact(n + np.abs(m))))
41
42      if m < 0:
43          scale = np.sin(np.abs(m) * azi)
44      else:
45          scale = np.cos(np.abs(m) * azi)
46
47      return norm * special.lpmv(np.abs(m), n, np.sin(ele)) * scale
48
49  def relative_angle(x, y):
50      """Assuming y is clockwise from x, increment y by 360 until it's
        not less
51      than x.
52
53      Parameters:
54          x (float): start angle in degrees.
55          y (float): end angle in degrees.
56
57      Returns:
58          float: y shifted such that it represents the same angle but is
        greater
59          than x.
60      """
61      while y < x:
62          y += 360.0
63      return y
64
65  def acn(n, m):
66      """Calculating Ambisonics Channel Number (ACN) Formula (5)
67      form n = order and m = degree"""
68      if abs(m) <= n:
69          acn = n**2 + n + m
70          return acn
71      else:
72          print "Error: abs(m) can't be greater than n."
73          return None
74
75  def order_from_chnr(acn):
76      """Computes ambisonics order from incoming multichannel file"""
77
```

```
78      order = np.sqrt(acn)-1
79
80      if (order%1 == 0):
81          return int(order)
82      else:
83          print "Channelnumber is not valid for 3D-ambisonics"
84
85
86  def changeOrder(signals, pot, inv=True):
87
88      """takes encoded signals and changes continousely the ambisonics
        order
89      by pot [0-360] or inverted [360-0] if inv = True"""
90      pot = float(pot)/360
91      order = order_from_chnr(len(signals))
92      gains = np.zeros(order)
93      gains[0] = 1
94
95      if inv == True:
96          pot = -pot*order+order
97      else:
98          pot = pot*order
99
100     for n in range(1,order):
101         if n < int(pot):
102             gains[n] = 1
103         elif n == int(pot):
104             gains[n] = pot-int(pot)
105         else:
106             gains[n] = 0
107
108     for n in range(1,order+1):
109         for m in range(-n,n+1):
110             signals[:,acn(n,m)] *= gains[n-1]
111
112     return None
113
114
115 class Ambi_Panner(object):
116
117     def __init__(self, order):
118         self.order = order
119         self.init_spherical_harmonics()
120
```

```
121     def init_spherical_harmonics(self):
122         ls = []
123         ms = []
124         for l in range(0, self.order+1):
125             for m in range(-l,l+1):
126                 ls.append(l)
127                 ms.append(m)
128 
129         ls = np.array(ls)
130         ms = np.array(ms)
131 
132         delta_ms = np.choose(ms == 0, [0, 1])
133         norms = np.sqrt(((2.0 - delta_ms) / (4 * np.pi)) * (fact_vec(ls
    - np.abs(ms)).astype(float) / fact_vec(ls + np.abs(ms))))
134 
135         angle_offsets = np.choose(ms < 0, [np.pi/2, 0])
136 
137         self.angle_offsets = angle_offsets
138         self.norms = norms
139         self.ls = ls
140         self.ms = ms
141 
142 
143     def ambi_pan(self, azi, ele):
144         azi = -azi # convert to counter-clockwise azimuth
145 
146         abs_ms = np.abs(self.ms)
147         scale = np.sin(self.angle_offsets[np.newaxis] + abs_ms[np.
    newaxis] * azi[:, np.newaxis])
148 
149         return self.norms[np.newaxis] * special.lpmv(abs_ms[np.newaxis
    ], self.ls[np.newaxis], np.sin(ele[:, np.newaxis])) * scale
```

C.2 Widening

```
import numpy as np
import scipy as sp
import math as m
import sounddevice as sd
import matplotlib.pyplot as plt
from scipy import signal



def acn(n, m):
    """Calculating Ambisonics Channel Number (ACN) Formula (5)
    form n = order and m = degree"""
    if abs(m) <= n:
        acn = n**2 + n + m
        return acn
    else:
        print "Error: abs(m) can't be greater than n."
        return None


def load_Ry_matrix(order):
    """takes order of ambisonics system and returns a list of 90 degree
    y-axis
    transformation matrices"""

    matlist = []

    if order < 1:
            print 'Error: Order not valid, must be >= 1.'
    elif 1 <= order < 22:
        for n in range(1,order+1):
            if n < 10:
                matlist.append(np.loadtxt('Ry90_CGs/Ry90_0' + str(n)))
            elif 10 <= n < 22:
                matlist.append(np.loadtxt('Ry90_CGs/Ry90_' + str(n)))
        return matlist
    else:
        print 'Sorry, no matrices available for order > 21.'

def rotate_y_90(inputsignals):
    """performs a 90 degree rotation on the y-axis"""
```

```
41
42      order = order_from_chnr(np.shape(inputsignals)[1])
43      matlist = load_Ry_matrix(order)
44
45      outputsignals = []
46      outputsignals.append(inputsignals[:,0])
47
48      for n in range (1,order+1):
49          temp = np.dot(inputsignals[:,acn(n,-n):acn(n,n)+1],matlist[n
    -1])
50          outputsignals.extend(temp.T)
51
52      return np.asarray(outputsignals).T
53
54  def horizontal_widening(inputsignals,wideningangle,rotationangle,Lambda
    ,Q):
55      first_z_90 = z_rotator_wide(inputsignals,0,m.pi/2,1,1)
56      first_y_90 = rotate_y_90(first_z_90)
57      widened = z_rotator_wide(first_y_90,wideningangle,rotationangle,
    Lambda,Q)
58      second_z_90 = z_rotator_wide(widened,0,m.pi/2,1,1)
59      second_y_90 = rotate_y_90(second_z_90)
60      return second_y_90
61
62
63  def order_from_chnr(acn):
64      """Computes ambisonics order from incoming multichannel file"""
65
66      order = m.sqrt(acn)-1
67
68      if (order%1 == 0):
69          return int(order)
70      else:
71          print "Channelnumber is not valid for 3D-ambisonics"
72
73  def z_rotation_matrix(M,wideningangle,rotationangle,Lambda,Q):
74
75
76      zero_zero = build_ir(Q,h(M*wideningangle,M*rotationangle,Lambda)
    [0])
77      zero_one = build_ir(Q,h(M*wideningangle,M*rotationangle+m.pi/2,
    Lambda)[0])
78      one_zero = build_ir(Q,h(M*wideningangle,M*rotationangle-m.pi/2,
    Lambda)[0])
```

```
79      one_one = zero_zero # same as zero_zero
80
81      return np.asarray([[zero_zero, zero_one],[one_zero, one_one]])
82
83  def z_rotator_wide_pairwise(inputsignalpair,rotationmatrix):
84      """performs an rotation sezified by a rotation matrix usincs 2x2
        convolution"""
85
86      outzero = np.convolve(inputsignalpair[0],rotationmatrix[0,0]) + np.
        convolve(inputsignalpair[1],rotationmatrix[1,0])
87      outone = np.convolve(inputsignalpair[0],rotationmatrix[0,1]) + np.
        convolve(inputsignalpair[1],rotationmatrix[1,1])
88
89      return np.asarray([outzero, outone]).T
90
91  def z_rotator_wide(inputsignals,wideningangle,rotationangle,Lambda,Q):
92      """Performs z-rotation and widening depending on rotation and
        widening angle
93      Lambda and Q defined in build_ir"""
94
95      order = order_from_chnr(np.shape(inputsignals)[1])
96      rotmatrix_list = []
97
98      filterlen = 2*Lambda*Q+1
99      insiglen = inputsignals.shape[0]
100     outlen = filterlen+insiglen-1
101
102     outputsignals = np.zeros((outlen,inputsignals.shape[1])) #
        initialize output vector
103
104     for n in range(1,order+1): # write list of rotation matricec for
        every m
105         rotmatrix_list.append(z_rotation_matrix(n,wideningangle,
        rotationangle,Lambda,Q))
106
107     for n in range(1,order+1): # perform rotation for all nonzero m
        components
108         for m in range(1,n+1):
109             temp = z_rotator_wide_pairwise(np.asarray([inputsignals[:,
        acn(n,-m)],inputsignals[:,acn(n,m)]]),rotmatrix_list[n-1])
110             outputsignals[:,acn(n,-m)] = temp[:,0]
111             outputsignals[:,acn(n,m)] = temp[:,1]
112             print "rotated channels: " + str(acn(n,-m)) + " " + str(acn
        (n,m))
```

```
113
114     for n in range(0,order+1):  # copy zero m elements from
    inputsignals
115         outputsignals[0:insiglen,acn(n,0)] = inputsignals[:,acn(n,0)]
116         print "copied channels: " + str(acn(n,0))
117
118     return  outputsignals
119
120
121 def h(alpha,beta,offset):
122     """Generates Coeficients of widening filter depending on alpha,
    beta and Lam"""
123
124     lam = range(0,2*offset+1)
125
126     hc = np.multiply(sp.cos(m.pi/2*np.abs(lam) + beta),sp.special.jv(np
    .abs(lam),alpha))
127     hs = np.multiply(sp.cos(m.pi/2*np.abs(lam) - beta),sp.special.jv(np
    .abs(lam),alpha))
128
129     return hc , hs
130
131
132 def H(alpha,beta):   # frequency dependent direction vector
133
134     t = np.linspace(0, 1, num=100)
135     Omega = t*2*m.pi
136
137     s = (np.cos(alpha*np.cos(Omega)+beta),np.sin(alpha*np.cos(Omega)+
    beta))
138
139
140     plt.figure(1)
141     plt.subplot(211)
142     plt.ylabel('H(alpha,beta,Omega)')
143     plt.xlabel('normalized frequency Omega in 360*degree/Hz')
144     plt.axis([0, 1, 0, 1])
145     plt.plot(t, s[0], 'bo', t, s[1], 'k')
146
147     return s[0] , s[1]
148
149
150 def build_ir(Q,hc):
151     """Computes impule response widening filter with Q steps between
```

```
    the
152     entries in hc or hc"""
153     Q = Q - 1
154     ir = np.zeros(Q*len(hc)+len(hc)-Q)
155     c = 0
156 
157     for n in range(0,len(ir),Q+1):
158         ir[n] = hc[c]
159         c = c + 1
160 
161     plt.figure(1)
162     plt.subplot(211)
163     plt.xlabel('discrete time q')
164     plt.ylabel('h(alpha,beta,q]')
165     plt.axis([-len(ir)/2, len(ir)/2, ir.min(), ir.max()])
166     plt.plot(range(-len(ir)/2,len(ir)/2), ir, 'o')
167 
168     return ir # normalisation should be applied, cause output signals
    vary a lot in loudness
169 
170 
171 def stereo_widening(inputsignal, disperion, Q, offset):
172     """Computes Comvolution for two channels"""
173 
174     l,r = h(disperion,m.pi/4,offset)
175 
176     left = np.convolve(inputsignal,build_ir(Q,l))
177     right = np.convolve(inputsignal,build_ir(Q,r))
178 
179     return left, right
180 
181 def stereo_widen_file_example(inputfilename,outputfilename,disperion):
182     """stereo widening example with Q=110, offset = 9 fs=44100
183     reads wavfile, takes the left channel and writes widened wav file
    """
184 
185     sound_mono = scikits.audiolab.wavread(inputfilename)[0]
186     sound_mono = sound_mono[:,0]
187     scikits.audiolab.wavwrite(np.transpose(stereo_widening(sound_mono,
    disperion,110,9)),outputfilename,44100)
188 
189     return None
```

C.3 Feedback Delay Network

```
import numpy as np
import scipy as sp
import math as m
import pandas as pd

def white_noise_gen(samplerate,duration,maxamp):
    signal = np.zeros(samplerate*duration)
    for n in range(0,len(signal)):
        signal[n] = np.random.random()*2-1

    if maxamp > 1:
        maxamp = 1
        print "maxamp can't be greater than 1, otherwise clipping
    appears. maxamp set to 1."
    elif maxamp < -1:
        maxamp = -1
        print "maxamp can't be smaller than -1, otherwise clipping
    appears. maxamp set to -1."
    else:
        pass

    return signal*maxamp

def impulse_train(samplerate,duration,interval,maxamp):
    signal = np.zeros(samplerate*duration)
    for n in range(0,len(signal),interval):
        signal[n] = 1
    return signal*maxamp

def get_primes(n):
    numbers = set(range(n, 1, -1))
    primes = []
    while numbers:
        p = numbers.pop()
        primes.append(p)
        numbers.difference_update(set(range(p*2, n+1, p)))
    return primes

def conv_1Ord(inputsig,filters):

    sig0 = signal.convolve(inputsig[:,0],filters[:,0])
```

```
    sig1 = signal.convolve(inputsig[:,1], filters[:,1])
    sig2 = signal.convolve(inputsig[:,2], filters[:,2])
    sig3 = signal.convolve(inputsig[:,3], filters[:,3])

    return [sig0, sig1, sig2, sig3]

def fdn_1Ord(fs, gain):

    # Create an impulse

    B, A = signal.butter(1, 0.5, 'low', output = 'ba')

    x = np.zeros(fs)
    x[0] = 1

    y0 = np.zeros(fs)
    y1 = np.zeros(fs)
    y2 = np.zeros(fs)
    y3 = np.zeros(fs)

    b = [1, 1, 1, 1]

    c = [1, 1, 1, 1]

    # Feedback matrix

    a = np.zeros((4,4))

    a[0]=[0, 1, 1, 0]
    a[1]=[-1, 0, 0, -1]
    a[2]=[1, 0, 0, -1]
    a[3]=[0, 1, -1, 0]

    a = a*(1/m.sqrt(2)) * gain;

    # Delay lines, use prime numbers

    M = np.array([823, 829, 853, 863])

    z0 = np.zeros(M.max())
    z1 = np.zeros(M.max())
    z2 = np.zeros(M.max())
    z3 = np.zeros(M.max())

```

```
84      for n in range(0,len(y1)):
85
86          tmp = [z0[M[0]-1], z1[M[1]-1], z2[M[2]-1], z3[M[3]-1]]
87          #y[n] = x[n] + c[0]*z0[M[0]-1] + c[1]*z1[M[1]-1] + c[2]*z2[M
    [2]-1] + c[3]*z3[M[3]-1]
88          y0[n] = c[0]*z0[M[0]-1] + x[n]
89          y1[n] = c[1]*z1[M[1]-1] + x[n]
90          y2[n] = c[2]*z2[M[2]-1] + x[n]
91          y3[n] = c[3]*z3[M[3]-1] + x[n]
92
93          z0[1:len(z0)+1] = z0[0:len(z0)-1]
94          z0[0] = x[n]*b[0] + np.dot(tmp,a[0,:])
95          z0 = signal.filtfilt(B,A,z0)
96
97          z1[1:len(z1)+1] = z1[0:len(z1)-1]
98          z1[0] = x[n]*b[1] + np.dot(tmp,a[1,:])
99          z1 = signal.filtfilt(B,A,z1)
100
101         z2[1:len(z2)+1] = z2[0:len(z2)-1]
102         z2[0] = x[n]*b[2] + np.dot(tmp,a[2,:])
103         z2 = signal.filtfilt(B,A,z2)
104
105         z3[1:len(z3)+1] = z3[0:len(z3)-1]
106         z3[0] = x[n]*b[3] + np.dot(tmp,a[3,:])
107         z3 = signal.filtfilt(B,A,z3)
108
109     return np.array([y0, y1, y2, y3]).T #y/max(y)
110
111 def fdn(fs, gain):
112
113     # Create an impulse
114
115     x = np.zeros(fs)
116     x[0] = 1
117
118     y = np.zeros(fs)
119
120     b = [1, 1, 1, 1]
121
122     c = [0.8, 0.8, 0.8, 0.8]
123
124     # Feedback matrix
125
126     a = np.zeros((4,4))
```

```
127
128     a[0]=[0, 1, 1, 0]
129     a[1]=[-1, 0, 0, -1]
130     a[2]=[1, 0, 0, -1]
131     a[3]=[0, 1, -1, 0]
132
133     a = a*(1/m.sqrt(2)) * gain;
134
135     # Delay lines, use prime numbers
136
137     M = np.array([149, 211, 263, 293])
138
139     z0 = np.zeros(M.max())
140     z1 = np.zeros(M.max())
141     z2 = np.zeros(M.max())
142     z3 = np.zeros(M.max())
143
144     for n in range(0,len(y)):
145
146         tmp = [z0[M[0]-1], z1[M[1]-1], z2[M[2]-1], z3[M[3]-1]]
147         y[n] = x[n] + c[0]*z0[M[0]-1] + c[1]*z1[M[1]-1] + c[2]*z2[M
    [2]-1] + c[3]*z3[M[3]-1]
148
149         z0[1:len(z0)+1] = z0[0:len(z0)-1]
150         z0[0] = x[n]*b[0] + np.dot(tmp,a[0,:])
151
152         z1[1:len(z1)+1] = z1[0:len(z1)-1]
153         z1[0] = x[n]*b[1] + np.dot(tmp,a[1,:])
154
155         z2[1:len(z2)+1] = z2[0:len(z2)-1]
156         z2[0] = x[n]*b[2] + np.dot(tmp,a[2,:])
157
158         z3[1:len(z3)+1] = z3[0:len(z3)-1]
159         z3[0] = x[n]*b[3] + np.dot(tmp,a[3,:])
160
161
162     return y/max(y)
163
164
165 def fdn_filt(fs, gain):
166
167     # Create an impulse
168
169     B, A = signal.butter(2, 0.3, 'low', output = 'ba')
```

```
170
171     x = np.zeros(fs)
172     x[0] = 1
173
174     y = np.zeros(fs)
175
176     b = [1, 1, 1, 1]
177
178     c = [0.8, 0.8, 0.8, 0.8]
179
180     # Feedback matrix
181
182     a = np.zeros((4,4))
183
184     a[0]=[0, 1, 1, 0]
185     a[1]=[-1, 0, 0, -1]
186     a[2]=[1, 0, 0, -1]
187     a[3]=[0, 1, -1, 0]
188
189     a = a*(1/m.sqrt(2)) * gain;
190
191     # Delay lines, use prime numbers
192
193    M = np.array([149, 211, 263, 293])
194
195     z0 = np.zeros(M.max())
196     z1 = np.zeros(M.max())
197     z2 = np.zeros(M.max())
198     z3 = np.zeros(M.max())
199
200     for n in range(0,len(y)):
201
202         tmp = [z0[M[0]-1], z1[M[1]-1], z2[M[2]-1], z3[M[3]-1]]
203         y[n] = x[n] + c[0]*z0[M[0]-1] + c[1]*z1[M[1]-1] + c[2]*z2[M
    [2]-1] + c[3]*z3[M[3]-1]
204
205         z0[1:len(z0)+1] = z0[0:len(z0)-1]
206         z0[0] = x[n]*b[0] + np.dot(tmp,a[0,:])
207         z0 = signal.filtfilt(B,A,z0)
208
209         z1[1:len(z1)+1] = z1[0:len(z1)-1]
210         z1[0] = x[n]*b[1] + np.dot(tmp,a[1,:])
211         z1 = signal.filtfilt(B,A,z1)
212
```

```
213         z2[1:len(z2)+1] = z2[0:len(z2)-1]
214         z2[0] = x[n]*b[2] + np.dot(tmp,a[2,:])
215         z2 = signal.filtfilt(B,A,z2)
216 
217         z3[1:len(z3)+1] = z3[0:len(z3)-1]
218         z3[0] = x[n]*b[3] + np.dot(tmp,a[3,:])
219         z3 = signal.filtfilt(B,A,z3)
220 
221     return y/max(y)
222 
223 def lop(): # lowpass filter
224     b, a = signal.butter(4, 0.01, 'low', output = 'ba')
225     w, h = signal.freqs(b, a)
226     plt.semilogx(w, 20 * np.log10(abs(h)))
227     plt.title('Butterworth filter frequency response')
228     plt.xlabel('Frequency [radians / second]')
229     plt.ylabel('Amplitude [dB]')
230     plt.margins(0, 0.1)
231     plt.grid(which='both', axis='both')
232     plt.axvline(100, color='green') # cutoff frequency
233     plt.show()
234     return None
```