Institute for Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Computation of 3D Audio Using Ambisonics for an Immersive Audio-Visual 3D Experience

Lukas Piller, B.Sc.

Course of Study:	Informatik
Examiner:	Prof. Dr. Dirk Pflüger
Supervisor:	Prof. Dr. Dirk Pflüger
Commenced:	November 3, 2022
Completed:	August 3, 2023

Acknowledgments

I am extremely grateful to Prof. Dr. Dirk Pflüger for his great patience and feedback. This project would not have been possible without the expertise provided by Jun.-Prof. Dr. Miriam Akkermann and Prof. Dr. Jens Martin Loebel. Additionally, I also could not have undertaken this journey without the help of Dr. Stefan Zimmer who helped my with many problems on the way. I am also grateful to M.A. Uğur Can Akkaya for his help and assistance. Special thanks to Dr. Christian Stein, the former students of the Universität Bayreuth Andreas Egginger, Cai Kuhn, Laura Rosinger, Linus Harwig, Raphael Stieglbauer and the former student of TU Berlin Felicitas Fiedler who made the recordings that were used for the experience together with Miriam Akkermann and Jens Martin Loebel. Lastly, I would also like to thank my friends and family for all the support they provided me.

Abstract

In this thesis an immersive audio-visual 3D experience was created in Unity using VR. This experience is based on high quality spatial audio and 360° video recordings. In order to process the audio without loss of quality, it was encoded into the Ambisonics surround sound format from which it was decoded to an optimal loudspeaker layout. Thus, the mathematical background for Ambisonics is presented in this thesis. The processing was done using the software REAPER and the IEM plug-in suite for Ambisonics. Additionally, since the audio and video had been recorded separately, they were synchronized not only in time but also in angle using a beamforming technique. The processed audio was combined with the synchronized video in a VR experience with gamification elements, which were created in Unity with the help of the Windows Mixed Reality feature packages. Finally, an experiment showing the quality of the angular synchronization was performed.

Contents

List of Figures

Acronyms

ACN	Ambisonic channel number. 28
AllRAD	All-Round Ambisonic Decoding. 34
AR	Augmented Reality. 13
DAW	Digital Audio Workstation. 35
DOA	Direction of Arrival. 23
FOA	First-Order Ambisonics. 13
HOA	Higher-Order Ambisonics. 13
HRIR	Head-related Impulse Response. 18
HRTF	Head-related Transfer Function. 18
ILD	Interaural Level Differences. 18
ITD	Interaural Time Differences. 18
MMSE	minimum mean squared error. 32
MRTK	Mixed Reality Toolkit. 50
REAPER	Rapid Environment for Audio Production, Engineering and Recording. 9, 35
SRP	Steered Response Power. 9, 44
VBAP	Vector Based Amplitude Panning. 34
VR	Virtual Reality. 13
XR	Extended Reality. 13

1 Introduction

With the rise of Extended Reality (XR) and big data it has become possible to experience places in completely new ways. Places can be captured with high quality video and audio recordings and experienced at special moments, or even though they are far away. Even completely virtual places can be created, and experienced at any time or place using Virtual Reality (VR). Unlike just seeing a video on a screen, this allows the user to experience the place in an immersive way, being able to look around and sometimes even walk around. These possibilities are attractive not only for 3D video games and 3D videos but also for educational and business purposes. Ancient ruins can be visited without ever leaving the classroom - or even reconstructed and experienced as they were in their prime. A new building can be viewed before it is even built or a construction site can be viewed in its finished state with the use of Augmented Reality (AR) headsets.

However, these techniques often focus more on the visual aspect than the audio aspect. In purely virtual environments, like 3D video games, there often are spatial audio components but their quality and usage is much lower compared to video. While for replay of audio-visual 3D recordings the visual experience is fully in high resolution 3D, the audio might only have poor spatial resolution or no spatial component at all. While the theory and to some extension plugins for higher quality spatial audio exist, they often are not usable out-of-the-box in many commercial tools for use in VR experiences. For example, Unity only has native support for First-Order Ambisonics (FOA) but not the better Higher-Order Ambisonics (HOA). This is additionally complicated by the fact that nowadays, 3D video is nearly always recorded directly as a 360° video or as multiple videos that can be easily stitched, However, for spatial audio, there are many different capturing setups. While some of these can be direct spatial audio recordings with special microphones, they can also be made with more personalized microphone arrays, which makes a standardized signalflow for data processing more difficult.

Thus, in this thesis an audio-visual 3D VR experience with gamification elements is created using Unity. We start from given high quality audio and video recordings, with a focus on preserving the high audio quality using HOA. Additionally, the data processing from given input data to output for use in Unity is chosen to be modular to allow for future adaptations of single components without needing to change everything.

The audio and video materials used were recorded in the course of a student project of the Universität Bayreuth in 2018, led by Miriam Akkermann and Jens Martin Loebel from the Universität Bayreuth (now at TU Dresden and FH Magdeburg-Stendal respectively) and Christian Stein from the HU Berlin. The video was captured using a *Freedom360 F360 Mount for GoPro*[1] holding six GoPros in spherical configuration. The videos were then stitched together into one 360° video. The audio was captured with an eight channel circular microphone with an equidistant distribution of microphones. Thus, only 2D spatial audio in the horizontal plane is captured. Each channel was recorded with

[1] https://www.bhphotovideo.com/c/product/1203763-REG/freedom360_f360m_f360_mount.html

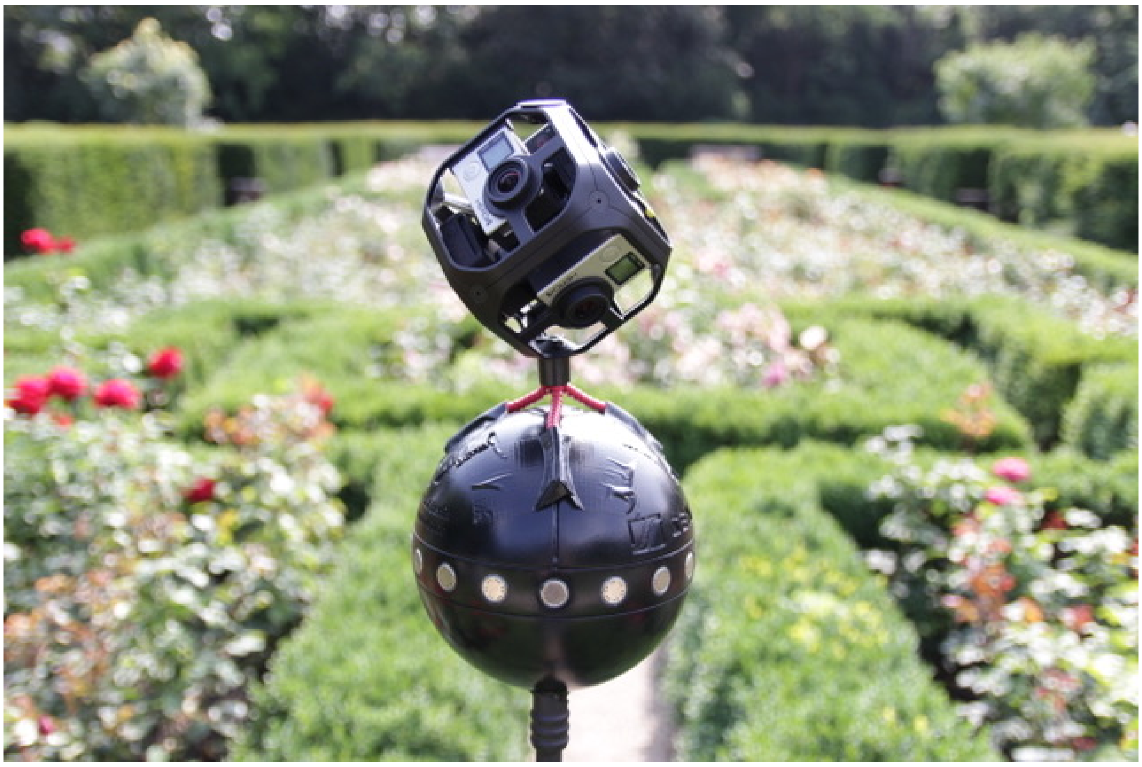


Figure 1.1: The recording setup used to create the audio and video recordings. On the top is the mount with the six GoPros and on the bottom we can see the circular microphone. Picture provided by Miriam Akkermann.

a sample rate of 48kHz and a bit depth of 32-bit. The complete recording setup can be seen in Figure 1.1. The recordings were made at 11 different locations in Berlin and Bayreuth. The total amount of data recorded was over 1 TB in size.

Since the data was provided after already having been recorded and preprocessed, the audio recording setup could not be chosen to be optimal for the intended purpose.

To create the experience there are three major components: Firstly, the audio data needs to be preprocessed for use in Unity. The recordings are of what the microphone array hears from the outside world, but for Unity we need audio tracks for a loudspeaker array that recreate the outside world as well as possible for a listener. For this, HOA was used as an intermediate data format to preserve the spatial properties of the recordings.
Secondly, since the audio and video are recorded separately, both need to be synchronized. For normal audio and video recordings, only the time component needs to be synchronized so that visual and auditory cues are matched. For spatial audio and 360° video we also have an angular component: The sound source in the 2D audio needs to be at the same direction as the horizontal direction of the source is in the 3D video.
Finally, a VR application needs to be created with Unity to bring all parts together and allow for playback of the audio-visual experience. This experience includes a gamification element.

These components will be chosen to be relatively independent of each other, so changes to one component will not have much influence on the other components. For example, using a different method for synchronization should have no influence on data processing and replay. Or replay using a sound dome and a VR headset only needs a new replay application, but can use the data processing and synchronization presented here. This allows for easier future adaptation to different settings or improvement of components.

The structure of the thesis will be as follows: The following chapter will discuss some foundations of spatial audio. In the next chapter, Ambisonics and its mathematical foundations will be introduced. In particular, it will be discussed how to encode into the Ambisonics format from a microphone array and how to decode from Ambisonics to a loudspeaker array. The following three chapters will discuss the three major components: How the audio is encoded and decoded using Ambisonics in REAPER, how the audio and video are synchronized both in time and angle, and finally how everything comes together in Unity and how the experience is implemented. The last chapter will recap the thesis and draw conclusions.

2 Foundations of Spatial Audio

In this chapter, the necessary foundations in spatial audio will be introduced. First we will take a short look at the history of spatial audio with a focus on Ambisonics. Then we will talk about some basics of how the auditory system can determine the location of sound sources. Finally, we will introduce useful perceptual properties of auditory events and estimators for them.

Spatial audio refers to methods and technologies that allow us to create immersive sound experiences for listeners. The goal is to replicate the way humans perceive sound from different directions and distances in the real world. Thus, in comparison to more traditional audio systems, which focus on left and right like stereo or additionally on front and back like surround sound, spatial audio tries to recreate the whole 3D sound experience.

This is achieved through various techniques taking advantage of psychoacoustic principles and the way humans perceive and interpret sound to create the illusion of sound sources in space.

2.1 History of Spatial Audio

The idea to spatialize audio is nearly as old as the first audio recording and replay technologies. However, the first stage of spatial audio was only developed in the 1930s in the form of stereophonic sound. This allowed for the perception of sound coming from left and right beyond the original mono recordings [Xie20].

The next great step were the quadraphonic technologies from the 1970s. The aim was to create a 360 degree sound field using four loudspeakers. Many different systems were developed but in the end, the technology was not financially successful, thus becoming more of a gimmick for enthusiasts than something for a mainstream audience. Nevertheless, it still laid the foundation for what became the modern surround sound systems (e.g. 5.1, 7.1) [Xie20].

However, Ambisonics also came out of these developments. FOA was invented by Michael Gerzon in 1975 [Ger75]. Ambisonics focuses on capturing and reproducing sound as a spherical sound field. It aims to provide accurate sound representation in three dimensions, allowing for a flexible and immersive listening experience. Similar to the other technologies it never became a commercial success [Art15]. Since Ambionics has many advantages, research into Ambionics continued and finally, led to the development of HOA in the 1990s [ZF19].

Unlike stereo and more traditional multichannel surround sound, Ambisonics does not assume that the loudspeakers are in fixed positions. This means the encoding is completely layout-independent. Additionally, it isn't object based, meaning it doesn't encode the location and other properties of audio objects [Art15].

With the rise of XR, spatial audio techniques have become much more prevalent and are the topic of research. Ambisonics has become the spatial audio technique of choice for VR as the number of commercial and open source plugins integrating especially FOA into existing systems is rising [Art15].

2.2 Spatial Hearing

Spatial hearing refers to the ability of the human auditory system to perceive and interpret auditory cues to localize sound sources in space. It is the process which our brains use to determine direction and distance to sounds in relation to our own position.

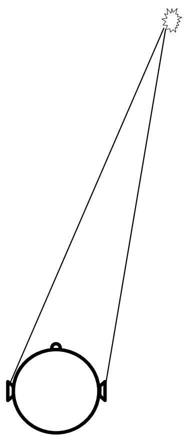
The human ears are positioned on the left and right side of our heads. Thus, in each ear there will be differences in time of arrival and sound pressure of sounds from the same source. Our brains use these differences as cues for lateral localization [Iid19]. These cues can be categorized into two main types: Interaural Time Differences (ITD) and Interaural Level Differences (ILD).

ITD are the difference in time of arrival of a sound between our ears. A sound coming from the front will arrive at both ears without time difference. However, a sound coming from slightly to the right will arrive a bit earlier at the right ear (see Figure 2.1a). This time difference increases until the sound comes directly from the right, where the sound will arrive about 1 ms earlier at the right ear than the left ear [Iid19]. If we continued rotating the sound source around us, the time difference gets smaller again as the sound starts coming more from behind. Again, there is no difference for a sound coming from directly behind. Even if only given ITD cues it is possible for humans to localize a sound [Bla97].

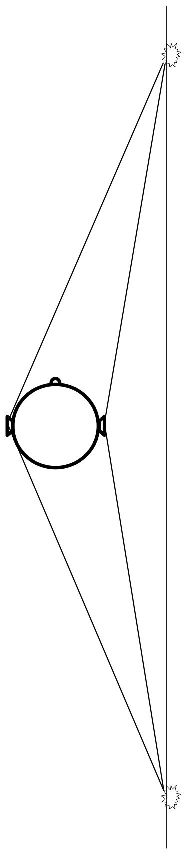
ILD are the differences in loudness or sound intensity between our two ears. This behaves quite similar to ITD. Again a sound coming from the front will be perceived equally loud in both ears, but a sound coming from the left side will be louder in the left ear than the right ear. This difference in loudness depends not only on the direction of the sound source but also on the frequency of the sound. For different frequencies the difference in loudness will vary [Iid19]. Again it is possible for humans to localize a sound given only ILD cues [Bla97].

If we perceived only ITD and ILD cues we would have a problem known as the cone of confusion. The interaural differences only give us the sagittal plane (i.e. the position on the left-right axis in relation to our body) in which the sound source is located [Iid19]. But without further information it is impossible to tell if as sound is to the front, back, up or down. This cone of confusion from which all interaural differences are the same (see Figure 2.1b) As humans can resolve this confusion, there must be more than interaural differences that allow us to localize sounds.

The sound waves that arrive at our ear canal are affected by our head, torso and pinnae. They act as acoustic filters changing the sound pressure depending on the frequency of the arriving sound at our ear canal. We call this total filtering effect the Head-related Transfer Function (HRTF). However, HRTFs are not only dependent on the frequency of the sound but also its relative position to us. Sounds from different angles and distances are filtered differently. Thus, we can describe a HRTF as a function $H(\varphi, \vartheta, r, f)$ where φ, ϑ, r, f are the azimuth angle, elevation angle, distance and frequency of the source in comparison to the listener respectively. Often we obtain the HRTF through a Fourier transform of their time domain counterpart the Head-related Impulse Response



(a) Example for one sound source.



(b) Cone of confusion for sound in the horizontal plane.

Figure 2.1: Sound arriving at both ears at different times in the horizontal plane.

(HRIR) $h(\varphi, \vartheta, r, t)$ with time t. The HRIR can be measured using specialized microphones. The best method is, to measure the IR to different sounds from different angles in an anechoic chamber to capture the unique acoustic characteristics[Iid19].

As the HRTF depends on properties of our body, not only different people, but even both ears have different HRTFs. While there are generic HRTFs, which are gained through averaging or simulation, a personalized HRTF leads to the best listening experince for oneself [Iid19]. Many applications still use generic HRTFs, as personalization for every user is usually not feasible and generic HRTFs can still be useful. The quality of these HRTFs can vary wildly from listener to listener. If the generic HRTF is similar to the one of the listener, they might be able to localize as well as with their own, but if they are very different, it might be very hard to localize sounds [Iid19].

Let $s(t)$ be a signal at a position defined by φ, ϑ and r and $S(f) = \mathcal{F}(s(t))$ its frequency domain counterpart. Then, to apply an HRTF H to this signal there are two possibilities [Iid19]. Either apply the HRTF by multiplication in the frequency domain

$$Y(f) = H(\varphi, \vartheta, r, f)S(f) \tag{2.1}$$

and then transform $Y(f)$ back into the time domain by inverse Fourier transform, or apply the HRIR by convolution directly in the time domain

$$y(t) = h(\varphi, \vartheta, r, t) * s(t). \tag{2.2}$$

Even under ideal conditions, the localization accuracy for most sound signals coming from the forward direction is between 1° and 10° depending on the signal type [Bla97]. For signals originating from the back this accuracy is even lower and the worst accuracy is for signals coming from the left or the right. But the ability to turn the head around and listen from different angles improves the localization quality [Bla97].

2.3 Perceptual Properties of Auditory Events of Multi-loudspeaker Playback

In this section, we will take a look at perceptual properties of auditory events. Auditory events are the sound images created by distributing a signal on loudspeakers with different amplitudes [ZF19]. These differences in amplitude are what is used in amplitude panning methods to localize the sound images in terms of direction and width. The perceptual properties which we introduce and their corresponding estimators are based on listening experiments. These properties and estimators will allow us to talk about the quality of Ambisonics decoding introduced in the next chapter [Ger92; ZF19].

Loudness When a signal is distributed over multiple loudspeakers, each loudspeaker contributes to the perceived loudness of the listener. Depending on the distance of the individual loudspeakers to the listener and the acoustic properties of the room this interference can be constructive or stochastically constructive. Constructive interference happens if all loudspeakers are equalized and located at the same distance to the listener. In this we can expect the perceived loudness to be [ZF19]

$$P = \sum_{l=1}^{L} g_l \tag{2.3}$$

where L is the number of loudspeakers and g_l is the amplitude gain of loudspeaker l.

In most cases these assumptions are not true as the room is not anechoic, the distances are not all equal etc. In such cases it is better to assume that the interference is stochastically constructive. Then it can be expected that the squared perceived loudness is [ZF19]

$$E = \sum_{l=1}^{L} g_l^2. \tag{2.4}$$

We will work with E as a measure of the loudness.

Direction The auditory event of the distributed signal is perceived to be in a direction. If only one loudspeaker is active this perceived direction is the same as that loudspeaker's direction. For more loudspeakers the perceived direction more complicated. For two loudspeakers with direction vectors $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ with gains g_1, g_2, we could model the perceived direction linearly as [ZF19]

$$\mathbf{r}_V = \frac{g_1\boldsymbol{\theta}_1 + g_2\boldsymbol{\theta}_2}{g_1 + g_2}. \tag{2.5}$$

For equal gains $g_1 = g_2$ this leads to $\mathbf{r}_V = \frac{1}{2}\boldsymbol{\theta}_1 + \frac{1}{2}\boldsymbol{\theta}_2$, the sound source being perceived from the point between both loudspeakers, and for $g_1 = 0$ ($g_2 = 0$) to $\mathbf{r}_V = \boldsymbol{\theta}_2$ ($\mathbf{r}_V = \boldsymbol{\theta}_1$), the sound source being perceived at the position of the active loudspeaker. As shown through listening experiments, in many cases using squared gains leads to better results. This gives us [ZF19]

$$\mathbf{r}_E = \frac{g_1^2\boldsymbol{\theta}_1 + g_2^2\boldsymbol{\theta}_2}{g_1^2 + g_2^2}. \tag{2.6}$$

If we extend this model for the perceived direction to L loudspeakers we get [ZF19]

$$\mathbf{r}_E = \frac{\sum_{l=1}^{L} g_l^2\boldsymbol{\theta}_l}{\sum_{l=1}^{L} g_l^2}. = \frac{\sum_{l=1}^{L} g_l^2\boldsymbol{\theta}_l}{E} \tag{2.7}$$

This is the model for perceived direction that will be used in this thesis.

Width Lastly, the auditory event has a perceived width. If it is produced by a single active loudspeaker we will only perceive the sound from that direction. However, for multiple active loudspeakers we will perceive sound from all of them. While the resulting auditory event only has one perceived direction, it is perceived as having a greater width than if only one loudspeaker had produced the auditory event in the perceived direction. In many practical cases it is desirable to have a narrow perceived width or at least a relatively constant perceived width. A good measure for the perceived width is the length r_E of $\mathbf{r}_E$ [ZF19]. The length is always between 0 for an unclear direction (e.g. two loudspeakers with opposite directions to the listener and equal gain) and 1 for only one active loudspeaker. While this estimator does not give us the width directly, it is a good measure nonetheless. If an estimator for the perceived width itself is preferred, it has been shown that [ZF19]

$$ASW = \frac{5}{8}\frac{180°}{\pi} 2 \arccos||\mathbf{r}_E|| \tag{2.8}$$

is quite accurate.

3 Ambisonics

The following chapter will introduce Ambisonics. Ambisonics is a surround sound format that encodes the sound field at a point. First, the circular/spherical coordinate system will be introduced as it is used in Ambisonics. After this, we will introduce circular/spherical harmonics series, which allow us to perfectly encode the sound field and are the mathematical basis of Ambisonics. Then, we will take a look at the Ambisonics representation itself, as well as some Ambisonics formats. After this, we will show how a single source can be encoded into the sound field, which is also called panning. Next, we will show how multiple sources can be encoded. Finally, we will talk about decoding Ambisonics to a loudspeaker array. For more information on Ambisonics, see [ZF19], who gives a good foundation for Ambisonics.

The general goal behind Ambisonics is to find a suitable approximation of the sound field $p(x, y, z, t)$ around a point $\mathbf{x} = (x_0, y_0, z_0)$. In Ambisonics this is achieved by combining the directional derivatives of the sound field p at $\mathbf{x}$ in a some ways similar to the Taylor expansion. This leads to a representation based on encoding the Direction of Arrival (DOA) of sounds. As such we think of the sound field as a function in spherical coordinates $p(\varphi, \vartheta, t)$. We will see that this spherical representation is based on spherical harmonics and is related to the spherical Fourier transform [Raf15; ZF19].

For our circular microphone array we only need 2D Ambisonics. Thus, 2D Ambisonics will be the focus of this chapter. Still, since in practice most of the plugins we will use later on are based on 3D Ambisonics, these will also be briefly presented in less detail.

Additionally, from now on we will ignore the time component t of the sound field as Ambisonics encodes each time frame separately. Hence, we will talk about functions only in φ in the 2D case or in (φ, ϑ) in the 3D case.

3.1 Functions on the Circle/Sphere

Point $\mathbf{x} = [x, y, z]^T \in \mathbb{R}^3$ in the Cartesian coordinate system can be represented in spherical coordinates by radius, azimuth angle and elevation angle as $\mathbf{r} = [r, \varphi, \vartheta]^T$ with [Raf15]

$$\begin{aligned} x &= r\sin(\vartheta)\cos(\varphi) \\ y &= r\sin(\vartheta)\sin(\varphi) \\ z &= r\cos(\vartheta) \end{aligned} \tag{3.1}$$

shown in Figure 3.1. Similarly $\mathbf{x} = [x, y]^T \in \mathbb{R}^2$ can be represented in circular coordinates by radius and azimuth angle as $\mathbf{r} = [r, \varphi]^T$ with

$$\begin{aligned} x &= r\cos(\varphi) \\ y &= r\sin(\varphi). \end{aligned} \tag{3.2}$$

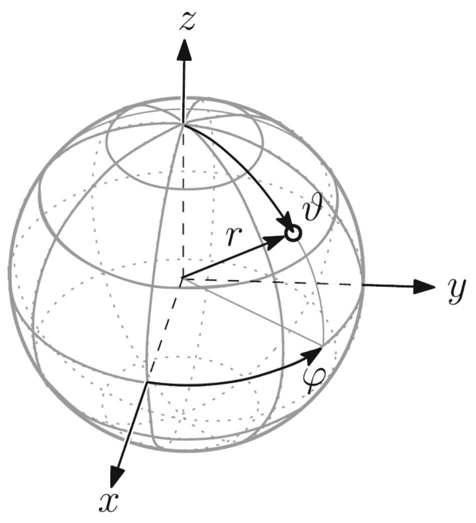


Figure 3.1: The spherical coordinate system. Without change from [ZF19] under CC BY 4.0.

This representation is quite useful for circular and spherical functions, as they are defined over the surface of the unit circle and the surface of the unit sphere respectively. In the Cartesian coordinate system

$$S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : ||\mathbf{x}|| = 1\} \tag{3.3}$$

represents the surface of the unit sphere for $d = 3$ and the surface of the unit circle for $d = 2$. Thus, a spherical function f in Cartesian coordinates can be written as [Raf15]

$$f(\boldsymbol{\theta}), \boldsymbol{\theta} = [\sin(\vartheta)\cos(\varphi), \sin(\vartheta)\sin(\varphi), \cos(\vartheta)]^T \in S^2. \tag{3.4}$$

However, in spherical coordinates we can write f just in terms of azimuth and elevation angle as

$$f(\varphi, \vartheta) = f(\boldsymbol{\phi}) \tag{3.5}$$

with $\boldsymbol{\phi} = (\varphi, \vartheta)$ as one parameter.

Similarly, for circular function f in Cartesian coordinates we get

$$f(\boldsymbol{\theta}), \boldsymbol{\theta} = [\cos(\varphi), \sin(\varphi)]^T \in S^1. \tag{3.6}$$

and $f(\varphi)$ only in the azimuth angle in circular coordinates.

3.2 Circular/Spherical Harmonics Series

In this section we will show that any square-integrable circular function $f(\varphi)$ can be represented as a weighted sum of basis functions [ZF19]

$$f(\varphi) = \sum_{m=-\infty}^{\infty} \gamma_m \Phi_m(\varphi) \tag{3.7}$$

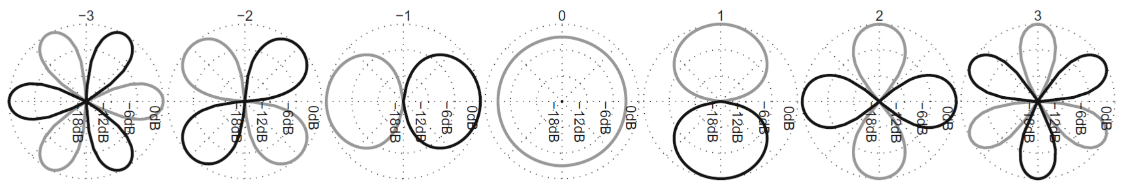


Figure 3.2: Circular harmonics with $m = -3, \dots, 3$ plotted as polar diagram using the radius $R = 20\lg|\sqrt{\pi}\Phi_m|$ and grayscale to distinguish between positive (grey) and negative (black) signs. Without change from [ZF19] under CC BY 4.0.

where $\Phi_m(\varphi)$ are the circular harmonics

$$\Phi_m(\varphi) = \frac{1}{\sqrt{2\pi}} \begin{cases} \sqrt{2}\sin(|m|\varphi), & \text{for } m < 0 \\ 1, & \text{for } m = 0 \\ \sqrt{2}\cos(m\varphi), & \text{for } m > 0 \end{cases} \tag{3.8}$$

and

$$\gamma_m = \int_{-\pi}^{\pi} f(\varphi)\Phi_m(\varphi)d\varphi. \tag{3.9}$$

We will call this the circular harmonics series. The circular harmonics with $m = -3, \dots, 3$ are shown in Figure 3.2.

There are different possibilities on how to introduce this representation. We will use the Fourier series to derive the circular harmonics series in 2D.

We know any square-integrable circular function $f(\varphi)$ can be written as a Fourier series [DB08]

$$f(\varphi) = a_0 + \sum_{m=1}^{\infty} a_m \cos(m\varphi) + b_m sin(m\varphi) \tag{3.10}$$

with coefficients given by

$$a_0 = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(\varphi)d\varphi \tag{3.11}$$

$$a_m = \frac{1}{\pi}\int_{-\pi}^{\pi} f(\varphi)\cos(m\varphi)d\varphi, \qquad \text{for } m > 1 \tag{3.12}$$

$$b_m = \frac{1}{\pi}\int_{-\pi}^{\pi} f(\varphi)\sin(m\varphi)d\varphi, \qquad \text{for } m < 1. \tag{3.13}$$

Now we rewrite Equation (3.10) as

$$f(\varphi) = \sum_{m=-\infty}^{\infty} a_m\tilde{\Phi}_m(\varphi) \tag{3.14}$$

with $\tilde{\Phi}_0 = 1$, $\tilde{\Phi}_m = \cos(m\varphi)$ for $m > 0$, $\tilde{\Phi}_m = \sin(|m|\varphi)$ for $m < 0$ and $a_m = b_{-m}$ for $m < 0$.

Finally, while the $\tilde{\Phi}_m$ are orthogonal to each other, since none of the products produce a constant component unless $i = j$, they are not orthonormal [ZF19]. For $i = j$ we have

$$\int_{-\pi}^{\pi} \tilde{\Phi}_0^2 d\varphi = \int_{-\pi}^{\pi} 1^2 d\varphi = 2\pi \tag{3.15}$$

$$\int_{-\pi}^{\pi} \tilde{\Phi}_i^2 d\varphi = \int_{-\pi}^{\pi} \cos(i\varphi)^2 d\varphi = \pi, \qquad \text{for } i > 0 \tag{3.16}$$

$$\int_{-\pi}^{\pi} \tilde{\Phi}_i^2 d\varphi = \int_{-\pi}^{\pi} \sin(|i|\varphi)^2 d\varphi = \pi, \qquad \text{for } i < 0. \tag{3.17}$$

To get orthonormal basis functions we normalize $\tilde{\Phi}_0$ by $\frac{1}{\sqrt{2\pi}}$, $\tilde{\Phi}_m$ for $m \neq 0$ by $\frac{1}{\sqrt{\pi}}$ and scale the coefficients a_m accordingly. This leads us to the circular harmonics Φ_m and the corresponding coefficients γ_m that we want. Thus, in 2D, representing a circular function as a circular harmonics series is nothing else but a scaled Fourier series representation.

The 3D case looks quite similar. Any square-integrable spherical function $f(\boldsymbol{\phi}) = f(\varphi, \vartheta)$ can be written as a weighted sum of basis functions [ZF19]

$$f(\boldsymbol{\phi}) = f(\varphi, \vartheta) = \sum_{n=0}^{\infty} \sum_{m=-n}^{m} \gamma_{nm} Y_n^m(\varphi, \vartheta) \tag{3.18}$$

where the Y_n^m are the spherical harmonics

$$Y_n^m(\boldsymbol{\phi}) = Y_n^m(\varphi, \vartheta) = \Theta_n^m(\vartheta)\Phi_m(\varphi) \tag{3.19}$$

and

$$\gamma_{nm} = \int_{-\pi}^{\pi} \int_{0}^{\pi} f(\varphi, \vartheta) Y_n^m(\varphi, \vartheta). \tag{3.20}$$

The spherical harmonics for $0 \leq n \leq 3$ and $-n \leq m \leq n$ are shown in Figure 3.3. The Φ_m are the already established circular harmonics and

$$\Theta_n^m(\varphi) = N_n^{|m|} P_n^{|m|}(cos(\vartheta)) \tag{3.21}$$

where P_n^m are the associated Legendre function and N_n^m the corresponding normalization term to ensure orthonormality. This normalization is called N3D normalization and defines each harmonic except for an arbitrary phase shift. Ambisonics prefers real valued functions and in practice, the SN3D normalization $N_{n,SN3D}^m = \sqrt{(2 - \delta_{0,n}) \frac{(m-n!)}{(m+n)!}}$ is often used for its useful property that no component will ever exceed the peak value of the $n = 0, m = 0$ component [Dan03; ZF19].

Again, this representation is closely related to the Fourier series of a spherical function (cf. [Raf15]).

As mentioned before, there are different ways to derive the circular/spherical harmonics. In [ZF19], a slightly different approach is used, coming more from the viewpoint of loudspeakers and microphones using the definition of harmonic functions. [Raf15] has a more general look at the spherical Fourier transform, which has complex-valued spherical harmonics. Additionally, see [Dic+03] for further reading, which introduces different sound field representations and compares them.

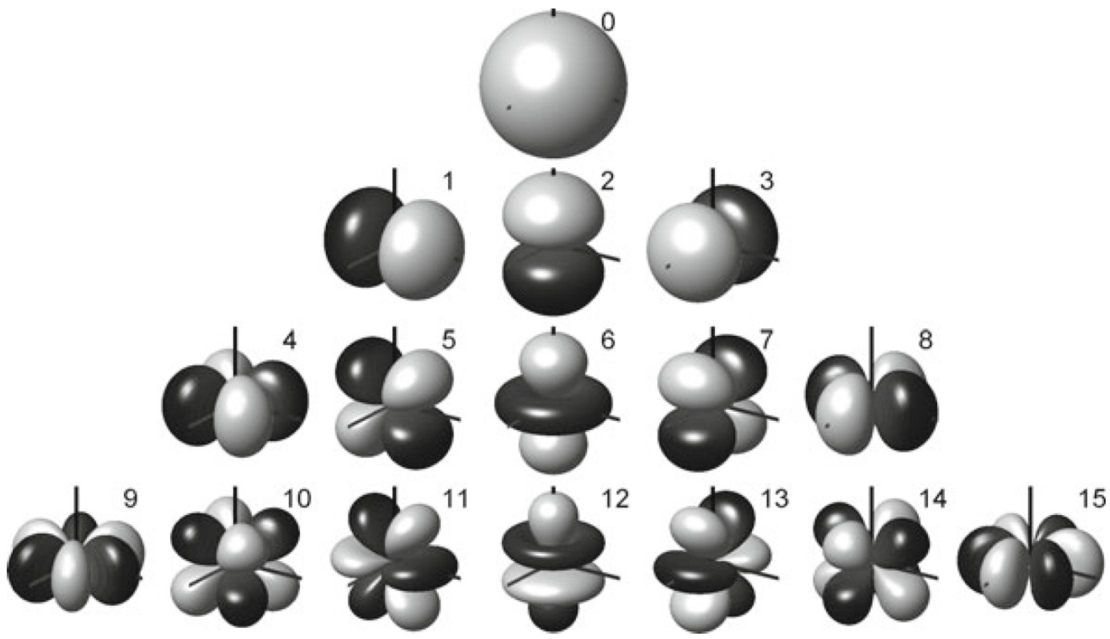


Figure 3.3: Spherical harmonics for $0 \leq n \leq 3$ and $-n \leq m \leq n$. Indexed by ACN channel numbering (see Equation (3.24)). What is plotted is a polar diagram with radius $R = 20\lg|Y_n^m|$ normalized to the upper 30 dB of each pattern, with positive (grey) and negative (black) color indicating the sign. Without change from [ZF19] under CC BY 4.0.

3.3 Ambisonics Representation

In this section we will introduce the Ambisonics representation which uses the circular/spherical harmonics series introduced in the last section to get an approximation of the sound field p with a finite amount of channels. Then we will discuss some Ambisonics formats used in practice.

In both 2D and 3D, the idea to derive the Ambisonics representation is the same. The higher frequency components are truncated leading to a loss in spatial resolution while still giving a good approximation. This leads to [ZF19]

$$p_N(\varphi) = \sum_{m=-N}^{N} \gamma_m \Phi_m(\varphi) \tag{3.22}$$

in the 2D case or [ZF19]

$$p_N(\varphi, \vartheta) = \sum_{n=0}^{N} \sum_{m=-n}^{m} \gamma_{nm} Y_n^m(\varphi, \vartheta) \tag{3.23}$$

in the 3D case. Depending on N we call this N-th Order Ambisonics.

The simplest form of Ambisonics are Zeroth Order Ambisonics. There is only one channel (often called W) that encodes the sound pressure of the sound field at $\mathbf{x}$. This leads to a complete loss of DOA.

The most common form of Ambisonics is FOA with 3 (or 4 for 3D) channels. Additionally to the sound pressure, the pressure gradients along the axes of the Cartesian coordinate system are encoded. These further channels are often called X and Y (and Z).

For $N > 1$, the so called HOA encode higher order derivatives of the sound pressure for a better spatial resolution of the sound field. For N-th Order Ambisonics there are $2N + 1$ channels in 2D and $(N + 1)^2$ channels in 3D.

The most widespread way to number the channels in 3D is the Ambisonic channel number (ACN). In the ACN the cannel number is calculated as follows [ZF19]

$$ACN = n^2 + n + m. \tag{3.24}$$

In practice there are competing standards for 3D Ambisonic with different normalization (e.g. N3D and SN3D) and channel numbering, but most can easily be transformed into each other. Some formats are the proposed AmbiX standard (with SN3D and ACN) [NZDS11], Furse-Malham higher order format and the MPEG-H 3D Audio standard [HHKP15]. 2D Ambisonics is even less standardized as it is not used that much in practice and 3D Ambisonics can just be used to represent 2D Ambisonics (cf. Section 4.2).

At this point it should be noted that there are microphones that can record directly into an 3D Ambionics format. Most of these are for recording FOA only, but microphones recording in HOA exist. While some of them record the channels as they are with an omnidirectional and three figure-of-eight microphones (called the B-format), most record in the so-called A-format using four cardioid microphones. This makes the microphone design easier as the individual microphones influence each other much less. The A-format can be directly transformed into the B-format [ZF19]. Most manufacturers provide software for this transformation for their microphones. Examples for Ambisonics microphones are the *NT-SF1* by RØDE, which records in FOA, or for HOA the *ZYLIA ZM1*, which records 3rd Order Ambisonics.

In our case, the recording was not done with such a microphone. As a result, there is further need to encode the recordings into the Ambisonics format. The next sections will provide the foundation for such an encoding.

3.4 Panning a Source in Ambisonics

This section shows how a single audio source placed at an arbitrary angle φ_s can be encoded into Ambisonics. This is called Ambisonic panning.

In 2D, we represent our infinitely narrow audio point source s positioned around φ_s by a Dirac delta distribution [ZF19]

$$s(\varphi) = \delta(\varphi - \varphi_s) = \begin{cases} \lim_{\varepsilon \to 0} \frac{1}{2\varepsilon}, & \text{for } |\varphi - \varphi_s| \leq \varepsilon \\ 0, & \text{otherwise.} \end{cases} \tag{3.25}$$

Using the transformation integral we get the coefficients

$$\begin{aligned}\gamma_m &= \int_{-\pi}^{\pi} \delta(\varphi - \varphi_s) d\varphi = \Phi_m(\phi_s) \lim_{\varepsilon \to 0} \int_{-\varepsilon}^{\varepsilon} \frac{1}{2\varepsilon} d\varphi \\ &= \Phi_m(\phi_s)\end{aligned} \tag{3.26}$$

Thus, for infinite order Ambisonics we have

$$s(\varphi) = \delta(\varphi - \varphi_s) = \sum_{m=-\infty}^{\infty} \gamma_m \Phi_m(\varphi) = \sum_{m=-\infty}^{\infty} \Phi_m(\phi_s)\Phi_m(\varphi) \tag{3.27}$$

but for finite order N we only get an approximation

$$s_N(\varphi) = \sum_{m=-N}^{N} \Phi_m(\phi_s)\Phi_m(\varphi). \tag{3.28}$$

If we look at the quality of this approximation we can see that so-called side lobes are created. These side lobes increase the perceived width of our sound source.

It should be noted that these side lobes can be designed by adding additional weights a_m

$$s_N(\varphi) = \sum_{m=-N}^{N} a_m \Phi_m(\phi_s)\Phi_m(\varphi). \tag{3.29}$$

There are different goals that we can try to achieve in designing the side lobes. Here, we will present the max-r_E weights first introduced by [DRP99]. The goal of these weights is to maximize the length r_E of the vector $\mathbf{r}_E$, i.e. minimize the perceived width of the panning function.

This leads to the weights [ZF19]

$$a_m = \cos\left(\frac{\pi m}{2(N+1)}\right). \tag{3.30}$$

The side lobes for basic weights $a_m = 1$ and max-r_E weights can be seen in Figure 3.4.

Another useful fact is that we can rewrite the panning function of a single source s at φ_s as a sum of Chebyshev polynomials T_m[ZF19]

$$s_N(\varphi) = \sum_{m=0}^{N} c_m T_m(\cos(\varphi - \varphi_s)) = \sum_{m=0}^{N} c_m T_m(\cos(\phi)) \tag{3.31}$$

where $\phi = \varphi - \varphi_s$.

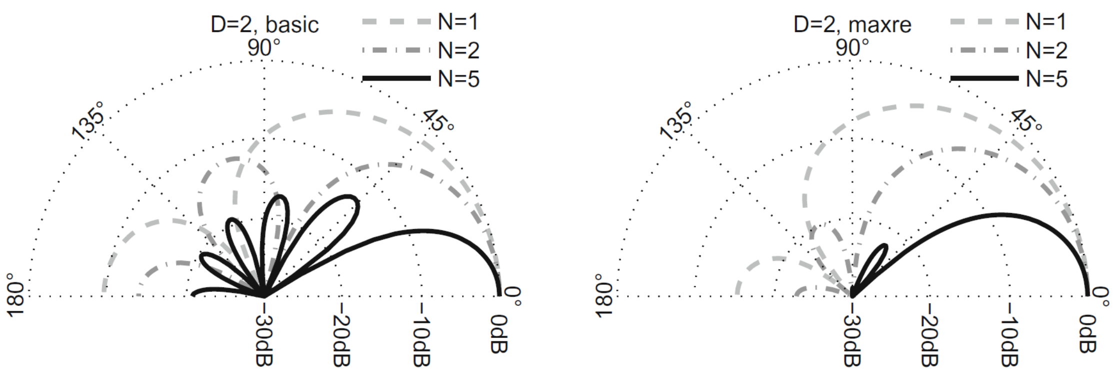


Figure 3.4: Side lobes for 2D Ambisonics for unweighted $a_n = 1$ (left) and weighted max-r_E (right) Ambisonics functions for the order $N = 1, 2, 5$. Without change from [ZF19] under CC BY 4.0.

This uses the fact that $T_m(\cos(\phi)) = \cos(m\phi)$ and that the panning function can also be written as a Fourier series [Han02].

$$s_N(\varphi) = \sum_{m=0}^{N} c_m T_m(\cos(\varphi - \varphi_s)) = \sum_{m=0}^{N} c_m \cos\left[m(\varphi - \varphi_s)\right] \tag{3.32}$$

$$= \sum_{m=0}^{N} c_m \left[\cos(m\varphi)\cos(m\varphi_s) + \sin(m\varphi)\sin(m\varphi_s)\right] \tag{3.33}$$

$$= \sum_{m=0}^{N} c_m \left[\tilde{a}_m \cos(m\varphi) + \tilde{b}_m \sin(m\varphi)\right] \tag{3.34}$$

$$= \sum_{m=0}^{N} a_m \cos(m\varphi) + b_m \sin(m\varphi) \tag{3.35}$$

With $\tilde{a}_m = \cos(m\varphi_s)$, $a_m = c_m\tilde{a}_m$, $\tilde{b}_m = \sin(m\varphi_s)$ and $b_m = c_m\tilde{b}_m$.

Being able to write the panning function only dependent on ϕ also gives us information about the perceived loudness, direction and width of the panned source. In the continuous case the perceived loudness E is the integral $\int_{-\pi}^{\pi} s_N(\phi)^2 d\varphi$. But if the value of the panning function only depends on ϕ, the perceived loudness is always the same, independent of φ_s. Thus, the loudness is panning invariant for the continuous panning function. Similarly, it follows that $\mathbf{r}_E$ points perfectly in direction φ_s as the continuous panning function is symmetrical around φ_s and has its maximum in direction $\mathbf{r}_E$. Additionally, we also have panning invariant perceived width. If we rotate the Cartesian coordinate system in such a way that the x-axis points in the direction of φ_s the y-component of $\mathbf{r}_E$ becomes 0. The x-component (and with this $||\mathbf{r}_E||$) can then be calculated as an integral [ZF19]

$$||\mathbf{r}_E|| = r_{E,x} = \frac{\int_{-\pi}^{\pi} s_N(\phi)^2 \cos(\phi) d\varphi}{E} \tag{3.36}$$

meaning that it also only depends on ϕ and the perceived width is panning invariant. These invariances, as well perfect directionality, are both useful properties of Ambisonics.

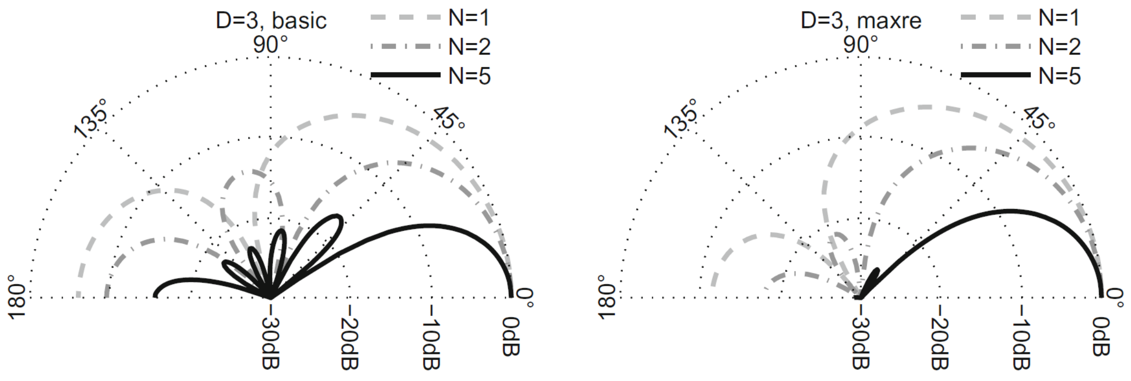


Figure 3.5: Side lobes for 3D Ambisonics for unweighted $a_n = 1$ (left) and weighted max-r_E (right) Ambisonics functions for the order $N = 1, 2, 5$. Without change from [ZF19] under CC BY 4.0.

In 3D, we again have a very similar case if we want to encode an infinitely narrow audio source s positioned around (φ_s, ϑ_s). For such a source we get [ZF19]

$$s_N(\varphi, \vartheta) = \sum_{n=0}^{N} \sum_{m=-n}^{n} Y_n^m(\varphi_s, \vartheta_s) Y_n^m(\varphi, \vartheta). \tag{3.37}$$

This again leads to side lobes which can be designed by weights a_n

$$s_N(\varphi, \vartheta) = \sum_{n=0}^{N} \sum_{m=-n}^{n} a_n Y_n^m(\varphi_s, \vartheta_s) Y_n^m(\varphi, \vartheta). \tag{3.38}$$

If the max-r_E panning function is used this leads to

$$a_n = P_n \left(\cos \left(\frac{137.9}{N + 1.51} \right) \right) \tag{3.39}$$

where P_n are the Legendre polynomials. The side lobes for basic weights $a_m = 1$ and max-r_E weights can be seen in Figure 3.5.

Lastly, we can also write a 3D panning function of a single source s at (φ_s, ϑ_s) as a sum of polynomials. Unlike in 2D, when it comes to 3D, this representation is based on Legendre polynomials P_n and the angle ϕ between (φ_s, ϑ_s) and (φ, ϑ) [ZF19].

$$s_N(\phi) = \sum_{m=0}^{N} c_m P_m(\cos(\phi)) \tag{3.40}$$

This uses the spherical harmonics addition theorem $\sum_{m=-n}^{n} Y_n^m(\varphi_s, \vartheta_s) Y_n^m(\varphi, \vartheta) = \frac{2n+1}{4\pi} P_n(cos(\phi))$ [ZF19].

Since it is again possible to write the continuous panning function only depending on ϕ, in the 3D case the continuous panning function also has panning invariant perceived loudness and width with perfect perceived directionality of the sound source. The arguments are similar as in the 2D case.

3.5 Encoding Multiple Sources in Ambisonics

In the following section we will show how to encode multiple sources. There are different strategies to do this of which two will be presented.

Given L sound sources at angles $\{\varphi_l\}_{l=1,...,L}$ in 2D, the simplest way to encode them would be adding up the individual solutions as they represent the encoded sources. This leads to a resulting sound field $p(\varphi)$ of

$$p(\varphi) = \sum_{l=1}^{L} \left(\sum_{m=-N}^{N} \Phi_m(\phi_l)\Phi_m(\varphi) \right) = \sum_{m=-N}^{N} \left(\Phi_m(\varphi) \sum_{l=1}^{L} \Phi_m(\phi_l) \right) \tag{3.41}$$

This means in practice we can just add up the individual channels of the Ambisonic format over the different encoded sources [Art15].

However, instead of having multiple sound sources to encode, we have L samples of the same sound field $s(\varphi_l)$ taken at different angles $\{\varphi_l\}_{l=1,...,L}$. This means for $l = 1, \ldots, L$ we know

$$s(\varphi_l) = \sum_{m=-N}^{N} \gamma_m \Phi_m(\varphi_l) \tag{3.42}$$

for unknown coefficients γ_m. We can then use a minimum mean squared error (MMSE) method to calculate suitable coefficients γ_m. [ZF19]

$$\min||\mathbf{e}||^2 = \min \sum_{l=1}^{L} \left[s(\varphi_l) - \sum_{m=-N}^{N} \gamma_m \Phi_m(\varphi_l) \right]^2 = \min||\mathbf{s} - \mathbf{\Phi}_N^T \boldsymbol{\gamma}_N||^2 \tag{3.43}$$

with $\mathbf{s} = [s(\varphi_1), \ldots, \varphi_l]^T$, $\mathbf{\Phi}_N = [\mathbf{y}_N(\varphi_1), \ldots, \mathbf{y}_N(\varphi_L)]$ with $\mathbf{y}_N(\varphi_l) = [\Phi_{-N}(\varphi_l), \ldots, \Phi_N(\varphi_l)]^T$ and $\boldsymbol{\gamma}_N = [\gamma_{-N}, \ldots, \gamma_N]^T$.

To find the minimum we calculate the derivative

$$\frac{\partial}{\partial \gamma_N} \mathbf{e}^T \mathbf{e} = 2 \left(\frac{\partial \mathbf{e}}{\partial \gamma_N} \right)^T \mathbf{e} = 2\mathbf{\Phi}_N \mathbf{e} = 2\mathbf{\Phi}_N \mathbf{s} - 2\mathbf{\Phi}_N \mathbf{\Phi}_N^T \gamma_N. \tag{3.44}$$

By setting the derivative to zero we get

$$\gamma_N = \left(\mathbf{\Phi}_N \mathbf{\Phi}_N^T \right)^{-1} \mathbf{\Phi}_N \mathbf{s}. \tag{3.45}$$

With this we have a way to encode our recordings into Ambisonics.

In 3D we can do practically the same. For the simple encoder we just add up the individual channels. For the MMSE method the idea is also exactly the same, as well, and in the end we just get [ZF19]

$$\gamma_N = \left(\mathbf{Y}_N \mathbf{Y}_N^T \right)^{-1} \mathbf{Y}_N \mathbf{s}. \tag{3.46}$$

with $\boldsymbol{Y}_N = [\mathbf{y}_N(\boldsymbol{\phi}_1), \ldots, \mathbf{y}_N(\boldsymbol{\phi}_L)]$ with $\mathbf{y}_N(\varphi_l) = [Y_0^0(\boldsymbol{\phi}_l), Y_1^{-1}, \ldots, Y_N^N(\boldsymbol{\phi}_l)]^T$ and $\boldsymbol{\gamma}_N = [\gamma_{00}, \gamma_{1-1}, \ldots, \gamma_{NN}]^T$

3.6 Decoding to a Loudspeaker Array

In this section we will discuss Ambisonic Decoding. First, we will take a look at the simplest decoder, the sampling decoder. Then, we will show how we can optimally sample the panning function. Finally, we take a look at more advanced and flexible decoders.

The sampling decoder is quite intuitive. We sample the panning function s_N at the L different angles $\{\varphi_l\}_{l=1,...,L}$ at which our loudspeakers are placed. For each loudspeaker we get a sample $s_N(\varphi_l)$ of the panning function. We then assign each loudspeaker a signal $g_l = \sqrt{\frac{2\pi}{L}} s_N(\varphi_l)$. The additional weight $\sqrt{\frac{2\pi}{L}}$ comes from the fact that each loudspeaker contributes to the perceived loudness E on the circle of surrounding directions [ZF19].

For 3D we just need to adapt the weight to $\sqrt{\frac{4\pi}{L}}$ as we now operate on a sphere and 4π is the surface of the unit sphere.

It is easy to see that a bad choice of loudspeaker positions can lead to problems with perceived loudness E, perceived width $||r_E||$ and even correct perceived direction r_E. For example with a single loudspeaker the position of the loudspeaker will be the perceived direction of all sounds with width 1. Similarly, with only two loudspeakers in opposite directions a signal source placed orthogonal to both loudspeakers would lead to a perceived width of 0, thus losing all directionality. A solution to this problem is to look for loudspeaker positions that are optimal. For us this optimality means keeping the panning-invariant loudness E of the continuous panning function and a perfectly localized r_E with constant perceived width [ZF19].

These optimal loudspeaker layouts are the so-called t-designs. t-designs lead us to perfect quadrature rules for polynomials Q_m of degree $m \leq t$. A t-design in 2D is a set of L angles $\{\varphi_l\}_{l=1,...,L}$ with the property [ZF19]

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} Q_m(\cos(\phi))d\phi = \frac{1}{L}\sum_{l=1}^{L} Q_m(cos(\varphi_l - \varphi_l)) \tag{3.47}$$

with $\phi = \varphi - \varphi_s$ We already know that we can write the continuous panning function as a sum of Chebyshev polynomials (cf. Equation (3.31)). Each Chebyshev polynomial T_m is a polynomial of degree m and orthogonal to all other Chebyshev polynomials [Han02; ZF19]. Thus, the continuous panning function s_N is always a polynomial of degree N. Since the loudness E is an integral over s_N^2 (a polynomial of degree $2N$) in the continuous case, we need $t \geq 2N$ for perfect quadrature. But for r_E we integrate over $s_N^2 \boldsymbol{\phi}_s^T \boldsymbol{\phi} = s_N^2 \cos(\phi)$ (with $\boldsymbol{\phi}_s = [\cos(\varphi_s), \sin(\varphi_s)]^T, \boldsymbol{\phi} = [\cos(\varphi), \sin(\varphi)]^T$) a polynomial of degree $2N + 1$, and it follows we need $t \geq 2N + 1$.

Now for our optimal decoding we just need t-designs. In 2D this is quite simple as every regular polygon with L points is a t-design with $t = L - 1$ [ZF19]. This means we just use a regular polygon with $L \geq 2N + 2$ points as our loudspeaker layout. Possible corresponding angles are

$$\varphi_l = \frac{2\pi}{L}(l - 1) \tag{3.48}$$

for $l = 1, \ldots, L$.

As we can write the continuous panning function in 3D as a sum of Legendre polynomials P_m (cf. Equation (3.40)) of degree m, and considering Legendre polynomials are orthogonal as well, we also need t-designs with $t \geq 2N + 1$ in 3D. However, different from 2D, finding t-designs in 3D presents more of a challenge. While the five Platonic solids are t-designs, even the dodecahedron is only a 5-design. Thus, if we want to have optimal layouts for HOA of order ≥ 3, these layouts cannot be regular. Such irregular t-designs have been found by [HS02] and [GP11].

While optimal in theory, in practice t-designs are not always useful. In most situations, it is not possible to place the speakers exactly at the right locations. Thus, usually compromises have to be made that lead to the loss of optimality. For our use-case this is not a problem, as we place the sources in a virtual environment and have complete control over their exact position.

An especially useful method of decoding is All-Round Ambisonic Decoding (AllRAD) [ZF12]. It allows free placement of loudspeakers and is usable for HOA while trying to keep the perceived loudness and the perceived width as constant as possible. The idea behind AllRAD is to decode to a virtual t-design with large enough t as an intermediate step and then use Vector Based Amplitude Panning (VBAP) to map to the actual positions of our loudspeakers. In short, in VBAP a phantom source at position ϑ_s between three loudspeakers at positions $L = [\vartheta_1, \vartheta_2, \vartheta_3]$ is created through linear combination. This is means that the weights can be calculated through [ZF12; ZF19]

$$\vartheta_s = L\tilde{g} \Rightarrow \tilde{g} = L^{-1}\vartheta_s \tag{3.49}$$

and normalization to $g = \frac{\tilde{g}}{||\tilde{g}||}$. For more than three loudspeakers the convex hull of the loudspeakers is triangulated and the three loudspeakers between which the phantom source lies are chosen. This approach works best if the origin is in the convex hull. For more on VBAP see [Pul97].

The AllRAD approach combines the optimality of t-design decoding with the arbitrary choice of loudspeaker positions of VBAP. This leads to a flexible method that has close to optimal properties.

4 Ambisonics Encoding and Decoding in REAPER

This chapter is about the first part of the implementation which encodes the audio data into Ambisonics and decodes it to loudspeakers around the listener. This was done using the software REAPER with the IEM Plug-in Suite. Thus, in the first section we will take a general look at REAPER, the IEM Plug-in Suite and their functionalities. Next, the high level idea of the signal flow will be shown, and the choices made will be explained. In the last section, the implementation in REAPER will be presented in detail.

4.1 REAPER and the IEM Plug-in Suite

In the following section the software REAPER will be introduced with a focus on its functionalities. Rapid Environment for Audio Production, Engineering and Recording (REAPER) is a Digital Audio Workstation (DAW), a software for recording, editing and producing audio files [REAPER06]. It is a proprietary software but offers a free 60 day evaluation period that allows access to all features. There are two licenses identical in features available for purchase: a commercial one and a discounted license for private use, small businesses and educational use. At $60 the discounted license is relatively affordable for a DAW [Rad15]. REAPER is available for Microsoft Windows, macOS and Linux in 32-bit and 64-bit. It offers full multitrack audio processing with up to 64 channels per track. This allows for up to 7th order Ambisonics (which needs $(7+1)^2 = 64$ channels) to be in one track. Internally, the audio is processed at 64-bit and REAPER allows for import from and rendering to almost any bit depth and sample rate. Through this, there is no loss in audio quality for most high quality audio files. Additionally, REAPER supports third party plug-ins in many standards like VST. This allows for use of many specialized effects directly in REAPER which we will use for Ambisonics processing as there is no native Ambisonics support.

In our case, we use the IEM Plug-in Suite which are a VST2 plug-ins. The IEM Plug-in Suite is a free and Open-Source audio plug-in suite mainly for Ambisonics plug-ins up to 7th order Ambisonics. It was created by the Institute of Electronic Music and Acoustics [IEM17]. The Plug-in Suite is usable with most modern DAWs that have enough channels per track to support HOA, but it is suggested to be used with REAPER. There are compiled release versions available and it is possible to clone the repository and compile it yourself. The Plug-in Suite provides plug-ins for many different uses related to Ambisonics of which two are most important for us: the MultiEncoder and the AllRADecoder. Both will be presented in detail in Section 4.3. It is important to note that the IEM Plug-in Suite uses the AmbiX convention, meaning ACN channel ordering and SN3D normalization. N3D normalization is also available but discouraged.

4.2 What are the Goals of the Implementation?

Before we talk about the implementation in detail in the next section, in this section we will have a look at what exactly we want to implement and which additional decisions need to be made for this implementation.

First, let us recap our audio data situation. We have audio recorded by a circular microphone array with 8 microphones. This gives us 8 samples of the sound field on the horizontal plane. The number of samples limits which order of Ambisonics we can actually capture with the microphones. A circular microphone with 3 microphones is enough to capture 2D directly FOA [ZF19]. In general, for order N we need $L \geq 2N + 1$ microphones for 2D sound reproduction to capture the $2N + 1$ channels of N-th Order Ambionics [WFB11]. Thus, with our 8 microphones it does not make sense to use an Ambisonics order higher than 3.

Additionally, as the IEM Plug-in Suite only supports 3D Ambisonics, we will use 3D 3-rd Order Ambisonics instead of 2D. As we can see from $Equation$ (3.19), if we only look at the horizontal plane 3D Ambisonics will only hold information about the circular harmonics

$$\begin{aligned} p_N\left(\varphi, \frac{\pi}{2}\right) &= \sum_{n=0}^{N} \sum_{m=-n}^{m} \gamma_{nm} Y_n^m \left(\varphi, \frac{\pi}{2}\right) \\ &= \sum_{n=0}^{N} \sum_{m=-n}^{m} \gamma_{nm} N_n^{|m|} P_n^{|m|} \left(cos\left(\frac{\pi}{2}\right)\right) \Phi_m(\varphi) \\ &= \sum_{n=0}^{N} \sum_{m=-n}^{m} \gamma_{nm} \tilde{N}_n^{|m|} \Phi_m(\varphi) \\ &= \sum_{m=-N}^{N} \left(\sum_{n=|m|}^{N} \gamma_{nm} \tilde{N}_n^{|m|} \right) \Phi_m(\varphi) \\ &= \sum_{m=-N}^{N} \tilde{\gamma}_m \Phi_m(\varphi) \end{aligned} \tag{4.1}$$

with $\tilde{N}_n^{|m|} = N_n^{|m|} P_n^{|m|} \left(cos\left(\frac{\pi}{2}\right)\right)$ and $\tilde{\gamma}_m = \sum_{n=|m|}^{N} \gamma_{nm} \tilde{N}_n^{|m|}$.

Hence, encoding into 3D Ambisonics leads to a 2D Ambisonics encoding if we only look at the horizontal plane. Furthermore, there is no information in channels that only have information about the z-axis encoded as there are no changes on that axis (cf. Figure 4.8).

For encoding, we choose the optimal t-designs. Thus, for 3-rd Order Ambisonics we are looking for a 7-design. Even though we have 3D Ambisonics, we only care about the 2D Ambisonics component on the horizontal plane. Thus, there is no reason to add loudspeakers that are not on the horizontal plane, as they would just be scaled versions of the ones on the horizontal plane and influence the optimality of the 2D 7-design. Adding a loudspeaker below the listener might even be confusing as it would produce sound from below the ground under the listener (as there would be at least a contribution to the loudspeaker from the 0-th order component). If we were to place the loudspeakers in a real setting like a sound dome, there might still be a benefit to place loudspeakers above the listener, but as we place them in Unity and movement of the listener is thus impossible, there is no reason. Hence, we will decode to a 2D 7-design, ensuring the optimality without unnecessary loudspeakers. As regular polygons with $L = t + 1$ points are t-designs, an octagon layout fulfills the requirements.

To recap, we want to encode the 8 microphones into 3D 3-rd Order Ambisonics and then decode to an octagon layout of 8 loudspeakers on the horizontal plane.

4.3 Implementation in REAPER Using the IEM Plug-in Suite

In this section we will talk about the implementation in REAPER in detail. While the theory behind the encoding and decoding is complex, the actual implementation is much simpler thanks to the use of the IEM Plug-in Suite. The signal flow in REAPER can be seen in Figure 4.1. Every box corresponds to a track. The plug-ins applied on thos tracks, if any, are marked in italics. The routing between tracks is shown by arrows from the sending track to the receiving track. The number of channels sent is noted at each routing. The Input and Loudspeakers track do not apply any plug-ins and are mainly for inserting the audio input and for rendering the audio output. Additionally, the signal flow is a bit more complicated than necessary. Encoder and Decoder track are theoretically enough, but Visualizer and Headphones track are both useful for working on the project. The visualizer allows us to see the sound in the sound field and the binaural decoder allows us to listen to the finished product inside REAPER.

First let us take a high level look at the signal flow before looking at the steps in detail. The general idea of the signal flow is that we insert the 8 audio recordings on the 8 channels of the Input track. From there we just route the signal to the Encoder track. Here, the encoding into 3-rd Order Ambisonics happens through the *MultiEncoder* plug-in. From there the now 16 channels are routed to the Visualizer track that applies the *EnergyVisualizer* plug-in. This has no influence on the actual signal and is just a visualization of the current signal. Next the routing splits into two parallel flows. One is the necessary decoding path to the Decoder track, the other is an additional path to the Headphones track that allows us to listen to the encoded audio. The Decoder track decodes to the loudspeakers using the *AllRADecoder* plug-in. The 8 channels of the decoded signal are then routed to the Loudspeakers track from where they can be rendered. The Headphones track decodes the Ambisonics signal to a stereo signal with the *BinauralDecoder* plug-in and sends the result to the Master track. The Master track then plays the stereo signal to headphones connected to the computer.

Inside REAPER, this looks as shown in Figure 4.2. On the left and at the bottom we see the tracks and in middle we see the waveforms of the audio files (mainly empty as the audio files are only inserted in the subtracks of the Input track). A waveform is a graphical representation of an audio signal that shows its amplitude.

The routing in REAPER is quite simple: It can be done by interacting with the *Route* button on each track (or even through drag-and-drop). It can be specific to each channel of a track. The routing of the Visualizer track can be seen in Figure 4.3a. All channels are sent to both the Decoder and the Headphones track and all channels are received from the Encoder track. In the routing of the Decoder track shown in Figure 4.3b we can see that each channel is sent to a different track.

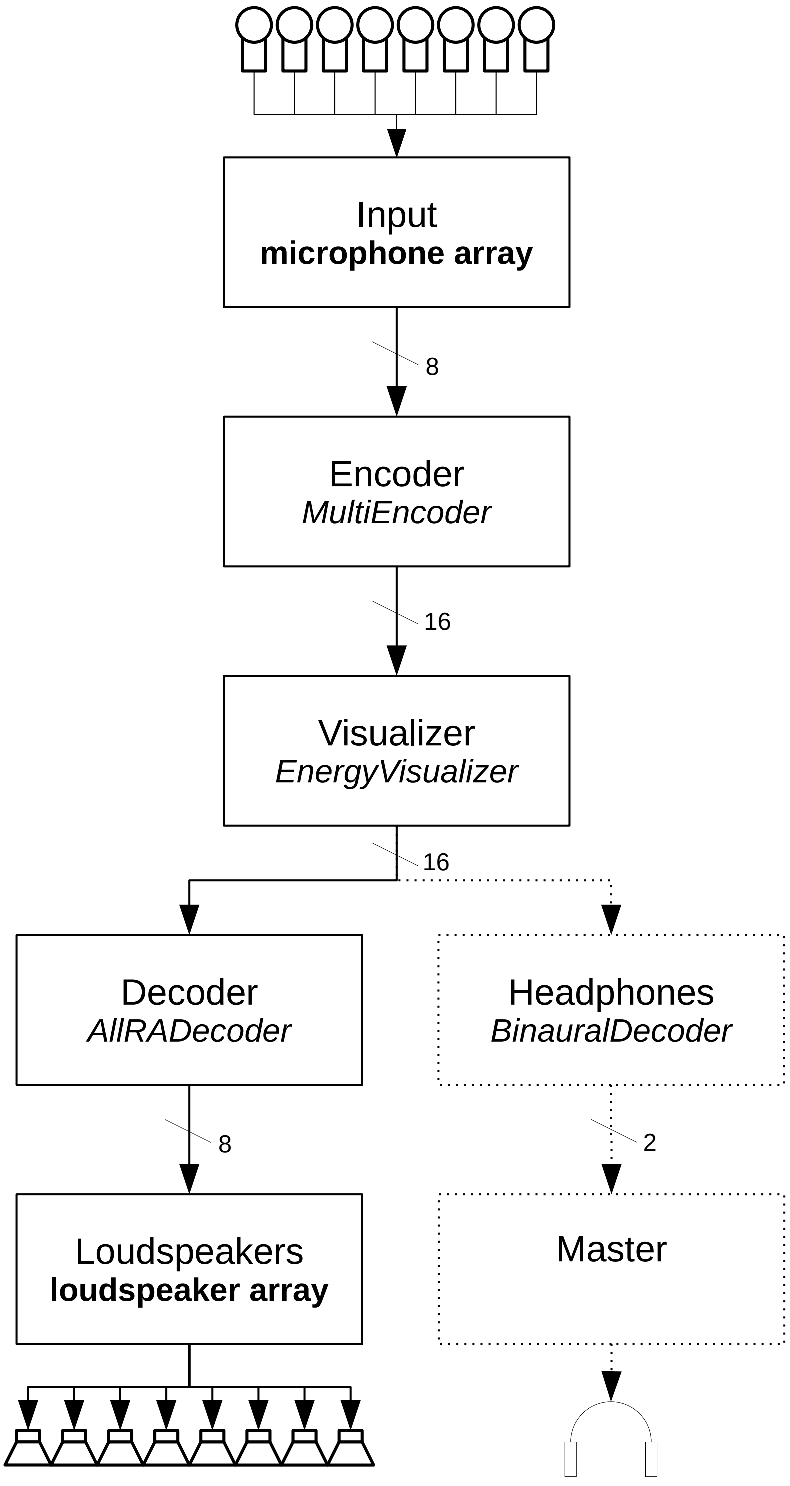


Figure 4.1: Signalflow in REAPER. Boxes represent tracks with applied plug-in in italics. Routing shown by arrows with number of channels noted.

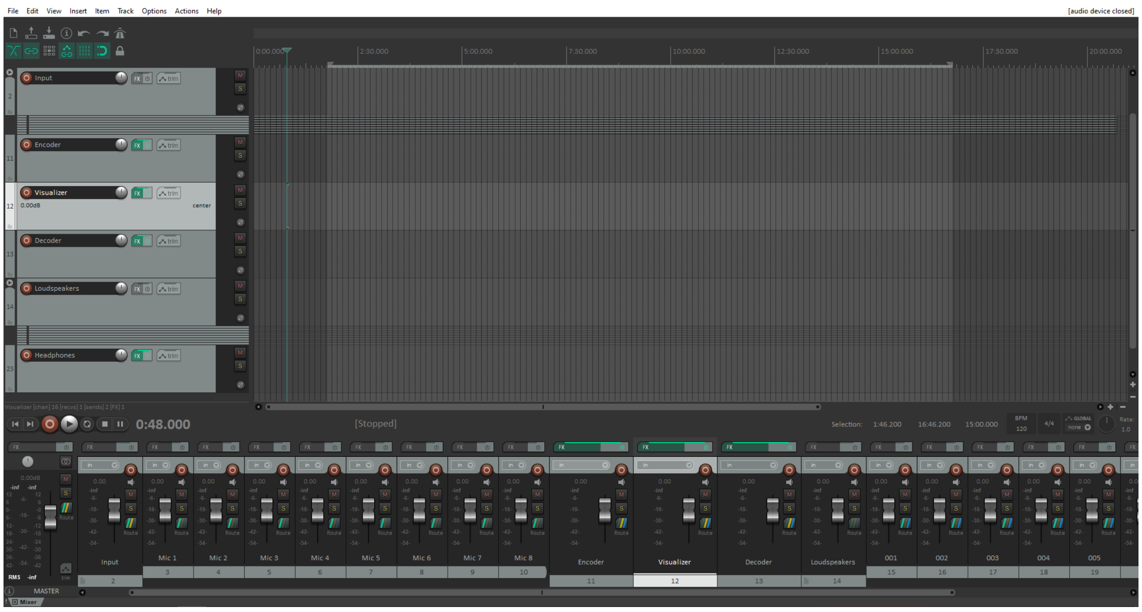
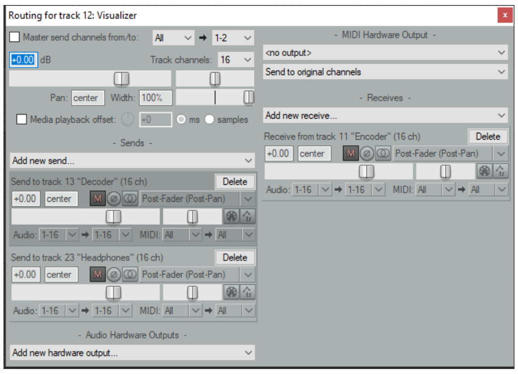
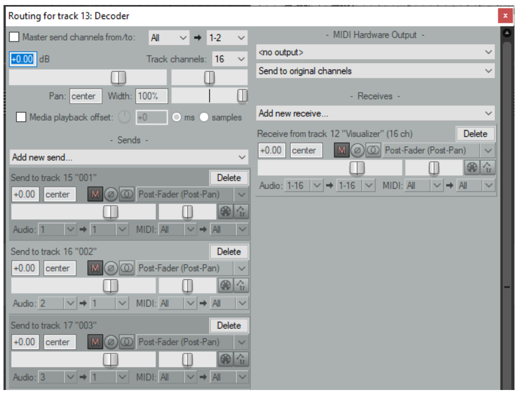


Figure 4.2: User Interface in REAPER.


Routing for track 12: Visualizer
Master send channels from/to: All → 1-2
+0.00 dB Track channels: 16
Pan: center Width: 100%
Media playback offset: +0 ms samples
- Sends -
Add new send...
Send to track 13 "Decoder" (16 ch) Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 1-16 → 1-16 MIDI: All → All
Send to track 23 "Headphones" (16 ch) Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 1-16 → 1-16 MIDI: All → All
- Audio Hardware Outputs -
Add new hardware output...
- MIDI Hardware Output -
<no output>
Send to original channels
- Receives -
Add new receive...
Receive from track 11 "Encoder" (16 ch) Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 1-16 → 1-16 MIDI: All → All

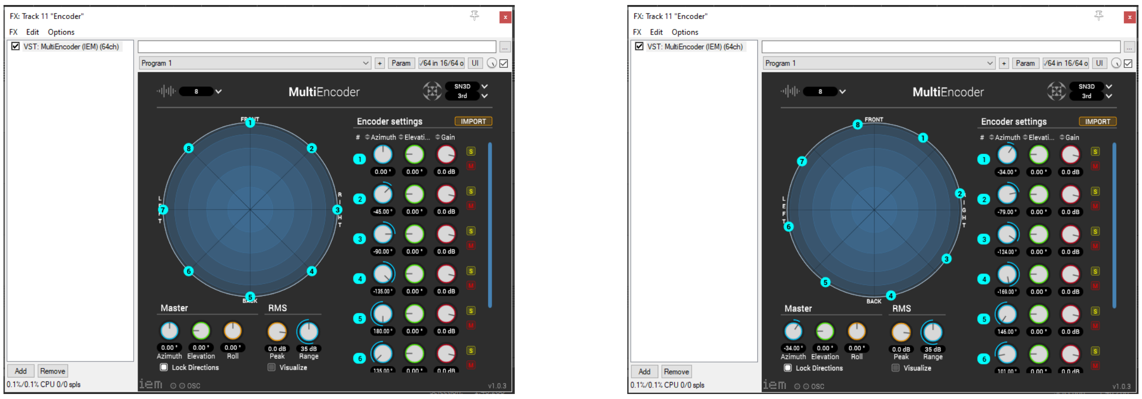
(a) Routing of the Visualizer track.


Routing for track 13: Decoder
Master send channels from/to: All → 1-2
+0.00 dB Track channels: 16
Pan: center Width: 100%
Media playback offset: +0 ms samples
- Sends -
Add new send...
Send to track 15 "001" Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 1 → 1 MIDI: All → All
Send to track 16 "002" Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 2 → 1 MIDI: All → All
Send to track 17 "003" Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 3 → 1 MIDI: All → All
- MIDI Hardware Output -
<no output>
Send to original channels
- Receives -
Add new receive...
Receive from track 12 "Visualizer" (16 ch) Delete
+0.00 center Post-Fader (Post-Pan)
Audio: 1-16 → 1-16 MIDI: All → All

(b) Part of the routing of the Decoder track.

Figure 4.3: Examples of routing in REAPER.

Now to the implementation of each step in detail. For the encoding in the Encoder track we use the *MultiEncoder* plug-in. This looks like shown in Figure 4.4a. On the top left the number of channels to be encoded is specified and on the top right the normalization and Ambisonics order can be chosen. Below, the plug-in allows us to specify the azimuth and elevation of each channel (each representing one microphone), as well as the gain. At the bottom all microphones can be rotated at once if needed. This can be quite useful, especially for testing purposes. Here it was used to rotate the microphones so that the front is exactly at the position of the synchronization clap (the signal in each recording that was used as a clapperboard for synchronization) as will be determined in Chapter 5. This will be useful for audio-video synchronization in Unity in Chapter 6. The rotated version is shown in Figure 4.4b.



(a) Encoding of the signals.

(b) Rotated encoding of the signals.

Figure 4.4: *MultiEncoder* plug-in.

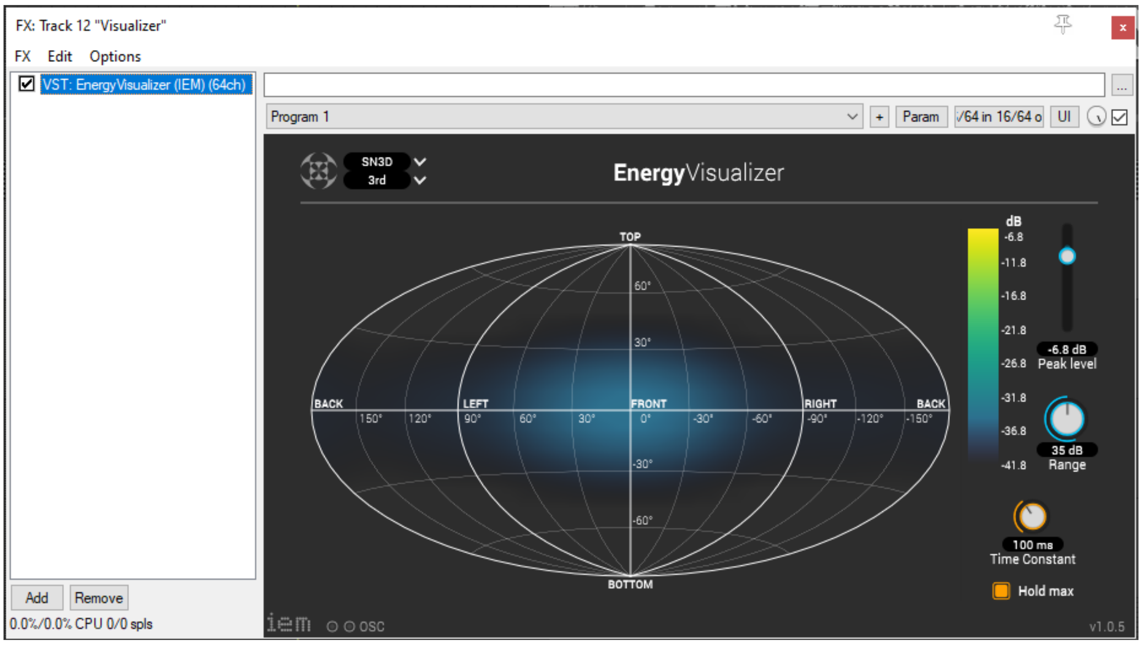
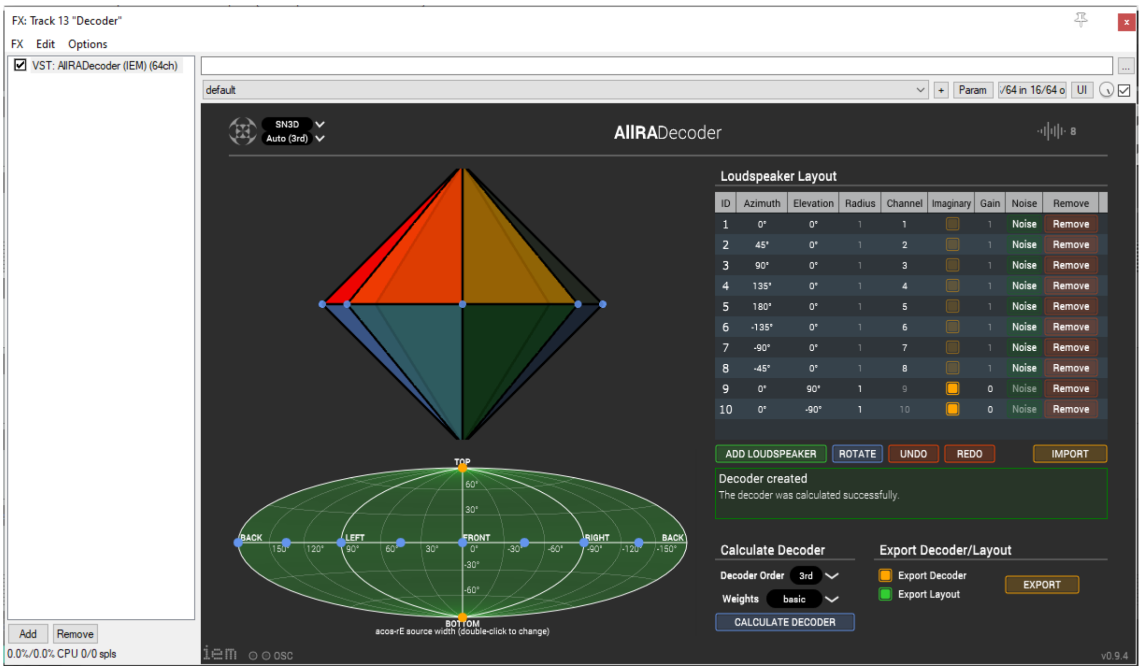


Figure 4.5: *EnergyVisualizer* plug-in visualizing the synchronization clap in the front.

In the next step the sound field is visualized with the *EnergyVisualizer* plug-in. The visualization of the energy for a synchronization clap is shown in Figure 4.5. We can see that the clap is already rotated to the front. In theory this could already be used to estimate find the correct angle by eye, but we will only use it to check the calculated angle.

For the decoding in the Decoder track we use the *AllRADecoder* plug-in. This plug-in allows us to specify any loudspeaker layout (where the point of origin is within the convex hull of the loudspeakers) and then decode to it using the AllRAD approach. As shown in Figure 4.6, every loudspeaker position can be specified. Here, the loudspeakers are placed at the points of the octagon in order to form a 7-design. As the point of origin needs to be inside the convex hull, two imaginary loudspeakers were added. Imaginary loudspakers are used to make the triangulation complete and unique but are not actually decoded to. The signal that is calculated for the imaginary loudspeakers



Loudspeaker Layout

ID	Azimuth	Elevation	Radius	Channel	Imaginary	Gain	Noise	Remove
1	0°	0°	1	1	☐	1	Noise	Remove
2	45°	0°	1	2	☐	1	Noise	Remove
3	90°	0°	1	3	☐	1	Noise	Remove
4	135°	0°	1	4	☐	1	Noise	Remove
5	180°	0°	1	5	☐	1	Noise	Remove
6	-135°	0°	1	6	☐	1	Noise	Remove
7	-90°	0°	1	7	☐	1	Noise	Remove
8	-45°	0°	1	8	☐	1	Noise	Remove
9	0°	90°	1	9	🟧	0	Noise	Remove
10	0°	-90°	1	10	🟧	0	Noise	Remove

ADD LOUDSPEAKER ROTATE UNDO REDO IMPORT

Decoder created
The decoder was calculated successfully.

Calculate Decoder
Decoder Order: 3rd
Weights: basic
CALCULATE DECODER

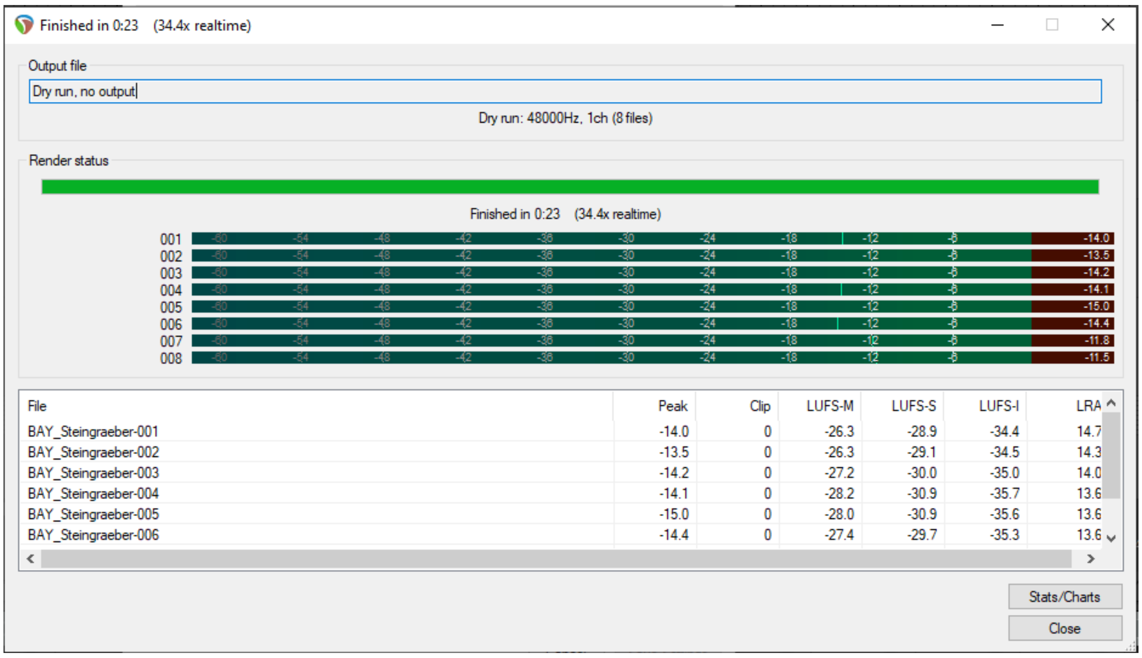
Export Decoder/Layout
🟧 Export Decoder
🟩 Export Layout
EXPORT

Figure 4.6: *AllRADecoder* plug-in decoding to the 7-design with additional imaginary loudspeakers above and below with gain 0.

is distributed to its neighbors. This allows for more flexible loudspeaker layouts [ZF12]. Here, the imaginary loudspeakers are placed directly above and below the listener, but both have the gain set to 0. Hence, their signal is 0, too, and they do not influence the signals of the non-imaginary loudspeakers. This leads to a pure decoding to the 7-design layout. It should be noted that this decoding is only an approximation of a direct decoding to the 7-design because of how AllRAD decoding works. But while AllRAD decoding does not guarantee our optimality, it is very close for sensible loudspeaker layouts [ZF12]. For the perceived width this can even be shown in the bottom left (Figure 4.6). Here we see that the perceived width is constant along the horizon. While it gets worse (brighter green larger perceived width) at the poles, however the perceived width here is already as bad as possible through the microphone layout, so this is not a problem. Sounds from directly above are distributed evenly to all microphones and are hence encoded as omnidirectional sounds.

The decoded signals can then be rendered to files for use in Unity. This can be done very fast, even for longer recordings. Figure 4.7 shows the rendering for a 13 minute 22 second long audio recording rendered in 23 seconds.

Finally, we have the other path that allows us to listen to the sound field in REAPER. In the Headphones track the *BinauralDecoder* plug-in transforms the Ambisonics signal directly to a binaural headphone signal using a HRTF short about binaural decoder. The signal is calculated as if the head would look at the front direction as seen in the other plug-ins. This allows us to test listen to the recording, but if we wish to turn the head, the signal needs to be rotated in the program, e.g. by rotating all microphones in the *MultiEncoder*. This can also be used to check the calculated synchronization clap position by ear.



Finished in 0:23 (34.4x realtime)

Output file

Dry run, no output

Dry run: 48000Hz, 1ch (8 files)

Render status

Finished in 0:23 (34.4x realtime)

001	-14.0
002	-13.5
003	-14.2
004	-14.1
005	-15.0
006	-14.4
007	-11.8
008	-11.5

File	Peak	Clip	LUFS-M	LUFS-S	LUFS-I	LRA
BAY_Steingraeber-001	-14.0	0	-26.3	-28.9	-34.4	14.7
BAY_Steingraeber-002	-13.5	0	-26.3	-29.1	-34.5	14.3
BAY_Steingraeber-003	-14.2	0	-27.2	-30.0	-35.0	14.0
BAY_Steingraeber-004	-14.1	0	-28.2	-30.9	-35.7	13.6
BAY_Steingraeber-005	-15.0	0	-28.0	-30.9	-35.6	13.6
BAY_Steingraeber-006	-14.4	0	-27.4	-29.7	-35.3	13.6

Figure 4.7: Rendering of the loudspeaker audio for a 13 minute and 22 second long recording.

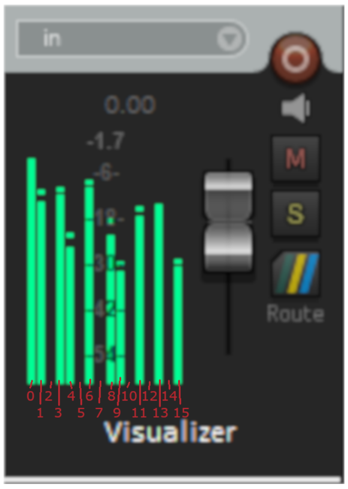


Figure 4.8: Ambisonics channels in REAPER with ACN numbering. The channels that carry only information about the z-axis are 0.

Lastly, we can also check the statement that no information is encoded into the channels that only carry information about the z-axis. In Figure 4.8, we can see that these channels are all constant 0. We can additionally see that through the SN3D normalization the W channel has the highest peak value.

5 Synchronization of Audio and Video

This chapter will be concerned with the second part of the implementation, namely the synchronizing of the audio and the video. As both audio and video were recorded separately, they still need to be synchronized so that the audio matches to what is happening in the video. We not only need to do this for time (i.e. that both audio and video start and end at the same moment), but for the angle as well. The sound of a sound source should come from the direction of the visual sound source and not an unrelated one. As we only work on the horizontal plane, there is only one angle to be matched. To help us with the synchronization for most recordings, a clapperboard was imitated by a simple hand clap, which we call the synchronization clap. This is a signal which is relatively easy to localize in both audio and video (which also has its own non-spatial audio track). In the following section we will discuss the time synchronization and in the section after the angular synchronization.

5.1 Time Synchronization

Synchronizing audio and video in time is the simpler task. In REAPER, we just need to align the audio tracks of the spatial audio and the video audio. The synchronization clap would make this task very easy as it gives a distinctive form as shown in Figure 5.1, but in the stitching process the clap was cut out of the video. Thus, while still found in the unstitched videos, it cannot easily be used to synchronize the stitched video. This complicates the time synchronization. Luckily, the time between the synchronization clap and the start of each stitched video was written down.

Using the recorded times, it is possible to get the synchronization down to less than a second. However, listening to both audio tracks at the same time, it becomes apparent that some echo remains due to them not being aligned perfectly. For most videos it is possible to improve this by looking for

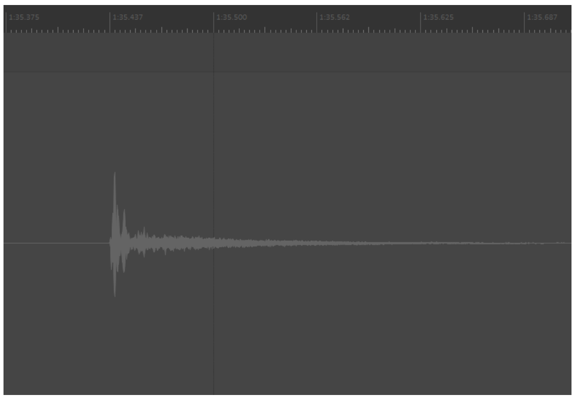
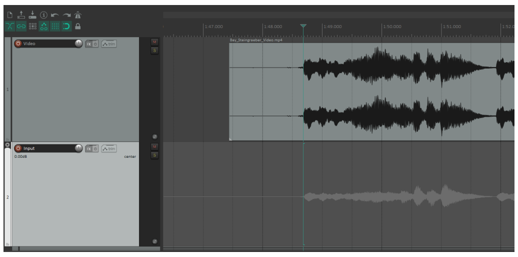


Figure 5.1: Waveform of the synchronization clap.


(a) Time synchronization with waveform possible.

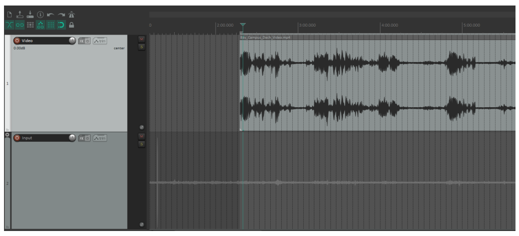

(b) Time synchronization with waveform not possible.

Figure 5.2: Time synchronization in REAPER. For both pictures the audio is already synchronized.

other distinctive sounds and match the waveforms by eye. This can be seen in Figure 5.2a. Listening to both audio tracks at the same time now confirms that it is correctly synchronized as each sound is only heard once.

However, another problem occured for some locations: The microphones of the GoPros are much worse than the microphones for the audio recording. Thus, in some locations the waveform is dominated by noise mainly from wind. This is shown in Figure 5.2b. Here, the tracks are already synchronized but the waveforms still do not match as the wind noise dominates the video waveform. While this makes the visual matching more difficult, it is sometimes still possible to find matches when the wind is silent. But even if not, loud sounds can still be heard on both tracks so matching by ear is possible as we already knew the time difference up to less than a second.

If we listen to the sound and watch the video we can see that the result is satisfactory.

5.2 Angular Synchronization

The angular synchronization is a bit more difficult to achieve with satisfactory results. In practice, this synchronization is often simply achieved by ear. One listens to the recording with a binaural decoder in REAPER and rotates the audio until it sounds like it comes from the right direction. As we would like to have a ground truth for the angular position, this approach is not good enough, since, as was already discussed in Section 2.2, the human ability to localize sounds is not always that good (even though it is quite good for clear signals like the synchronization clap). Thus, we can check our result by ear, but we will use a better sound source localization method to actually calculate the direction of the sound.

The sound source localization method used is a beamforming technique called Steered Response Power (SRP) map. In beamforming techniques a beam is directed in different directions to calculate the audio signal that would be perceived from that direction [VB88]. For many microphone arrays this can be quite complicated, but as we already encode into Ambisonics we can directly use beamforming on the Ambisonics signal. In Ambisonics beamforming can be achieved by just calculating the function $p_N(\varphi, \vartheta, t) = \sum_{n=0}^{N} \sum_{m=-n}^{m} \gamma_{nm}(t) Y_n^m(\varphi, \vartheta)$ for the desired direction (φ, ϑ) [JHN10].

To get the SRP we do not look at just one moment t but at a small window from t_0 to t_k. Additionally, as the amplitude can be negative we square each time step. This leads to [JHN10]

$$M(\varphi, \vartheta) = \sum_{t=t_0}^{t_k} p_N(\varphi, \vartheta, t)^2. \tag{5.1}$$

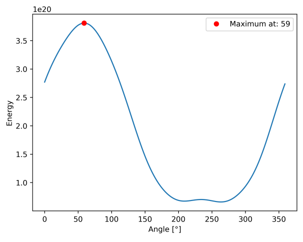
In order to get a SRP map, $M(\varphi, \vartheta)$ is calculated for different angles. The sound source should then be positioned in the direction of the highest response. This approach gives quite good results for clear signals [JHN10].

In our case, since we are only interested in the direction on the horizontal plane, we thus only have a 1D response map. The best audio source to localize should be the synchronization clap. It is clear, distinct and much louder than any other sound source. Hence, we calculate the SRP map with a resolution of $1°$ for a short time window around the clap for each location. $1°$ is chosen as it is difficult for humans to be much more precise. This can be done by rendering the Ambisonics signal directly in REAPER and writing a short program to calculate the response map from the file. The results can be seen in Figure 5.3. For most locations, the DOA of the clap is a clear maximum. Sometimes, the maximum is flatter, as in Figure 5.3g, but the general direction is still clear. For two locations there are quite large local maxima close to the global maximum (see Figure 5.3i and 5.3j). The problem was that for these two recordings there was no synchronization clap. Instead, other less distinct audio sources have been chosen as synchronization signal in the provided data. In Figure 5.3i, this was the word “zwei” spoken close to the microphone array and for Figure 5.3j loud step close to the microphone. While for the word the maximum is still rather distinct, for the step the maximum is not that clear. The listening experiments performed in Section 6.3 even suggest that the local maxima close to the global maxima is actually the synchronization step. This shows that a good synchronization signal is important to calculate a ground truth. All calculated maxima were still used as ground truths for the next part of the implementation.

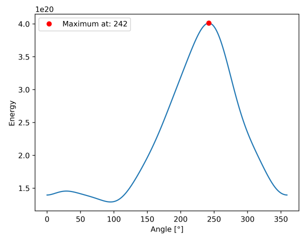
The main reason for this is that we have other ways to check the correctness of the calculations. On the one hand, the visualizer shows us the direction of highest energy (cf. Figure 4.5). On the other, we can still use the basic method of judging the direction by ear to check the calculated solution. For both methods the ground truth seemed to be correct.

This ground truth does not give us the complete angular synchronization, as only the DOA of the sound in the audio was calculated. The final part of matching the calculated direction of the clap to the position of the clap in the video was done in Unity by eye. To make this easier, as mentioned before, the direction of the clap was always encoded at the front direction in REAPER. This lead to the clap always being located at the same loudspeaker. Then this loudspeaker was matched up visually with the position where the clap took place. This will be explained in more detail in Section 6.2.

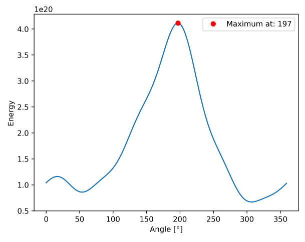
The SRP map method was chosen for its simplicity while still having good results. However, there are many different methods for sound source localization. Most standard methods can be used with Ambisonics but were not directly developed for it [LMR+21]. Thus, they do not take advantage of the properties of Ambisonics. However, there are some methods developed directly for Ambisonics. The pseudointensity vector method introduced by [JHN10] was developed for Ambisonics with similar quality but less computational complexity than the SRP map. Another method based on Ambisonics and CRNN was devolped by [GKGG21]. This approach even allows for multiple sound source localization. It should be noted that in both cases the methods described work only on FOA.



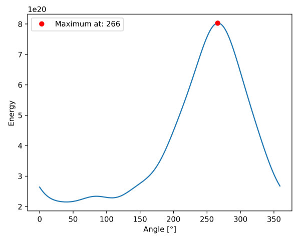
(a) SRP map for Bayreuth Campus Dach.



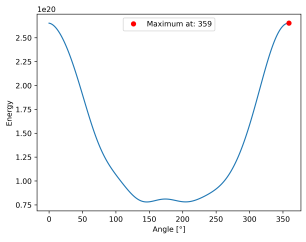
(b) SRP map for Bayreuth Campus See.



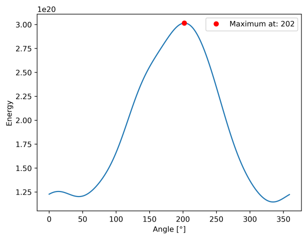
(c) SRP map for Bayreuth Eremitage.



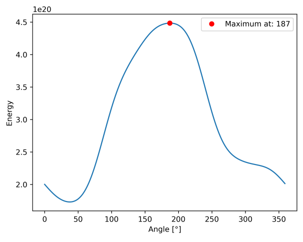
(d) SRP map for Bayreuth GSP Innenhof.



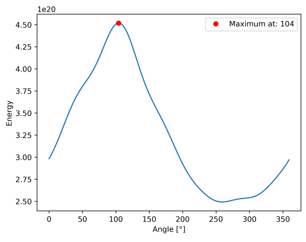
(e) SRP map for Bayreuth Steingraeber.



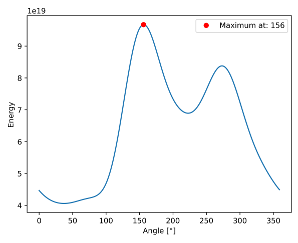
(f) SRP map for Berlin Dachterasse.



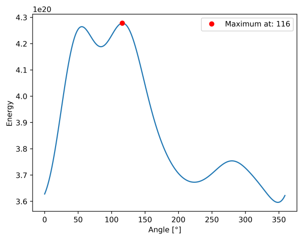
(g) SRP map for Berlin Experimentalzone.



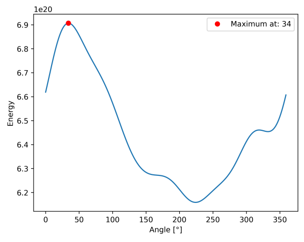
(h) SRP map for Berlin Hackescher Markt.



(i) SRP map for MIME Zentrum.



(j) SRP map for Berlin Rosengarten.



(k) SRP map for Berlin Unterfuehrung.

Figure 5.3: SRP maps for all different locations.

Lastly, there are approaches that might even be used to do the complete angular synchronization. The approach in [MBY+20] can match spatial sound sources to their visual sound source objects in 360° images using DNNs. This could make the manual visual alignment of loudspeaker and clap position unnecessary.

6 Audio-Visual 3D Experience in Unity

This chapter is concerned with the final part of the implementation, namely the experience that brings together audio, video and synchronization. This was done using the software Unity and the Microsoft Mixed Reality feature packages for Unity. The first section of this chapter will give a short overview on Unity and the Microsoft Mixed Reality feature packages. The next section recounts the implementation in Unity, its functionality and structure. In the last section, we will take a look at the results of a small scale listening experiment that was done with the experience.

6.1 Unity and Microsoft Mixed Reality

In this section, we take a short look at the real-time 3D development engine Unity. Unity is a cross-platform development engine that can be used to create immersive and interactive experiences for video games, films, architecture and many more industries [Unity05]. Originally developed as a desktop game engine for 2D and 3D games, it has extended to many different platforms like mobile and XR, and has found usage even outside video games. Unity is a proprietary software, but it is free for students and small businesses making it especially attractive for beginners and indie developers. Unity offers a user-friendly visual editor with drag and drop functionality that can be used without much programming knowledge (see Figure 6.1) and a scripting API in C# that allows for implementation of complex game logic and other interactions. The implementation can be tested directly in the editor, and to an extent it is even possible to edit the running game, making Unity a real-time editor. Additionally, Unity provides an extensive online marketplace called the Unity Asset Store where developers can buy and sell assets, scripts, plug-ins and more allowing for an improved game development process.

In general, Unity has many features for different use cases, most of which were not needed for the implementation of the audio-visual experience as the functionality of the experience is limited. The only non-standard part is the VR component. For this we used the Windows Mixed Reality platform [WMR15], which provides XR support for compatible XR-headsets such as the *HP Reverb*, which was used for development and testing. In order to use Windows Mixed Reality with Unity, Microsoft provides the Windows Mixed Reality feature packages, which uses on the OpenXR standard [Khr19]. In theory, these packages should make VR development in Unity quite simple, but in practice quite a lot of problems presented themselves. While they provide many useful assets and scripts for VR experiences, which for example allow the developer to simply make an object in Unity grabbable with the controller without needing to implement all the functionality behind it, the usage of the feature tools proved to be quite complicated, especially for a beginner in Unity and VR development. Even though the documentation claims differently, some versions of the feature packages do not work with the headset and some versions of Unity. This made trial and error testing of different versions for compatibility necessary. Furthermore, it is quite easy to make the whole Unity project practically unusable by changing a small setting, making it easier to go

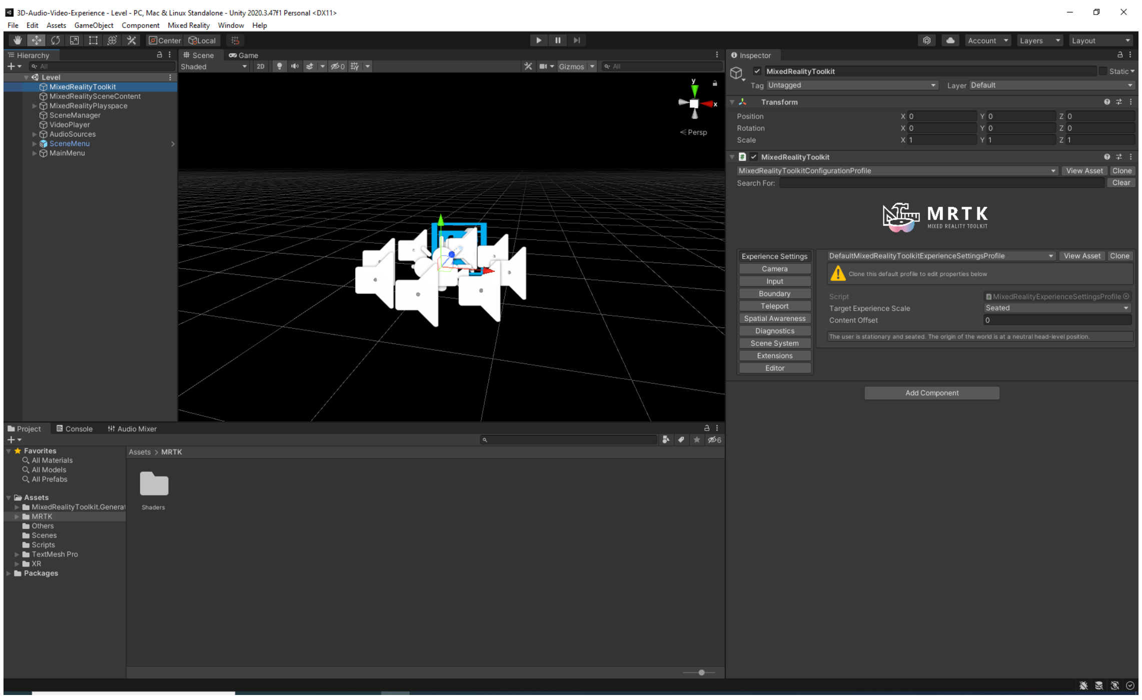


Figure 6.1: User Interface in Unity. On the top left the Hierarchy of the game objects is shown, in the middle the scene is shown visually, on the right the Inspector shows information about the selected object and on the bottom the project folder and the assets are displayed.

back to an earlier version than attempt to find out what went wrong and how to fix it. Additionally, some bugs were observed for which no previous reports were found and as such needed to be solved by workarounds or just not using the corresponding feature. In general, it feels like some features for older Windows Mixed Reality headsets that especially are not relevant to the Hololens 2 have been abandoned or at least are not a priority. Nonetheless, the utility that the feature packages provide is quite substantial and most of the time the developer can use many VR functionalities directly out-of-the-box. Two very important feature packages should be mentioned: the Mixed Reality Toolkit (MRTK) and the Microsoft Spatializer. The MRTK provides features which can manage, among other things, the input system. This allows us to use the controllers intuitively without thinking much about implementation. The Microsoft Spatializer allows us to apply an HRTF in Unity, which gives us spatial audio beyond the cone of confusion.

6.2 Implementation in Unity

In the following section we will take a more in-depth look at the implementation in Unity. The goal of the application is to replay spatial audio and 360° video. Since the goal is that the player can look around and experience the location in an immersive way, this will be done using VR. The spatial audio should obviously be replayed in a way that allows the player to actually hear the changes when turning their head which can easily be done in Unity 3D and improved by the use of an HRTF provided by the Microsoft Spatializer.

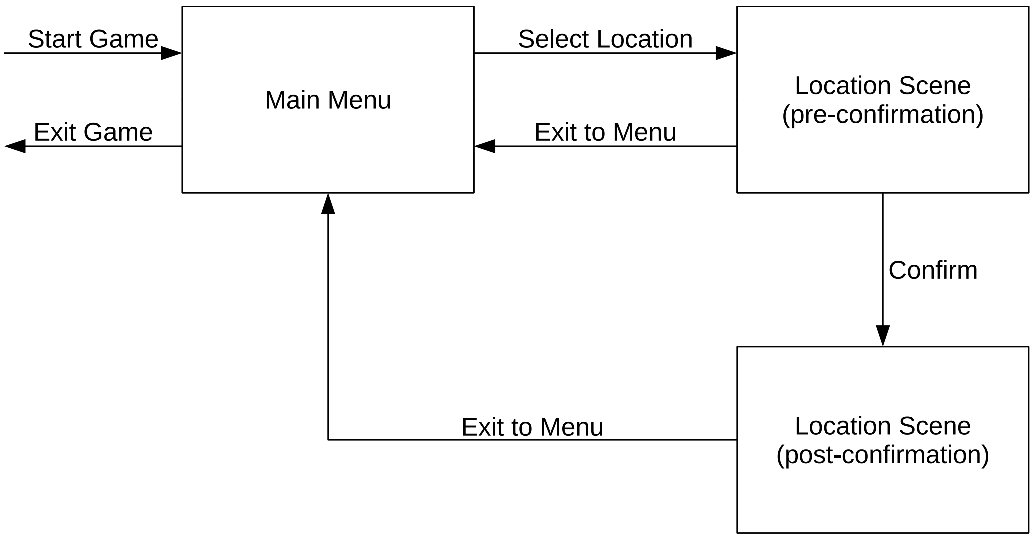


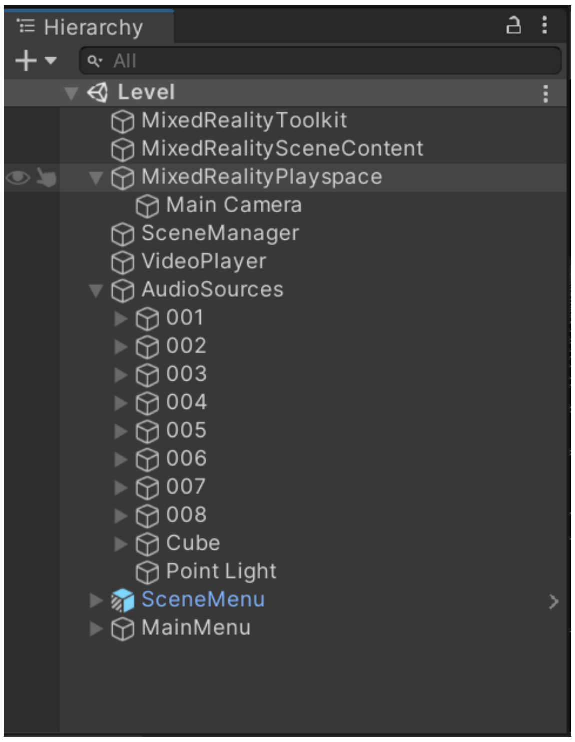
Figure 6.2: State diagram for the experience.

Additionally, to these minimum requirements the need for the ability to choose from the locations and change between arose. Since it would be especially annoying for the player to restart the game at each location, considering it is done in VR, this choice will be made available in-game.

Lastly, a small gamification was introduced to the experience. Upon choosing a location, the audio is rotated in the horizontal plane by a random angle. Thus, the audio and video are again unsynchronized in angle. The player then has to try to rotate the audio into the right position using auditive and visual cues provided by the recordings. When the player thinks the audio and video are matched up, they can confirm the matching. Then, the angle between the correct audio position and the player chosen audio position is calculated and shown to the player. This makes the experience more interesting and allows checking the quality of the angular synchronization and the human ability to match audio and video.

The experience was implemented in Unity Version 2020.3.47f1 using the Windows MRTK and the Microsoft Spatializer. It was developed and tested with the *HP Reverb Professional*.

The general functionality behind the experience can be explained using the state diagram shown in Figure 6.2. Upon starting the application, the player is confronted by the main menu and the paused video of a location is loaded as visual background without sound. From here, the player can select any location or exit the game. Selecting a location closes the main menu, shows the scene menu and in the background loads the video and audio. Then the audio is randomly rotated. The scene menu allows the player to start and pause the playback of the recordings, confirm the rotation of the audio and go back to the main menu. As long as the direction is not confirmed, the audio sources can be rotated around the player by grabbing and moving anywhere. Confirming the direction makes it impossible to rotate the audio further and shows the error that was calculated in comparison to the ground truth. From here it is only possible to play and pause the playback or go back to the main menu.



```
Hierarchy
All
Level
    MixedRealityToolkit
    MixedRealitySceneContent
    MixedRealityPlayspace
        Main Camera
    SceneManager
    VideoPlayer
    AudioSources
        001
        002
        003
        004
        005
        006
        007
        008
        Cube
        Point Light
    SceneMenu
    MainMenu
```

Figure 6.3: Scene hierarchy of the Unity application.

The whole experience in Unity was implemented in only one scene. A scene in Unity is an asset that contains part of the application. In terms of video games, it can be thought of as a level. Making the whole experience one scene makes it easier to work with VR. Managing the VR components between each scene is more difficult, as the MRTK needs to create and destroy some of the components each time a scene is changed. However, only using one scene does not make the implementation confusing as it is still a rather simple scene. The scene hierarchy (cf. Figure 6.3 and Figure 6.1 on the left) gives an overview of all game objects placed in the scene. A game object can be any part of the scene from physical objects over invisible light or audio sources to objects that only have attached game logic without direct physical representation. The complexity of an object can also vary significantly, as they can be nested into complex parent-child relationship trees and an object can have game logic applied to it by many different scripts (cf. Figure 6.4).

Now let us take a look at the implementation of the experience using the hierarchy. As all parts of the game are either game objects themselves or in some way attached to game objects, the hierarchy is a useful view on the important parts of the implementation. It should be noted that not everything which is implemented is found in the hierarchy. Elements like render textures and audio mixers are their own assets and therefore not in the hierarchy. However, they are attached to corresponding components like video players and audio players respectively, which again are game objects. The scene hierarchy of the experience can be seen in Figure 6.3.

The *MixedRealityToolkit* provides a significant amount of VR related functionality (cf Figure 6.1 on the right) from the MRTK. Of these, only the basic input part, which allows easy usage of the controllers of the *HP Reverb*, was utilized for this project. The *MixedRealitySceneContent* and the *MixedRealityPlayspace* objects are also components related to the MRTK. They both manage objects that are created in the runtime by the MRTK - like for example the visual representation of the controllers. It should be noted that the *MainCamera* is a child of the *MixedRealityPlayspace* object. The *MainCamera* is the representation of the player camera and takes care of the head tracking in the game, which allows the player to look around. It is also the position of the audio listener, which means that any audio we hear through the headphones is heard from the position of the main camera. As a result, the player sees and hears as if they were at the position of the *MainCamera*.

The *SceneManager* is mainly responsible for custom game logic relating to managing the scene. The logic is implemented in an attached script and is responsible for the change between the “main menu” and the “location scene” parts of the experience.

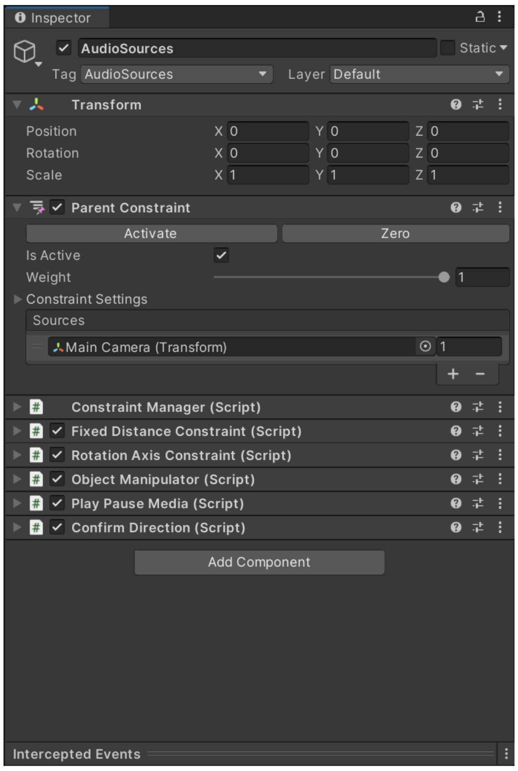
The *VideoPlayer* is the object that manages the replay of the video. The video itself is played on the skybox, making it unnecessary to play it on a game object. Through the *VideoPlayer*, scripts can play, pause or change the video shown on the skybox.

The *AudioSources* object is the parent object of the virtual representation of the loudspeakers that replay the audio tracks calculated in Chapter 4. It additionally has a lot of game logic attached to it, which takes care of the following problems. The loudspeakers need to be kept around the head so that the relative position of *MainCamera* and *AudioSources* is always the same, otherwise the player could move out of the center of the loudspeakers. However, while the loudspeakers need to move with the player, they should not rotate and tilt with head movements, so scripts were employed to create this functionality. The audio needs to be grabbable with the controller to rotate it around the player. As such, there is an invisible box part of *AudioSources* which makes sure that the player can simply grab anywhere to rotate the audio. Another script takes care of the grabbing, allowing the player to actually rotate the audio. But this rotation needs to only be allowed around the y-axis (up-down axis in Unity) and it should again be impossible to move the audio in relation to the player. This was made possible using logic and scripts provided by Unity and the MRTK. Still, there was a need for additional custom logic. The audio and video for different locations needs to be managed and loaded in. Additionally, the player should be able to play and pause audio and video together. Lastly, there was a need for logic that calculates the error upon confirmation and outputs it.

The audio sources that are children of *AudioSources* are positioned in an octagon around the *MainCamera*. While they are invisible in the position of each there is a small visible ball (see Figure 6.1 in the central scene view). This is quite useful for the player as it gives them a sense of how much they have rotated the surrounding audio. Each audio source sends its audio to an audio mixer which applies the HRTF using the Microsift Spatializer. With this, the player always hears the audio as if they were directly in the middle of an octagon of loudspeakers playing the audio.

Lastly, *SceneMenu* and *MainMenu* are the parent objects of the menus. *MainMenu* is the menu from where the different locations can be selected and the game can be quit. The *SceneMenu* gives functionality after a location is selected. It allows for play/pause, confirmation of the audio rotation and a return to the main menu. The game logic behind these actions is taken care of by the *SceneManager* and *AudioSources*.

The final step of the angular synchronization is, as mentioned before, also done in Unity. While we know from Section 5.2 that the synchronization clap is always at the position of loudspeaker 1 (or game object *001* in Unity) this is not necessarily the position in the video in Unity where the clap actually took place. To fix this we need to know the angle by which we rotate the audio in Unity to get loudspeaker 1 to the right position. This was done by eye measurement. It was additionally


Inspector

✓ AudioSources Static

Tag AudioSources Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

✓ Parent Constraint

Activate Zero

Is Active ✓

Weight 1

Constraint Settings

Sources

Main Camera (Transform) 1

Constraint Manager (Script)

✓ Fixed Distance Constraint (Script)

✓ Rotation Axis Constraint (Script)

✓ Object Manipulator (Script)

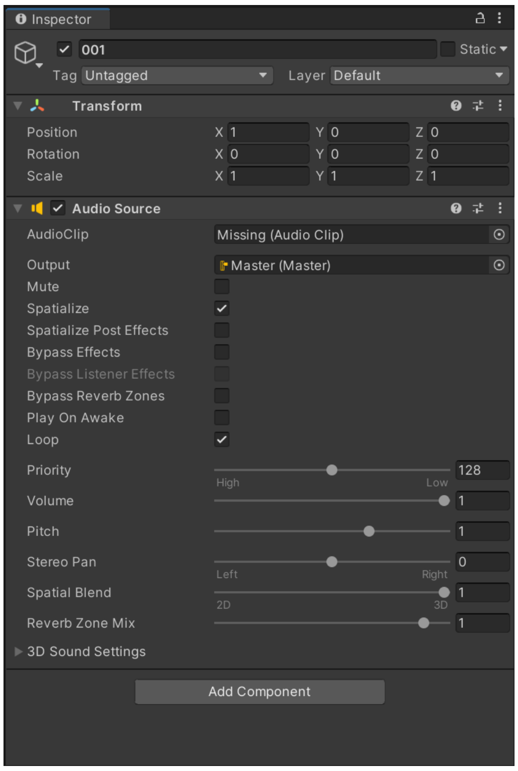
✓ Play Pause Media (Script)

✓ Confirm Direction (Script)

Add Component

Intercepted Events

(a) Inspector of the *AudioSources* game object.


Inspector

✓ 001 Static

Tag Untagged Layer Default

Transform

Position	X 1	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

✓ Audio Source

AudioClip	Missing (Audio Clip)
Output	Master (Master)
Mute	
Spatialize	✓
Spatialize Post Effects	
Bypass Effects	
Bypass Listener Effects	
Bypass Reverb Zones	
Play On Awake	
Loop	✓
Priority (High – Low)	128
Volume	1
Pitch	1
Stereo Pan (Left – Right)	0
Spatial Blend (2D – 3D)	1
Reverb Zone Mix	1

3D Sound Settings

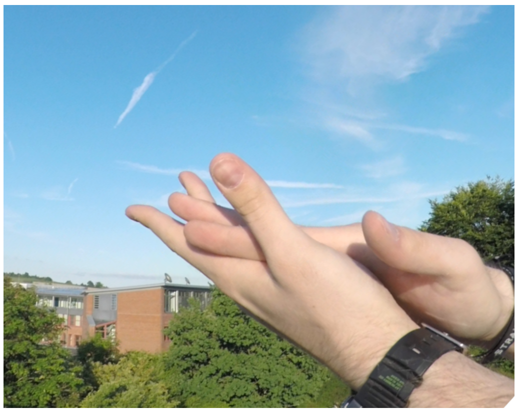
Add Component

(b) Inspector of the *001* game object which represents the loudspeaker 1.

Figure 6.4: Inspector views on audio related game objects.

made more difficult by the fact that the clap is already cut from the video that is replayed in Unity. It can be still found in unstitched and uncut versions from where we can get a screenshot of the clap (see Figure 6.5a). We can use this to find the corresponding location in Unity and match up loudspeaker 1 as shown in Figure 6.5b. Through the use of the inspector on the running game, we can then get the rotation of the audio sources that lead to this position. This rotation is the second part of the angular synchronization which can now be saved to be used to calculate the angular error in Unity.

Finally, there was one problem in Unity that could not be explained and should be noted. While everything works when running the game in editor, when running a build the Microsoft Spatializer wasn't working at all, i.e. there was spatial audio but without an HRTF leading to a cone of cofusion. For unknown reasons when creating a build a necessary dll for the Microsoft Spatializer was always missing from the build. The only workaround to this problem seems to be to find the dll in the Unity project files and copy it into the right folder of the build by hand.


(a) Screenshot of the Synchronization clap.

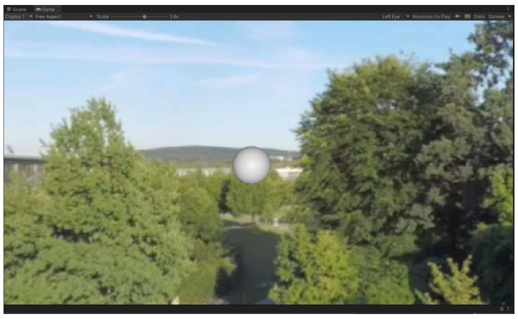

(b) Loudspeaker 1 rotated to the same position in Unity.

Figure 6.5: Finding the position of the synchronization clap in Unity.

6.3 Listening Experiment

In this section we will discuss the small scale listening experiment that was performed using the experience. This experiment was conducted to check how well it was possible for humans to match up the audio and the video in the experience. If the ground truth is correct and it is possible to localize sounds correctly in the recording, we should see matchings that are at least close to the ground truth.

The experiment had 14 participants and was done using 5 locations. Hence, the experiment is not expected to be representative but might still give some clues on the quality. The chosen locations were *Bayreuth Campus Dach*, *Bayreuth Steingraeber*, *Berlin Hackescher Markt*, *Berlin MIME-Zentrum* and *Berlin Rosengarten*. *Bayreuth Campus Dach* was chosen as it has sound sources only in one general direction which should make matching easy. Similarly, *Bayreuth Steingraeber* only has one sound source but is inside, which should influence localization. *Berlin Hackescher Markt* is a recording of a diffuse sound field outside, which again influences localization. Lastly, *Berlin MIME-Zentrum* and *Berlin Rosengarten* were chosen due to the fact that for both, the ground truth quality wasn't clear as discussed in Section 5.2. Thus, the experiments should tell us something about the quality of the ground truth for less than ideal situations. *Berlin MIME-Zentrum* is an echoic inside location with two moving sound sources, while *Berlin Rosengarten* was an outside location with only few sound sources. For each location the participants had as much time as they wanted, but were not allowed to restart the location. The locations were always played in the order in which they were listed here.

The result for each test was the calculated signed angle between the ground truth and the player determined position, which is saved by the experience. The results for all locations can be seen in Figure 6.6. From this we can see that for the first three locations the results are very similar. Most players matched the audio and video up quite well. Additionally, only few participants were wrong by more than 50° and if they were, it was only on the first two locations. For *Berlin MIME-Zentrum* the median is again close to 0°, but the variance between the participants was much larger than for the first three locations. *Berlin Rosengarten* was the only location for which the median was not close to 0°, but instead was at -50°. The variance in the guesses was worse than for the first three locations, but much better than for *Berlin MIME-Zentrum* and there were no outliers.

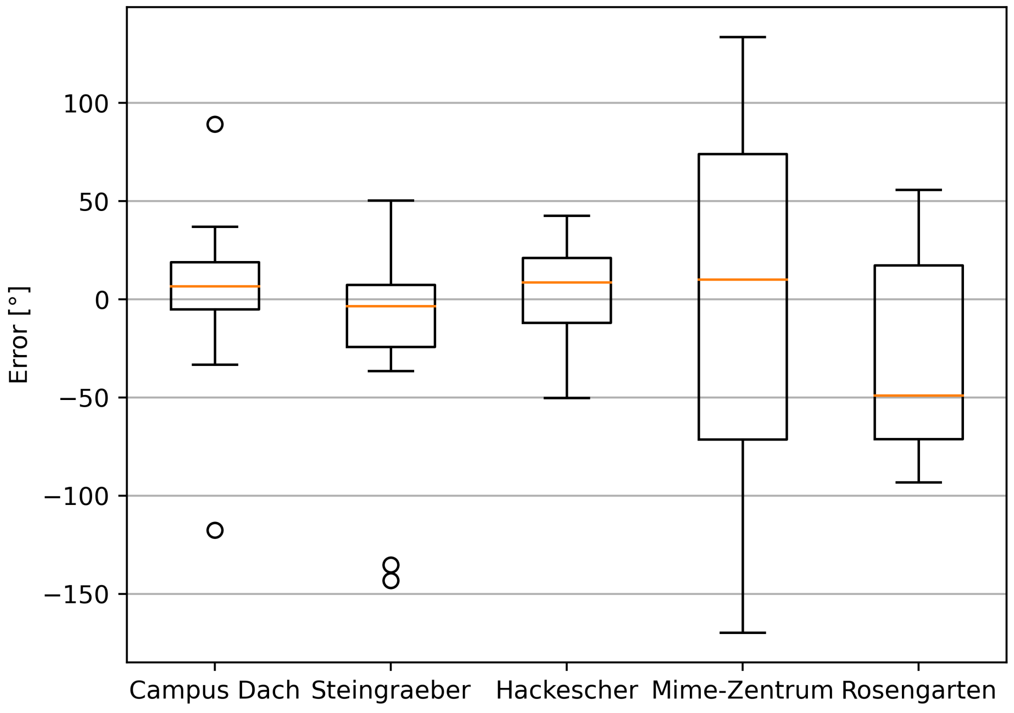


Figure 6.6: Boxplots of the results of the listening experiment.

The results confirm that it is possible to match up audio and video at least to the general direction and that the ground truth is correct for at least the first three locations. In case of *Berlin MIME-Zentrum*, it seemed like most participants were more or less guessing the location as there is no concentration of guesses around a certain angle. This would also explain the expected median close to 0°. The reason for this seems to be the echoic nature of the room and the two moving sound sources, both of which are difficult to localize. Additionally, this makes it impossible to tell if the ground truth is correct. For a different ground truth the boxplot would look practically the same with the same guesses. Lastly, the *Berlin Rosengarten* results indicate that the location is more difficult to match than the first three and that the ground truth might be wrong, as the median is at -50°. This is similar to the position of the local maximum in the SRP, visible in Figure 5.3j, which is about -60° different from the global maximum, meaning that this could, in fact, be the synchronization clap (which here was a step) and the global maximum is some other sound.

This shows the importance of a good synchronization signal. Without it, the calculated ground truth is much less reliable and we are back to matching by ear until it sounds right. Additionally, the results show that not all the locations allow for good matching as sound sources cannot be localized good enough. However, the ground truth seems generally correct and it is, in fact, possible to match audio and video quite well if some sound can be localized in both audio and video. It should be noted that, when it comes to quality, the use of a generic HRTF (as provided by the Microsoft Spatializer) could have lead to bad results for some participants. If the generic HRTF is similar to one's own HRTF, the matching is much easier than when it is very different. This could explain some of the outliers, especially on the first few locations, as it is possible for humans to get used to an HRTF besides ones own [Iid19]. Finally, while working on the thesis I myself have done the matching quite often, and I noticed that I got much better at matching the locations, which could again indicate that there is a need to get used to maybe not only the HRTF but other aspects of the experience before it is possible to match up audio and video well.

7 Conclusion and Outlook

This thesis presents an approach to creating an immersive audio-visual experience in Unity. The basis of this experience was pre-recorded high quality spatial audio and 360° video data that was provided for use in this project. As the video was already stitched, there was no need for video processing. However, the audio still needed to be extensively processed before it could be used. This audio processing utilized HOA with the mathematical background previously presented The actual processing was done in REAPER using the IEM Plug-in Suite. As the audio and video was provided unsynchronized, it was additionally synchronized in both time and angle. This angular synchronization became necessary as the audio was spatial and it was done using beamforming techniques. Finally, a Unity application was created that implements the experience with gamification elements. The application randomizes the rotation of the audio in relation to the video and the player needs to resynchronize the audio to the video. The quality of this matching is measured by the angle by which the player synchronization differs from the calculated synchronization. The experience with this gamification element was employed in a small scale listening experiment to estimate the quality of both the calculated synchronization and the player's ability to match audio and video in the experience.

This experiment allowed us to make three observations: Firstly, if the player can identify an audio source both in the video and the audio, they can often match both up to a ±30° error.
Secondly, if sound sources cannot be properly located, it seems that the player can only guess the direction the sound is coming from.
Lastly, for at least one recording location the angular synchronization calculated seems to be false. Unlike as for the other locations, the median of the error is not close to 0° but at -50°.

The first observation shows that the experience works as intended for many locations. The player can match up audio and video and really experience the location in an immersive way. This is very important for experiences like this since the actual sound reproduction is not as important as a reproduction that feels correct to the listener [Bla97].
From the second observation we can speculate that it is very hard to match up audio and video if the location is unsuitable for such a task. In an echoic location with continuous or randomly starting sounds it is very hard to tell where a sound is coming from. Thus, the choice of location and sounds is important to create good recordings for the experience.
The third observation points to the relevance of a good synchronization clap. Since the synchronization clap used to match the location with a median error of -50° was not a proper synchronization clap but only a stand-in and the SRP map had a second local maxima at -60°, it is probable that this was, in fact, the synchronization clap stand-in and the bad quality of this stand-in lead to the false result. With a proper clap this location would most likely have performed similarly to the other locations, where an audio source could be located well.

Additionally, it became clear that the choice of audio and video recording setup and preprocessing steps can significantly influence the difficulty when it comes to further processing. Since the synchronization claps were cut from the videos the synchronization was more difficult. A setup which always ensures the same angle between audio and video would also make the synchronization easier as then, matching would only need to be done once, instead of for every location. Furthermore, the audio was recorded with 8 microphones, which allows for capturing 2D 3rd Order Ambisonics. However, 7 microphones are enough to capture 2D 3rd Order and 9 would be enough to capture 4th. Thus, the eighth microphone does not improve the experience as much as a ninth would when it comes to audio quality. These are all things that should be kept in mind if more locations were ever to be recorded.

In addition, the three main components were implemented pretty much independently. This should allow for easier future adaptation or improvement of components. For example, if audio and video were processed differently for the same loudspeakers, there would be no need to change the Unity experience, as it only needs the audio, video and the synchronization angle between them. Another possibility would be calculating the ground truth for the angle with a new method, which would not require much change in the other two components.

There are many ideas for possible improvements and adaptations: The Microsoft Spatializer uses a generic HRTF but a personalized one would lead to better results. As such the experience could be extended to allow for use of a personalized HRTF if provided and defaulting to a generic one only if not. Additionally, it is possible to directly calculate the binaural signals with HRTF from Ambisonics [ZF19]. However, this would need to be implemented in Unity as the calculation is obviously dependent on the head movements. This was not done in this thesis, as Unity supports only up to 8 channels, which is only enough for FOA (2nd already needs 9 channels) and using FOA as an intermediate format would lead to a large loss of spatial quality. However, it would be interesting to see if this is possible in some other way and would lead to an improvement in matching quality, as it would remove some intermediate steps.

Another possible adaptation is to use different methods for calculating the ground truth. Especially methods that directly synchronize audio and video using NNs, like the one presented in [MBY+20], could be promising improvements, because the synchronization claps could become superfluous.

Finally, the whole experience could be implemented using a sound dome or similar technologies instead of headphones. This would remove the need for HRTFs. Additionally, AllRAD allows to simply adapt to any loudspeaker layout making it unnecessary to develop new decoding strategies. It should be noted that using a VR display influences the human ability to localize sound negatively in such a setting [HAM21]. Thus, it is unclear if this would improve the localization quality or would just be another way to experience the location.

Bibliography

[Art15] D. Arteaga. "Introduction to ambisonics". In: *Escola Superior Politècnica Universitat Pompeu Fabra, Barcelona, Spain* (2015) (cit. on pp. 17, 18, 32).

[Bla97] J. Blauert. *Spatial hearing: the psychophysics of human sound localization*. MIT press, 1997. ISBN: 9780262268684. DOI: 10.7551/mitpress/6391.001.0001 (cit. on pp. 18, 20, 59).

[Dan03] J. Daniel. "Spatial sound encoding including near field effect: Introducing distance coding filters and a viable, new ambisonic format". In: *Audio Engineering Society Conference: 23rd International Conference: Signal Processing in Audio Recording and Reproduction*. Audio Engineering Society. 2003 (cit. on p. 26).

[DB08] G. Dahlquist, Å. Björck. *Numerical methods in scientific computing, volume I*. SIAM, 2008. ISBN: 978-0-89871-644-3. DOI: 10.1137/1.9780898717785 (cit. on p. 25).

[Dic+03] G. Dickins et al. "Soundfield representation, reconstruction and perception". In: (2003) (cit. on p. 26).

[DRP99] J. Daniel, J.-B. Rault, J.-D. Polack. "Acoustic properties and perceptive implications of stereophonic phenomena". In: *Audio Engineering Society Conference: 16th International Conference: Spatial Sound Reproduction*. Audio Engineering Society. 1999 (cit. on p. 29).

[Ger75] M. A. Gerzon. "The Design of Precisely Coincident Microphone Arrays for Stereo and Surround Sound". In: *Audio Engineering Society Convention 50*. 1975. URL: http://www.aes.org/e-lib/browse.cfm?elib=2466 (cit. on p. 17).

[Ger92] M. A. Gerzon. "General Metatheory of Auditory Localisation". In: *Audio Engineering Society Convention 92*. 1992. URL: http://www.aes.org/e-lib/browse.cfm?elib=6827 (cit. on p. 20).

[GKGG21] P.-A. Grumiaux, S. Kitić, L. Girin, A. Guérin. "Improved feature extraction for CRNN-based multiple sound source localization". In: *2021 29th European Signal Processing Conference (EUSIPCO)*. 2021, pp. 231–235. DOI: 10.23919/EUSIPCO54536.2021.9616124 (cit. on p. 45).

[GP11] M. Gräf, D. Potts. "On the computation of spherical designs by a new optimization approach based on fast spherical Fourier transforms". In: *Numerische Mathematik* 119.4 (2011), pp. 699–724. DOI: 10.1007/s00211-011-0399-7 (cit. on p. 34).

[HAM21] T. Huisman, A. Ahrens, E. MacDonald. "Ambisonics Sound Source Localization With Varying Amount of Visual Information in Virtual Reality". In: *Frontiers in Virtual Reality* 2 (2021). ISSN: 2673-4192. DOI: 10.3389/frvir.2021.722321. URL: https://www.frontiersin.org/articles/10.3389/frvir.2021.722321 (cit. on p. 60).

[Han02] D. C. Handscomb. *Chebyshev polynomials*. CRC press, 2002. ISBN: 0-8493-0355-9 (cit. on pp. 30, 33).

[HHKP15] J. Herre, J. Hilpert, A. Kuntz, J. Plogsties. “MPEG-H 3D Audio—The New Standard for Coding of Immersive Spatial Audio”. In: *IEEE Journal of Selected Topics in Signal Processing* 9.5 (2015), pp. 770–779. DOI: 10.1109/JSTSP.2015.2411578 (cit. on p. 28).

[HS02] R. H. Hardin, N. J. A. Sloane. *McLaren’s Improved Snub Cube and Other New Spherical Designs in Three Dimensions*. 2002. arXiv: math/0207211 [math.CO] (cit. on p. 34).

[IEM17] IEM. *IEM Plug-in Suite*. 2017. URL: http://plugins.iem.at (cit. on p. 35).

[Iid19] K. Iida. *Head-related transfer function and acoustic virtual reality*. Springer, 2019. ISBN: 978-981-13-9745-5. DOI: 10.1007/978-981-13-9745-5 (cit. on pp. 18, 19, 57).

[JHN10] D. P. Jarrett, E. A. P. Habets, P. A. Naylor. “3D source localization in the spherical harmonic domain using a pseudointensity vector”. In: *2010 18th European Signal Processing Conference*. 2010, pp. 442–446 (cit. on pp. 44, 45).

[Khr19] KhronosGroup. *OpenXR*. 2019. URL: https://www.khronos.org/openxr/ (cit. on p. 49).

[LMR+21] M. U. Liaquat, H. S. Munawar, A. Rahman, Z. Qadir, A. Z. Kouzani, M. A. P. Mahmud. “Localization of Sound Sources: A Systematic Review”. In: *Energies* 14.13 (2021). ISSN: 1996-1073. DOI: 10.3390/en14133910. URL: https://www.mdpi.com/1996-1073/14/13/3910 (cit. on p. 45).

[MBY+20] Y. Masuyama, Y. Bando, K. Yatabe, Y. Sasaki, M. Onishi, Y. Oikawa. “Self-supervised Neural Audio-Visual Sound Source Localization via Probabilistic Spatial Modeling”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 4848–4854. DOI: 10.1109/IROS45743.2020.9340938 (cit. on pp. 47, 60).

[NZDS11] C. Nachbar, F. Zotter, E. Deleflie, A. Sontacchi. “Ambix-a suggested ambisonics format”. In: *Ambisonics Symposium*. Vol. 2011. 2011 (cit. on p. 28).

[Pul97] V. Pulkki. “Virtual Sound Source Positioning Using Vector Base Amplitude Panning”. In: *J. Audio Eng. Soc* 45.6 (1997), pp. 456–466. URL: http://www.aes.org/e-lib/browse.cfm?elib=7853 (cit. on p. 34).

[Rad15] R. Rader. *REAPER: An Exhaustive Review*. 2015. URL: https://www.extremraym.com/en/reaper-5-review (cit. on p. 35).

[Raf15] B. Rafaely. *Fundamentals of spherical array processing*. Vol. 8. Springer, 2015. ISBN: 978-3-662-45664-4. DOI: 10.1007/978-3-662-45664-4 (cit. on pp. 23, 24, 26).

[REAPER06] Cockos. *REAPER*. 2006. URL: http://www.reaper.fm (cit. on p. 35).

[Unity05] UnityTechnologies. *Unity*. 2005. URL: https://unity.com/ (cit. on p. 49).

[VB88] B. Van Veen, K. Buckley. “Beamforming: a versatile approach to spatial filtering”. In: *IEEE ASSP Magazine* 5.2 (1988), pp. 4–24. DOI: 10.1109/53.665 (cit. on p. 44).

[WFB11] T. Weller, S. Favrot, J. M. Buchholz. “Application of a circular 2D hard-sphere microphone array for higher-order Ambisonics auralization”. In: *Proceedings of Forum Acusticum*. 2011, pp. 2269–2274 (cit. on p. 36).

[WMR15] Microsoft. *Windows Mixed Reality*. 2015. URL: https://www.microsoft.com/en-us/windows/windows-mixed-reality (cit. on p. 49).

[Xie20] B. Xie. “Spatial Sound-History, Principle, Progress and Challenge”. In: *Chinese Journal of Electronics* 29.3 (2020), pp. 397–416. DOI: 10.1049/cje.2020.02.016 (cit. on p. 17).

[ZF12] F. Zotter, M. Frank. “All-round ambisonic panning and decoding”. In: *Journal of the audio engineering society* 60.10 (2012), pp. 807–820 (cit. on pp. 34, 41).

[ZF19] F. Zotter, M. Frank. *Ambisonics: A practical 3D audio theory for recording, studio production, sound reinforcement, and virtual reality*. Springer Nature, 2019. ISBN: 978-3-030-17207-7. DOI: 10.1007/978-3-030-17207-7 (cit. on pp. 17, 20, 21, 23–34, 36, 60).

All links were last followed on August 01, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

---

place, date, signature