*Proceedings of the 17th Linux Audio Conference (LAC-19), CCRMA, Stanford University, USA, March 16–19, 2019*

# WORKSHOP PROPOSAL: AMBISONIC DEVELOPMENT WITH JUCE

*Gabriel Zalles* *

UCSD Computer Music Dept.
UC San Diego, USA
gzalles@ucsd.edu

## ABSTRACT

This proposal details the nature of a workshop this author intends to conduct during the Linux Audio Conference (LAC) at Stanford in 2019. The workshop consists of teaching any parties interested the process required to develop some simple ambisonic plug-ins using the JUCE framework. JUCE is a free C++ library used extensively in the industry to create music application and Virtual Studio Technologies (VSTs). The intention of the workshop is to spark the students' curiosity, hopefully encouraging them to develop experiments in the future to help us resolve some of the underlying questions behind the psycho-acoustic principles used in ambisonics.

## 1. INTRODUCTION

Ambisonic has been a subject of great interest to audio engineers for over four decades since its inception in the 70's. Ambisonics is a full-sphere capture and reproduction method developed by Michael Gerzon. Born on December 4th 1945, Michael Gerzon, from the University of Oxford, is widely credited, along with Peter Craven, with popularizing ambisonics technology ([1]).

By expanding Blumlein's technique, ambisonics can, not only record sounds arriving from all directions, as an omnidirectional microphone would, but encode directional information of sound sources as well. Decoding of these formats is designed to overcome limitations of stereo which provide listeners with sounds on strictly the horizontal plane.

Part of the motivation behind Gerzon's work was figuring out the optimal way to successfully commercialize and improve upon quadraphonic sound, which, during the seventies, was predicted to become the new standard for audio reproduction. In a sense, the development of the tetrahedral array can be seen as a direct response to the struggles of quadraphonic sound, which never gained enough favour from the public to be considered a success.

During this workshop we will attempt to develop three VSTs for simple FOA encoding and decoding. Namely, we will implement in JUCE a FOA microphone array encoder (*FOAmicEnc*), a FOA panner (*FOApanner*) and a FOA quadraphonic decoder (*FOAquadDec*) which should work with the playback system provided for the workshop. While it might also be beneficial to show attendees how to develop an ambisonic rotator, these are most useful for binaural reproduction. Since convolution is complex topic to explain, and even harder to code, we will focus on loudspeaker reproduction at this moment, since it does not require convolution.

## 2. FIRST ORDER AMBISONICS

Generating an ambisonic *soundfield* can be accomplished either by encoding the output of a tetrahedral microphone, such the one shown in Figure 1, or by encoding a monophonic signal using Equation 1.

In the first case the raw A-format signals captured from our tetrahedral microphone are said to be encoded into B-format by using Equation 1. In order for the equation to work properly the center of our ambisonic microphone has to be labeled and the four channels must be recorded in the correct order.

In the latter case a monophonic signal is encoded by taking the horizontal (azimuth) and vertical (elevation) angle relative to the listener. Similarly to the microphone encoding equation, the formula results in in three figure-eight microphones, each pointing in the x, y and z directions, and one omnidirectional W channel, with its origin at the center of the other three channels. In both cases the resulting B-format signals can be decoded over loudspeakers or binaurally over headphones.

Ambisonic decoding is a subtle and challenging art form. Psychoacoustic and physical considerations must be taken in order to create a *good* decoder. For this workshop, because of the time limitations, a simple ambisonic decoder will be shown. In the future this author would like to conduct another workshop specifically dealing with the difference between various types of decoders and how to implement these in JUCE.

The following formula shows how to encode the output of an FOA mic into corresponding B-format signals. The ordering here is not Ambix, as we will want for our plug-in. A simpler way to derive this equation is to consider the polarities of our three coincident figure-eight microphones. The positive side of the Z channel is above, the positive side of the X channel is in front and the positive side of the Y channel is on the left.

$$\begin{aligned} W &= FLU + FRD + BLD + BRU \\ X &= FLU + FRD - BLD - BRU \\ Y &= FLU - FRD + BLD - BRU \\ Z &= FLU - FRD - BLD + BRU \end{aligned} \tag{1}$$

Where:

1. FLU = Front Left Up
2. FRD = Front Right Down
3. BLD = Back Left Down
4. BRU = Back Right Up

The following shows the encoding equation for a FOA *panner*.

$$\begin{aligned} W &= s * (1/sqrt(2)) \\ X &= s * (cos(\phi)cos(\theta)) \\ Y &= s * (sin(\phi)cos(\theta)) \\ Z &= s * sin(\theta) \end{aligned} \tag{2}$$

* This work was supported by the UC San Diego

Where $\phi$ is our azimuthal angle and $\theta$ is our elevation angle. Equation (2) is the encoding equation only for first order ambisonic (FOA). To understand this formula, consider the simplest case. For channel X, when elevation equals 0, the reduced formula is simply a cosine function. When the azimuth is 0, then, we will have unity gain. At $\pi$ the phase will invert and the gain will also be one - this consisten with the output of a figure-eight microphone since these have negative phase for sounds arriving from the back. At any null points, or sides, when the angle is equivalent to $90^\circ$, the gain will be zero. In order to make B-format signals carry more-or-less equal average energy, X, Y, and Z channels have a gain of $sqrt(2)$ [2] or, alternatively, the W channel can be divided by this amount.

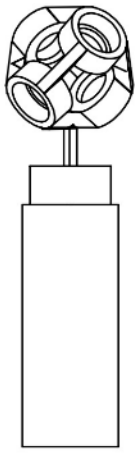

Figure 1: *Tetrahedral Mic*

For the decoder, we will use a decoding matrix taken from the Blue Ripple Sound technical notes. According to their site, the decoding coefficients originate from the implementation of Csound's *bformdec1* opcode converted to SN3D. Table 1 shows the decoding coefficients.

Table 1: *Quad Decoding Coefficients (FOA)*

| ACN | W | Y | X |
|---|---|---|---|
| FL | 0.25 | 0.177 | 0.177 |
| BL | 0.25 | 0.177 | -0.177 |
| BR | 0.25 | -0.177 | -0.177 |
| FR | 0.25 | -0.177 | 0.177 |

Note that this simple quadraphonic decoder is not *periphonic*: it does not include height information. To accomplish this decoder one simply has to take the W, Y and X channels, multiply them by the coefficients, and output them to the correct channel as a sum of vectors. Since the coefficients are the same except for the polarity we can simply multiply by negative one using a `for` loop and `if` statements. Naturally, I will suggest attendees to look for ways to optimize their algorithms independently, as this is often the most demanding and fulfilling part of software development.

### 2.1. Some more conventions

The naming convention used for our FOA microphone encoding equation is one of two naming conventions [3]. It is the same naming convention that is used by Soundfield and CoreSound microphone manufacturers. Sennheiser also uses this naming convention. An important thing to note is that the labeling of these capsule should always be done from the sound engineers perspective so that when the musician is facing the microphone the FLU capsule is actually on their right.

For our ambisonic *panning* equation which takes $\phi$ and $\theta$ for azimuth and elevation it is important to note that this is opposite to the standard American notation of spherical coordinate systems in which $\theta$ is used for azimuth and $\phi$ is used for elevation.

The right hand rule, however, is followed in the development of these plug-ins since this *is* something that is maintained by the community. The rule dictates that the azimuth angle should increase counterclockwise from $0^\circ$ degrees to $359^\circ$, where $180^\circ$ is directly behind the listener. The rule also dictates that, for elevation, values should span from $-180^\circ$ to $180^\circ$ with $90^\circ$ being exactly above the listener.

There are two main conventions for naming and normalizing B-format ambisonic signals:

FuMa = [W X Y Z]
(*Usually paired with MaxN normalization.*)
ACN = [W Y Z X]
(*usually paired with SN3D normalization (also called AmbiX).*)

In the development of these plug-ins we will use ACN ordering since it is this author's opinion that *it* is the more common ordering system used today. To determine our normalization values we used equation 3 in the Ambix specification paper [4].

The convention is to use the letter *n* for ambisonic order, in our case 1, and *m* for spherical harmonic number. The spherical harmonic number in our case will be 0 for W and 1 for channels X, Y, Z. [4] also suggests using $\delta_m$ which is 1 for $m = 0$ but 0 otherwise.

Plugging in those values we get normalization values of $sqrt(1/4\pi)$ for W and 0 for X, Y and Z. The author decided to not include the equation herein since the solution is very simple to derive for low ambisonic orders. This normalization will be applied to our array encoder and can replace the $1/sqrt(2)$ attenuation for our W channel in our panner VST.

## 3. THE JUCE FRAMEWORK

JUCE is a library of C++ methods integrated with powerful development tools ideal for anyone interested in developing audio software. Beyond allowing for complex audio processes and having built-in collections of familiar digital signal processing (DSP) routines, JUCE is also capable of some *fancy* graphic manipulation like animations and openGL 3D graphics. Beyond this, JUCE has also become recently one of the most popular development frameworks for audio software and is thus in high-demand.

### 3.1. DSP Terminology and Process

One of the fundamental aspects of software development is making the interface simple to understand. To that end the beginning section of every VST development will focus on making an intuitive interface that gives *all* the information necessary to the user.

In our case we need to graphically represent the FOA array on the Graphical User Interface (GUI), for our array encoder, and specify which ordering and normalization scheme is being implemented. In order to quickly represent the four capsules' names and channels the instructor will provide attendees with a transparent image that they can load into their VST. The author will also provide them with A-format recordings so they can *test* their VSTs. We will manually incorporate other elements of the VST such as the title and supplemental information. This specific GUI does not need any sliders - perhaps we could incorporate a gain slider but it really isn't *necessary*. Time permitting the author might add some features as informed by similar software developed by major audio companies and independent audio engineers.

For the ambisonic panner we will create an *animation* that represents the position of the source on the azimuth plane. The position of the source on the azimuth place can be easily calculated by passing the value from our azimuth slider to a simple trigonometric function. Each time the *canvas* is re-drawn, we can then update the position of our source around the circle. Finally, for the decoder, we will focus on having a graphical description of the position of the speakers, to inform the listener exactly what kind of decoder this is. For this we will also use a transparent image, since this will allow more time for questions or other improvements.

After the GUI has been developed, in each instance, the workshop will demonstrate to participants how to attain values from their sliders, add sliders to a value tree state (VTS), which allows the user to modulate parameters, and how to use read and write pointers to modify data being passed to our processing function in a buffer.

## 4. PREREQUISITES

Participants entering this workshop should be familiar with programming terminology and general coding practices. The participant does not need to be an expert but should have taken two or perhaps three programming courses in the past. The attendee should preferably have some ambisonics experience, or alternatively, recording engineering experience. It should not be necessary to limit the number of participants.

## 5. CONCLUSIONS

This workshop proposal has outlined the concepts and process this author hopes to undertake at the LAC 2019. The workshop will hopefully inspire young students and new audio engineers to develop more ambisonic tools and keep learning the JUCE development framework, which is currently widely utilized in the audio industry. The workshop will also hopefully be valuable for anybody not yet familiar with the JUCE framework since we will cover how to create sliders and use their values to modify sound.

## 6. ACKNOWLEDGMENTS

I want to thank Tom Erbe of UCSD, MakeNoise and SoundHack for teaching me how to make these plug-ins last fall quarter at UCSD as a PhD candidate. I hope to make many more and keep improving them under his tutelage.

## 7. REFERENCES

[1] Stephen Thornton, *Michael Gerzon - Audio Pioneer*, 2009.

[2] Michael A Gerzon, "Practical periphony: The reproduction of full-sphere sound," in *Audio Engineering Society Convention 65*. Audio Engineering Society, 1980.

[3] Angelo Farina, "A-format to b-format conversion," 2006.

[4] Christian Nachbar, Franz Zotter, Etienne Deleflie, and Alois Sontacchi, "Ambix-a suggested ambisonics format," in *Ambisonics Symposium, Lexington*, 2011.