Ambitools: Tools For Sound Field Synthesis Using Higher Order Ambisonics - V1.0

Pierre Lecomte

▶ **To cite this version:**

Pierre Lecomte. Ambitools: Tools For Sound Field Synthesis Using Higher Order Ambisonics - V1.0. 1st International Faust Conference (IFC-18), Jul 2018, Mainz, Germany. hal-03162948

HAL Id: hal-03162948
https://hal.science/hal-03162948

Submitted on 11 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AMBITOOLS : TOOLS FOR SOUND FIELD SYNTHESIS WITH HIGHER ORDER AMBISONICS - V1.0

Pierre Lecomte

LVA - INSA de Lyon
25 Avenue Jean Capelle
69621 Villeurbanne cedex, France
pierre.lecomte@gadz.org

ABSTRACT

This paper presents the toolbox Ambitools: an implementation of several Ambisonic tools with the FAUST language. The code is designed to be scalable and flexible, offering tools working at various Ambisonic order and compiled for various architectures. The implementation of the spherical harmonics for an efficient computation is detailed. Also, the implementation procedure for radial filters encountered with Ambisonics is described. Finally, an overview of all the tools available in the current version is given.

1. INTRODUCTION

Higher Order Ambisonics (HOA), simply called Ambisonics throughout this paper, is a set of techniques to capture, manipulate and synthesize sound pressure fields [1]. The flexibility and scalability of the approach makes it possible to synthesize sound fields over a wide range of configurations, from loudspeaker arrays to binaural synthesis with headphones. With the increasing demand of spatial audio applications, especially for virtual reality, several implementations of Ambisonics are nowadays available on the market [2, 3, 4, 5, 6, 7, 8]. The implementation presented in this paper, denoted "Ambitools", is one of them. The core Digital Signal Processing (DSP) is described in FAUST language. Ambitools development started during the author's PhD thesis [9] and a first presentation was made in 2015 on this topic [7]. Since then, the code has been re-designed, and Ambitools won the FAUST award 2016. [1] [2] The development of Ambitools tries to fulfill three main objectives:

1. **Real-time DSP.** The first objective is to implement a real-time synthesis and manipulation of the Ambisonic sound scene in order to offer maximal interactivity. In this context FAUST is of great help as its compiler provides efficient C++ code. However, even if the FAUST compiler performs several rounds of algebraic simplifications on the DSP algorithm, one has to keep in mind that Ambisonics deals with spherical coordinates, therefore involves trigonometry. Trigonometric simplifications for the code is a non trivial topic and special attention is needed to reduce the number of operations in the DSP algorithm.
2. **Scalable code.** The second objective of Ambitools is to provide scalable code. In fact, Ambisonics is a scalable technique which can be described at different Ambisonic orders. The spatial resolution of the sound field increases as the Ambisonic order increases (Sec. 3.2). Thus, the implementation should provide a straightforward possibility of compilation at arbitrary Ambisonic order.
3. **Flexible code.** The third objective of the code is to be flexible. That is to say, the user should be able to customize the Graphical User Interface (GUI) and DSP by setting only a few parameters in the code at compilation time. For instance, the choice of VU-Meters and/or Mute-toggle is available by changing a variable value (Sec. 6.11) . Moreover, the code should produce plug-ins in various format architecture, which is the essence of FAUST.

The paper is organized as follows: Definitions are introduced in Sec. 2. A brief overview of Ambisonics principle is done in Sec. 3. Then, a description of FAUST spherical harmonic implementation is given in Sec. 4. In the same vein, the implementation of radial filters needed with Ambisonics is introduced in Sec. 5. An overview of the FAUST tools available in the current version is given in Sec. 6. Finally, some extras tools are presented in Sec. 7.

2. DEFINITIONS

2.1. Spherical Coordinate System

The spherical coordinate system used in Ambitools is given by:

$$x = r\cos(\theta)\cos(\delta),\ y = r\sin(\theta)\cos(\delta).\ z = r\sin(\delta), \quad (1)$$

and shown in Fig. 1:

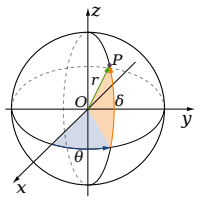


Figure 1: Spherical coordinate system. A point $P(x, y, z)$ is described by radius r, azimuth θ and elevation δ.

2.2. Spherical Harmonics

The real-valued spherical harmonics are used in Ambitools. They are denoted $Y_{m,n}$ and are defined as follows:

$$Y_{m,n}(\theta,\delta) = \mathcal{N}_{m,n} P_{m,|n|}(\sin(\delta)) \begin{cases} \cos(|n|\theta) & \text{if} \quad n \geq 0 \\ \sin(|n|\theta) & \text{if} \quad n < 0 \end{cases}. \quad (2)$$

[1] http://faust.grame.fr/news/2016/10/17/Faust-Awards-2016.html

[2] All URLs in this paper were verified on March 27, 2018.

In Eq. (2), $(m, n) \in (\mathbb{N}, \mathbb{Z})$ with $|n| \leq m$, $P_{m,|n|}$ are the associated Legendre polynomials of degree m and order $|n|$. $\mathcal{N}_{m,n}$ is a normalization factor. For each Ambisonic order m, there are $(2m + 1)$ spherical harmonics. Thus, a basis truncated at Ambisonic order M contains $(M + 1)^2$ functions.

In Ambitools, the full 3D normalization factor (N3D), denoted $\mathcal{N}^{\text{N3D}}_{m,n}$ is used [1]:

$$\mathcal{N}^{\text{N3D}}_{m,n} = \begin{cases} \sqrt{2m+1} & \text{if } n = 0 \\ \sqrt{2(2m+1)\dfrac{(m-|n|)!}{(m+|n|)!}} & \text{if } n \neq 0 \end{cases}. \tag{3}$$

This normalization factors ensures that the family of spherical harmonics constitutes an orthonormal basis in $L^2(\mathbb{S}^2)$: the vector space of square-integrable functions on the unit sphere $\mathbb{S}^2 : \left\{(x, y, z) \in \mathbb{R}^3,\ x^2 + y^2 + z^2 = 1\right\}$. The orthonormality property is often required for physical decoding [10] or accurate sound field transformation [11]. A review of the different normalization factors and their impacts in Ambisonics is provided in [12].

The spherical harmonics can be indexed using different conventions. In Ambitools, the default spherical harmonics indexing follows the Ambisonic Channel Number (ACN) convention [13]:

$$\text{ACN} = m^2 + m + n. \tag{4}$$

From the ACN number, the subscripts m and n are given by:

$$\begin{aligned} m &= \lfloor \sqrt{\text{ACN}} \rfloor \\ n &= \text{ACN} - m^2 - m \end{aligned} \tag{5}$$

where $\lfloor \cdot \rfloor$ is the integer part of the argument.

3. AMBISONICS PRINCIPLES

This section briefly introduces the principles of Ambisonics. More details can be found in [10] and references cited therein.

3.1. Fourier-Bessel Series

A sound pressure field in a spherical volume of radius r_s, where the acoustic sources are outside the domain, can be described with the Fourier-Bessel series, given in the frequency domain by [1]:

$$p(r, \theta, \delta, f) = \sum_{m=0}^{\infty} i^m j_m(2\pi f/cr) \sum_{n=-m}^{m} B_{mn}(f) Y_{mn}(\theta, \delta). \tag{6}$$

In Eq. (6), p is the acoustic pressure, c is the speed of sound, f the frequency, i the imaginary unit ($i = \sqrt{-1}$), and j_m are the spherical Bessel functions of order m. Following this approach, the sound pressure field is fully described in the spherical volume if one knows the coefficients B_{mn}, called the Ambisonics components.

3.2. Spatial Resolution

From Eq. (6) one observes that the sound pressure field is described with an infinite sum over Ambisonic orders m. However, for practical applications, the sum is truncated at a maximum Ambisonic order M. This truncation limits the spatial resolution of the sound field. A rule-of-thumb for the sound pressure field approximation with finite order Ambisonics is given in a spherical volume of radius r [14]:

$$r = \frac{Mc}{2\pi f}. \tag{7}$$

From Eq. (7), one observes that this "sweet spot" size decreases with increasing frequency when working at a fixed order M.

3.3. Ambisonics Workflow

The Ambisonic technique allows for the synthesis and manipulation a sound pressure field by operating on the Ambisonics components B_{mn}. A typical Ambisonic workflow is schematized in Fig. 2:

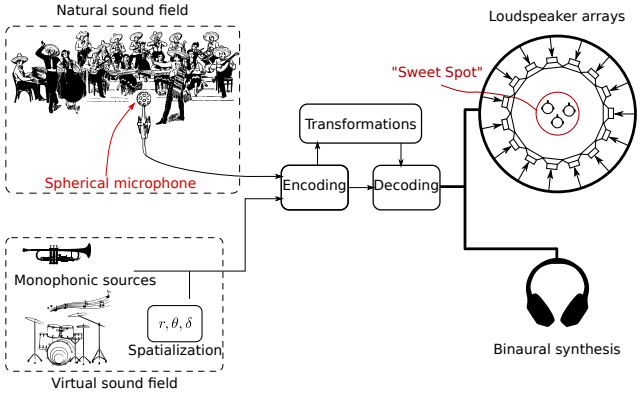


Figure 2: Ambisonic workflow: a natural sound pressure field is captured with a spherical microphone array and encoded in the Ambisonic domain. Alternatively, monophonic signals are encoded in space as simple acoustic sources. Once in the Ambisonic domain, transformations can be applied to the sound scene. Finally, the sound scene is decoded on various loudspeakers arrays or over headphones.

3.3.1. *Encoding*

Working with Ambisonic order M, the Ambisonic components are stored in a vector of signals $\mathbf{b}(z) \in \mathbb{R}^{(M+1)^2 \times 1}$, which gives with ACN indexing:

$$\mathbf{b}(z) \ = \ [b_0(z), b_1(z), \cdots b_{(M+1)^2-1}(z)]^T, \tag{8}$$

where ^{T} is the transpose operator.

For spatialization, analytical expressions are known for simple acoustic sources, such as plane or spherical waves [1, 7]. This is schematized in the bottom left branch of Fig. 2

Also, it is possible to estimate the Ambisonic components of a natural sound pressure field by means of Spherical Microphone Arrays (SMA) [15]. This is schematized in the top left branch of Fig. 2

3.3.2. *Ambisonic Sound Scene Manipulation*

Once in the Ambisonic domain, the sound scene is manipulated by doing matrix operations on the input vector $\mathbf{b}$ [16]:

$$\tilde{\mathbf{b}} = \mathbf{T}\mathbf{b}. \tag{9}$$

In Eq.(9) $\tilde{\mathbf{b}}(z) \in \mathbb{R}^{(\tilde{M}+1)^2 \times 1}$ is the output Ambisonic sound scene described up to order $\tilde{M}$. The sound field transformation is described in the matrix $\mathbf{T} \in \mathbb{R}^{(\tilde{M}+1)^2 \times (M+1)^2}$ which is not necessarily square. In fact, some transformation may require a description of the output Ambisonic sound scene at higher orders [17, 11].

3.3.3. Decoding

The decoding step aims to synthesize the sound pressure field corresponding to the Ambisonic sound scene with a loudspeaker array or over headphone through binaural synthesis [18]. This is schematized on the right side of Fig. 2. Various strategies are available to design a decoder adapted to the loudspeaker configuration. A good review is provided in references [2, 19].

4. SPHERICAL HARMONICS IMPLEMENTATION

In Ambitools, the spherical harmonics are implemented in a FAUST library file called `ymn.lib`. Several strategies can be used to implement the spherical harmonics with FAUST. In the current implementation, one assumes that evaluating the trigonometric functions cos and sin in Eq. (2) is costly and should be done a minimum amount of time. Moreover, a scalable implementation is desirable: the computation of a spherical harmonics can be done at arbitrary Ambisonic order m and degree n.

4.1. Hard-Coded Implementation

4.1.1. Straightforward implementation

A first approach for the implementation would be to pre-compute closed-form of the spherical harmonics $Y_{m,n}$ up to a maximum order M and hard-code the resulting expression with pattern matching in the library. For instance, evaluation of Eq. (2), for $(m = 3, n = 2)$, gives:

$$Y_{3,2}(\theta, \delta) = \frac{1}{2}\sqrt{105}\sin(\delta)(1 - \sin(\delta)^2)\cos(2\theta) \qquad (10)$$

The corresponding FAUST code is:

```
ymn(3,2,t,d) = 0.5*sqrt(105)*sin(d)*(1 -
    sin(d)^2)*cos(2*t);
```

where `t` and `d` stand for θ and δ variables respectively. In this case, the sin and cos functions are called once each.

For arbitrary subscripts (m, n), the associated Legendre polynomial $P_{m,|n|}(\sin(\delta))$ in Eq. (2) can be expressed as a polynomial of the variable $\sin(\delta)$. Consequently, the implementation requires the evaluation of $\sin(\delta)$ and $\cos(n\theta)$ or $\sin(n\delta)$, depending on subscripts (m, n).

Note that further trigonometric simplifications could be operated, such as $1 - \sin(\delta)^2 = \cos(\delta)^2$ in Eq. (10). However, by doing so, a supplementary $\cos(\delta)$ function should be evaluated, which may be more costly than using only one evaluation of $\sin(\delta)$ and polynomial operations.

Furthermore, this approach can become costly when a vector of spherical harmonics up to Ambisonic order M is required:

$$\mathbf{y}^{(M+1)^2 \times 1}(\theta, \delta) = [Y_{0,0}(\theta, \delta) \cdots Y_{m,m}(\theta, \delta)]^T \qquad (11)$$

In fact, the FAUST compiler evaluates $\sin(\delta)$, in a variable and uses it for the different $P_{m|n|}(\sin(\delta))$ polynomial evaluation. But, $2M$ additional variables are also required for $\cos(n\theta), \sin(n\theta)$, with $n \in \{-M, -M+1 \cdots -1\} \cup \{1, 2 \cdots M\}$. As an example, if one requires $\mathbf{y}(\theta, \delta)$ at $M = 3$, the generated C++ code calls $1 + 2 \times 3 = 7$ times the sin and cos functions.

4.1.2. Cartesian formulation

To reduce the number of trigonometric functions evaluations, an implementation using Cartesian coordinates is possible. The principle is to express the variables θ and δ in Cartesian coordinates, using Eq. (1), and express the spherical harmonic as a polynomial of x, y, z variables. In the case of $Y_{3,2}(\theta, \delta)$ the FAUST code would be:

```
ymn(3,2,t,d) = 0.5*sqrt(105)*z*(x^2-y^2)
    with
{
x = cos(t)*cos(d);
y = sin(t)*cos(d);
z = sin(d);
};
```

Following this approach, the trigonometric function cos and sin are called four times: twice for θ and twice for δ. Then, a polynomial expression of x, y, z variables is computed. In this case, when a vector $\mathbf{y}^{((M+1)^2 \times 1)}(\theta, \delta)$ is required, the compiler will call 4 times the sin and cos functions and the resulting variables x, y and z are shared for the evaluation of the different spherical harmonics.

The main drawback of the aforementioned approach is that preliminary simplifications are operated on the spherical harmonics expression before implementation. These simplifications are different depending on the subscript (m, n). As a consequence, the implementation is hard-coded and limits the code scalability up to a maximum order M, where the spherical harmonics are implemented.

4.2. Scalable Implementation with Recurrence Formulas and Trigonometric Expansion

In order to fulfill the scalability objective mentioned in Sec. 1, an alternate approach is proposed in the current version of Ambitools. The computation of a spherical harmonic at arbitrary Ambisonic order m and degree n is done using recurrence formulas. Special care is taken to limit the number of trigonometric functions and numerical errors at higher orders. The different terms of Eq. (2) are considered separately for the implementation.

4.2.1. Normalization factor implementation

The normalization factor of Eq. (3) are implemented in FAUST as follows:

```
n3d(m,n) =  sqrt((2*m+1)*factorial(m-abs(n)
    )/factorial(m+abs(n)))*
case{
(0) => 1;
(n) => sqrt(2);
}(n);
```

where `factorial` is the factorial function, which implementation is described in Sec. 4.2.2.

4.2.2. Factorial function numerical issues

The factorial function, denoted !, is needed for the normalization factor $\mathcal{N}_{m,n}^{\text{N3D}}$ in Eq. (3), and for the double factorial function !! of Eq. (16). A straightforward implementation of this function can be done by recurrence as follows:

```
factorial = case {
(0) => 1;
(n) => n*factorial(n-1);
};
```

However, this implementation produces wrong results as n increases. For instance FAUST compiler (v-2.5.21) output of:

```
process = factorial(13);
```

gives 1932053504, instead of 6227020800. This error may be due to the finite precision arithmetic used in FAUST compiler. As an example, the factor $\mathcal{N}_{7,6}^{\text{N3D}}$ has a wrong numerical value with this implementation and so will be the corresponding spherical harmonic.

To circumvent this issue and produce more accurate results at higher orders, one uses an implementation with the Gamma function Γ available in the FAUST standard libraries. In fact, one has the following formula:

$$\Gamma(n+1) = n!. \tag{12}$$

Thus, the corresponding FAUST code is simply:

```
factorial(n) = ma.gamma(n+1)
```

4.2.3. Associated Legendre polynomial

Recalling Eq. (2), the associated Legendre polynomial $P_{m,n}(x)$ can be expressed with the following recurrence formula [20, Eq. 12.92, p. 775]:

$$P_{m,n}(x) = \frac{1}{m-n}\left((2m-1)xP_{m-1,n}(x) - (m-1+n)P_{m-2,n}(x)\right). \tag{13}$$

In Eq. (13), two special situations should be considered. The first one is when $m = n$, which produces a division by 0. The second is when $n = m-1$, in which case the term $P_{m-2,n}(x) = P_{m-2,m-1}(x)$ is not defined, as the condition $|n| \leq m$ is not fulfilled. Thus, additional identities are used:

$$P_{m,m}(x) = (2m-1)!!(1-x^2)^{(m/2)}, \tag{14}$$
$$P_{m,m-1}(x) = x(2m-1)P_{m-1,m-1}(x), \tag{15}$$

In Eq. (14), !! is the double factorial operator [20]. This latter can be expressed in term of factorial operator ! as follows:

$$n!! = \begin{cases} 2^{n/2}\left(\frac{n}{2}\right)! & \text{if } n \text{ even}, n \geq 0 \\ \frac{(n+1)!}{2^{(n+1)/2}\left(\frac{n+1}{2}\right)!} & \text{if } n \text{ odd}, n \geq 1 \end{cases}. \tag{16}$$

The corresponding FAUST implementation gives:

```
factorial2(0) = 1;
factorial2(1) = 1;
factorial2(n) = case{
(1) => 2^(n/2)*factorial(n/2);// n even
(0) => factorial(n+1)/(2^((n+1)/2)*
    factorial((n+1)/2));// n odd
}(n%2==0);
```

Finally, the recurrence is initialized with $P_{0,0}(x) = 1$. The FAUST code for the associated Legendre function $P_{m,n}(x)$ is given by:

```
alegendre(0,0,x) = 1;
alegendre(m,n,x) =  case{
(1,0) => factorial2(2*m-1)*(1-x^2)^(m/2);
(0,1) => x*(2*m-1)*alegendre(m-1,m-1,x);
(0,0) => 1/(m-n)*((2*m-1)*x*alegendre(m-1,n
    ,x)-(m-1+n)*alegendre(m-2,n,x));
}(n==m,n==(m-1));
```

4.2.4. Trigonometric expansion with Chebyshev polynomial

As pointed out in Sec. 4.1.1, the terms $\cos(n\theta)$ and $\sin(n\theta)$ in Eq. (2) produce with the FAUST compiler as many variables as there are values of n to handle. In the case of a spherical harmonics vector, $\mathbf{y}^{(M+1)^2\times 1}(\theta,\delta)$, there are $2M$ different value of n. To reduce this number, Chebyshev polynomials are used for trigonometric expansion. In fact, one has [20]:

$$\cos(n\theta) = T_n(\cos(\theta)), \tag{17}$$
$$\sin(n\theta) = U_{n-1}(\cos(\theta))\sin(\theta), \tag{18}$$

where T_n and U_n are the Chebyshev ponylomials of the first and second kind respectively. These polynomials can be implemented in FAUST with recurrence formulas:

```
chebyshev1(n,x) = case{
(0) => 1;
(1) => x;
(n) => 2*x*chebyshev(n-1,x) - chebyshev(n
    -2,x);
}(n);


chebyshev2(n,x) = case{
(0) => 1;
(1) => 2*x;
(n) => 2*x*chebyshev2(n-1,x) - chebyshev2(n
    -2,x);
}(n);
```

where `chevychev1(n,x)` and `chevychev2(n,x)` stand for $T_n(x)$ and $U_n(x)$ respectively. These trigonometric expansions produce polynomials of variables $\cos(\theta)$ and $\sin(\theta)$ respectively.

4.2.5. Spherical harmonics implementation

Finally, the spherical harmonics $Y_{mn}(\theta,\delta)$ is implemented in FAUST as follows:

```
ymn(m,n,t,d) = n3d(m,n)*alegendre(m,abs(n),
    sin(d))*
case{
(1) => chebyshev2(abs(n)-1,cos(t))*sin(t);
```

```
(0) => chebychev1(abs(n),cos(t));
}(n<0);
```

With this implementation, a vector of spherical harmonics $\mathbf{y}^{((M+1)^2\times 1)}(\theta,\delta)$ now uses only three trigonometric function, namely $\sin(\delta)$, $\cos(\theta)$ and $\sin(\theta)$, which are shared for the different spherical harmonics evaluation.

The implementation with ACN indexing of Eq. (4) is done with the following FAUST code:

```
yacn(i,t,d) = ymn(m,n,t,d) with
{
m = int(sqrt(i));
n = int(i - m^2 -m);
};
```

5. RADIAL FILTERS IMPLEMENTATION

In the current version of Ambitools, two types of radial filters are encountered: near field filters and equalization filters for rigid SMA.

5.1. Near Field Filters

The near field filters are used to encode the near-field of acoustic sources at radius r_1 [21]. Those filters are denoted $F_{m,r_1}(z)$. Their expression in the frequency domain is [10]:

$$F_{m,r_1}(f) = \frac{i^{-(m+1)}}{4\pi}\frac{2\pi f}{c}h_m^{(2)}(2\pi f/cr_1), \tag{19}$$

where c is the speed of sound and $h_m^{(2)}$ are the spherical Hankel function of second kind.

For the decoding of a sound field with a spherical loudspeaker array of radius r_0, the near field of the loudspeakers is compensated with the inverse of near field filters: $1/F_{m,r_0}(z)$ [21].

5.2. SMA Equalization Filters

Rigid SMA are used to capture a 3D sound field and encode it in the Ambisonic domain. To do so, the first step is to operate a Discrete Spherical Fourier Transform (DSFT) from the different capsule signals and equalize the output components with equalization filters [10]. Those filters depend on the microphone radius r_2 and are denoted $E_{m,r_2}(z)$. Their expression in the frequency domain is:

$$E_{m,r_2}(f) = i^{-m+1}\left(\frac{2\pi f}{c}r_2\right)^2 h'^{(2)}_m(2\pi f/cr_2), \tag{20}$$

where $h'^{(2)}_m$ are the first derivative of the spherical Hankel function of second kind.

5.3. Procedure of Implementation

The implementation of filters of Eqs. (19) and (20) is done in the form of a recursive filter. To do so, they are first described as polynomials of variable s in the analogue Laplace domain. This results in the expressions $F_{m,r}(s)$ and $E_{m,r}(s)$, respectively. Then, the polynomials are factorized in First Order Sections (FOS) and Second Order Sections (SOS). Finally, a s-to-z mapping is applied to each section with the bilinear transform [21] or corrected impulse invariance method [22]. Note that the near-field filters $F_{m,r}(z)$ and $E_{m,r}(z)$ are both unstable as they have a pole at $z=0$. In Ambitools, they are stabilized by the near-field compensated filters for a spherical loudspeaker array of radius r_0: $1/F_{m,r_0}(z)$. Therefore, the implemented filters are:

$$\mathrm{H}_{m,r_1,r_0}(z) = \frac{F_{m,r_1}(z)}{F_{m,r_0}(z)} \tag{21}$$

$$\mathrm{Eq}_{m,r_1,r0}(z) = \frac{E_{m,r_1}(z)}{F_{m,r_0}(z)} \tag{22}$$

The detailed implementation of filters $\mathrm{H}_{m,r_1,r_0}(z)$ is described in [7]. The implementation of filters $\mathrm{Eq}_{m,r_1,r_0}(z)$ is inspired by the method described in [22].

The radial filters are available in a library called `radial.lib`: The near-field filters are denoted `nf(m,r1,r0)` and correspond to the filters $\mathrm{H}_{m,r_1,r_2}(z)$ of Eq. (22). The near field compensation filters are denoted `nfc(m,r0)` and correspond to the filters $1/F_{m,r_0}(z)$. Finally the equalization filters for rigid SMA are denoted `eq(m,r2,r0)` and correspond to the filters $\mathrm{Eq}_{m,r_2,r_0}(z)$ of Eq. (22).

An additional Finite Impulse Response (FIR) implementation of the rigid SMA equalization filters $E_{m,r_2}(z)$ of Eq. (20) is also available. The filters are stabilized with a Tikhonov regularization as proposed in [15]. As the FIR filters have several thousands taps, a fast convolution software is recommended (see Sec. 7.3).

5.4. Scalability Limitation

In the implementation procedure, factorization of the polynomials $F_{m,r}(s)$ and $E_{m,r}(s)$ into FOS and SOS requires to compute their roots. This is not a trivial task and it is not available in FAUST. As a consequence, the FOS and SOS coefficients are pre-computed and stored manually in the library `radial.lib`. This limits the scalability of the code up to a maximum order M, where the FOS and SOS coefficients are available. In the current version of Ambitools the radials filters are available up to order $M_{\text{max}} = 10$. Note that at higher order, such as $M \geq 7$, numerical instabilities are observed in the current implementation. Moreover, unrealistic excessive amplification at low frequencies arises [21].

6. OVERVIEW OF THE FAUST TOOLS

This section describes briefly the different tools available in Ambitools in the version 1.0. Most of the tools are scalable to arbitrary order M. However, some transformation requires pre-computation of coefficients, which limits the scalability up to a maximum order M_{max}. As well, the decoders involving spherical grids are limited to a maximum Ambisonic order M_{max}.

6.1. hoa_encoder

- Inputs: N
- Outputs: $(M+1)^2$

This tool encodes N monophonic signals as N sources in space with Ambisonics. The variable N is set in the code at compilation time. The output sound scene is described up to Ambisonic order M with $(M+1)^2$ components. Two types of acoustic sources are proposed: plane wave or spherical wave. In the case of the spherical wave, near field filters are involved [7] (Sec. 5). In this case, it is required to set a spherical loudspeaker array radius r_0 in the text-box **`Speaker radius`**. Finally, for each source, the

spatial coordinates and gain are set with sliders **Gain**, **Radius**, **Azimuth** and **Elevation**.

6.2. hoa_decoder_*

- Inputs: $(M+1)^2$
- Outputs: L

Several decoders for spherical loudspeakers arrays are available in the current version of Ambitools. They are based on different L-node Lebedev grids [10]. The decoders can be near field compensated at execution time (Sec. 5) with a check-box **NFC**. In this case, the spherical loudspeaker array radius r_0 is required in a text-box **Speaker Radius** at execution time. The user can adjust the inputs and outputs global gain with the sliders **Inputs Gain** and **Output Gains**.

6.3. hoa_panning_*

- Inputs: N
- Outputs: L

For some particular grids, equivalent panning law can be used to compute the L loudspeaker signals, without the necessity of going into the Ambisonic domain [7]. In the current version, the panning laws are proposed for various Lebedv grids [10]. Thus, with these tools, the user can spatialize N sources on the proposed grids. As for the `hoa_encoder` tool (Sec. 6.1), the parameters **NFC**, **Speakers radius**, **Gain**, **Radius**, **Azimuth** and **Elevation** are adjustable at execution time with sliders.

6.4. hoa_decoder_binaural_lebedev50

- Inputs: $(M+1)^2$
- Outputs: 2

Ambisonics allows for the decoding of a sound scene for binaural synthesis with headphones. The principle is described in [18]: The sound pressure field is firstly decoded on a virtual loudspeaker grid. Then, each virtual loudspeaker signal is filtered with the corresponding Head Related Impulse Response (HRIR) pair. The resulting signals for left and right ears are finally summed to obtain the headphone signals. Following this approach, Ambitools proposes a binaural decoder using a 50-node virtual Lebedev grid, which works up to Ambisonic order $M_{\text{max}} = 5$ [10]. The HRIRs used are from a Neumann©KU-100 dummy head [23]. At execution time, the input and output gains are adjusted with sliders **Inputs Gain** and **Outputs Gain** respectively.

Note that with this FAUST tool, the DSP involves massive linear convolutions, which are very costly for real-time synthesis. Therefore, an alternate approach is available with several HRIRs provided as FIR filters and a fast convolution software (Sec 7.3).

6.5. hoa_converter_*

- Inputs: $(M+1)^2$
- Outputs: $(M+1)^2$

As pointed in Sec. 4, various normalization conventions and channel orderings are available with Ambisonics [13, 12]. For compatibility with other Ambisonics software, Ambitools provides various converters. Scaling factors and channel rearrangement are described in the FAUST code. In the current version, the available Ambisonics formats are : ACN-SN3D, ACN-N3D and FuMa.

6.6. hoa_mic_encoder_*

- Inputs: 6, 12, 26, 50 or 32
- Outputs: $(M+1)^2$

These tools operate the DSFT [7] for various SMA. In the current version, the available grids are 6, 12, 26, 50-node Lebedev grids and 32-node EigenMike grid [24]. The maximum Ambisonic order M_{max} where the DSFT operation is accurate up to Ambisonic orders $1, 2, 3, 5$ and 4 respectively. A global gain on the inputs is set at execution time with a slider **Gain**.

Note that the equalization filters $\text{Eq}_{m,r_2}(z)$ of Eq. (22) are not yet included in these tools. However, this equalization is mandatory to accurately estimate the Ambisonic components.

6.7. hoa_mirroring

- Inputs: $(M+1)^2$
- Outputs: $(M+1)^2$

This tool operates a mirroring transformation on the Ambisonic sound scene. The front-back, left-right and top-down directions can be inverted by changing the sign of several Ambisonic channels. This transformation exploits the symmetries of the spherical harmonics [25]. At execution time, the user can activate/bypass the direction inversion with check-boxes **front-back**, **left-right** and **up-down**.

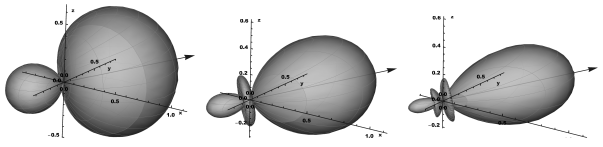
6.8. hoa_beamforming_*

In an Ambisonic sound scene, directions can be enhanced or attenuated [16]. Directional filtering is implemented in Ambitools for hypercardioid beampatterns and directionnal Dirac function, as described in [11].

6.8.1. hoa_beamforming_hypercardioid_to_mono

- Inputs: $(M+1)^2$
- Ouputs: 1

This tool extracts a monophonic signal from the Ambisonic sound scene as if it was recorded with a virtual microphone placed at the origin. The virtual microphone has a hyper-cardioid beampattern of order M' [26]. Currently, the hyper-cardioid beampatterns are implemented up to order $M'_{\text{max}} = 5$. The first three are shown in Fig. 3. At execution time, the user can set the steering angle of the hyper-cardioid, denoted (θ_0, δ_0) with the sliders **Azimuth**, **Elevation** respectively. Also the hyper-cardioid order is chosen with the slider **Order** and the output gain with the slider **Gain**.



(a) $M' = 1$ (b) $M' = 2$ (c) $M' = 3$

Figure 3: Hyper-cardioids beampattern at various order with steering angle $\theta_0 = 20°, \delta_0 = 0°$.

6.8.2. *hoa_beamforming_hypercardioid_to_hoa*

- Inputs: $(M+1)^2$
- Outputs: $(M+M'+1)^2$

This tool applies a directional filtering on the Ambisonic sound scene, according to a hyper-cardioid beampattern. It outputs a new Ambisonic sound scene where the directions have been altered accordingly. As shown in [11], this transformation requires a higher Ambisonic order for the outputs, depending on the order of the hypercardioid. The output scene Ambisonic order is denoted $\tilde{M}$ and is such that:

$$\tilde{M} = M + M', \tag{23}$$

where M is the input Ambisonic sound scene order and M' is the hypercardioid order. For instance, if one filters a 5-order Ambisonic sound scene with a 2-order hypercardioid, the output Ambisonic sound scene order is $\tilde{M} = 5 + 2 = 7$. In the current version, the maximum order for the output Ambisonic scene is $\tilde{M}_{\text{max}} = 10$. At execution time, the user can set the steering angle of the hypercardioid, its order, and adjust the outputs gains with sliders **`Azimuth`**, **`Elevation`**, **`Order`** and **`Gain`** respectively.

6.8.3. *hoa_beamforming_dirac_to_hoa*

- Inputs: $(M+1)^2$
- Outputs: $(M+1)^2$

This tool retains only one direction in the Ambisonic sound scene. In fact, the beampattern used here is a truncated directional Dirac [11]. That is to say, a function which is zero everywhere, except in the steering angle direction. Thus, this tool allows listening to only one direction in the sound field. At execution time, the user can set the steering angle of the directional Dirac with the sliders **`Azimuth`** and **`Elevation`**. Moreover, a crossfader to bypass the effect is provided. The duration of the crossfader is set with the text-box **`Crossfade duration`**.

6.9. hoa_azimuth_rotator

- Inputs: $(M+1)^2$
- Outputs: $(M+1)^2$

This tool applies a rotation of the Ambisonic sound scene around the z-axis, corresponding to yaw angle. The rotation matrix is described in the spherical harmonic domain [1]. At execution time, the user can set the rotation angle with the **`Azimuth`** slider. This tool can notably be used to compensate the head rotation around z-axis for a head-tracked binaural synthesis (Sec. 7.2).

6.10. hoa_rotator

- Inputs: $(M+1)^2$
- Outputs: $(M+1)^2$

This tool allows to rotate the Ambisonic sound scene around x-, y- and z-axes, corresponding to roll, pitch and yaw angles respectively. The implementation is done with recurrence formulas as described in [27]. At execution time, the user can set the rotation angles with the sliders **`yaw`**, **`pitch`** and **`roll`**. This tools can notably be used to compensate the head rotation for a head-tracked binaural synthesis (Sec. 7.2).

6.11. VU-Meters Flexibility

For most tools, the GUI VU-Meters for the inputs and outputs channels are respectively controlled by the variables `insvumeter` and `outsvumeter` in the code. The user can set one of the following value for these variables, at compilation time:

- `0`: no VU-Meters are generated in the GUI.
- `1`: VU-Meters are generated and grouped by Ambisonic order
- `2`: VU-Meters are generated and grouped by Ambisonic order. Moreover a mute check-box is available for each channel and for all the channels of an Ambisonic order.

7. EXTRAS TOOLS

On top of the DSP tools implemented in FAUST, Ambitools provides several side tools presented in the next sections.

7.1. 3D VU-Meters

A flexible 3D VU-Meter is implemented with the PROCESSING language. An example of the meter for a 50-node Lebedev grid is shown in Fig. 4. The loudspeakers are represented with balls in space, whose size and color changes according to the RMS level in dBFS. A human head is placed at the origin and is facing the front direction. In the code, the number and position of the loudspeakers are set in a `.cvs` file which can be easily adapted to the user configuration. Moreover, in a spatialization context, the acoustic sources are represented with little colored balls with fading trajectories. The 3D VU-Meter is driven with Open Sound Control (OSC) messages, generated by the FAUST tools `hoa_encoder` and `hoa_decoder_*`.

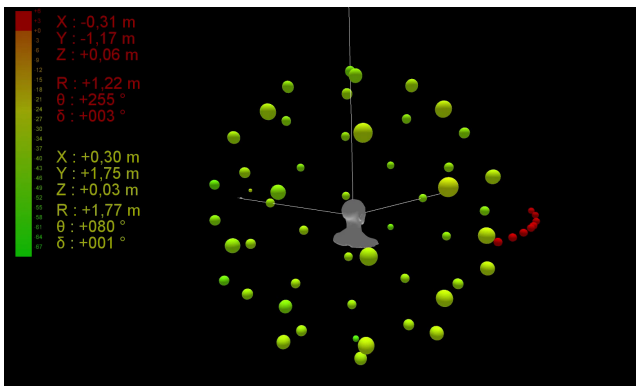


Figure 4: Spherical VU-Meter using PROCESSING for a 50-node Lebedev grid. Each loudspeaker is represented by a ball whose size and color are proportional to the RMS level in dBFS. The virtual source are represented with small balls with fading trajectories. The instantaneous coordinate is displayed in Cartesian and spherical coordinates.

7.2. Head-Tracking

For binaural synthesis, Ambisonics offers an elegant approach to compensate the head movements of the listener, by rotating the

Ambisonic scene accordingly before decoding [18]. In this context, an affordable head-tracker system is proposed by the Razor AHRS project based on Arduino microchips. [3] The PROCESSING code was adapted for Ambitools in order to genereate OSC messages containing the yaw pitch and roll angles values to be applied for the tools `hoa_azimuth_rotator` or `hoa_rotator`. An example of use is shown in Fig. 5. The blueprints of the 3D-printed box for the head-tracker are also provided.

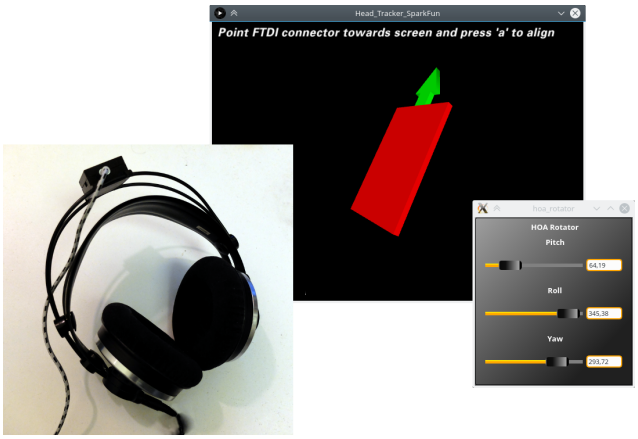


Figure 5: The Razor AHRS head-tracker project integration with Ambitools. The Arduino microchip is fixed on top of the headphones in the 3D-printed box (left of the picture). Orientation angles are transmitted to the PROCESSING interface (middle of the picture). The latter generates OSC messages which drive the sliders of `hoa_rotator` tool (right of the picture).

7.3. FIR Fast Convolution

As mentioned in Secs. 5 and 6.4, several filters involved in Ambisonics are proposed with FIR implementations, such as SMA equalization filters and various HRIRs sets. Currently, the multichannel linear convolutions implementation with FAUST is very costly and compromises the real-time synthesis. Thus, a fast convolution software such as Jconvolver is recommended instead. In Ambitools, the coefficients of the provided filters are given in `.txt` files stored in a `FIR` folder.

7.4. Automatic Online Compilation

For composers and Ambisonics enthusiasts who do not wish to learn the FAUST language, an interactive form is available online to compile the different FAUST tools introduced in the Sec. 6. [4] The corresponding PHP script queries the list of available tools from Ambitools repository. The user is asked via a form to select the tools he wishes and to provide some extra information such as the choice of Ambisonic order M and GUI parameters (VU-meters, Mute check boxes) needed at compilation time. He can also choose the output format for the plug-in. Then, the `.dsp` files are automatically edited according to the user requests. Finally, the edited codes are sent with required dependencies to the GRAME servers for compilation. [5] The output plug-ins are sent back to the user for download.

[3] `https://github.com/Razor-AHRS/razor-9dof-ahrs`

[4] `http://www.sekisushai.net/ambitools/index.php/download/`

8. CONCLUSIONS AND FUTURE WORKS

This paper introduced the toolbox "Ambitools" in its current version (v-1.0). The code design was shown to be scalable and flexible, allowing the user to adapt it to its need. Details were given for the scalable implementation of spherical harmonics. Then, some insights were given for the Ambisonic radial filters implementation, and the tools available in the current version were introduced. Thanks to FAUST, the DSP tools are compiled in various formats. This enables the possibility for sound engineers or composers to have an efficient workflow with Ambisonics by integrating the tools as plug-ins in their audio software. Future works includes the implementation of modern Ambisonics effect, such as Ambisonic spatial blur [28] or Ambisonic warping [17].

9. REFERENCES

[1] Jérôme Daniel, *Représentation de champs acoustiques, application à la transmission et à la reproduction de scènes sonores complexes dans un contexte multimédia*, Ph.D. thesis, Université Paris 6, Paris, 2000.

[2] Aaron J Heller, Eric M Benjamin, and Richard Lee, "A toolkit for the design of ambisonic decoders," in *Linux Audio Conference*, Stanford, 2012, pp. 1–12, LAC.

[3] Matthias Geier and Sascha Spors, "Spatial Audio with the SoundScape Renderer," in *27th Tonmeistertagung–VDT International Convention*, 2012.

[4] Matthias Kronlachner, "Plug-in Suite for mastering the production and playback in surround sound and ambisonics," in *Linux Audio Conference*, 2013.

[5] Julien Colafrancesco, Pierre Guillot, Eliott Paris, Anne Sèdes, and Alain Bonardi, "La bibliothèque HOA, bilan et perspectives," in *Journées d'Informatique Musicale*, 2013, pp. 189–197.

[6] Thibaut Carpentier, Markus Noisternig, and Olivier Warusfel, "Twenty years of Ircam Spat: looking back, looking forward," in *41st International Computer Music Conference (ICMC)*, 2015, pp. 270–277.

[7] Pierre Lecomte and Philippe-Aubert Gauthier, "Real-Time 3D Ambisonics using Faust, Processing, Pure Data, And OSC," in *15th International Conference on Digital Audio Effects (DAFx-15)*, Trondheim, 2015, DAFx.

[8] Thibaut Carpentier, "A versatile workstation for the diffusion, mixing, and post-production of spatial audio," in *Linux Audio Conference*, 2017.

[9] Pierre Lecomte, *Ambisonie d'ordre élevé en trois dimensions : captation, transformations et décodage adaptatifs de champs sonores*, Ph.D. thesis, Université de Sherbrooke - Conservatoire National des Arts et Métiers, Sherbrooke - Paris, 2016.

[10] Pierre Lecomte, Philippe-aubert Gauthier, Christophe Langrenne, Alain Berry, and Alexandre Garcia, "A Fifty-Node Lededev Grid and Its Applications to Ambisonics," *Journal*

[5] `http://faustservice.grame.fr`

of the Audio Engineering Society, vol. 64, no. 11, pp. 868–881, 2016.

[11] Pierre Lecomte, Philippe-Aubert Gauthier, Alain Berry, Alexandre Garcia, and Christophe Langrenne, "Directional filtering of Ambisonic sound scenes," in *Audio Engineering Society Conference: Spatial Reproduction*, Tokyo, 2018, pp. 1–9, AES.

[12] Thibaut Carpentier, "Normalization schemes in Ambisonic: does it matter?," in *Audio Engineering Society Convention 142*. 2017, AES.

[13] Christian Nachbar, Franz Zotter, Etienne Deleflie, and Alois Sontacchi, "Ambix - A suggested ambisonics format," in *Ambisonics Symposium*, Lexington, 2011, p. 11, IEM.

[14] Darren B Ward and Thushara D Abhayapala, "Reproduction of a plane-wave sound field using an array of loudspeakers," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 6, pp. 697–707, 2001.

[15] Sébastien Moreau, Jérôme Daniel, and Stéphanie Bertet, "3d sound field recording with higher order ambisonics-objective measurements and validation of spherical microphone," in *Audio Engineering Society Convention 120*, Paris, 2006, pp. 1–24, AES.

[16] Matthias Kronlachner, "Spatial Transformations for the Alteration of Ambisonic Recordings," 2014.

[17] Franz Zotter and Hannes Pomberger, "Warping of the recording angle in Ambisonics," in *1st International Conference on Spatial Audio*, Detmold, 2011, pp. 1–3, ICSA.

[18] Markus Noisternig, Alois Sontacchi, Thomas Musil, and Robert Holdrich, "A 3d ambisonic based binaural sound reproduction system," in *Audio Engineering Society Conference: 24th International Conference: Multichannel Audio, The New Reality*. 2003, pp. 1–5, AES.

[19] Aaron J Heller and Eric M Benjamin, "The Ambisonic Decoder Toolbox: Extensions for Partial-Coverage Loudspeaker Arrays," in *Linux Audio Conference*, Karlsruhe, 2014.

[20] George B Arfken and Hans J Weber, *Mathematical Methods for Physicists*, Elsevier, 6th edition, 2005.

[21] Jérôme 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*, Helsingør, 2003, pp. 1–15, AES.

[22] Hannes Pomberger, "Angular and radial directivity control for spherical loudspeaker arrays," M.S. thesis, University of Music and Dramatic Arts, Graz, 2008.

[23] Benjamin Bernschütz, "A spherical far field hrir/hrtf compilation of the neumann ku 100," in *40th Italian Annual Conference on Acoustics (AIA) and the 39th German Annual Conference on Acoustics (DAGA) Conference on Acoustics*, Merano, 2013, pp. 1–4, DAGA.

[24] Gary Elko, Robert A Kubli, and Jens Meyer, "Audio system based on at least second-order eigenbeams," 2009.

[25] Michael Chapman, "Symmetries of Spherical Harmonics: applications to ambisonics," in *Ambisonics Symposium*, Graz, 2009, pp. 1–14, IEM.

[26] Jens Meyer and Gary Elko, "A highly scalable spherical microphone array based on an orthonormal decomposition of the soundfield," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*. 2002, vol. 2, pp. 1781–1784, IEEE.

[27] Joseph Ivanic and Klaus Ruedenberg, "Rotation matrices for real spherical harmonics. Direct determination by recursion," *The Journal of Physical Chemistry*, vol. 100, no. 15, pp. 6342–6347, 1996.

[28] Thibaut Carpentier, "Ambisonic spatial blur," in *Audio Engineering Society Convention 142*, Berlin, 2017, pp. 1–7, AES.