FDN reverberation in Ambisonics

Seminar project for Algorithmen in Akustik und Computermusik 2

Matthias Blochberger

Supervisors: Dr. Franz Zotter, Dr. Matthias Frank

Graz, July 5, 2017

institut für elektronische musik und akustik

Abstract

Artificial reverberation in Ambisonics often utilizes convolution to achieve a convincing effect. This requires efficient implementation and computational effort when Ambisonics is used in higher orders and impulse responses are long. The goal of this project was to explore the implementation and customization of a *feedback delay network* to produce convincing and natural reverberation while using a fraction of used computational load, compared to convolution reverberation.

Contents

1 Introduction

The motivation for this project was to explore the options of using a Feedback Delay Network (FDN) to implement well-sounding and efficient reverberation for Ambisonics. Implementations using convolution are common, but with long impulse responses and higher-order Ambisonics (e.g. 5th order: 36 channels) its need for processing power can can pose an obstacle. FDNs are an common way to simulate reverberation in audio-signal processing in single-channel and stereo applications. Stautner and Puckette [SP82] first proposed the basic form of the system on which this project is based on. Section 2 provides a general introduction to FDNs. Further Sebastian J. Schlecht and Emanuel A. P. Habets [SH15] proposed a way to make the FDN time-variable. This general proposition has been adapted to allow a pure real-valued implementation in the programming language Pure data.

The main focus of the project was applying an FDN to Ambisonics and extending the functionality and parametrization of the reverberation effect far enough to make it useful as a plug in. During literature study for this project, there appeared to be no larger or substantial work applicable to systems with more channels, especially targeted to massive multi-channel surround/3D audio. An FDN implemented by Daniel Rudrich for a performance by Al Di Meola at the *Institute of Electronic Music and Acoustics* of the *University of Music and Performing Arts Graz* [RZF16] was taken as an inspiration. The multi-channel structure is outlined in section 2.4. The practical implementation of the reverberation effect is explained in section 3.

2 Feedback Delay Networks

In 1982 John Stautner and Miller Puckette proposed a form of artificial reverberation which would later be commonly named *feedback delay network*.

$$y[n] = x[n-m] + a \cdot y[n-m] \tag{1}$$

represents a recursive comb filter. Replacing the m-sample delay by an array of delay lines of different lengths and the feedback gain a by a feedback matrix $\mathbf{A}$ gives us the general structure of the Stautner-Puckette FDN.

In time domain the entire FDN is given by the relations

$$\begin{aligned} y[n] &= \mathbf{c}^T\mathbf{s}[n] + dx[n], \\ \mathbf{s}[n+\mathbf{m}] &= \mathbf{A}\mathbf{s}[n] + \mathbf{b}x[n] \end{aligned} \tag{2}$$

where $\mathbf{m}$ a vector of different delay lengths used by the network, m_1 to m_N. Figure 1 show the structure of such a network.

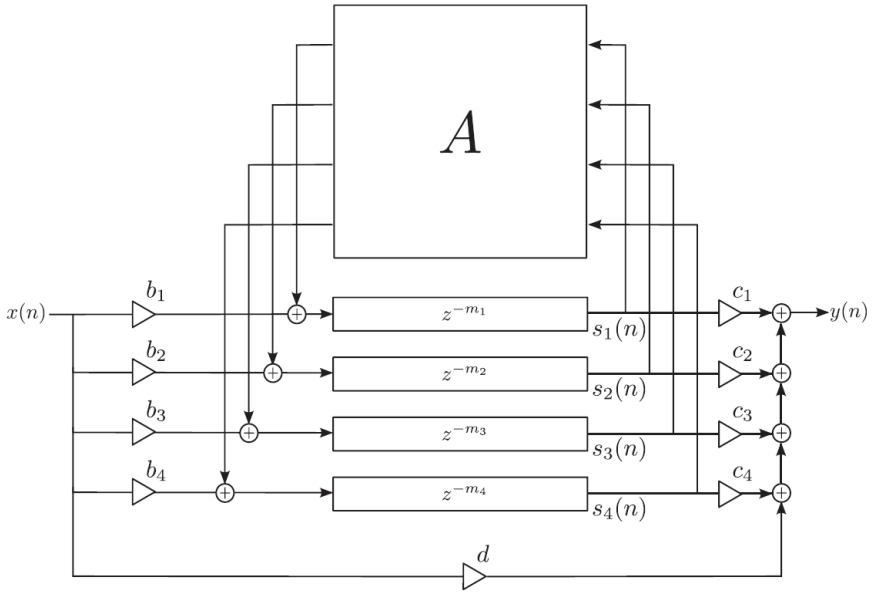

Figure 1 – *Basic feedback delay network*

2.1 Stability of FDNs - Unitary feedback matrix

The general structure introduced in section 2 is the rudimentary basis of an FDN. As figure 2 shows, the insertion of filters provides attenuation which achieves frequency-dependent reverberation times. The energy at the inputs and the outputs of the matrix has to be preserved, otherwise the effect of the attenuation is undetermined. The energy preservation ensures a stable system and is described by a unitary matrix $\mathbf{A} \in \mathbb{C}^{N\times N}$ [SH15] or an orthogonal real-valued matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, so that $\mathbf{A}^H\mathbf{A} = \mathbf{I}$.

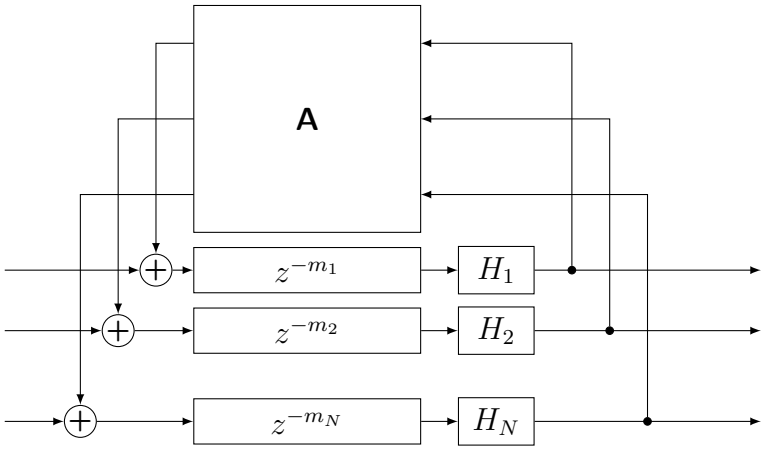

Figure 2 – *Feedback delay network with filters in the loop*

2.2 Time-variant FDN

FDN reverberation can yield an undesirable metallic tin-can-like sound. Sebastian J. Schlecht proposed time-varying feedback matrices as a solution and reached satisfying improvements [SH15]. The time variation with a particular parametrization is achieved by updating the feedback matrix with an *update matrix*

$$\begin{aligned} \mathbf{A}[n] &= \mathbf{A}[n-1]\mathbf{R}, \\ \mathbf{A}[n] &= \mathbf{A}[0]\mathbf{R}[n]. \end{aligned} \tag{3}$$

To ensure stability, this matrix must to be unitary/orthogonal. The eigenvalue decomposition of $\mathbf{R}$ provides control of the time variation,

$$\mathbf{A}[n] = \mathbf{A}[0]\mathbf{U}\Lambda[n]\mathbf{U}^H. \tag{4}$$

$\Lambda[n]$ is the diagonal matrix of the eigenvalues $diag([\lambda_1[n], ..., \lambda_N[n]])$. $\mathbf{U}$ is an arbitrary unitary matrix. Modulating these eigenvalues $\lambda_1[n], ..., \lambda_N[n]$ achieves the time variation. The eigenvalues have an absolute value of 1, due to the requirement of a unitary/orthogonal feedback matrix. This leaves the phases of the eigenvalues as a degree of freedom and hereby as the only way to modulate.
This method can be simplified by leaving out the initial feedback matrix $\mathbf{A}[0]$ and recalculating the new feedback matrix from variable eigenvalues:

$$\mathbf{A}[n] = \mathbf{U}\Lambda[n]\mathbf{U}^H \tag{5}$$

Such a time variation had to be adapted to allow implementation in the target graphical programming language Pure data, because it only allows processing with real values.

The eigenvalue decomposition can be written as summation of the products of the complex eigenvectors $\mathbf{v}_i$ and eigenvalues λ_i.

$$\mathbf{A}[n] = \sum_{i=1}^{N} \mathbf{v}_i \lambda_i[n] \mathbf{v}_i^H \tag{6}$$

Considering the required limitation to real numbers and the fact that audio signals are real signals, the goal is to represent the real-valued feedback matrix resulting from complex conjugate eigenvalue pairs and eigenvector pairs by something purely real-valued. We may formulate this for a single complex-valued pair as

$$\hat{\mathbf{A}} = \mathbf{v}_1 \lambda \mathbf{v}_1^H + \mathbf{v}_2 \lambda^* \mathbf{v}_2^H \tag{7}$$

with the real and imaginary parts

$$\begin{aligned} \mathbf{v}_1 &= \mathbf{a} + i\mathbf{b}, \\ \mathbf{v}_2 &= \mathbf{a} - i\mathbf{b}, \end{aligned} \tag{8}$$

with $\mathbf{a}$ and $\mathbf{b}$ being real-valued linear independent column vectors with a norm $\|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 = 1$. Rewriting Eq. (7) yields

$$\hat{\mathbf{A}} = (\mathbf{a} + i\mathbf{b})\lambda(\mathbf{a} + i\mathbf{b})^H + (\mathbf{a} - i\mathbf{b})\lambda^*(\mathbf{a} - i\mathbf{b})^H, \tag{9}$$

$$\hat{\mathbf{A}} = (\mathbf{a} + i\mathbf{b})(\mathbf{a}^T - i\mathbf{b}^T)\lambda + (\mathbf{a} - i\mathbf{b})(\mathbf{a}^T + i\mathbf{b}^T)\lambda^*, \tag{10}$$

and by expanding the products, we get

$$\hat{\mathbf{A}} = (\mathbf{a}\mathbf{a}^T - i\mathbf{a}\mathbf{b}^T + i\mathbf{b}\mathbf{a}^T + \mathbf{b}\mathbf{b}^T)\lambda + (\mathbf{a}\mathbf{a}^T + i\mathbf{a}\mathbf{b}^T - i\mathbf{b}\mathbf{a}^T + \mathbf{b}\mathbf{b}^T)\lambda^* \tag{11}$$

Inserting the real and imaginary components of the eigenvalue pair $\lambda = 1e^{i\omega} = \cos(\omega) - i\sin(\omega)$ we get

$$\begin{aligned}\hat{\mathbf{A}} = (\mathbf{a}\mathbf{a}^T - i\mathbf{a}\mathbf{b}^T + i\mathbf{b}\mathbf{a}^T + \mathbf{b}\mathbf{b}^T)(\cos(\omega) - i\sin(\omega)) + \\ (\mathbf{a}\mathbf{a}^T + i\mathbf{a}\mathbf{b}^T - i\mathbf{b}\mathbf{a}^T + \mathbf{b}\mathbf{b}^T)(\cos(\omega) + i\sin(\omega)).\end{aligned} \tag{12}$$

After expansion, many terms cancel in the above expression and we obtain

$$\hat{\mathbf{A}} = 2(\mathbf{a}\mathbf{a}^T + \mathbf{b}\mathbf{b}^T)\cos(\omega) + 2(\mathbf{b}\mathbf{a}^T - \mathbf{a}\mathbf{b}^T)\sin(\omega). \tag{13}$$

We define rank-one vector products as

$$\begin{aligned}\mathbf{a}\mathbf{a}^T &=: \mathbf{B}, \\ \mathbf{b}\mathbf{b}^T &=: \mathbf{C}, \\ \mathbf{a}\mathbf{b}^T &=: \mathbf{D}, \\ \mathbf{b}\mathbf{a}^T &= \mathbf{D}^T.\end{aligned} \tag{14}$$

This leaves us with a symmetric part $2(\mathbf{B}+\mathbf{C})\cos(\omega)$ and an antisymmetric part $2(\mathbf{D}^T - \mathbf{D})\sin(\omega)$. The symmetric part represents the real component of the eigenvalue pair, the antisymmetric represents the imaginary component. Hereby we get one pair of eigenvalues on the unit circle with the phase ω from the positive real axis on the complex plane (Figure 3):

$$\hat{\mathbf{A}} = 2(\mathbf{B} + \mathbf{C})\cos(\omega) + 2(\mathbf{D}^T - \mathbf{D})\sin(\omega) \tag{15}$$

Putting this back together to the form of Eq. (6) we get an $N \times N$ feedback matrix for $\frac{N}{2}$ eigenvalue pairs which are defined by the phases ω_i and the real-valued rank-one matrices $\mathbf{B}_i$, $\mathbf{C}_i$ and $\mathbf{D}_i$ corresponding to every complex conjugate eigenvector pair

$$\mathbf{A}[n] = \sum_{i=1}^{N} 2(\mathbf{B}_i + \mathbf{C}_i)\cos(\omega_i[n]) + 2(\mathbf{D}_i^T - \mathbf{D}_i)\sin(\omega_i[n]). \tag{16}$$

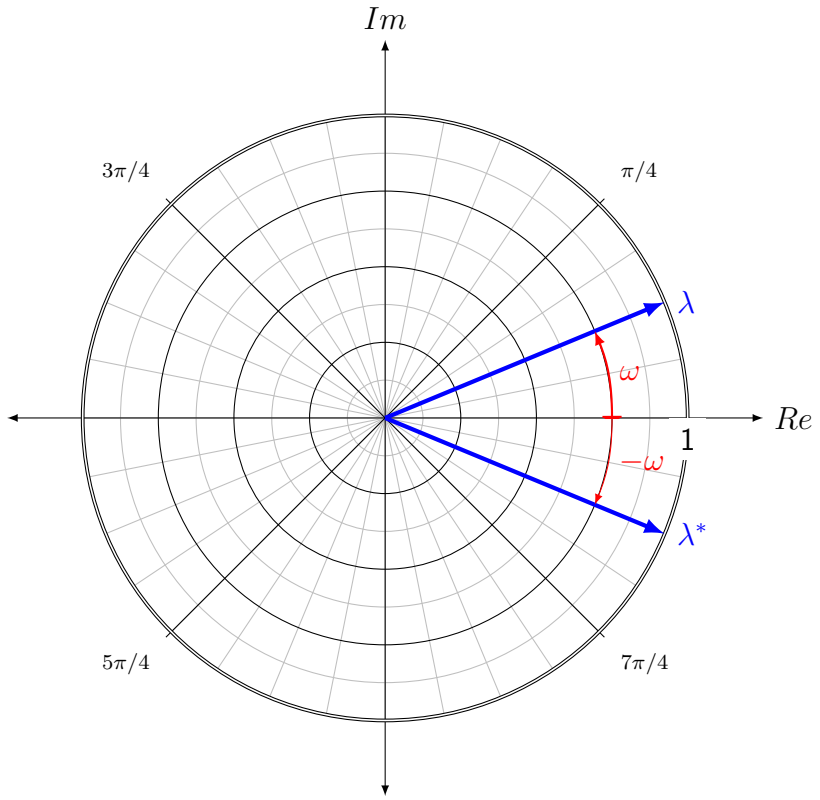

Figure 3 – *A pair of complex conjugate eigenvalues on the unit circle*

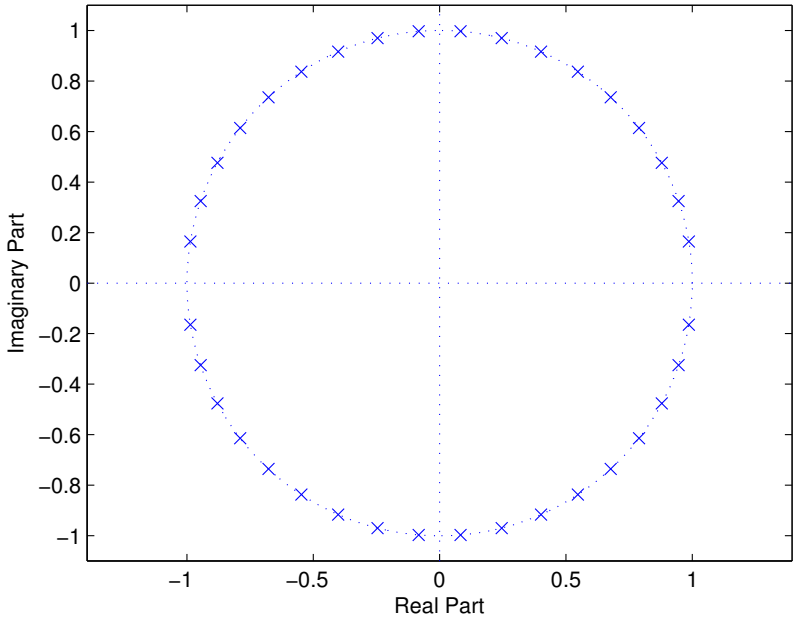

Figure 4 – *Polar plot of the eigenvalues of a unitary* 36×36 *matrix*

The $N \times N$ matrices $\mathbf{B}_i$, $\mathbf{C}_i$ and $\mathbf{D}_i$ are real-valued and constant for each eigenvalue λ_i. They are by definition (Eq. (14)) products of the arbitrary vectors $\mathbf{a}_i$ and $\mathbf{b}_i$ which can for example be randomly generated but must be orthogonal. Specifically for the practical implementation it is helpful to store the results for the expressions $2(\mathbf{B}_i + \mathbf{C}_i)$ and $2(\mathbf{D}_i^T - \mathbf{D}_i)$ in configuration files. Time variation of the feedback matrix is implemented by variation of the phases of the eigenvalue pairs with low-frequency oscillators for each eigenvalue pair.

2.3 Pre-calculations in Matlab

To take load off the computation and to simplify the processing in Pure data, some parts can be pre-calculated and stored in files. Those items were generated using Matlab to implement what is explained in section 2.2.

```
n=36;
eig_nr=18;
A=zeros(36,36);
M=orth(rand(n,n));
BC_n=zeros(36,36,eig_nr);
D_n=zeros(36,36,eig_nr);
k1=1;
```

For $\mathbf{A}$ with dimensions $N \times N$ with $N = 36$ a orthogonal matrix $\mathbf{M}$ with the same dimensions is randomly generated. The feedback matrix $\mathbf{A}$ and the variables to store the symmetric and antisymmetric parts of the feedback matrices for each of the 18 eigenvalue pairs are initialized.

```
for k=2:2:eig_nr*2
  a=M(:,k-1);
  b=M(:,k);
  a=a/sqrt(2);
  b=b/sqrt(2);
  B=a*a.';
  C=b*b.';
  D=a*b.';
  BC_n(1:n,1:n,k1)=B+C;
  DtD_n(1:n,1:n,k1)=D.'-D;
  w=pi*k/(eig_nr*2+2);
  temp=2*BC_n(:,:,k1)*cos(w)+2*DtD_n(:,:,k1)*sin(w);
  A=A+temp;
  k1=k1+1;
end
```

Taking two rows or columns from the orthogonal matrix $\mathbf{M}$ gives two linear independent vectors as deemed necessary in equation (8). The vectors are normalized by the square root of 2 so $\|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 = 1$. $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are calculated as defined at equation (14) as $\mathbf{B} = \mathbf{a}\mathbf{a}^T$, $\mathbf{C} = \mathbf{b}\mathbf{b}^T$ and $\mathbf{D} = \mathbf{a}\mathbf{b}^T$. The phase ω ($\mathbf{w}$ in script) was chosen in

dependency of the number of eigenvalue pairs to ensure a uniform spread of eigenvalues on the unit circle. The summation of the partial results in **temp** yields the feedback matrix **A** which was only used for testing purposes and visualization (figure 3) in the Matlab code. The matrices **BC_n** and **DtD_n** are stored in files for the use in the real-time Pd implementation (section 3).

2.4 Using the FDN in an Ambisonics system

An FDN implemented by Daniel Rudrich for 3D audio live effects in a performance of Al Di Meola used the output of the delay lines as Ambisonic channels. This concept proved to be promising in early tests and got adopted.

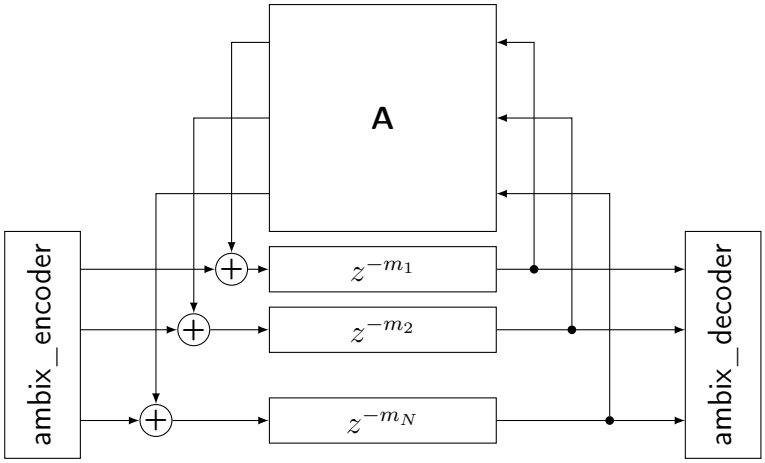

Figure 5 – *Using the ambisonics channel as inputs and outputs of the delay-lines*

This requires the size of the feedback matrix to assume the size $N = (\text{Ambisonics-order}+1)^2$.

3 Implementation

Pure data is a real-time graphical programming environment for audio and graphical processing. [Puc16]
It follows an object-oriented approach. Pure data was chosen as a fast and flexible way of implementation. The graphical interface is relatively intuitive to use in general and gives the possibility to get working audio effects in a short amount of time. To mention is the fact that the development of this effect showed some drawbacks, which are mainly a weak support for coding bigger multi-channel applications and the very limited possibilities to optimize the algorithms.
Input and output is an ambisonics-encoded audio signal. The Amisonic channels have to be routed from a DAW to the reverberator. While developing, REAPER was used which provides an ASIO driver called ReaRoute [rea16] to accommodate in and output signals to the real-time implementation in Pure Data.

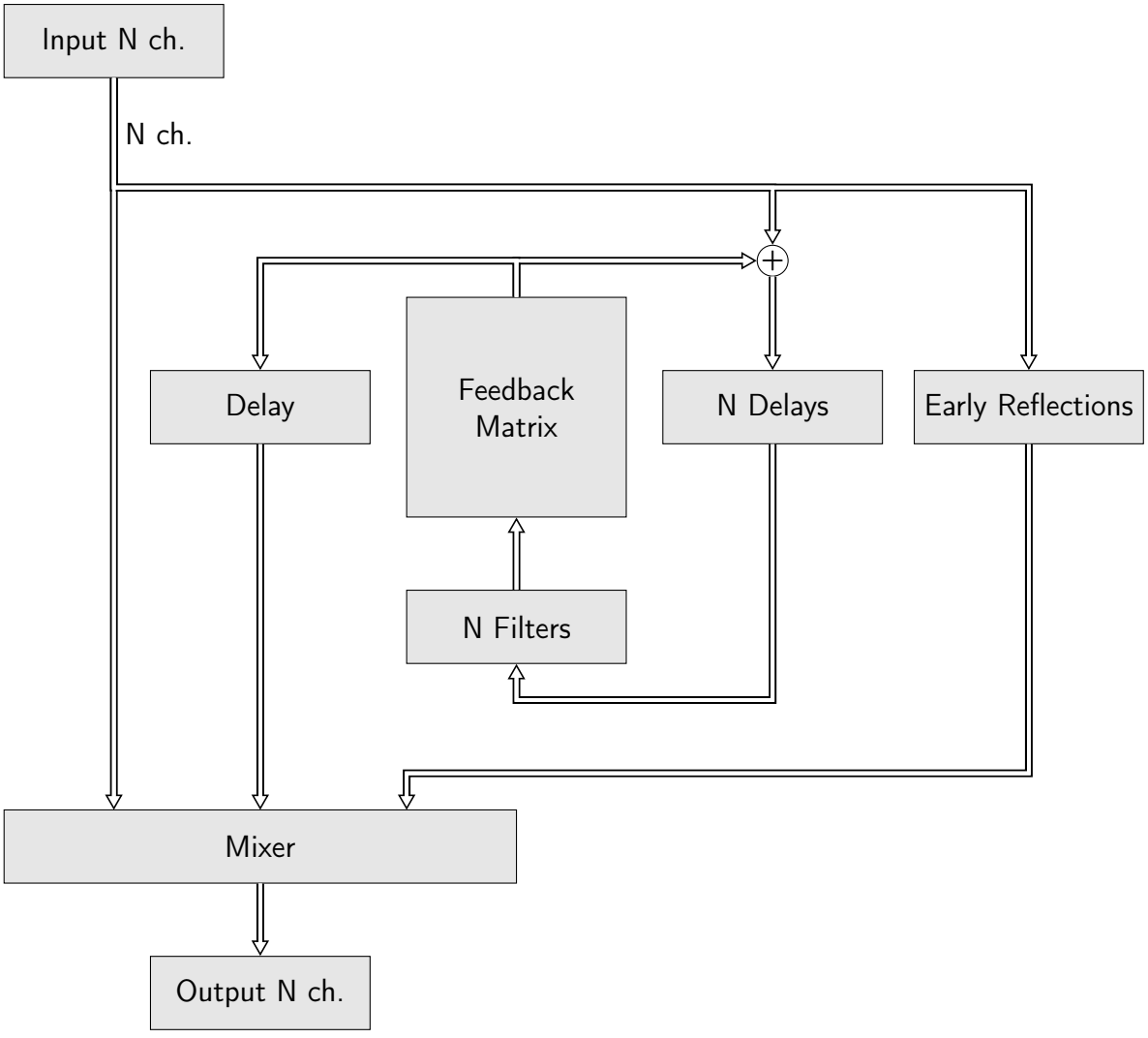

Figure 6 – Block diagram of the implementation

Figure 6 gives an overview of the general process trough different subsystems of the implementation. Parameters of the algorithm are controlled on a graphical user interface. The project is composed of the main-patch and several sub patches:

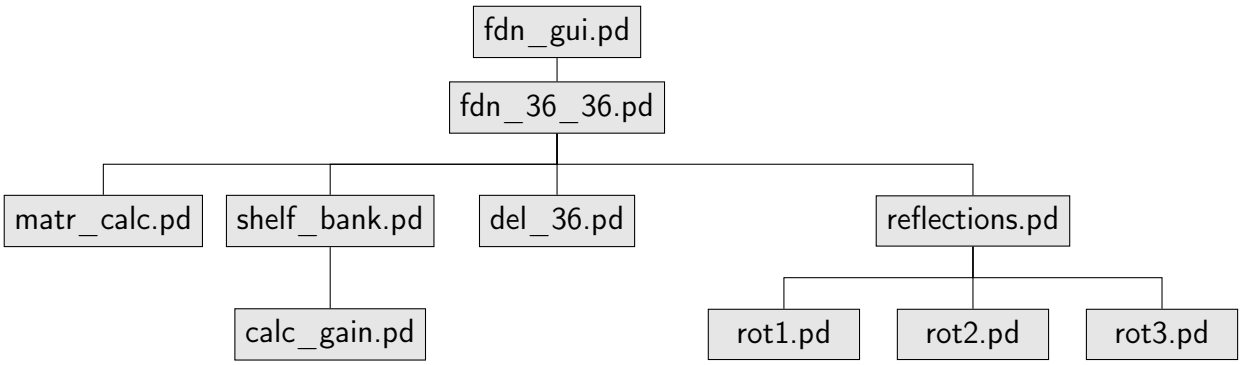

Figure 7 – Pure data patch-hierarchy

3.1 fdn_gui.pd

The patch [*fdn_gui*] contains the elements for users to control parameters of the reverberation. Values are sent to other patches by [*send*] objects.

- Dry-Wet slider (1)
- Button for activation of modulation for feedback matrix (2)
- Decay-time sliders for low, mid and high-band (3)
- Cut-off frequency sliders for bands (4)
- Slider for level of the FDN reverberator (5)
- Pre-delay for FDN reverb in ms (6)
- Rotation for early reflections (7)
- Delays of early reflections in ms (8)
- Levels of the early reflections (9)

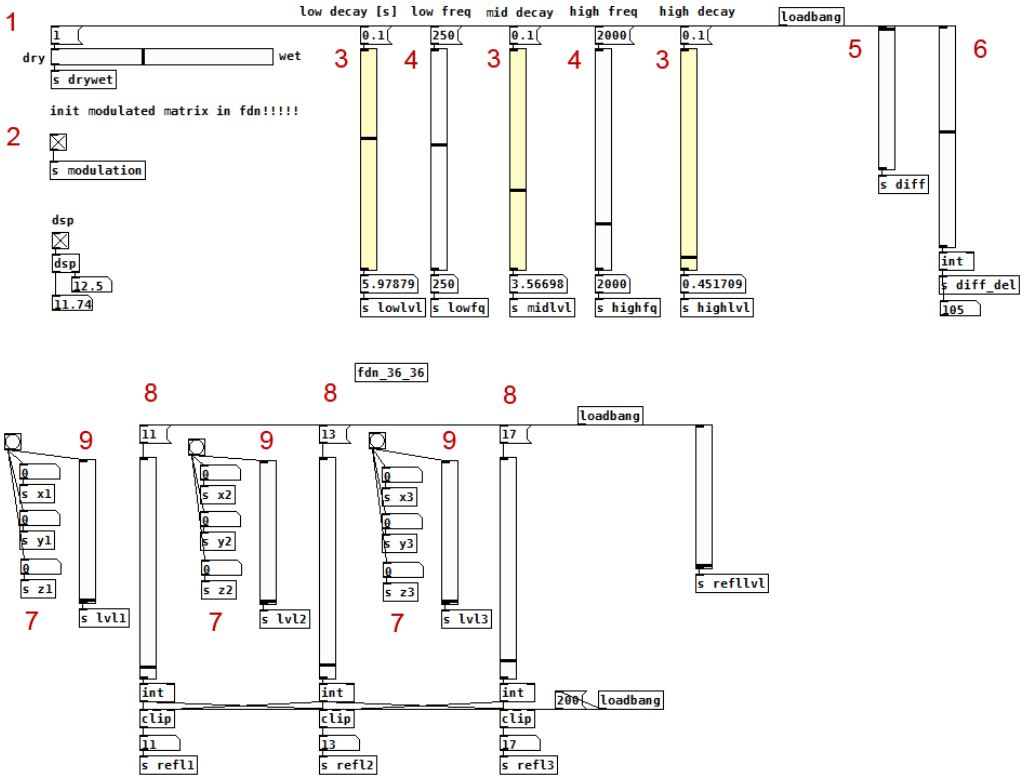

Figure 8 – *GUI*

3.2 fdn_36_36.pd

The main functionality is implemented in [*fdn_36_36*]. The used matrix operations are taken from the *iemmatrix* library for Pd. The structure of this patch is shown in Figure 6. Thee feedback loop with the delay-lines is realized by using the [*delwrite~*] and [*delread~*] objects. The length of the delays in milliseconds are the first 36 prime numbers starting at 11 to prevent resonances.

3.3 matr_calc.pd

The feedback matrix is calculated in [*matr_calc*] as explained in section 2.2 by

$$\mathbf{A}[n] = \sum_{i=1}^{N} 2(\mathbf{B}_i + \mathbf{C}_i)\cos(\omega_i[n]) + 2(\mathbf{D}_i^T - \mathbf{D}_i)\sin(\omega_i[n]).$$

$\mathbf{B}, \mathbf{C}, \mathbf{D}$ are generated from random orthogonal vectors as explained in section 2.2 and stored as $2(\mathbf{B}_i + \mathbf{C}_i)$ and $2(\mathbf{D}_i^T - \mathbf{D}_i)$ in **.mtx* files by the aforementioned Matlab code.

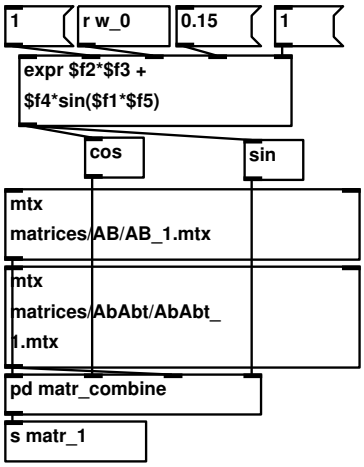

Figure 9 – *Calculating the feedback matrix for one eigenvalue*

For every eigenvalue, a component of the feedback matrix is calculated. The sum of these components yields the feedback matrix **A**. This happens every 30 milliseconds. (File names in figure 9: $2(\mathbf{B}_i + \mathbf{C}_i)$ stored in *AB_i.mtx*, $2(\mathbf{D}_i^T - \mathbf{D}_i)$ stored in *AbAbt_i.mtx*)

3.4 shelf_bank.pd

A 3-band filter is part of each channel in the feedback loop. The bank of filters for each channel in implemented in [*shelf_bank*] and uses the [*hml_shelf~*] objects from the *iemlib* library.

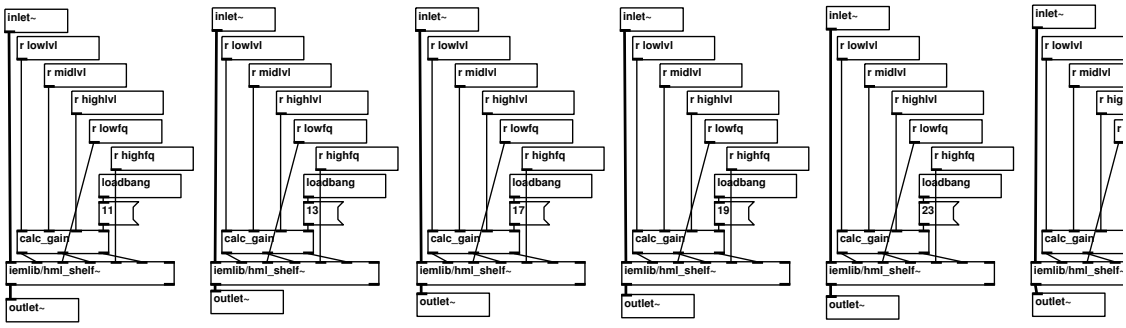

Figure 10 – *Each of the N channels is attenuated by a 3-band shelving-filter*

3.5 calc_gain.pd

The gain values for the bands of the shelving filters in *shelf_bank.pd* are derived from a desired decay curve called y in the following explanation. For a delay line with delay

t_m and filter with gain g_m the decay curve y at time t is

$$y = g_m^{\frac{t}{t_m}}.$$

The desired decay curve can also be described with

$$y = 10^{-\frac{60}{20} \cdot \frac{t}{t_{60}}}$$

where t_{60} is the desired decay time (RT60). Solving for g_m gives us the gain for the filter

$$\begin{aligned} g_m^{\frac{t}{t_m}} &= 10^{-3\frac{t}{t_{60}}}, \\ log_{10}(g_m) \cdot \frac{t}{t_m} &= -3\frac{t}{t_{60}}, \\ log_{10}(g_m) &= -3\frac{t_m}{t_{60}}, \\ g_m &= 10^{-3\frac{t_m}{t_{60}}}, \\ g_{m,dB} &= -30\frac{t_m}{t_{60}}. \end{aligned}$$

Implementation:

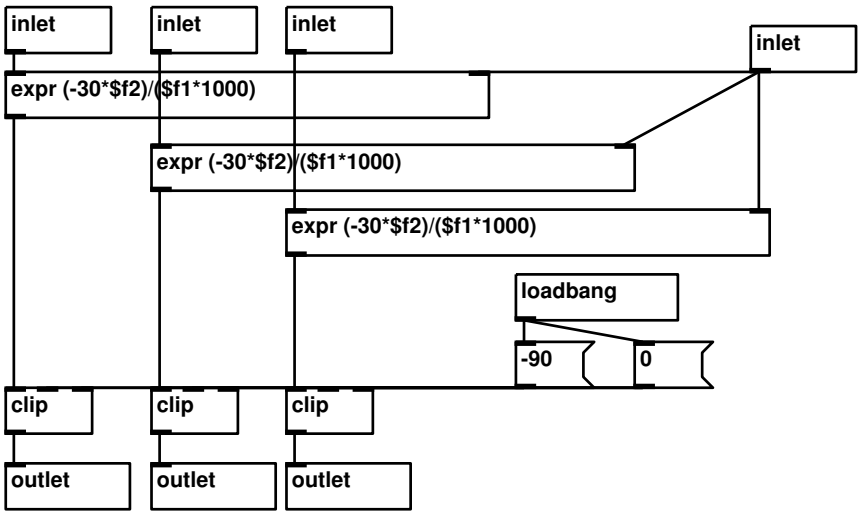

Figure 11 – *The Pd patch [calc_gain] calculates the gain in dB for each band per delay line.*

Figure 12 to Figure 15 show the Energy Decay Reliefs for selected RT60 values ranging from 1 to 10 seconds.

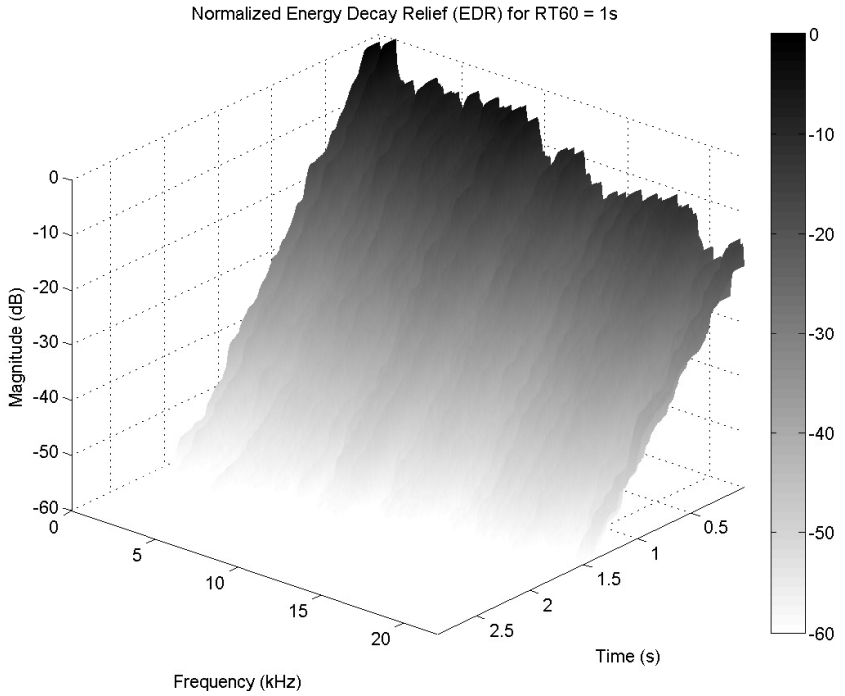

Figure 12 – *EDR for RT60 = 1s*

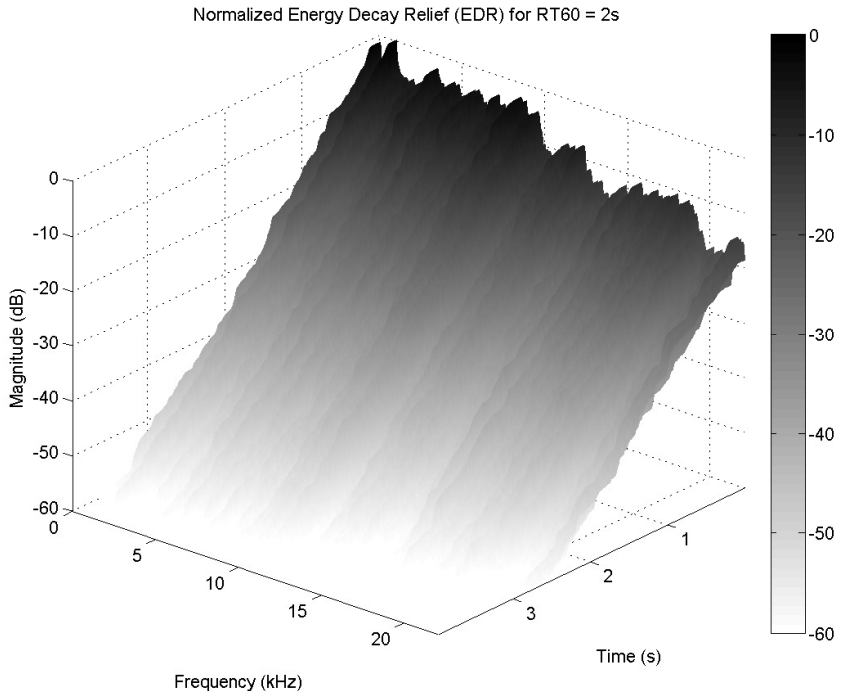

Figure 13 – *EDR for RT60 = 2s*

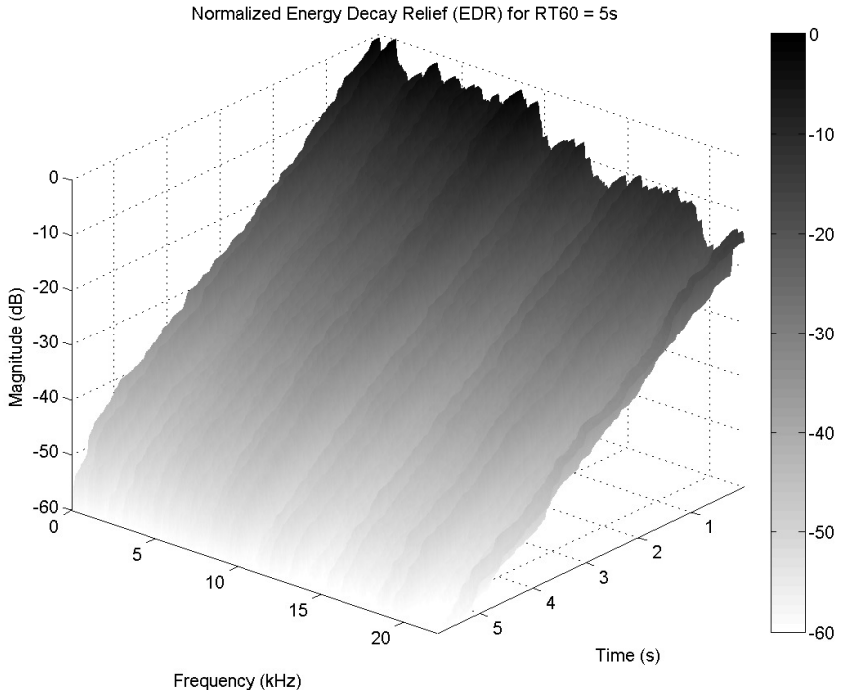

Figure 14 – *EDR for RT60 = 5s*

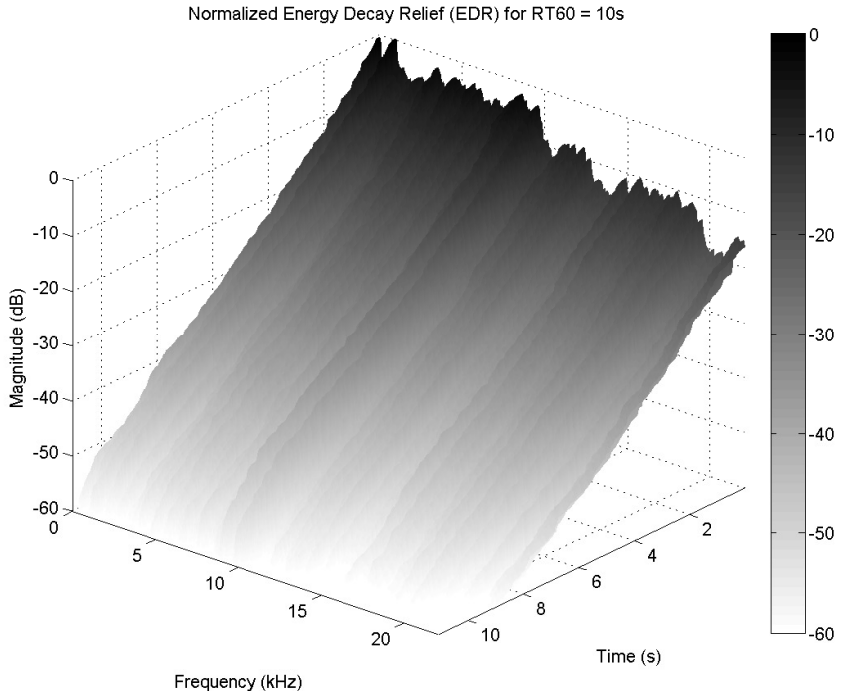

Figure 15 – *EDR for RT60 = 10s*

3.6 del_36.pd

The [*del_36*] patch is a uniform set of 36 delays for each channel. It is used as a delay of the FDN reverb. Each channel uses the [*delwrite~*] and [*delread~*] objects.

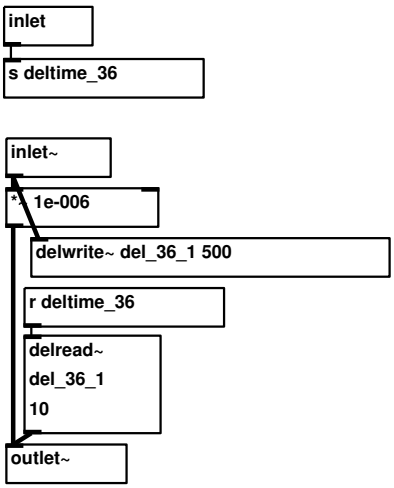

Figure 16 – *Delay for one channel*

3.7 reflections.pd

For an impulse response resembling the characteristics of a real room, the option of adding early reflections is given. The relative rotation, level and delay in milliseconds can be changed via the GUI. This implementation has 3 reflections.

3.8 rot1/2/3.pd

The rotation and delay of the ambisonics signal of the reflections is processed in patches *rot1.pd*, *rot2.pd* and *rot3.pd*. Rotation of the signal around the x,y and z axis is done by calculating the rotation-matrix for the ambisonic signal from desired rotation angles (figure 17). The delay is achieved as explained in section 3.6.

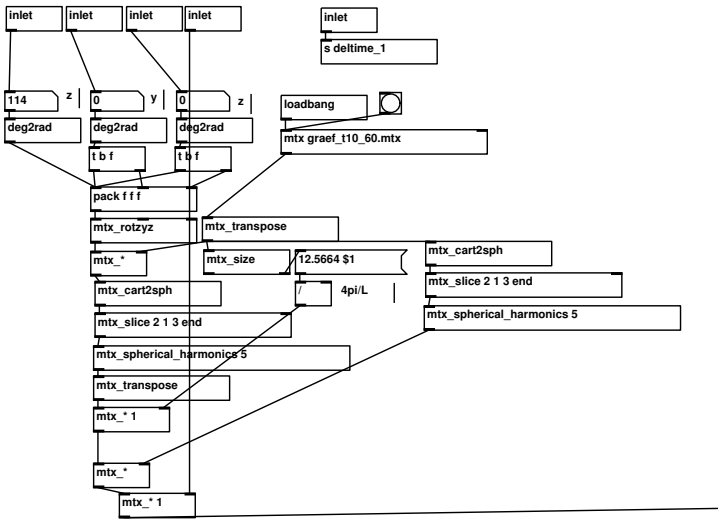

Figure 17 – *Calculating the rotation matrix.*

4 Conclusion and outlook

The general consensus after a listening session at IEM was that the quality of the reverberation achieved here is higher than expected. The conduction of proper listening tests was out of scope for this work, so it remains at least to report the impressions by expert listeners when presenting the algorithm. As expected, the number of delay lines contributes the most in terms of sound quality. The usage of Ambisonic channels as inputs of the FDN gives a very "organic" reverberation. Adding the modulation of the feedback matrix minimizes the possibility of unwanted resonances, which can be perceived as a "tin-can-like" reverberation. This effect is more prominent in FDNs with a lower number of channels. An optimized implementation in C/C++ as a VST plug in would be promising, considering that high performance is already achieved by the Pd prototype. This would also give the possibilities of a per-sample calculation of the feedback matrix and better parametrization/calculation of properties such as accurately controlled frequency-dependent reverberation time and the addition of more early reflections, maybe generically rendered.

References

[Puc16] M. Puckette, "Pd documentation chapter 2: theory of operation," 2016.

[rea16] "Rearoute," http://wiki.cockos.com/wiki/index.php/ReaRoute, 2016.

[RZF16] D. Rudrich, F. Zotter, and M. Frank, "Efficient spatial ambisonic effects for live audio." Cologne, Germany: 29. TMT 2016, 11 2016.

[SH15] S. J. Schlecht and E. A. P. Habets, "Time-varying feedback matrices in feedback delay networks and their application in artificial reverberation," *The Journal of the Acoustical Society of America*, vol. 138, 2015.

[SP82] J. Stautner and M. Puckette, "Designing multi-channel reverberators," *Computer Music Journal*, vol. 6, no. 1, 1982.