# Real-Time Implementation of Binaural Rendering of High-Order Spherical Microphone Array Signals

Hannes Helmholz, Carl Andersson, and Jens Ahrens
*Audio Technology Group*
*Division of Applied Acoustics*
*Chalmers University of Technology*
*SE-412 96 Gothenburg*
*{hannes.helmholz,carl.andersson,jens.ahrens}@chalmers.se*

## Abstract

We present ReTiSAR[1] (Real-Time Spherical Array Renderer), an open-source implementation of real-time binaural rendering of signals obtained from spherical microphone arrays. The implementation was performed in Python and bases on the previously published SOFiA toolbox as well as on sound_field_analysis-py. We can confirm that Python together with the other tools employed constitutes a viable framework for this kind of heavy-computation application even under real-time constraints. The current version of ReTiSAR is able to render signals of up to 8th order on a standard laptop computer.

## Introduction

A number of implementations are available for binaural rendering of data obtained from spherical microphone arrays – or equivalently – Ambisonics signals. Amongst the available ones are a set of plug-ins from the IEM Plug-in Suite [1], the SPARTA & COMPASS Audio Plug-in Suite [2], the SOFiA toolbox [3], and sound_field_analysis-py [4] (SFApy). [1,2] consist of plug-ins for digital audio workstations (DAWs). [3,4] are implementations based on measured impulse responses (IRs) of static scenarios whose output can be rendered with head tracking using additional software such as the SoundScape Renderer[2] (SSR) [5]. A large set of such measurement data is available from [6].

The implementations that are based on measured IRs cannot be employed to investigate dynamic scenarios such as moving sources or the effect of microphone self-noise on the output. We therefore present ReTiSAR as an extension of [3,4] for the processing of streamed signals. We found in [4] that porting the MATLAB implementation of [3] to Python resulted in a speed-up of the execution by a factor of 10. For this reason, as well as the availability of an established rich open source ecosystem consolidated the choice to implement the pipeline in Python.

[1] Available at https://github.com/AppliedAcousticsChalmers/ReTiSAR
[2] For detailed instructions see https://nbviewer.jupyter.org/github/AppliedAcousticsChalmers/sound_field_analysis-py/blob/master/examples/Exp4_BinauralRendering.ipynb

## Rendering Concept

ReTiSAR does not employ virtual loudspeakers in the binaural rendering of the data but decodes the signals directly onto the head-related transfer functions (HRTFs) in the spherical harmonics (SH) domain. The following approach was presented, for example, in [7] and in greater detail in [8,9].

Any arbitrary interior sound field can be modeled as a continuum of plane waves that propagate in all possible directions. The plane wave coefficients, sometimes also referred to as *signature function*, represent the sound field. Assuming that an HRTF is the acoustic response to an incident plane wave, the signals arising at the ear of the listener due to the sound field under consideration can be computed by weighting the HRTFs with the plane wave coefficients for the corresponding direction and integrating over all possible directions. Exploiting the orthogonality of the spherical harmonics allows for representing the signals $E_{L,R}(\omega)$ for the left and right ear, respectively, as a sum over the SH coefficients of the involved quantities as follows:

$$E_{L,R}(\omega) = \sum_n \sum_m \underbrace{a_m \, d_n(\omega) \, \breve{H}_n^m(\omega)}_{B_n^m(\omega)} \, \breve{S}_n^{-m}(\omega) \, \mathrm{e}^{-\mathrm{i}m\alpha_{\mathrm{head}}} \quad (1)$$

$d_n(\omega)$ are the radial filters, $\breve{S}_n^{-m}(\omega)$ and $\breve{H}_n^m(\omega)$ are the SH coefficients of the sound field that evolved on the surface of the spherical microphone array and of the HRTFs respectively. $a_m$ is a frequency-independent factor that depends on the definition of SH coefficients, e.g. 1 or $(-1)^m$. We refer the reader to [10] for details. The term $\mathrm{e}^{-\mathrm{i}m\alpha_{\mathrm{head}}}$ accounts for the instantaneous azimuth of the listener's head orientation as measured by a head-tracking system.

## Signal Processing

Eq. (1) consist of a multiplication of three frequency-dependent quantities, which may be interpreted as a sequence of two fast convolutions. The quantity $B_n^m(\omega)$ can be computed offline as neither of the involved quantities change during execution. The real-time execution therefore requires computing $\breve{S}_n^{-m}(\omega)$ from the microphone array signals as well as a set of parallel fast convolutions of $B_n^m(\omega)$ and $\breve{S}_n^{-m}(\omega) \cdot \mathrm{e}^{-\mathrm{i}m\alpha_{\mathrm{head}}}$, whereby appropriate zero-padding to the utilized length $l$ has to be applied in the block-wise execution. We chose to implement this convolution as overlap-save similarly to SSR. We included an option to compute the ear signals for the current as well as the previous head orien-

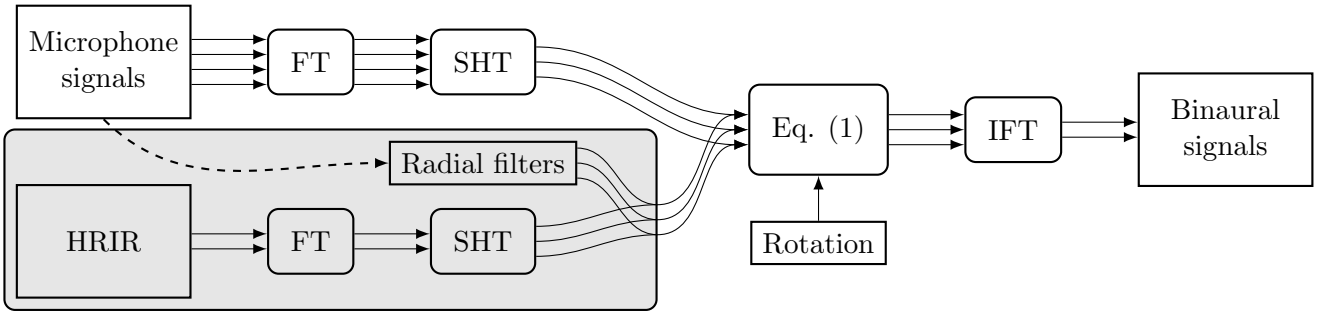


**Fig. 1**: Signal flow with pre-computed quantities during startup marked in gray; FT: Fourier transform, SHT: Spherical harmonics transform, IFT: inverse Fourier transform,

tation and crossfade between the two signals for avoiding discontinuities. We found that only specific scenarios require this. The signal flow in depicted in Fig. 1.

The radial filters are currently computed via frequency-domain sampling of the analytical expressions. Alternative designs were presented, for example, in [11,12].

## Employed Frameworks

We chose to use JACK Audio Connection Kit[3] as framework for routing audio signals from/to the hardware and different components of the software. This allows for directly connecting hardware like the Eigenmike by mh-acoustics whose microphone signals can be accessed directly through JACK where the Eigenmike shows up as a standard audio device. We used jackclient-python[4] to access JACK.

Independent processing steps and utility functions are realized as sub-processes initialized by the Python multiprocessing[5] interface. This allows for running and terminating components separately and prevents potential performance bottlenecks being propagated to other parts of the pipeline.

Numpy[6] is being used as the core numerical computing package. It provides a powerful array object to efficiently operate on n-dimensional matrices during the processing. Designing matrix structures consciously turned out to be very influential on real-time performance. Whenever possible, arrays are organized *C-contiguous* to allow for most efficient access when iterating over anterior dimensions while keeping posterior dimensions as continuous memory blocks.

We found that setting up the Python environment via Anaconda[7] was the option that led to the most efficient execution of the implementation. This is due to Anaconda deploying pre-compiled packages for the individual system architecture. Particularly the ubiquitous numerical matrix operations benefit from introduced parallelization, for example by deploying numpy with Intel MKL (Math Kernel Library) bindings.

Another regularly occurring processing operation is the DFT (Discrete Fourier Transform) and its inverse. In this case, a one-dimensional transformation from real inputs into frequency domain is performed via FFT (Fast Fourier Transform), while the redundant Hermitian-symmetric part is neglected. Extensive analysis showed pyFFTW[8] to yield the best real-time performance when using advanced features of the established FFTW library. ReTiSAR utilizes the concept of *wisdom* to determine and recall pre-optimized and parallelized DFTs for the individual system hardware.

We borrow some functionality from the offline Python implementation [4], for example functionality to manage HRIR and array IR data sets in the MIRO [6] format. Currently supported head trackers include Polhemus Patriot and Razor AHRS by utilizing pyserial[9].

Finally, we implemented an OSC[10] (Open Sound Control) interface for remotely controlling runtime functionality through arbitrary OSC clients such as Pure Data[11] for which we provide an example interface.

So far, the implementation has only been tested on MacOS.

## Execution Modes

ReTiSAR can be used in three different modes:

- Live-Rendering of streamed microphone array signals such as those from the Eigenmike.
- Rendering of recorded microphone array signals that are streamed from a storage medium. This reading process can be prohibitive in terms of the maximum order that can be handled due to bandwidth limitations of the storage medium. ReTiSAR provides multichannel file playback. Alternatively, ecasound[12] can be utilized conveniently for recording and playback as it routes signals directly into JACK and connects to arbitrary parts of the rendering pipeline.
- Rendering based on measured microphone array impulse responses. In this case, one channel is streamed from a storage medium containing arbitrary audio content. This signal is convolved in real-time with array IRs and fed into the renderer. The convolution is partitioned so that no

[3] http://www.jackaudio.org/
[4] https://github.com/spatialaudio/jackclient-python/
[5] https://docs.python.org/3/library/multiprocessing.html
[6] https://www.numpy.org/
[7] https://www.anaconda.com/
[8] https://github.com/pyFFTW/pyFFTW
[9] https://github.com/pyserial/pyserial
[10] http://opensoundcontrol.org/
[11] https://puredata.info/downloads/osc
[12] https://ecasound.seul.org/ecasound/

| $l$ in samples | $\alpha_{\text{head}}$ in degrees | difference RMS in dBFS | |
|---|---|---|---|
| 4096 | 0 | -79.03 | -79.08 |
| | 40 | -79.75 | -75.22 |
| | 80 | -86.56 | -80.39 |
| | 120 | -109.43 | -106.51 |
| | 160 | -80.83 | -76.88 |
| 1024 | 0 | -78.94 | -78.99 |
| | 40 | -79.70 | -75.21 |
| | 80 | -86.32 | -80.33 |
| | 120 | -99.08 | -98.96 |
| | 160 | -80.38 | -76.88 |

**Table 1**: Instrumental evaluation against SFApy for different block lengths $l$ and head azimuth rotations $\alpha_{\text{head}}$; difference RMS for left and right ear respectively

hard limitations exist regarding the IR lengths. This process, termed *pre-rendering*, is interesting for research purposes to emulate streamed microphone array signals through sequentially measured IR sets such as [6].

## Instrumental Validation

We measured the impulse responses of ReTiSAR for different head azimuth orientations and computed the RMS (root-mean-square) of the difference to the output of SFApy [4]. SFApy has been validated thoroughly and was used in a number of psychoacoustic studies such as [8].

Both implementations operate for the exact same scenario, i.e. `R1_VSA_110RS_L` from [6], where 0° head rotation resembles facing the auralized loudspeaker.

To achieve more stable and comparable results regarding different room conditions, i.e. IR lengths, the reference has been normalized for every comparison and truncated after decaying to -100 dBFS respectively. Afterwards, time and level alignment are performed for both ears simultaneously to account for extraneous unrelated influences of the IR "recording" procedure.

Table 1 lists the calculated RMS differences for different combinations of $l$ and $\alpha_{\text{head}}$, for both ears respectively. The results show variations in the range of -75 to -109 dBFS. Fig. 2 visualizes the difference of the compared IRs in time domain for the highest RMS of the analyzed combinations. On a logarithmic scale for the entire signal (left), we see the biggest difference around the peak of the IRs at a value of around -43 dBFS, i.e. less than a hundredth of the signal.

RMS differences and visual inspection of the impulse responses (Fig. 2 on the right) suggest that the signals correspond well. Informal listening confirms that the apparent differences are not audible.

The apparent differences occur due to the fact that the radial filters $d_n(\omega)$ from Eq. (1) are computed for different lengths in the two implementations. The length of the radial filters is limited to the block length in ReTiSAR but not limited in SFApy.

## Current Research

Our current focus is on investigating the effect of microphone self-noise on the ear signals, which is a scenario that

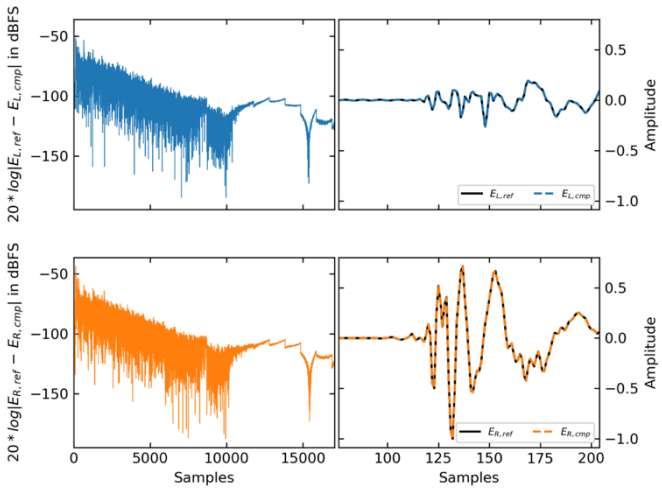


**Fig. 2**: Example of instrumental evaluation against SFApy for $l = 1024$ and $\alpha_{\text{head}} = 40°$; IR difference magnitude (left) and IR segments of 128 samples around peaks of SFApy (dashed lines) and ReTiSAR (solid lines) (right)

cannot be covered by offline implementations. We implemented a module that continuously generates uncorrelated noise that is being added to the output signals of the pre-rendering stage, i.e. the emulated microphone signals. The OSC interface allows for changing the noise level during execution in order for being able to conduct comparative listening experiments. Fig. 3 depicts a screenshot of the JACK connections as visualized by Patchage[13] for this scenario.

## Conclusions

We have found that Python constitutes a viable framework for real time audio processing applications. The performance

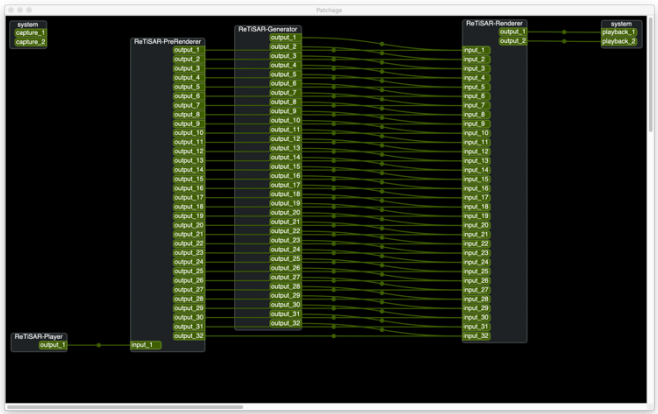

**Fig. 3**: Screenshot illustrating the JACK connections when pre-rendering and addition of synthetic microphone self-noise are applied at 4th order

[13] https://drobilla.net/software/patchage

limitation of the current version is at 8th order for IR sets utilizing the described pre-rendering on a standard laptop. This contains additive noise generation as well as an estimated equivalent of two fast-convolutions for each of 110 channels. However, this currently requires a block length of as long as $l = 4096$ when dropouts of the audio signal are not tolerated. Eigenmike live and recorded streams can currently be rendered at $l = 512$, resulting in a lower head-tracking and overall processing latency. Improvements of the efficiency by further optimizations are expected.

Gaining flexibility regarding supported head-tracking interfaces as well as head related or array IR set, i.e. support of the SOFA (Spatially Oriented Format for Acoustics) [13] standard, are planned for the near future.

We have not found that operations such as garbage collection, which are being performed by the underlying Python framework, cause audible impairment. When reaching the system individual performance capabilities, dropouts start to occur when switching between different applications by the operating system, which is MacOS in the present case.

## Acknowledgements

We thank David Lou Alon, Sebastià V. Amengual Garí and Ravish Mehra of Facebook Reality Labs for advice and financial support.

## References

[1] Schörkhuber, C., Zaunschirm, M., and Höldrich, R.: "Binaural Rendering of Ambisonic Signals via Magnitude Least Squares" in Proc. of DAGA, Deutsche Gesellschaft für Akustik, Munich, Germany, p. 339–342, 2018 (online: https://plugins.iem.at/)

[2] McCormack, L. and Politis, A.: "SPARTA & COMPASS: Real-time implementations of linear and parametric spatial audio reproduction and processing methods" in Proc. of the Conf. on Immersive and Interactive Audio, Audio Engineering Society, York, UK, 2019 (online: http://research.spa.aalto.fi/projects/sparta_vsts/plugins.html)

[3] Bernschütz, B., Pörschmann, C., Spors, S. and Weinzierl, S.: „SOFiA Sound Field Analysis Toolbox," in Proc. of the Int. Conf. on Spatial Audio (ICSA), Detmold, Germany, 2011 (online: http://audiogroup.web.th-koeln.de/SOFiA_wiki/ABOUT.html)

[4] Hohnerlein, C. and Ahrens, J.: "Microphone Array Processing in Python with the sound_field_analysis-py Toolbox" in Proc. of DAGA, Deutsche Gesellschaft für Akustik, Kiel, Germany, 2017 (online: https://github.com/AppliedAcousticsChalmers/sound_field_analysis-py)

[5] Geier, M., Spors, S. and Ahrens, J.: "The SoundScape Renderer: A Unified Spatial Audio Reproduction Framework for Arbitrary Rendering Methods" in Proc. of 124th Convention of the AES, Amsterdam, The Netherlands, 2008 (online: http://spatialaudio.net/ssr/)

[6] Stade, P., Bernschütz, B. and Rühl, M.: "A Spatial Audio Impulse Response Compilation Captured at the WDR Broadcast Studios" in Proc. of 27th Tonmeistertagung, Cologne, Germany, 2012 (online: http://audiogroup.web.th-koeln.de/wdr_irc.html)

[7] Avni, A., Ahrens, J., Geier, M., Spors, S., Wierstorf, H., and Rafaely, B.: "Spatial perception of sound fields recorded by spherical microphone arrays with varying spatial resolution" in Journal of the Acoustical Society of America 133(5), p. 2711–2721, 2013

[8] Ahrens, J. and Andersson, C.: "Perceptual Evaluation of Headphone Auralization of Rooms Captured with Spherical Microphone Arrays with Respect to Spaciousness and Timbre," in Journal of the Acoustical Society of America, 2019 (accepted for publication)

[9] Zaunschirm, M., Schörkhuber, C. and Höldrich, R.: "Binaural rendering of ambisonic signals by head-related impulse response time alignment and a diffuseness constraint" in Journal of the Acoustical Society of America 143(6), p. 3616–3627, 2018

[10] Andersson, C.: "Headphone auralization of acoustic spaces recorded with spherical microphone arrays" Master's thesis, Chalmers University of Technology, 2017

[11] Lösler, S., and Zotter, F.: "Comprehensive Radial Filter Design for Practical higher-order Ambisonic Recording" in Proc of DAGA, Nuremberg, Germany, March, 2015

[12] Zotter, F.: "A Linear-Phase Filter-Bank Approach for Rigid-Sphere Mic-Array Recording" in Proc of IcETRAN Conference, Palic, Serbia, June, 2018

[13] Majdak, P., Iwaya, Y., Carpentier, T., Nicol, R., Parmentier, M., Roginska, A., Suzuki, Y., et al.: "Spatially oriented format for acoustics: A data exchange format representing head-related transfer functions," in Proc. of 134th Convention of the AES, Rome, Italy, 2013