Models for Real-Time Binaural Rendering using Church Impulse Responses

Olle Persson

Master's thesis
2026:E39

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Master's Theses in Mathematical Sciences 2026:E39
ISSN 1404-6342

LUTFMS-3554-2026

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/

Models for Real-Time Binaural Rendering using Church Impulse Responses

Olle Persson

May 2026

Abstract

Reproducing the spatial acoustics of a room is key to the feeling of spatiality of audio in headphones, with applications in, for example, cultural preservation. The propagation of sound between a point source and a receiver in a room is commonly modelled as a linear time-invariant system, where its room impulse response fully models how sound propagates. By combining the room impulse response and head-related impulse response, which model how a person's body filters sound, we can create a binaural room impulse response, which enables spatial audio reproduction via headphones. However, the room impulse responses are often long, leading to substantial computational demand for naive real-time processing.

This thesis looks into different modelling approaches for real-time filtering of room impulse responses measured in the Santi Marcellino and Pietro church, Cremona, Italy. Different models are used to investigate how room impulse responses can be represented compactly and computationally efficiently, while preserving accuracy for use in real-time processing. These methods include linear convolution, partitioned frequency-domain convolution, and a model exploiting the low-rank structure of the binaural room impulse responses.

Each model represents the impulse response differently and therefore gives rise to different trade-offs. Linear convolution preserves the full time-domain representation and serves as a natural reference. Partitioned frequency-domain convolution uses a blockwise frequency-domain representation, which provides the lowest computational cost but introduces an inherent latency. The low-rank model instead enforces a reduced-dimensional approximation, which, in turn, lowers storage requirements while offering a balance between computational cost and model complexity. These findings give rise to a general guideline for which application each method is most suited for.

Preface

This work was carried out during the spring semester of 2025/2026 and serves as the capstone project for my Master's in Engineering Mathematics at the Faculty of Engineering, Lund University.

I would like to thank my supervisors, David Sundström and Oskar Keding, for their equal support throughout this thesis. Their guidance in both theory and writing, together with their genuine interest in the project, has been greatly appreciated.

Table of Contents

Abstract i
Preface ii
Acronyms v

1 Introduction **1**

2 Background **3**
2.1 Room Impulse Responses 3
2.2 Head Related Impulse Responses 3
2.3 Spherical Harmonics 4
2.4 Convolution for Filtering 5

3 Method **7**
3.1 Data Sets 7
3.1.1 Church Impulse Responses 7
3.1.2 Head-Related Impulse Responses and Source Signals 8
3.2 Models for Offline Filtering 8
3.2.1 Basic Pipeline for Offline Filtering 8
3.2.2 Binaural Room Impulse Response 9
3.2.3 Acceleration Techniques 10
3.2.4 Low-Rank approximation 11
3.3 Models for Real-time Filtering 14
3.3.1 Linear Convolution 14
3.3.2 Low-Rank Model 14
3.3.3 Partitioned Frequency-Domain Convolution 14
3.3.4 Downsampling and Truncation 16
3.4 Complexity of methods 16

4 Results **17**
4.1 Rank Approximation 17
4.2 Truncation 19
4.3 Comparison of Methods 19

5 Discussion **23**
5.1 Selection of Rank 23
5.2 Avenues for improvements 24

6 Conclusion **26**

Bibliography **27**

Disclosure of AI Use **29**

7 Appendix **30**

Acronyms

BRIR binaural room impulse response

DFT discrete Fourier transform

FFT fast Fourier transform

GUI graphical user interface

HRIR head-related impulse response

IFFT inverse fast Fourier transform

IR impulse response

LTI linear time-invariant

MAC multiply–accumulate

MSE mean square error

PFD partitioned frequency-domain

PSVD partitioned singular value decomposition

RIR room impulse response

SH spherical harmonics

1

Introduction

A desire to preserve the experience of culturally significant buildings, for either future use or accessibility, is evident. One facet of preserving a building's characteristics is its acoustics, that is, the auditory experience of a listener inside the building. However, reproducing this experience accurately and in real-time is often a challenge in spatial audio.

Typically, the sound pressure from a point source to a recipient in a room is modelled as a linear time-invariant (LTI) system, meaning that for a source position and a receiver position, the acoustic behaviour is fully characterised by its room impulse response (RIR). By convolving an arbitrary source signal with the RIR, we can recreate how that signal would sound inside that room. Combining this with a head-related impulse response (HRIR) will give us a binaural room impulse response (BRIR), further enhancing the spatial quality of the audio, especially for headphones, by giving a positional and body-aware filtering process. However, in large open rooms such as churches, RIRs can be several seconds long, making sample-by-sample filtering using convolution impossible with current hardware. This necessitates a faster filtering approach, achievable by modelling the BRIR.

There already exist several methods for efficient filtering using BRIRs. Frequency domain methods make use of the convolution theorem [1] to accelerate filtering. Often, a version of frequency domain convolution using partitioning is used for real-time filtering [2], which allows for faster processing, but comes with a latency. Low-rank approximation methods were proposed by Mitra et al. [3], and then extended specifically for BRIRs by Atkins et al. [4]. The low-rank structure of RIRs has also been studied extensively in [5], where several methods are presented for different application scenarios.

From a modelling perspective, the central question is how long BRIRs can be represented in a computationally efficient manner while still preserving relevant acoustics structure. Since BRIRs can contain considerable redundancies, finding a model that exploits this may reduce computational load, while still preserving the characteristics of the signal. Although real-time filtering is not the primary question, it acts as a consistent motivator and a performance benchmark for the models evaluated throughout this thesis.

This thesis investigates three mathematical representations of BRIR filtering: discrete convolution, partitioned frequency domain convolution and a low-rank model based on singular value decomposition. Also, focus will be given to the trade-off between these three

modelling approaches, in terms of their approximation accuracy, computational complexity, storage requirements and perceptual quality.

The thesis first introduces the mathematical tools required for modelling BRIRs, and then describes the different models and how their interpretation changes depending on whether they are used for offline or online filtering. To evaluate the methods, results for a number of metrics are presented. Finally, the viability of each method and possible future improvements are discussed.

2

Background

Background on the mathematical tools used.

2.1 Room Impulse Responses

The impulse response (IR) of a system is defined as the output of the system when an impulse is applied as the input. A LTI system, $\mathcal{L}(\cdot)$ is defined by:

- Linearity: $\mathcal{L}(ax_1(t) + bx_2(t)) = a\mathcal{L}(x_1(t)) + b\mathcal{L}(x_2(t))$, where a and b are constants.
- Time invariance: $\mathcal{L}(x(t)) = y(t) \implies \mathcal{L}(x(t+T)) = y(t+T)$.

An LTI system is completely defined by its IR, meaning theoretically if it is known for a system, we can perfectly emulate it, with a convolution between the impulse response $h(t)$, and a source signal $x(t)$,

$$y(t) = h(t) * x(t). \tag{2.1}$$

We assume our RIRs to be causal, meaning that they only depend on the current or previous inputs to the system. Moreover, they are presumed to be of finite length. In real-world scenarios, getting the exact impulse response is impossible, and we have to rely on approximations. On top of this, there is noise added to the measurements, and in this thesis, this will not be accounted for.

How sound propagates from an origin position to a receiver in a room is commonly modelled as an LTI system, where a short loud noise can be interpreted as the relating RIR. Under this assumption, we can recreate the sound of a source signal from the location of the original impulse, at the position where the RIR sound pressure was originally measured, using Eq. 2.1. However, for particularly large rooms, such as a church, these RIRs can be of hundreds of thousands of samples long, making real-time filtering challenging.

2.2 Head Related Impulse Responses

Head-related impulse responses (HRIRs) describes how sound from a specific direction is filtered by a human's body before reaching the ear. This includes the effect of the head and outer ear, which can affect the exact time audio comes into our ears. This is important for spatial hearing in humans, since this is what allows our brain to determine direction

using only two ears.

Similarly to RIRs, HRIRs can be modelled as the IR of an LTI system. However, they differ in that HRIRs are inherently direction dependent, meaning that for each direction $\Omega = (\theta, \phi)$, where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$, there exists an IR for each ear

$$h^e[\Omega, k] \quad e \in \{L, R\},$$

where L denotes the left ear, and R denotes the right ear.

When a single channel source signal $x[k]$ is filtered with a HRIR, for a given Ω, a binaural signal is produced

$$y^e[k] = x[k] * h^e[\Omega, k]$$

This allows for spatial rendering of audio for headphones, since the HRIR for the left and right ear does the filtering the body would otherwise have done, which is what informs the brain of the direction of sound. This method is often referred to as binaural rendering, since it produces two audio channels, one for the left ear $y^L[k]$, and one for the right ear $y^R[k]$.

HRIRs are measured/collected by using an artificial head or a real human in an anechoic chamber. The source is placed in multiple directions on a spherical grid around the listener, while the impulse responses are measured for all directions in the ear canals of the listener [6]. These measurements can later on be represented in the spherical harmonic basis.

In this thesis, the implicit body filtering of HRIRs is used in combination with the room-specific RIRs to create binaural room impulse responses (BRIRs), which will be used for binaural rendering.

2.3 Spherical Harmonics

Spherical harmonics (SH) [7] are a set of orthogonal basis functions defined on the surface of a sphere. They are often used as a representation for directional data, in this particular case, sound fields on a sphere.

Let $\Omega = (\theta, \phi)$ be a point on the unit sphere, where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$. The (complex-valued) spherical harmonics are defined as

$$Y_{lm}(\Omega) = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}} P_{lm}(\cos(\theta)) e^{im\phi},$$

where $l \geq 0$ is the degree, $m \in [-l, l]$ is the order and $P_l^m(\cdot)$ are the associated Legendre polynomials.

The spherical harmonics form an orthonormal basis for L2 functions on a sphere. If $\chi(\Omega)$ is such a function, it can be expressed as

$$\chi(\Omega) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \chi_{lm} Y_{lm}(\Omega),$$

where

$$\chi_{lm} = \int_{\Omega \in S^2} \chi(\Omega) Y_{lm}(\Omega) d\Omega.$$

However, in this thesis, only real-valued signals are considered, and thus, the real SH basis $R_{lm}(\Omega)$ is used

$$R_{lm}(\Omega) = \begin{cases} \frac{i}{\sqrt{2}}(Y_{lm}(\Omega) - (-1)^m Y_{l(-m)}(\Omega)) & m < 0 \\ Y_{l0}(\Omega) & m = 0 \\ \frac{1}{\sqrt{2}}(Y_{l(-m)}(\Omega) - (-1)^m Y_{lm}(\Omega)) & m > 0 \end{cases}.$$

Using the real basis, we get

$$\chi(\Omega) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \chi_{lm} R_{lm}(\Omega),$$

where

$$\chi_{lm} = \int_{\Omega \in S^2} \chi(\Omega) R_{lm}(\Omega) d\Omega$$

is called the SH component, and is a scalar coefficient that, from a filtering perspective, can be interpreted as a weight associated with each basis function $R_{lm}(\Omega)$.

Let $y_i[k]$ be a discrete signal sampled at directions Ω_i, $i = 1, \dots n$, we can estimate $\chi_{lm}[k]$ with

$$\boldsymbol{\chi}[k] = \min_{\boldsymbol{\chi}[k]} \sum_{i=1}^{n} \left(y_i[k] - \sum_{l=0}^{L} \sum_{m=-l}^{l} \chi_{lm}[k] R_{lm}(\Omega_i) \right)^2, \tag{2.2}$$

where

$$\boldsymbol{\chi}[k] = \begin{bmatrix} \chi_{00}[k] & \chi_{1(-1)}[k] & \chi_{10}[k] & \chi_{11}[k] & \chi_{2(-2)}[k] & \dots & \chi_{LL}[k] \end{bmatrix}^T \in \mathbb{R}^{(L+1)^2}.$$

In this thesis, spherical harmonics are used as a basis for the representation of both RIRs and HRIRs, allowing for the encoding of spatial information in a direction-independent manner. These are leveraged to compute BRIRs, which is a 2-channel filter which allows for binaural rendering of audio.

2.4 Convolution for Filtering

Convolution is important since, as stated above, it is the filtering process between an arbitrary source signal and an IR if we have a discrete LTI system.

Convolution between functions/signals f and g, in the continuous case is defined as

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau.$$

However, as we will only be working with discrete samples, the discrete equivalent is of interest

$$f[k] * g[k] = \sum_{m=-\infty}^{\infty} f[m] g[k - m].$$

In applications, our signals are not of infinite samples, and thus, the convolution can be written as

$$f[k] * g[k] = \sum_{m=0}^{M-1} f[m]g[k-m],$$

where M is the length of f.

The convolution theorem [1] states that convolution in the time domain is equivalent to multiplication in the frequency domain, that is, after a Fourier transform. So one could use the Fourier transform on both signals, multiply them, and then apply the inverse Fourier transform to obtain the same results as with linear convolution in the time domain. Often, a fast Fourier transform (FFT) and its inverse (IFFT) are used for the calculation of the transform.

The discrete Fourier transform (DFT) is defined as

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{k}{N} n},$$

where $\boldsymbol{X}$ and $\boldsymbol{x} = [x_0\ x_1\ \ldots\ x_{N-1}]$ are sequences of complex numbers of length N.

Where the DFT differs from the continuous case is that, instead of multiplication in the frequency domain being equivalent to convolution in the time domain, now multiplication in the frequency domain is equivalent to circular convolution

$$f[k] \circledast g[k] = \sum_{m=0}^{M-1} f[m]g[k-m] \mod M.$$

We can make the circular convolution between two discrete signals

$$\boldsymbol{f} = [f[0]\ f[1]\ \ldots\ f[n_f - 1]]$$
$$\boldsymbol{g} = [g[0]\ g[1]\ \ldots\ g[n_g - 1]]$$

equivalent to linear convolution by zero-padding both signals

$$\boldsymbol{f}_0 = [f_0[0]\ f_0[1]\ \ldots\ f_0[n_{f_0} - 1]] = [\boldsymbol{f}\ \mathbf{0}_f] \tag{2.3}$$
$$\boldsymbol{g}_0 = [g_0[0]\ g_0[1]\ \ldots\ g_0[n_{g_0} - 1]] = [\boldsymbol{g}\ \mathbf{0}_g], \tag{2.4}$$

where $\mathbf{0}_f$ is a zero column vector, of length greater than or equal to: $n_g - 1$, similarly $\mathbf{0}_g$ is a zero column vector, of length greater than or equal to: $n_f - 1$, and $n_{f_0} = n_{g_0}$. Having zero-padded both signals, we get the desired outcome

$$f[k] * g[k] = f_0[k] \circledast g_0[k].$$

This means that if we properly zero-pad two signals, they can be identically filtered in the frequency domain, as in the time domain.

3

Method

This section describes the mathematical models used to represent and approximate binaural room impulse responses. The methods differ in how the BRIR is represented: either directly in the time-domain, as blocks in the frequency-domain, or in a low-rank manner. Data sets used are initially presented.

3.1 Data Sets

The data used for this thesis is presented below, where the church IR data set is of particular interest, as open-access data sets recorded with multichannel microphones in church environments are scarce.

3.1.1 Church Impulse Responses

The RIRs used for this thesis are a total of 90 different RIRs measured in the Santi Marcellino and Pietro church, Cremona, Italy [8].

Since we can not produce a pure impulse in real-life scenarios, the source signal used in place of it is the logarithmic sine sweep

$$s[k] = A_{in} \sin\left(2\pi f_1 L \exp\left(\frac{k}{f_s L}\right)\right),$$

where k is the sample index, $A_{in} = 0.5$ is the amplitude of the sweep, $f_s = 48$ kHz is the sampling rate, $L = \frac{T}{\log(f_2/f_1)}$, where $f_1 = 22$ Hz and $f_2 = 22$ kHz, which are the intial and final frequencies. These measurements are deconvolved and normalised, resulting in the approximated RIRs.

The RIRs were measured with two types of microphones, one omnidirectional microphone and one spherical multi-channel microphone, the latter being the em32 Eigenmike®. For this thesis, only the em32 Eigenmike®, which is a 4th-order 32-channel microphone, will be considered.

The measurements were conducted by placing the source in one of three source locations, and the microphone at one of thirty microphone locations, giving a total of 90 RIRs.

The different placements of the source and microphones can be seen in Fig. 3.1, where the red stars indicate source location, and the blue triangles indicate microphone locations.

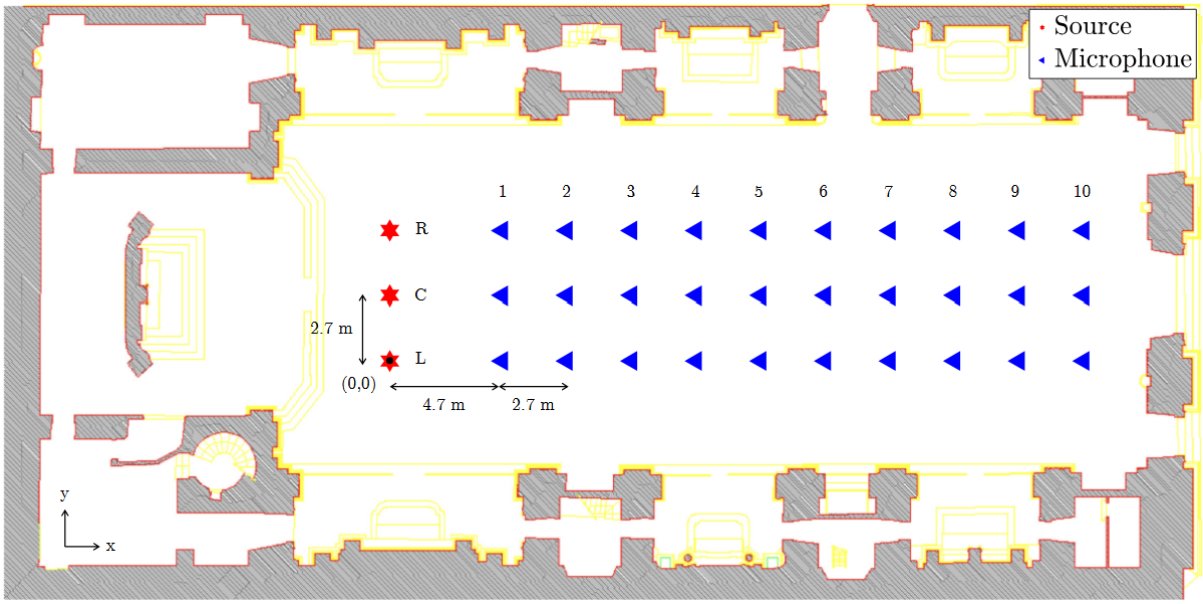


Figure 3.1: Top view of the church, where red stars indicate source positions, and blue triangles indicate microphone positions [8].

The RIR used for plots and metrics was the centre row and bench 1, with the original sampling rate of 48 kHz, unless otherwise specified.

3.1.2 Head-Related Impulse Responses and Source Signals

For HRIR, the ARI database was chosen to source an HRIR [9], the HRIR in itself was arbitrarily chosen from the database. The HRIR has a sampling rate of 48 kHz. Due to HRIRs being highly personalised, the resulting spatial feeling might vary from person to person, which depends on one's overall body and ear shape, relative to the original person.

To test the methods proposed later on, an audio file sourced from the Open Speech and Language Resources website [10] is used. The signal has a sampling rate of 16 kHz, but has been upsampled or downsampled depending on the method used. The audio itself is a snippet of speech from an audiobook, and is 31.7 s long.

3.2 Models for Offline Filtering

While real-time filtering might be of more interest with respect to modelling of RIRs, the offline case is still worth investigating, if only to characterise the problem. In offline computing, the computational efficiency is not of as great importance as in real-time, as long as the computational time is reasonable.

3.2.1 Basic Pipeline for Offline Filtering

Let $x[k]$ denote a single-channel source signal. The objective is to render a binaural signal using RIRs and HRIRs represented in the SH domain of the Lth maximum order

(in this particular case $L = 4$). Assume that the RIRs are measured at $D \geq (L+1)^2$ directions Ω_d, yielding $h_d[k]$ for $d = 1, \ldots, D$. At each sample k, the directional responses are approximated as

$$h_d[k] \approx \sum_{l=0}^{L} \sum_{m=-l}^{l} R_{lm}(\Omega_d)\, \chi_{lm}[k].$$

We can solve for $\chi_{lm}[k]$ using Eq. (2.2). The source signal is filtered independently by each SH component,

$$s_{lm}[k] = x[k] * \chi_{lm}[k].$$

Similarly to the RIR, we approximate the HRIR for each ear $e \in \{L, R\}$, from the $\Delta \geq (L+1)^2$ measured directions Ω_δ, yielding $\eta^e_\delta[k]$ for $\delta = 1, \ldots, \Delta$ as

$$\eta^e_\delta[k] \approx \sum_{l=0}^{L} \sum_{m=-l}^{l} R_{lm}(\Omega_\delta)\, \psi^e_{lm}[k].$$

Again, we use Eq. (2.2) to solve for $\psi^e_{lm}[k]$. The binaural signal is obtained by filtering each SH component in $s_{lm}[k]$ and the HRIR, and then summing each component

$$y^e[k] = \sum_{l=0}^{L} \sum_{m=-l}^{l} s_{lm}[k] * \psi^e_{lm}[k].$$

Substituting $s_{lm}[k]$ gives the system

$$y^e[k] = \sum_{l=0}^{L} \sum_{m=-l}^{l} x[k] * \chi_{lm}[k] * \psi^e_{lm}[k]. \tag{3.1}$$

Before $y^e[k]$ can be written as an audio file, it needs to be normalised such that $y^e[k] \in [-1, 1]$. This can be done in many ways, but for this purpose, all samples are divided by the maximum absolute value in both y^L and y^R

$$p = \max(\max(|y^L|), \max(|y^R|)), \tag{3.2}$$

$$y^L \leftarrow \frac{1}{p} \cdot y^L$$
$$y^R \leftarrow \frac{1}{p} \cdot y^R.$$

3.2.2 Binaural Room Impulse Response

With precomputation, we can leverage the fact that we are dealing with an LTI system in two ways, firstly, $\chi_{lm}[k]$ and $\psi^e_{lm}[k]$ can be precomputed for each combination of l, m and k. Secondly, the order of convolutions and summations in Eq. (3.1) does not matter, and thus we can first compute

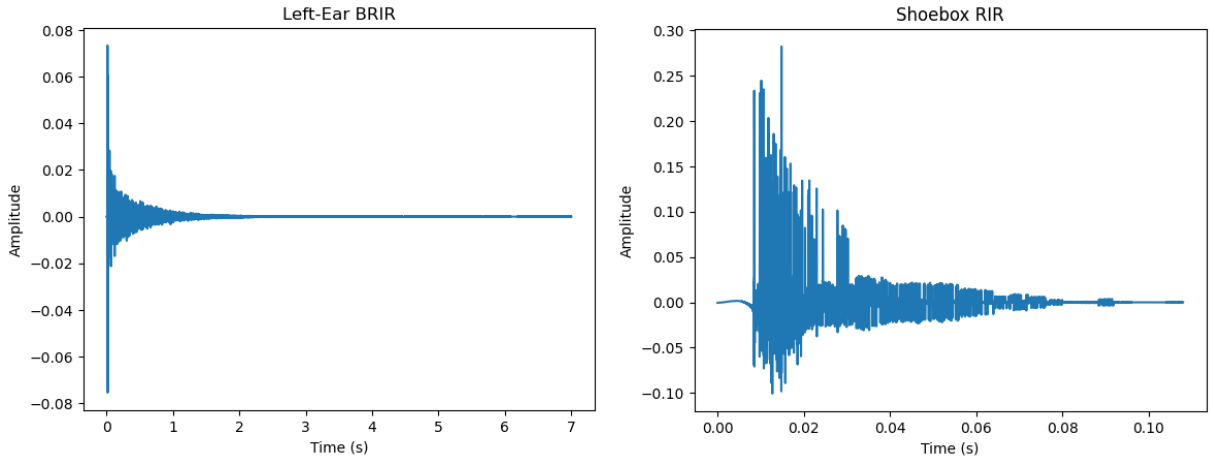
$$w^e[k] = \sum_{l=0}^{L} \sum_{m=-l}^{l} \chi_{lm}[k] * \psi^e_{lm}[k],$$

and then

$$y^e[k] = x[k] * w^e[k]. \tag{3.3}$$

By computing and storing every possible $w^e[k]$, the $n_h + 2n_\eta$ least-squares computations and $3(L+1)^2$ convolutions, one between signals of length n_x and n_h, and two between signals of length $n_x + n_h - 1$ and n_η, reduce to just two convolutions between signals of length n_x and $n_w = n_h + n_\eta - 1$. This new set of samples, $w^e[k]$, can be referred to as a binaural room impulse response (BRIR).

Figure 3.2 shows the left channel BRIR in the time domain together with a simulated RIR for a shoebox room with dimensions (2.75, 3.40, 2.50) m. This illustrates how much longer the BRIR is compared with the RIR of a smaller room.



(a) Left ear BRIR in the time-domain.

(b) Shoebox RIR in the time-domain.

Figure 3.2: Comparison between used BRIR and simulated RIR in a shoebox room of dimensions (2.75, 3.40, 2.50) m.

3.2.3 Acceleration Techniques

Filtering by Eq. (3.3) is computationally efficient compared to doing the full pipeline each time, therefore, all methods will be based on it. Below are some methods to further decrease computation time.

Downsampling. The BRIR can be downsampled to decrease computation, for example, if both the original RIR and original HRIR have a sampling rate of 48 kHz, downsampling with a factor of two results in a sampling rate of 24 kHz. This downsample of IRs will directly lead to a reduction in computational cost, where we essentially can lessen the amount of multiply-accumulate operations by having fewer samples in our filter.

Truncation. We can truncate the BRIR, meaning that we choose a sample, and then simply remove all samples after that one specific sample, essentially compacting the BRIR to the detriment of approximation accuracy. We truncate the BRIR as

$$w_T^e[k] = \begin{cases} w^e[k] & k \leq n_T \\ 0 & k > n_T \end{cases},$$

and thus we can simply remove all samples after n_T. One wants to choose n_T such that we can remove a meaningful amount of samples while still producing good results when

used for filtering.

This paper's method of deciding n_T is by finding the split point where the tails are a small percentage of the total energy found in the BRIR

$$E^e_{\text{tail}} \leq \varepsilon \cdot E^e_{\text{total}},$$

where

$$E^e_{\text{total}} = \sum_{k=1}^{n_w} w^e[k]^2,$$

$$E^e_{\text{tail}} = \sum_{k=n_T}^{n_w} w^e[k]^2$$

and ε is a factor by our choosing. To ensure that the two left and right channels have the same length, we let $n_T^L = n_T^R = \max(n_T^L, n_T^R)$.

Frequency-Domain Convolution. Lastly, by using the discrete Fourier transform (and later its inverse), implemented with a fast Fourier transform algorithm, on the discrete signals to be convolved, such as, $x[k]$ and $h[k]$, we can utilise that discrete circular convolution in the time domain, which corresponds to pointwise multiplication in the discrete frequency domain. However, since we are actually interested in linear convolution, we need to zero-pad $x[k]$ and $h[k]$ at the end so that they both have a length of at least $n_x + n_h - 1$, as seen in Eq. (2.4).

3.2.4 Low-Rank approximation

Approximating a matrix by one of lower rank is particularly useful when the matrix has a low effective rank, that is, when its singular values are heavily dominated by a few large ones. This indicates that most of the matrix's information is captured by only a small number of components. The low-rank approximation considered here can be interpreted as a dimensionality reduction model. Instead of representing the BRIR by all of its samples, the purpose is to capture the dominant components of the signal. If the effective rank is low, a compact representation can preserve much of the signal content while reducing model complexity.

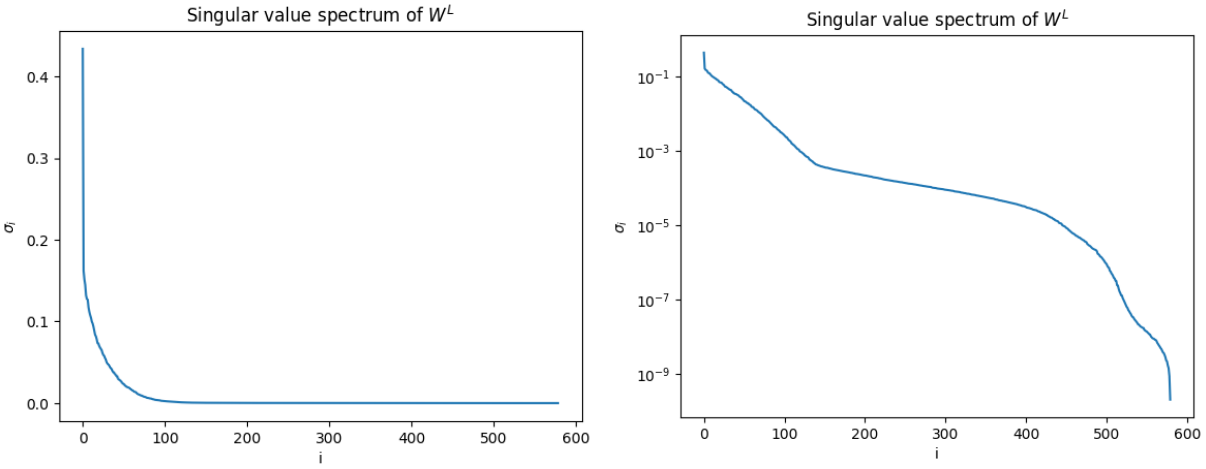
BRIRs are often very long and can possibly contain redundancies, as seen in Fig. 3.2a, thus, low-rank approximation as a premise for a model can be useful. However, to do a low-rank approximation, we need to be able to express our BRIR as a matrix.

If we let $\boldsymbol{w}^e = [w^e[0]w^e[1] \dots w^e[n_w - 1]]$ be one channel of our BRIR, we can reshape it into a matrix as

$$\boldsymbol{W}^e = \begin{bmatrix} w^e[0] & w^e[N] & \dots & w^e[(P-1)N] \\ w^e[1] & w^e[N+1] & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w^e[N-1] & \dots & \dots & w^e[P(N-1)] \end{bmatrix} \in \mathbb{R}^{N \times P}. \tag{3.4}$$

Where we let $P = \lfloor \sqrt{n_w} \rfloor$, $N = \lceil \frac{n_w}{P} \rceil$ and, if needed, zero-pad $\boldsymbol{w}^e$ such that it is of length $P \cdot N$.

A good indicator that a low-rank approximation of a matrix is motivated is if the sorted singular values of the matrix in descending order have an early sharp decay. To find an appropriate rank approximation of the left channel, $\boldsymbol{W}^L$, and to check if a low-rank approximation is motivated to begin with, in Fig. 3.3, the singular values are found and plotted in descending order.



(a) Linear y-axis.

(b) Logarithmic y-axis.

Figure 3.3: Singular value spectrum of $\boldsymbol{W}^L$, plotted in descending order.

In Fig. 3.3, we can clearly see an early sharp decay, and we can also conclude that a matrix rank approximation around 100 is reasonable.

To find a low-rank approximation of $\boldsymbol{W}^e$, we use singular value decomposition (SVD) [11]

$$\boldsymbol{W}^e = \boldsymbol{U}^e \boldsymbol{S}^e \boldsymbol{V}^{eT}, \tag{3.5}$$

where $\boldsymbol{U}^e$ and $\boldsymbol{V}^{eT}$ are $N \times N$ and $P \times P$ matrices. $\boldsymbol{S}^e$ is an $N \times P$ matrix with $\boldsymbol{W}^e$'s singular values on the diagonal in descending order.

With this, we can create an R-rank matrix, $\boldsymbol{W}_R^e$, which approximates $\boldsymbol{W}^e$ as

$$\boldsymbol{W}_R^e = \boldsymbol{U}_R^e \boldsymbol{S}_R^e \boldsymbol{V}_R^{eT} = \begin{bmatrix} \sigma_0^e \boldsymbol{u}_0^{eT} \\ \sigma_1^e \boldsymbol{u}_1^{eT} \\ \vdots \\ \sigma_{R-1}^e \boldsymbol{u}_{R-1}^{eT} \end{bmatrix}^T \begin{bmatrix} v_0^{e0} & v_0^{e1} & \dots & v_0^{e(P-1)} \\ v_1^{e0} & v_1^{e1} & \dots & v_1^{e(P-1)} \\ \vdots & \vdots & \ddots & \vdots \\ v_{R-1}^{e0} & v_{R-1}^{e1} & \dots & v_{R-1}^{e(P-1)} \end{bmatrix} \tag{3.6}$$

where $\boldsymbol{U}_R^e = [\boldsymbol{u}_0^e \ \boldsymbol{u}_1^e \ \dots \ \boldsymbol{u}_{R-1}^e]$, where $\boldsymbol{u}_r^e$ is the r-th column of $\boldsymbol{U}^e$. $\boldsymbol{S}_R^e$ is an $R \times R$ matrix with the R-th biggest singular values of $\boldsymbol{W}^e$ on the diagonal in descending order.

This low-rank approximation $\boldsymbol{W}_R^e$, produces the lowest error in the Frobenius norm $||\boldsymbol{W}^e - \cdot||_F$, for matrices of rank R [12].

Mitra et. al. [3] proposed this method

$$y^e[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} v_r^{ep} \sigma_r^e \boldsymbol{u}_r^{eT} \boldsymbol{x}[k - pN],$$

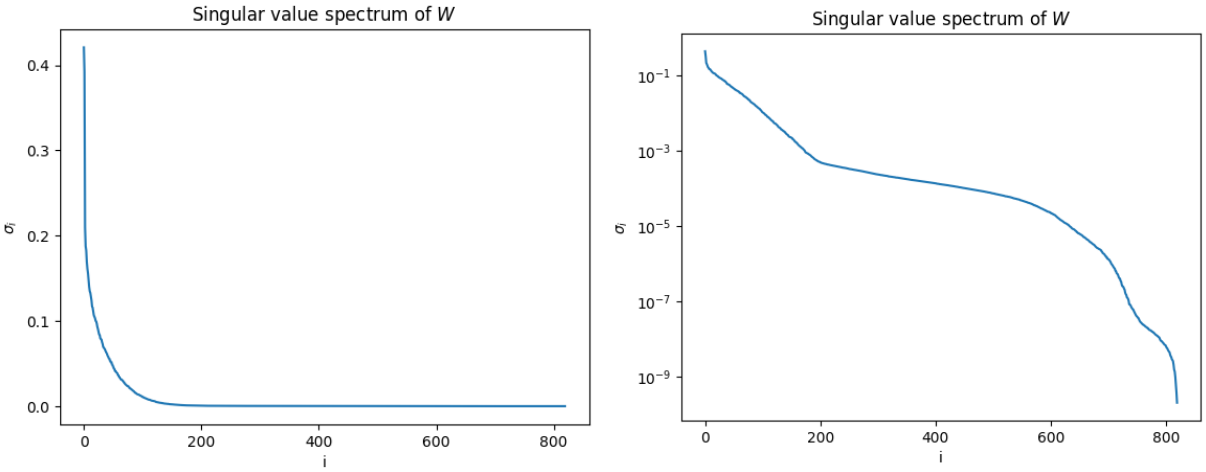
where $\boldsymbol{x}[k]$ is the N last samples from k, and v_r^{ep} is the element of V_R^{eT}. With this, we can create a filterbank of R filters,

$$z_r^e[k] = \sigma_r^e \boldsymbol{u}_r^{eT} \boldsymbol{x}[k].$$

Then we let

$$y^e[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} v_r^{ep} z_r^e[k - pN].$$

Instead of doing this twice for $\boldsymbol{w}^L$ and $\boldsymbol{w}^R$ separately, this can be further accelerated for BRIR filtering. Atkins et al. [4] propose a method which makes use of this low-rank approximation while simultaneously working well for BRIRs. Let $\boldsymbol{w} = [w^L[0]\ w^L[1]\ \dots\ w^L[n_w - 1]\ w^R[0]\ w^R[1]\ \dots\ w^R[n_w - 1]]$, we reshape in a similar way as eq. (3.4) to $\boldsymbol{W}$, we check that a low-rank approximation is motivated:



(a) Linear y-axis.

(b) Logarithmic y-axis.

Figure 3.4: Singular value spectrum of $\boldsymbol{W}$, plotted in descending order.

Seeing the sharp decay in Fig. 3.4, a low-rank approximation is indeed motivated.

Then we follow Eq. (3.5) and Eq. (3.6), and get the low-rank approximation of $\boldsymbol{W}$ as

$$\boldsymbol{W}_R = \boldsymbol{U}_R \boldsymbol{S}_R \boldsymbol{V}_R^H = \begin{bmatrix} \sigma_0 \boldsymbol{u_0} \\ \sigma_1 \boldsymbol{u_1} \\ \vdots \\ \sigma_{R-1} \boldsymbol{u_{R-1}} \end{bmatrix}^T \begin{bmatrix} v_0^0 & v_0^1 & \dots & v_0^{(P-1)} & q_0^0 & q_0^1 & \dots & q_0^{(P-1)} \\ v_1^0 & v_1^1 & \dots & v_1^{(P-1)} & q_1^0 & q_1^1 & \dots & q_1^{(P-1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{R-1}^0 & v_{R-1}^1 & \dots & v_{R-1}^{(P-1)} & q_{R-1}^0 & q_{R-1}^1 & \dots & q_{R-1}^{(P-1)} \end{bmatrix}.$$

The filtering methods become

$$y^L[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} v_r^p \sigma_r \boldsymbol{u}_r^T \boldsymbol{x}[k - pN]$$

$$y^R[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} q_r^p \sigma_r \boldsymbol{u}_r^T \boldsymbol{x}[k - pN],$$

where they can share the same filterbank

$$z_r[k] = \sigma_r \boldsymbol{u}_r^T \boldsymbol{x}[k].$$

Giving us the model

$$y^L[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} v_r^p z_r[k - pN]$$
$$y^R[k] = \sum_{p=0}^{P-1} \sum_{r=0}^{R-1} q_r^p z_r[k - pN],$$

which will be referred to as the partitioned singular value decomposition (PSVD) model.

3.3 Models for Real-time Filtering

Real-time filtering with minimal latency is done sample by sample, meaning that a sample to be filtered is read, we filter it (including potential previous samples), and then we output one sample. To make this work, we need to be able to read, convolve and write in the inverse of the sample rate. Assuming a sample rate of 48 kHz, this allows for $\frac{1}{48000} \approx 20.8$ µs to do all computations. Often, there are other overheads in the real-time system, meaning that the convolution will need to be quite a lot faster than 20.8 µs.

3.3.1 Linear Convolution

In real-time processing, where we filter samples by sample, convolution becomes equivalent to a scalar product between the BRIR, $w^e[k]$ and a buffer vector $x[k]$ of length n_w

$$y^e[k] = \sum_{i=0}^{n_w-1} w^e[i]x[i].$$

The buffer vector is built by inserting the latest sample at $x[0]$ and shifting each value to the right, removing the last. This buffer vector is initialised with zeros.

3.3.2 Low-Rank Model

Real-time processing does not change the algorithm much of the low-rank model, all $\boldsymbol{u}_r$, σ_r, v_r^p and q_r^p are precomputed beforehand and are all loaded into memory when needed.

3.3.3 Partitioned Frequency-Domain Convolution

As opposed to offline filtering, we cannot FFT the whole source signal, since we do not have access to it, considering we strive for sample-by-sample filtering. However, the Fourier transform for just one sample contains all frequencies. To solve this, we can sample in blocks of samples instead of a single sample. Unfortunately, this will introduce a delay as we will need to wait for samples before we can turn them into one block and convolve. Moreover, too small a block could reduce the quality of the output due to lower resolution in the frequency domain, in other words, we encounter an uncertainty principle problem. Specifically, we need to do a tradeoff between time delay and resolution in the frequency

domain.

For real-time filtering, a partitioned frequency-domain (PFD) convolver is used, which differs a bit when compared to the previously explained frequency-domain convolution. Instead of using a FFT algorithm on both the whole sequence of the source signal and the BRIR, multiplying and then using the inverse fast Fourier transform (IFFT), the BRIR is split into blocks, which are then subsequently run through the FFT. This newer version of the frequency-domain convolver is of interest due to the time complexity of the FFTs: $\mathcal{O}(n \log(n))$ [13], where n is the number of discrete points transformed. For the previous method, it would have to massively zero-pad the incoming samples according to Eq. 2.4. Instead, the BRIR are partitioned such that less zero-padding is needed, thus quickening the FFT.

The filtering method goes as follows. Offline, we create K partitions of our BRIR such that

$$\boldsymbol{w}^e = [\boldsymbol{w}_0^e \; \boldsymbol{w}_1^e \ldots \boldsymbol{w}_{K-1}^e],$$

where

$$\boldsymbol{w}_b^e = [w^e[bB] \; w^e[1 + bB] \; \ldots \; w^e[(B-1) + bB]],$$

where B can favourably be chosen as a power of two [13]. If needed, $\boldsymbol{w}^e$ will be zero-padded such that it is of a length divisible by B. Again, offline, we zero-pad each $\boldsymbol{w}_b^e$ to a length of $2 \cdot B$ as in eq. (2.4), Then we compute the FFT of each block

$$\boldsymbol{\mathcal{W}}_b^e = \mathrm{FFT}(\boldsymbol{w}_b^e).$$

As for the source signal $x[k]$, it will be sampled online in blocks of length B. This block $\boldsymbol{x}$ will be zero-padded to the length $2 \cdot B$. We compute the FFT for the newest block

$$\boldsymbol{\mathcal{X}}_0 = \mathrm{FFT}(\boldsymbol{x}),$$

and then store it in a buffer vector:

$$\boldsymbol{\mathcal{X}} = [\boldsymbol{\mathcal{X}}_0 \; \boldsymbol{\mathcal{X}}_1 \; \ldots \; \boldsymbol{\mathcal{X}}_{K-1}],$$

which contains the Kth latest $\boldsymbol{\mathcal{X}}_b$'s, where we move each block one step to the right and remove the last block. After this, we multiply each block piecewise as

$$\boldsymbol{\mathcal{Y}}^e = \sum_{i=0}^{K-1} \boldsymbol{\mathcal{X}}_i \boldsymbol{\mathcal{W}}_i^e.$$

We do the IFFT:

$$\boldsymbol{y}_b^e = [y_b^e[0] \; y_b^e[1] \; \ldots \; y_b^e[2(B-1)]] = \mathrm{IFFT}(\boldsymbol{\mathcal{Y}}^e).$$

Due to the zero-padding, $\boldsymbol{y}_b^e$ will have twice as many samples as we would like, so only the B first samples are kept for output. However, due to the partitioning, we need to implement overlap-add, where we save the B last samples from $\boldsymbol{y}_{b-1}^e$ and add on to $\boldsymbol{y}_b^e$. Then we can output $\boldsymbol{y}_b^e$.

3.3.4 Downsampling and Truncation

Since both downsampling and truncation lessen the number of samples we use to represent the BRIR, the two of them can be used in combination with either the PSVD model or the PFD convolver to speed up computation and reduce the storage required for the saved precomputed parts.

3.4 Complexity of methods

As a measure of the proposed method's performance, we can examine its multiply–accumulate (MAC) per sample, variables stored in memory, and sample lag. One MAC is defined as

$$x \leftarrow x + (y \cdot z).$$

Method	MAC/sample	Variables in memory	Latency
Convolution	$2n$	$4n$	0
PFD	$6\alpha \log_2(2B) + 8K + 2$	$8n$	B
PSVD	$R(N + 2P)$	$R(N + 2P + 2n) + N$	0

Table 3.1: Complexity of convolution methods for binaural rendering [4], where $n = \#$ of samples in filter, $B =$ block size, $K = \#$ of partitions, $R =$ rank used for approximation, N and P as in eq. (3.4). α is a constant depending on the FFT algorithm used.

In Table 3.1, we see that regarding MAC/sample, the PFD convolution method is by far the most efficient due to $K \ll n$ and that with a reasonable block size K and $2P$ will be close. PSVD comes in second due to N and P both roughly being the square root out of n. Memory-wise, the PFD convolution needs twice as much memory as linear convolution. We also see that the memory complexity of the PSVD heavily depends on the rank used, since if we neglect N and P, we are left with $R \cdot 2n$, which, depending on R, can be orders of magnitude larger than both other methods.

We see that each method has merit, the linear convolution uses less memory than the others, while the PFD convolution performs best MAC/sample-wise, and the PSVD has the fewest MAC/sample of the methods with 0 lag.

Downsampling and truncation can both be used to lower n, which in turn lowers K, N and P, which lessen the MAC/sample and variables in memory for all methods.

4

Results

4.1 Rank Approximation

Figure 4.1 showcases how the mean square error (MSE)

$$\frac{1}{n}\sum_{k=0}^{n-1}(y[k]-\hat{y}[k])^2$$

and MAC/sample depend on the rank used for PSVD. This is of interest since it showcases how the rank parameter characterises the PSVD approximation, where a larger rank gives a better approximation, while a smaller rank leads to stronger dimensionality reduction.

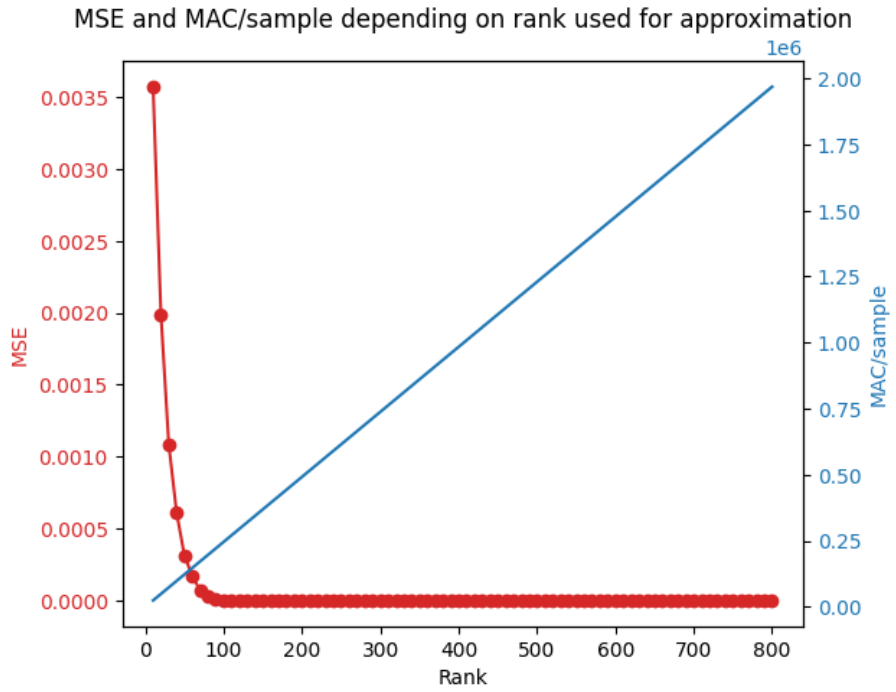


Figure 4.1: On the left axis: MSE between linear convolution and PSVD depending on rank, starting at rank 10, ending at 800, with increments of 10. On the right axis: MAC/sample depending on the rank used.

We can see in Fig. 4.1 that the MSE decreases very fast from rank 10 to rank 100, afterwards, the MSE stays more or less constant. The line flattening at approximately

100 suggests that most of the dominant structure in the signal is captured by relatively few components, while higher-order components add comparatively little to approximation accuracy. This indicates that a rank approximation somewhere around rank 100 should give a good trade-off between dimensionality reduction and approximation accuracy.

As expected, the MAC/sample grows linearly with the rank, as seen in Table 3.1. This means that rank 200 requires double the MAC operations of rank 100, while only giving a negligible improvement in the MSE. However, the MSE can be considered a flawed metric for listening experience [14], and thus a rank with considerably higher MSE than another rank may yield a similar listening experience.

In an effort to conduct qualitative evaluations of rank, the author has conducted a small listening test independently. The source file used is filtered either with linear convolution or with the PSVD method at rank R. The test is in the form of an ABX test [15], where one can play file A and B, which one know which is which, and then X and Y, which randomly is one of A or B, where one try to map which of A and B is X or Y. Rank 30 for the PSVD is compared to rank 10, 100 and linear convolution. Rank 30 was chosen as the baseline for comparison due to it having a relatively high MSE while performing well in the initial listening test. Rank 10 was chosen as the lowest rank visible in Fig. 4.1, and rank 100 as it looks like a good rank if going by MSE alone. The ABX test was performed using foobar2000's [16] ABX component [17], where a total of 20 trials were performed, allowing one to obtain a p-value using a binomial test.

Method	Number correct	p-value
PSVD Rank 10	20/20	0
PSVD Rank 100	10/20	0.59
Linear Convolution	11/20	0.41

Table 4.1: ABX testing of sound signals when compared to PSVD Rank 30.

The results in Table 4.1 correspond with the author's experience of the test, that is, the difference between rank 10 and rank 30 is most definitely noticeable. However, for rank 100 and linear convolution, the results suggest that the author was essentially guessing, which is reflected in the number of correct answers. This suggests that a rank of 30 is an appropriate rank to use for approximation.

4.2 Truncation

Figure 4.2 shows how the MSE compared to the original signal, and the filter length in seconds depend on the truncation parameter ε. It is of interest to investigate how the truncation used affects both the BRIR itself and the output.

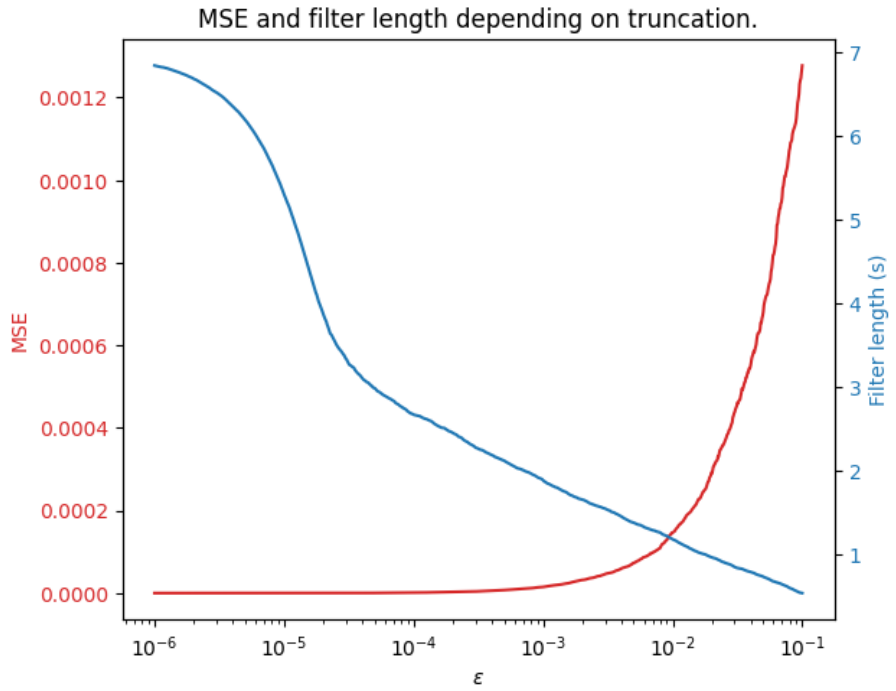


Figure 4.2: MSE between truncated BRIR and full BRIR (left-axis), and the resulting filter length in seconds (right-axis), as a function of truncation coefficient ε.

In Fig. 4.2, it can be seen that with a truncation smaller than 10^{-3}, the MSE mostly stays still, while the filter length has a sharp drop around 10^{-5}. This suggests that, numerically, the great majority of the BRIR is captured in the first two seconds, however, perceptually, this might not carry over.

Here, it is important to carefully take both MSE and filter length into account. Looking at the MSE by itself, a truncation of 10^{-3} might appear to be an optimal choice. However, while the BRIR will keep 99.9% of its energy, roughly $\frac{2}{7} \approx 29\%$ of the samples are kept, which will remove all of the later reverberation from the filtering process, which in turn perceptually could change the sound heard. A more careful truncation of $\varepsilon = 10^{-4}$ strikes a reasonable balance between filter length in terms of computational saving and listening experience, according to the author.

4.3 Comparison of Methods

In Table 4.2, it can be seen how the three different methods, when combined with different levels of sampling rate and truncation, compare to each other. For PSVD, a rank of 30 has been used, while for the PFD, a block size of 256 has been used. As a reminder, the source signal used is the one described in Section 3.1.2. The metrics compared are: time to convolve, storage necessary for precomputing, lag and real-time capabilities.

Method	f_s(kHz)	ε	T(s)	T_{norm}	Storage(kB)	S_{norm}	Latency(ms)	Real-Time
Convolution	48	0	251.9	1	5 251	1	0	No
Convolution	16	0	76.8	0.305	1 751	0.333	0	No
Convolution	48	$1 \cdot 10^{-4}$	211.5	0.840	2 004	0.382	0	No
Convolution	16	$1 \cdot 10^{-4}$	76.8	0.305	1 751	0.333	0	No
PFD	48	0	4.0	0.016	5 273	1.004	5.3	Yes
PFD	16	0	0.5	0.002	1 759	0.335	16	Yes
PFD	48	$1 \cdot 10^{-4}$	1.2	0.005	2 012	0.383	5.3	Yes
PFD	16	$1 \cdot 10^{-4}$	0.5	0.002	1 759	0.335	16	Yes
PSVD	48	0	77.7	0.308	386	0.074	0	No
PSVD	16	0	2.9	0.012	224	0.043	0	No
PSVD	48	$1 \cdot 10^{-4}$	49.1	0.195	239	0.046	0	No
PSVD	16	$1 \cdot 10^{-4}$	2.7	0.011	224	0.043	0	No

Table 4.2: Comparison between different filtering methods, where f_s indicates the sampling rate of the BRIR ε the truncation used. T is the average time it took to filter, Storage is the storage for the precomputed parts, Latency, if there is any intrinsic lag for real-time filtering and Real-time, if the method can filter with the BRIR in real-time. T_{norm} and S_{norm} are the time and storage relative to the first column.

It is important to note that even though all methods have been run 10 times for T, the results can vary vastly depending on the hardware used, even on a single PC, due to, for instance, background processes. This means that T should be considered with caution and is mainly used as a reference to other methods in the table. The convolution was calculated with an AMD Ryzen 5800X CPU. The time for filtering using PSVD using a different rank than 30 grows, as shown in Table 3.1, meaning that an approximation of rank 60 would take around twice as long. The PFD method by far outperforms the other two methods for a sampling rate of 48 kHz, however, for the lower sampling rate of 16 kHz, the PSVD closes the gap a lot.

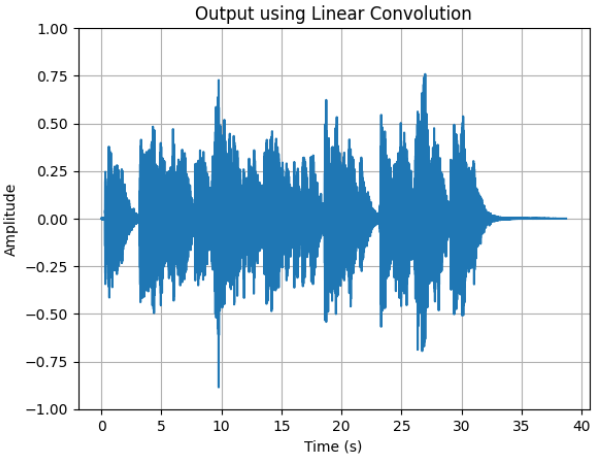
For storage, all precomputed parts were stored as .npy files, for the sake of speed and comparison. Also, note that the storage needed is computed for only one BRIR. Meaning that for the whole set, the total storage needed would be 90 times larger than in the table. The PFD method storage just being above the linear convolution is likely due to the zero padding needed to get $K \cdot B$ samples. We see that the PSVD method can save between 7 and 14 times the storage when compared to convolution, depending on the truncation and sampling rate.

Real-time performance was evaluated by implementing real-time filtering for each method. Samples were read either sample-wise or in blocks, depending on the method, and then the outputted. A method is seen as real-time if it can maintain this process without any output underflow (fewer samples ready for output than expected), while also not having regular noticeable artefacts. It should also be noted that real-time capabilities depend on hardware. All methods have been tested on both a PC with an AMD Ryzen 5800X CPU with 32 GB DDR4 RAM and a MacBook Air with an M1 chip and 8 GB memory.

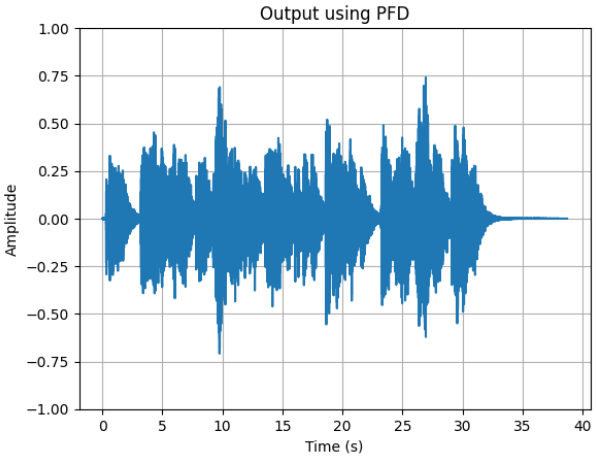
The latency introduced by the PFD is $\frac{B}{f_s}$ seconds, meaning it depends on the block size used. For example, a block size of $B = 256$ and a sampling rate of $f_s = 48$ kHz will introduce a latency of approximately 5.3 ms, which is a non-noticeable latency.

Figure 4.3 shows what the output for each method looks like in the time domain. The

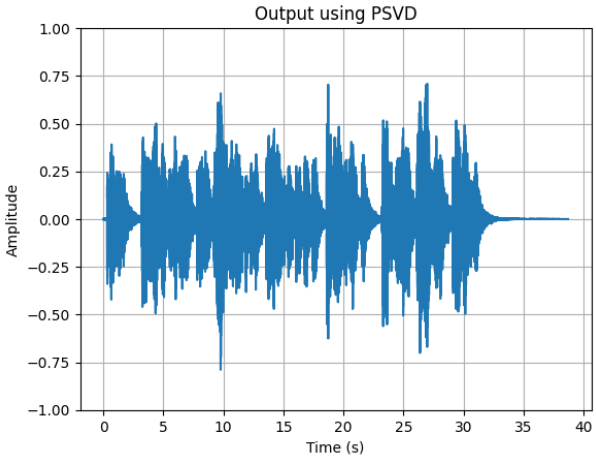
output signal in Fig. 4.3 generally looks quite similar for all methods, however, inspecting them closer, we can see some differences. For instance, around 10 seconds, both PFD and PSVD do not decrease sufficiently when compared to convolution. Figure 4.4 shows the spectrogram of each output in Fig. 4.3. On close inspection, small differences in the frequency content above roughly 2.5 kHz can be observed for the PSVD method, otherwise, the spectrograms appear very similar.



(a) Linear convolution.

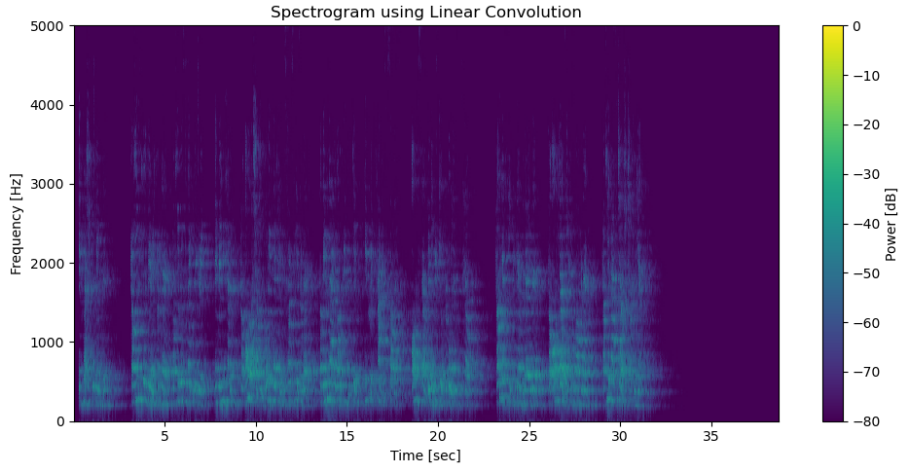


(b) The PFD method.

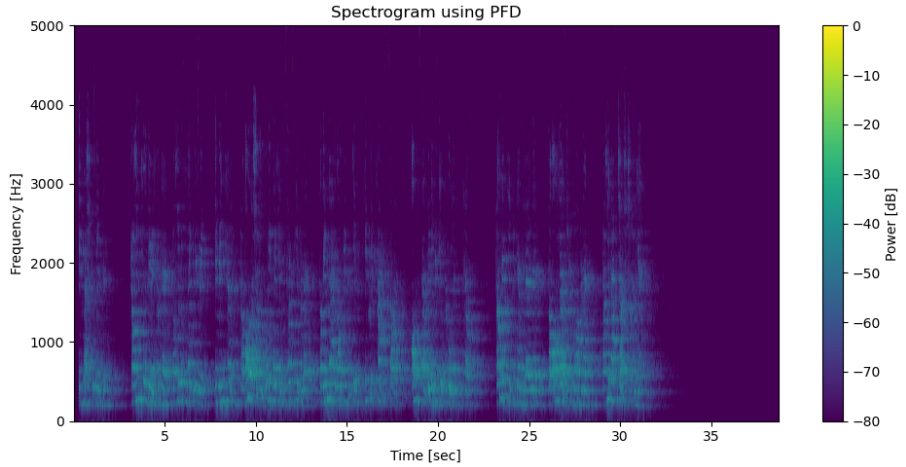


(c) The PSVD method.

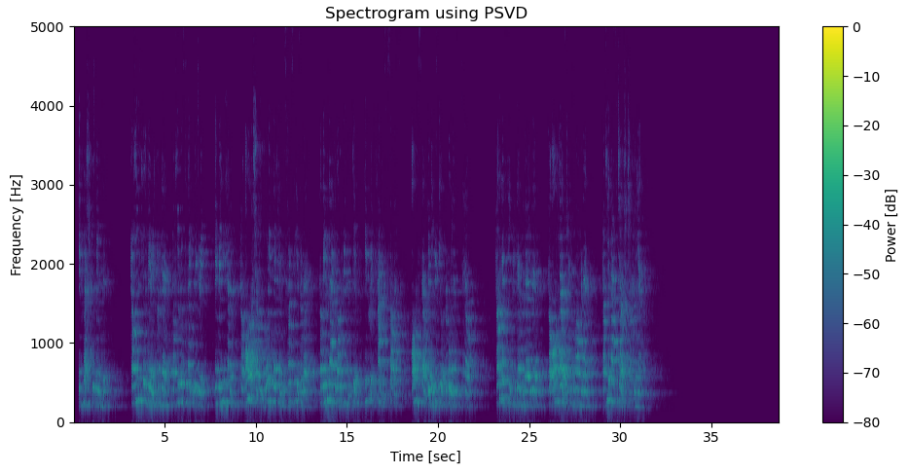
Figure 4.3: Output for the left channel using the three convolution methods.



(a) Linear convolution.



(b) The PFD method.



(c) The PSVD method.

Figure 4.4: Spectrogram for the left channel using the three convolution methods, shown up to 5 kHz.

5

Discussion

Comparing the three methods of linear convolution, PFD and PSVD, they all have advantages and disadvantages when compared to each other. Linear convolution's major advantage is its simplicity compared to the other two methods. If we can filter in real time using linear convolution and storage is not a major concern, then it is likely the most straightforward approach. However, if linear convolution is either too slow or saving on storage for precomputing is a major concern, the PSVD is the recommended approach, since it is both faster and uses less storage than linear convolution. Its downside is that it is an approximation, while it is likely a good approximation, it still is not a fully "true" filtering process. Also, in terms of perceived audio quality, there is no simple way to choose which rank to use for the approximation, where lower yields faster convolution and lower storage, and higher leads to better accuracy. Lastly, if neither linear convolution nor PSVD can filter fast enough for real-time, then PFD is likely to be the only viable option. Other than being fast (which often can be the deciding factor), the PFD method has no other obvious advantage over the other methods. Considering that it adds latency and requires the greatest amount of storage, if either of the other two methods works in real time, the PFD method serves no purpose.

Downsampling and truncation have been used in tandem with the other methods, where both reduce the number of samples in the BRIR with different strategies. Downsampling is equivalent to discarding higher frequencies, due to the Nyquist criterion, meaning we can only accurately represent frequencies in a signal that are less than or equal to half of the sampling rate. Due to this, depending on what kind of audio signal we are expecting to filter, one might want to keep the sampling rate at 48 kHz, for music for instance, while if we expect a speech signal, then downsampling to 16 kHz or lower is justified due to speech containing lower frequencies than music. As for truncation, this simply shortens the BRIR by cutting off at a point where computation is likely wasted on less important samples. This was done via the energy of the signal, however, one can do this in other ways to determine the cutoff point.

5.1 Selection of Rank

To aid in the selection of the rank for the PSVD method, an ABX test was conducted by the author. While the results indicate that the author could not reliably distinguish between rank 30 and linear convolution, a more rigorous evaluation would require a

substantially larger sample size and cross-testing between multiple ranks. Nevertheless, both the conducted test and Table 4.2 suggest that rank 30 provides a reasonable balance between audio quality, computational speed, and storage requirements.

If computational speed and/or storage are the primary concern, the author's subjective experience is that rank 20 approximation appears to be the lowest acceptable choice, below this, the spatial characteristics are largely lost. Conversely, if the rank 30 approximation leaves sufficient computational and storage headroom, the rank could simply be increased, since, at least theoretically, a higher rank should always improve the listening experience, even if the improvement is only marginally perceptible.

Choosing the final rank to use for approximation is not as straightforward as one might wish. While both Fig. 3.4 and Fig. 4.1 show very similar results, that is, diminishing results around rank 100, this does not necessarily translate into listening experience. The MSE is indeed a good metric to measure how similar two audio signals are numerically, however, measuring how alike two audio signals are to a human is hard. Therefore, even if one rank approximation has double the MSE of another rank, it might not be noticeable in a listening test. For instance, the author had a hard time telling the difference between the use of a rank 30 approximation and a rank 100 approximation, even though the former has much higher MSE than the latter. This led to an approach of choosing the rank, where Fig. 4.1 was used as a rough guideline for which interval to manually look for the rank that would give the best balance of speed and listening quality.

5.2 Avenues for improvements

One improvement on the PSVD method that the author had hoped to implement is a low-rank tensor approximation, as outlined in [5]. For instance, if one has a 3-dimensional low-rank approximation of a BRIR, the MAC/sample can be reduced to $2R(n_1 + n_2 + n_3)$, lowering the computional complexity when compared to the PSVD method, where R is the rank of the tensor, which is of shape $n_1 \times n_2 \times n_3$ and $n_1 \cdot n_2 \cdot n_3 \approx n_w$. However, this proved much trickier than a low-rank approximation of a matrix, mainly because finding a low-rank approximation tensor of the BRIR is less well posed than for a matrix, unlike the matrix SVD, which provides a unique and optimal low-rank approximation in the Frobenius norm. For tensors, there exists a multitude of methods, which produce non-unique approximations and are hard to implement. Due to this, more focus was put on other methods, however, low-rank approximations remain a worthwhile endeavour.

This project has been entirely programmed in Python. Why this is relevant is that Python is notoriously slow and should often only be used for analysis. While using libraries such as NumPy, which do some calculations in C, helps speed programs up a lot, some things were still written in raw Python. For serious implementation, the program used for real-time filtering should be written in a more efficient language, for example, C/C++. The program would likely also benefit from parallelisation, where we let the program do many calculations in parallel, such as multiple blocks at the same time. However, this was not implemented, as this was not the focus of the thesis. The PSVD method would most likely be the method that would benefit the most from this.

While the PFD convolver works in real-time, it can be optimised even further. We-

fers and Vroländer [18] aimed to optimise the partition size of the BRIR, where instead of using a uniform partition size, they allow for nonuniform partition sizes. These partition sizes are optimised to minimise the arithmetic operations per sample. Since the proposed PFD convolver worked in real time in the author's testing, this was not implemented, however, this is an obvious next step for improving the filtering.

6

Conclusion

This thesis has sought to test and investigate the possibility of real-time binaural rendering of long room impulse responses (RIRs) measured in a church environment. Three different representations of RIRs have been presented and used: linear convolution, partitioned frequency-domain (PFD) convolution, and a model utilising the low-rank structure of the binaural room impulse response (BRIR), which we called the partitioned singular value decomposition (PSVD) model. On top of this, downsampling and truncation have been used as a simple way to lessen the number of samples in the BRIRs.

The results show that the PFD method is the only one to accomplish real-time processing, where both the linear convolution method and the PSVD methods fail. However, we have also seen that the PSVD provides a considerable reduction in storage needed for the precomputed part when compared to the other two methods, while still being considerably faster than linear convolution.

Evaluating the best rank for the PSVD method proves to be a non-trivial choice. As seen, using a standard numerical metric for closeness of two signals, the mean square error (MSE), does not necessarily translate to a measure of perceived closeness. To counteract this, a small listening test was conducted, which indicated that a much lower rank than what the MSE would suggest was the best choice. This emphasises the difference between numerical evaluations and what is actually perceivable.

With this, we can say that the filtering method chosen for a problem can depend on many things. If computational speed is by far the highest priority, then the PFD convolver is the safest way to go. However, if speed is not as important as saving on storage, the low-rank method is the best way to go, since it blends speed with a small storage footprint. Linear convolution is slow but simple, so it might be useful for offline prototyping.

For future work, the main goal would be to build on top of the low-rank method. For example, fully implementing a tensor-based approach. Implementing the filtering process in a low-level programming language is also a likely next step to eliminate the Python bottleneck. The nonuniform PFD is also a branch one could take to further optimise the PFD.

To close, it is satisfactory that at least one of the methods achieved real-time filtering. However, ideally, it would come with zero intrinsic latency.

Bibliography

[1] Eric Weisstein. Convolution theorem. From MathWorld–A Wolfram Web Resource. `https://mathworld.wolfram.com/ConvolutionTheorem.html`. Accessed: 2026-04-27.

[2] Eric Battenberg and Rimas Avižienis. Implementing real-time partitioned convolution algorithms on conventional operating systems. In *Proceedings of the 14th International Conference on Digital Audio Effects*, Paris, France, September 2011.

[3] Sanjit K. Mitra, M. D. Grosen, and Yrjö Neuvo. Design of computationally efficient fir filters using singular value decomposition. In *Proceedings of the China International Conference on Circuits and Systems*, pages 268–271, Beijing, China, 1985.

[4] Joshua Atkins, Adam Strauss, and Chen Zhang. Approximate convolution using partitioned truncated singular value decomposition filtering for binaural rendering. In *Proceedings of Meetings on Acoustics*, volume 19. Acoustical Society of America, 2013.

[5] Martin Jälmby. *Low-rank Modeling in Room Acoustics*. Phd dissertation, KU Leuven, Leuven, Belgium, March 2024.

[6] Brandenburg Labs. Introduction to head-related transfer function (HRTF). Brandenburg Labs Tech Blog, October 2025. https://brandenburg-labs.com/introduction-to-head-related-transfer-function-hrtf/. Accessed: 2026-05-08.

[7] Daniel P. Jarrett, Emanuël A. P. Habets, and Patrick A. Naylor. *Theory and Applications of Spherical Microphone Array Processing*, volume 9 of *Springer Topics in Signal Processing*. Springer International Publishing, 2017.

[8] Riccardo Giampiccolo, Sofia Parrinelli, and Fabio Antonacci. Churchir: A data set of multichannel church impulse responses for spatial audio applications. In *Proceedings of the 33rd European Signal Processing Conference (EUSIPCO)*, pages 161–165, Isola delle Femmine, Italy, September 2025. IEEE.

[9] Acoustics Research Institute, Austrian Academy of Sciences. Ari hrtf database (sofa format), 2024. `https://sofacoustics.org/data/database/ari/`. Accessed: 2026-03-25.

[10] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[11] Anders Holst and Victor Ufnarovski. *Matrix Theory*. Studentlitteratur AB, 2014.

[12] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[13] James W. Cooley and John W. Tukey. An algorithm for the machine computation of the complex fourier series. *Mathematics of Computation*, 19:297–301, April 1965.

[14] Pranay Manocha, Zeyu Jin, and Adam Finkelstein. Audio similarity is unreliable as a proxy for audio quality. *arXiv preprint arXiv:2206.13411*, 2022.

[15] W. A. Munson and Mark B. Gardner. Standardizing auditory tests. *The Journal of the Acoustical Society of America*, 22(5):675, 1950.

[16] foobar2000. `https://www.foobar2000.org/`. Accessed: 2026-05-07.

[17] Peter. ABX Comparator (foo_abx). foobar2000 Components Repository, September 2025. `https://www.foobar2000.org/components/view/foo_abx`. Accessed: 2026-05-07.

[18] Frank Wefers and Michael Vorländer. Optimal filter partitions for real-time fir filtering using uniformly-partitioned fft-based convolution in the frequency domain. In *Proceedings of the 14th International Conference on Digital Audio Effects*, pages 155–161, Paris, France, September 2011.

Disclosure of AI Use

In this thesis, AI tools such as Grammarly and ChatGPT have been used for enhancing the quality of the author's written text and as a coding tool. Specifically for coding, ChatGPT has helped with the creation of the graphical user interface (GUI), while all other coding, such as computation and plotting, has been hand-coded with minimal help from ChatGPT, since this is a part of the process that the author enjoys.

7

Appendix

A

In real-time processing, there is no access to future samples as there is for offline processing, this has a slight effect on resampling and normalisation of output samples.

A.1

While filtering offline, it is easy to resample the entire source signal to align with the BRIR. Online, this will not work, since we only have access to the most current block (and the old ones if saved).

How sampling is done is by setting a set time window to gather multiple samples: $\frac{B}{f_{\text{BRIR}}}$ s, where B is the block size and f_{BRIR} is the sampling rate of the BRIR. If the source sampling rate: f_s equals f_{BRIR}, we will have sampled B samples in the time window, and no resampling is needed. However, if $f_s \neq f_{\text{BRIR}}$ we will either have fewer or more than B samples. To fix this, we resample the current block to be a block of B samples.

A.2

To ensure that the output signal stays in the range [-1,1] in real time without clipping, a dynamic gain is applied per block. Unlike offline normalisation, this does not need any of the future samples.

Let the gain g, ceiling $c \leq 1$ and boost $b > 1$ be initialised as

$$g = 1, \quad c = 0.9, \quad b \approx 1.$$

For each output block (y^L, y^R) we find the peak p as in Eq. (3.2). Then we update the gain as

$$g = \begin{cases} \frac{c}{p} & \text{if } p \cdot g > c \\ \min(g \cdot b, 1) & \text{otherwise} \end{cases}.$$

Finally, we apply the gain to both channels

$$y^L[k] \leftarrow g \cdot y^L[k]$$
$$y^R[k] \leftarrow g \cdot y^R[k],$$

such that $(y^L[k], y^R[k]) \in [-1, 1]$, which is the final output.

B

A graphical user interface (GUI) for real-time filtering has been created. The GUI allows for filtering of audio files such as wav, flac and mp3, it can also filter the audio from one's system. During filtering, we can change which BRIR to use in real time.

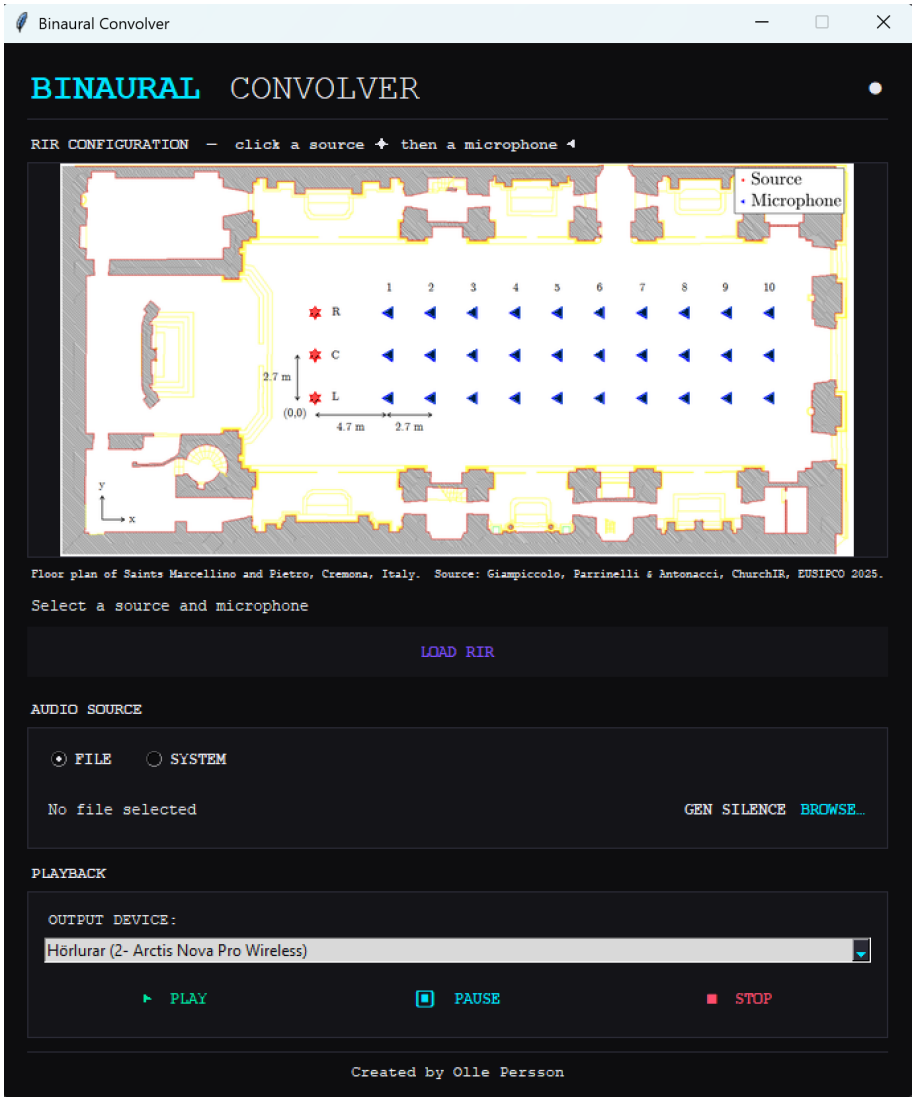


Figure 7.1: Screenshot of GUI.

More information about the GUI, and how to use it can be found on the author's GitHub[1].

[1] https://github.com/ollep333/Master-Thesis