OMNI: A HYBRID ENVIRONMENT FOR LIVE GRANULAR AND SPATIAL PROCESSING

Jason Hoopes
Northeastern University
Boston, MA
hoopes.j@northeastern.edu

ABSTRACT

Granular synthesis is a powerful audio processing and synthesis tool used extensively by contemporary sound designers and musicians to transform, re-imagine, and re-contextualize sound sources. While many granular synthesis tools may pan grains into stereo space, they neglect the opportunity to use more advanced binaural spatialization techniques to position individual grains in virtual 3D space. I introduce Omni, a software environment for live granular processing of a input, in conjunction with grain-dependent frequency based processing for interactive immersive live performances. Omni's hybrid processing engine has applications in live musical performance, audio-visual art, film, and gaming.

1. INTRODUCTION

Since its artistic debut in Barry Traux's *Riverrun* [1] over 30 years ago, granular-based processing techniques have undergone an drastic evolution. Generally, granular processing refers generally to any form of audio processing or synthesis that includes breaking up a waveform into small chunks, called grains, and repositioning, overlapping, and re-synthesizing them into a new waveform. While historically, granular processing is most commonly performed in the context of synthesis, on a fixed buffer, recent effects processors like the Hologram Microcosm, leverage granulation of a live signal for glitchy, ambient reverberations and delays. Due to the *temporal* and *spectral* transformations that are possible with this method, it is favored by sound designers, performers, and producers for its large range of sonic possibilities, from immersive organic textures, to rich pads, to choppy, percussive fills. However, with Omni, I propose the use of modern spatial rendering techniques, to explore space as a third mode of transformation that can be leveraged with granulation.

2. SPATIALIZATION

2.1 Background

The recent increasing popularity of immersive media, such as Virtual and Augmented reality devices, has encouraged a rise in the practical applications of *virtual audio spatialization*, or the rendering of a virtual sound source in a simulated 3D space. The goal of spatialization is for the listener to be able to percieve sound playback from a specific position around them, without needing to surround the listener with loudspeakers. While spatial rendering techniques can include varying types and amounts of sound sources, Omni's spatial exploration mainly targets rendering for headphones. However, before addressing Omni's implementation of spatialization, it is important to first understand the physiological and anatomical concepts that inform it.

2.2 Psychoacoustics of Localization

Lord Rayleigh's exploratory 1907 "duplex theory"[2] provided the foundations for two main *binaural* (meaning both ears) cues that allow the human brain the ability to discern a sound source's position of origin. Rayleigh's work, bolstered by the findings of following researchers, showed that the human ear is able to accurately decipher extremely small temporal variances in pressure between ears, identifiying two main binaural cues: Interaural Time Differences and Interaural Intensity (Sound Pressure Level) Differences [3].

Interaural time difference (ITD) refers to the small difference in time of arrival between ears, as a sound wave must travel a longer distance to reach to the further ear. This difference is highly dependent on the sound source's azimuth - the horizontal angle, relative to the position directly in front of the listener's head (see Figure 1). For example, A sound source at an azimuth of 0, would experience an ITD near or equal to 0, since each ear is equidistant from the source. Likewise, a scenario with an azimuth of 90 would produce the maximal difference, since the source is distance d closer to the listener, where d is the width of the listeners head. While the temporal information given by the ITD is important, it cannot inform accurate localization alone, since the ITD is not unique to each azimuth.

Interaural Intensity difference (IID) refers to the difference in wave energy between each ear, due to the obstruction of wave propagation by the physical anatomy of the listeners head. Again, in a scenario with a sound source at an azimuth of 90 deg, the produced wave would reach the right ear in a direct path, while the wave would only reach the right ear via reflection and diffraction around the head, resulting in frequency-dependent transformations in intensity. Because of this complex wave behavior, the IID

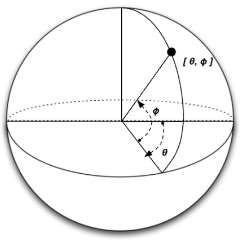


Figure 1. Angular Azimuth and Elevation, from NYU [4]

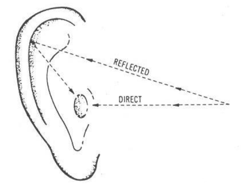


Figure 2. Wave interference from pinnae reflection [5]

is heavily dependent not only on a sound source's azimuth, but also its elevation (Figure 1) and frequency content.

Further work by Wightman and Kistler [6] and others [3] showed that localization is also informed, arguably most importantly, by *monaural* (single ear) cues. These cues refer to the spectral transformations that result from wave interference produced by the many reflections inside the complex folds of the *pinnae* (outer ear) before reaching the inner ear (Figure 2). The human brain learns the effect of these in-ear reflections, as a frequency-to-position transformation, referred to as the *Head-Related Transfer Function*. [3]

As proposed by Zotkin Et Al., this frequency-domain transfer function $H_{l,r}$ (for left and right ears, independently), can be formally defined as the "frequency-dependent ratio of the sound pressure level (SPL) at the corresponding eardrum $\Phi_{l,r}$, to the free-field SPL at the center of the head as if the listener were absent Φ_f". [7]

$$H_{l,r} = \frac{\Phi_{l,r}(\omega, \varphi, \theta)}{\Phi_f(\omega)} \tag{1}$$

2.3 Measurement and Reproduction

To record and measure the effect of pinnae anatomy, azimuth, and elevation on the frequency-domain HRTF, a microphone can be placed inside the ear canal, to most closely replicate the outer ear filtering that occurs. While initial measurements used a probe microphone placed inside a subject's ear canal [8], modern setups use a series of custom anatomic models attached to a dummy head, molded specifically to represent a variety of pinnae anatomy [9].

Loudspeakers are then placed around the listener in a spherical formation, where each loudspeaker is a consistent radius r away from the center of the head. Each loudspeaker position represents a discrete point (a, e) in the spherical coordinate system, with azimuth a, and elevation e (see Figure 3). In order to avoid the effect of room reflections that are not related to physical anatomy, this recording is usually done in an *anechoic chamber*, an acoustically treated space that prevents any wave reflections on the walls, ceilings, or other supporting surfaces. From each of these loudspeakers, an impulse response is generated, representing the time-domain analog the HRTF, the Head-Related Impulse Response (HRIR).

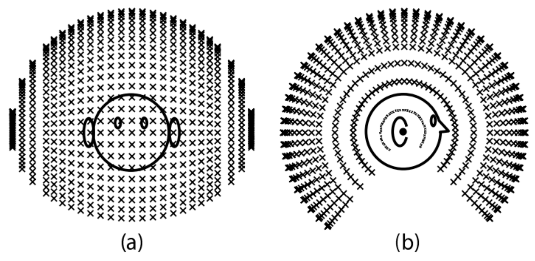


Figure 3. Spherical positions of loudspeaker placements for HRIR recording, from CIPIC [10]

From this time-domain representation of the ear response for a sound source at (a, e), we can obtain both the difference in time of arrival (ITD), and the change in frequency response via Fourier Analysis (see Figure 4). It is important to note here, that since the recorded HRIR models every frequency-based transformation that happens between the sound source and the ear canal (including those resulting from head-related obstruction), the information needed for IID cues is implicitly encoded within the HRTF and HRIR. This means that, in order to achieve an accurate reproduction of the localization cues, we do not need to separately measure or implement IID information.

2.3.1 Reproduction

To reproduce the measured head-related transformations for any mono input signal x, placed at a virtual spherical position (a, e), we convolve it with the corresponding measured HRIR for that position, for each ear, to generate a spatialized, binaural signal y.

$$y_l = x \circledast IR_l \tag{2}$$
$$y_r = x \circledast IR_r \tag{3}$$

This per-ear filtering, will allow the headphone-wearing listener to perceive the signal x as if it was coming from position (a, e). Since the above filtering spatializes a signal as a *single* sound source, it is crucial that the signal is single-channel, with no pre-existing positional information.

2.4 Software Implementation

The proposed Omni environment provides a *BinauralSpatializer* interface for fast, real-time spatialization using the above methodology.

2.4.1 Fast Convolution

In a real-time context, performing convolution in the time-domain is slow and computationally expensive. An implementation of circular convolution defined in (2) would

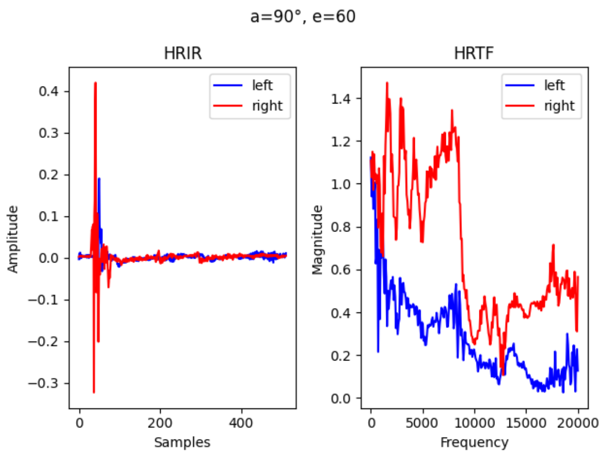


Figure 4. The HRIR and HRTF for azimuth=90, and elevation=60

have time complexity $O(N^2)$. This complexity can be improved by instead implementing the filtering as frequency-domain multiplication, which as stated by the *convolution theorem*, is equivalent to time-domain convolution [11]. In this case, the complexity bottleneck is the calculation of the discrete Fourier Transform (DFT) which has time complexity $O(NlogN)$. In Omni, the frequency-domain head-related filter data is pre-calculated and resampled so that the number of bins is consistent with the bin size of the input signal's DFT. This allows each call to render a spatialized block of audio, only requires a single DFT on the input block, and an inverse DFT on the filtered frequency-domain signal.

By default, Omni uses an FFT size of 512, and uses the overlap-add method for reconstructing the time-domain signal [11]. When the FFT size is small, less latency is added, which increases the playability and interactivity of the system for the performer. However, this also means that the filter has less resolution, which affects the perceptual success of the spatialization. An FFT size of 512 was chosen to balance these considerations, by prioritizing the listener's experiences, and taking a small latency hit.

To perform fast, optimized discrete Fourier Transforms, Omni utilizes the cross-platform FFTW3 library [12].

2.4.2 Data handling

On instantiation, a *BinauralSpatializer* loads data from a csv file containing HRTF information for 187 spherical positions. This data is generated in a Python script, which loads recorded HRIRs from a single subject in the NYU MARL database [4], performs a Fourier Transform on each impulse response, and coallesces the data into CSV file, in a format which can be read by Omni. The data is loaded into a Hashed map, where the key is the azimuth and elevation pair (a, e), and the value is the corresponding ITD value, and HRTF. This allows the HRTF and ITD information to be quickly retrieved in O(1) time. By storing all of the filter data in Memory, we avoid having to load the data from the file system every time the rendering position changes. This operation would not be efficient enough for our real-time use case, and would require a separate high-priority thread running simultaneously behind the audio thread. The loaded data is shared between any number of Spatializer instances that are created, avoiding unnecessary memory consumption.

2.5 Accuracy

One of the biggest issues with the HRTF method, is that its perceived success depends on psychoacoustic principles that are heavily individualized to each person's physical anatomy.

Everyone's ear is different, as is the way that the reflections from their ear anatomy will filter the found. This means that one listener's learned frequency change for a certain position, will differ from what others might hear from the same position. Using a single, generalized head related transfer function for everyone means that it will be much more accurate for some people and a lot less accurate for others. There are proposed solutions to this issue of individualisation with varying success, like using transfer learning techniques [13] to generalize a few small measurements from a subject, or Computer Vision to estimate a listener's HRTF using a camera [14]. However, these techniques are overly complex for Omni, which exists as a proof of concept for the relationship between granular and spatial processing. So, for now, Omni will use a generalized head related transfer function, and assume that there will be some margin of error for a percentage of the listeners. Future improvements to Omni would prioritize a more robust solution to this problem of individualization.

2.6 Use in live performance

Another downfall of headphone-based spatialization, is the difficulty of its practical extension to a live performance environment. Due to the proximity of the speaker driver to the ear canal, headphones lead to the most accurate, and personalized reproduction of binaural and monaural localization cues. However, in many applications, especially live performance, headphone listening is not always practical, or even possible. While headphone-based live performances do exist, the technology and materials required to equip an audience with a personal headphone mix, pose a physical and logistical challenge [15]. Additionally, if audio processed for headphone reproduction with the above techniques were played through a traditional two-channel loudspeaker system, the sound waves would undergo many reflections in the room before reaching a listener. Any positional information encoded in the signal would be completely lost by the time it reached a listener's ear canal.

In order to create an live performance experience that is perceptually similar within a reasonable margin, Omni includes an additional *LoudspeakerSpatializer* class for a 2D, circular array of n loudspeakers, arranged around the listener. Like the *HRTF*-based binaural spatializer, the loudspeaker spatializer is not capable of rendering sounds at a specific distance, only at a specific azimuth around the circle of loudspeakers. The Loudspeaker Spatializer uses a basic panning algorithm, dividing the circle into $n - 1$

quadrants, and given angle to reproduce, calculates a scalar gain value for each speaker in the array.

3. GRANULAR PROCESSING

3.1 Real-Time Implementation

To perform granular processing in real-time, Omni maintains a circular buffer, which is incrementally filled with subsequent blocks of an input audio source, at the position of a sliding write pointer. Grains are then extracted from this buffer, by a series of read pointers, whose behavior is controlled by 3 main parameters, grain length, playback speed, and grain position. For a grain with length N, a window function (such as the Hanning window) of length N, is applied to the grain via element-wise multiplication, to smooth out the edges of the grain. This is crucial, because when playing back two grains from different locations consecutively, an amplitude discontinuity will be created along the edges of each grain in the output waveform, causing clicking artefacts. By smoothing these edges with a window, a zero-crossing is created at each edge, avoiding this wave discontinuity.

3.2 Grain Spatialization

After grains are extracted and windowed, they are traditionally summed into mono or stereo signals and shipped to the output device. However, Omni collects the output of each grain, and sends them individually to separate post-processing objects, where they can be rendered at a specific position in space by the *Spatializer* object referenced in 2.4. The relationship between a grain and a *Spatializer* is many-to-many. One grain can be sent to multiple *Spatializers*, and one *Spatializer* can receive multiple grains. Both the number of grains and the number of *Spatializers* are configurable based on the needs of the user. The architecture of this signal chain is visualized in Figure 5. This method of independent processing of each extracted grain does not currently exist in any similar tool available today.

3.3 Embracing Musicality

A significant contributor to the archetypal 'sound' of granular synthesis is the mapping of controlled randomness to the main parameters, like length, position, and playback speed. While a wide range of interesting, sonic material can be generated from this randomness, testing of early Omni prototypes during development found that relying primarily on stochasticity neglected a crucial characteristic of live performance - musicality. Since the main goal behind Omni was to augment the performance of (specifically acoustic) instruments, I decided to investigate ways to interact and control with the granular processing in a more musical way.

3.3.1 Transient Warping

Traditionally, the start position of the grains move freely around the buffer, driven only by manually set parameters and randomness, and oblivious of the audio content in the buffer itself. In many explorative tests, I often desired the behavior of the grain pointers to move dynamically with my playing, deliberately accentuating and augmenting certain harmonic and rhythmic choices. To achieve this, A "warping" mechanism was added that *pulls* the start of each grain towards location of transients that are detected in the buffer.

The user can control, in real-time, the number of the most-recent transients that a grain will be pulled towards. When this value is 1, all grains will latch on to the most recent transient. When this value is greater than 1, each grain will latch on to a random transient within the current range, allowing performers to re-sequence melodies and rhythmic patterns, or even hold create long sustained polyphony with a single monophonic instrument.

To detect transients, I employ a modified version of a basic method described by Gueorguieff [16] that involves calculating the root mean square energy of consecutive 5 millisecond blocks. The function to determine if the current block contains a transient can be defined as follows, for the rms energy at time e_t, and a user-defined threshold α:

$$(e_{t-1} > e_{t-2}) \wedge (e_t > e_{t-2} \cdot \alpha) \qquad (4)$$

3.3.2 Grain Pitching and Harmonization

Nuanced re-pitch control allows the playback speed to be independently controlled for each grain, which is especially helpful for using Omni with a monophonic instrument. Each grain pitch can be manually controlled, or quantized to a series of scales and chords. When combined with other parameters like Transient warping, this allows a instrumentalist harmonize with themselves in real-time, or create generative, evolving harmonic scenes.

3.3.3 Complex Buffering

Because of the unpredictability of many aspects of granulation, it is common that the grains land on a certain sonic moment that a performer would like to build on, or further transform. However, due to the rotating nature of the circular buffer, the samples at the portion of the buffer may be overwritten within seconds, destroying the sonic moment. To enable this exploration without fear of overwriting, users can "freeze" a buffer, to momentarily disable the write pointer, and indefinitely explore the buffer in its frozen state. Additionally, a *Hybrid Buffering* approach allows the user to fill an additional m buffers with audio data from other input sources, sound files, or historical data from the main buffer. This allows a user to morph between different sound sources / timbres / or moments in time. The number of buffers allowed is configurable, and constrained mostly by the allowed maximum memory consumption by the user. When using M buffers of N samples each, the hybrid buffering will introduce $M * N * 4$ bytes of memory consumption.

4. APPLICATION

4.1 A Portable Framework

In its current state, Omni is realized, and usable as a standalone MacOS application. The popular c++ framework

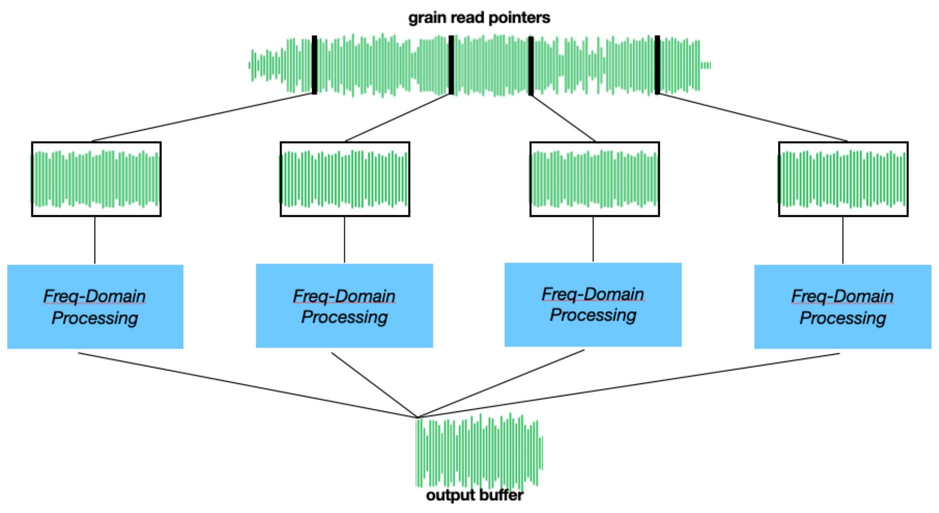
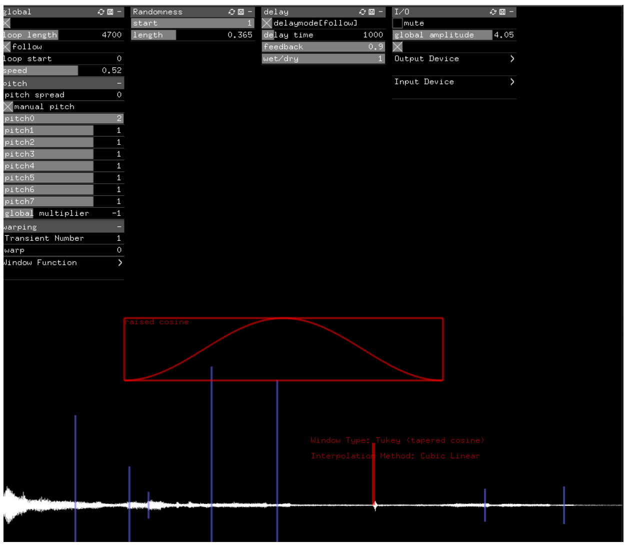


Figure 5. Visualization of the Omni granular spatialization chain

openFrameworks [17] is used as main engine, providing a light wrapper around the native MacOS Cocoa Application Layer, and providing easy access to quick GUI elements like sliders, and buttons, efficient openGL graphics rendering, and most importantly, a native, high priority audio thread. However, throughout the ideation and development, I envisioned the final product as being decoupled from a computer/screen based environment, and existing as a standalone, physical device. A large emphasis was placed on ensuring that the main code was not tied to a specific framework, or programming environment. As a result, the entirety of Omni is wrapped within a custom-built audio engine with a simple API. The only dependency is the cross-platform FFTW3 library for DFT calculation [12], all other audio code is built from scratch in portable C++. This way, I can easily create a port of the code that can run in a plugin, embedded device, or mobile phone.



```
global                    Randomness              delay                     I/O
loop length     4700      start          1        delaymode[follow]         mute
follow                    length     0.365        delay time     1000       global amplitude  4.05
loop start         0                              feedback        0.9
speed           0.52                              wet/dry           1       Output Device  >
pitch              -
pitch spread       0                                                        Input Device   >
manual pitch
pitch0             2
pitch1             1
pitch2             1
pitch3             1
pitch4             1
pitch5             1
pitch6             1
pitch7             1
global multiplier -1
warping            -
Transient Number   1
warp               0
Window Function    >

raised cosine

Window Type: Tukey (tapered cosine)
Interpolation Method: Cubic Linear
```

Figure 6. A mid-developement debug gui in OFX

4.2 Parameter Control

While the current openFrameworks-based GUI is helpful for prototyping new features and debugging, it is not very useful in a performance context. It acts nothing like an instrument - It is filled with small text and technical logs, and any attempt at interacting with Omni at all through this interface, forces a user to shift their focus to the laptop, incrementing finicky sliders with the trackpad 6. Especially in a performance-oriented environment, shifting focus away from the music for even this short amount of time is not conducive to successful, fluid performances, neither for the performer, or the audience. To address this consideration, while still providing flexibility of control mapping, every configurable parameter can be set via through OSC, by sending a message to the corresponding channel name for each parameter, following the channel naming convention ”Omni/parameter_name”. By using OSC, the control methods are completely decoupled from the audio processing environment, and can even be remapped while Omni is running. The generality and flexibility of OSC means there is no limit to the kinds of control surfaces that can be used to interact with Omni. For the final performance and demo, I will be controlling Omni with a series of midi knobs, sliders, and expression pedals, by sending OSC messages from a PureData patch.

5. NEXT STEPS

While the current state of Omni provides an excellent basis for the exploration of spatial granulation, there are a series of improvements and extensions that would increase the success and artistic possibilites of the system.

5.0.1 Interpolation

The first, and most important, is the inclusion of a real-time HRTF interpolation algorithm. Currently, at the time of writing this paper, Omni does not include a working interpolation for the HRTFs that are sued in the *BinauralSpatializer* object.

As mentioned in section 2.4, the recorded data models the transfer function at specific spherical coordinates, which means the *Spatializer* is only able to render sounds at these discrete points in space. However, to effectively create a realistic auditory environment, it is important that sounds are able to be smoothly rendered in continuous positions around a listener. If a sound is jumping around in large distances, it will will create a choppy effect that will ruin the perceived realism of the localization. Because granulation is already "choppy" in nature, the decrease in success is less pronounced than an application where pure realism or simulation is the goal, such as Virtual or Augmented reality scenes. Yet, it is still important that a HRTF reproduction algorithm is able to quickly, accurately, and continuously interpolate between the transfer function for any spherical coordinate (a, e).

5.0.2 Additional Frequency Processing Techniques

The use of grain-independent processing in this framework has proven to be a useful tool in pushing the transformation abilities of granulation. In the future, it would interesting to examine how additional advanced frequency domain processing techniques, like physical modeling, or even autoencoder-based timbre transfer [18], would perform in this framework. Luckily, due to the architecture of Omni, it would be relatively trivial to extend the functionality to include these methods.

5.0.3 Interface design

Lastly, this processing environment would benefit largely from a dedicated, thoughtful interface that allows a user to seamlessly interact with and control the system performing. This would allow Omni to be ported into a variety of other formats, like a VST/AU plugin for DAW use, or a embedded program in the form of a rack-based processor for use with other modular synthesizers.

6. REFERENCES

[1] B. Traux, "Riverrun," 1986.

[2] L. Rayleigh, "Xii. on our perception of sound direction," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 13, no. 74, pp. 214–232, 1907.

[3] D. Wang and G. J. Brown, *Computational auditory scene analysis: Principles, algorithms, and applications*. IEEE Press, 2006.

[4] A. Andreopoulou and A. Roginska, "Documentation for the marl-nyu file format description of the hrir repository," 2011.

[5] M. Drolet, "Acoustics and psychacoustics," https://jac.michaeldrolet.net/acoustics/acoustics.htm, accessed: 2024-04-20.

[6] F. L. Wightman and D. J. Kistler, "Monaural sound localization revisited," *The Journal of the Acoustical Society of America*, vol. 101, no. 2, pp. 1050–1063, 02 1997. [Online]. Available: https://doi.org/10.1121/1.418029

[7] D. Zotkin, R. Duraiswami, and L. Davis, "Rendering localized spatial audio in a virtual auditory space," *IEEE Transactions on Multimedia*, vol. 6, no. 4, pp. 553–564, 2004.

[8] H. Møller, M. F. Sørensen, D. Hammershøi, and C. B. Jensen, "Head-related transfer functions of human subjects," *J. Audio Eng. Soc*, vol. 43, no. 5, pp. 300–321, 1995. [Online]. Available: https://www.aes.org/e-lib/browse.cfm?elib=7949

[9] A. Vidal, P. Herzog, C. Lambourg, and J. Chatron, "Hrtf measurements of five dummy heads at two distances," in *2021 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*, 2021, pp. 1–8.

[10] V. Algazi, R. Duda, D. Thompson, and C. Avendano, "The cipic hrtf database," in *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, 2001, pp. 99–102.

[11] J. O. Smith, *Introduction to Digital Filters with Audio Applications*. `http`:http://ccrma.stanford.edu/ jos/filters///-`ccrma.stanford.edu/~jos/filters/`, accessed 2024, online book.

[12] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[13] X. Qi and L. Wang, "Parameter-transfer learning for low-resource individualization of head-related transfer functions." in *Interspeech*, 2019, pp. 3865–3869.

[14] R. Duraiswami, L. Davis, S. Shamma, H. Elman, R. Duda, V. Algazi, Q.-H. Liu, and R. Raveendra, "Individualized hrtfs using computer vision and computational acoustics," *Acoustical Society of America Journal*, vol. 108, pp. 2597–, 11 2000.

[15] C. Wenn, "Headphone listening in live performance: a phenomenology of sound design," *Theatre and Performance Design*, vol. 1, no. 3, pp. 236–255, 2015.

[16] L. Gueorguieff, "Fast transient detection and processing algorithms for time scaling of audio files via a rigid phase-locked vocoder," *information technologies and control*, vol. 108, pp. 22–29, 3 2012.

[17] openframeworks, "openframeworks," https://github.com/openframeworks/openFrameworks, 2023.

[18] A. Caillon and P. Esling, "Rave: A variational autoencoder for fast and high-quality neural audio synthesis," 2021.