Zerr~
Autogenous Spatialization in PD and Max

Zeyu Yang
Independent Researcher
Berlin
zeyu-yang@outlook.com

Henrik von Coler
School of Music
Georgia Institute of Technology
hvc@gatech.edu

ABSTRACT

Zerr~ is a collection of externals for autogenous spatialization and spatial sound synthesis, available for both Pure Data and Max. This channel-based approach distributes audio signals to arbitrary loudspeaker setups, using audio-rate modulations. This paper describes implementation details of the Pure Data and Max externals and introduces basic techniques and strategies for performing and composing with the objects. Examples from a live-performance that uses only PD vanilla with the Zerr~ externals show how different basic synthesis techniques can be used with an 8-channel loudspeaker setup.

1. INTRODUCTION

Autogenous spatialization is a method for distributing monophonic signals to arbitrary loudspeaker configurations in real-time. It dynamically shifts the input signal between the individual loudspeakers, based on the signal's audio features or other control trajectories. Depending on the exact algorithm and signal qualities, this results in spatial modulations at high rates, thus significantly altering the original sound. This makes autogenous spatialization an edge case between channel-based spatialization and spatial sound synthesis.

The concept of autogenous spatialization is closely related to that of spatiomorphology [1]. Timbre, texture, and spatial distribution are inherently connected. Since the audio-rate modulation of loudspeaker gains can cause significant distortions, artifacts and additional spectral components, this technique is best suited for simple input signals like modulated sine waves, basic wave forms like square wave or sawtooth as well as filtered noise.

Zerr~ the software library for autogenous spatialization, has been presented in a previous publication [2] and is intended as a foundation for implementations in various music coding environments. Pure Data was chosen as the first build target in the first stage due to its modularity and possibilities to explore the concept through patching. With this paper we accompany the release of additional externals for Cycling '74 Max to make it more accessible.

After contextualizing autogenous spatialization in Section 2, details on the implementation in PD and Max are presented in Section 3. Section 4 explains basic concepts for the composition and performance with the externals in Pure Data for an application in an improvised study. A final Section 5 discusses both the implementation and application.

2. RELATED WORK

2.1 Channel-Based Spatialization

Unlike VR and cinema's perceptual rendering methods, channel-based spatialization emphasizes fixed loudspeaker configurations and reproducibility in concrete settings. Tools such as the BEASTmulch system [3] and SpatDIF frameworks [4] enable composers to design adaptable spatial configurations tailored to venue-specific speaker layouts.

Early explorations in electroacoustic music employed channel-based spatialization, notably in quadraphonic and octophonic formats [5]. The spatial trajectories and form are incorporated into compositional structure through analog mixers, diffusion scores, and matrix-based routing during live spatialization. In this context, spatialization serves both compositional and performative roles. Techniques such as trajectory-based panning, dynamic source grouping, and algorithmic diffusion are commonly employed in contemporary music practices [6].

Some works integrate spatialization so deeply with algorithmic composition that it creates emergent spatial forms that are inseparable from the underlying musical material. This body of work demonstrates that spatialization functions not merely as a technological affordance but as an expressive and structural dimension of musical language.

2.2 Spatial Sound Synthesis

Spatial sound synthesis is deeply rooted in the artistic and technical innovations of music. This approach integrates spatial aspects into the early stages of sound synthesis rather than treating them as post-processing effects. This integration allows sounds to be directly synthesized in physical space, with spatiality as an inherent characteristic of the sound itself.

Purpose-built instruments and synthesis systems – such as Xenakis's UPIC system and Stockhausen's rotating table – demonstrate the rich historical lineage of spatial sound synthesis, even in the analog domain [7]. The advent of digital synthesis introduced audio programming environments such as SuperCollider, Max, and Csound, allowing composers to program spatial behaviors directly into

the synthesis process without custom hardware [8].

Innovative spatial sound synthesis strategies were developed through both the extension of traditional synthesis methods and the establishment of entirely new approaches. Works by Natasha Barrett and Denis Smalley showcase how spatial motion and distribution become fundamental to timbral morphology and formal development [9, 10].

2.3 Audio-Rate Modulation

Although audio-rate modulation is primarily used for timbral synthesis, it presents compelling possibilities for spatial sound synthesis and spatialization. In multichannel and electroacoustic music contexts, composers have explored using audio-rate signals to modulate spatial parameters—azimuth, elevation, and diffusion trajectories [11]. This approach treats spatial attributes as dynamic synthesis parameters, enabling the generation of spatial tremolo, rotating fields, and pulsating source locations that evolve at rates perceptually fused with the timbral domain.

Natasha Barrett and Øyvind Brandtsegg's works demonstrate real-time audio-rate control of spatial trajectories, generating spatial textures that function like spectral phenomena [9, 12]. Such techniques often exploit amplitude- or phase-modulated multichannel panning networks, where modulator oscillators drive spatial movements across fixed loudspeaker arrays. In contrast to envelope-based automation or control-rate panning, audio-rate spatial modulation introduces rapid fluctuations that blur the perceptual boundary between space and timbre [13].

Audio programming environments facilitate this experimentation by enabling direct audio-rate control of spatial parameters—including routing, spatial interpolation, and per-speaker gain factors [14]. Zerr~ extends these capabilities through customized externals that adhere to the notion of autogenous spatialization, building upon the previously discussed concepts and practices.

3. THE ZERR~ EXTERNALS

3.1 General Design

Zerr~ is an work-in-progress implementation of the autogenous spatialization conceptual framework. For details about the system design and foundational concepts, please refer to the published paper [2]. The code is open source under MIT license and available on GitHub [1] .

The Zerr~ system consists of core modules and host environment encapsulations. The core modules are linked as static libraries for each encapsulation. These modules follow the signal flow diagram structure, with each module (except the Speaker Manager) functioning as an independent audio-rate signal processor.

The Feature Tracker module provides a standardized interface for audio feature extraction algorithms, with centralized buffering and FFT processing to optimize performance. It handles multiple feature algorithms simultaneously. The host environment provides the Feature Processor functionality. The loudspeaker array layout is defined in a configuration file. The Speaker Manager loads each speaker's parameters from this file and controls properties and speaker selection in response to control signals at runtime. The Envelope Generator produces a separate modulation signal for each speaker. The Envelope Combinator merges matching modulation signals, while the Audio Disperser multiplies the input audio signal with these modulation signals to create the final output.

Two fully functional encapsulations of the Zerr~ system have been developed: one for Pure Data and one for Max. Other environments remain in the research and preliminary development phase. The JACK-based command-line version, originally used as a proof of concept, is now outdated compared to the Pure Data and Max implementations. The hard-coded mapping strategies from the JACK version have been translated into presets in the current Pure Data and Max packages.

3.2 Pure Data Package

The Pure Data Package was the first encapsulation to offer sufficient flexibility for exploration and has since been used in several evaluations and creative projects. In this implementation, each audio-rate signal processing module is compiled into a separate external using pd-lib-builder [2] . The implementation follows Pure Data's conventions for inlets/outlets, arguments, and messages, in line with its design philosophy. Audio-rate I/O is handled via the external's inlets and outlets. Initialization parameters are set through inline arguments, while parameters modifiable at runtime are handled through messages. The following externals have been developed:

- **zerr_features~**
- **zerr_envelopes~**
- **zerr_disperser~**
- **zerr_combinator~**

Detailed usage instructions are documented in each external's corresponding help patch. These patches provide comprehensive examples demonstrating the external's functionality, as well as documentation of all supported arguments and message formats. The package includes the compiled externals, their help patches, example speaker array configuration files, and a collection of regularly updated presets to support various spatialization scenarios.

Figure 1 shows an example patch containing all the Zerr~ externals. Audio from adc~ feeds into zerr_features~ and zerr_disperser~. zerr_features~ performs real-time audio feature extraction, including spectral centroid, spectral flux, and spectral rolloff. The audio feature signals are processed with Pure Data's numerical objects and then sent to two instances of zerr_envelopes~, each configured with different modes (trajectory/trigger) but initialized with the same "ring" configuration file. The resulting modulation signals are merged by zerr_combinator~, which then modulates the source signal in zerr_disperser~ and outputs to an 8-channel loudspeaker array.

The Pure Data Package was developed before Pure Data 0.54-0 [3] , which introduced official support for multichannel audio signals. As a result, current patching in PD

[1] https://github.com/ringbuffer-org/Zerr

[2] https://github.com/pure-data/pd-lib-builder

[3] https://puredata.info/downloads/pure-data/releases/0.54-0

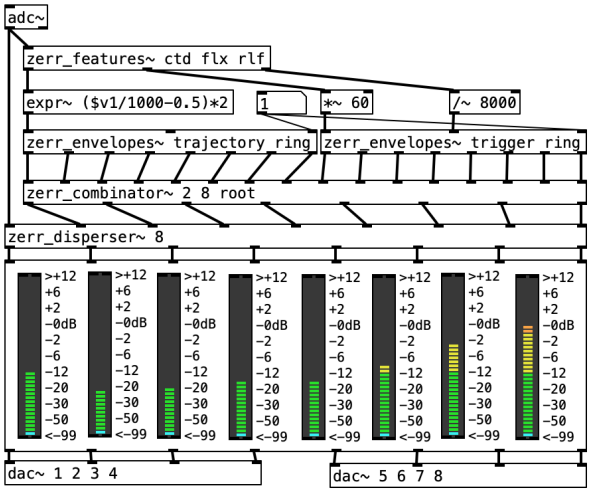

Figure 1. Example Patch using Zerr~ Externals in Pure Data for spatialization on 8 loudspeakers.

remains redundant, requiring repetitive manual single-channel cord connections. Adding multichannel audio signal support for the Pure Data Package is the highest priority for future development.

3.3 Max Package

The Max Package is developed using the Min-Devkit [4], which provides modern C++ interfaces through Min-API [5] and convenient CMake setups. However, under the hood, the externals are developed directly with the low-level Max-API [6] to achieve better multichannel signal support. Thanks to the similarity between Max and PD APIs, porting wrapper code from Pure Data to Max is straightforward and requires minimal changes to encapsulation strategies. The following externals have been developed:

- **mc.zerr.envelopes~**
- **mc.zerr.combinator~**
- **mc.zerr.disperser~**
- **mc.zerr.features~**
- **zerr.features~**

Multi-channel externals using the mc. prefix were developed as the initial Zerr~ externals for the Max Package. The only non-mc external is zerr.features~, since audio feature signals typically require separate processing. Each external includes detailed usage instructions in its corresponding help patch as well.

Figure 2 shows an example Max patch containing all necessary Zerr~ externals with functionality identical to the Pure Data example in Figure 1. The patches demonstrate similar patching conventions between the two environments. The key difference lies in simpler inline arguments for the Max externals. Since the multi-channel cords dynamically adjust output channels based on input channel count, certain externals no longer need explicit port count specification during initialization.

[4] https://github.com/Cycling74/min-devkit
[5] https://github.com/Cycling74/min-api
[6] https://sdk.cdn.cycling74.com/max-sdk-8.2.0/index.html

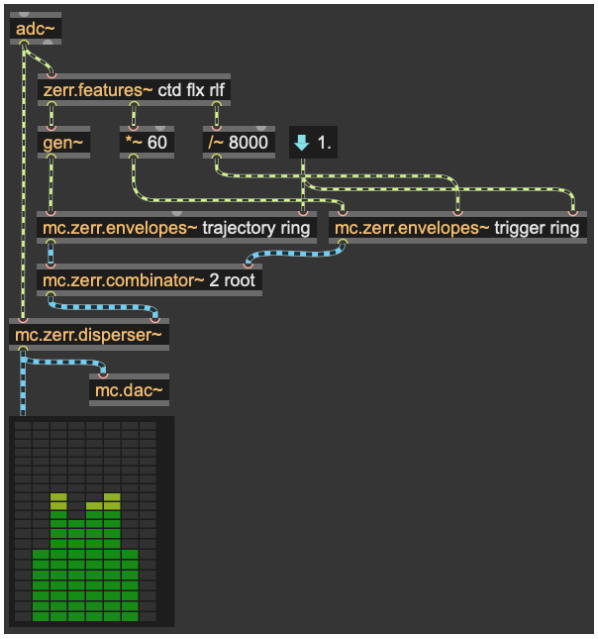

Figure 2. Example Patch using Zerr~ Externals in MAX

3.4 Comparison & Further Development

Although similar patching conventions and consistent functionality are well preserved across Zerr~ externals from Pure Data and Max packages, there are still some notable differences. Currently, using Zerr~ externals in Max provides a better patching experience, leveraging the platform's mature multi-channel system development. The distinct single-channel and multi-channel signal styling more intuitively displays the signal flow in patches, making it more user-friendly. Additionally, Max offers more native multi-channel objects to use alongside the Zerr~ Externals. The ability to dynamically adapt to multi-channel signal input changes during runtime using a customizable function to handle corresponding multichannel signal output changes further streamlines the patching process and expands creative possibilities.

The advantage of implementing our Zerr~ externals in Pure Data lies in its more developer-friendly nature. As open-source software, developers can directly reference the source code of native objects with similar functionality. This helps avoid niche issues that official documentation and examples may not cover. One example is the switch from MIN-API to Max-API during this project's development.

The choice to use Max-API over Min-API came after extensive testing. Min-Devkit lacks thorough documentation for multi-channel external development and does not provide appropriate example externals for dynamic multi-channel management. While dynamic multi-channel signal support is possible through the *maxclass_setup* special method using low-level APIs, our project's highly customized needs made this approach problematic. The Max-API, however, directly resolves all multi-channel signal and runtime inlet/outlet control challenges.

This project remains in an early development stage with several aspects requiring engineering optimization. While we plan to explore extensions for other hosts, our imme-

diate focus is improving the user experience in Pure Data and Max environments. This includes making initialization parameters dynamically adjustable, implementing intuitive visual feedback, and developing a convenient tool for loudspeaker configuration file generation.

4. EXPLORING ZERR~ IN PD

4.1 Basic Considerations

In this section we present a workflow for exploring the capacities of autogenous spatialization in PD and discuss its application in an improvised study. A first performance with an earlier version of the PD objects was created in 2024. It relied on a setup of modular and semi-modular synthesizers as input, which is one of the intended core applications for this spatialization paradigm.

With the approach presented in this paper we are aiming towards a more formalized and reproducible performance setup, completely realized in Pure Data. Performance parameters are controlled through a native PD GUI with a touchscreen or via MIDI controller mappings. This concept emphasizes the use of the computer with visual programming environments as a self-contained musical instrument. The concepts in this section were developed for the composition of an improvised study, that was premiered at the PdMaxCon25.

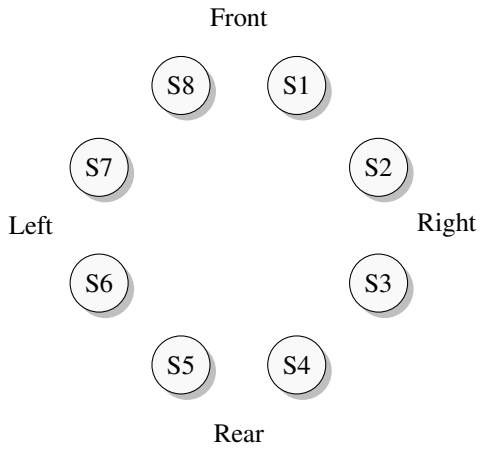

Figure 3. Ideal 2D setup with eight loudspeakers.

Although autogenous spatialization specifically allows the use of arbitrary and unconventional loudspeaker setups, this exploration is presented with an eight channel circular 2D loudspeaker setup, as shown in Figure 3. Similar configurations are still among the most common setups in electroacoustic music, thus increasing the portability and revivals of this study.

4.2 Basic Structure

The design goal behind the performance patch for this study is to allow instantaneous and flexible combinations of sound synthesis patches with different spatialization algorithms. This is realized through a send-bus system that can send each synthesis patch to any algorithm with an individual gain. Figure 4 visualizes this for a setup with four

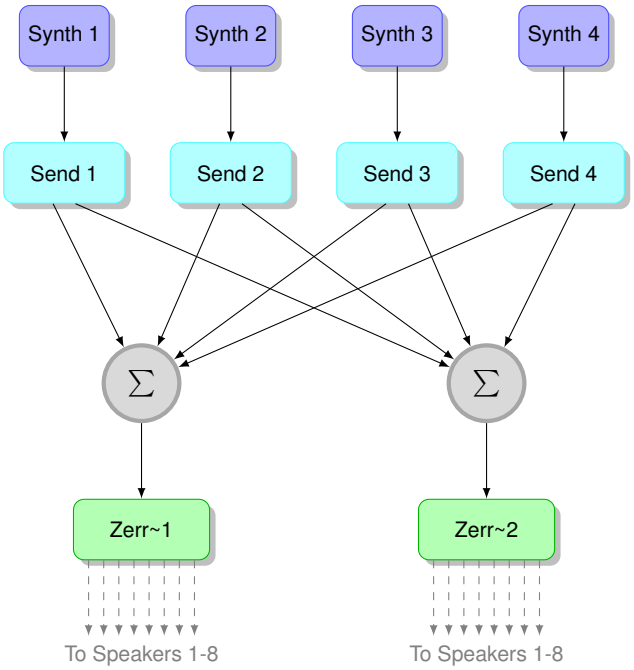

Figure 4. General abstraction with 4 synthesis blocks (purple), 4 send modules (turquoise) and 2 spatialization units (green) and output to 8 speakers.

synthesis patches and two spatialization algorithms. This system is based on three main building blocks: the synth modules (purple) for generating signals, the send modules (turquoise) for routing the signals, and the spatialization modules (green) to distribute them to the loudspeakers. The actual implementation for the performance works with more blocks, with up to eight synthesis patches and eight spatialization algorithms.

4.3 Building Blocks

Since each synthesis patch has its own send module, these are usually placed together for easy control. Figure 5 shows a synth module with orange faders for parameter control, together with its send module with turquoise faders. The additional sub-patch with VU meter and yellow/blue sliders is an optional LFO module to modulate the send gains.

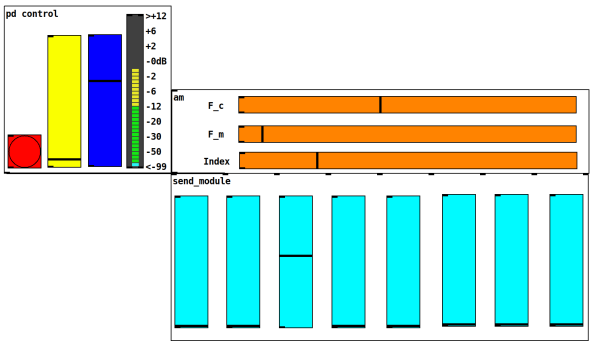

Figure 5. FM synth module (orange) with send module (turquoise) and additional controls for LFO (blue, yellow) and trigger (red).

4.3.1 *Synth Modules*

The synth modules are the only signal generators in this concept. They implement simple synthesis techniques with few parameters to tweak. For the recent study, the following techniques are included:

- 2-operator FM synthesis
- white noise with resonant low-pass
- wave folding
- ring modulation
- subtractive synthesis

Synthesis parameters are presented as wide horizontal sliders, as shown for the FM module in Figure 5. It gives access to the basic parameters: carrier frequency, modulator frequency and modulation index.

4.3.2 *Send Module*

The send module is a simple bank of eight vertical sliders, as shown in Figure 5. Each slider controls the send gain of the input signal (the first inlet) to eight individual spatialization patches. Eight additional inlets allow the use of control signals to modulate each send gain individually.

4.3.3 *Spatialization Modules*

The different spatialization modules are all patches similar to the example Zerr~ patch shown in Figure 1. Each of these modules implements a spatialization algorithm, explained in detail in the following Section 4.4.

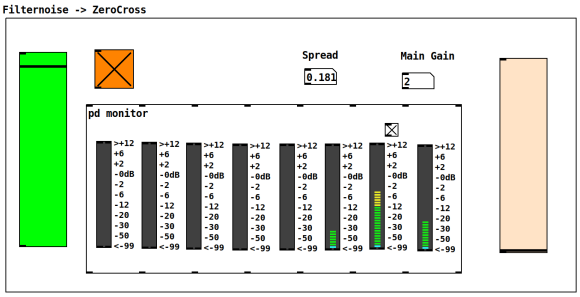

Figure 6. Spatialization module with meters for every loudspeaker and additional control sliders.

Figure 6 shows an example spatialization module. While most details of the patch are hidden, it shows VU meters for each of the output channels, respectively loudspeakers. This is the most important information for the performer, helping to anticipate the spatialization effect and tune it. Some spatialization modules contain additional sliders for real-time tweaking of parameters, in this case two vertical sliders.

4.4 Spatialization Algorithms

The send-based structure allows seamless exploration of different combinations between synthesis methods and spatialization algorithms. This is crucial, since the algorithms can have a very different behavior for different input signals.

Each algorithm defines how the features of the incoming audio signal affect its distribution along the loudspeakers. The fundamentals of this are described in a previous publication on autogenous spatialization [2]. The following examples show fundamental principles to distribute sound on the ring of eight loudspeakers. While this section is by no means exhaustive, it covers the techniques used for exploring the effect of different input signals and audio features.

4.4.1 *Vector Panning*

Vector panning is one of the most basic and effective algorithms in autogenous spatialization. In this eight-channel circular setup, it maps any control trajectory, typically an audio feature, to the panning azimuth of the signal. This is visualized in Figure 7. If, for example, a feature like the spectral centroid is used as control trajectory, the brightness of the signal affects the position of the sound accordingly.

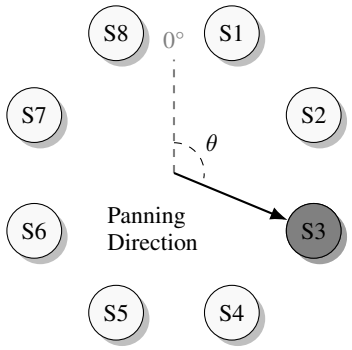

Figure 7. Azimuth (θ) controlled by a control trajectory, here panned to play from speaker 3.

4.4.2 *Cartesian Panning*

Cartesian panning can be used to shift a sound along any dimension in the Cartesian coordinate system. Figure 8 shows a panning between the front and rear direction in the loudspeaker ring, with the panning parameter x. In contrast to the directional vector panning, this concept can shift the signal to multiple speakers that align with a coordinate along a dimension, indicated through the gray horizontal bar. This kind of panning has for example been used with the signal's instantaneous amplitude as the control trajectory in the study.

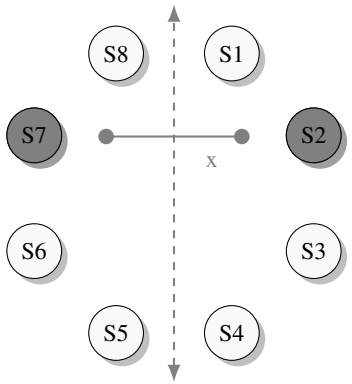

Figure 8. Front-Rear panning through control variable (x), here activating speakers S2 and S7 that align with the position.

4.4.3 Spread Modulation

The zerr_envelopes~ object has an additional audio-rate inlet to control the spread - or width, a method frequently used in object-based spatialization. Figure 9 shows a signal with spread, coming mainly from speaker 3 but also with less intensity from the neighboring speakers. For control trajectory values near 0, the signal will only come a single speaker - or two neighboring ones, when panned in between them, for a value of 1 it will come from all loudspeakers with the same level.

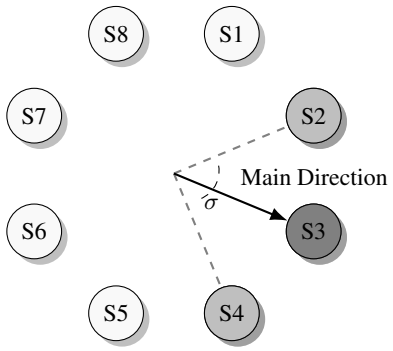

Figure 9. Width (σ) controlled by an audio feature, activating S3 fully, S3 and S4 partially.

4.4.4 Triggered Jumps

In trigger-based spatialization, a signal jumps from one speaker to another when receiving a trigger impulse. These impulses are a value of 1 in any audio rate signal. Which speaker is picked next can be selected in the Envelope Generator object. Modes include random, as shown in Figure°10 or sequenced.

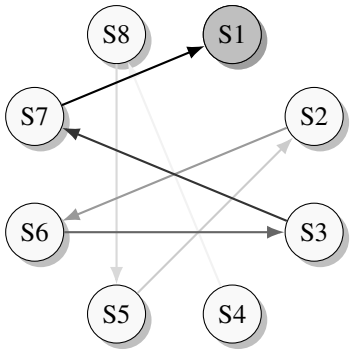

Figure 10. Trigger causing a jump to a random next speaker.

Trigger signals can be created in many ways. One method that has been used while working on the study is zero crossings as trigger when using white noise as input. This leads to a complete spatial diffusion. Using RMS and a threshold to create triggers causes a signal to jump when attack transients occur.

5. CONCLUSION

5.1 Externals and Implementation

The Zerr~ system was designed as a collection of externals rather than a monolithic structure, reflecting a commitment to modular design principles. By clearly separating functionality into discrete components, composers can build custom spatialization patches tailored to their artistic goals. This modularity invites experimentation and supports unconventional workflows—users are free to adapt or repurpose the externals beyond the intended autogenous spatialization concept.

Meanwhile, the core ideas of autogenous spatialization do not depend on Zerr~. Similar functionality can be implemented using built-in objects in Pure Data or Max, or through other spatialization packages. For example, the gen~ sub-environment of Max provides similar audio-rate modulation capabilities, offering an alternative way to achieve Zerr~-like functionality. However, we believe offering a complete, ready-to-use toolkit not only enhances creative efficiency but also provides a stable foundation for exploring diverse spatial techniques.

As an open-source project, Zerr~ encourages contributions from the community—whether in the form of bug reports, new presets, or creative works built with the system.

5.2 Exploration and Performance

Working on a study for the workflow proposed in Section 4 showed that a visual programming language like Pure Data is a perfect environment for the exploration of autogenous spatialization. It offers the necessary flexibility and does not involve highly complex patches that break the flow when designing and performing.

While the 'native GUI only' approach makes the setup as compact and reproducible as possible, it comes with significant limitations. The lack of multi-touch capacities makes fluent changes a challenge - and often impossible. One solution would be an external control software, sending OSC to Pure Data. This was not considered for the sake of keeping everything in a single software. Another option could be an add-on to enable multi-touch in pd-native GUIs, provided in a public repository [7] . Again, this would add additional dependencies and was thus not included.

And while the study created with this PD-only paradigm offers a more structured approach to exploring the capacities of autogenous spatialization, it still takes additional, more formalized concepts to display the full possibilities and idiosyncrasies of autogenous spatialization.

6. REFERENCES

[1] S. James, "Spectromorphology and Spatiomorphology of Sound Shapes: audio-rate AEP and DBAP panning of spectra," 2015.

[2] Z. Yang and H. von Coler, "Autogenous Spatialization for Arbitrary Loudspeaker Setups," in *2023 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*, 2023, pp. 1–6.

[3] J. Harrison, "BEASTmulch: Tools and techniques for 3D composition and performance," in *Proceedings of the International Computer Music Conference (ICMC)*, 2011.

[7] https://github.com/jyg/multitouch

[4] J. C. Schacher, C. Miyama, and T. Lossius, "The SpatDIF library–Concepts and Practical Applications in Audio Software," in *Proceedings of the International Computer Music Conference (ICMC)*, 2014.

[5] G. Emerson, "Acousmatic issues in sound-based music," in *The Cambridge Companion to Electronic Music*, N. Collins and J. d'Escriván, Eds. Cambridge University Press, 2007, pp. 17–31.

[6] K. Norman, "Spatiality and performativity in electroacoustic music," in *Proceedings of the Electroacoustic Music Studies Network (EMS)*, 2011.

[7] C. Roads, *Composing Electronic Music: A New Aesthetic*. Oxford University Press, 2015.

[8] N. Wilson, J. McCartney, R. Cottle, and N. Collins, "New developments in SuperCollider," in *Proceedings of the International Computer Music Conference (ICMC)*, 2011.

[9] N. Barrett, "Spatio-musical composition strategies," *Organised Sound*, vol. 7, no. 3, pp. 313–323, 2002.

[10] D. Smalley, "Space-form and the acousmatic image," *Organised Sound*, vol. 12, no. 1, pp. 35–58, 2007.

[11] A. Field, "Sound distribution and spatial movement: A composer's perspective," in *Proceedings of the International Computer Music Conference (ICMC)*, 2000.

[12] O. Brandtsegg, "SpatMod: Real-time spatial audio modulation," in *Audio Mostly Conference*, 2007.

[13] B. Blesser, "The spatiality of sound," *Soundscape: The Journal of Acoustic Ecology*, vol. 1, no. 1, pp. 15–18, 2001.

[14] N. Collins, "Live coding of audio-rate synthesis parameters in SuperCollider," in *Proceedings of the International Computer Music Conference (ICMC)*, 2011.