Solomon Rosenthal

California Institute of the Arts

BFA Thesis Project

Music Technology, Interaction, Intelligence and Design

Documents of a Practice of Audio Toolmaking

Contents:

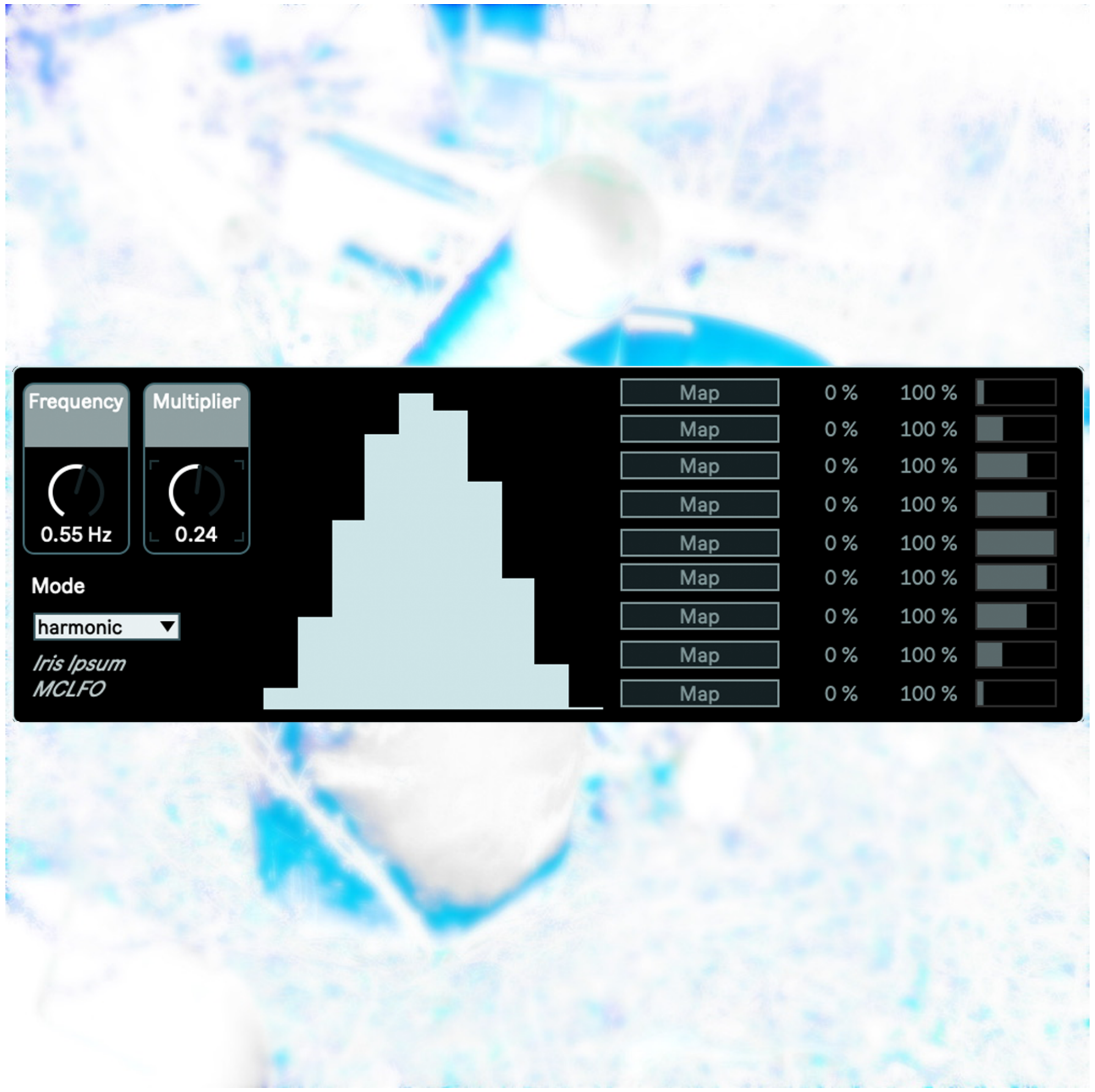
Abstract

This thesis project documents my recent work in the research and development of an arsenal of creative audio tools, covering VST and max for live plugins, physical MIDI interfaces, and microphones. The project will discuss both the technical specifications of each device and the conceptual basis of developing an artistic practice that is engaged with the process of building its own tools.

Section 1

Max for Live Projects

This following section covers a selection of the various Max for Live audio tools I have created and released over my time at Calarts. They include modulation devices, experimental audio processing tools, visualizers, and sound generators. Globally I have sold over 7000 of these units in 94 different countries using an open source pay what you want model. In terms of their research and development, many of these devices were created in response to a lack of similar options available in the audio production marketplace and to satisfy my own creative need for interesting audio tools within my sound practice. These devices and their demo videos are hosted online on Gumroad, where they can be downloaded with a pay what you want model: https://irisdevices.gumroad.com/



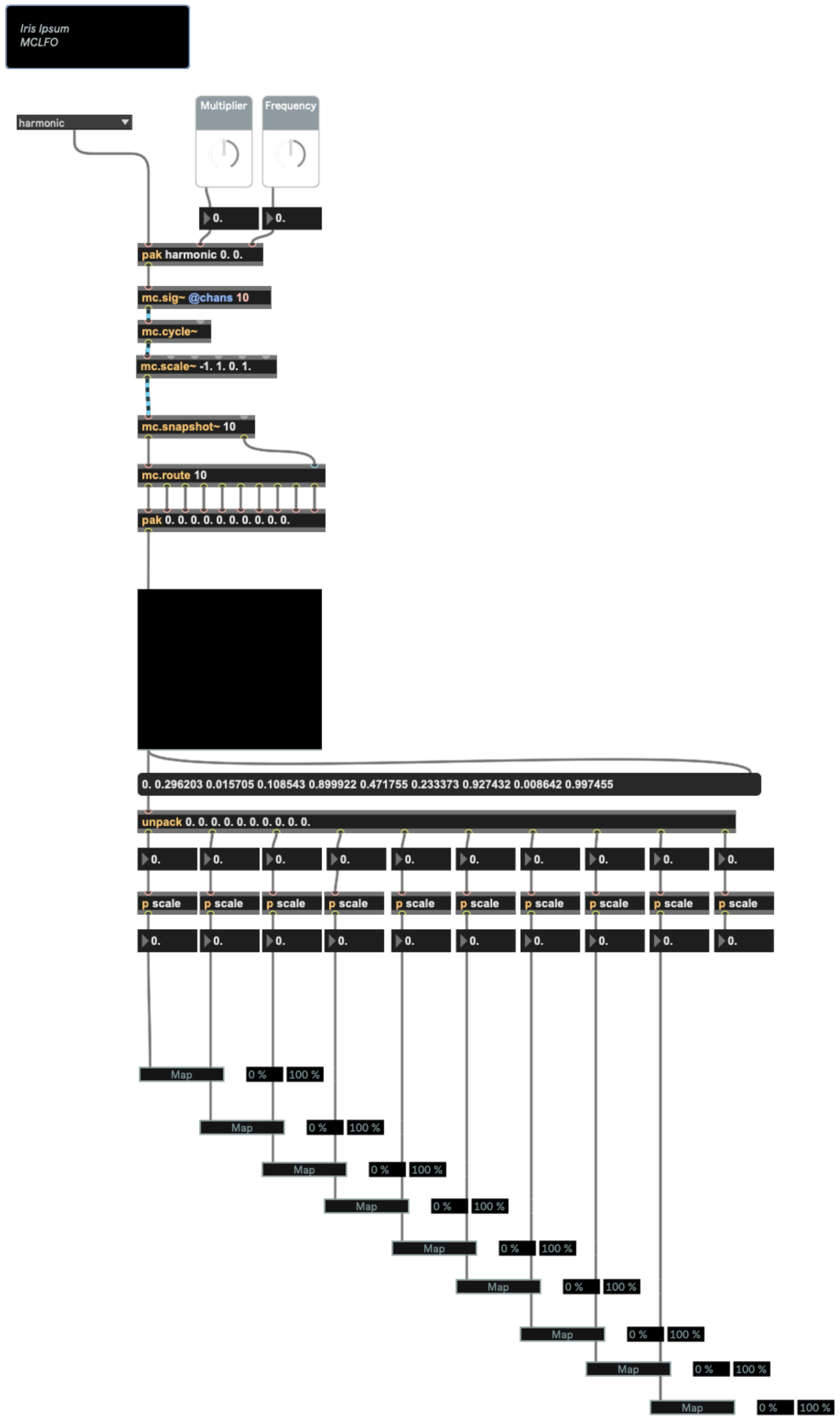
Frequency
0.55 Hz
Multiplier
0.24
Mode
harmonic

1.1

MCLFO

MCLFO is a Max for Live modulation device that utilizes the theory of the harmonic series to control the frequencies of a 9-band, 9-oscillator sine wave multi-channel LFO. The values of each LFO can be mapped to separate parameters inside Ableton Live, and each mapping has variable range settings. The oscillator's frequency ratios are synchronized to each other and can be adjusted with seven modes: Harmonic, Subharmonic, Exponential, Deviate, Decide, Increment, and Spread. The device has two primary adjustable parameters: frequency and multiplier. The frequency parameter sets the fundamental frequency that is used to generate the harmonic series across the LFOs, and the multiplier parameter modifies the ratio between successive oscillators.

The core of the device is implemented with Max's mc objects. An mc.sig~ receives a pak message comprised of a mode symbol and two floating-point parameters (which consist of the fundamental and multiplier values). These signals then drive an mc.cycle~ object that produces the ten-channel bank of phase-coherent sine oscillators. The raw bipolar output of this oscillator bank in the range of −1 to 1 is converted to a unipolar modulation signal via mc.scale~ and then down-sampled to a numerical value using mc.snapshot~. The resulting ten floating



point values are routed with mc.route and visualized with a ten band multislider. These values are then mapped to the standard 0–127 range for direct use as automation and MIDI-style control values.

Each device mode applies a unique formula to determine the frequency of the oscillators and their relationships to each other. Harmonic mode creates a harmonic series across the LFOs with the following formula:

$$F_c = F \cdot \left(1 + N \cdot C\right)$$

where N is the multiplier and C represents the oscillator index. For example, in Harmonic mode a frequency input of 440 and a multiplier input of 1 would generate the following series: 440, 880, 1320, 2640, etc, whereas a frequency input of 440 and a multiplier input of 0.5 would generate this series: 440, 660, 880, 1100, etc. Subharmonic mode generates a subharmonic series using the following formula:

$$F_c = F \cdot \left(1 - N \cdot C\right)$$

For example, with a frequency input of 100 and a multiplier of 1, the device would output the following frequency values: 100, 50, 25, 12.5, etc. Exponential mode creates an exponential harmonic series across the oscillators, governed by the following equation:

$$F_c = F \cdot e^{-N \cdot C}$$

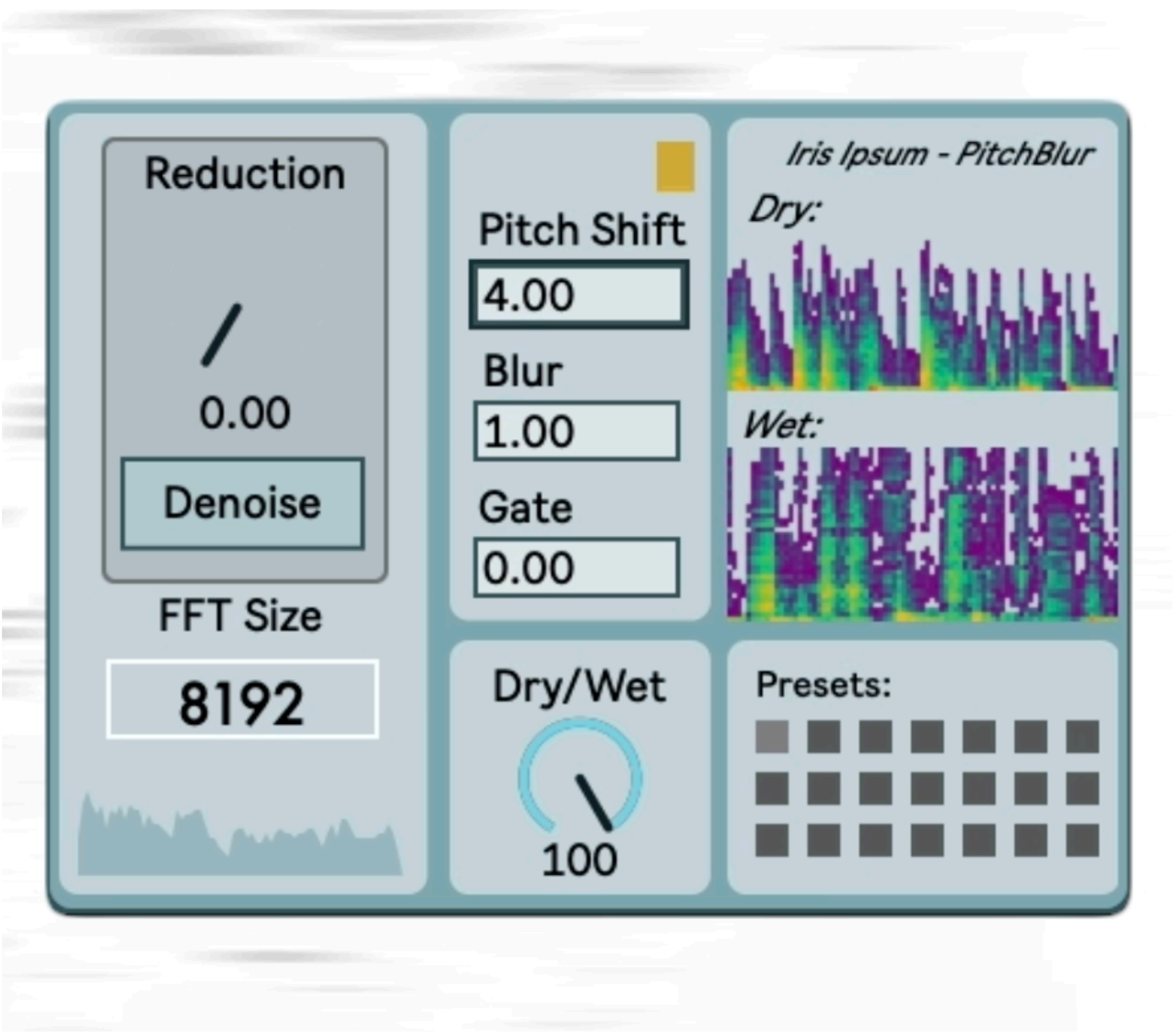
The Spread mode distributes the LFO frequencies linearly across a user-defined range, yielding evenly spaced rates between a minimum and maximum value. For example, in the Spread mode, if the device used 4 LFOs and had a frequency input of 10 and a multiplier input of 0, it would output LFO speeds of 0, 2.5, 5, and 7.5 hertz.

Increment mode treats the multiplier as an additive step size, increasing the frequency value of each LFO by an increment specified by the multiplier parameter. For example, in Increment mode with a frequency of 10 and a multiplier of 10, the device would output 10, 20, 30, 40, 50, 60. This mode is governed by the equation:

$$F_c = F + N \cdot C$$

.The remaining modes—Deviate and Decide—offer other creative variations centered around randomness.

The device enables a wide range of modulation patterns tailored to experimental and musical contexts. The frequency parameter can also be brought into negative hertz, which reverses the direction of the oscillation. This device expands the application of harmonic theory beyond sound synthesis by applying harmonic series principles to parameter automation in music production. By bridging these domains, it offers a novel approach to integrating natural acoustic processes and phenomena into sound design and modulation workflows.[6]



Reduction 0.00

Denoise

FFT Size 8192

Pitch Shift 4.00

Blur 1.00

Gate 0.00

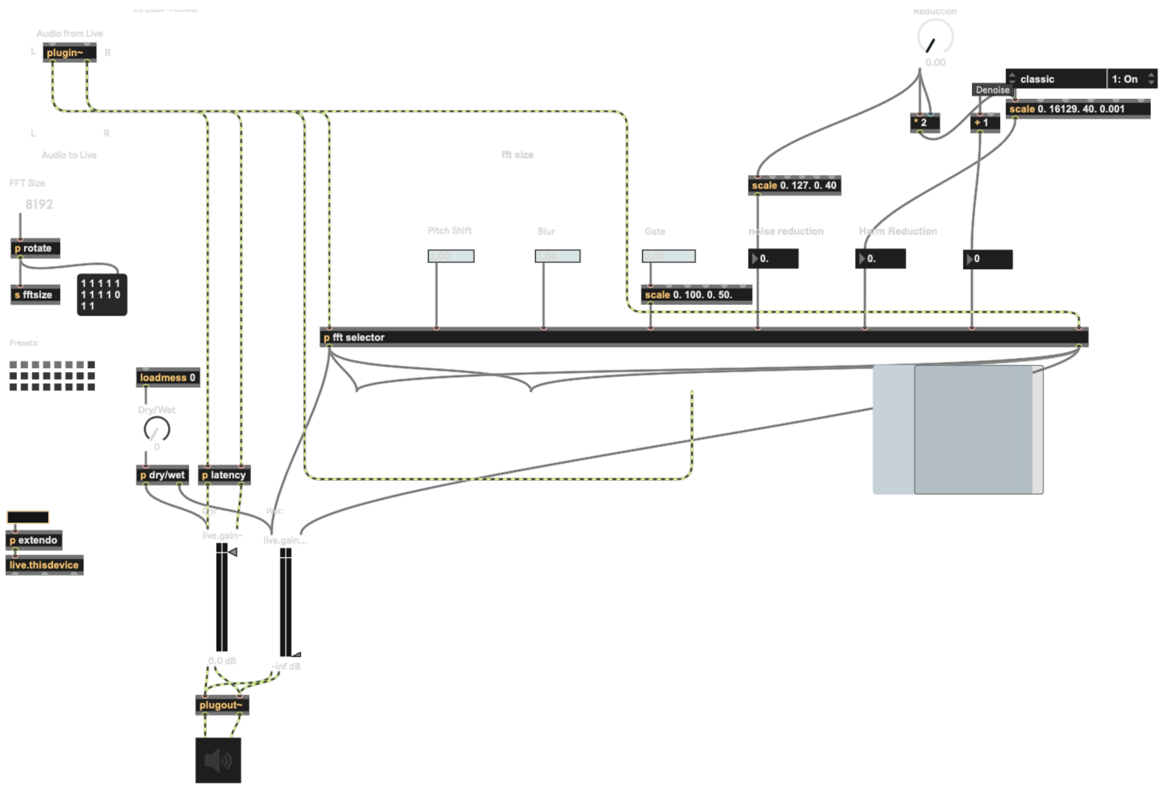
Dry/Wet 100

1.2

Pitchblur

Pitchblur is an Fast Fourier Transform based spectral multi-effect audio device. Its core DSP is built around a bank of per-bin processes running inside a series of *pfft~* subpatches. Each subpatch operates on the complex spectrum of the input signal to provide spectral denoise, deharmonic, blur, gate, and pitch-shift operations that can be combined to produce a myriad of interesting audio transmutations.

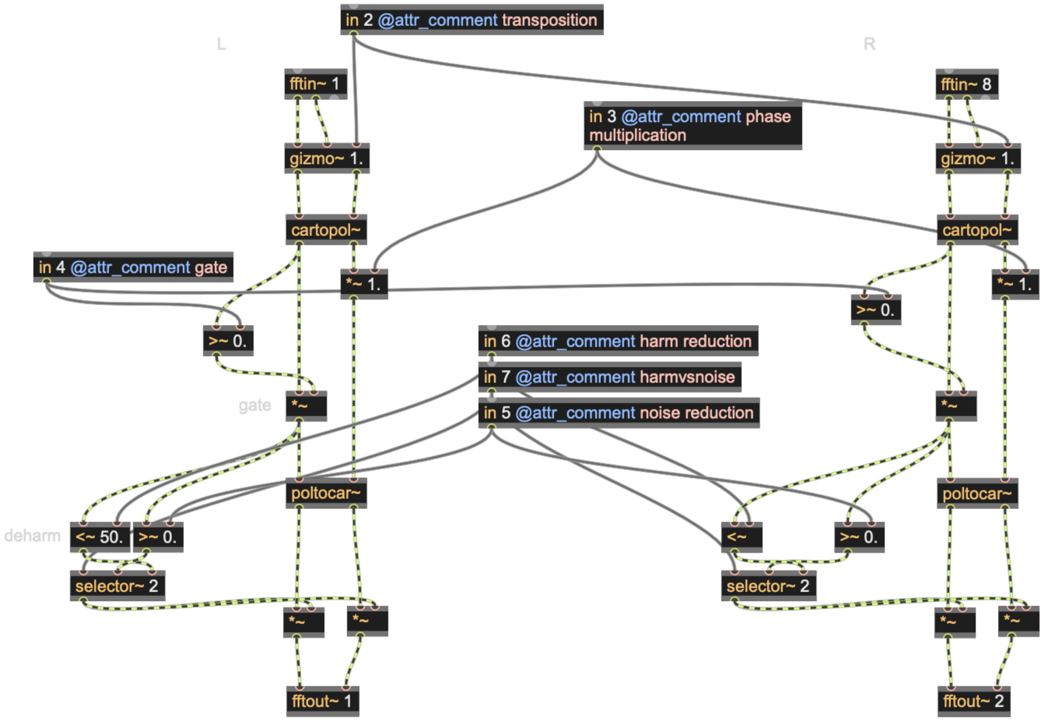
The device uses a variable FFT size (from 16 up to 32768 samples), allowing the user to move between high temporal resolution and high spectral resolution. Inside each pfft~, the incoming audio is analyzed via an *fftin~* object, yielding the real and imaginary components for each frequency bin. These are converted to magnitude and phase using *cartopol~*, and processed by the various effect stages. Then these phase and amplitude values are resynthesized with *poltocar~* and *fftout~*. All control parameters (transposition, phase multiplication, noise reduction, harmonic reduction, gate amount, etc.) are passed into the FFT domain via dedicated in objects, making the spectral processing fully modulatable from the Max for Live interface.



The spectral blur section is driven by a phase multiplication control routed to the phase stream of each bin. This control scales the per-bin phase with *~ objects before resynthesis, effectively decorrelating the phase relationships between neighbouring bins. As the multiplier moves away from 1, inter-bin phase relationships become increasingly unstable, producing a characteristic "smearing" or reverb-like diffusion. This blur is computed per FFT frame and is further shaped by the chosen FFT size, with larger windows yielding more pronounced spectral smear.

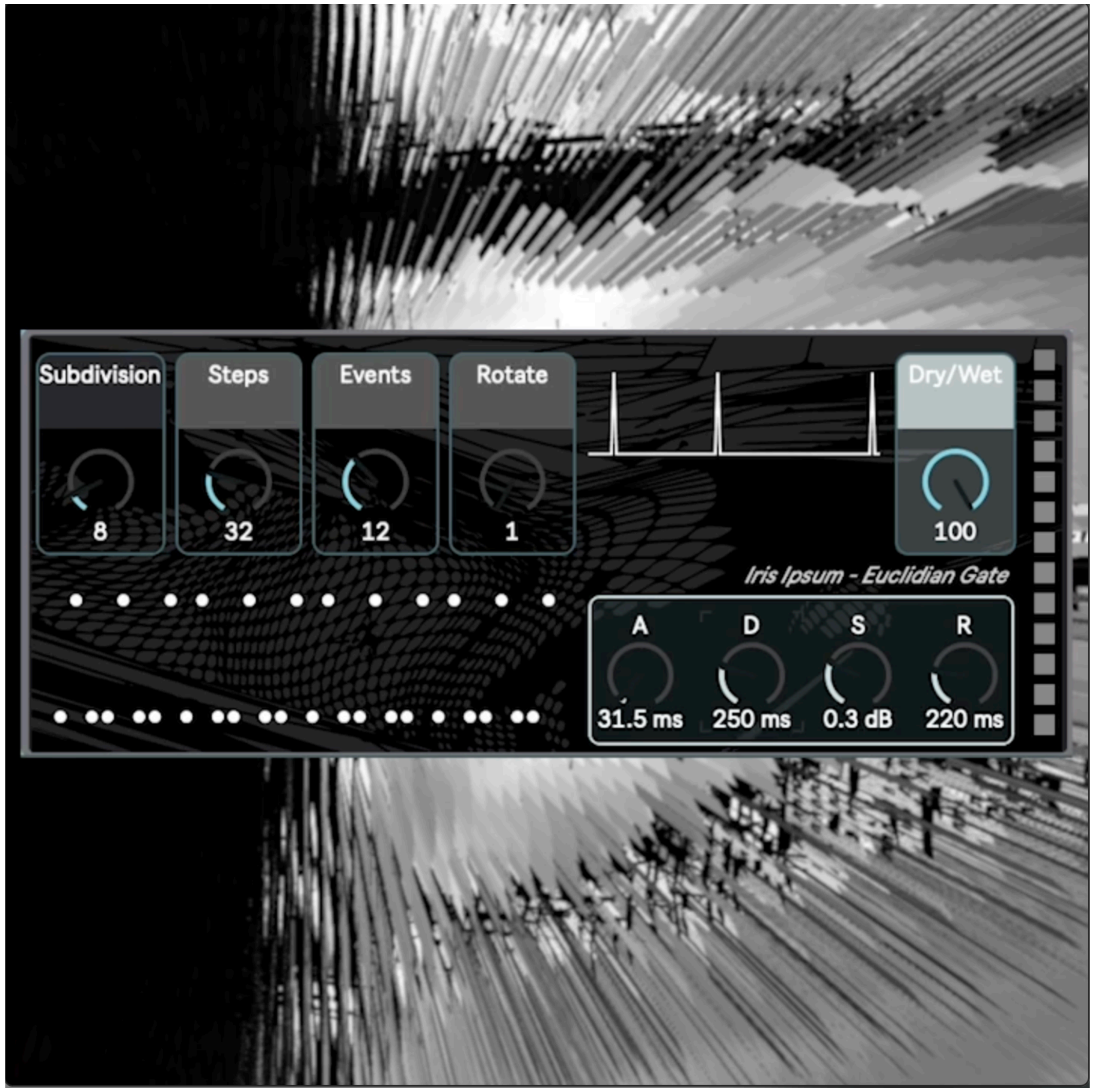
The denoise and deharmonic behaviours are implemented as complementary amplitude-gating operations on the magnitude of each bin. A set of *noise reduction* and *harmonic reduction* controls modulate thresholds that feed a network of comparison objects (>~ and <~). In the denoise configuration, bins whose magnitude exceeds a user-defined threshold are allowed to pass, suppressing low-level noise and room tone. In the deharmonic configuration, the inverse logic is applied: lower-amplitude, noise-like bins are preserved while stronger, more harmonic bins are attenuated, effectively inverting the typical denoiser and isolating inharmonic and noisy components of the spectrum. These gated magnitude streams are then recombined with their respective phase components and routed through *selector~* objects and a crossfade system to control the relationship between dry and processed spectra.

Pitch shifting is handled by phase-vocoder-style transposition using *gizmo~*. The transposition control is injected into the FFT processing via a dedicated inlet, and gizmo~ operates directly



on the complex spectrum to shift the perceived pitch without altering the overall duration. Additional phase-domain utilities such as *phasewrap~* and *framedelta~* are used to manage inter-frame phase continuity, ensuring that even extreme transpositions and blur amounts remain relatively artifact-stable and musically usable.

Pitchblur functions as a modular spectral laboratory: a single Max for Live device that exposes the inner workings of FFT analysis/resynthesis and allows denoising, deharmonization, spectral blurring, gating, and pitch shifting to be combined within one coherent interface. Conceptually, the project extends the classic phase vocoder into a performable, parameter-driven tool, translating the mathematics of the FFT and complex plane into an intuitive set of controls for sound design, diffusion, and experimental audio processing.[4]



Subdivision 8
Steps 32
Events 12
Rotate 1
Dry/Wet 100
A 31.5 ms
D 250 ms
S 0.3 dB
R 220 ms

1.3

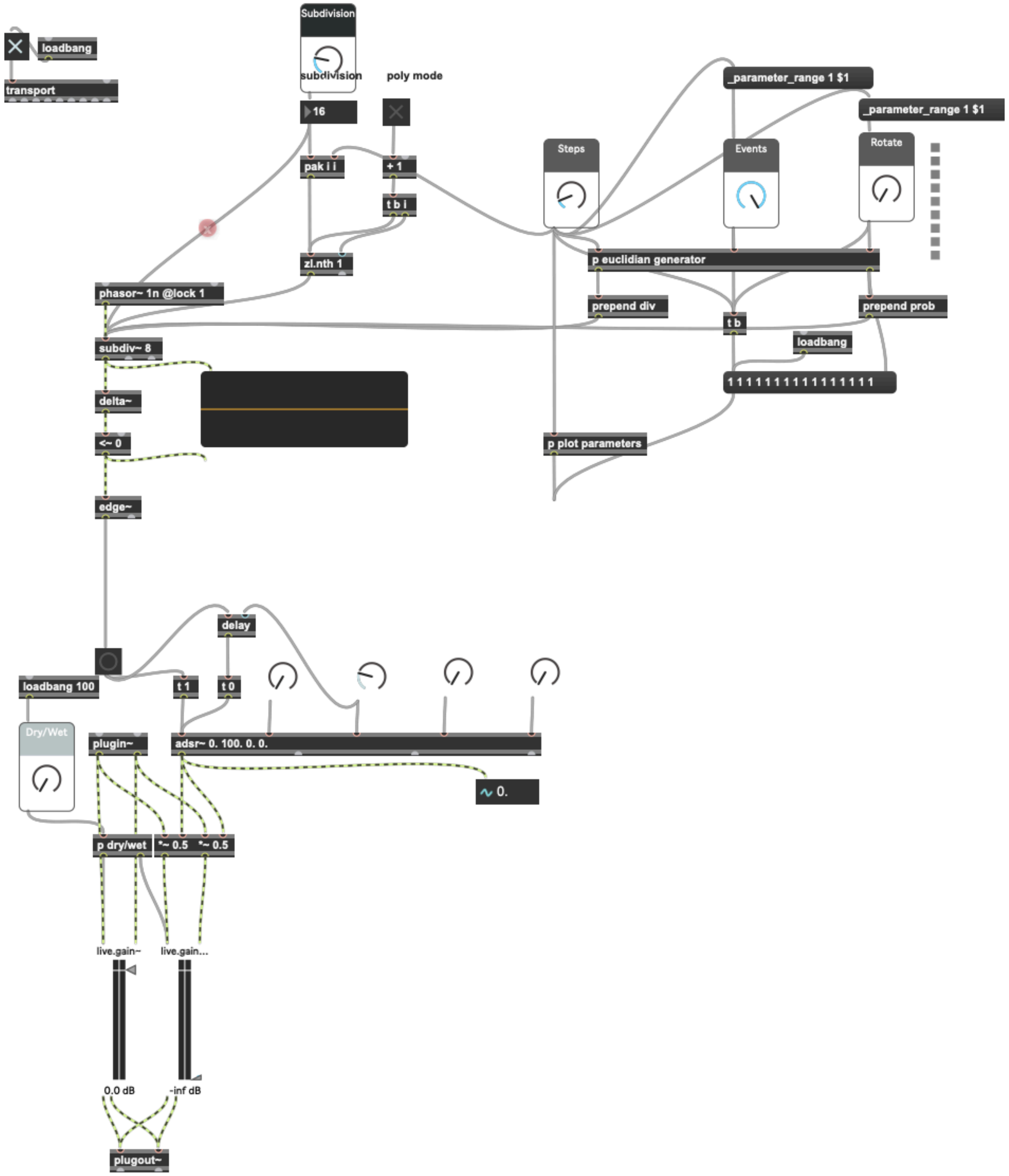
Euclidian Gate

Euclidian Gate is a gate sequencer built on the Euclidean rhythm algorithm. Euclidean rhythms utilize mathematical functions to distribute a chosen number of events as evenly as possible across a fixed number of steps. This logic appeared in musical traditions across the world including clave patterns and Balkan rhythms long before it was mathematically described. In Euclidian Gate, this mathematical structure is utilized as a flexible rhythmic modulator that allows for the generation of evolving and complex gate sequencer patterns.

The *Subdivision* parameter of the device defines how the Euclidean pattern aligns to musical time in Ableton. A subdivision value of 16 would generate 16th notes, a subdivision value of 8 would generate 8th notes, and a subdivision value of 3 would generate ¼ note triplets for example. Changing the subdivision value effectively scales the temporal resolution of the output gate pattern.

The *Steps* parameter determines the total number of steps or possible binary events within the Euclidean sequence. This corresponds to the length of one Euclidean cycle.

The *Events* parameter defines the number of active triggers distributed across the sequence. Increasing or decreasing the number of events changes the density of the resulting pattern.

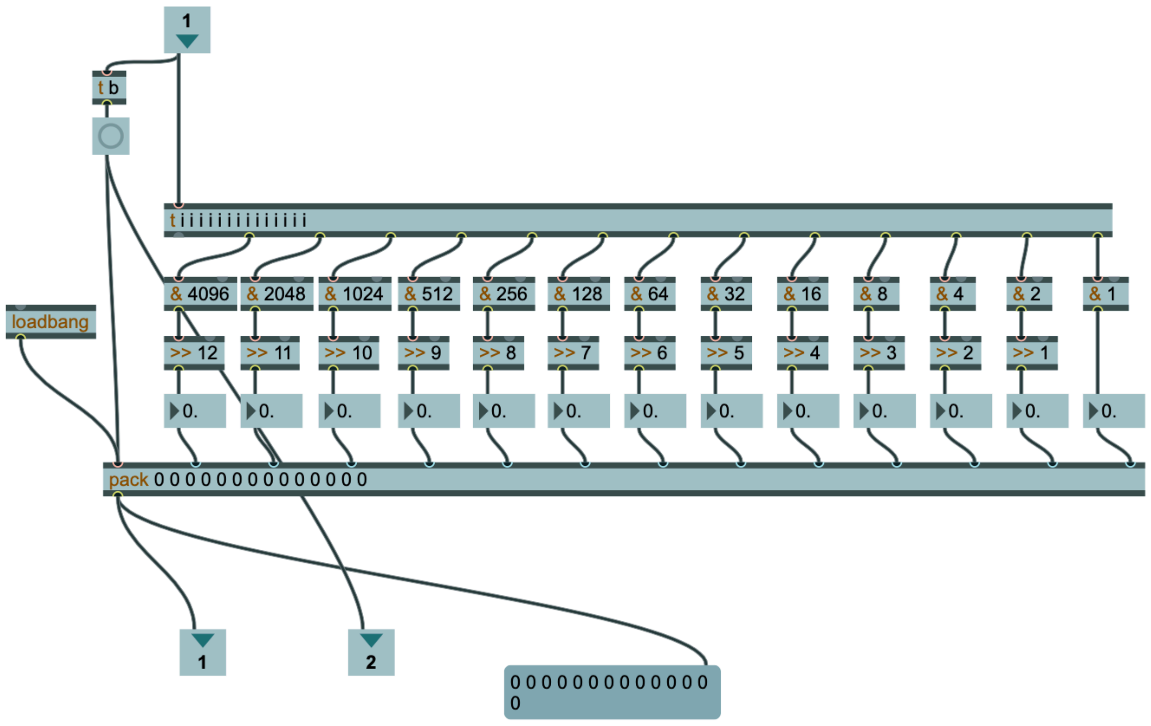


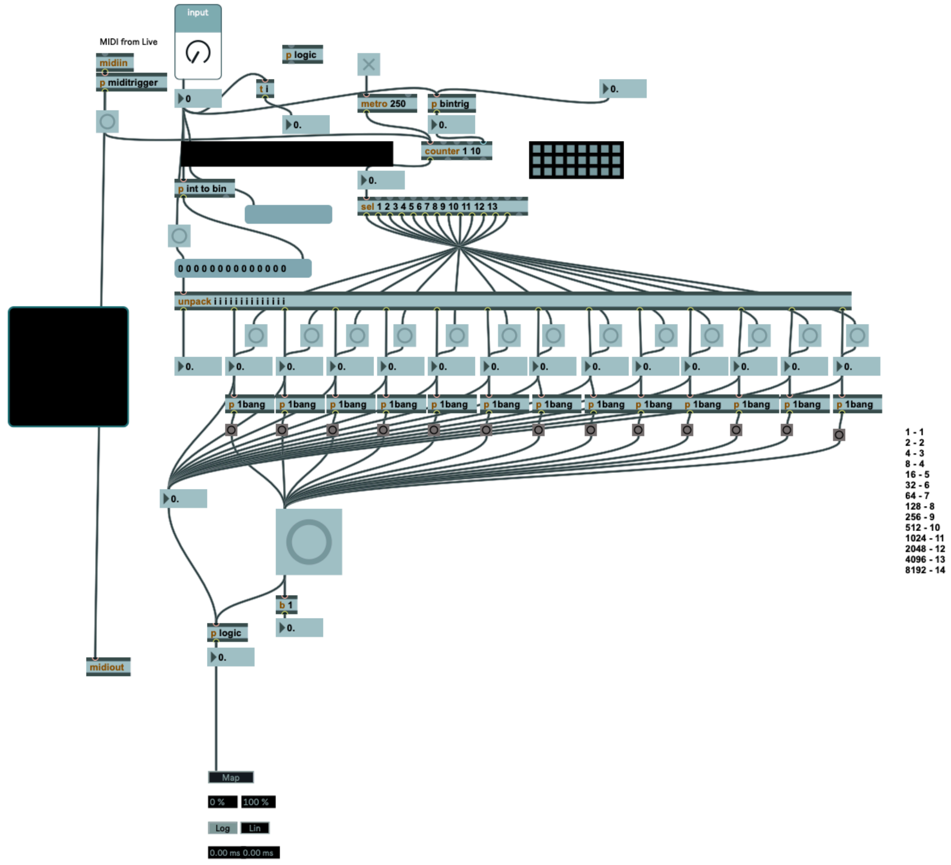
The *Rotate* parameter circularly shifts the pattern, allowing the user to offset the generated Euclidean rhythm.

Each triggered step passes through a built-in ADSR envelope, allowing for dynamic control over the amplitude shape of each gated pulse. This transforms the sequencer from a purely binary trigger source into a dynamic rhythmic sculpting tool.

Euclidian Gate also includes sixteen preset slots represented as small squares in the UI. Shift-clicking stores the current configuration and clicking recalls it. This enables performance-ready switching between rhythmic structures.

The device hopes to bring a mathematically elegant and globally rooted rhythmic principle into an intuitive Max for Live environment. By combining Euclidean distribution, envelope shaping, rotation, and preset recall, the device offers a fluid and expressive way to generate complex polyrhythmic modulation patterns for rhythmic gating inside Ableton Live.[10]





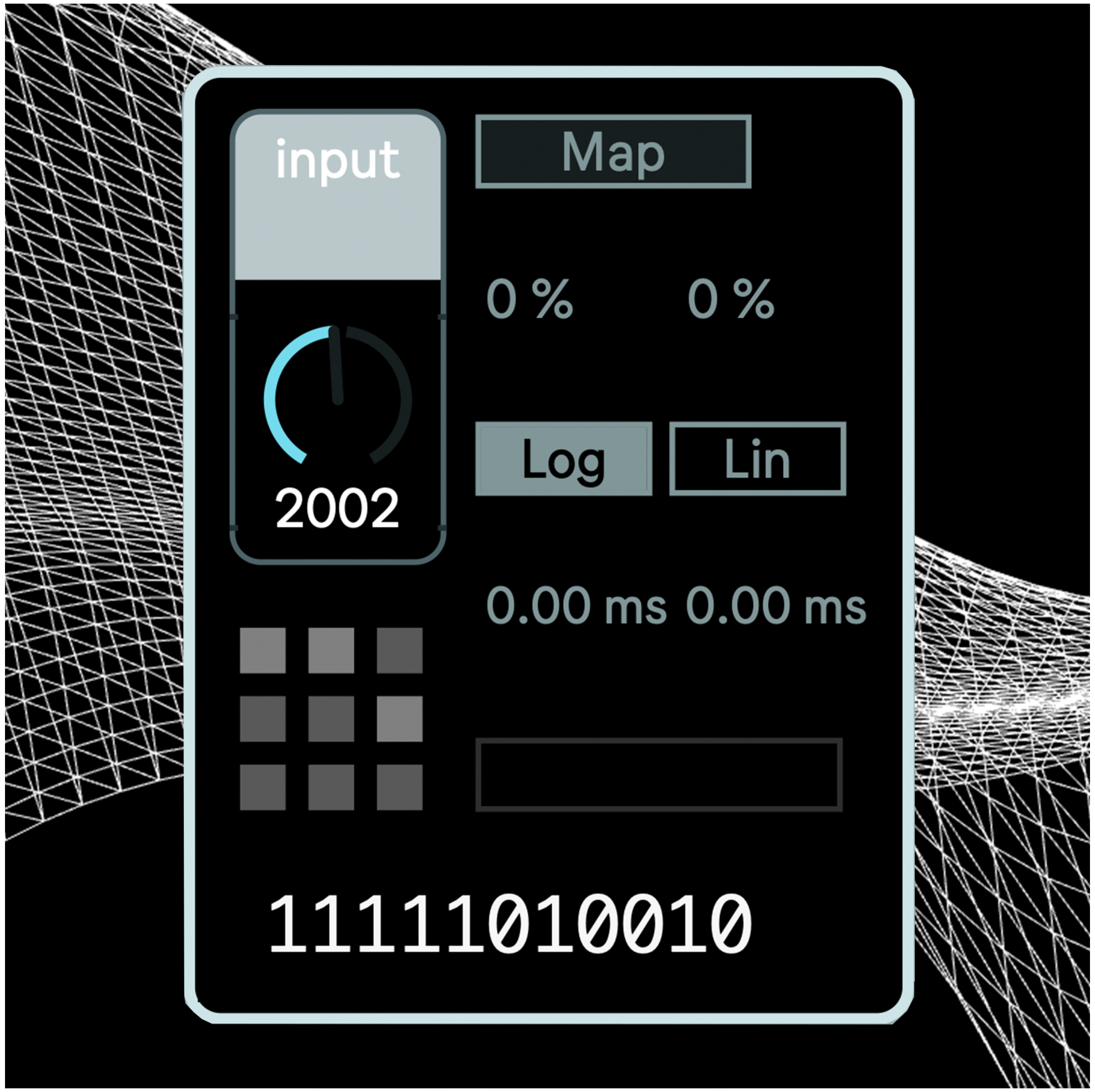
1.4

Binseq

Binseq is a Max for Live modulation device that generates rhythmic switching patterns using evolving binary sequences. Instead of relying on fixed step sequencing, Binseq converts user-specified base 10 integers into binary form and uses these bit patterns as modulation states. This approach makes it possible to explore a large and highly varied space of rhythmic and structural patterns using a single parameter. Because binary sequences naturally encode on/off states, they map intuitively onto rhythmic gates, switches, and parameter toggles within Ableton Live.

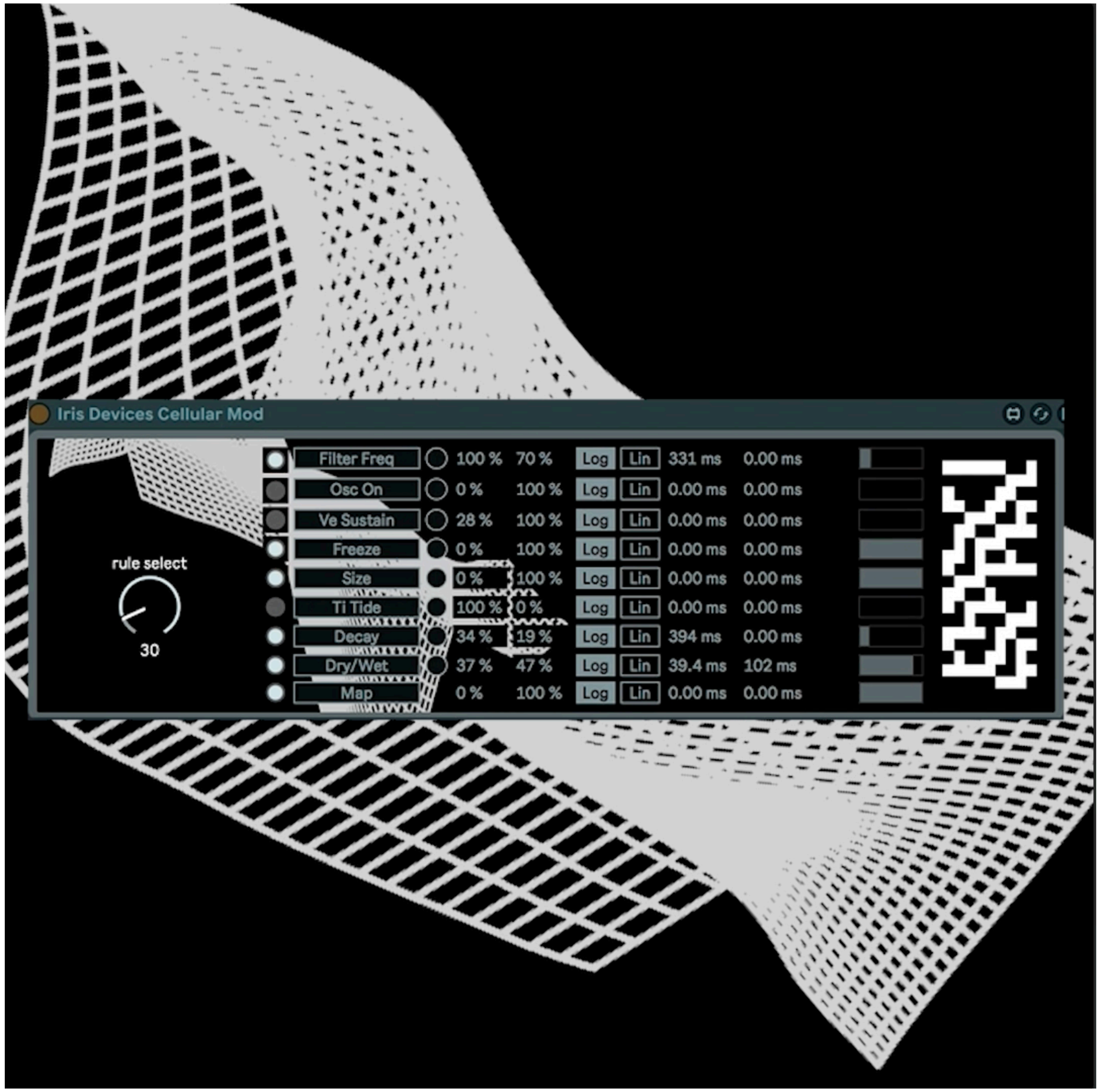
The device accepts a user input value *N* in the range 0 ≤ *N* ≤ 4096. Binseq then computes the binary representation of this number using an algorithm of bitwise intersection and bitshifting displayed on the left. This binary pattern is then looped continuously to form the modulation sequence. This allows for 4096 distinct rhythmic states using a single control.

Advancing through the binary sequence is driven entirely by incoming MIDI notes. Each MIDI event increments the sequence index, allowing the user to inject highly precise timing, syncopation, and groove into the modulation. Because the rhythm comes from MIDI rather than an internal clock, Binseq can be tightly aligned to any performance or sequencing workflow.



Each bit in the binary sequence outputs a switching signal that can be mapped to any parameter inside Ableton Live. When a bit equals 1, the device outputs a high signal; when it equals 0, it outputs low. This makes Binseq useful for rhythmic gating, switching between parameter states, toggling effects and building generative rhythmic systems among many other use cases. Because the sequence changes every time a new number is entered, it becomes possible to create evolving and generative modulation behaviors not achievable with traditional sequencers.

Similarly to Euclidian Gate, Binseq also includes a preset recall system, wherein nine preset squares allow for quick storing and recalling of patterns. This enables seamless performance transitions between binary patterns.



Iris Devices Cellular Mod

rule select
30

Filter Freq	100 %	70 %	Log	Lin	331 ms	0.00 ms
Osc On	0 %	100 %	Log	Lin	0.00 ms	0.00 ms
Ve Sustain	28 %	100 %	Log	Lin	0.00 ms	0.00 ms
Freeze	0 %	100 %	Log	Lin	0.00 ms	0.00 ms
Size	0 %	100 %	Log	Lin	0.00 ms	0.00 ms
Ti Tide	100 %	0 %	Log	Lin	0.00 ms	0.00 ms
Decay	34 %	19 %	Log	Lin	394 ms	0.00 ms
Dry/Wet	37 %	47 %	Log	Lin	39.4 ms	102 ms
Map	0 %	100 %	Log	Lin	0.00 ms	0.00 ms

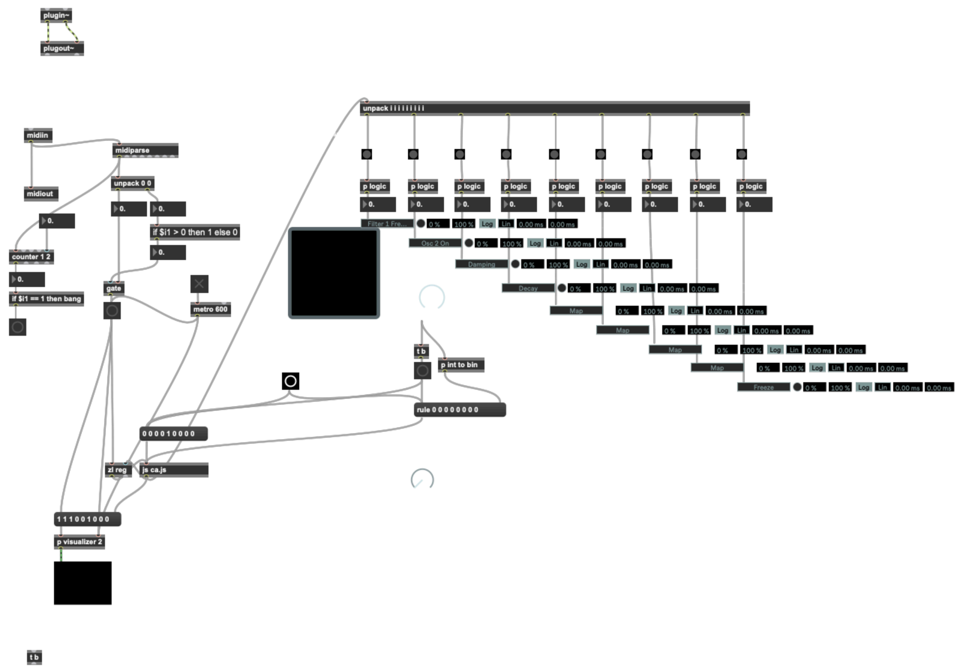
1.5

Cellular Mod

Cellular Mod is a Max for Live modulation device that generates complex control signals using a 9-bit elementary cellular automata system. Popularized by Stephen Wolfram, Elementary Cellular Automata systems are one-dimensional arrays where each digit has two possible states (0 or 1) and evolves according to deterministic rules based solely on the states of its nearest neighbors. Despite their simplicity, these systems can exhibit a wide range of behaviors, from stable repetition to chaotic unpredictability. By translating these binary states into modulation signals, Cellular Mod brings algorithmic complexity into Ableton's ecosystem in a musically accessible way.

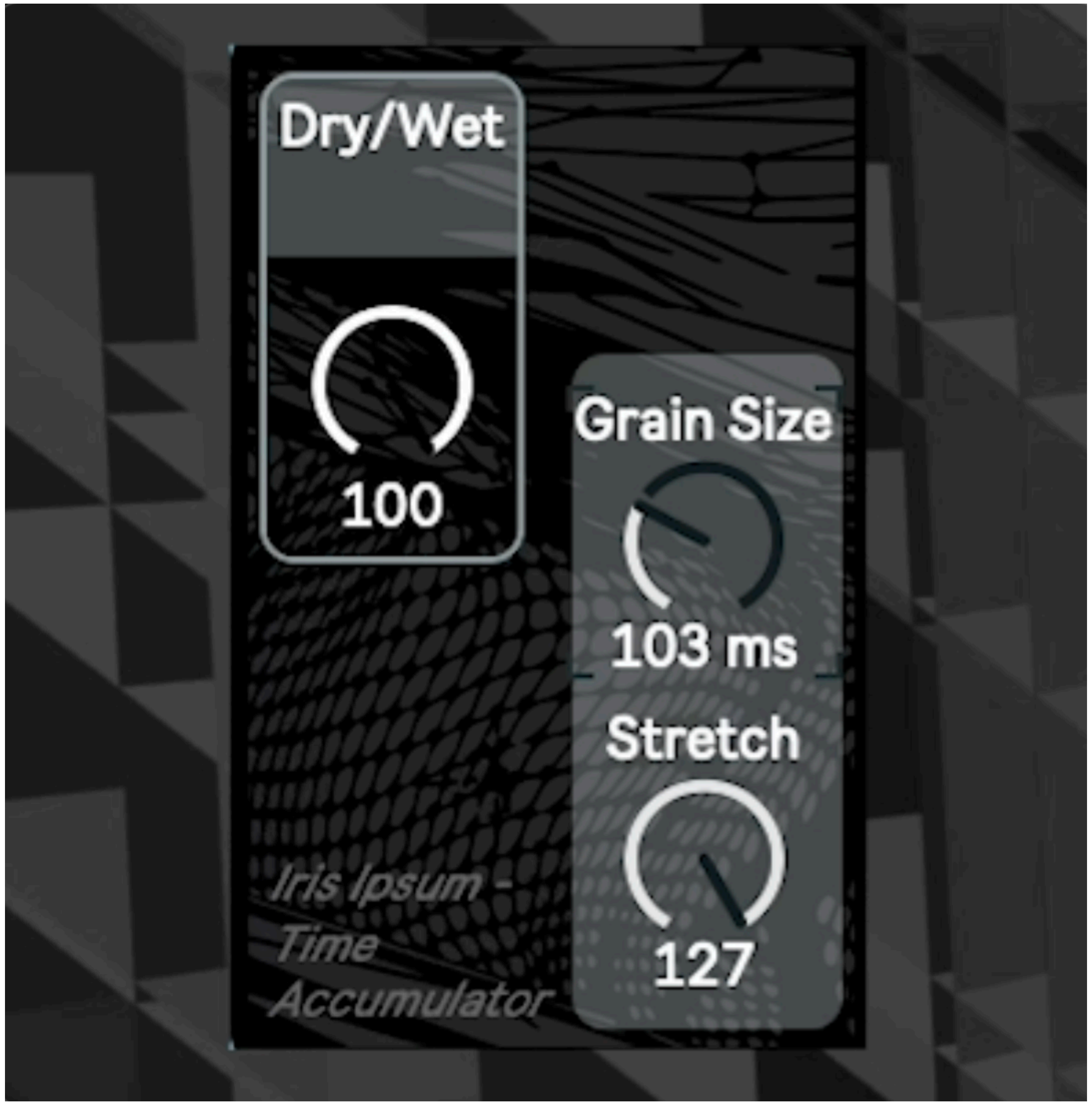
The device contains a row of 9 binary cells, each of which can be either high (1) or low (0). The next generation of the automaton is computed according to a user-selected rule. Each rule defines a distinct behavior pattern. Some produce stable patterns, some oscillate, and others generate unpredictable complexity.

Similar to Binseq and Euclidian Gate, instead of updating on an internal clock, Cellular Mod advances to the next generation whenever a MIDI note is received. This allows the automaton's evolution to be rhythmically precise and fully integrated into a performance or sequencer workflow. Every MIDI trigger shifts



the system forward by one generation, producing a new 9-bit pattern.

Each of the nine cells outputs a modulation signal that can be mapped to parameters across Ableton Live. Each binary cell has included rise and fall slope parameters with the option of logarithmic or linear change over time, as well as variable range settings. This makes the output smoother and more musically usable than raw binary on/off values, while still preserving the underlying generative structure of the automaton. [7]



Dry/Wet 100

Grain Size 103 ms

Stretch 127

1.6

Time Accumulator

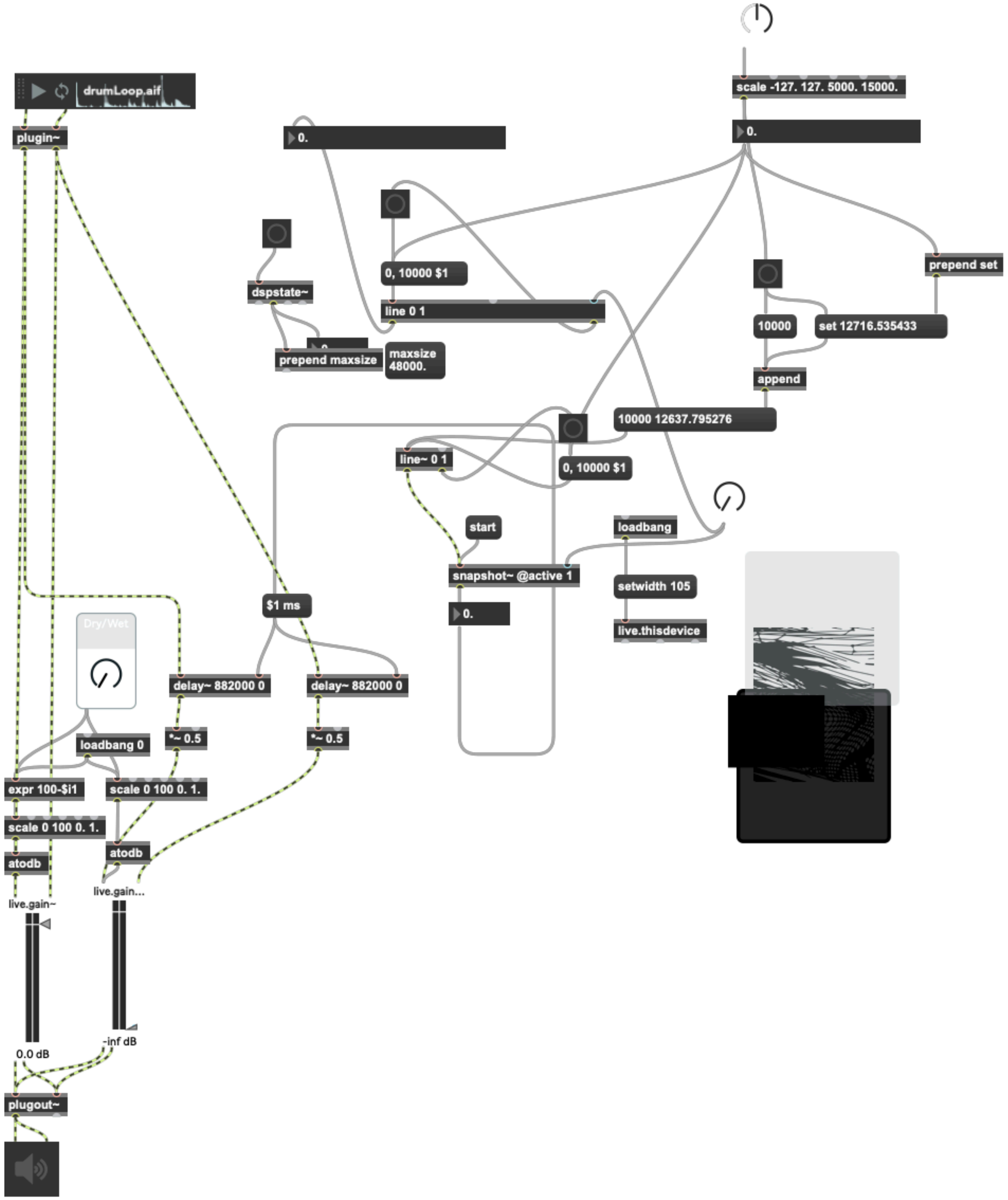
Time Accumulator is a real-time temporal processing device that generates stretched, frozen, and smeared audio textures by scanning through a continuously updated circular delay buffer. Rather than employing granular or frequency-domain techniques, the device performs its time manipulation entirely in the time domain by dynamically adjusting the read position of a long delay line.

Incoming audio is continuously written into a pair of 20-second delay buffers implemented using Max's delay~ object. A read head then retrieves samples from an offset position within this buffer. The output at each time sample can be defined as:

$$y[n]=buffer[n-r(t)]$$

where r(t) is a time-varying delay offset measured in samples. The motion of the read head is generated by a linear envelope produced by max's line~ object.

The stretch parameter determines the maximum temporal window the device scans through, while grain size determines the rate of traversal. Smaller stretch sizes produce rapid scans through the buffer (resulting in jittery micro-slices), whereas larger stretch sizes create slow, highly smeared temporal washes. Because the entire process occurs in the time domain

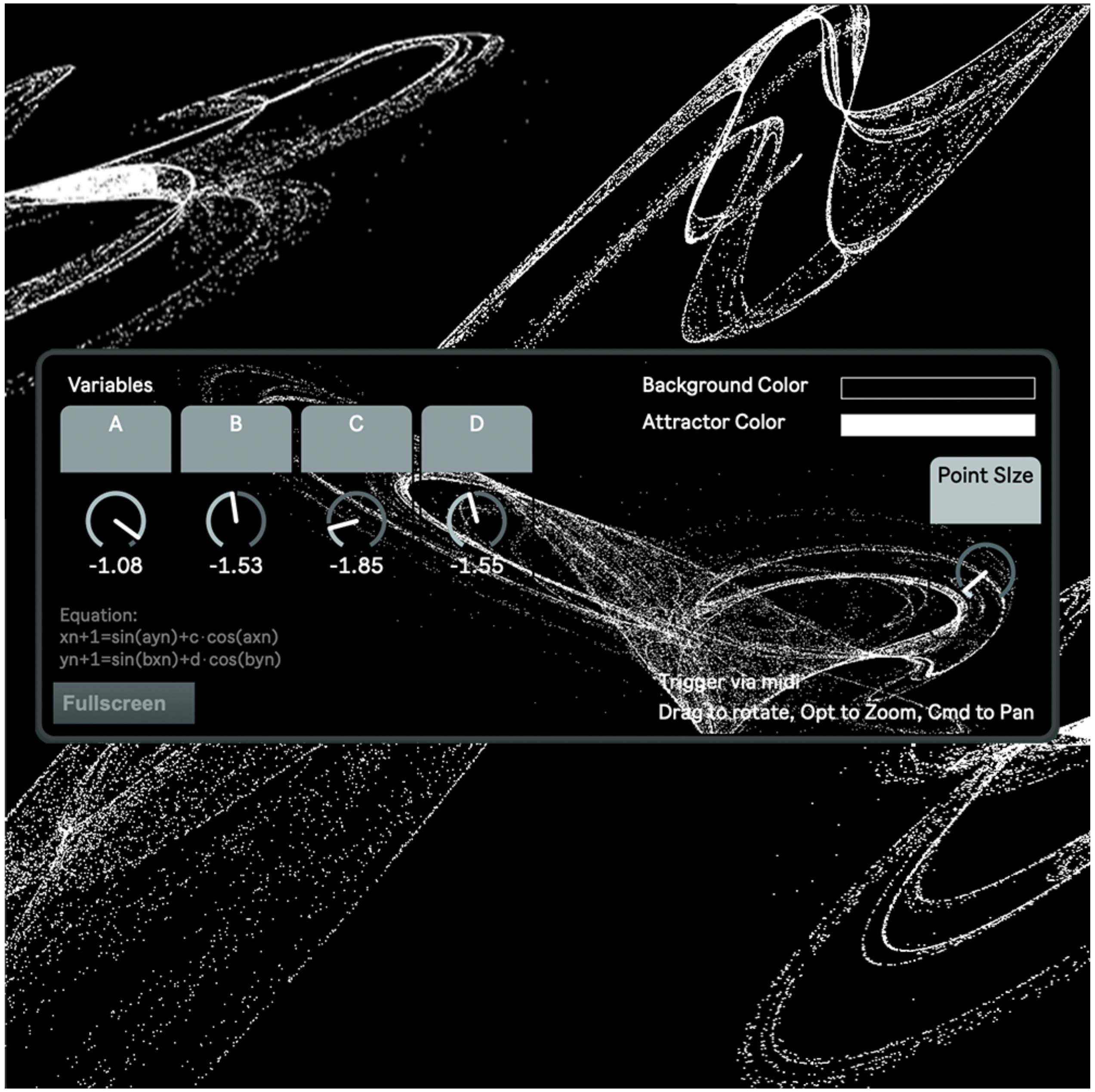


and uses smooth-linear interpolation, the resulting sound retains the original timbral identity while dramatically altering its temporal structure.

The overall effect is a fluid, continuously evolving temporal smear where the device effectively plays back a moving window of the recent past. By manipulating the speed and range of the read head, Time Accumulator produces textures ranging from subtle widening and softening to dense, evolving pads and frozen harmonic masses. This technique provides a computationally simple yet sonically rich alternative to granular or FFT-based time-stretching.

1.7

Fractal Dream



Variables

A: -1.08

B: -1.53

C: -1.85

D: -1.55

Equation:
xn+1=sin(ayn)+c·cos(axn)
yn+1=sin(bxn)+d·cos(byn)

Trigger via midi

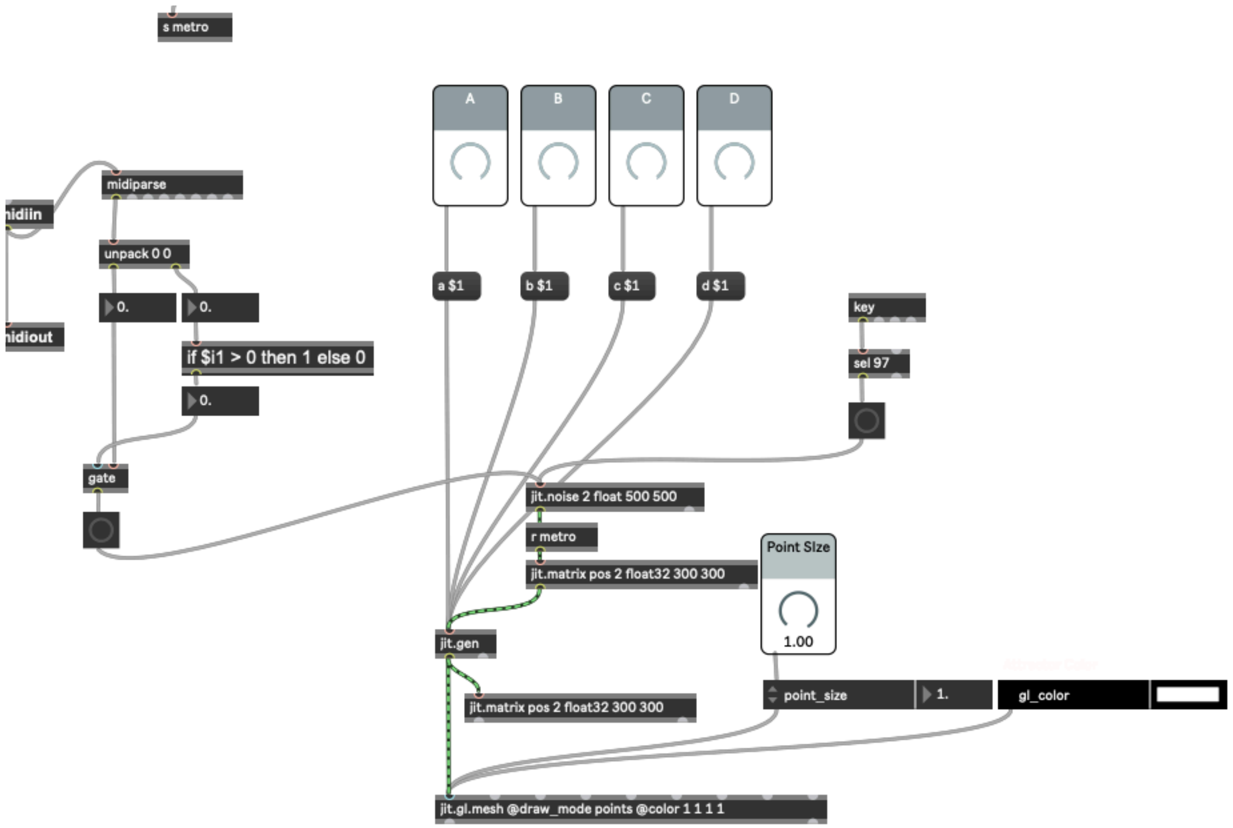
Drag to rotate, Opt to Zoom, Cmd to Pan

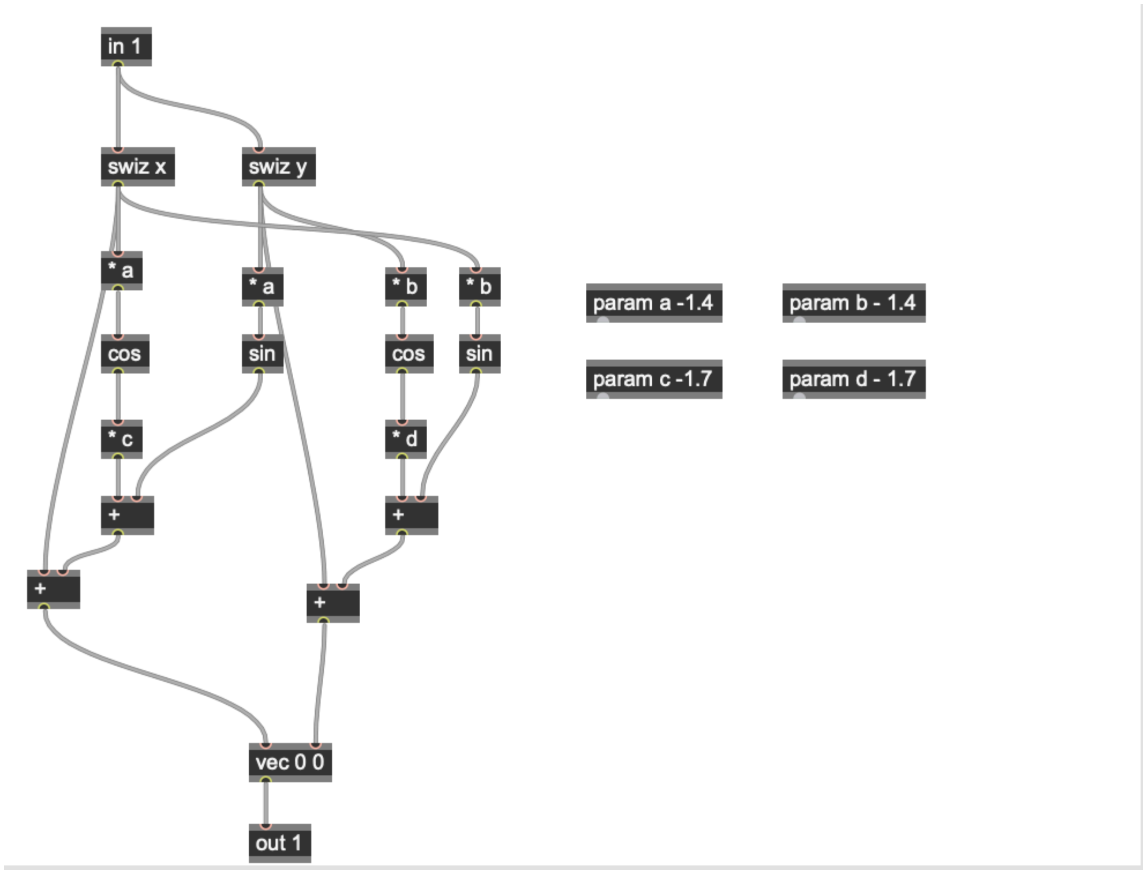
Fractal Dream is an Ableton Live visualizer that renders a two-dimensional strange attractor in real time and exposes its underlying parameters as performative controls. The patch is implemented in jit.gen and iteratively updates a point (xn,yn) according to a four-parameter non-linear map inspired by the fractal dream strange attractor, governed by the following equations:

$$xn+1= xn+\sin(a\,yn) + c\cos(a\,xn)$$

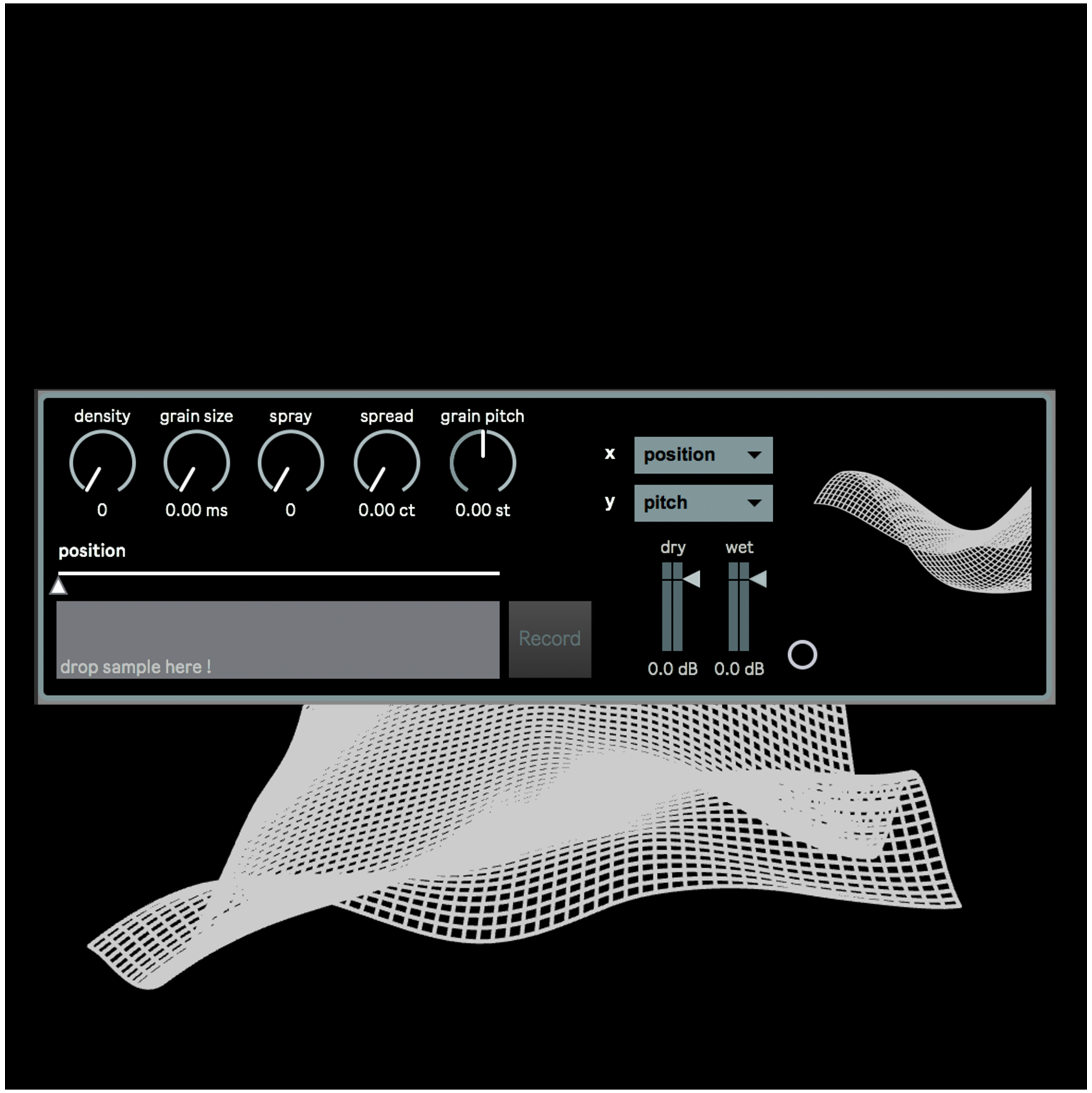
$$yn+1= yn+\sin(b\,xn) + d\cos(b\,yn)$$

In these equations, a, b, c and d are user-controllable coefficients. In the jit.gen patch, the incoming position vector is split into its x and y components using the swiz object. Each component is multiplied by the corresponding parameter (a or b), passed through both sine and cosine functions, and then scaled by the secondary parameters (c and d). The scaled cosine term and raw sine term are summed to create a non-linear offset, which is then added back to the original coordinate. The result is packed into a new vector and fed back into the system on the next iteration, producing the familiar dense, filament-like trajectories characteristic of strange attractors.





From a performance perspective, incoming MIDI notes act as triggers to reseed and reanimate the attractor. Each note can initiate a new trajectory or clear and redraw the point cloud, effectively tying the visual evolution of the attractor to the rhythmic structure of a performance or piece of music. Modifying the a, b, c and d parameters allow for continuous changes that dramatically reshape the attractor from tight, orbiting loops to chaotic, diffuse clouds.[5]


density 0
grain size 0.00 ms
spray 0
spread 0.00 ct
grain pitch 0.00 st
x position
y pitch
dry 0.0 dB
wet 0.0 dB

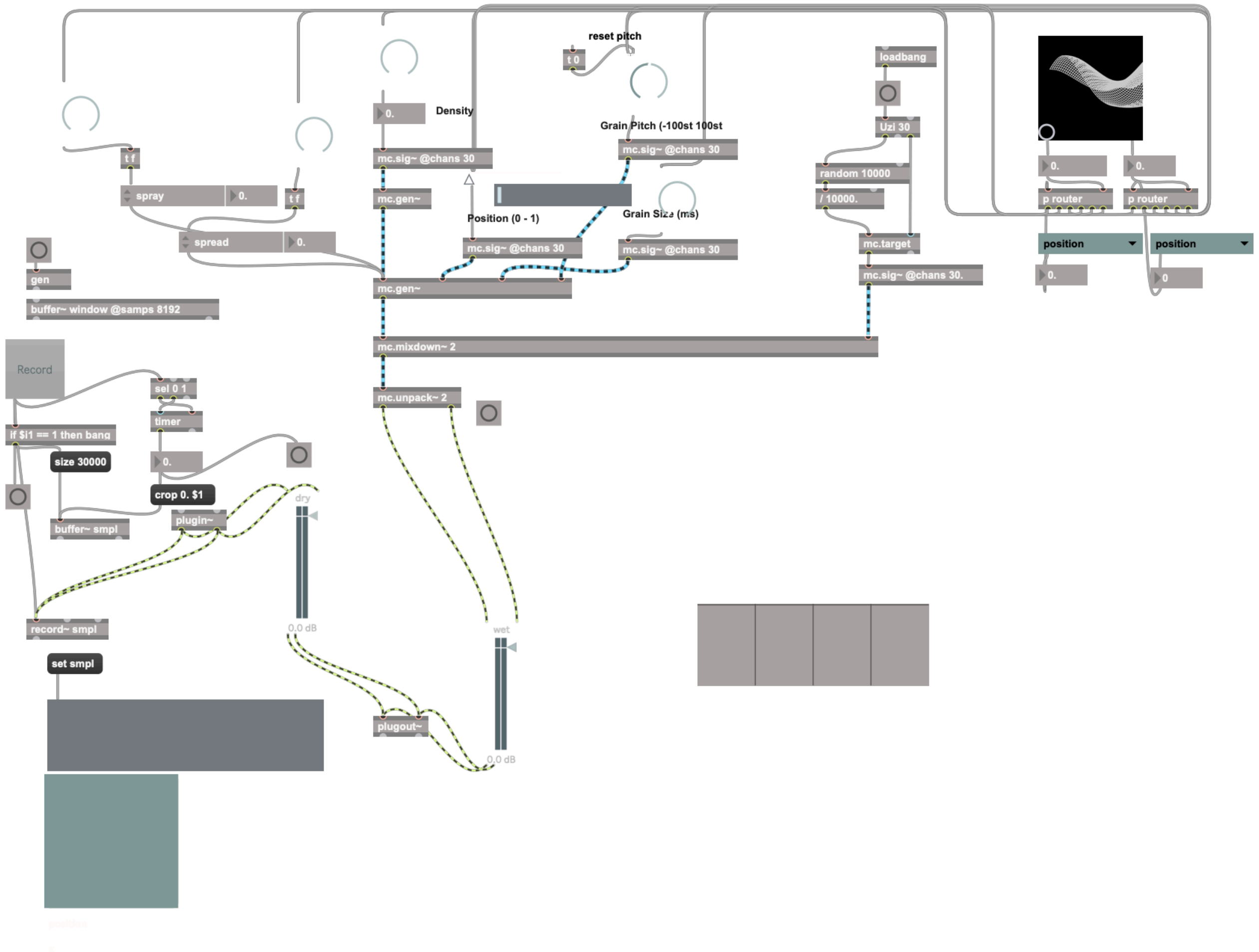
1.8

Quanta Granulator

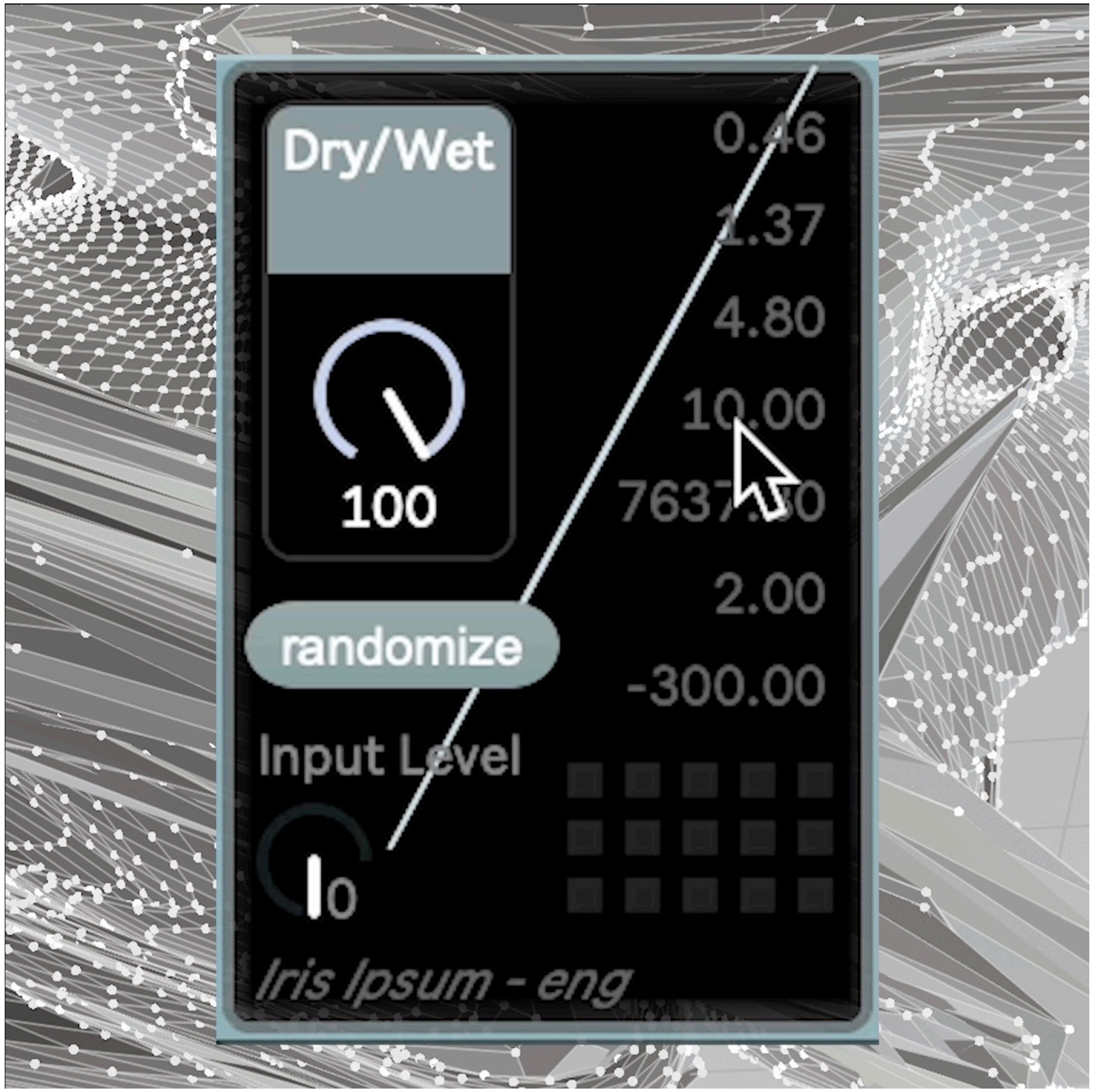
Quanta Granulator is a granular sampling and microsound processing device that can also function as a spectral freeze. The device operates by segmenting incoming or recorded audio into discrete grains and rearticulating them according to a set of controllable parameters. Quanta Granulator enables a wide range of temporal, spectral, and textural audio transmutations.

The density parameter controls the number of grains generated over time, directly affecting the perceived thickness and continuity of the output texture. Grain size determines the duration of each individual grain, spray introduces stochastic variation to the playback position of each grain, spread applies randomized pitch deviation across grains, and grain pitch sets the global transposition applied to the granular system.

In addition to these core parameters, the device includes a built-in two-dimensional XY control matrix, allowing pairs of parameters to be modulated simultaneously for expressive, performance-oriented interaction. A dry/wet control enables continuous blending between processed and unprocessed signal, and an integrated record function allows incoming audio to be captured directly into the internal buffer for immediate granular manipulation. Together, these features position Quanta



Granulator as a flexible tool for microsound composition, time-domain abstraction, and real-time spectral freezing.[16]



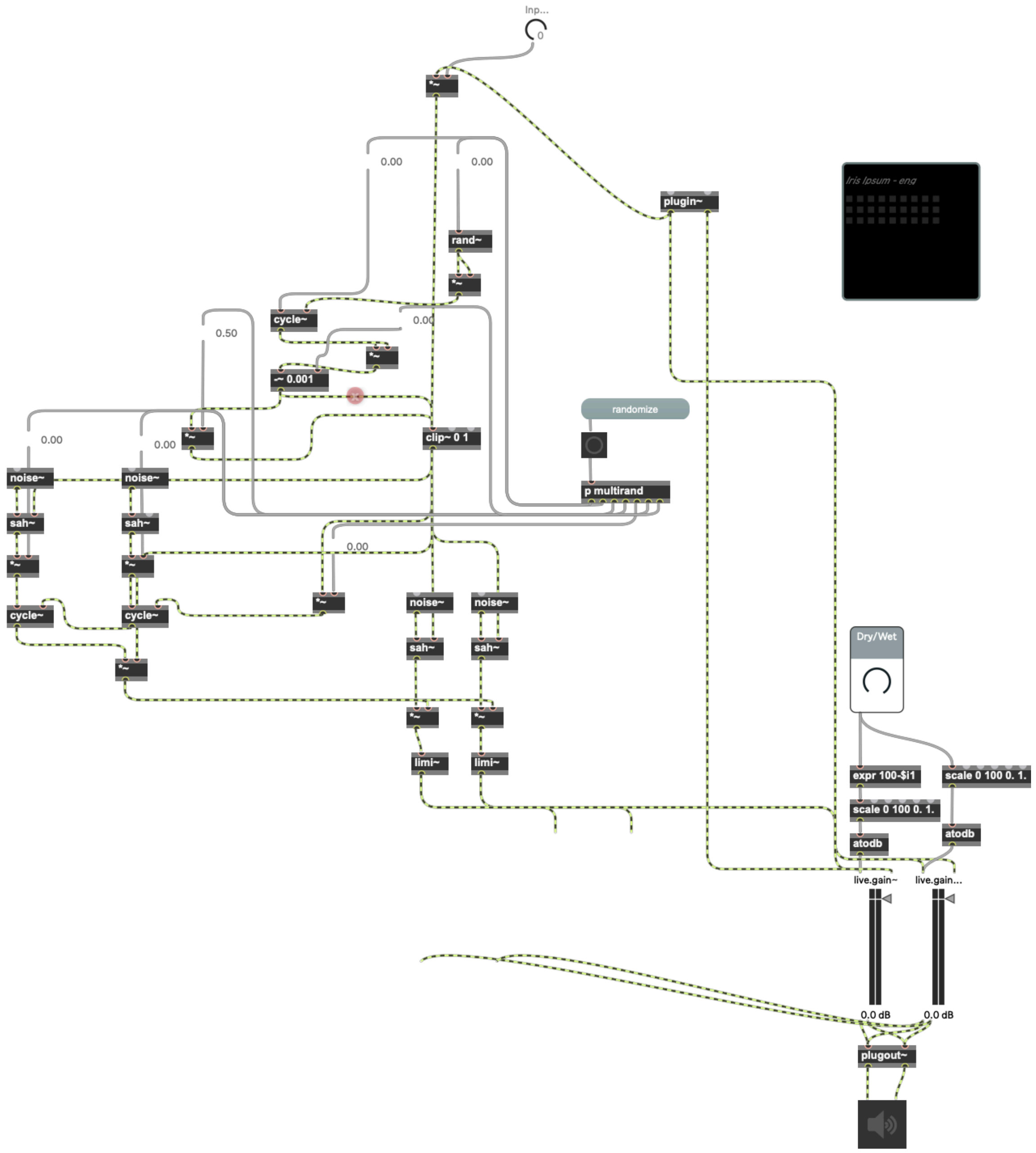
1.9

Esoteric Noise Generator

Esoteric Noise Generator is a stochastic noise generator and extreme distortion implemented as a Max for Live audio effect. The device functions through a network of interlocked noise sources, sinusoidal oscillators, and sample-and-hold units, all driven by the incoming audio's amplitude and a bank of randomizable control parameters. The result is a responsive noise instrument that can function both as an autonomous sound source and as a dynamically keyed layer over existing material.

Stereo audio enters the device and is immediately scaled by an input level control, implemented as a live.dial driving a *~ object. This gain-controlled input is then passed to a clip~ stage. In parallel, the unprocessed input is routed to a dedicated live.gain~ object to form the dry path.

The core of the devices behavior is built from several parallel chains of noise~, sah~, and cycle~ objects, combined through multiplication. Each chain begins with a white noise source (noise~) whose output is periodically sampled by sah~. The sampling input of each sah~ is driven by the clipped input signal, so the rate and pattern of value changes in the control streams are coupled to the dynamics of the incoming audio. The sampled values are then scaled by user-accessible



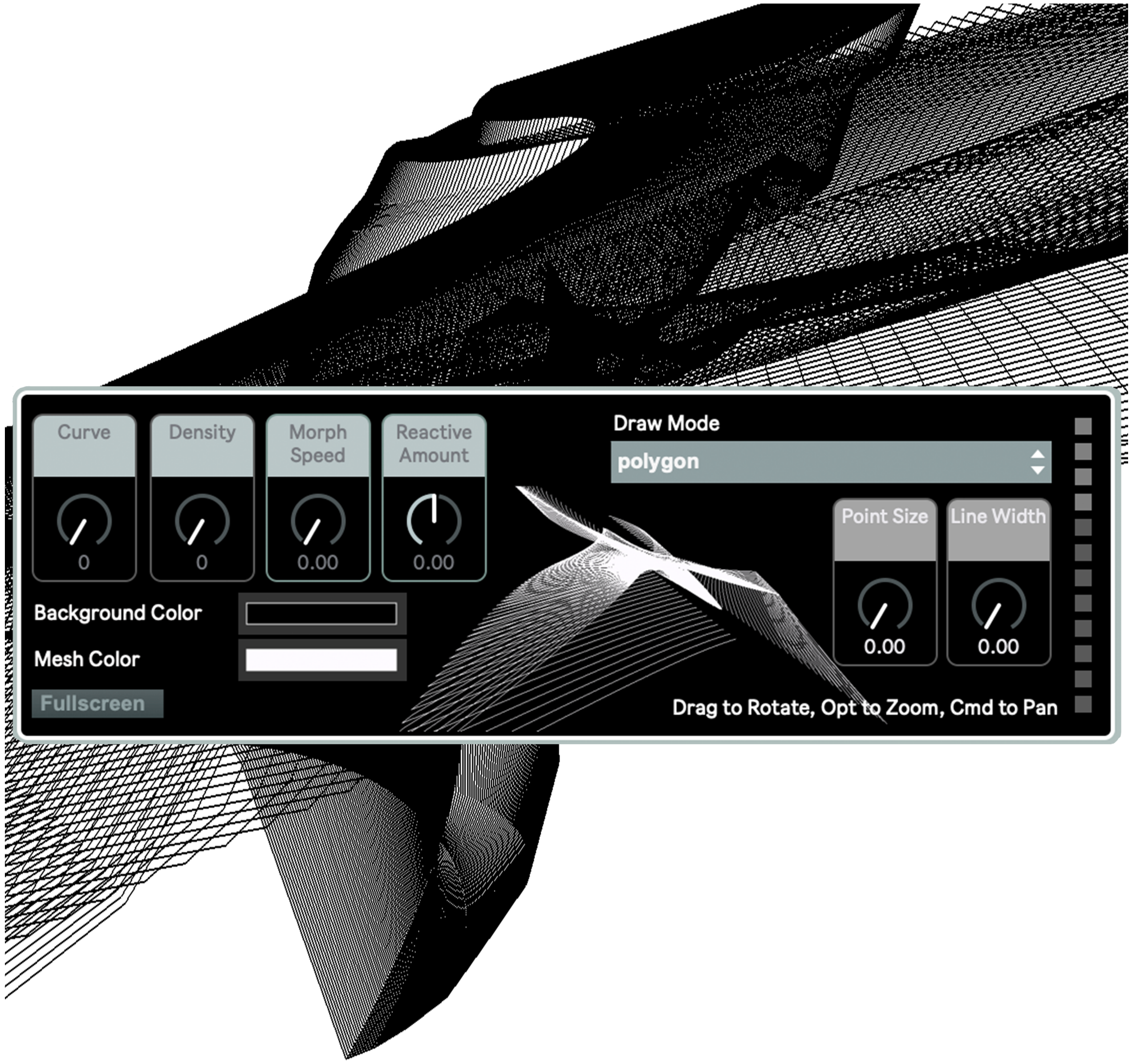
parameters and used in two main ways, both as a frequency control for cycle~ objects which function as noise-modulated sinusoidal oscillators, and as amplitude controls for various *~ stages, effectively acting as stochastic VCA envelopes.

In one branch, a rand~ object is multiplied by itself and used to modulate the frequency of a cycle~ unit, which creates a slowly wandering, quasi-chaotic low-frequency control signal that is then subtracted and re-scaled before being injected back into the main modulation bus feeding the clip~ object. Conceptually, this turns the device into a small feedback network: the audio input excites the modulators, but the modulators also self-condition the envelope that drives them.

Multiple noise/S&H oscillator paths are instantiated in parallel, feeding two limi~ objects (one per channel). These act as safety limiters on the wet noise output, ensuring that extreme parameter combinations and randomizations remain within a controlled dynamic range.

A key aspect of Esoteric Noise Generator is its dedicated randomization engine implemented through a multirand subpatch. Clicking the randomize text button triggers a button that bangs the multirand subpatch. Inside this subpatch a bank of random objects generate random integer values which are then mapped into musically meaningful domains via multiple scale objects, which are then routed to various parameters in the feedback network. A single click randomizes oscillator frequencies, modulation depths, offset ranges, and the internal nonlinear scaling factor that shapes the envelope feeding clip~.

Because of the complexity of each parameter, each randomization yields a unique and distinct characteristic.[2]



Curve 0
Density 0
Morph Speed 0.00
Reactive Amount 0.00
Draw Mode polygon
Point Size 0.00
Line Width 0.00

Drag to Rotate, Opt to Zoom, Cmd to Pan

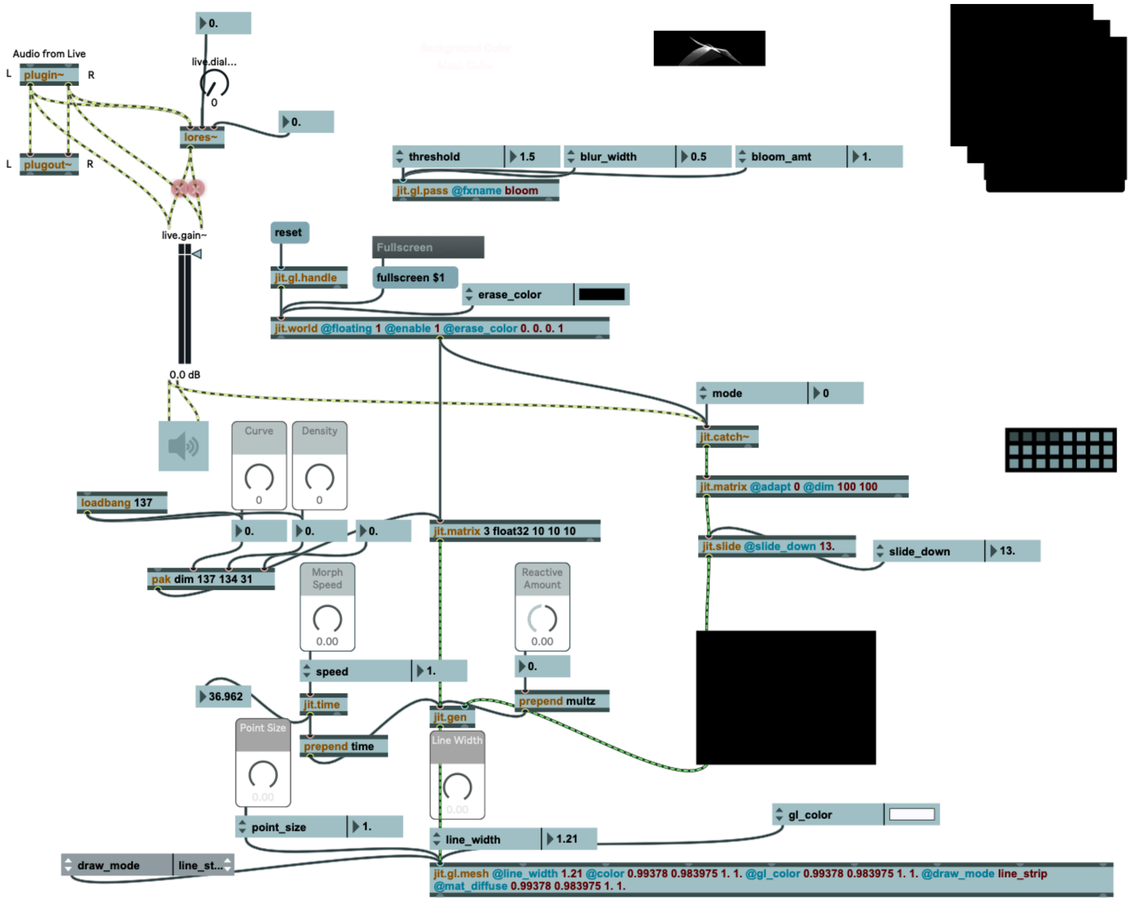
1.10

Mesh Visualizer

Mesh Visualizer is an audio-reactive Max for Live Device that uses audio from Ableton Live to modulate a generative mesh lattice. The device analyzes incoming audio and maps amplitude and spectral information to parameters governing the deformation, density, and motion of a three-dimensional mesh structure, producing visuals that evolve in direct correspondence with the sonic material.

Implemented using Max's Jitter framework, the patch operates by routing audio-derived control signals into a matrix-based mesh system, where continuous parameter modulation results in fluid, time-varying transformations of the lattice. This approach allows rhythmic, timbral, and dynamic characteristics of sound to be rendered visually, reinforcing the relationship between auditory and visual domains within a performance or compositional context.

The design of Mesh Visualizer is informed by prior generative visualization practices, including the work of Duncan Wilson, and was developed through the adaptation and extension of existing educational resources on audio-reactive mesh synthesis in Max. By integrating visual generation directly into the Ableton Live environment, the device functions as both a real-time performance visualizer and an exploratory tool for studying the coupling between sound, motion, and form.[14]



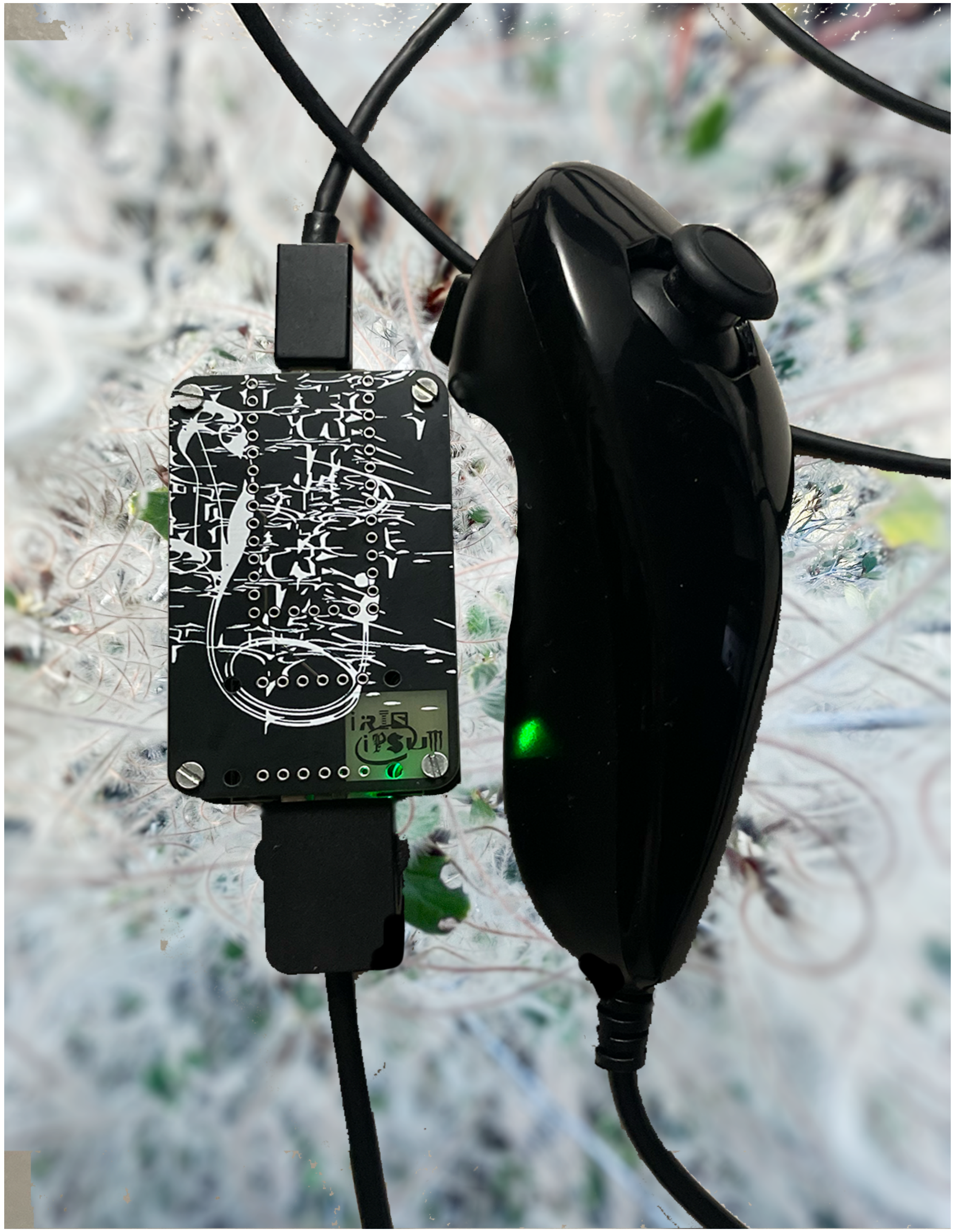
```
plugin~
plugout~
lores~
jit.gl.pass @fxname bloom
jit.gl.handle
fullscreen $1
jit.world @floating 1 @enable 1 @erase_color 0. 0. 0. 1
jit.catch~
jit.matrix @adapt 0 @dim 100 100
jit.slide @slide_down 13.
loadbang 137
pak dim 137 134 31
jit.matrix 3 float32 10 10 10
jit.time
prepend time
jit.gen
prepend multz
jit.gl.mesh @line_width 1.21 @color 0.99378 0.983975 1. 1. @gl_color 0.99378 0.983975 1. 1. @draw_mode line_strip
@mat_diffuse 0.99378 0.983975 1. 1.
```

threshold 1.5
blur_width 0.5
bloom_amt 1.
erase_color
mode 0
slide_down 13.
speed 1.
point_size 1.
line_width 1.21
gl_color
draw_mode line_st...

Section 2

Midi Controllers

The following section covers a series of midi control devices built using Teensy microcontrollers and custom printed PCBS, intended to provide novel control interfaces for digital audio systems. I have distributed and sold these devices in small scales both preassembled and as kits.



2.1

Nunchuck Controller

Nunchuck Controller is a gestural MIDI interface that repurposes a Nintendo Wii Nunchuck as a compact, one-handed control surface for digital audio systems. The device exposes the nine continuous control sources available on a nunchuck, including a two–axis joystick (X, Y), three axes of acceleration (accel X, Y, Z), and two buttons (C and Z). Each stream of sensor data is translated into MIDI Control Change (CC) messages that can be mapped to any parameter in DAWs or stand alone instruments, allowing performers to shape sound through small, intuitive hand movements rather than mouse or trackpad gestures.

The hardware consists of a custom PCB hosting a Teensy 3.1 microcontroller and an Adafruit Wii Nunchuck Breakout Adapter. The firmware, written in Arduino C++ using the *WiiChuck* and Teensy usbMIDI libraries, continuously polls the Nunchuck over I²C and unpacks the raw sensor array into individual channels. To reduce jitter and make the motion feel smooth, each accelerometer and joystick stream is low-pass filtered using a simple running average:

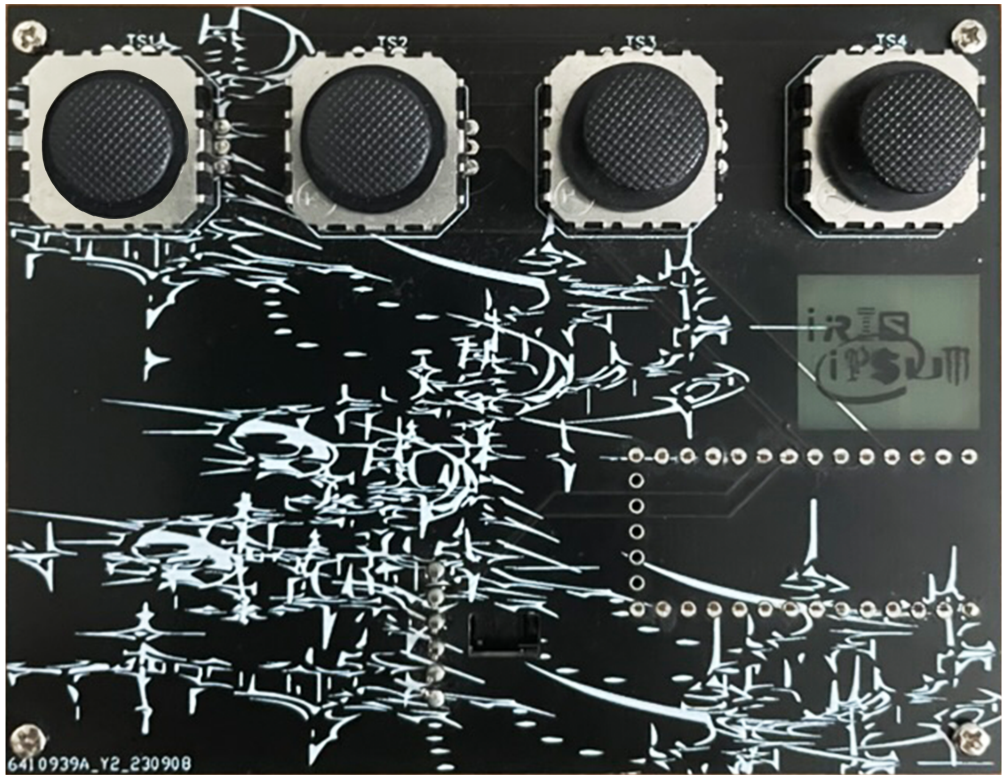
$$x_{\text{lp}}(n) = \frac{x_{\text{prev}}(n-1) + x_{\text{raw}}(n)}{2}$$

where xraw is the latest unfiltered reading and xlp is the smoothed value stored as the new "previous" sample.

After filtering, all values are linearly mapped and constrained into the standard 7-bit MIDI range [0,127]. For example, joystick values arrive as 8-bit integers [0,255] and are scaled to 127 utilizing the following Arduino code:

$$x_{\mathrm{MIDI}} = \mathrm{constrain}\big(\mathrm{map}(x_{\mathrm{raw}}, 0{,}255, 0{,}127), 0, 127\big)$$

The firmware also performs basic noise-gating and change-detection. CC messages are only sent when a parameter moves beyond a small threshold (e.g. a difference of more than one step for the joystick, or a minimum delta in the accelerometer data). This prevents unnecessary MIDI traffic and makes the device feel responsive rather than noisy. Each sensor is routed to a fixed CC number (e.g. joystick X/Y on CC 102–103, accelerometer axes on CC 106–108, buttons on CC 104–105) on a dedicated MIDI channel, so mappings can be recalled consistently across projects.[18]



2.2

4Joy

4Joy is a compact gestural MIDI controller designed to maximize expressive multidimensional control through minimal motion of a single hand. The device captures nine continuous control streams using four miniature two-axis joysticks and a single time-of-flight distance sensor. Like Nunchuck Controller, it is built around a Teensy microcontroller and two custom-designed printed circuit boards. Each joystick provides independent X and Y control parameters, and the ToF sensor provides a distance based control channel. Together these sensors create a dense and high-resolution gestural interface suitable for performance, sound design, and live modulation workflows.

At the firmware level, each joystick axis is sampled as an analog input, then linearly mapped into the standard 7-bit MIDI CC range.

To counteract jitter and sharpen the response of the physical hardware, each axis is processed with the same lightweight low-pass filter used in the Nunchuck Controller:

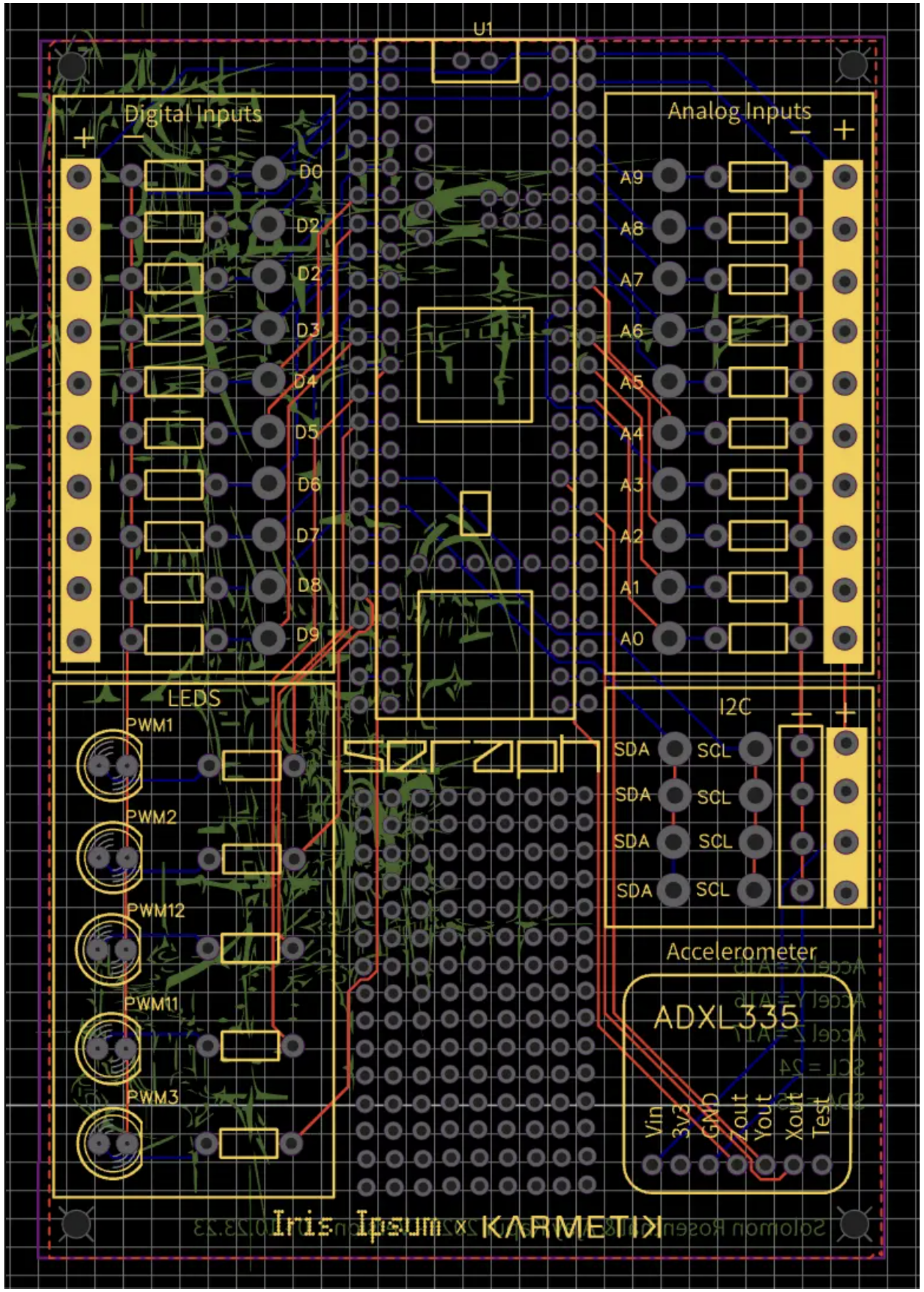
$$x_{\text{lp}}(n) = \frac{x_{\text{prev}}(n-1) + x_{\text{raw}}(n)}{2}$$

The time-of-flight sensor (a VL6180X) is sampled over I²C and filtered using the same scheme, then compressed into a musically useful range.

To keep the device responsive but not noisy, 4Joy uses a dead-zone strategy around the joystick’s mechanical center. For each axis, the firmware checks whether the filtered value has moved sufficiently far from the central region (approx. 60–66). Only when the gesture exceeds this threshold does the firmware transmit a MIDI CC message. This ensures expressive resolution when the performer intends to move, but prevents the analog noise floor from flooding the host with unnecessary data.

Each control source is routed to a fixed, predictable CC assignment (CC 16–23 for the four joysticks and 24 for the ToF sensor). This consistency enables reproducible mappings across DAWs and other digital environments. All nine channels operate on a single MIDI port and can be played simultaneously,

4Joy is a small-format but highly articulate gestural instrument. Its ability to shape nine parameters at once with one hand offers a unique performance vocabulary, particularly suited for electronic musicians seeking continuous, fluid and dynamic modulation without relying on large control surfaces or touchscreens. [18]



2.3

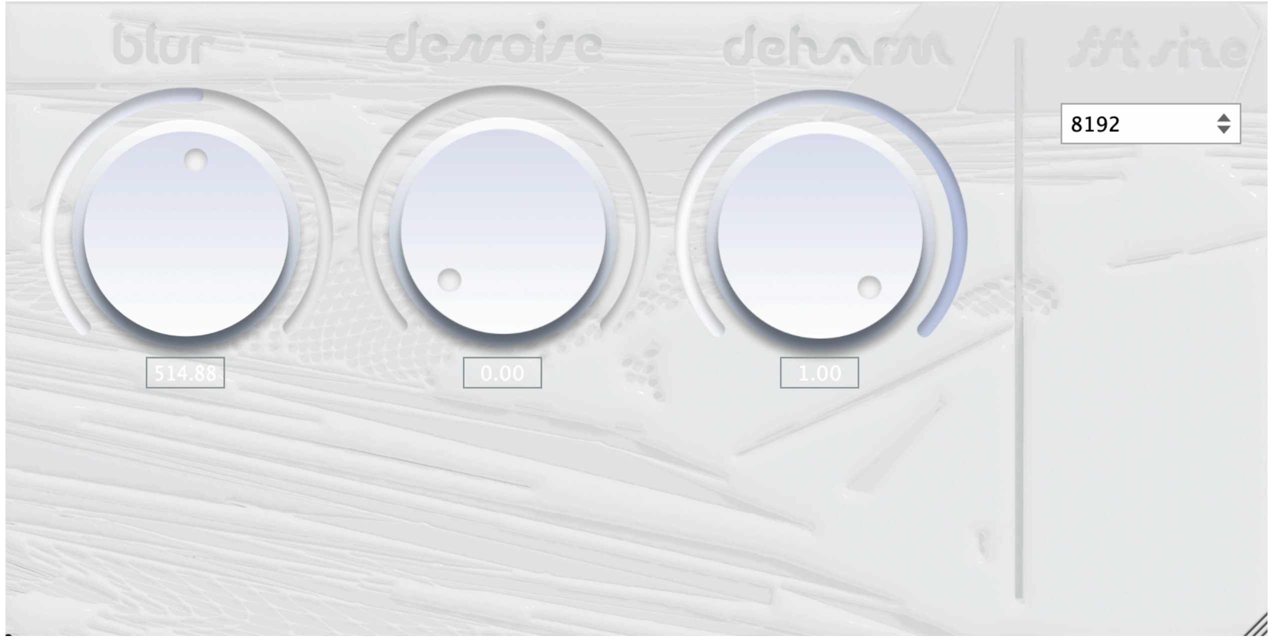
Seraph

Seraph is an open-source hardware and software platform intended to facilitate the rapid creation of USB-MIDI controllers using the Teensy 4.1 microcontroller. The platform comprises a purpose-built PCB breakout board and a complementary firmware/codebase leveraging the Teensy USB-MIDI library. The breakout board routes input/output pins on the teensy for peripherals including but not limited to buttons, potentiometers, LEDs, and 12C sensors, and exposes the microcontroller's full pin-set for easy expansion, while the codebase provides a modular framework for mapping physical controls to MIDI messages, and supports customization for musical controllers. The Seraph platform is intended to reduce development and prototyping time, and lowers the barrier of entry for experimental MIDI hardware projects. Because the full Teensy pin-set is exposed and the code mapping is modular, users can adapt the board for many controller formats. The open-source documentation encourages remixing and derivative builds. For the past two years the board has been distributed to students in the Interface Design class at Calarts, who have utilized it as an educational resource and conducted a variety of prototype builds. Seraph was developed at CalArts over several years by myself and Ajay Kapur. [18]

Section 3

VSTs

The following section covers a series of VST audio effect plugins built in JUCE C++, which are designed to produce novel and interesting audio transmutations for experimental sound design and production practices.

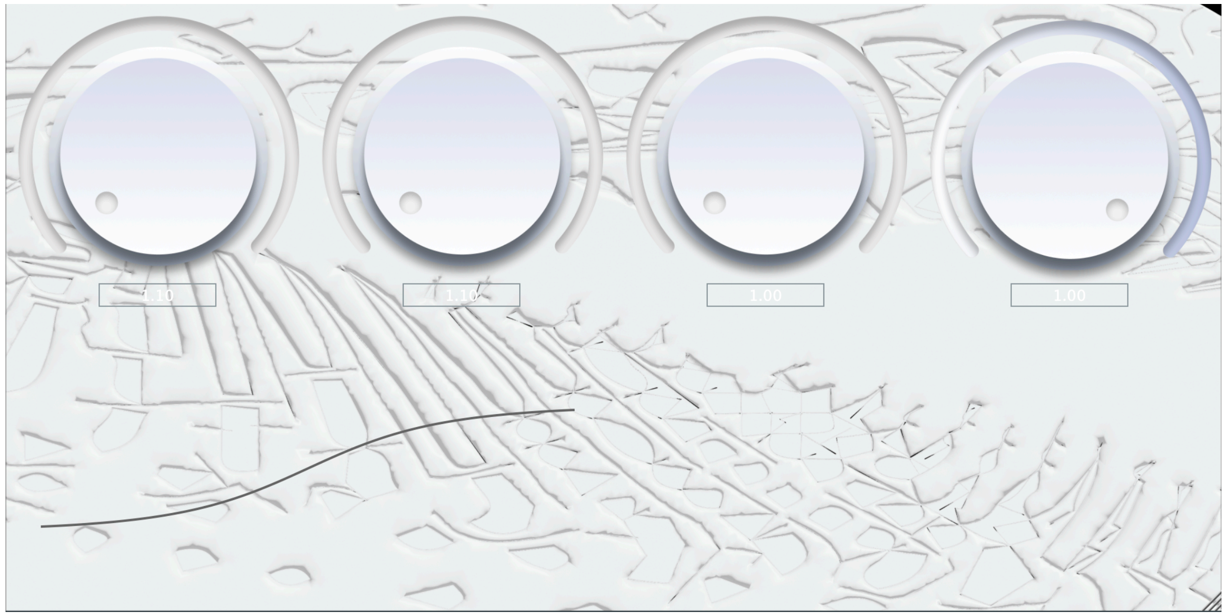


blur 514.88
denoise 0.00
deharm 1.00
fft size 8192

3.1

PitchBlur VST

Pitchblur VST is an FFT-based spectral multi-effect processor that operates via a Short-Time Fourier Transform (STFT) of its incoming audio signal, and as the name suggests, is essentially a VST port of the previously mentioned Pitchblur Max for Live device, built in JUCE C++. The plugin can be used in Windows, Mac and Linux operating systems either as a standalone application or within a DAW. In terms of DSP, each incoming block of samples is windowed and transformed into the frequency domain, where per-bin magnitude and phase are manipulated before being resynthesized via an overlap–add function. Like its Max for Live equivalent, the spectral blur stage performs phase multiplication across successive FFT frames creating diffuse, reverb-like clouds of partials whose character depends strongly on the chosen FFT size. The denoise and deharm processes implement complementary per bin amplitude-based gating. For the denoise function, bins whose magnitudes fall below a user-defined threshold are attenuated or removed, suppressing low-level noise and room tone, while for the deharm function, the inverse mapping is applied so that quieter, noise-like components are emphasized and more tonal, harmonic partials are reduced. Together, these operations allow the plugin to traverse a large range of spectral effect processes, seamlessly moving between highly abstracted and decorrelated textures all while preserving the phase coherence sufficient for stable resynthesis. The FFT algorithm used in this device was borrowed from the audio dev blog: https://audiodev.blog/fft-processing/.[8]



3.2

RMDistortion

RMDistortion is a JUCE C++ based nonlinear distortion processor designed to produce extreme harmonic spectral instability through a nested waveshaping function. Rather than relying on conventional clipping or smooth saturation curves, the processor implements a compound nonlinear transfer function defined as:

$$y = tanh(sin(tanh(x3 \cdot s1) \cdot s2) \cdot s3)$$

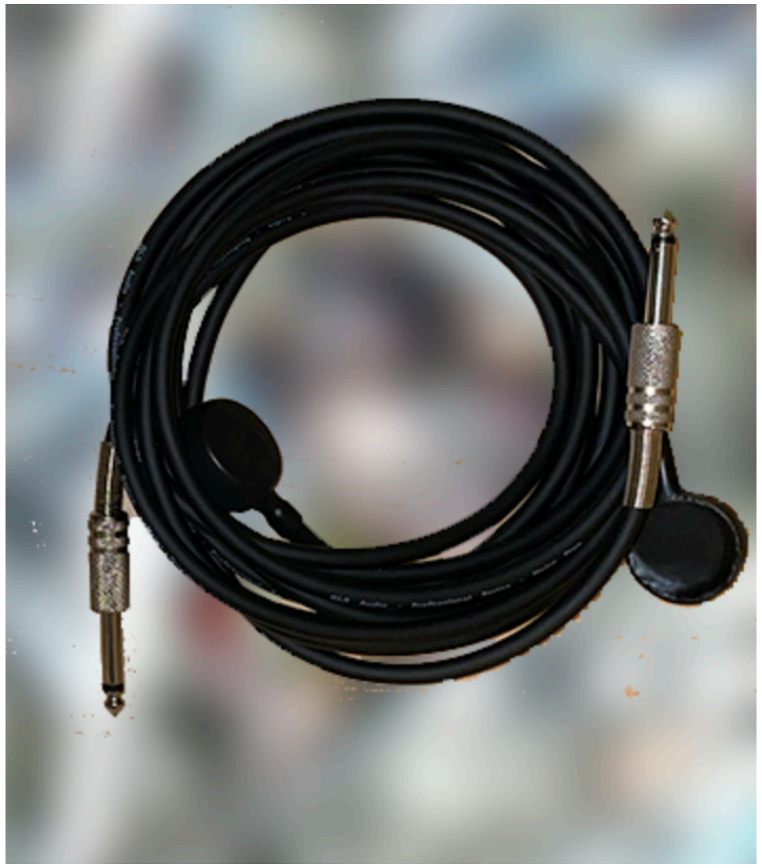
where x is the input sample, and s1,s2 and s3 correspond to user-controlled shaping coefficients.

The shaping pipeline operates in three conceptual stages: first the incoming signal is cubed (x3), exaggerating dynamic asymmetry. This term is then scaled by s1 before passing through an inner hyperbolic tangent layer, which reduces numerical runaway while retaining some curvature. This intermediate output is then scaled again and passed through a sine mapping, introducing periodic folding and discontinuous zero-cross deviations. This stage acts like a wavefolder, but with a smooth central response that excites chaotic behavior at higher drive levels. A final scaling and tanh function bound the output, preventing digital clipping while maintaining the harsh and harmonically rich structure created upstream. The device produces extreme and unpredictable distortion effects, and with four relatively unmarked and esoteric parameter control inputs, it allows the user to explore its chaotic and unstable processing behavior intuitively through a process of listening. The plugin is compatible in Mac, Windows and Linux operating systems and can be used as a standalone application or as a plugin inside of a DAW. The devices custom UI and animated parameter knob were designed in Illustrator and Figma.

Section 4

Microphones

The following section documents a series of experimental microphones I have designed, manufactured, and distributed, each intended to extend human auditory perception into typically inaccessible domains. Whereas conventional microphones capture air-borne pressure fluctuations, these devices are engineered to couple with non-traditional mediums such as water, electromagnetic fields, and mechanical resonances, and through a variety of processes, translate these vibrational activities into audible signals. By treating the microphone not only as a recording tool but as a means of revealing otherwise imperceptible phenomena, these designs function as instruments of sonic inquiry, expanding the palette of field recording, sound design, and electronic music practices.



4.1

Hydrophones

My custom hydrophones are built from waterproofed piezoelectric disk transducers designed to capture pressure waves in underwater environments. Each piezo element is housed inside a custom 3D-printed enclosure which is subsequently filled with a casting resin formulated to match the density of water. Utilizing resin of this specific density ensures that the hydrophone accurately reproduces underwater sound with reduced spectral deviation, distortion, or muffling.

To ensure stable submersion, a weighted steel nut is embedded within the enclosure. This component adds mass to the microphone, allowing it to sink reliably and remain positionally stable in moving water. After assembly, the enclosure is coated in multiple layers of PlastiDip to create a fully waterproof and electrically isolated seal capable of withstanding long-term immersion.

These hydrophones are constructed using high-grade shielded and gold-plated microphone cable to minimize interference and maintain signal integrity over long cable runs, and I have produced both balanced XLR and unbalanced ¼ inch versions. These hydrophones are intended to be accessible tools for capturing the dense and often overlooked sonic worlds of underwater environments and ecosystems.

4.2

EMF Mics

My EMF microphones (electromagnetic field microphones) are designed to detect electromagnetic fields rather than acoustic vibrations, translating invisible EM activity into audible signals. Each microphone consists of a tightly wound copper coil (either air-cored or wrapped around a small ferromagnetic core) soldered directly to a shielded microphone cable and terminated at a 1/4-inch jack.

Movement of alternating electromagnetic fields near the coil induces a small electrical current, which is then amplified and monitored like any other audio source. These microphones allow the user to listen to circuitry, power transformers, LEDs, household appliances, computers, motors, and any other EM-active device, revealing hidden phenomena and sonic information embedded in technological infrastructures. These devices serve as both investigative tools and creative instruments, enabling the recording and perception of electromagnetic activity that typically remains outside the limited domain of human perception.

Section 5

Conclusion

This thesis documents my recent work in the development of a diverse set of audio and production tools, and positions toolbuilding as an integral component of my creative practice. Rather than treating instruments, software, and interfaces as neutral or fixed, I hope to approach them as sites of artistic inquiry in their own right. Through the process of designing and constructing my own tools, I have been able to shape the conceptual and procedural frameworks through which I make work. Toolbuilding functions here as a creative act, and resists reliance on standard or industrialized workflows. The tools presented in this thesis reflect an ongoing commitment to toolbuilding as an artistic approach, and serve as both practical instruments and artifacts of evolving research in the area.

An important aspect of this project is that many of the tools described in this thesis are not only built for personal use, but are also freely distributed to a broader community of artists. Through releasing software instruments, hardware devices, and open-source platforms, my practice of toolbuilding extends beyond individual authorship and into a shared creative ecosystem. Free distribution allows these tools to circulate, be adapted, and generate outcomes that exceed the scope of a single practice.

Making tools available to others introduces an additional layer of responsibility. Designing for distribution requires clarity, robustness, and usability, but also invites unforeseen interpretations and uses. In this way, each tool functions less as a finished and discreet product and more as a framework for creative possibility, one that enables other artists to explore sounds, workflows, and ideas that may not have been possible otherwise.

Engaging in the distribution of creative tools reframes toolmaking as a social act. It positions the artist not only as a producer of work, but as a contributor to the conditions under which creative work can take place. By sharing these tools, this project embraces a collaborative ethos in which technical knowledge, experimentation, and curiosity are multiplied across a community of users, with a goal of pushing the envelope of what is possible in the realm of creative sound practices.

Section 6

References

1. Bayle, J. (2025). *The Art of Max for Live*.

2. Cipriani, A., Giri, M., & Richard, D. (2020). *Electronic Music and Sound Design: Theory and practice with Max 8. Volume 2*. ConTempoNet.

3. Cipriani, A., Giri, M., & Richard, D. (2010). *Electronic Music and Sound Design: Theory and practice with Max MSP*. ConTempoNet.

4. Cycling 74. (2017, March 14). *Advanced Max FFTs, Part 3*. YouTube. https://www.youtube.com/watch?v=1TDUqeqrg9k&list=PLasl9I6VeCCo4ASETZ5fgcflFx09ZJ6EJ

5. Dunn, R. (2019, March 5). *Clifford attractor tutorial Max/MSP/Jitter*. YouTube. https://www.youtube.com/watch?v=Xqlmd3i24VU

6. Eldem, U. (2021, July 27). *Multichannel Harmonic Additive Synthesis - Max/MSP Tutorial*. YouTube. https://www.youtube.com/watch?v=x6ZgoKPQGJk

7. Weisstein, Eric W. *Elementary cellular automaton*. from Wolfram MathWorld. (n.d.). https://mathworld.wolfram.com/ElementaryCellularAutomaton.html

8. Hollemans, M. (n.d.). *FFT processing in Juce*. Audio Dev Blog. https://audiodev.blog/fft-processing/

9. Horner, A., & Ayres, L. (2002). *Cooking with csound. part 1, woodwind and brass recipes*. A-R Editions.

10. Meyer, P. (2022, September 26). *Euclidean Rhythms in Max MSP*. YouTube. https://www.youtube.com/watch?v=G3uaqN7Glx8

11. Roads, Curtis. (2023). *The Computer Music Tutorial*. Second edition. Cambridge, Massachusetts: The MIT Press.

12. Roads, Curtis. (2015). *Composing Electronic Music : A New Aesthetic*. Oxford: Oxford University Press.

13. Roads, Curtis. (2001). *Microsound*. Cambridge, Mass.: MIT Press

14. Rush, N. (2021, November 11). *Max MSP Tutorial - Audio Reactive Mesh = Ned Rush*. YouTube. https://www.youtube.com/watch?v=nhLVLCK_90o

15. Taylor, G. (2018, August 20). *Step by step: Adventures in sequencing with Max/MSP*. Cycling '74. https://cycling74.com/books/step-by-step

16. ToneParticle. (2021, March 14). *Max/MSP Tutorial: Building a granular synthesiser in Max with mc objects and gen~*. YouTube. *https://www.youtube.com/watch?v=rbeLoYrdyPc*

17. Scoates, C. (2019). *Brian Eno: Visual music*. Chronicle Books.

18. *Using USB MIDI*. PJRC. (n.d.). https://www.pjrc.com/teensy/td_midi.html

19. Wakefield, G., & Taylor, G. (2022a). *Generating sound & organizing time: Thinking with gen~ book 1*. Cycling '74.

20. Wakefield, G., & Taylor, G. (2022b). *Generating sound & organizing time: Thinking with gen~ book 2*. Cycling '74.