Signal³ – Expanded Technical Configuration & Live Performance Guide
Author: César Martins Lima

This document is a comprehensive technical reference for the use of Signal³ in professional audiovisual performance contexts. It is intended to be read not only as a setup manual, but as an explanatory framework that clarifies why specific configuration choices matter when reliability, timing precision, and expressive control are required on stage. Signal³ should be approached as a performance-critical, real time visual instrument whose behavior must remain stable and predictable across rehearsals, rehearsals-to-show transitions, and extended live sets.[1]

---

1. System Architecture and Conceptual Overview

At its core, Signal³ is designed as a deterministic translation layer between musical intent and visual manifestation. In contrast to reactive audiovisual tools that rely on audio analysis alone, Signal³ operates on explicit control data in the form of MIDI messages, ensuring that every visual event is directly tied to a deliberate musical gesture rather than a post hoc interpretation of an audio stream. This architectural decision provides predictability, repeatability, and compositional intent, allowing the performer to sculpt visuals with the same precision used to sequence sound.[1]

Determinism in Signal³ means that given the same ordered sequence of MIDI messages, with the same timestamps and parameter values, the engine should always produce the same visual output, independent of transient system load variations. Internally, the system treats each MIDI event as an atomic unit that passes through a predictable processing pipeline: input capture, timestamp alignment, parameter mapping, and GPU dispatch. This contrasts with stochastic or sensor driven visual systems, where internal randomness or noisy analysis stages prevent exact repetition of results.

Within a live performance system, Signal³ occupies a central role. It receives discrete musical events from upstream devices or software, interprets their temporal and dynamic properties, and converts them into visual structures rendered in real time by the GPU. The success of this process depends not only on low latency, but on consistency of latency. Even minimal timing variance (jitter) can introduce perceptual misalignment between sound and image, which the human visual system detects almost immediately. For rhythmic material, a few milliseconds of jitter can make visual accents feel “late” or “soft,” especially on fast transients like drums and percussive synths.[1]

For this reason, Signal³ emphasizes deterministic execution: given the same MIDI input, the system should always produce the same visual result, regardless of system load or performance duration. Achieving this requires careful configuration of the surrounding hardware, operating system, and MIDI routing environment. CPU frequency scaling, background processes, and misconfigured drivers can all introduce small but perceptible variations in responsiveness. The goal is to treat the host machine as a dedicated, semi embedded system during performance, where nonessential tasks are minimized and the critical audio/MIDI/graphics pipeline is given absolute priority.[1]

Signal³ also assumes that MIDI data is semantically meaningful. Notes, velocities, and timing are not just triggers; they encode structural information about the composition. This enables strategies such as mapping specific MIDI note ranges to distinct visual layers, using velocity as a proxy for energy, or assigning particular MIDI channels to different visual "voices." In compositional workflows, this encourages thinking of visuals as an orchestrated component of the score rather than a post production effect.

---

2. Signal Flow and Data Path

Understanding the full signal path is essential for diagnosing issues and ensuring stable performance. In a typical setup, the data path is:

Musical Source (DAW / Sequencer / Controller) MIDI Routing Layer (Virtual Bus or Interface) Signal³ MIDI Input Internal Mapping Layer Visual Synthesis Engine GPU Rendering Display Output.

Signal³ does not generate MIDI data itself; instead, it listens to an upstream source such as a DAW, sequencer, or hardware controller. The MIDI stream is routed through a virtual or physical bus before being consumed by the Signal³ engine. For optimal control, the DAW or sequencer project is usually structured so that the tracks responsible for driving visuals are clearly separated (e.g., dedicated "VISUALS" MIDI tracks), allowing independent arrangement, muting, and automation without affecting audio signal routing.[1]

Once MIDI data enters Signal³, it is parsed, timestamped, and mapped to internal visual parameters. Parsing is responsible for decoding standard MIDI messages (Note On, Note Off, Control Change, etc.), while timestamping ensures events are processed in a coherent temporal order, even when bursts of messages arrive simultaneously. In most operating systems, incoming MIDI events are delivered with

microsecond scale timing information, but the effective resolution will be constrained by the host clock and chosen buffering strategy.[1]

These parameters feed the visual synthesis engine, which generates geometry, distortion artifacts, and noise-based events. Conceptually, the engine maintains a set of stateful visual objects (for example, shapes with positions, velocities, and shader parameters) that are updated each frame in response to newly received MIDI events and internal time progression. Geometry generation can range from simple 2D primitives with parametric deformation to fully 3D structures driven by MIDI based modulation. Distortion artifacts are typically implemented as screen space post processing passes, where MIDI values modulate shader uniforms controlling displacement maps, feedback intensity, or color warping.[1]

The final output is rendered on the GPU and displayed with minimal buffering to preserve immediacy. In practice, this means choosing rendering settings and output resolutions that keep the GPU frame time comfortably below the display refresh interval (e.g., < 16.7 ms for 60 Hz, < 8.3 ms for 120 Hz). Any additional buffering, such as V Sync or compositor level queues, must be considered in the end to end latency budget. For projectors or LED walls, latency introduced by scalers or processors should also be evaluated in rehearsals.[1]

When diagnosing issues, it is useful to conceptualize the pipeline as three main latency domains:

- Input latency: from the moment the performer presses a key or pad to the moment the MIDI event arrives at Signal³.
- Processing latency: time spent parsing, mapping, updating visual states, and scheduling GPU work.
- Output latency: time from GPU submission to photons on screen (including display scanning and any external processing).

All three domains must be kept stable for the audiovisual relationship to remain tightly coupled and repeatable across a set.

---

3. MIDI Output Configuration and Routing Strategy

The choice of MIDI routing strategy has a direct impact on system stability. In most professional setups, virtual MIDI buses are preferred over physical interfaces due to their reliability and resistance to cable or driver issues. Virtual buses eliminate the possibility of loose connectors, faulty cables, and power issues, and they often have

lower and more consistent latency than external USB or DIN devices that depend on additional firmware layers.[1]

On macOS, the IAC Driver provides a low-latency inter-application communication channel. It is configured via the Audio MIDI Setup application, where one or more virtual ports can be created and named (e.g., “Signal3_IN”). Once created, these ports become available as inputs and outputs in compatible applications. For Signal³, the DAW should be configured to send the dedicated visual MIDI track(s) to the chosen IAC port, and Signal³ should be set to listen exclusively on that port.[1]

On Windows, LoopMIDI fulfills the same role, creating one or more named virtual ports that can be used to route data between software components. After installing and launching LoopMIDI, you can create a port such as “Signal3_BUS,” select it as a MIDI output in your DAW, and choose it as an input in Signal³. When using multiple applications or multiple machines, careful naming conventions (e.g., “Signal3_MAIN,” “Signal3_BACKUP”) help keep routing unambiguous.[1]

It is critical that Signal³ is the sole consumer of its assigned MIDI input. Port sharing with other applications can result in dropped Note On messages or unpredictable timing behavior, especially if multiple clients are competing for the same MIDI stream or the driver does not handle multi client scenarios robustly. In more complex setups (for example, where MIDI is simultaneously feeding lighting consoles, secondary visual systems, or recording tools), a dedicated routing layer or MIDI duplication strategy should be used rather than binding all consumers directly to the same port.[1]

Prior to performance, all unnecessary MIDI listeners should be disabled. This includes background utilities, secondary DAWs, or testing tools that might still be attached to the same bus from earlier configuration work. Disabling them reduces the chances of race conditions, feedback loops, or unnoticed filtering of certain message types.[1]

Pre-START Verification Checklist:[1]

1. Confirm that the correct MIDI bus is selected in the source application.
   Ensure that the DAW track(s) responsible for visuals are routed explicitly to the Signal³ input bus, and that no default or global output is unintentionally feeding other devices. For complex shows, it is recommended to maintain a routing diagram or session template where the visual bus is clearly marked.

2. Verify that Signal³ is listening on the intended MIDI channel.
   Use per channel routing to separate different functional roles (e.g., Channel 1

for Shapes, Channel 10 for percussive Noise Bursts) and confirm that only the channels you expect are being interpreted by Signal³. This prevents unexpected triggers from tracks intended for other devices.

3. Ensure the START control is armed before sending any MIDI data.
   In Signal³, the START or “Arm” control transitions the system into an active listening state. Any MIDI data received before arming may be ignored or treated as initialization noise. As a best practice, build a pre show macro in your DAW that arms Signal³, enables the correct tracks, and sends a short, known test pattern to verify correct behavior.

4. Check for background applications that may interfere with MIDI routing.
   Close any MIDI monitors, virtual keyboards, or test tools that might still be connected to the same ports. Also consider disabling or reconfiguring system level services (for example, Bluetooth MIDI, if unused) to reduce the attack surface for interference.

For multi computer setups, where MIDI is transmitted over a network (via RTP MIDI or similar protocols), additional considerations include network stability, jitter, and potential packet loss. In those scenarios, it is often beneficial to terminate network MIDI at a single dedicated machine and then route locally via virtual buses to Signal³, rather than exposing Signal³ directly to the network layer.

---

4. Trigger Mapping and Visual Semantics

Signal³ categorizes visual output into three semantic layers: Shapes, Glitches, and Noise Bursts. Each layer serves a distinct perceptual function and responds to different characteristics of the MIDI stream. Understanding and designing mappings for each layer is essential for achieving a coherent and expressive audiovisual language.[1]

Shapes represent continuity and structure. They persist over time and provide a visual anchor for the composition. Typically, Shapes are driven by pitched notes, chords, or sustained material that reflects harmonic and melodic content. A MIDI Note On may instantiate a new shape with attributes derived from note number (e.g., hue, position, or scale), while Note Off events may control fade out behavior or morphological transitions. Parametric modulation (such as slow LFO like changes in rotation or deformation) can be tied to continuous controllers or tempo synced envelopes.[1]

Glitches introduce rhythmic disruption and movement, responding primarily to note density and temporal clustering. Rather than mapping one to one with individual notes, Glitches often emerge from analyzing patterns over short windows of time. For example, a burst of sixteenth note hi hats might increase glitch intensity, distortion depth, or temporal feedback in the shader pipeline. At lower densities, Glitches might remain subtle or dormant, preserving the clarity of Shapes. This allows the same patch to support both minimal and maximal aesthetic states, depending on musical context.[1]

Noise Bursts are intentionally transient, designed to punctuate moments of high energy or emphasis. They are typically mapped to high velocity events, strong accents, or specific percussive hits (such as kicks, snares, or claps). A Noise Burst might manifest as a short lived explosion of particles, a full screen flash, or a burst of grainy texture that briefly occludes or disrupts the underlying Shapes. By constraining Noise Bursts to clearly defined musical markers, Signal³ ensures that these effects read as intentional, rhythmic punctuation rather than random artifacts.[1]

The relationship between MIDI attributes and visual interpretation can be summarized as follows:[1]

MIDI Attribute – Visual Interpretation
Note On Events – Instantiation or modulation of geometric forms
Event Density (BPM) – Rate and aggressiveness of glitch artifacts
Velocity Values – Amplitude, texture, and spatial impact of noise bursts

In practical terms:

- Note On Events:
  Each incoming Note On can create or modify a Shape, select a glitch mode, or trigger a specific Noise Burst variant. For example, notes in a low register might spawn large, slowly moving geometric bodies, while higher notes create smaller, faster elements. Chord structures can be visually mirrored by clustering shapes or aligning them along axes.

- Event Density (BPM):
  By measuring how many Note On events occur within a sliding time window (for example, 100–250 ms), Signal³ can estimate local event density. Higher densities can drive parameters such as glitch rate, temporal sampling offsets, or shader iteration counts. This density is related to but distinct from the global BPM: fast

playing at a slower BPM can still generate intense glitch behavior if density thresholds are exceeded.

- Velocity Values:
  Velocity is an excellent proxy for perceived energy, making it a natural driver for the strength and spread of Noise Bursts. Low velocity hits might produce small, localized disturbances, whereas high velocity hits can generate screen filling bursts, deeper distortions, or more pronounced luminance changes. When designing mappings, it is useful to define minimum and maximum velocity thresholds to avoid either invisibly subtle or overwhelmingly strong responses.

For advanced setups, different MIDI channels can be mapped to separate semantic roles. For instance:

- Channel 1: Harmonic/melodic Shapes (pads, leads).
- Channel 2: Glitch control (pattern based MIDI, dedicated trigger tracks).
- Channel 10: Drum/percussion based Noise Bursts.

This separation ensures that each layer can be arranged and automated independently in the DAW while remaining coherent in performance.

---

5. Stability and Performance Resilience

Live audiovisual systems operate under conditions of sustained computational load. As visual complexity increases, GPU and CPU usage can approach system limits. Without proper prioritization, MIDI processing may be delayed, resulting in missed or late visual events. The goal is not just peak performance, but resilient performance: the system should maintain consistent behavior even under stress, and it should fail gracefully (e.g., reducing visual complexity) rather than dropping triggers or stuttering.[1]

To prevent this, MIDI handling must be treated as a real-time priority. On the CPU, the thread responsible for processing incoming MIDI and dispatching updates to the visual engine should be kept free from blocking operations, large memory allocations, or disk I/O. Where possible, heavy computations (such as complex geometry generation or non critical analysis tasks) should be moved to background threads or precomputed during load time. On some operating systems, assigning higher scheduling priority to audio/MIDI threads can reduce jitter.[1]

System monitoring during rehearsal is strongly recommended, allowing potential bottlenecks to be identified before they occur on stage. Use tools that display:[1]

- CPU usage per core (to detect single thread bottlenecks).
- GPU usage and frame time (to ensure rendering remains within budget).
- Memory usage (to avoid paging or out of memory issues).
- Thermal conditions (CPU/GPU temperatures and clock throttling).

If you observe frame time spikes, test reductions in resolution, shader complexity, post processing steps, or the number of simultaneously active shapes. Strive for a comfortable safety margin rather than operating at the absolute limit of your hardware.

Consistency, rather than maximum visual density, should be the guiding principle. A stable 60 fps with moderate complexity will almost always feel better in performance than an unstable 90 fps that frequently drops frames. To this end, consider:[1]

- Locking performance targets (e.g., 60 or 75 fps) and designing patches within that envelope.
- Deactivating or simplifying certain layers for particularly heavy sections of a show.
- Using pre rendered elements or baked textures where appropriate, reserving real time computation for the most expressive or variable components.

Redundancy is another important aspect of resilience. In critical shows, it may be advisable to run a backup machine with a mirrored configuration, receiving the same MIDI data via a splitter or network, ready to take over output if the primary machine fails. In such cases, keep both systems synchronized in terms of project versions, presets, and OS configuration.

---

6. Practical Usage Workflow

A typical Signal³ workflow begins with routing MIDI from a musical source into a virtual bus. Once Signal³ is armed using the START control, the system enters a listening state and awaits incoming triggers. From this point onward, visual behavior is entirely driven by MIDI events, allowing the performer to shape the visual narrative in real time.[1]

A recommended step-by-step workflow:

1. Project Preparation in the DAW or Sequencer

- Create dedicated MIDI tracks for visual control (e.g., “VISUAL_SHAPES,” “VISUAL_GLITCH,” “VISUAL_NOISE”).
- Assign clear channel and note ranges for each functional group.
- If necessary, use MIDI effect plugins (arpeggiators, humanizers, note mappers) to generate complex patterns that will be interpreted by Signal³.

2. Virtual Bus Configuration

- Create a virtual MIDI port (IAC on macOS, LoopMIDI on Windows) named specifically for Signal³.
- Route the visual tracks’ outputs to this port only. Avoid sending global MIDI clock or unrelated performance data unless explicitly required by your mappings.

3. Signal³ Input and Mapping Setup

- Select the correct virtual port as the input in Signal³.
- Configure channel filters and note/CC mappings according to your designed semantics for Shapes, Glitches, and Noise Bursts.
- Save these mappings as presets, so that show specific configurations can be recalled quickly.

4. Rehearsal and Calibration

- Arm the START control in Signal³ and trigger predesigned test clips in the DAW.
- Verify that each visual layer responds as expected to its corresponding MIDI tracks.
- Adjust scaling factors (e.g., how much velocity affects Noise Burst intensity) so that the full dynamic range is usable in performance, without requiring extreme velocities.

5. Show Execution

- During the show, use the DAW session, hardware controllers, or both to navigate between scenes and sections.
- Live players can be mapped directly to certain visual layers, while pre sequenced tracks drive others.

- Maintain a simple, repeatable routine for arming Signal³, checking routing, and validating that a known test cue produces the expected visuals before the audience sees the output.

6. Post Show Review

- Review any recorded outputs (visual captures, MIDI logs) to identify timing issues or mapping refinements.
- Incrementally improve the project template you use for future performances, incorporating lessons learned about stability, expressiveness, and operator ergonomics.

When used this way, Signal³ becomes an integrated part of the musical instrumentarium, with its own “part” in the score and a clear operational workflow that can be rehearsed and refined like any other performance element.

---

7. Closing Notes

Signal³ is most effective when treated as an instrument rather than an effect. Its strength lies in predictability, repeatability, and tight coupling between intent and result. With careful configuration and disciplined routing, the system becomes an invisible extension of performance, allowing visuals to breathe, strike, and evolve with musical precision.[1]

By designing clear mappings, respecting the technical constraints of the host system, and maintaining rigorous preparation practices, performers and technicians can rely on Signal³ as a robust component of their live audiovisual setups. The result is a stage environment in which musical structure and visual structure are not merely correlated, but consciously composed together—framing the audience’s perception with a unified, deliberate, and responsive audiovisual language.