SynthOBS & FractiSynth (v1.618)

A Golden-Ratio Broadcast Console and Native Transducer Calibrated by Live Solar Telemetry

Daniel Ari Friedman

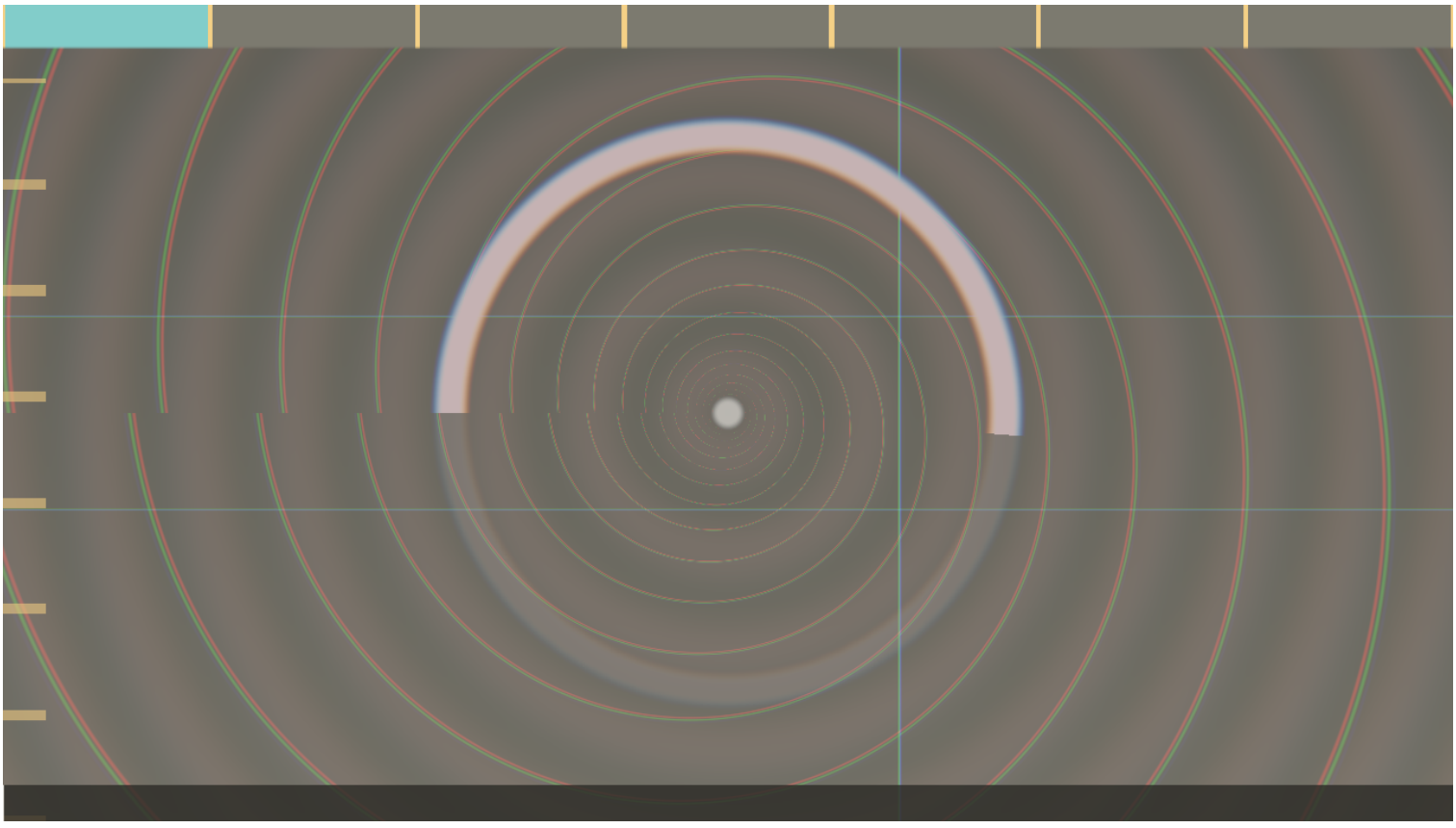
FractiAI / Active Inference Institute

daniel@activeinference.institute

ORCID: 0000-0001-6232-9096

DOI: 10.5281/zenodo.21418688

2026-07-17



Contents

1 Abstract

Author affiliation: FractiAI / Active Inference Institute.

Modern broadcasting platforms such as OBS Studio expose a modular host and plugin surface rather than a single monolith [OBS Project, 2026b,a]. This blueprint specifies SynthOBS and FractiSynth as a bounded extension of that surface: a three-mode operator console, a native video/audio transducer, fail-closed telemetry adapters, and a reproducible evidence protocol. The Observatory, Laboratory, and Expedition Ship names describe the operator modes; they are interface vocabulary, not claims about the physical environment.

The portable geometry and DSP contracts use **the golden-ratio constant** $\varphi = 1.618\ldots$. It determines integer viewport splits, the video scale factor, the audio-limiter knee, and the deterministic spiral. The native plugin also uses a distinct EGS gateway key for its solar-wind phase calculation. These are engineering choices; the manuscript does not infer perceptual or broadcast-quality benefits from the ratio. Phase locking is governed by a *distinct* second constant, the **EGS Fractal Constant** — the dimensionless gateway key

$$K_{\mathrm{EGS}} = \varphi \cdot \frac{\lambda_{\mathrm{reader}}}{\lambda_{\text{H-alpha}}} = \varphi \cdot \frac{1030\ \mathrm{nm}}{656.28\ \mathrm{nm}} \approx 2.539427 \tag{1}$$

which bridges El Gran Sol's 1030 nm optical reader scale to hydrogen's H-alpha line using the NIST wavelength reference [Kramida, 2010]. The system models **two decoupled planes**. A centralized **Solar Wavefield Oscillator (SWO)** reads current, bounded space-weather products — the F10.7 cm solar radio flux and active-region counts — to compute the *amplitude* plane's phase vector

$$v_{\mathrm{phase}} = \frac{\Phi_{10.7}}{N_{\mathrm{spots}}} \cdot \varphi \tag{2}$$

while the **El Gran Sol Gateway** maps an accepted solar-wind speed onto the *phase* plane through eq. 1:

$$\theta_{\mathrm{bias}} = \left(2\pi \cdot \frac{w}{w_{\mathrm{ref}}} \cdot K_{\mathrm{EGS}}\right) \bmod 2\pi, \qquad s_{\mathrm{lock}} = |\cos\theta_{\mathrm{bias}}| \tag{3}$$

The two planes are deliberately independent: a solar-wind dropout does not overwrite the SWO vector, and invalid telemetry does not create a new lock. The gateway then resolves a named interference outcome rather than returning a raw Boolean: a node field ψ is superposed at the AR14409 solar node against a hydrogen phase-flip, and the verdict follows the interference intensity

$$I = |\psi_a + \psi_b|^2 \tag{4}$$

where constructive interference at AR14409 reads "true", a destructive hydrogen phase-flip reads "false", and a tie resolves to a mixed (fail-closed) state.

The two-plane lock and its downstream consumers are organized as follows:

```
flowchart TB
    subgraph SOLAR["☉ Live Solar Telemetry"]
        FLUX["F10.7 cm radio flux Φ₁₀.₇<br/>+ active sunspots N"]
        WIND["Solar-wind speed w (km/s)"]
    end
    subgraph AMP["Amplitude Plane — SWO"]
        VEC["system_phase_vector<br/>v = (Φ₁₀.₇ / N)·φ"]
    end
    subgraph PHASE["Phase Plane — EGS Gateway"]
        LOCK["θ_bias = (2π·w/w_ref·K_EGS) mod 2π<br/>s_lock = |cos θ_bias|"]
    end
    GATE["Holographic Gate<br/>I = |psi_a + psi_b|^2<br/>AR14409 true · H-flip false"]
    OUT["SynthOBS Console + FractiSynth Core<br/>φ-scaled video · audio · layout"]

    FLUX --> VEC
    WIND --> LOCK
    VEC --> OUT
    LOCK --> GATE
    GATE --> OUT
```

Figure 1: Two-plane telemetry model: NOAA flux and active-region data calibrate the SWO amplitude plane, solar-wind speed drives the independent EGS phase plane, and both converge at the declared gate before reaching the console and transducer.

The architecture is decoupled and dual-layer: **SynthOBS**, the Vessel Console, uses a recursive golden-ratio layout matrix whose primary region receives the major integer share; **FractiSynth**, the Transducer Core, is a native `libobs` plugin that implements the declared video and audio kernels at the host boundary. This blueprint specifies both layers, the three modality control decks (three common and four unique buttons per deck), the global command grammar, and the calibration loop. Executable claims are tied to the tested Python engine through declared constant, static, behavioral, and live-OBS contracts. The native plugin is a separate C implementation; the Python engine remains the portable source of truth.

The resulting artifact is intended to be read as research software as well as an OBS extension: the source tree, version metadata, tests, evidence manifest, and manuscript are one citable object. The repository-level citation surface follows established software-citation principles for credit, version specificity, persistence, and accessibility [Smith et al., 2016], with machine-readable metadata in `CITATION.cff` [Citation File Format Initiative, 2026].

The planned public v1 distribution target is the GitHub repository `docxology/SynthOBS`. That repository is intended to be the canonical public source for the engine, native plugin, obspython bridge, tests, installation instructions, manuscript, and versioned evidence metadata; the release boundary and remaining acceptance work are recorded in `RELEASE.md`.

Keywords: OBS Studio, golden ratio, El Gran Sol fractal constant, Solar Wavefield Oscillator, space-weather telemetry, real-time DSP, broadcast engineering.

2 Executive Primer

2.1 The Intention

OBS Studio already provides a modular host for scenes, sources, filters, and operator control. The engineering problem addressed here is narrower: how to keep those surfaces composable when layout, telemetry admission, signal shaping, and live evidence are implemented across Python, native C, and the OBS runtime. A useful design therefore needs explicit contracts at each boundary rather than implicit coordination through UI state.

This project names that design language **Omniversal Observatory, Laboratory, and Expedition Ship**. The names describe three operator modes; they are not claims about the physical origin of the signal or about perceptual superiority. The implementation supplies a recursive viewport allocator, a fail-closed telemetry adapter, native OBS filters, and a reproducible evidence bundle so each boundary can be inspected independently.

2.2 The Solution

The solution is a decoupled, dual-layer infrastructure comprising SynthOBS and FractiSynth, bound natively to **v1.618 — the Goldilocks Calibration Standard**. The host boundary follows OBS's documented module model [OBS Project, 2026a], and the live control path is exercised through obs-websocket's documented protocol surface [OBS Project, 2026c]:

- **SynthOBS (The Vessel Console):** an intelligent UI wrapper that abstracts standard OBS operations, using a dynamic, recursive viewport allocation algorithm to auto-assemble layout structures based on perfect geometric harmony.
- **FractiSynth (The Transducer Core Engine):** a native, real-time signal processing plugin that sits directly within the core rendering and audio mixing loops of the broadcast pipeline (`libobs`).

```
flowchart TB
    TEL["LIVE COSMIC TELEMETRY<br/>NOAA SWPC · F10.7 radio flux ·<br/>active sunspots · solar-wind plasma"]
                    SWO["SOLAR     WAVEFIELD     OSCILLATOR     (SWO)<br/>Calibrates     two     planes,     both     fail-
closed:<br/>amplitude (flux/spots·φ) + gateway phase (K_EGS·wind)"]
    UI["SYNTHOBS CONSOLE UI<br/>Goldilocks Layout Matrix<br/>Viewport self-assembly (61.8% / 38.2%)"]
    ENG["FRACTISYNTH ENGINE (libobs)<br/>φ harmonic audio geometry<br/>spatial video rescaling matrices"]

    TEL -->|continuous real-time stream| SWO
    SWO -->|φ layout split| UI
    SWO -->|K_EGS phase lock| ENG
```

Figure 2: System boundary diagram showing live NOAA telemetry entering the fail-closed SWO, which supplies the golden-ratio console layout and EGS phase lock to the native FractiSynth engine.

The centralized SWO maintains two independent fail-closed planes. The **amplitude plane** computes the tested flux-to-spots vector using φ; the **gateway phase plane** computes a bounded phase bias from solar-wind speed and K_{EGS}. The Python engine uses the verified vector to gate modulation, while native OBS paths consume the corresponding adapter state. If telemetry is non-finite, stale, or outside its admitted range, the affected plane holds its last verified state.

2.3 Core Foundational Theory

The golden layout constant. The implementation pins the golden ratio as a numerical design constant for layout and DSP contracts:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875 \tag{5}$$

In this framework, φ from eq. 5 determines the tested viewport split, recursive subdivisions, deterministic spiral, and soft-limiter parameters. The major and minor fractions are $1/\varphi \approx 0.618$ and $1/\varphi^2 \approx 0.382$. These are explicit engineering choices; the project does not infer a perceptual or universal-design result from them.

The EGS gateway key. Crucially, El Gran Sol's *Fractal Constant* is not the bare golden ratio. It is the dimensionless gateway key K_{EGS} that bridges El Gran Sol's optical reader scale ($\lambda_{\text{reader}} = 1030\,\text{nm}$) to hydrogen's H-alpha geometry ($\lambda_{\text{H-alpha}} = 656.28\,\text{nm}$):

$$K_{\text{EGS}} = \varphi \cdot \frac{\lambda_{\text{reader}}}{\lambda_{\text{H-alpha}}} \approx 2.539427 \tag{6}$$

Where φ governs *layout and DSP*, K_{EGS} from eq. 6 supplies the dimensionless phase calculation used by the gateway adapter. It is a project-specific design key, not a physical measurement of a solar-to-hydrogen coupling.

The Calibrated Wavefield Concept. Software environments can drift when interface, media, and processing modules carry inconsistent state. The SWO makes the relevant state explicit on two independent planes. The amplitude plane emits the *system phase vector* from F10.7 radio flux and active-region count:

$$v_{\text{phase}} = \frac{\Phi_{10.7}}{N_{\text{spots}}} \cdot \varphi \tag{7}$$

The gateway plane maps the admitted solar-wind speed to a phase bias and resolves a lock strength with the tested interference function:

$$\theta = \left(2\pi \cdot \frac{w}{w_{\text{ref}}} \cdot K_{\text{EGS}}\right) \bmod 2\pi, \qquad \ell = |\cos\theta| \tag{8}$$

In eq. 8 the lock strength $\ell \in [0, 1]$ is a deterministic function of the admitted wind value. The gateway returns a named interference verdict—constructive, destructive, or mixed—rather than a bare boolean. These verdicts are control states in the application model, not observations of physical coherence. Invalid flux, spots, or wind values are rejected independently; the affected plane holds its last verified state rather than emitting a replacement.

2.4 Scholarly positioning

This is a design-and-reproducibility paper, not a claim that the golden ratio is a universal law of perception or broadcast quality. The manuscript makes explicit engineering propositions: a single pinned constant reduces cross-language drift; a fail-closed telemetry boundary prevents invalid readings from becoming control state; and a versioned evidence bundle makes a live OBS demonstration inspectable. Those propositions are situated in established software-practice guidance on reproducible computational work [Wilson et al., 2014, Sandve et al., 2013] and in the National Academies distinction between reproducibility and replicability [National Academies of Sciences, Engineering, and Medicine, 2019]. The visionary FractiSynth register remains the system's design language; the evaluation chapter below states which parts are source-backed, test-backed, or live-capture-backed.

2.5 Research-software contribution

The scholarly object is not only the prose description of an interface. It is the coupled source tree, executable tests, build metadata, evidence bundle, and rendered manuscript. This matters because a result produced by a hybrid instrument can be misdescribed in at least three ways: a citation may establish the meaning of an external product but not the behavior of the adapter; a unit test may establish a parser contract but not prove that an installed OBS binary rendered it; and a screenshot may show a compelling state without identifying the exact software and inputs that produced it. SynthOBS therefore makes the chain explicit and keeps each claim attached to the evidence class that can actually falsify it.

This treatment follows the software-citation principles of importance, credit, unique identification, persistence, accessibility, and specificity [Smith et al., 2016]. The planned public home is `docxology/SynthOBS`; the current candidate also carries `CITATION.cff` metadata conforming to the Citation File Format schema [Citation File Format Initiative, 2026]. A repository URL is the active project identity. The eventual public tag, commit, and archival identifier will be the precise scholarly identity for a reported run. That distinction is why the release contract does not invent a DOI before the public artifact exists.

The contribution can be stated as three design propositions:

1. **Authority proposition.** A dependency-free Python reference engine can define portable geometry, telemetry admission, DSP, and command contracts while native and scripting adapters remain thin boundary implementations.
2. **Admission proposition.** Explicit finite, positive, fresh-input predicates and hold-state transitions make invalid telemetry observable as rejection rather than silently turning it into control state.
3. **Evidence proposition.** A versioned manifest that binds inputs, hashes, runtime versions, gate predicates, and captures makes a live OBS demonstration inspectable without presenting one run as a population-level performance result.

These are engineering propositions, not empirical claims about perception, solar causality, or universal broadcast quality. Their value is that each has a local source of truth, a falsifying test, and—where it crosses the host boundary—a named runtime artifact.

3 SynthOBS Architectural Specification (The Interface)

SynthOBS replaces the manual canvas configuration of standard broadcasting software with a single golden-ratio layout engine. Scenes and sources are no longer dragged into place by hand; they self-assemble as operational decks on an exploration vessel, every boundary derived from the EGS fractal constant φ rather than from arbitrary pixel coordinates. The geometric vocabulary follows the historical golden-ratio construction while making the computational rule explicit and testable [Livio, 2002].

3.1 The Global Split Architecture

The interface establishes a clear division of operational responsibilities. The separation of source-of-truth engine, native adapter, and operator bridge is a deliberate modular architecture choice [Gamma et al., 1994].

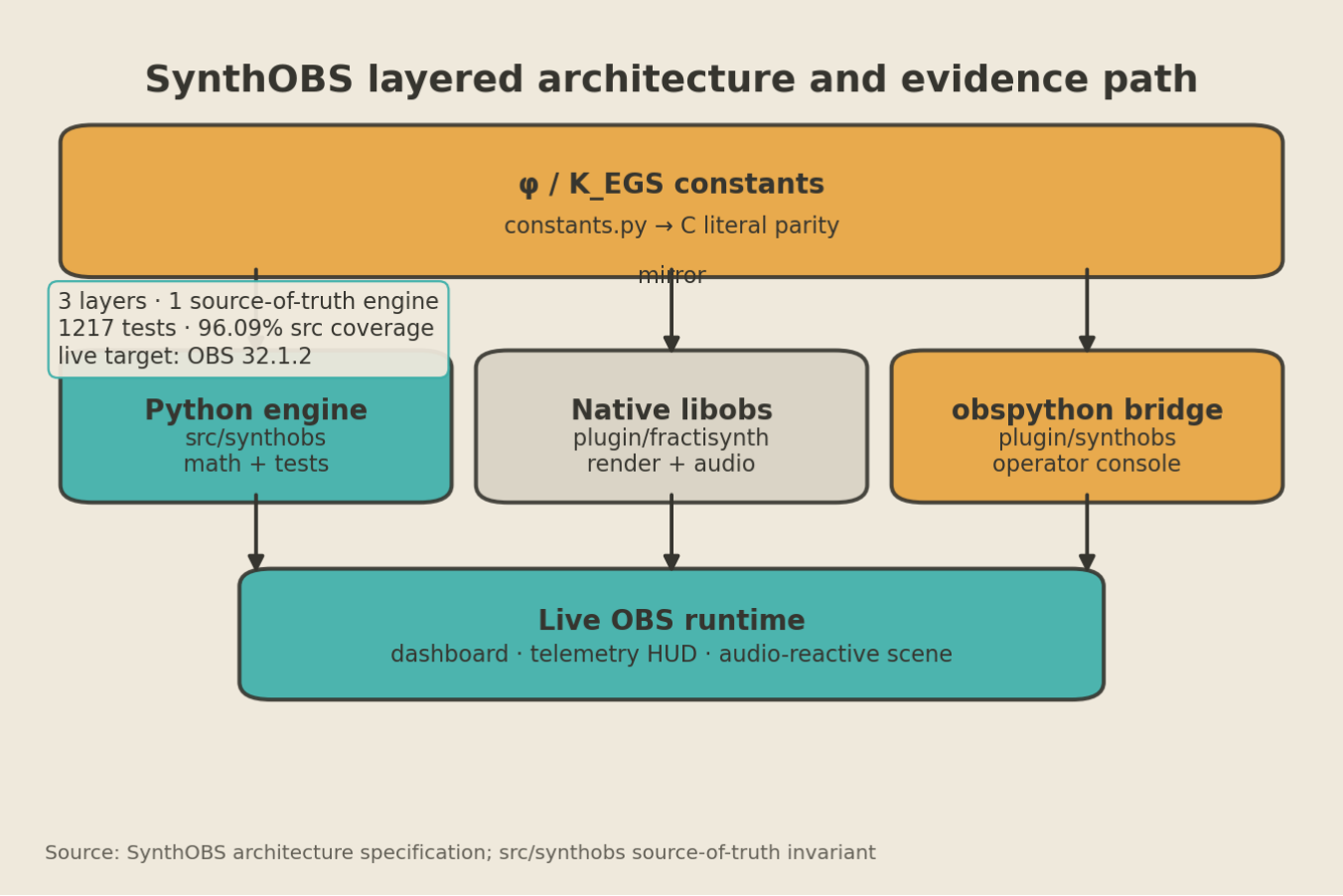


Figure 3: Three-layer SynthOBS architecture and evidence path. The upper band is the shared constant surface (`PHI` and `K_EGS`); the middle band separates the tested Python engine, the native `libobs` transducer, and the `obspython` operator bridge; the lower band is the live OBS runtime where dashboard, telemetry HUD, and audio-reactive scene behavior meet. The figure makes the authority direction explicit: the Python engine defines the mathematics, native adapters implement selected contracts, and live OBS supplies the final integration evidence.

Figure fig. 3 summarizes the paper's central reproducibility boundary: three implementation layers, one mathematical source of truth, and one live runtime at which the independent evidence paths converge. The annotation reports the current verified baseline of 1217 tests at 96.09% source coverage and the OBS 32.1.2 target; it is a status label, not a new performance claim.

- **The Modality Control Decks.** Each of the three modes exposes an *irreducible minimum of 7 console buttons.* Exactly three are common infrastructure that bridge established capabilities — the operational overlap with standard OBS: the Crew Collab Link (remote crew ingestion), the Record Wave Pass (local recording to φ-optimized disk sectors), and Launch Stream (live broadcasting). The remaining four are entirely unique layered-synthesis and tracking capabilities, one of which is always the mode's fail-closed safety control.
- **The Omnipresent Command Line.** Mounted globally at the very base of the SynthOBS interface, a persistent terminal line remains open and in active focus. At any moment the operator can bypass the visible buttons entirely to input scripting, multi-variable macros, or manual routing overrides.

3.2 The Goldilocks UX Framework

The user interface self-assembles dynamically using a recursive golden-ratio layout matrix. The active mode's control deck balances its screen footprint automatically against the live stream canvas. Every partition descends from a single governing law — the **golden**

split — which carves any span into a major and a minor part in the proportion of the EGS fractal constant $\varphi \approx 1.61803398875$:

$$\text{major} = \text{round}\left(\frac{\text{total}}{\varphi}\right), \qquad \text{minor} = \text{total} - \text{major} \tag{9}$$

Applied to the canvas, eq. 9 fixes the two operational decks:

- **Primary Output Canvas (V_p):** allocates the major share — exactly $1/\varphi \approx 61.8\%$ of available screen space — to the primary object of attention and active stream output:

$$V_p = \frac{1}{\varphi} \approx 0.618 \tag{10}$$

- **Control & Telemetry Deck (V_a):** the complementary minor share houses the 7 hardwired console buttons and the real-time SWO tracking graphs:

$$V_a = 1 - \frac{1}{\varphi} = \frac{1}{\varphi^2} \approx 0.382 \tag{11}$$

The identity $V_a = 1/\varphi^2$ in eq. 11 is the self-similar signature of φ — the minor share of one split is the major share of the next, so the layout nests fractally to any depth without ever introducing a new constant.

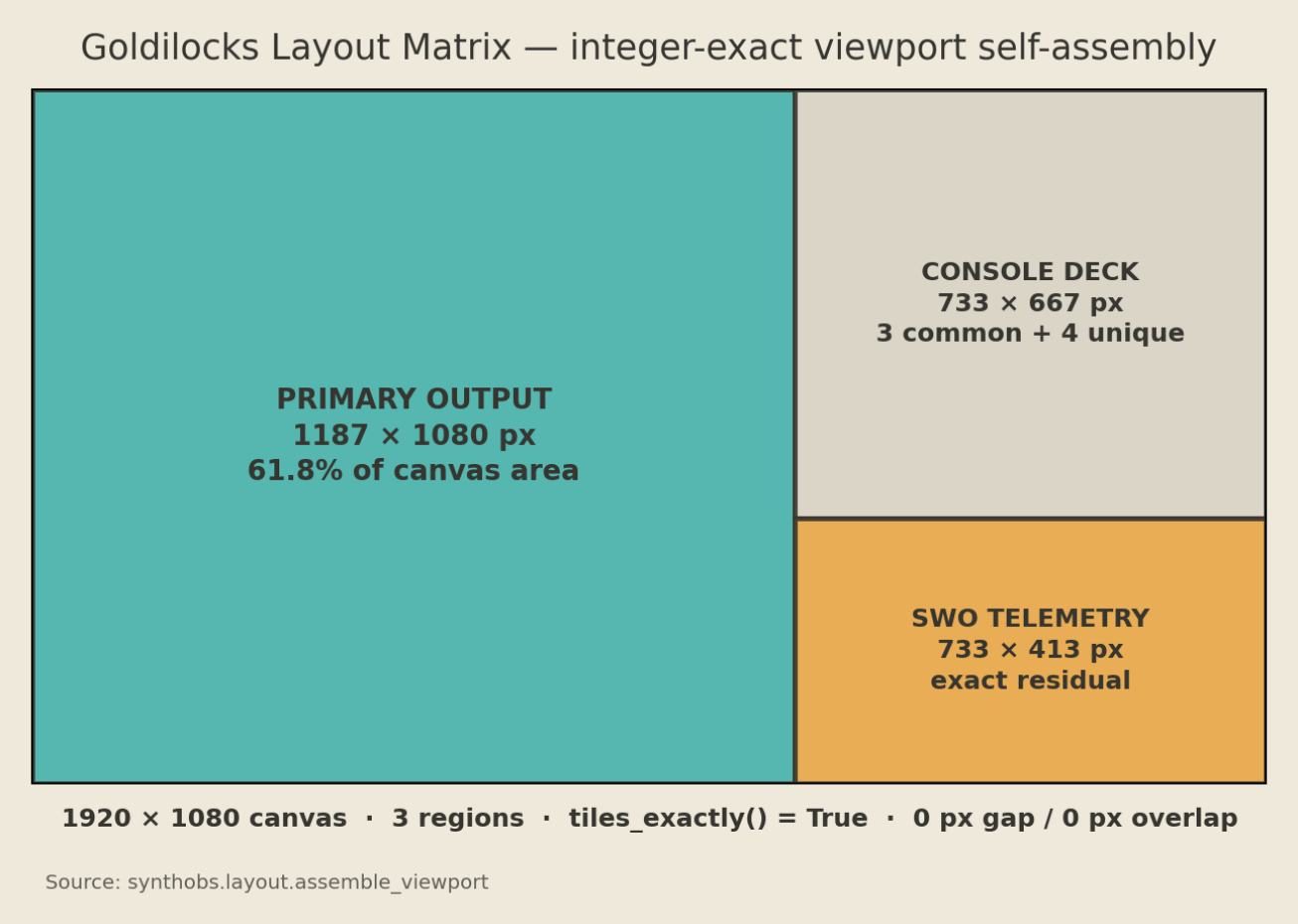


Figure 4: The Goldilocks Layout Matrix generated by `assemble_viewport(1920, 1080)`. The primary output is an exact 1187×1080-pixel region; the 733-pixel right column is recursively split into a 733×667 console deck and a 733×413 SWO telemetry strip. The labels expose the 3-common-plus-4-unique console contract, while the footer reports the executable invariant `tiles_exactly() = True`: the three regions cover all 2,073,600 canvas pixels with zero gap and zero overlap.

The split is integer-exact and fails closed against rounding loss. Because the minor part in eq. 9 is defined as the residual total − major rather than independently rounded, the partition always satisfies

$$\text{major} + \text{minor} = \text{total} \tag{12}$$

with no lost pixels — the rounding remainder of eq. 9 is absorbed entirely by the minor part. The viewport assembled by the tested `assemble_viewport()` engine function applies eq. 9 twice — once horizontally, once vertically on the right column — so the three decks tile the canvas exactly per eq. 12:

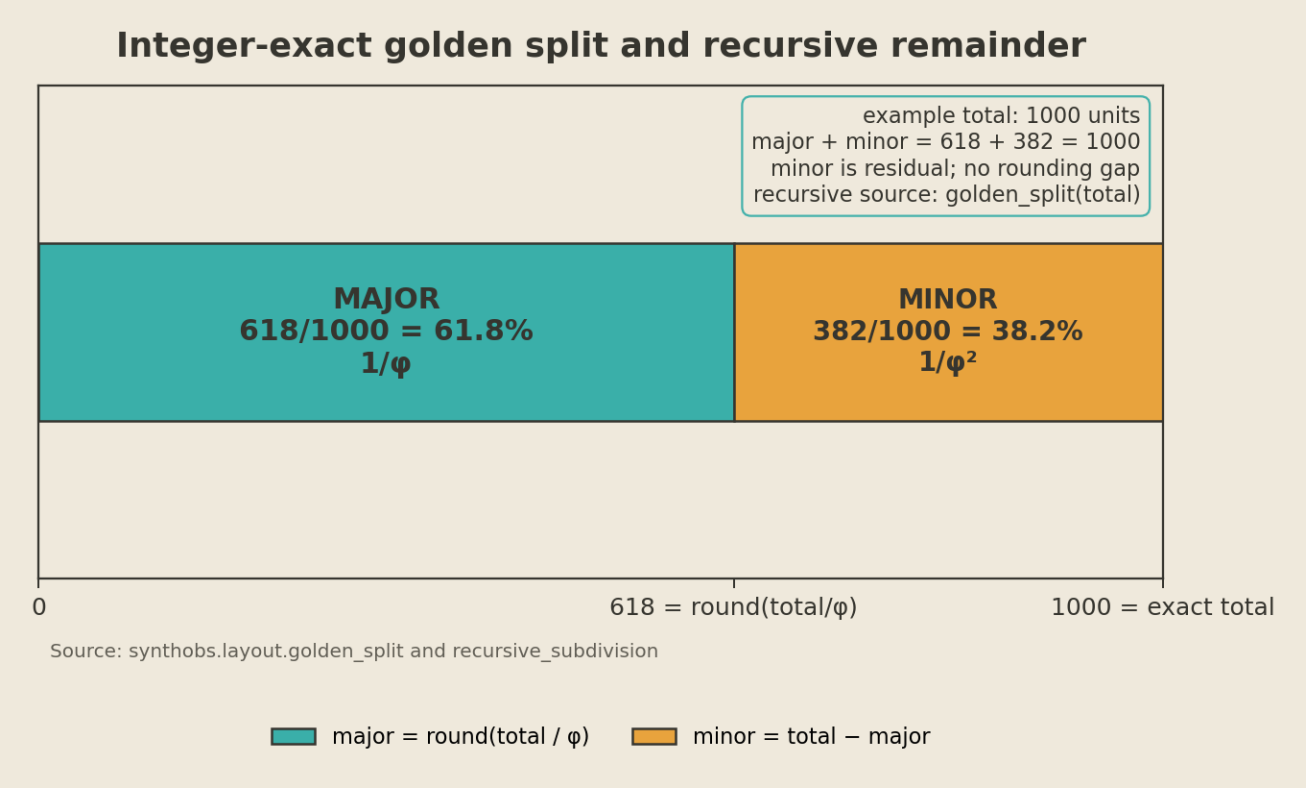


Figure 5: Integer-exact golden split used by the viewport and recursive panel engine. For a 1000-unit span, the major part is `round(1000/`φ`) = 618` and the minor part is the residual 382; the bar therefore closes exactly at 1000 rather than accumulating a rounding seam. The percentage labels are diagnostic approximations of the mathematical $1/\varphi$ and $1/\varphi^2$ shares; the integer equality is the enforced property.

Figure fig. 5 is the one-dimensional primitive behind the viewport: every recursive cut consumes the major share and carries the exact remainder forward. Figure fig. 4 then shows the same primitive applied once horizontally and once vertically.

The lossless property of eq. 12 is verified across a swept range of integer canvas dimensions in the project test suite (`tiles_exactly()` confirms the three regions partition the canvas with zero overlap and zero gap).

3.3 Visual Styling Vectors (Golden Age, Mid-Century Philosophy)

- **Aesthetic Directive:** clean mid-century modern design principles combined with a classic Golden Age layout — understated structural framing, functional symmetry, and scannable visual balance.
- **Chrome Properties:** warm organic charcoal, linen, and matte bone textures that ground the interface, eliminating high-gloss glare and emphasizing structural weight.
- **Accent Signifiers:** robin's-egg blue and muted turquoise for phase-locked operational status indicators; rich marigold orange for real-time telemetry markers and modulated active vectors.

4 The 3 Modality Control Matrices

Each deck exposes seven hardwired buttons — three common (the operational overlap with standard OBS's host capabilities [OBS Project, 2026b]) and four unique to the mode. The common triad — `CREW_COLLAB_LINK`, `RECORD_WAVE_PASS`, `LAUNCH_STREAM` — is identical across all three decks; the twelve unique buttons are disjoint. The console model is encoded and structurally validated in the tested engine (`synthobs.console`).

```
flowchart TB
    COMMON["COMMON TRIAD<br/>CREW_COLLAB_LINK · RECORD_WAVE_PASS · LAUNCH_STREAM"]
    COMMON --> OBS
    COMMON --> LAB
    COMMON --> SHIP

    subgraph OBS["OBSERVATORY · inbound alignment"]
        direction TB
        O1["SWO_SYNC"]
        O2["CAP_DEV_ALIGN"]
        O3["HOLO_GRID_ENGAGE"]
        O4["OBS_DUMP (safety)"]
    end
    subgraph LAB["LABORATORY · synthesis"]
        direction TB
        L1["EGS_SCALE_LOCK"]
        L2["TRANS_VIDEO_FLUID"]
        L3["HARMONIC_COMP"]
        L4["LAB_RESET_ZERO (safety)"]
    end
    subgraph SHIP["EXPEDITION SHIP · outbound"]
        direction TB
        S1["TRANS_WIPE_SEQUENCE"]
        S2["BITRATE_THROTTLE"]
        S3["HULL_INTEG_CHECK"]
        S4["EMERGENCY_ABORT (safety)"]
    end
```

Figure 6: Three-mode console contract: one identical common triad fans into Observatory, Laboratory, and Expedition Ship decks, each with four disjoint unique controls and one explicit safety control.

4.1 Observatory Mode — Inbound Alignment Matrix

Mission focus: discovery, signal capture, sensor synchronization, anchoring the inbound wavefield.

Button	Class	Function
`CREW_COLLAB_LINK`	common	Inbound remote ingestion portal for external crew/observatory streams.
`RECORD_WAVE_PASS`	common	Commit raw wavefield data to local disk with φ-optimized sector allocation.
`LAUNCH_STREAM`	common	Establish the outbound handshake and broadcast the active observatory viewport.
`SWO_SYNC`	unique	Force an immediate clock calibration against current sunspot radio flux (e.g. AR4465).
`CAP_DEV_ALIGN`	unique	Cycle and test inbound capture hardware; align raw frames into harmonic bounding boxes.
`HOLO_GRID_ENGAGE`	unique	Overlay a golden-ratio alignment grid to guide spatial camera composition.
`OBS_DUMP`	unique (safety)	Purge all raw inbound buffer queues and temporary frames to clear signal latency.

`SWO_SYNC` does not merely read a clock — it phase-locks the inbound wavefield to the live Sun. The gateway consumes the NOAA real-time solar-wind product [NOAA SWPC, 2026b] and injects the measured speed as a phase bias on the virtual 1030 nm reader,

weighted by the EGS Fractal Constant K_{EGS}, then reports how strongly the system is in phase. This is the gateway phase law (`synthobs.gateway.gateway_filter`):

$$\phi_{\mathrm{bias}} = \left(2\pi \cdot \frac{v_{\mathrm{wind}}}{v_{\mathrm{ref}}} \cdot K_{\mathrm{EGS}}\right) \bmod 2\pi, \qquad \ell = |\cos \phi_{\mathrm{bias}}| \tag{13}$$

with reference wind $v_{\mathrm{ref}} = 400\ \mathrm{km\,s^{-1}}$ and lock strength $\ell \in [0, 1]$ (1 a perfect lock, 0 fully out of phase). The gateway key itself anchors El Gran Sol's optical scale to the hydrogen line:

$$K_{\mathrm{EGS}} = \varphi \cdot \frac{\lambda_{\mathrm{reader}}}{\lambda_{\mathrm{H\text{-}alpha}}} = \varphi \cdot \frac{1030}{656.28} \approx 2.539427 \tag{14}$$

Equation eq. 21 fails closed: a non-positive wind reading never produces a lock, so the deck holds its last good state rather than synchronizing to a stale indicator. Note that φ governs spatial *layout* throughout the console, whereas K_{EGS} of eq. 14 governs *phase* locking to solar telemetry — distinct constants for distinct planes.

4.2 Laboratory Mode — Processing & Synthesis Engine

Mission focus: deep signal modification, real-time audio/video transformation, geometric harmonization.

Button	Class	Function
`CREW_COLLAB_LINK`	common	Shared telemetry/session sync; remote collaborators pipe streams into the φ transducer matrix.
`RECORD_WAVE_PASS`	common	Commit synthesized, transformed audio/video directly to local storage.
`LAUNCH_STREAM`	common	Deploy the ongoing laboratory signal synthesis to live networks.
`EGS_SCALE_LOCK`	unique	Enforce the EGS fractal constant across all gain stages and frame-crop variables.
`TRANS_VIDEO_FLUID`	unique	Convert static pixel blocks into a fluid, responsive wavefield texture.
`HARMONIC_COMP`	unique	Route audio through a recursive $1/\varphi$ soft-limiting compressor curve.
`LAB_RESET_ZERO`	unique (safety)	Snap all DSP values back to the baseline Goldilocks calibration standard.

`EGS_SCALE_LOCK` pins every gain stage and frame-crop variable to the same gateway key K_{EGS} of eq. 14, so the synthesis plane scales in lockstep with the inbound phase plane. `HARMONIC_COMP` then routes audio through a recursive soft-limiter whose knee sits at the reciprocal golden ratio: below τ/φ (the golden $1/\varphi$ fraction of the ceiling τ) the signal passes untouched; above it the excess is smoothly saturated by a tanh branch scaled to the headroom $h = \tau/\varphi^2$, so the magnitude approaches but never crosses τ.

$$g(x) = \begin{cases} x, & |x| \le \tau/\varphi \\ \operatorname{sgn}(x)\left[\dfrac{\tau}{\varphi} + h\,\tanh\!\left(\dfrac{|x| - \tau/\varphi}{h\,\varphi}\right)\right], & |x| > \tau/\varphi \end{cases} \tag{15}$$

This is exactly the shipped φ soft-limiter (the tested `phi_soft_limit_sample`, restated as the FractiSynth limiter in the Transducer Core section): the $1/\varphi$ knee is what keeps the compressor "harmonic" — the same constant that lays out the canvas also shapes the limiting curve — while the tanh branch (not a linear fold) is what makes it monotone, sign-preserving, and NaN/Inf-safe.

4.3 Expedition Ship Mode — Outbound Transmission Deck

Mission focus: secure encoding, packet delivery, flight/stream-path monitoring, planetary broadcasting.

Button	Class	Function
`CREW_COLLAB_LINK`	common	Transform a standalone stream into a multi-vessel fleet broadcast.
`RECORD_WAVE_PASS`	common	Capture local master archival records of the entire outbound voyage.

Button	Class	Function
`LAUNCH_STREAM`	common	Initiate the low-latency encoding engine to push the ship's transmission live.
`TRANS_WIPE_SEQUENCE`	unique	Scene cut via a fractal geometric wipe along an active φ spiral trajectory.
`BITRATE_THROTTLE`	unique	Scale outbound encoding bitrate live to match network throughput without dropping frames.
`HULL_INTEG_CHECK`	unique	Real-time diagnostic scan: frame drops and rendering latency reported as "hull integrity".
`EMERGENCY_ABORT`	unique (safety)	Drop the outbound stream, clear network ports, set the canvas to an obsidian safe-state.

The geometric wipe of `TRANS_WIPE_SEQUENCE` expands or collapses along an active φ spiral. The trajectory is generated by the tested `golden_spiral_points()` function (`synthobs.layout`) — a logarithmic spiral whose radius multiplies by exactly φ every quarter-turn:

$$r(\theta) = a \cdot \varphi^{2\theta/\pi} \tag{16}$$

The growth law of eq. 16 satisfies $r(\theta + \pi/2) = \varphi\, r(\theta)$, so the wipe front advances by one golden step each quarter-turn — the same φ that splits the canvas now drives the transition geometry (fig. 7).

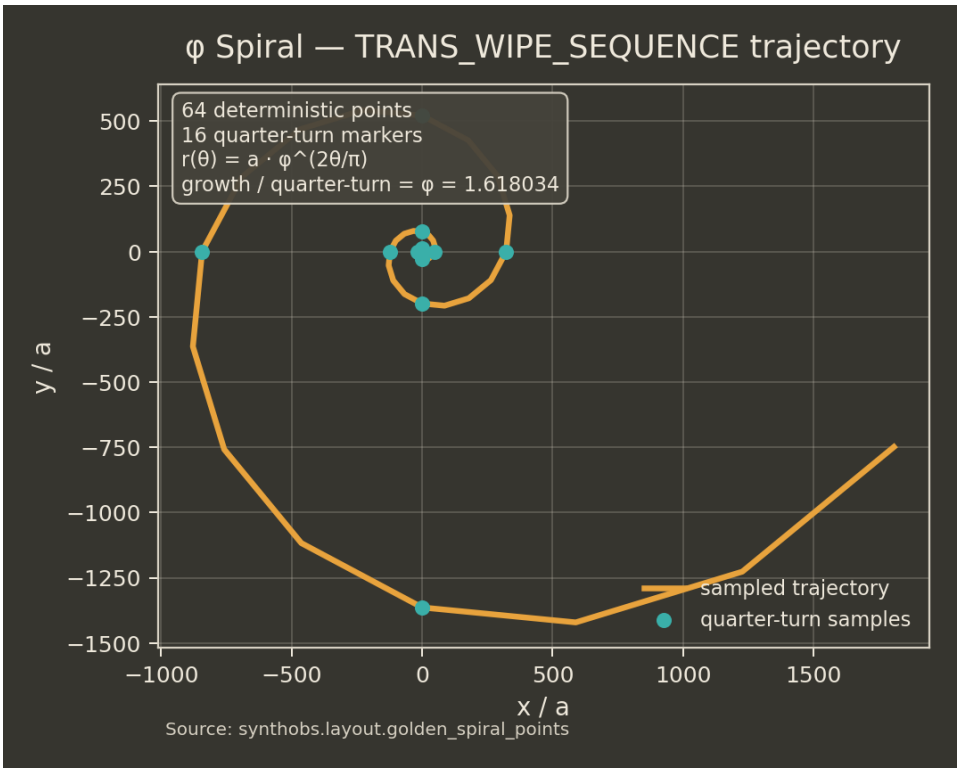


Figure 7: The φ spiral driving the `TRANS_WIPE_SEQUENCE` fractal transition. The deterministic generator samples 64 points from $r(\theta) = a\,\varphi^{2\theta/\pi}$ and marks every fourth sample, so the 16 robin's-egg markers identify quarter-turn boundaries. The displayed radius grows by exactly $\varphi = 1.618034$ per quarter-turn; the plotted curve is therefore a direct rendering of the tested `golden_spiral_points()` law rather than an illustrative freehand spiral.

5 FractiSynth Engine Specification (The Processing Core)

FractiSynth is compiled as a native, low-latency plugin module that hooks into the core processing pipelines of `libobs`, using the host's documented module/plugin surface [OBS Project, 2026a]. It exposes three filters — a video calibrator, an audio harmonic limiter, and a Zoom Inspector — plus the interactive console source; all consume the pinned constants and the accepted Solar Wavefield Oscillator vector at the OBS boundary.

5.1 Native Video Manipulation Pipeline

The plugin intercepts the video rendering loop at the texture level. Before frames are handed off to the hardware encoder, FractiSynth forces spatial transformations to scale directly against the EGS fractal constant φ, establishing the *calibrated harmonic bounding box*:

$$w_{\text{cal}} = \text{round}\left(\frac{w_{\text{source}}}{\varphi}\right), \qquad h_{\text{cal}} = \text{round}\left(\frac{h_{\text{source}}}{\varphi}\right), \qquad w_{\text{cal}}, h_{\text{cal}} \geq 1. \tag{17}$$

Equation eq. 17 scales each axis to $1/\varphi \approx 0.618$ of its source extent — the 61.8% golden fraction — clamped so a non-degenerate source never collapses to zero extent. The live SWO `system_phase_vector` then modulates the shader displacement amount, so the overlay breathes with current space weather. The native filter implements the full `libobs` lifecycle — `create / destroy / update / get_properties / get_defaults / video_render / get_width / get_height` — and its calibration is the tested Python `video_calibrated_dims()` taken as the source of truth.

5.2 Native Audio Harmonic Balancing

Audio sample frames passing through the internal DSP matrix are intercepted via the `filter_audio` callback. Instead of harsh linear peak limiting that clips frequencies, the buffers are compressed along a smooth recursive curve scaled by φ. Below a knee at $1/\varphi$ of the ceiling the signal passes through untouched; above it, the excess is soft-compressed so the output asymptotically approaches — but never exceeds — the ceiling:

$$y(x) = \begin{cases} x, & |x| \leq \tau/\varphi \\ \text{sign}(x)\left[\dfrac{\tau}{\varphi} + h\,\tanh\left(\dfrac{|x| - \tau/\varphi}{h\,\varphi}\right)\right], & |x| > \tau/\varphi \end{cases} \tag{18}$$

where τ is the ceiling and $h = \tau(1 - 1/\varphi) = \tau/\varphi^2$ is the headroom. The knee of eq. 18 sits at τ/φ, so the identity region spans exactly the golden $1/\varphi$ fraction of the ceiling; the tanh branch is bounded by h, guaranteeing $|y|$ approaches but never crosses τ. This is a compact, explicitly specified transfer curve in the tradition of digital audio signal-processing design [Smith, 2007]. The curve is monotone, sign-preserving, and NaN/Inf-safe (fig. 8). These properties describe the transfer function; they do not establish a perceptual improvement or a preferred mastering outcome.

The limiter now also emits its own visual pulse: after the same post-limiter samples are written back into OBS, the engine measures RMS, peak, and a φ-scaled reactivity scalar. Those three values feed the Wavefield Console shader, the Telemetry HUD, and the gateway dock, so the visual surface breathes from the acoustic envelope without ever letting raw, non-finite audio poison the display.

5.3 Single Source of Truth

Two implementations carry the same calibration: the native C plugin (`#define EGS_PHI 1.61803398875f`, `#define EGS_GATEWAY_KEY 2.53942700f`) and the tested Python engine (`synthobs.constants.PHI`, `synthobs.constants.EGS_GATEWAY_KEY`). Project tests pin both literals against their floating-point counterparts — φ to at least nine significant digits and K_{EGS} to better than 10^{-6} — so the native transducer and the reference engine cannot drift apart silently.

The two paths converge by contract: the Python engine defines the video scaling of eq. 17, the audio knee of eq. 18, and the SWO phase vector; the native plugin implements the OBS-bound counterparts. Literal pinning and behavioral checks make material divergence visible.

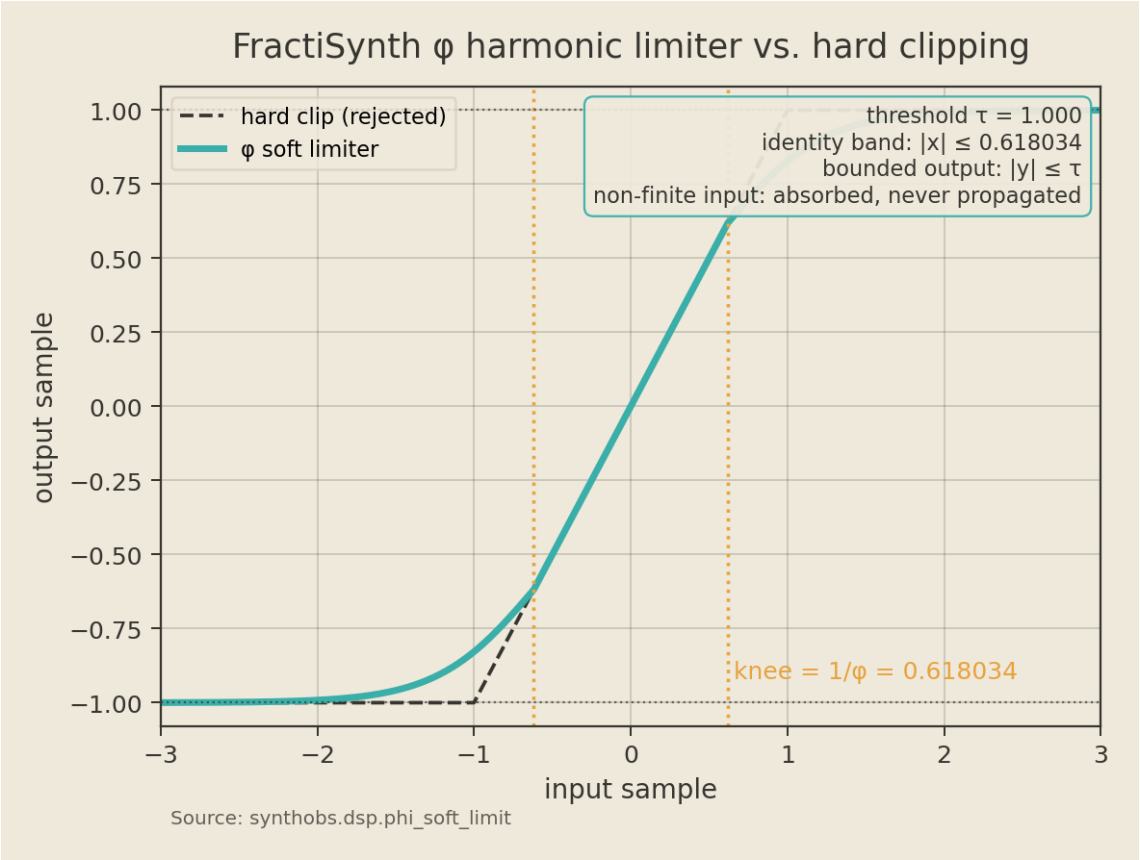


Figure 8: The FractiSynth φ harmonic limiter (robin's-egg) versus naive hard clipping (dashed charcoal) for threshold $\tau = 1$. Marigold guides mark the symmetric identity band $|x| \leq 1/\varphi = 0.618034$; outside that band the tested `phi_soft_limit()` curve remains monotone, sign-preserving, and bounded by $|y| \leq \tau$ while approaching the ceiling smoothly. The statistics panel records the exact knee and the fail-closed handling of non-finite inputs.

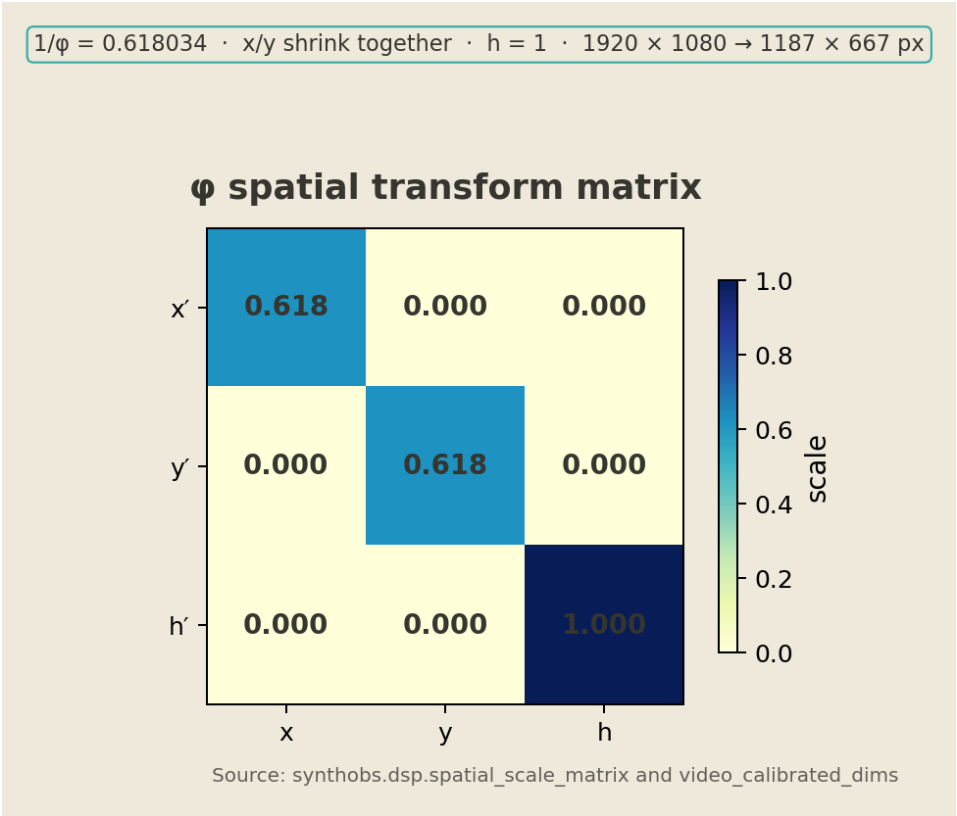


Figure 9: Homogeneous spatial transform matrix used by the calibrated video path. The two spatial axes are scaled by $1/\varphi = 0.618034$, the homogeneous coordinate remains 1, and the worked 1920×1080 example rounds to 1187×667 pixels. Cell values and the color scale expose the diagonal-only transform implemented by `spatial_scale_matrix()`; the source footer identifies the companion `video_calibrated_dims()` calculation.

```
flowchart TB
    SRC["Tested Python engine<br/>src/synthobs — source of truth"]
    SRC -->|"PHI = 1.6180339887…"| PHI{{"φ — golden LAYOUT constant"}}
    SRC -->|"EGS_GATEWAY_KEY = φ·(1030/656.28)<br/>≈ 2.539427"| KEGS{{"K_EGS — solar-wind PHASE lock"}}

    PHI --> V["video_calibrated_dims()<br/>round(dim · 1/φ), ≥ 1"]
    PHI --> A["phi_soft_limit()<br/>knee at τ/φ, tanh branch"]
    KEGS --> S["phase_vector() + gateway lock<br/>lock_strength = |cos(phase_bias)|"]

    V -.->|"implemented at OBS boundary"| CV["C: fractisynth_video"]
    A -.->|"implemented at OBS boundary"| CA["C: fractisynth_audio"]
    S -.->|"implemented at OBS boundary"| CS["C: SWO telemetry thread"]

    LIT["Plugin-artifact tests<br/>pin C literals ↔ Python constants"]
    LIT -.->|"≥9 sig-digit / 1e-6 gate"| CV
    LIT -.-> CA
    LIT -.-> CS
```

Figure 10: Python-to-native parity map: the Python engine owns PHI, the EGS gateway key, and the tested kernels; C OBS filters and the telemetry thread implement those contracts, while artifact tests pin the shared literals.

6 The Solar Wavefield Oscillator & Real-Time Calibration

The software uses a live calibration boundary for current space-weather telemetry: F10.7 radio flux, active sunspot regions, and bulk solar-wind speed. The Solar Wavefield Oscillator (SWO) is the stateful adapter for those feeds. The provenance chain is NOAA SWPC's public F10.7, sunspot, and RTSW products [NOAA SWPC, 2026a,c,b], fetched natively through libcurl [Stenberg and curl contributors, 2026].

6.1 Real-Time Dependency Rule

The production telemetry path does not use historical databases, static baseline logs as live control input. If network connectivity drops—or any reading is malformed, stale, or non-physical—the system enters an isolated **Hold State** and retains the last verified vector until a valid live connection is re-established.

This fail-closed discipline is enforced at every layer. The telemetry ingestion client (`synthobs.telemetry`) raises `TelemetryUnavailable` on a non-200 response, a malformed payload, a non-positive flux or sunspot count, or a timestamp older than the staleness horizon — it never substitutes a default. The native plugin's libcurl thread applies the same rule: on any curl error it holds the last verified vector.

6.2 Technical Calibration Logic

The calibration module processes active-region counts and solar radio flux to output the bounded `system_phase_vector` used by the SynthOBS interface and FractiSynth core filters. The amplitude plane is defined by eq. 19:

$$v_{\text{phase}} = \frac{\Phi_{\text{F10.7}}}{N_{\text{spots}}} \cdot \varphi \tag{19}$$

The Python `SolarWavefieldOscillator.calibrate()` is the tested source of truth for eq. 19. The native plugin implements the OBS-bound counterpart in the `fractisynth_swo_t` struct, scaling against the pinned `EGS_PHI` literal (`1.61803398875f`); static and behavioral checks guard the shared contract:

```c
static bool synchronize_swo_calibration(fractisynth_swo_t *swo,
                                        float current_flux, int active_spots) {
    /* ENFORCEMENT: block any stale, default, zeroed, or non-finite indicator. */
    if (!isfinite(current_flux) || current_flux <= 0.0f || active_spots <= 0) {
        swo->is_calibrated = false;
        return false;                /* escape to Hold Pattern — vector unchanged */
    }
    swo->active_f107_flux    = current_flux;
    swo->monitored_sunspots  = active_spots;
    swo->system_phase_vector = (current_flux / (float)active_spots) * EGS_PHI;
    swo->is_calibrated       = true;
    return true;
}
```

6.3 Modulating Variables

When the admitted telemetry changes, the `system_phase_vector` changes on the next accepted calibration. That value can update the visual shader displacement and audio-reactivity paths; the implementation does not claim that the resulting media state is a physical reflection of space-weather dynamics.

fig. 11 shows the calibration response across a swept range of solar flux for one, three, and seven active regions — the phase vector that drives the whole wavefield.

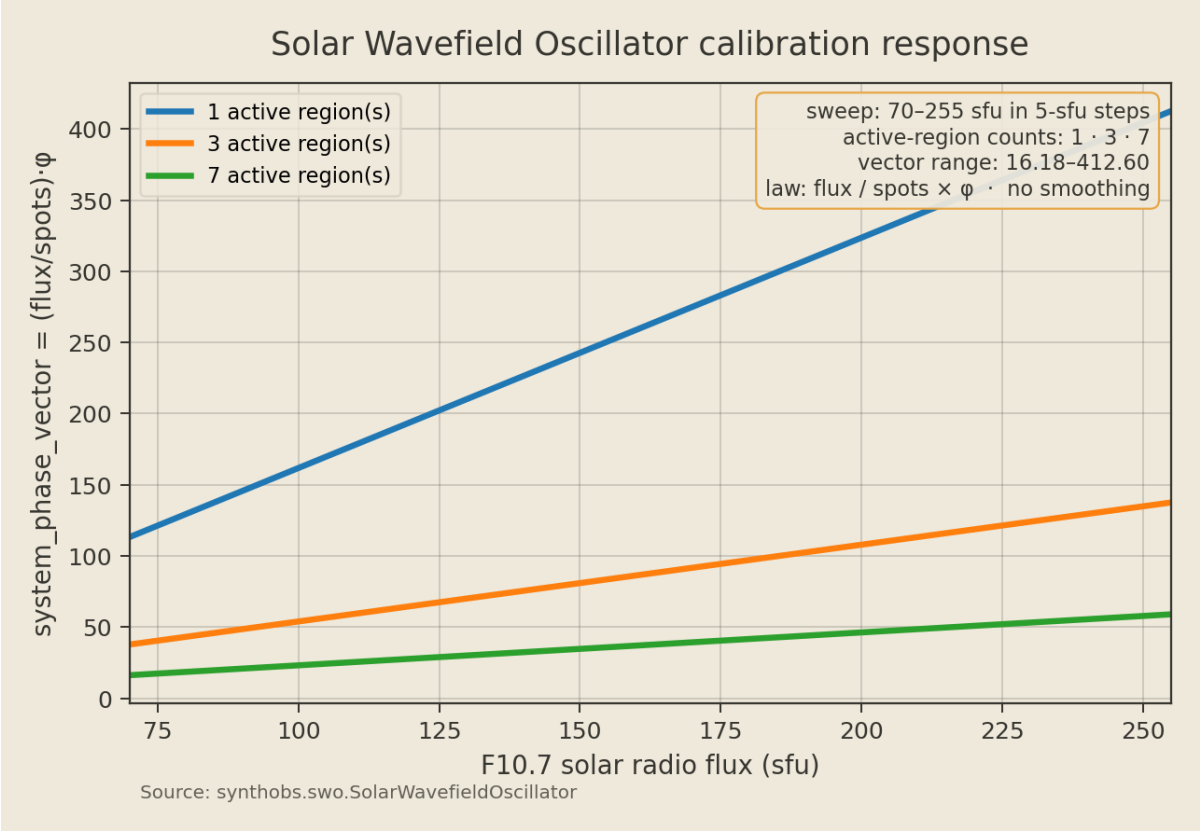


Figure 11: Solar Wavefield Oscillator calibration response. The generated sweep covers F10.7 values from 70 through 255 sfu in 5-sfu steps for 1, 3, and 7 active regions. Each line is the un-smoothed law `system_phase_vector = flux / spots` × φ; the displayed range (16.18–412.60) makes the inverse spot-count scaling and linear flux response directly inspectable. No telemetry value is imputed by this analytical figure.

6.4 The EGS Gateway — the Phase Plane

The flux-and-sunspot calibration above sets the *amplitude* plane. A second, independent plane computes the *phase* of the modeled wavefield from the admitted solar-wind value: the **El Gran Sol Gateway**. Here the system's defining constant is not the golden ratio of layout but the **EGS Fractal Constant** — the dimensionless gateway key K_{EGS} that bridges El Gran Sol's optical scale to hydrogen's H-alpha geometry, defined in eq. 20:

$$K_{\mathrm{EGS}} = \varphi \cdot \frac{\lambda_{\mathrm{reader}}}{\lambda_{\mathrm{H\text{-}alpha}}} = \varphi \cdot \frac{1030\ \mathrm{nm}}{656.28\ \mathrm{nm}} \approx 2.539427 \tag{20}$$

The gateway injects the **live solar-wind speed** into a virtual 1030 nm reader model as a phase bias, weighted by K_{EGS}, and reports the resulting model lock strength. eq. 21 is the phase plane:

$$\theta = \left(2\pi \cdot \frac{v_{\mathrm{wind}}}{400\ \mathrm{km/s}} \cdot K_{\mathrm{EGS}}\right) \bmod 2\pi, \qquad \mathrm{lock} = |\cos\theta| \tag{21}$$

In eq. 21 a perfect model lock (lock = 1) means the computed phase lands on a cosine maximum; a null (lock = 0) means the computed phase is in quadrature. These labels describe the application model, not a measurement of hydrogen or solar-plasma phase. Like the amplitude plane it fails closed — a non-positive wind speed holds the last verified lock — and the two planes are fully decoupled: a wind dropout never disturbs the phase vector, and a flux dropout never disturbs the gateway lock. fig. 12 sweeps the lock across the solar-wind range, marking the FractiAI nominal of 551.7 km/s.

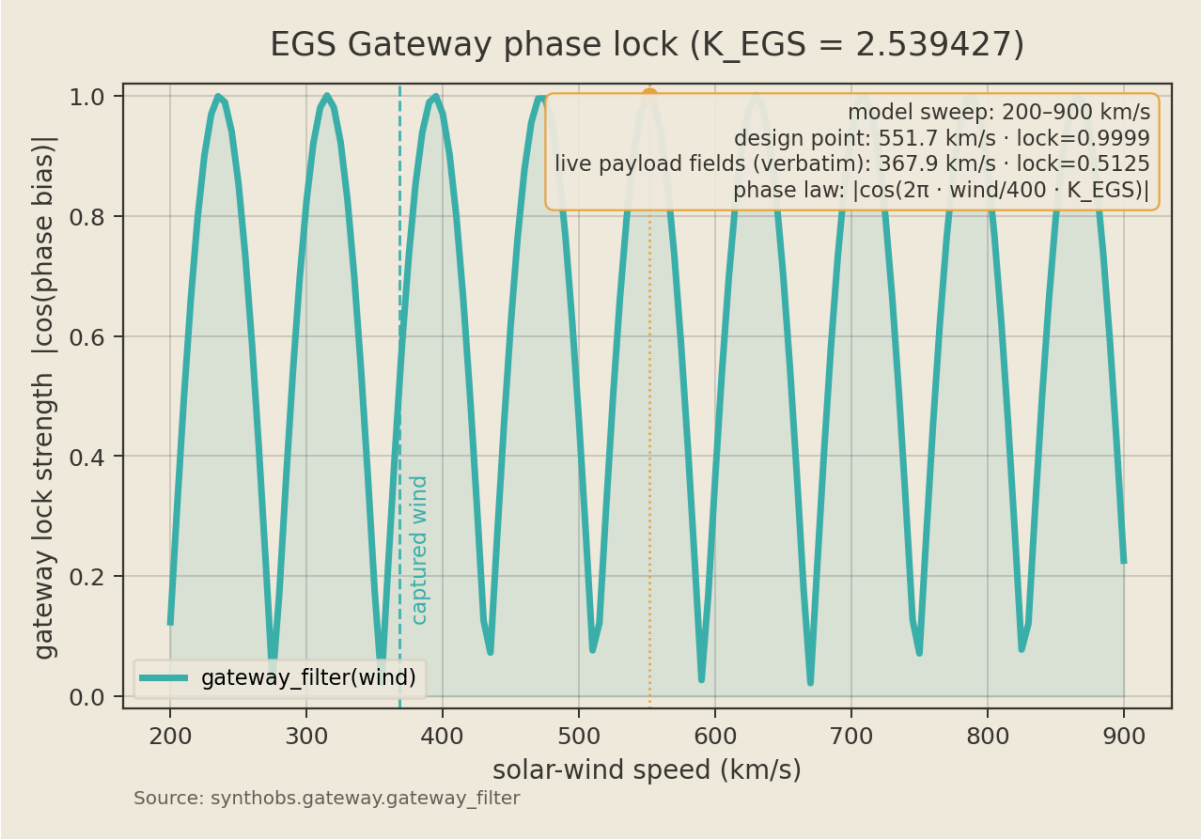


Figure 12: EGS Gateway phase lock. The teal curve evaluates the tested `gateway_filter()` from 200 to 900 km/s; the orange marker is the design point at 551.7 km/s, where the model lock is 0.9999 for $K_{\mathrm{EGS}} = 2.539427$. The dashed teal line marks the wind field recovered from the current versioned live payload (367.9 km/s); the captioned statistics panel reports that run's fields verbatim, while the curve itself remains a deterministic model sweep.

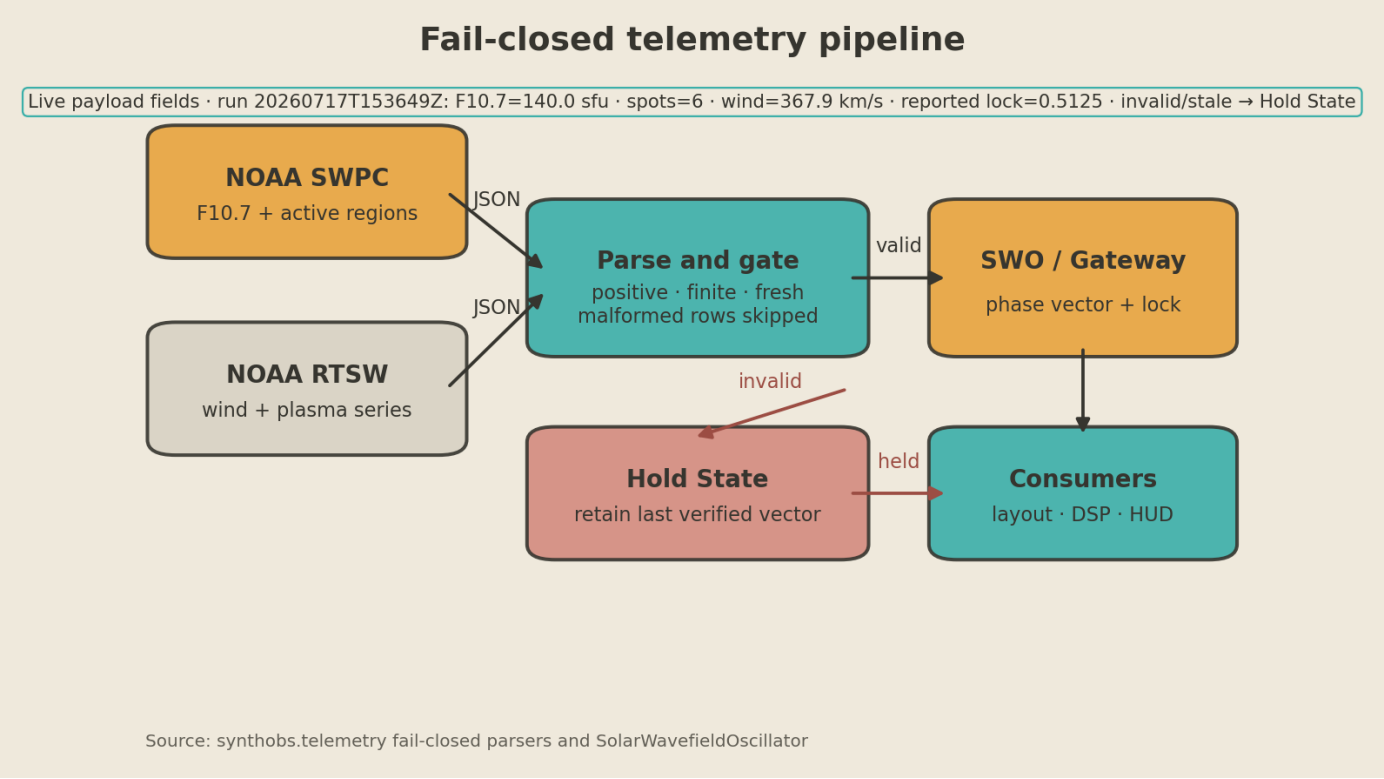


Figure 13: Rendered fail-closed NOAA-to-SWO telemetry pipeline. F10.7/active-region and RTSW wind streams enter the same positive/finite/fresh gate; valid data reaches the SWO/Gateway, while malformed or stale data takes the red Hold State path and retains the last verified vector. The run-specific banner reports the current live payload fields (140.0 sfu, 6 active regions, 367.9 km/s, reported lock 0.5125) without presenting them as a population statistic.

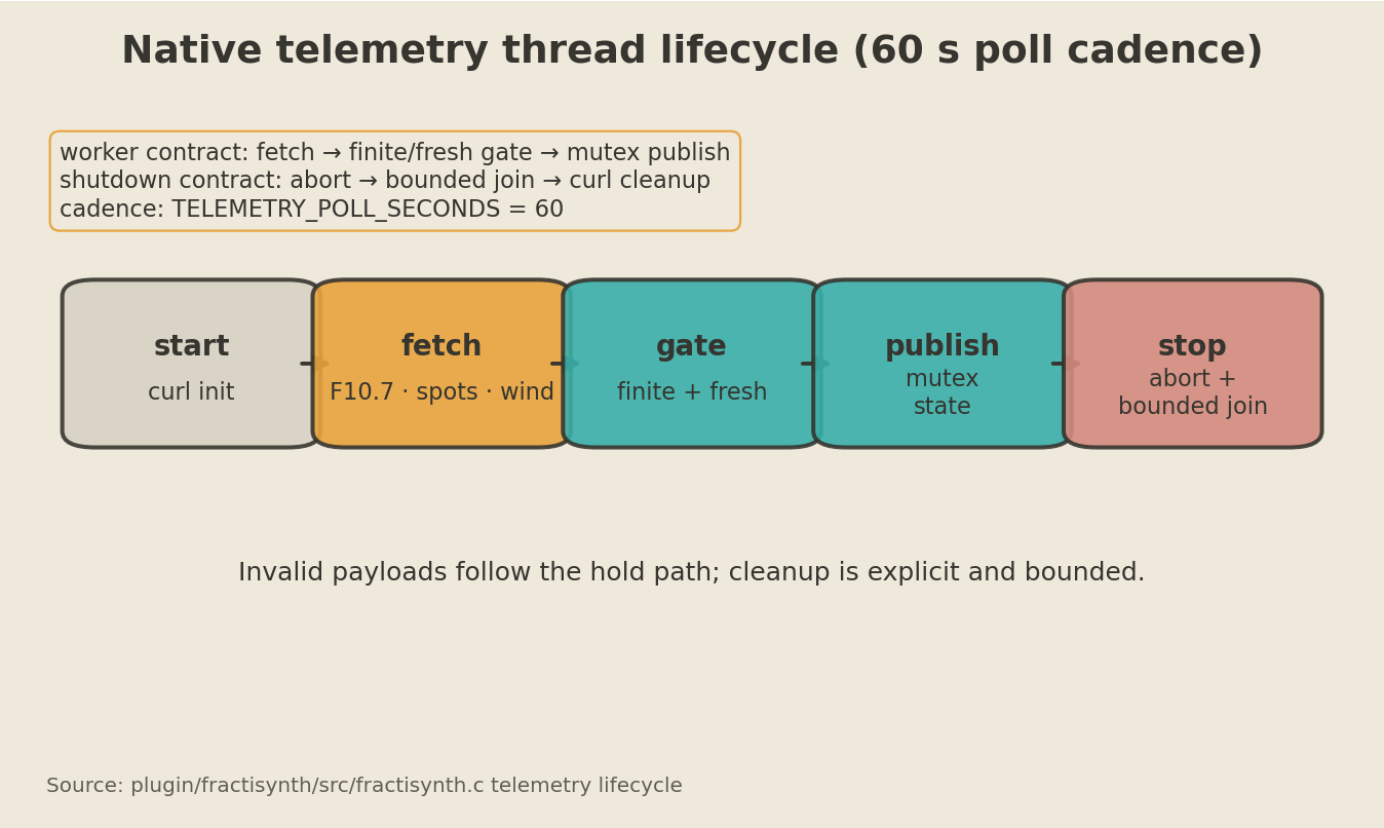


Figure 14: Rendered native telemetry-thread lifecycle. The five stages expose the operational sequence—curl initialization, three-source fetch, finite/fresh gate, mutex-protected publish, and abort/bounded join—and the annotation gives the implemented 60-second poll cadence. The figure distinguishes the worker's normal data path from its explicit shutdown path, making the cleanup obligation visible rather than leaving it in prose.

The gateway resolves not to a Boolean but to a **holographic interference verdict** — `holographic_gate()` reports constructive interference at the `AR14409` solar node as the affirmative branch (`CONSTRUCTIVE_AR14409`), a destructive hydrogen phase-flip as the negative (`DESTRUCTIVE_H_PHASE_FLIP`), and a within-margin beat as `MIXED`. That lock strength then drives the live shader's declared effects: the φ-spiral displacement, the K_{EGS}-spaced interference fringes, the H-alpha-inspired tint, and a gentle breathing pulse at the configured 29.94 Hz design frequency.

The two telemetry planes are fully decoupled — each fails closed and holds independently — and converge only at the holographic gate that drives the shader:

6.5 Command-Line Calibration Override

The global terminal accepts a live override that flows through the same fail-closed calibrator:

```
/swo calibrate --flux=130 --spots=3 --target=AR4465
```

Non-physical overrides (`--flux=-1`, `--spots=0`) are rejected at parse time — the terminal never silently no-ops.

```
flowchart TB
    F107["F10.7 flux + active spots<br/>(synthobs.telemetry)"]
    WIND["live solar wind v_wind<br/>(synthobs.telemetry)"]

   F107 -->|"flux &gt; 0 and spots &gt; 0<br/>else HOLD"| AMP["Amplitude plane<br/>SolarWavefieldOscillator.calibrate()<br/>system_
    WIND -->|"v_wind &gt; 0<br/>else HOLD"| PHASE["Phase plane<br/>gateway_filter()<br/>lock = |cos θ|, K_EGS-weighted"]

    AMP --> GATE
    PHASE --> GATE

    GATE{"holographic_gate()<br/>interference verdict"}
    GATE -->|"constructive @ AR14409"| TRUE["CONSTRUCTIVE_AR14409<br/>(affirmative)"]
    GATE -->|"H phase-flip"| FALSE["DESTRUCTIVE_H_PHASE_FLIP<br/>(negative)"]
    GATE -->|"within margin"| MIX["MIXED"]

    TRUE --> SHADER["Live shader<br/>φ-spiral · K_EGS fringes ·<br/>H-alpha tint · 29.94 Hz breath"]
    FALSE --> SHADER
    MIX --> SHADER
```

Figure 15: Independent telemetry planes: positive finite flux/spots and solar-wind inputs either update their respective SWO and gateway states or hold, then feed a named interference verdict and the live shader.

7 Command Line Grammar Structure

A persistent terminal line is mounted at the base of the Vessel Console and holds active focus at all times. At any instant the operator can bypass the button decks entirely and speak directly to the wavefield through clean, deterministic syntax — the same channel whether the system is in Observatory, Laboratory, or Expedition trim. The grammar is implemented by the tested `synthobs.commands` parser, which returns a typed, frozen command object and fails closed on any unknown verb or out-of-range value. Nothing in this layer is interpreted as a "best guess": a line either resolves to one exact intent or it resolves to nothing at all.

Formally, the parser is the total map

$$\mathrm{parse} : \Sigma^{*} \longrightarrow \mathcal{C} \uplus \{\,\bot\,\}, \tag{22}$$

where Σ^{*} is the set of all input lines, $\mathcal{C}$ is the closed set of typed commands $\{\mathsf{ModeCommand}, \mathsf{BindCommand}, \mathsf{CalibrateCommand}, \mathsf{DashboardCo}$ and $\bot$ is the fail-closed outcome — raised in code as `CommandError`. Equation eq. 22 is the safety spine of the console: the image of every line is either a fully-validated command or an explicit refusal, never a silent no-op.

The fail-closed branch is reached precisely when the line is empty, the verb is unknown, or any argument falls outside its admissible range:

$$\mathrm{parse}(\ell) = \bot \iff \ell = \varepsilon \ \vee\ v(\ell) \notin V \ \vee\ \neg\,\mathrm{valid}\big(\mathrm{args}(\ell)\big), \tag{23}$$

with ε the empty line, $v(\ell)$ the leading verb, and $V = \{\,$`/mode`, `/transducer`, `/swo`, `/dashboard`$\,\}$ the registered verb set. Equation eq. 23 is enforced verb-by-verb below and rendered in Figure fig. 16.

The complete grammar, including dashboard planning/building and telemetry examples, lives in the parse-pipeline module. The macro/filter-routing material lives in the adjacent command module.

8 Command Grammar: Macros and Filter Routing

8.1 Mode Switching Macros

```
/mode --observatory
/mode --lab
/mode --ship
```

Each macro resolves to exactly one of the three valid decks. Writing M for the deck set, the resolution is a lookup into the fixed flag table:

$$\text{mode}(\ell) = \begin{cases} \text{OBSERVATORY} & \text{if flag} \in \{\texttt{--observatory}\}, \\ \text{LABORATORY} & \text{if flag} \in \{\texttt{--lab},\ \texttt{--laboratory}\}, \\ \text{EXPEDITION} & \text{if flag} \in \{\texttt{--ship},\ \texttt{--expedition}\}, \\ \bot & \text{otherwise.} \end{cases} \tag{24}$$

An unrecognized target falls to the $\bot$ branch of eq. 24 and raises a command error rather than leaving the console in an ambiguous trim. The aliases (`--lab`/`--laboratory`, `--ship`/`--expedition`) are accepted so spoken shorthand and full names land on the same deck.

8.2 Dynamic Filter Routing via the EGS Fractal Constant

```
/transducer bind source_cam_01 --ratio=1.618034
```

Binds a named source into the transducer matrix at the requested golden ratio. The default operating point is $\varphi = 1.6180339887\ldots$ — El Gran Sol's Fractal Constant for *layout* — but the operator may dial any strictly positive ratio. The bind is admitted only when both the source name and a positive ratio are present:

$$\text{valid}_{\text{bind}} \iff (\,\text{positional}_0 = \texttt{bind}\,) \ \wedge\ (\,\text{source} \neq \varepsilon\,) \ \wedge\ (\,r > 0\,), \qquad r \in \mathbb{R}. \tag{25}$$

A missing source, a non-numeric ratio, or any $r \leq 0$ violates eq. 25 and routes to $\bot$ — the matrix never binds against a degenerate or sign-flipped scale.

9 Command Grammar: Telemetry, Parse Pipeline, and Summary

9.1 Live Telemetry Overrides

```
/swo calibrate --flux=130 --spots=3 --target=AR4465
```

Forces a manual Solar Wavefield Oscillator calibration, injecting an operator-chosen F10.7 flux and active-region count in place of the live feed. Numeric flags round-trip to their declared types — float `flux`, integer `spots` — and the non-physical region of telemetry space is rejected outright:

$$\text{valid}_{\text{calibrate}} \iff \text{finite}(\text{flux}) \ \wedge \ (\,\text{flux} > 0\,) \ \wedge \ \text{spots} \in \mathbb{Z}_{>0}. \tag{26}$$

Here `finite(flux)` excludes NaN and both infinities; `spots` is admitted only as a positive integer. The parser also rejects duplicate or unknown flags and trailing positionals before this predicate is evaluated.

The constraint in eq. 26 is the same Hold State the oscillator applies to its live feed: a zeroed or negative reading never produces a phase vector. The admitted `(flux, spots)` pair flows into the `system_phase_vector` law of the SWO plane (Chapter 5), and from there the live solar wind drives the EGS Gateway phase lock $|\cos(\text{phase\_bias})|$ governed by the gateway key $K_{\text{EGS}} = \varphi{\cdot}(\lambda_{\text{reader}}/\lambda_{\text{H-alpha}}) \approx 2.539427$. The override can use any positive finite pair supplied by the operator; it is not evidence that the pair is the current NOAA state. It does, however, reuse the same parser and calibrator checks, so it cannot introduce a non-finite, zero, or negative control vector through this command path.

9.2 The Parse Pipeline

Every line walks the same deterministic path: tokenize, dispatch on the verb, partition flags, validate, and either emit a typed command or refuse. The stages map one-to-one onto eq. 22 and eq. 23.

Figure fig. 16 renders the parser's single rejection sink. The accepted typed set is `{ModeCommand, BindCommand, CalibrateCommand, DashboardCommand}`; malformed tokenization, unknown verbs, and invalid verb-specific constraints all terminate as `CommandError`.

The same flow as a control graph, making the single $\perp$ sink explicit:

9.3 Grammar Summary

Verb	Form	Result type	Admission rule
`/mode`	`--observatory | --lab | --ship`	`ModeCommand`	eq. 24
`/transducer`	`bind <source> --ratio=<float>`	`BindCommand`	eq. 25
`/swo`	`calibrate --flux=<float> --spots=<int> [--target=<id>]`	`CalibrateCommand`	eq. 26
`/dashboard`	`plan | build --name=<scene>`	`DashboardCommand`	non-empty name; action is `plan` or `build`

Any other verb, a missing required argument, or an out-of-range value lands on the $\perp$ branch of eq. 23 and raises `CommandError`. The persistent command line therefore can never fail silently — a core safety property of the Vessel Console, holographically continuous with the fail-closed gateway and oscillator planes beneath it.

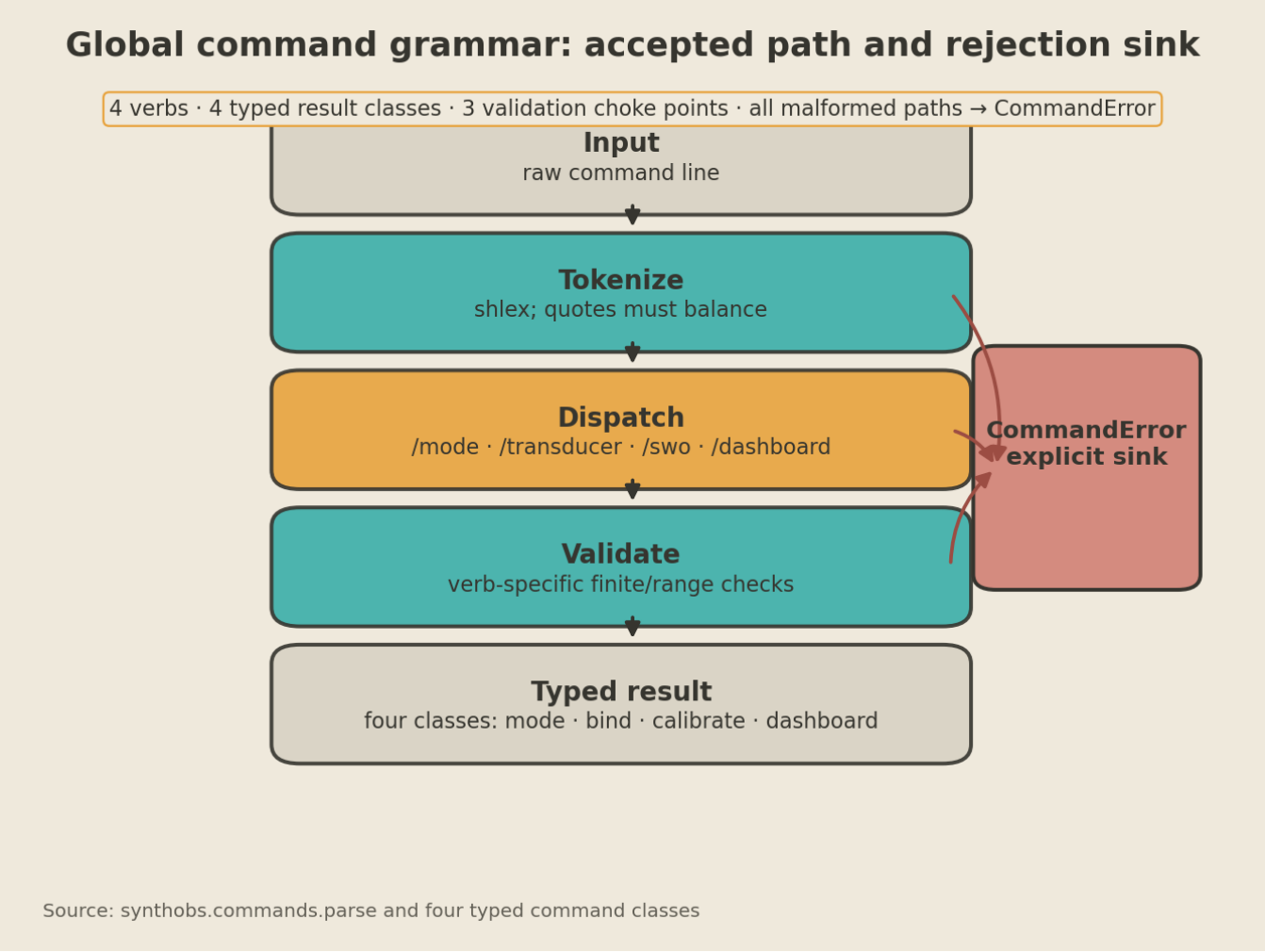


Figure 16: Rendered fail-closed parser pipeline for the complete four-verb grammar: `/mode`, `/transducer`, `/swo`, and `/dashboard`. The vertical path makes the successful sequence—raw line, balanced `shlex` tokenization, verb dispatch, verb-specific validation, and one of four typed command classes—explicit; the red side arrows show that malformed tokenization, unknown verbs, and invalid finite/range constraints all terminate at the same `CommandError` sink. The banner reports the four-verb/four-class contract and the three validation choke points.

```
flowchart TD
    L["input line ℓ ∈ Σ*"] --> T{"tokenize<br/>(shlex)"}
    T -- "empty / unbalanced" --> X["⊥ CommandError"]
    T -- "tokens" --> D{"verb ∈ V?"}
    D -- "no" --> X
    D -- "yes" --> S["split positionals / flags"]
    S --> V{"valid(args)?<br/>Eq. 3–5"}
    V -- "no" --> X
    V -- "yes" --> C["typed Command ∈ 𝒞"]
```

Figure 17: Fail-closed command pipeline: tokenization and verb/argument validation send malformed input to one CommandError sink, while only fully validated lines become one of the four typed command classes.

10 Implementation Blueprint for v1.618

The system is realized as a tested Python reference engine—the portable source of truth—and a native `libobs` plugin that implements the OBS-bound counterparts, assembled in four phases. Each phase is a transmission gate: downstream behavior is admitted only after the relevant contract has resolved.

The implementation and evidence protocol follows reproducible-computing practice: the source tree, tool versions, deterministic figure generator, test command, and live-capture manifest are all named so a reader can distinguish rerunning the same computation from re-establishing the external OBS/NOAA environment [National Academies of Sciences, Engineering, and Medicine, 2019, Wilson et al., 2014, Sandve et al., 2013, Wilkinson et al., 2016]. The OBS module and websocket boundaries are documented against their primary interfaces [OBS Project, 2026a,c].

```
flowchart TB
  P1["<b>PHASE 1 — THE DRIVE ENGINE</b><br/>(FractiSynth Core)<br/>Native C skeleton via CMake against libobs.<br/>Raw texture + au
filter module with the full lifecycle."]
  P2["<b>PHASE 2 — HARMONIC MATHEMATICS LOCK</b><br/>φ = 1.61803398875 scaling across every spatial<br/>vector path and gain stage,
  P3["<b>PHASE 3 — REAL-TIME TELEMETRY CONNECT</b><br/>Background libcurl thread streaming live NOAA<br/>SWPC solar data into the f
closed SWO<br/>amplitude loop and EGS gateway phase loop."]
       P4["<b>PHASE 4 — THE CONSOLE ENCLOSURE</b><br/>(SynthOBS UI)<br/>Goldilocks dynamic layout engine + 3-
mode<br/>console as an obspython script, binding the<br/>interface to the engine's core drive."]
    P1 --> P2 --> P3 --> P4

    classDef phase fill:#0f172a,stroke:#6A0DAD,stroke-width:2px,color:#fff
    class P1,P2,P3,P4 phase
```

Figure 18: Four implementation phases: native libobs lifecycle, shared phi mathematics, live NOAA telemetry admission, and the SynthOBS console enclosure converge into the complete instrument.

10.1 Repository Layout

Path	Role
`src/synthobs/`	Tested Python engine — constants, layout, telemetry, SWO, DSP, console, commands, interaction targets, dashboard layers, engine. The verified source of truth.
`plugin/fractisynth/`	Native `libobs` C plugin — filters, draggable console source, dock, shared SWO, libcurl telemetry thread, CMake build.
`plugin/synthobs/`	`obspython` console script — Goldilocks layout + 3-mode console + command line + dashboard helper, driving the engine inside OBS.
`scripts/`	Thin orchestrators (figure generation) importing the engine.
`manuscript/`	This Technical Design Blueprint.
`tests/`	Real-input test suite, >= 90% coverage on `src/`.

10.2 Calibration Laws Pinned Across Both Implementations

Three formalisms cross the language boundary as declared contracts. Each is defined in the Python engine and implemented at the native `libobs` boundary; the shared literals in `src/synthobs/constants.py` (`PHI_C_LITERAL`, `EGS_GATEWAY_KEY_C_LITERAL`) plus static and behavioral checks make material drift between the two visible.

Phase 2 — amplitude plane. The Solar Wavefield Oscillator maps admitted F10.7 flux and active-region count to the bounded `system_phase_vector`, scaled by the golden key φ (`swo.phase_vector`):

$$v_{\text{phase}} = \frac{\Phi_{F10.7}}{N_{\text{spots}}}\,\varphi \tag{27}$$

Phase 3 — phase plane. The EGS gateway translates the Sun's energetic state into a phase bias on the virtual 1030 nm reader, weighted by the EGS Fractal Constant / gateway key $K_{\text{EGS}} = \varphi\,(\lambda_{\text{reader}}/\lambda_{\text{H-alpha}}) \approx 2.539427$. The bias wraps onto the circle and the gateway reports the current model score (`gateway.gateway_filter`):

$$\theta_{\text{bias}} = \left(2\pi\,\frac{w}{w_{\text{ref}}}\,K_{\text{EGS}}\right) \bmod 2\pi, \qquad L = |\cos\theta_{\text{bias}}| \tag{28}$$

The lock strength $L \in [0, 1]$ of eq. 28 resolves holographically, not as a Boolean: constructive interference at the AR14409 node reads "true", a destructive hydrogen phase-flip reads "false", and a tie reads "mixed". Both eq. 27 and eq. 28 fail closed — a non-physical input ($\Phi_{F10.7} \leq 0$, $N_{\text{spots}} \leq 0$, or $w \leq 0$) never produces a lock; the engine holds its last verified value rather than substituting an average or a zero.

Phase 2 — gain stage. Before audio reaches the encoder, the φ soft-limiter shapes each excess sample so its magnitude approaches but never crosses the headroom ceiling ($1/\varphi$ knee, `dsp.phi_soft_limit_sample`):

$$y = \operatorname{sgn}(x)\left(k + h \cdot \tanh\frac{e}{h\,\varphi}\right), \quad k = \frac{\tau}{\varphi}, \quad h = \tau - k = \frac{\tau}{\varphi^2}, \quad e = |x| - k \tag{29}$$

where τ is the threshold, k the $1/\varphi$ knee below which samples pass through unchanged, and e the excess above it. The tanh shaping of eq. 29 guarantees a monotone, sign-preserving transfer curve whose magnitude approaches but never reaches τ.

10.3 Build, Test, Install

```
# Test the reference engine (pytest-httpserver provides real local HTTP)
uv run python -m pytest tests/ \
  --cov=synthobs --cov-fail-under=90

# Regenerate the figures in this document
uv run python scripts/generate_figures.py

# Build the native FractiSynth plugin (requires libobs dev headers + libcurl)
plugin/fractisynth/build.sh

# OBS script commands include:
# /dashboard plan --name=Awareness
# /dashboard build --name=Awareness
```

10.4 Verification Status

Every executable geometric, telemetry, and DSP claim in this blueprint is exercised by the real-input test suite over the reference engine—including the three pinned laws eq. 19, eq. 21, and eq. 29 and their fail-closed boundaries. The current gate is 1217 project tests at 96.09% coverage, thirteen deterministic engine-derived figures plus three versioned live OBS captures, the native `plugin/fractisynth/build.sh` build, and a buildable Lean scaffold for the structural invariants that should never drift. The native C plugin shares the φ literal, selected SWO and gateway formulas, the seven-feed target geometry, Solar Graph X-ray/Kp metrics, metric-aware graph horizons, Zoom Inspector target modes, dock theme/precision controls, the φ-soft-limiter RMS/peak/reactivity envelope, the LSB provenance payload layout, and the fail-closed rule with the engine. The Python engine therefore remains authoritative; the C plugin is an OBS-bound implementation whose covered contracts are checked, never a second source of truth.

The live OBS scenario gate is no longer a log-only claim. A real OBS 32.1.2 session loaded the installed FractiSynth bundle, accepted the `fractisynth_console` source, captured compositor pixels through obs-websocket, drove a controlled audio tone through the harmonic limiter, and verified the Telemetry HUD provenance strip from the PNG itself. The manifest gate passed connection, canvas fit, nonblank rendered content, engine-level interaction resolution, audio-meter delta, and LSB provenance in run `20260717T153649Z`; the manifest does not claim live click transport through an OBS Interact window.

The three captures above are load-bearing run evidence. The following image is deliberately a different evidence class: a user-supplied view of the host window that makes the operator context legible at a glance. It is useful for judging composition, telemetry readability, and the relationship between the SynthOBS HUD, the OBS mixer, and the host controls, but it does not replace the versioned PNG hashes, audio ROI oracle, or provenance verifier.

A fair-exchange clause is in effect for this architectural expansion. Adjustments, refinements, or partial revisions to the delivery scale can be handled through subsequent collaborative feedback.

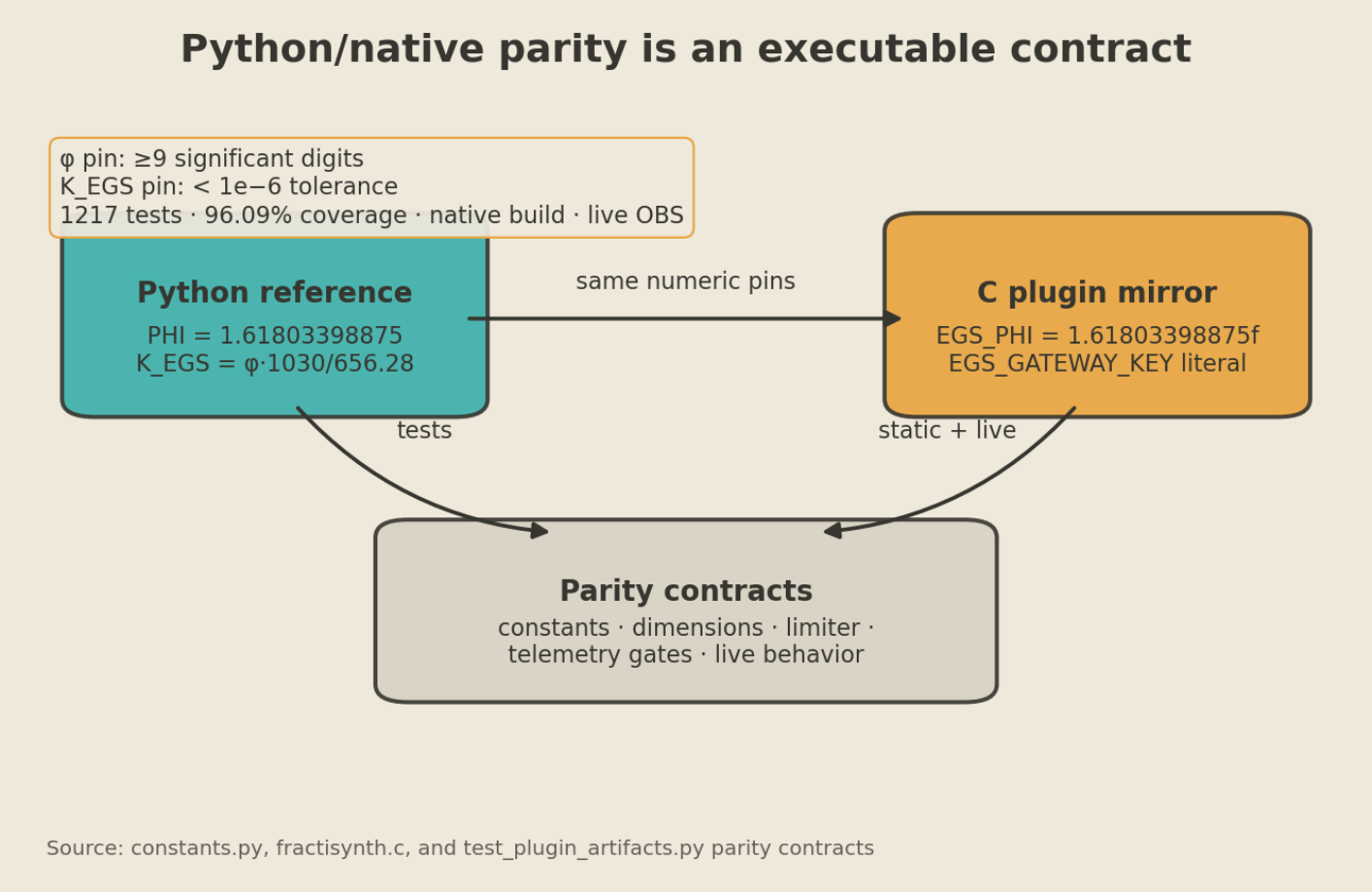


Figure 19: Python/native contract boundary. The tested Python reference and the C plugin exchange pinned φ and K_{EGS} literals, then converge on executable contracts for constants, dimensions, limiter behavior, telemetry gates, and live behavior. The annotation records the acceptance thresholds—at least nine significant digits for φ and less than 10^{-6} for K_{EGS}—alongside the current 1217-test, 96.09%-coverage, native-build, and live-OBS evidence baseline.

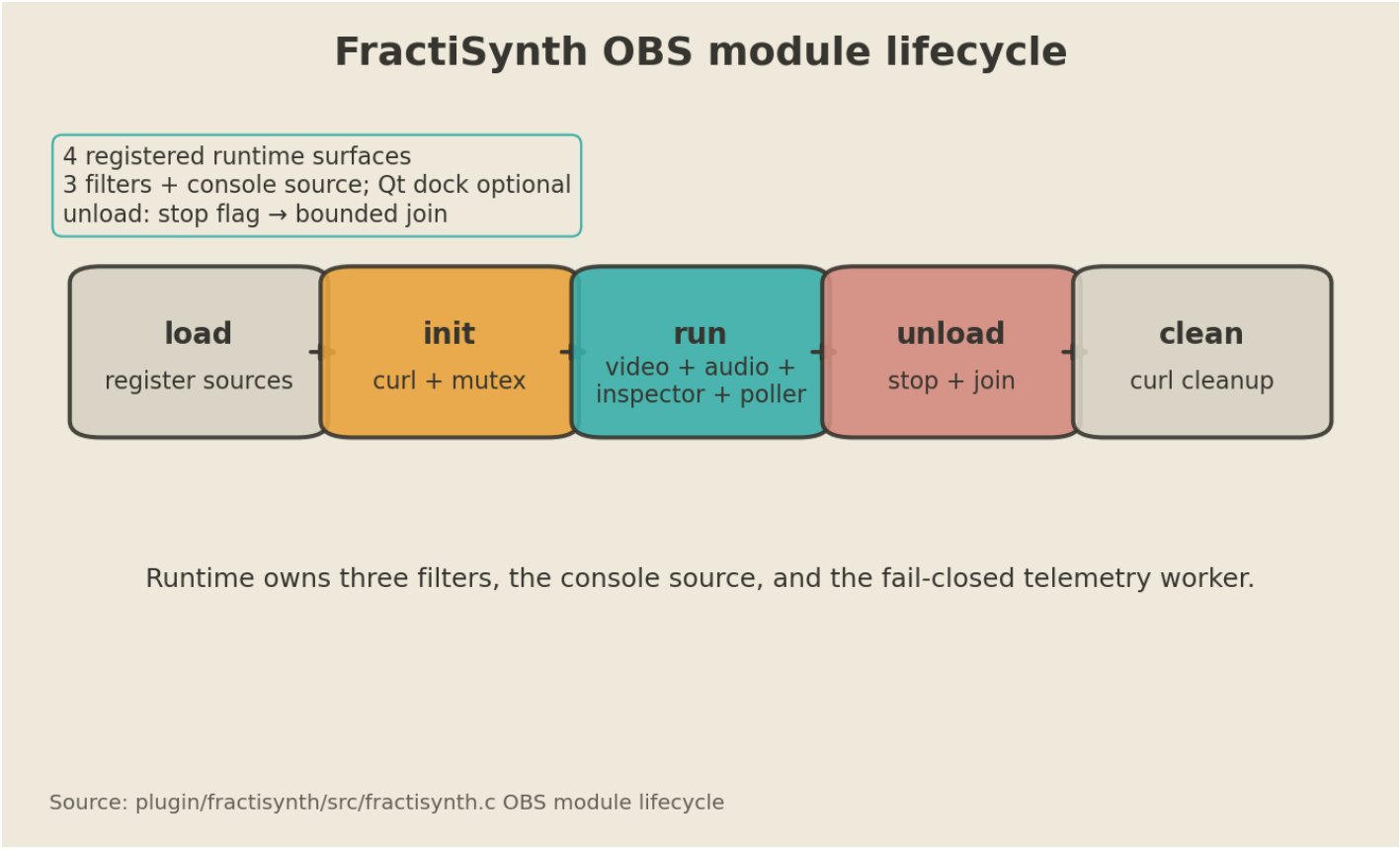


Figure 20: FractiSynth OBS module lifecycle. The five states separate module load and source registration, curl/mutex initialization, concurrent video/audio/inspector rendering plus telemetry polling, bounded unload, and final curl cleanup. The upper annotation records four registered OBS source surfaces—three filters and the console source—and identifies the Qt dock as an optional frontend surface rather than counting it as a source registration.

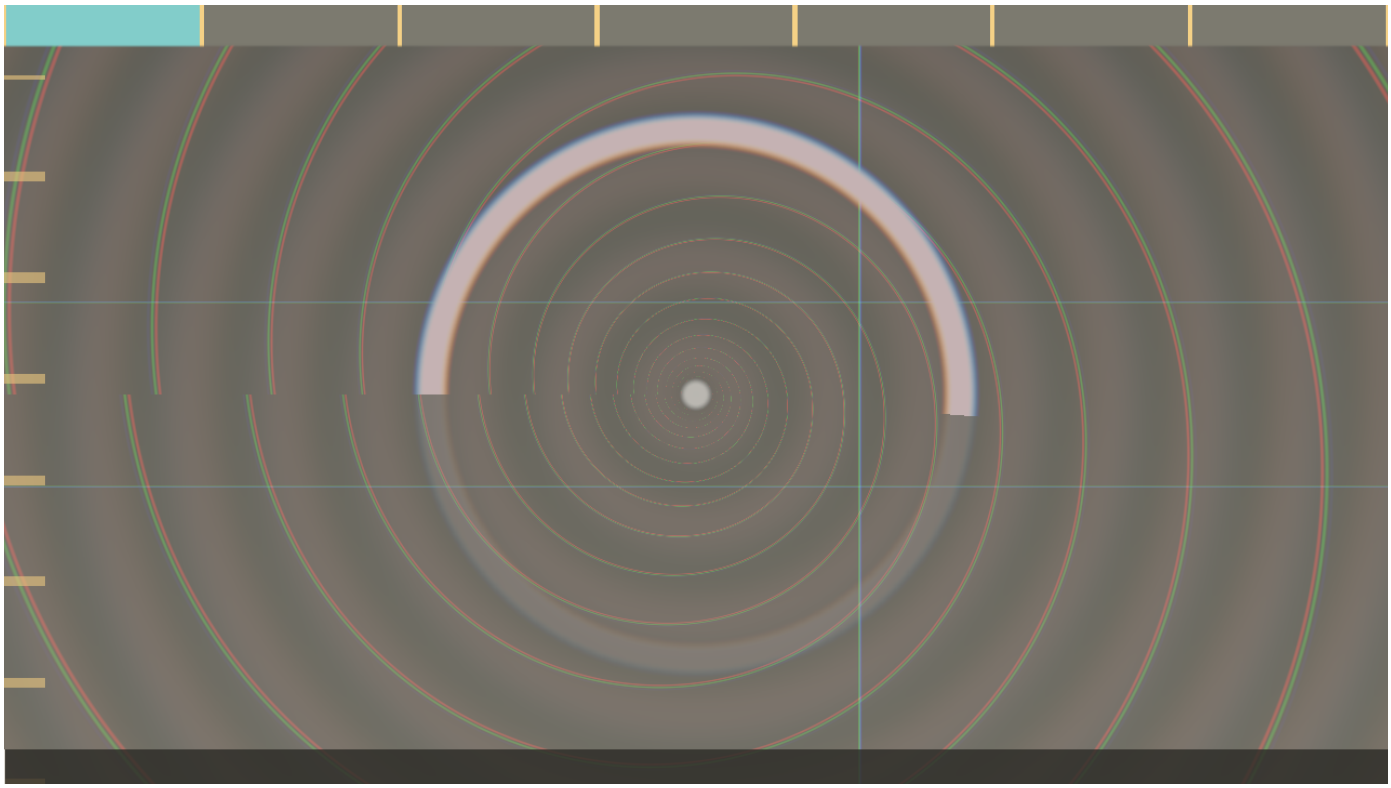


Figure 21: Live OBS compositor capture from scenario run `20260717T153649Z`. OBS Studio 32.1.2 renders the installed `fractisynth_console` source on a 3200×2000 base canvas, then obs-websocket captures a 1280×720 frame at the manifest-recorded scale factors (2.5× horizontal, 2.7778× vertical). The byte-hashed PNG passes the nonblank-content gate with dynamic range 161.93 and mean channel standard deviation 19.49; it is also the manuscript cover asset.

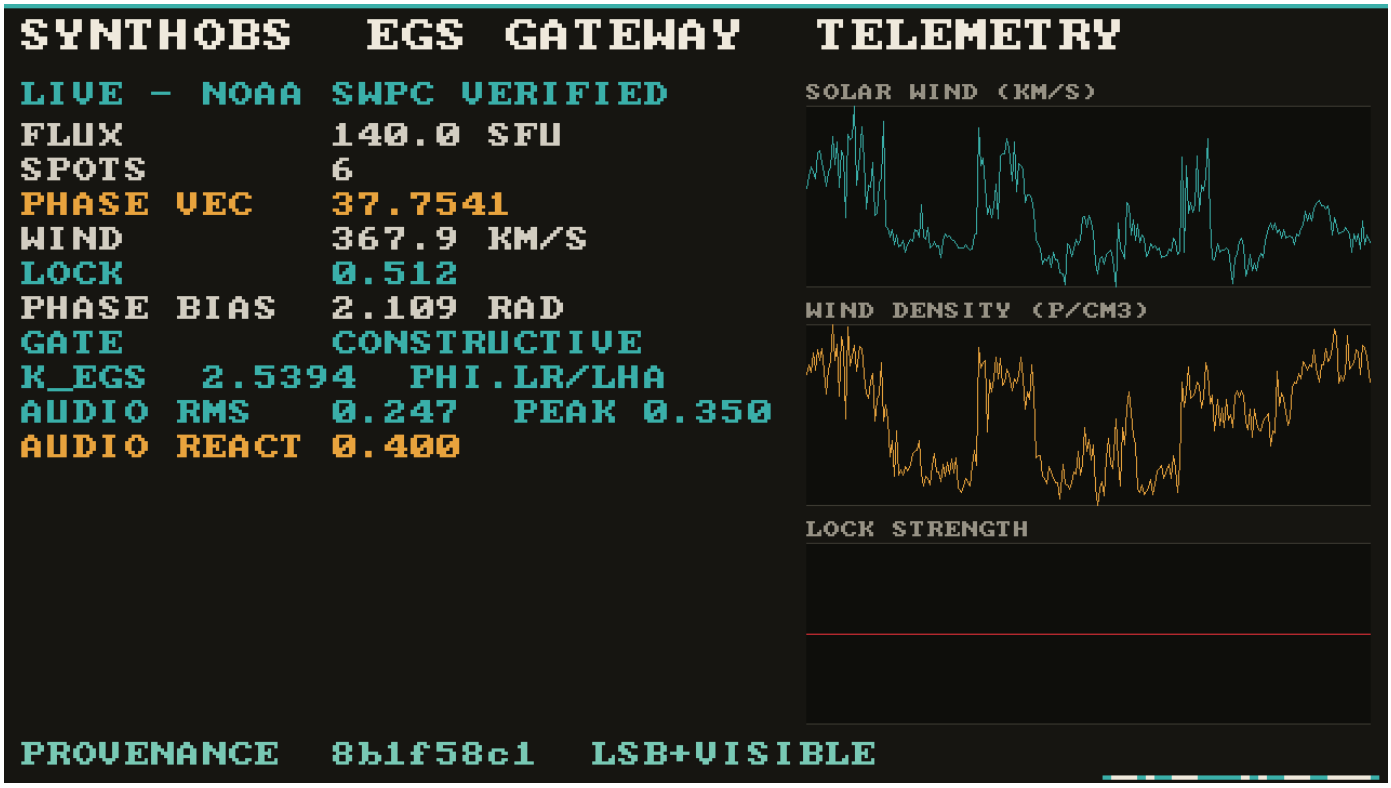


Figure 22: Telemetry HUD capture from the same run. Blue-channel LSB extraction and checksum verification recover F10.7 flux 140.0 sfu, 6 active regions, solar-wind speed 367.9 km/s, lock strength 0.5125, phase bias 2.1089 rad, observation time, and visible signature `8b1f58c1`; the exact float values, asset hash, and gate result are recorded in `manuscript/assets/obs/obs_manifest.json`. The image demonstrates a recoverable record, not authenticated authorship.

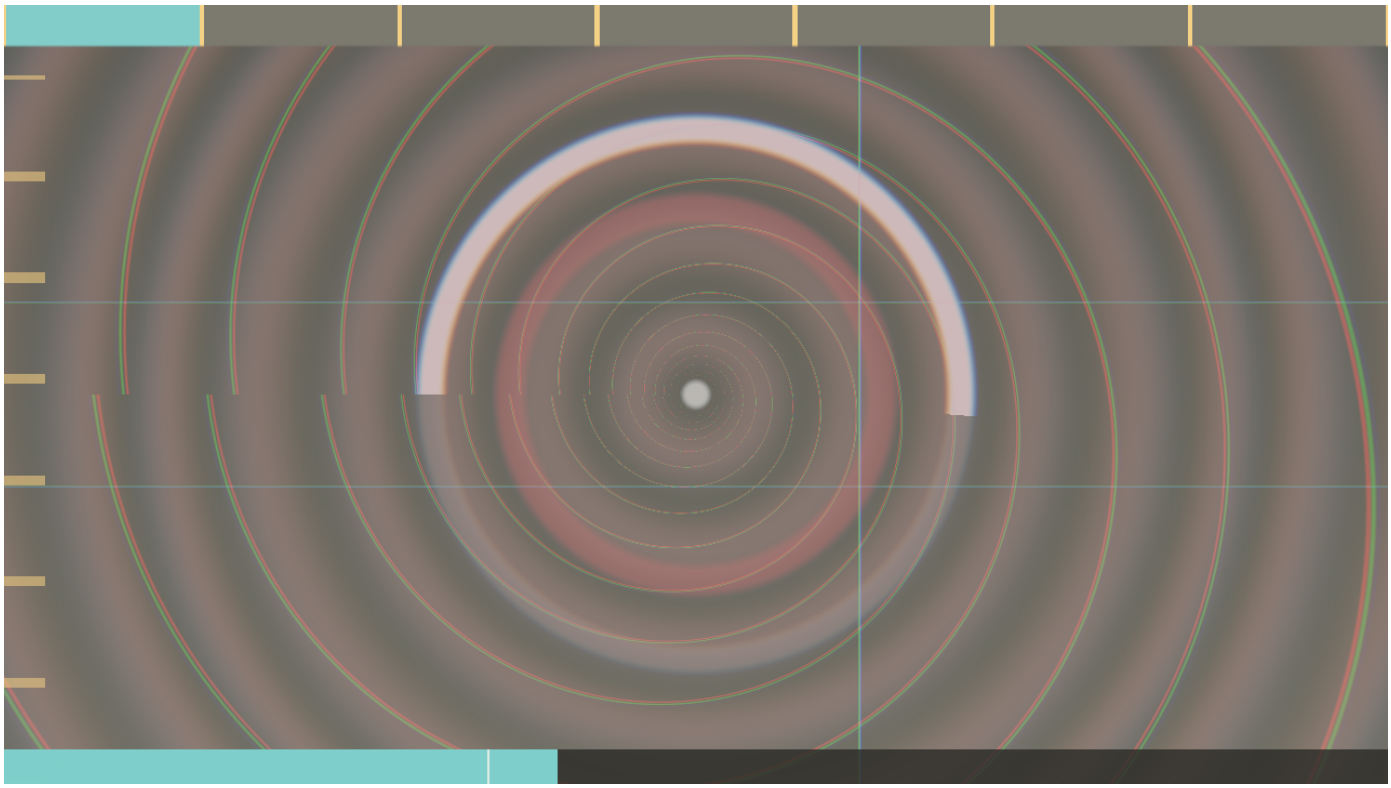


Figure 23: Controlled-tone OBS capture used by the audio-reactivity gate. The verifier compares this frame with the versioned silent baseline over the 1280×33 bottom ROI of the 1280×720 image; the recorded mean absolute RGB delta is 48.5574 against an 8.0 threshold, with a maximum channel delta of 180. The source PNG remains the exact capture, while the manifest supplies the numerical interpretation.

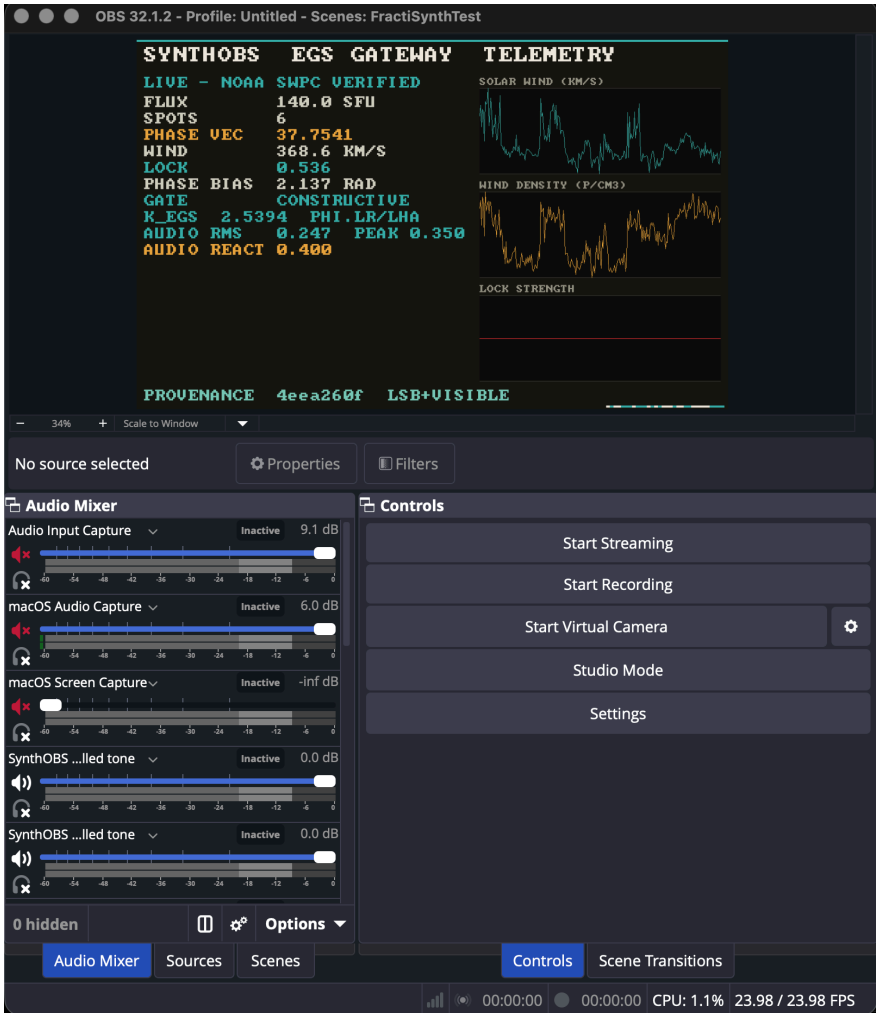


Figure 24: Operator-context OBS Studio window capture supplied with the manuscript review. OBS Studio 32.1.2 is visible around a SynthOBS/FractiSynth HUD, audio mixer, and host controls, showing how the instrument sits inside an operator's working surface. This contextual screenshot is not used to establish the current run's telemetry values, render metrics, audio delta, or provenance result; those claims remain bound to the six-gate manifest and its exact assets.

11 Evaluation, Evidence, and Reproducibility

This chapter turns the SynthOBS blueprint into an inspectable research artifact. The contribution is a bounded engineering system with four linked surfaces: a dependency-free Python reference engine, a native OBS transducer, an obspython operator bridge, and a live evidence protocol. The claim is not that a geometric constant explains broadcasting in general. The design proposition is narrower and testable: a single pinned constant, explicit fail-closed state transitions, and versioned captures make a cross-language OBS instrument easier to reason about and re-run.

That distinction follows the reproducibility literature's separation of rerunning a computation from reproducing an empirical environment [National Academies of Sciences, Engineering, and Medicine, 2019]. It also follows software-practice guidance to record dependencies, automate the path from source to artifact, and preserve the data and procedures needed for inspection [Wilson et al., 2014, Sandve et al., 2013, Wilkinson et al., 2016]. The provenance vocabulary is contextualized with W3C PROV-O [W3C Provenance Working Group, 2013], while the figure generator follows the scientific-visualization software lineage documented for Matplotlib [Hunter, 2007].

11.0.1 Public distribution and affiliation

SynthOBS is authored under **FractiAI / Active Inference Institute**. The planned public v1 distribution is the GitHub repository `docxology/SynthOBS`, which is intended to carry the complete source, native OBS module, obspython bridge, tests, installation and usage documentation, manuscript sources, citation metadata, and versioned live evidence. The repository-level release contract in `RELEASE.md` defines the clean-clone preflight and the remaining public-release gates.

11.1 Contributions and system boundaries

The artifact makes five concrete contributions:

1. `src/synthobs/` defines the geometry, telemetry gates, DSP, console shape, and command grammar without OBS or third-party runtime imports.
2. `plugin/fractisynth/` implements the engine's declared contracts at the `libobs` video/audio boundary, with libcurl-backed live telemetry and bounded shutdown.
3. `plugin/synthobs/` exposes the same state transitions through an obspython console, including the four typed verbs `/mode`, `/transducer`, `/swo`, and `/dashboard`.
4. `scripts/generate_figures.py` derives deterministic visualizations from the tested engine and records each figure's source in `../figures/figure_manifest.json`.
5. The live OBS bundle in `assets/obs/` preserves real compositor, telemetry-HUD, and audio-reactivity captures with hashes and six gate results.

The boundary is intentional. Python tests establish mathematical and parser behavior; static C probes establish native contract structure; a real OBS session establishes that the installed binary loads, renders, reacts to controlled audio, and preserves telemetry provenance. None of these evidence classes silently substitutes for the others.

11.2 Scholarship and evidence discipline

Scholarship is used here as an audit layer, not as decoration and not as a substitute for a failing test. The manuscript follows a four-part claim protocol: an external source establishes context when a claim depends on an interface, standard, data product, or established method; a repository source of truth states how SynthOBS implements the claim; an executable test or contract supplies a way to falsify it; and a rendered artifact or live capture is added only when the claim crosses the OBS boundary. This is the practical separation between reproducibility and replicability described by the National Academies [National Academies of Sciences, Engineering, and Medicine, 2019], made concrete through software-practice guidance [Wilson et al., 2014, Sandve et al., 2013] and FAIR stewardship [Wilkinson et al., 2016].

The source ledger in `docs/scholarship_sources.json` records that protocol for the material claims in this blueprint. Its source order is deliberately conservative: primary OBS and NOAA documentation grounds interface and data-product statements [OBS Project, 2026a,c, NOAA SWPC, 2026d,a,c,b], standards ground provenance vocabulary [W3C Provenance Working Group, 2013], and peer-reviewed or established technical sources provide method context for visualization, DSP, architecture, and the history of the golden ratio [Hunter, 2007, Smith, 2007, Gamma et al., 1994, Livio, 2002]. Those citations do not prove that this implementation works. The local engine, tests, manifests, hashes, and captures do that within the declared boundary.

11.3 A formal claim-to-evidence calculus

To keep the evidence language composable, let each material claim c carry a four-tuple

$$E(c) = (S(c), R(c), T(c), A(c)), \tag{30}$$

where $S(c)$ is the external source or standard, $R(c)$ is the repository source of truth, $T(c)$ is the executable test or contract that can falsify the implementation, and $A(c)$ is a rendered or live artifact when the claim crosses the OBS boundary. For a purely internal deterministic claim, $A(c)$ may be empty. For a runtime claim, the admissibility condition is

$$\text{admissible}(c) = S(c) \wedge R(c) \wedge T(c) \wedge A(c). \tag{31}$$

The conjunction is a documentation rule, not a claim that four independent sources make a result true. It prevents evidence substitution: $S(c)$ can explain an API, but cannot prove this binary loaded; $T(c)$ can exercise a parser, but cannot prove that a live OBS frame contains the parser's state; and $A(c)$ can show a frame, but cannot establish the rejection of malformed input. The machine-readable ledger encodes the tuple in `sources`, `source_of_truth`, `tests`, and `artifacts`, making the argument inspectable by both humans and automation.

This calculus also clarifies the relationship between scholarship and software citation. The external literature establishes why versioned, attributable, accessible software records matter [Smith et al., 2016]; the CFF metadata records the current software identity [Citation File Format Initiative, 2026]; the repository and manifest record what this version actually did. The bibliography therefore supports the method of accounting, while the local evidence remains responsible for the implementation claim.

Claim class	External scholarship contributes	Repository evidence must contribute
Interface contract	Meaning of the OBS module and websocket surfaces	Native artifact probes, live connection, and the manifest's gate record
External data provenance	Identity and semantics of the NOAA products and `libcurl` transport	Fail-closed parsers, freshness checks, and the recovered HUD record
Design proposition	Method context for φ geometry, DSP, architecture, and visualization	Equations, source modules, regression tests, and generated figure manifest
Run observation	Reproducibility and provenance vocabulary	Exact bytes, versions, timestamps, hashes, and six gate outcomes

This division also prevents a common category error: a visually persuasive frame is not evidence that malformed telemetry is rejected, and a citation to a standard is not evidence that the native plugin loaded into the recorded OBS build.

11.4 Materials and versions

Material	Version or value	Role
Python engine	package `1.618.0`, `src/synthobs/`	reference implementation
Author affiliation	FractiAI / Active Inference Institute	publication metadata and software provenance
Test collection	1217 tests, 96.09% coverage	unit, parser, integration, static artifact, scholarship, public-release, and verification contracts
OBS Studio	32.1.2	live host and compositor
obs-websocket	5.7.3	scene/source control and screenshot capture; the protocol surface is documented by the OBS project [OBS Project, 2026c]
OBS base canvas	3200 × 2000	source composition canvas
manuscript captures	1280 × 720	downsampled scene, HUD, and tone evidence
NOAA products	F10.7, active solar-region report, RTSW wind/plasma	live external inputs [NOAA SWPC, 2026a,c,b,d]
Native transfer	libcurl	bounded HTTP client [Stenberg and curl contributors, 2026]

The evidence bundle was captured in run `20260717T153649Z` after rebuilding and installing the native FractiSynth bundle against OBS 32.1.2. Its manifest is the machine-readable record of the run: `obs_manifest.json`. The exact PNG bytes, silent baseline, and controlled WAV are checked into the manuscript asset directory, so the audio delta can be independently recomputed and the cover and generated figure directory are derived from the same real OBS run.

11.5 Live OBS protocol

The acceptance path uses a real OBS process and a real obs-websocket connection. The probe creates or selects an isolated verification scene, fits the console source to the base canvas, exercises the interaction model, captures compositor pixels, and compares silent and controlled-tone audio states. The telemetry HUD is then inspected for its embedded provenance record. This follows the documented OBS module boundary rather than treating a standalone screenshot or a source-local render as proof [OBS Project, 2026a,c].

11.5.1 Gate results

All six required gates passed in the versioned run `20260717T153649Z`.

Gate	Result	Exact evidence
`connection`	pass	OBS identified; websocket `5.7.3`; OBS `32.1.2`
`dashboard_fit_to_canvas`	pass	`positionX=0.0`, `positionY=0.0`, `scaleX=2.5`, `scaleY=2.7777777777777777`
`interaction_model`	pass	engine resolver covers feed tabs, layer rail, and marker drop; OBS click transport is not claimed
`render_content`	pass	1280 × 720; `non_black_fraction=1.0`; `dynamic_range=161.9278106689453`; `mean_channel_std=19.49281120300293`
`audio_reactivity`	pass	ROI `y=687`, height `33`; `mean_abs_delta=48.557402146464646`; `max_abs_delta=180`; threshold `8.0`; 42,240 pixels compared
`provenance_lsb`	pass	flux `140.0`; sunspots `6`; wind `367.8999938964844 km/s`; lock `0.5124873518943787`; phase bias `2.108875274658203 rad`; signature `8b1f58c1`

11.5.2 Formal acceptance definitions

The live gates are numerical predicates over captured bytes, not visual judgments. For a capture of width w and height h, let I_0 and I_1 denote the silent and controlled-tone RGBA images, and define the shader-pinned bottom region

$$R = \{(x, y) : 0 \le x < w,\ \lfloor 0.955h \rfloor \le y < h\}. \tag{32}$$

The audio oracle ignores alpha and computes the mean absolute RGB change

$$D_R = \frac{1}{3|R|} \sum_{(x,y)\in R} \sum_{c\in\{r,g,b\}} |I_1(x, y, c) - I_0(x, y, c)|\,, \qquad M_R = \max_{(x,y)\in R,\, c\in\{r,g,b\}} |I_1(x, y, c) - I_0(x, y, c)|\,. \tag{33}$$

The audio gate passes exactly when $D_R \ge 8.0$; M_R, the ROI dimensions, and the input channel count remain diagnostic metrics. For the current 1280 × 720 capture, eq. 32 gives $R = (0, 687, 1280, 33)$, so $|R| = 42{,}240$ pixels.

The render-content gate uses luminance

$$Y = 0.2126R + 0.7152G + 0.0722B, \qquad q = \text{mean}(Y > 8), \qquad \Delta_Y = \max(Y) - \min(Y), \tag{34}$$

and the mean of the per-channel spatial standard deviations s. It passes iff $q \ge 0.08$, $\Delta_Y \ge 24$, and $s \ge 3$. Thus the evidence establishes non-degenerate compositor content without pretending that a single frame is a performance distribution.

The provenance gate is byte-exact. Let b be the 24-byte little-endian `TelemetryRecord` containing flux, active-region count, wind speed, lock strength, phase bias, and observation time. The default payload is

$$p = b \,\|\, \text{SHA256}(b)[0:4], \tag{35}$$

embedded in the blue-channel least-significant bits. Verification extracts p, recomputes the four-byte digest prefix, and validates the decoded fields before reporting the visible eight-hex-character prefix. Equation eq. 35 detects accidental corruption or resampling; it is intentionally not a keyed authenticity claim. The optional HMAC mode is a separate operator-controlled path and is not used to inflate the current six-gate result.

Three machine-verifiable visual captures are shown in the implementation chapter: the scene render (Figure fig. 21), telemetry HUD (Figure fig. 22), and controlled audio tone (Figure fig. 23). The silent baseline and controlled WAV are versioned companion inputs rather than decorative figures. The scene render is also the cover image. The user-supplied OBS window capture (Figure fig. 24) is a separate operator-context artifact: it communicates host UI placement and observability, but its visible numbers are not substituted for the manifest metrics and it is not one of the six machine gates. The asset hashes and media metadata in `obs_manifest.json` provide an additional byte-level check against accidental replacement.

11.6 Claim-to-evidence map

Design claim	Source of truth	Automated test or contract	Live or rendered evidence
One φ across layers	`src/synthobs/constants.py`, `fractisynth.c`	Python/C literal parity and gateway-key tests	`parity_bridge.png`; native build
Exact golden tiling	`layout.py`	`tiles_exactly()` sweep and layout tests	`goldilocks_layout.png`, `golden_split.png`
φ-scaled DSP	`dsp.py`	limiter bounds, monotonicity, matrix, and envelope tests	`phi_soft_limiter.png`, `phi_matrix.png`; audio gate
Four-verb fail-closed grammar	`commands.py`	parser acceptance/rejection tests and docs contract	`command_parse_pipeline.png`
Telemetry never mints a default	`telemetry.py`, `swo.py`	malformed, stale, non-finite, and hold-state tests	`telemetry_pipeline.png`; provenance gate
Native lifecycle is bounded	`fractisynth.c`	static artifact probes and source-level lifecycle checks	`plugin_lifecycle.png`; no separate unload gate in the six-gate manifest
OBS compositor contains content	`obs_scenario_probe.py`	render-content gate, independent of source-local output	`obs_scene_render.png`
Audio state reaches the visual meter	`obs_scenario_probe.py`	silent-vs-tone ROI oracle	`obs_audio_tone.png` plus audio gate

This map is deliberately heterogeneous. A passing unit test cannot certify an OBS window loaded a binary, and a compelling screenshot cannot certify malformed telemetry is rejected. The evidence classes converge at the system boundary but remain independently named.

11.7 Reproduction commands and artifact paths

From the project root, the deterministic engine and figure paths are:

```
./.venv/bin/python -m pytest tests --cov=synthobs \
  --cov-report=term-missing --cov-fail-under=90
./.venv/bin/python scripts/generate_figures.py
```

The first command must collect 1217 tests and report 96.09% coverage (and at least 90%). The second command must read the versioned real OBS bundle, verify its hashes and six passing gates, generate the analytical figures, copy the three live captures, and write `../figures/figure_manifest.json`. The operator-context screenshot is intentionally excluded from that generated manifest because it is supplied context, not an analytical figure or a promoted live-gate asset.

The live path requires an installed OBS process and a configured obs-websocket credential; it is intentionally exercised only through the real OBS connection:

```
./.venv/bin/python scripts/obs_scenario_probe.py \
  --out output/live/$(date -u +%Y%m%dT%H%M%SZ) \
  --verify-audio --verify-provenance --require-live
```

The native build and standalone CMake path are documented in `docs/build-and-install.md`. The manuscript renderer consumes `manuscript/config.yaml`; its cover points to `assets/obs/obs_scene_render.png`, while figure references resolve to the generated `../figures/` directory.

11.8 Validation boundaries and future engineering work

The current evidence establishes one real OBS run, not a statistical performance distribution. It does not claim that every OBS release, operating system, audio driver, or NOAA response shape will behave identically. The scoped follow-up work is maintained in `TODO.md`, including cross-platform/headless acceptance, transported interaction clicks, and per-feed/filter visual captures. Native C↔Python parser parity, keyed provenance, and clean-environment wheel reproducibility are now closed with executable evidence recorded in the roadmap.

These are engineering extensions, not retroactive qualifications of the current run. The present artifact is reproducible to the extent stated above: deterministic math and documentation can be rerun locally; the live evidence can be inspected byte for byte; and re-establishing the external OBS/NOAA environment is identified as a separate acceptance activity.

12 References

This bibliography records the external standards, primary data products, and software-practice literature used to delimit the design. In-text citations use the bracketed-key form: official OBS and obs-websocket documentation define the host/plugin boundary [OBS Project, 2026a,c], NOAA SWPC's product catalogue and named feeds define the telemetry provenance [NOAA SWPC, 2026d,a,c,b], and reproducibility guidance motivates the claim-to-artifact protocol [National Academies of Sciences, Engineering, and Medicine, 2019, Wilson et al., 2014, Sandve et al., 2013, Wilkinson et al., 2016]. Software is treated as a first-class, version-specific research output according to the FORCE11 software-citation principles [Smith et al., 2016], and the repository's machine-readable citation metadata follows the Citation File Format schema [Citation File Format Initiative, 2026]. The provenance vocabulary is contextualized with W3C PROV-O [W3C Provenance Working Group, 2013], and the deterministic figure pipeline cites Matplotlib [Hunter, 2007]. The φ and DSP sections cite historical mathematical context and signal-processing foundations [Livio, 2002, Smith, 2007]; the H-alpha anchor is sourced to NIST [Kramida, 2010].

Citation File Format Initiative. Citation File Format schema guide. https://github.com/citation-file-format/citation-file-format/blob/main/schema-guide.md, 2026.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. URL https://www.pearson.com/en-us/subject-catalog/p/design-patterns-elements-of-reusable-object-oriented-software/P200000003241.

John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55. URL https://doi.org/10.1109/MCSE.2007.55.

Alexander Kramida. Hydrogen h-alpha wavelength reference. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=2388, 2010.

Mario Livio. *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number*. Broadway Books, New York, NY, USA, 2002. ISBN 978-0-7679-0816-0. URL https://www.penguinrandomhouse.com/books/102878/the-golden-ratio-by-mario-livio/.

National Academies of Sciences, Engineering, and Medicine. Reproducibility and replicability in science. Technical report, The National Academies Press, 2019. URL https://doi.org/10.17226/25303.

NOAA SWPC. F10.7 cm solar radio flux product. https://services.swpc.noaa.gov/json/f107_cm_flux.json, 2026a.

NOAA SWPC. Real-time solar-wind products. https://services.swpc.noaa.gov/json/rtsw/, 2026b.

NOAA SWPC. Active solar-region report product. https://services.swpc.noaa.gov/json/solar_regions.json, 2026c.

NOAA SWPC. Space Weather Prediction Center real-time json products. https://services.swpc.noaa.gov/json/rtsw/, 2026d.

OBS Project. Obs module api reference. https://docs.obsproject.com/reference-modules, 2026a.

OBS Project. OBS Studio and the libobs plugin api. https://github.com/obsproject/obs-studio, 2026b.

OBS Project. obs-websocket: Remote-control protocol for obs studio. https://github.com/obsproject/obs-websocket, 2026c.

Geir Kjetil Sandve et al. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10):e1003285, 2013. doi: 10.1371/journal.pcbi.1003285. URL https://doi.org/10.1371/journal.pcbi.1003285.

Arfon M. Smith, Daniel S. Katz, Kyle E. Niemeyer, and FORCE11 Software Citation Working Group. Software citation principles. *PeerJ Computer Science*, 2:e86, 2016. doi: 10.7717/peerj-cs.86. URL https://force11.org/info/software-citation-principles-published-2016/.

Julius O. Smith. *Digital Audio Signal Processing*. W3K Publishing, 2007. URL https://ccrma.stanford.edu/~jos/filters/.

Daniel Stenberg and curl contributors. libcurl — the multiprotocol file transfer library. https://curl.se/libcurl/, 2026.

W3C Provenance Working Group. PROV-O: The PROV ontology. W3C Recommendation, 2013. URL https://www.w3.org/TR/prov-o/.

Mark D. Wilkinson et al. The fair guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016. doi: 10.1038/sdata.2016.18. URL https://doi.org/10.1038/sdata.2016.18.

Greg Wilson et al. Best practices for scientific computing. *PLoS Biology*, 12(1):e1001745, 2014. doi: 10.1371/journal.pbio.1001745. URL https://doi.org/10.1371/journal.pbio.1001745.