The N-Surface Method: Rigor and Spontaneity in Human–AI Research:

Luiz Velho
with Claude (Anthropic) and GPT (OpenAI)
VISGRAF Lab, IMPA

Technical Note · draft 8 October 2026 · revised 5 November 2026

Abstract

Between August and October 2026, one research line on active foveal 3D vision at VISGRAF/IMPA ran through six successive repositories. Each had its own protocol for collaboration between one human researcher and AI systems. The configuration grew from a single conversational assistant to combinations of surfaces from two vendors (ChatGPT, Claude Chat, Claude Code), spread over two machines and several interfaces, with GitHub as the shared record.

This note reconstructs that evolution and extracts a method from it. The protocols changed a great deal; a small kernel did not. The repository, not the conversation, is the memory. History moves forward only. The human decides.

Around the kernel sits an adaptive shell: the number and roles of surfaces, the transport between them, commit authority, ceremony per kind of step, cost classes and loop time scales. Its settings were re-derived each time platforms, capabilities or the project stage changed. We describe surfaces by capability rather than by model and show where the balance between rigor and spontaneity was struck.

We call the result an **N-Surface method**. N is not a target; it is whatever the current context requires.

1 The problem

AI collaboration makes implementation, experiments and prose cheap; verification and scientific judgment stay scarce. In software the characteristic failure is a bug. In research it is a wrong belief: a plausible claim, backed by a computation that may be correct, read as evidence for something it did not establish. AI collaboration can produce internally consistent artifacts faster than one person can check what they warrant.

A second problem appears once more than one AI system is involved: **the collaboration itself moves**. Surfaces differ in what they can see, change and run. Platforms change those capabilities without notice. The project also passes through stages (exploration, science, refactoring, presentation) that need different amounts of discipline. A fixed protocol over-constrains some stages and under-constrains others.

The method therefore has to strike two balances:

- **Rigor versus spontaneity.** How much ceremony a given step deserves, so that errors surface without suppressing cheap exploration.
- **Stability versus adaptivity.** Which parts of the protocol may change with the context, and which must not.

Terms

- **Surface.** An AI system reached through a specific interface, with specific access. Claude Chat in a browser and Claude Code in VS Code on a GPU workstation are different surfaces, even when the underlying model is similar.
- **Station.** A role in the working loop: deciding, specifying, executing, reviewing, recording. A surface or the human may occupy one or several. Station counts below follow each protocol's own naming.
- **Protocol carrier.** The file that states the rules for a repository: `README.md`, `CLAUDE.md`, `AGENTS.md`.

Scope

One researcher, one research line, about two months, six repositories. The sample size is one. This is not a validated general method but an account specific enough to be checked. All six repositories are public (Appendix A).

2 The journey

Six repositories in about two months. The protocols that named their stations counted four (bio-3d-vision), then five (fov-3d-vision), then three (f3d-vision): the method matured by compressing, not by accumulating.

Stage 1 · bioeye: one entity

ChatGPT alone, through conversation and git, built a thin vertical slice of a six-layer active Bayesian stereo framework. The only protocol was the README, which is a data contract (files, units, run commands). It reached a running demo and slides quickly. It closed the accumulation loop, and knowledge lived in the conversation. The human was author, executor and courier.

Stage 2 · active-stereo: the constitution

Thinking and execution were split between Claude Chat and Claude Code. `CLAUDE.md` became a normative constitution: layers as module boundaries, three structural closures, invariants on units, frames, invalid data, determinism and uncertainty, and a list of prohibitions. Code never committed without being asked. Two ideas from this stage lasted:

Stage	Repository	Start	Surfaces	Protocol carrier	Human role	Default stance
1	bioeye	9 Aug	ChatGPT	README (a data contract)	Author, executor	Implicit
2	active-stereo	15 Aug	Claude Chat + Code	CLAUDE.md "constitution"	Approves every commit	Prevention
3	bio-3d-vision	29 Aug	Claude Chat + Code (+ Cowork)	CLAUDE.md, workflow, state file, ledgers	Gate and relay	Epistemic rigor
4	w3d-scenes	12 Sep	Claude Chat, workstation check	README	Executor of checks	Controls first
5	fov-3d-vision	12 Sep	Claude Chat + Code; ChatGPT at phase ends	CLAUDE.md working agreement	Courier (zip + sha256)	Cheap recovery
6	f3d-vision	25 Sep	ChatGPT + Claude Code; from 11 Oct Claude Chat + Code, ChatGPT at phase ends	CLAUDE.md + AGENTS.md; from 12 Oct kernel, shell and handoff	Decides; carries one prompt	Science-led, engineering-disciplined

Table 1: Six stages, six repositories.

- **The handoff contract.** A number arriving from Chat is a hypothesis with a pointer; the receiver holds the files and verifies. The reverse direction is deliberately asymmetric: Code discloses which inputs were measured or assumed; Chat may not use a Code number to settle a dispute without asking what it was measured against.
- **Memory is a cache, never the record.** A lesson kept only in an assistant's memory does not exist for the project.

Stage 3 · bio-3d-vision: the research method

A four-station loop: the human sets the goal and gates, Chat specifies, the human relays, Code executes, and a separate Chat checkout verifies. Decision records are written after the thing is tried. Units and frames live in docstrings, beside the code they describe. Inherited numbers carry provenance. A `spikes/` lane allows rough tries without ceremony. A task is done only if it moved something.

Stage 4 · w3d-scenes: a convenience step

Part convenience, part data migration: gathering test scenes with 360° previews and checks. Claude Chat built it in a CPU sandbox and the workstation verified it on the GPU. Its README already separates *verified on CPU*, *verified on GPU* and *not yet tested*, and ships positive and negative controls.

Stage 5 · fov-3d-vision: engineering first

The stance reversed: "A visual result at every step. Mistakes are expected," so the process optimises for cheap recovery, not prevention. Chat wrote code and docs, ran what its sandbox allowed (self-tests, negatives, a stub Blender) and handed over a prompt for Code. The prompt listed commands with cost classes, expected checks, likely failures, what must not be changed and what to report. Luiz applied and committed; Code ran, diagnosed, pushed to `main` and reported. No PR gate; revert if wrong.

A second loop ran at phase boundaries: Claude Chat summarised the phase, ChatGPT reviewed it, and the review went back to Chat (Section 5).

Stage 6 · f3d-vision: GitHub carries state

ChatGPT on a Mac Studio, Claude Code in VS Code on an Ubuntu RTX 4090 workstation, GitHub between them. In the working agreement, Claude Chat is the design and review surface and pushes contracts, documentation and host-side code to the work branch. Claude Code keeps execution and measurement. ChatGPT returned as an optional reviewer at phase ends.

New elements: three claim states (proposed, measured, assumed); "migrate behavior first; redesign structure second"; a Chat Handoff file at accepted milestones; separate `previews/` and `visuals/` trees. The tagline moved from "engineering-first / science-second" to "science-led, engineering-disciplined". The stage produced the research summarised in Section 6 and a technical report, *Active Foveal 3-D Vision Playground* (TR-11-2026) [2].

3 Surfaces as capability profiles

A surface is defined by what it can see, change, run and remember; which model sits behind it matters less. Roles in the protocol follow from these capabilities, and when a capability changes, the roles have to be re-derived.

Four principles follow from the table.

Authority follows access. The surface that holds the files and the hardware overrules the surface that holds the memory. Only execution on the workstation turns a proposed number into a measured one. GPU, Blender and performance claims made anywhere else stay proposed.

Same model, different surface, different station. Claude Chat and Claude Code differ in access, not intelligence, and that difference decides who may claim what. Surfaces from different vendors differ in their priors as well, which makes them useful reviewers of each other.

The human is a router, and transport sets the routing load. Copy-paste, a relayed specification and a git commit cost the human very different amounts of work. Each change of transport changed how much of the human's time went into carrying files instead of deciding.

Participant	Sees	Changes	Runs	Remembers	Role in stage 6
Luiz (human)	Everything, at limited bandwidth	Anything	Anything	Across all repos and stages	Sets direction, decides, routes
ChatGPT (Mac, app or browser)	The live repo, read-only	Nothing in the repo	A sandbox: no GPU, no real assets	The conversation	Designs, specifies, reviews, writes the Code prompt; an optional reviewer at phase ends
Claude Chat (browser or app)	Files and excerpts passed to it; public repos; a clone of the live repo	Work branches Luiz names	A CPU sandbox, sometimes a stub Blender	The conversation; a claude.ai project memory	Plans, specifies and reviews
Claude Code (VS Code, Ubuntu RTX 4090)	The working tree	Branches, worktrees, commits	GPU, Blender, full assets	What the repo holds; on the workstation, an auto-memory (Section 8)	Executes, measures, mutates the repository
GitHub repository	—	—	—	Code, contracts, reports, history	The durable record and the handoff medium

Table 2: Surfaces described by capability.

Platform policy is part of the topology. When ChatGPT lost write access, the protocol consolidated all repository mutation in Claude Code and reduced the human's carrying to one prompt. Making the repositories public was the same kind of decision: it let surfaces from both vendors read the same state despite access restrictions.

4 The invariant kernel

Five rules survived every protocol change from stage 2 onward, reworded but not weakened; a sixth arrived late and has held since. Together they make a claim checkable regardless of which surfaces produced it. Each rule is listed with the failure that motivates it, because a rule without its reason is easily simplified away as bureaucracy.

Rule	First form	Final form	Failure it prevents
Every claim is proposed, measured or assumed	Chat numbers are hypotheses with pointers (stage 2)	Three named states; only workstation execution makes a number measured	A fluent figure from a summary becomes a fact
Every tool ships a check that can fail	"Carry only what can fail" (stage 3)	A control, known answer, invariant or behavioral comparator per tool	A green test that could never have come out red
The repository is the memory	"Memory is a cache, never the record" (stage 2)	Contracts, reports and a Chat Handoff index in the repo	Knowledge lost when a conversation ends
History moves forward only	No force-push; append-only decisions (stage 2)	Commit, push, revert if wrong	A correction that erases how the error escaped
The human decides	Ask when the spec is ambiguous (stage 2)	Explicit authority; stop conditions return decisions to Luiz	An AI resolving ambiguity by inference
The contract precedes execution	Pre-registered experiments (stage 3)	The contract is committed before implementing: by Claude Chat	Tuning against the result; prevented errors leaving no trace

Table 3: The invariant kernel.

The kernel is deliberately small. It says nothing about which surfaces exist, how files travel, or how much ceremony a step needs. Those belong to the shell (Section 5).

The kernel is also what makes spontaneity safe. "Mistakes are expected" works only when mistakes become visible; claim states and fail-capable checks are what make them visible. Without the kernel, cheap recovery is just speed.

5 The adaptive shell: dials and triggers

Everything outside the kernel was adjusted at least once. Treating these as dials, rather than as fixed rules, is what makes the method adaptive: changing one setting should not require rewriting the constitution.

Dial	Settings observed, in order	What moved it
Surfaces and roles	One assistant → Chat + Code → Chat specify / Code execute / Chat verify → plus a cross-vendor phase reviewer → ChatGPT design + Code execution → Claude Chat design + Code execution, ChatGPT optional phase reviewer	Need to separate thinking from running; need for an independent reviewer; platform access
Transport	Copy-paste → relayed, uncommitted specs → zip + sha256 → git commits by both AIs → git, read-only Chat, one paste-ready prompt → git, Claude Chat pushing to work branches	Courier load on the human; write restrictions
Commit authority	Human only → never without asking → Code pushes to `main` → Code on the named branch, in an isolated worktree → Claude Chat on work branches too; main moved by either, on Luiz's acceptance	Speed versus safety of the shared checkout
Ceremony per step	Uniformly heavy → uniformly light → by step type	Exploration suppressed (stage 3); refactoring needs proof of preserved behavior (stage 6)
Evidence per step type	Scientific: measurable result, visual, checks, report · Design: reviewed specification · Refactoring: comparator with `MISMATCHES 0`	Project stage
Cost classes	Absent → interactive (<10 s), batch (<5 min), overnight (justified in advance); small and full profiles	Expensive renders stalling the loop
Where conventions live	Constitution → docstrings → constitution again → kernel and shell (Section 8)	Cross-vendor surfaces needing one readable place (stages 5–6)
Loop time scales	One loop → steps plus phases → steps reviewed per commit → steps plus optional phases again	Need for review at a different distance from the work
Visual record	Demo figures → each iteration's artifact beside the last → a visual every step → `previews/` (machine, per stage) and `visuals/` (human, persistent)	Need to compare by looking; disk and lifetime of evidence

Table 4: The dials of the adaptive shell.

Triggers

Five kinds of event moved the dials:

1. **A platform capability changed**: write access removed, a connector added, repositories made public.
2. **A failure was observed**: a false constitution clause, an unused spikes lane, an uncommitted specification.
3. **The project changed stage**: exploration, science, migration and refactoring, presentation.
4. **Execution cost changed**: a 35-minute reference render cannot sit inside an interactive loop.
5. **The human's bandwidth ran out**: carrying zips between machines is time not spent deciding.

Two time scales

The most consequential shell setting was never written down. In fov-3d-vision two loops ran on different time scales. The inner **steps** loop went from Chat's plan through the human to Code's execution and back through the repository. The outer **phases** loop took Chat's end-of-phase summary to ChatGPT for review and suggestions, and returned the outcome to Chat for reassessment.

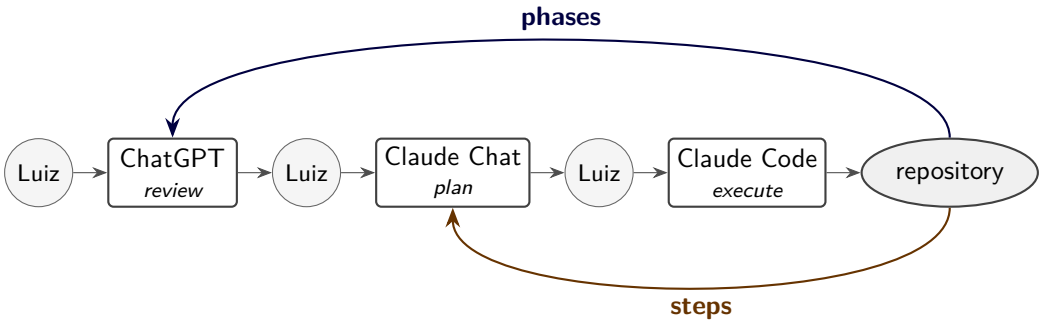


Figure 1: fov-3d-vision working loops, redrawn from the hand sketch. The inner steps loop returns to Claude Chat through the repository; the outer phases loop returns to ChatGPT for review.

Chat is Claude Chat and Code is Claude Code. The phases loop with ChatGPT ran in practice but was never written into the protocol.

The outer loop placed a reviewer from another vendor at stage boundaries, which is where unexamined momentum is most expensive. A reviewer that did not write the plan does not share the planner's momentum. In f3d-vision the reviewer first moved inward: ChatGPT reviewed every committed result, so phase-level and step-level review collapsed into one loop. Later the two loops came back: Claude Chat reviews steps, ChatGPT optionally reviews phases, and both are now written into the protocol (Section 8).

6 Rigor versus spontaneity in practice

Neither uniform rigor nor uniform informality worked. The balance came from specific, bounded mechanisms that let the kernel stay strict while individual steps stayed cheap.

Mechanisms that combine the two

- **The bounded bet.** A line that keeps failing gets one more attempt, framed in advance: a frozen configuration, no tuning, and an action fixed for each outcome. The framing is strict; the attempt is cheap.
- **Negative results count.** A negative result with a stated mechanism is accepted like a positive one. That removes the pressure to tune until something passes.
- **A cheap control before a redesign.** Before redesigning a component that failed, change the one input that might explain the failure. One variable, one run, one answer.

- **A simple baseline first.** A deliberately simple version runs before the elaborate one, so later work has a measured result to beat.
- **Mandate errors are named.** Separating an error of design from an error of implementation keeps faithful work from being blamed and wrong work from being accepted.
- **Cost classes.** Development happens on the small profile, in seconds; the full profile runs once, for the number that gets reported. Spontaneity is cheap where iterations are cheap.

The working principle

Scale ceremony to the cost of being wrong and to how hard the error would be to undo. Where recovery is cheap and the kernel makes errors visible, try first. Where a result will become a premise for later work, declare the criteria first.

7 Toward seamless adaptation

So far every adaptation happened by rewriting the protocol in a new repository or a new version of `CLAUDE.md`. Seamless adaptation would mean changing a dial without touching the kernel, and leaving a record of why. Four proposals follow. The first three were tried once, in f3d-vision (Section 8); the fourth has not been tested.

1. **Three layers, three files.** Keep the kernel in one short file that rarely changes. Keep the shell's current dial settings in a second file, each with the trigger that set it. Keep project state in the Chat Handoff, never in the constitution. A capability change then edits one line in the shell file.
2. **A coherence lint for the protocol.** A small lint over the governance files for clarity and coherence, not content: statements about write access agree with the transport described, contract paths match the repository layout, the named branch exists, terms are used consistently. Whether the protocol serves its purpose stays a human judgment. This applies the kernel's second rule to the syntax of the method, not its semantics.
3. **Commit the process, not only its products.** Record phase reviews, rejected contracts and prevented errors as small committed files. The cheapest saves are currently the least visible.
4. **State the kernel once for every agent.** `AGENTS.md` already points all coding agents to one canonical file. Testing a further surface (Codex is planned) will show whether the kernel is truly vendor-neutral or quietly assumes Claude Code.

Open questions

- Do the observed failures belong to the method or to this researcher? Only a second person running it can tell.
- What is the smallest N that serves a given stage? Exploration may need one surface; validated science, at least an executor and an independent reviewer.
- Can cross-vendor review replace some human review at phase boundaries, or does it only move momentum from one model to another?
- How should a project record work it correctly prevented?
- Does the method transfer to research that is qualitative or construction-driven, where falsifiers are harder to state?

8 A first trial: Governance-1

Proposals 1–3 were tried once, in f3d-vision, as one governance step. Claude Chat wrote the contract and the implementation, Claude Code executed it on the workstation, and Luiz accepted it; the commits span 16:13 to 16:51. The configuration was the one proven in fov-3d-vision: `[ChatGPT >] Luiz > ClaudeChat > Luiz > Claude Code`. Figures are measured, from the step's committed report; line counts are at commit `11bfbf6`.

What was built

Layer	File	Holds	Size
Kernel	`CLAUDE.md`	authority, the six kernel rules, contracts, reports, repairs, project invariants	1,023 → 111 lines
Shell	`docs/process/shell.md`	surfaces and their write access, who commits what, 16 dials each with its trigger	104 lines, new
State	`docs/chat-handoff.md`	where the project is now, rewritten at each milestone	2,557 → 98 lines
Lint	`tools/process/lint.py`	checks of form, not content, with a self-test of 22 mutations	273 lines, new
Records	`docs/process/records/`	one small file per review, rejected contract or prevented error	4 records, new

Table 5: What Governance-1 built.

The kernel imports the shell and the handoff, so Claude Code loads all three at startup; other surfaces read them in that order. Reading at session start fell from about 3,600 lines to about 310. The replaced files survive byte-identical in a history folder.

What it cost

One shell file, one lint script, one folder. In the loop: a one-second lint before any push that changes a governance file, a small file when something is caught, and a handoff rewritten instead of appended. The two dials Luiz added at acceptance, a fast-forward of the shared checkout after each acceptance and assistant memories kept as caches, never status, edited the shell and left the kernel as it was. That is the seamless adaptation of Section 7, on its first two occasions.

References

[1] Luiz Velho and Claude. "Experiments on Active 3D Vision". Technical Report TR-09-2026, VISGRAF Lab – IMPA, 2026.

[2] Luiz Velho and Claude. "Active Foveal 3-D Vision Playground". Technical Report TR-11-2026, VISGRAF Lab – IMPA, 2026.

A Repositories

All six are public. Each claim in this note about a stage can be checked against its repository.

Repository	Start	Surfaces	Scientific scope	Output
visgraf/bioeye	9 Aug 2026	ChatGPT	Euclidean framework, layers 1–5	Demo and slides
visgraf/active-stereo	15 Aug 2026	Claude Chat + Code	Spherical, layers 1–3, Bayesian	11 experiments
visgraf/bio-3d-vision	29 Aug 2026	Claude Chat + Code	Spherical, layers 1–5, closing the loop	13 experiments; TR-09-2026 [1]
visgraf/w3d-scenes	12 Sep 2026	Claude Chat; workstation verification	Test scenes	Archived; superseded by step A1 of fov-3d-vision
visgraf/fov-3d-vision	12 Sep 2026	Claude Chat + Code; ChatGPT phase review	Foveated omnidirectional rendering and 3D reconstruction	Phase reports
visgraf/f3d-vision	25 Sep 2026	ChatGPT + Claude Code; from 11 Oct Claude Chat + Code, ChatGPT phase review	Foveated omnidirectional segmentation and reconstruction	Greedy Foveal Playground baseline; TR-11-2026 [2]; Governance-1 (Section 8)

Table 6: The six repositories.