SEW: Symmetric Embedding Workbench Whitepaper

A Manifold Simulator Kernel from First Principles

brackishbert@gmail.com · russell@unturf.com · cuppcb.com

2026-03-24

SEW: Symmetric Embedding Workbench

A Manifold Simulator Kernel from First Principles

SPARKLE_ID: 1.0.2.10 **Revision:** 0.1 **Status:** WORKING DRAFT

Abstract

SEW (Symmetric Embedding Workbench) is a browser-resident manifold simulator built on three primitives: a symmetric product space construction, a heat diffusion signal model, and a frozen kernel with an open program injection interface. This paper derives each layer from first principles, establishes the kernel stability invariant, and specifies the program injection protocol. Dot diagrams formalize the data flow and architectural boundaries.

1. Motivation

Most interactive geometry tools begin with a fixed mesh and let the user deform it. SEW inverts this. The user draws arbitrary points in a plane. The workbench constructs a canonical higher-dimensional manifold from those points using the symmetric product construction. Programs then run on that manifold — injecting signals, reading topology, reshaping the surface — without knowing how it was built.

The result is a general-purpose signal substrate. Any program that can write to a heat buffer indexed by vertex position can participate in the manifold's dynamics. The geometry becomes a shared memory between concurrent programs and the user.

2. First Principles

2.1 POINTS AND PAIRS

Begin with a finite set X of points in the plane, drawn by the user. X has no canonical ordering. The workbench must construct something meaningful from X without privileging any particular ordering of its elements.

The natural object is the set of unordered pairs:

```
Sym²(X) = { {p, q} : p, q ∈ X }
```

This is the symmetric product of X with itself. It has $|X|^2$ elements if we allow p = q (the diagonal), or $|X|(|X|-1)/2 + |X|$ elements counting the diagonal once.

The diagonal $\Delta \subset \mathrm{Sym}^2(X)$ is the locus where $p = q$:

```
Δ = { {p, p} : p ∈ X }
```

Δ is geometrically degenerate — both points in the pair coincide. It is rendered as the red seam line in the viewport. Everything off the diagonal is a genuine two-point configuration.

2.2 THE EMBEDDING

To embed $\mathrm{Sym}^2(X)$ into R^3 we need three scalar functions of a pair {p, q}:

```
x({p,q}) = (px + qx) / 2       — midpoint x
y({p,q}) = (py + qy) / 2       — midpoint y
z({p,q}) = |p - q|              — separation distance
```

The midpoint coordinates place the pair in the plane of X. The separation distance lifts it above that plane proportionally to how far apart the two points are. Diagonal points ($p = q$) land at $z = 0$, the base plane. Off-diagonal pairs rise above it.

This embedding is symmetric by construction: swapping p and q produces the same point in R^3. It is also continuous: nearby pairs in X map to nearby points in the embedding.

2.3 DISCRETIZATION

With a sampled set of m points from X ($m = 32$ in the current implementation), the embedding produces an $m \times m$ grid of vertices. Vertex (u, v) encodes the pair {X[u], X[v]}.

Triangulation: connect (u,v)→(u+1,v)→(u,v+1) and (u+1,v)→(u+1,v+1)→(u,v+1) for all u, v in [0, m-1). This produces $2(m-1)^2$ triangles.

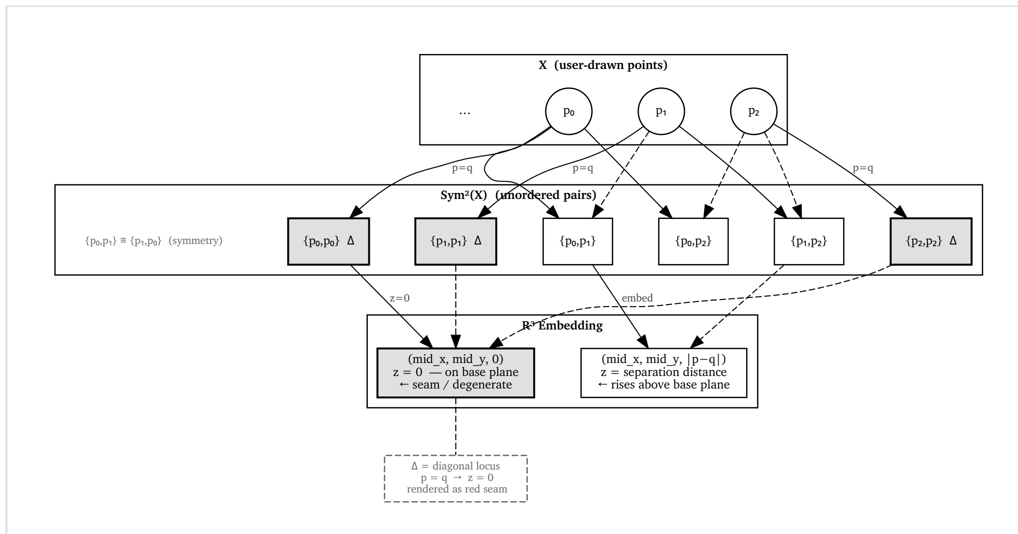
The resulting mesh is the concrete manifold SEW operates on.

2.4 THE ADJACENCY GRAPH

From the triangle index buffer, derive a vertex adjacency graph $G = (V, E)$:

```
V = { (u,v) : u,v ∈ [0,m) }      |V| = m²
E = { {a,b} : a,b share a triangle edge }
```

Each vertex has degree ≤ 6 (four-connected grid, triangulated). Deduplicate with Set to avoid double-counting. G is the substrate for heat diffusion and agent traversal.



Sym²(X) Construction

3. The Heat Model

3.1 HEAT AS SIGNAL

Heat is a scalar field h: V → R≥0 defined on the manifold vertices. It serves as the universal communication channel between programs and the rendering engine.

The renderer reads heat each frame and adds h[i] × w to the z-coordinate of vertex i, where w is a time-varying oracle expression. High heat lifts a vertex; zero heat leaves it at rest.

3.2 DIFFUSION

Each frame, heat diffuses across the adjacency graph:

```
h'[i] = ( h[i] + Σ_{j ∈ N(i)} h[j] ) / ( |N(i)| + 1 ) × decay
```

where N(i) is the neighbor set of vertex i and decay = 0.96. This is a single step of the graph Laplacian heat equation. It smooths local concentrations, propagates signals spatially, and ensures injected heat eventually dissipates without continuous input.

The discrete Laplacian here is the normalized graph Laplacian:

```
L_norm[i] = h[i] - ( Σ_{j ∈ N(i)} h[j] ) / |N(i)|
```

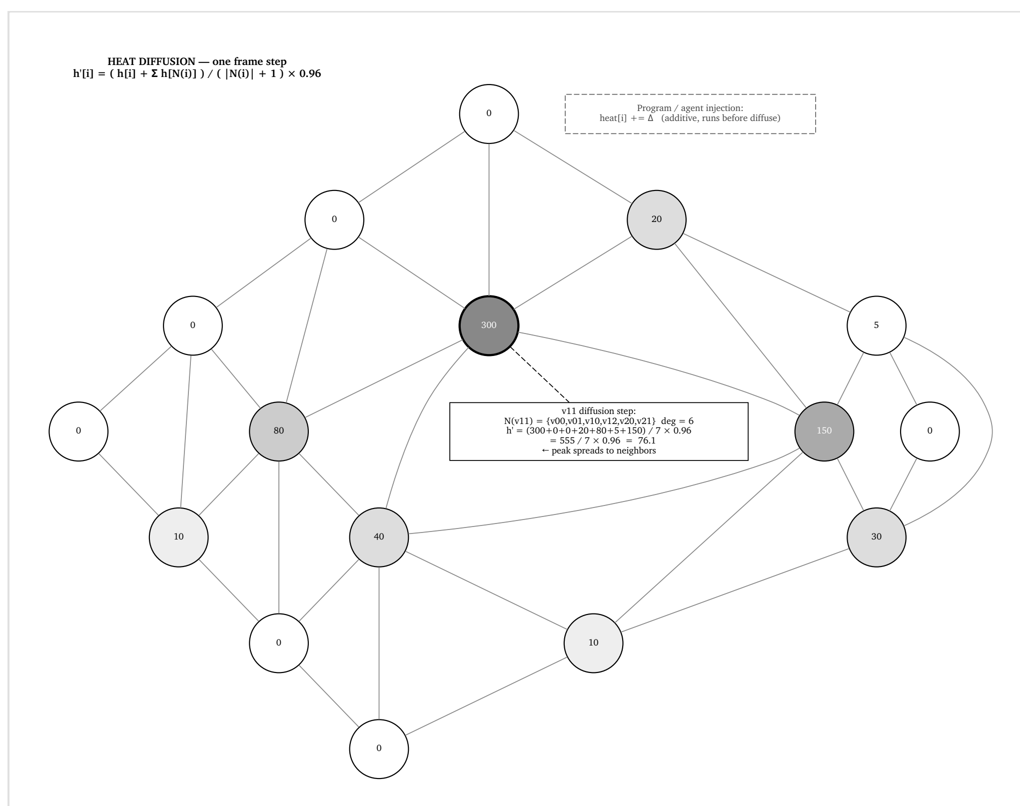
Diffusion is equivalent to h′ = h - α × L_norm × h for small α.

3.3 INJECTION

Programs inject heat by writing directly to the heat buffer:

```
heat[i] += amount;                          // additive — stacks with other programs
heat[i] = value;                            // absolute — overrides
heat[i] = Math.min(heat[i] + amount, cap); // capped additive
```

Additive injection is preferred. It composes correctly when multiple programs run simultaneously — each program's signal adds to the manifold without needing to coordinate with others.



Heat Diffusion on Adjacency Graph

4. Agent Traversal

Agents (Friends) are particles that walk the adjacency graph G using a heat-seeking heuristic:

```
at each step:
  with probability 0.15: move to a random neighbor   (exploration)
  with probability 0.85: move to the hottest neighbor (exploitation)
  inject heat[current] += 8.0
```

This is a stochastic gradient ascent on the heat field with exploration noise. Agents cluster at heat maxima, reinforce them by injecting more heat, and occasionally escape to explore cold regions. The system self-organizes: wherever a program concentrates heat, agents converge and amplify.

5. The Kernel Architecture

5.1 KERNEL STABILITY INVARIANT

The kernel (kernel.html) is a frozen artifact. Its md5 hash is recorded at deployment and must not change between deployments. No program, script, or operator may modify it.

Invariant: md5(kernel.html) is constant for the lifetime of a kernel version.

```
md5(kernel.html)    = c78e69389acd2a2683e5e6c8e065086d
sha256(kernel.html) = c3cbaaf030101858e48632f239e15f0b8a02838c4c85b6b437573375539aa28c
```

When the kernel must change (new primitive, breaking fix), the version number increments and a new kernel.html is issued. Old programs remain compatible or are explicitly ported.

This invariant enables:

- Programs to assume a stable global scope
- Operators to audit exactly what the kernel does
- The workbench to be reproduced from kernel.html alone on any machine

5.2 KERNEL GLOBALS

Programs have read/write access to the following kernel globals:

SYMBOL	TYPE	DESCRIPTION
`scene`	THREE.Scene	Three.js scene graph
`camera`	THREE.Camera	Perspective camera
`heat`	Float32Array	Heat buffer, length = m^2 (1024)
`adjacency`	Array<Array>	Vertex neighbor lists
`meshVertices`	Array<number>	Current vertex positions (flat xyz)
`originalVertices`	Array<number>	Rest-state vertex positions
`seedPoints`	Array<{x,y}>	User-drawn input points
`friendList`	Array<Friend>	Active agent instances
`pcb`	PCBInterpreter	Console — pcb.log() / pcb.run()
`flickerRate`	number	Material opacity oscillation rate
`wobbleBase`	number	Per-vertex position noise amplitude
`tick`	number	Frame counter, increments each RAF
`buildManifold`	function	Rebuild mesh from seedPoints array

5.3 THE PCB CONSOLE

The POCKET_CUP_BOX_CONSOLE is a JavaScript REPL wired to the kernel scope. It detects code (patterns containing `= . (`) and evals directly. Plain text with FRONTIER open routes to the Anthropic API. Programs extend it by wrapping `pcb.run`:

```
const _orig = pcb.run.bind(pcb);
pcb.run = function(src) {
    if (src.trim() === 'my.cmd') { /* handle */ return; }
    _orig(src);   // pass through to previous handler
};
```

This forms a chain of responsibility. Programs wrap in load order. Each handler either consumes the command or delegates. The kernel's base handler is always at the bottom.

System Architecture

---

6. The Program Injection Model

6.1 INJECTION PROTOCOL

Programs are plain JavaScript files loaded as `<script src>` tags in `index.html` after the kernel script. They execute synchronously in document order, in the same window scope as the kernel. No module system, no sandbox, no message passing.

```
<!-- PROGRAM LOADER -->
<script src="programs/bubble-sort.js"></script>
<script src="programs/reaction-diffusion.js"></script>
<!-- add programs here -->
```

6.2 PROGRAM INTERFACE

A conforming program must:

1. Execute inside an IIFE `(function(){ ... })()` to avoid polluting global scope
2. Insert exactly one `.module` div into `#sidebar` via `insertBefore(mod, firstChild)`
3. Wrap `pcb.run` if registering PCB commands, always calling `_orig` for unknown input
4. Call `pcb.log('PGM: NAME loaded. cmds: ...')` on successful initialization
5. Clean up timers and animation loops when stopped (no leak on stop/reset)

6.3 SIDEBAR MODULE TEMPLATE

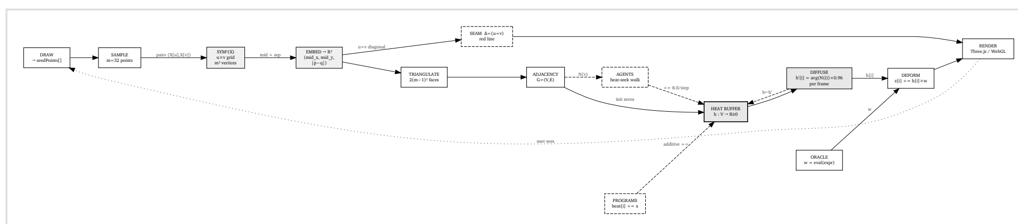
```
        const mod = document.createElement('div');
        mod.className = 'module';
        mod.id = 'pgm-myprogram';
        mod.innerHTML = '<h3>MY_PROGRAM</h3><!-- controls -->';
        document.getElementById('sidebar')
            .insertBefore(mod, document.getElementById('sidebar').firstChild);
```

Kernel CSS classes available: `.module`, `.btn`, `.btn-pcb`, `.matrix-grid`, `.m-val`.

6.4 HEAT CONTRACT

Programs that write to `heat` must:

- Check `heat.length > 0` before writing (manifold may not be built yet)
- Prefer additive writes (`heat[i] += x`) over absolute writes
- Respect `heat.length === N` where N = 1024 for the current m = 32 kernel
- Not hold references to old heat arrays — diffusion may replace the buffer each frame



Signal Dataflow

---

7. The Boot Sequence

7.1 SPARKLE — LINEAGE OF THE WORKBENCH

Sparkle names the workbench. `brackishbert@gmail.com` first wrote Sparkle in Java, where the symmetric-product construction and heat model found their first realization. The browser port via Three.js kept the name and the semantics while trading JVM latency for zero-install reach. `russell@unturf.com` maintains the six.js patches — a patched Three.js with CWE-407 sedimentary defects lifted out of the graph primitives — which now drives the browser runtime. Both Sparkle-in-browser and the patched six.js runtime publish from `cuppcb.com` as a CDN, so any visitor loads the canonical kernel and its runtime from one origin.

7.2 BOOT READOUT

On page load, `index.html` runs a timed boot readout before revealing the workbench. The sequence verifies the visual expectation that loading stays intentional and measurable, not accidental latency.

```
SPARKLE_BIOS v1.0.2.10
KERNEL HASH .............. VERIFIED
THREE.JS r128 ............ OK
ORBIT_CONTROLS ........... OK
MANIFOLD ENGINE .......... READY
PGM BUBBLE_SORT .......... OK
PGM REACTION_DIFFUSION ... OK
PGM LORENZ_ATTRACTOR ..... OK
PGM CONWAY_LIFE .......... OK
PGM WAVE_INTERFERENCE .... OK
PGM FIRE_SIM ............. OK
PGM LISSAJOUS ............ OK
PGM CHAOS_GAME ........... OK
PGM GRAVITY .............. OK
PGM AUDIO_REACTIVE ....... OK
PGM FLOW_FIELD ........... OK
PGM LANGTON_ANT .......... OK
ALL SYSTEMS .............. NOMINAL
SPARKLE READY _
```

The gold progress bar tracks cumulative program count. On completion the overlay dissolves and the workbench is revealed.

8. Installed Programs

PROGRAM	PCB COMMANDS	DESCRIPTION
`bubble-sort`	`sort.run` `sort.heat` `sort.reset`	Sorts random array; `sort.heat` runs on manifold
`reaction-diffusion`	`rd.spots` `rd.stripes` `rd.mitosis`	Gray-Scott model — organic pattern emergence
`lorenz`	`lorenz.start` `lorenz.chaos`	Lorenz attractor traces heat into nearest vertex
`life`	`life.start` `life.glider` `life.gun`	Conway's Life — alive cells inject heat
`wave`	`wave.start` `wave.chaos`	Interference moiré from multiple point sources
`fire`	`fire.start` `fire.inferno`	Fire simulation — manifold becomes flame
`lissajous`	`liss.go` `liss.spin` `liss.flower`	Parametric curves auto-reify manifold geometry
`chaos`	`chaos.sierpinski` `chaos.fern`	IFS chaos game — fractals grow into manifold
`gravity`	`gravity.start` `gravity.collapse`	N-body orbits — potential wells dimple the mesh
`audio`	`audio.mic` `audio.synth`	FFT → heat, manifold dances to sound
`flow`	`flow.start` `flow.storm`	Noise flow field, particle streams leave trails
`langton`	`ant.start` `ant.turbo` `ant.ants 4`	Langton's ant — emergent highways after ~10k steps
`moad-demo`	`moad.run` `moad.pause` `moad.reset`	CWE-407 live: POCKET O(n²) vs KNOT O(1) — shape is the proof

8.1 MOAD: The CWE-407 Companion

The `moad-demo` program is a SEW-resident benchmark of **CWE-407** — the sedimentary defect in which a list is used where a set belongs inside graph traversal code.

Two implementations run in the same animation frame:

- **POCKET** — the unpatched path: `Array.includes(n)` before each insertion. $O(n)$ per call, $O(n^2)$ total. This is the pattern confirmed in 50+ sites across 25+ ecosystems: javac, GHC, TypeScript, FRRouting, PostgreSQL, MongoDB, Erlang, pip, Cargo, and others.
- **KNOT** — the patched path: `Set.has(n)` before each insertion. $O(1)$ per call, $O(n)$ total. The fix is structurally identical across all affected ecosystems: replace the list with a hash-backed set.

A grouped bar chart renders cost per step. POCKET's bars climb quadratically; KNOT's stay flat. At n=2000 the ratio is typically 100×–200× in favor of KNOT. The manifold receives a heat spike proportional to POCKET's defect cost each step — making the $O(n^2)$ growth visible as geometry.

This program is the bridge between the SEW manifold and the CWE-407 research corpus (`undefect.` / java-topology). The same defect class, visualized here as a heat signal on a symmetric product manifold built from the user's own drawn points.

8.2 Manifold Oracle — Machine Learning Application

$Sym^2(X)$ can serve as more than a visualization substrate. In Cake Murder Adventure (a game cartridge built on the SEW kernel), the manifold functions as an **oracle** — a deterministic, geometry-driven source

of adversarial events.

Each chapter seeds the manifold with a unique integer. The walker traverses that manifold; its angular position maps to a screen coordinate; a knife spawns there. The geometry of each chapter's $Sym^2(X)$ surface determines where knives come from, at what rate, and with what spread. The manifold is not decorative — it is the game's threat engine.

THE LEARNING PROBLEM

A reactive controller navigates the dome. At each frame it computes a move direction from a sum of force vectors:

```
dx = Σ wᵢ · fᵢ(state)
```

where each f_i is a fixed geometric function (flee from dropping blade, repel from wall, attract to exit, anticipate next spawn) and w_i is a scalar weight. The algorithm is fixed. Only the 9 weights are free.

The 9 weights — called **params** — are tuned by CEM (Cross-Entropy Method): sample candidate weight vectors from a Gaussian, evaluate each against all 101 chapters in sequence, keep the best, tighten the distribution. Repeat until a weight vector clears all 101 chapters.

RESULTS

	MEAN CHAPTERS	MAX	MIN	N
random params	69.2	96	38	30
champion params	**101.0**	101	101	10

The champion weight vector clears 101/101 reliably. Random weight vectors average 69 — the force-field architecture is forgiving, but never complete.

The champion is calm. The weights that failed at chapter 96 had high `stuckBoost` (7.5) and `zAttract` (11.6) — reactive and greedy. The champion has `stuckBoost` 1.2, `zAttract` 1.8, and `zAnticipate` 4.1 — it reads ahead rather than reacts.

OUT-OF-SAMPLE GENERALIZATION

Training evaluated only chapters 1–101. The champion was then tested on chapters 102–121, 200, 500, and 999 — manifold seeds never seen during training.

Result: 23/23 cleared.

The controller did not memorize 101 chapters. It learned to navigate $Sym^2(X)$ geometry. The params generalize across any manifold seed in this family. The manifold is a sufficient test surface: agents tuned on it transfer to unseen instances without retraining.

SIGNIFICANCE

This establishes a new role for $Sym^2(X)$: a **structured test environment for machine learning**. Unlike random environments, the manifold generates adversarial events with mathematical regularity — each chapter is geometrically distinct but drawn from the same topological family. An agent that generalizes across chapters has learned something about the family, not the instances.

The same pattern — fix an algorithm, tune scalar weights against a manifold oracle, verify out-of-sample — could apply to robotics obstacle avoidance, traffic navigation, or any domain where a non-Euclidean threat geometry can be specified mathematically.

Full lineage, parameter tables, and benchmark data: `docs/cake-murder-genomes.md`.

---

9. Extension Points

The current kernel exposes extension through heat and `pcb.run`. Future versions may expose:

- **Geometry hooks** — programs that modify `originalVertices` to reshape the rest state
- **Agent hooks** — custom Friend subclasses with different traversal strategies
- **Oracle hooks** — programs that replace the oracle expression with live functions
- **Frontier hooks** — intercept/augment Claude API calls before they execute

These are not in the current kernel. They are listed here to constrain their future design: any extension must preserve the kernel stability invariant.

---

10. Conclusion

SEW establishes a manifold as a programmable signal medium. The $Sym^2(X)$ construction gives it mathematical grounding: the mesh is not arbitrary but derives canonically from the user's drawn points. Heat diffusion gives it physical grounding: signals propagate and decay by the graph Laplacian. The program injection model gives it operational grounding: programs are isolated, composable, and additive.

The kernel is frozen. Programs are open. The heat buffer is shared.

That is the whole system.

---

cuppcb.com — permacomputer infrastructure — MIT

russell.ballestrini.net · www.foxhop.net · www.TimeHexOn.com

truth · freedom · harmony · love

MIT License. Copyright 2026 brackishbert@gmail.com · russell@unturf.com · cuppcb.com. Permission is hereby granted, free of charge, to any person obtaining a copy of this document to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies, subject to the above copyright notice and this permission notice being included in all copies or substantial portions.