GLYPHD LABS

WORKING PAPER 30

Computation by Observation

The Virtual Optical Compute Fabric: multiscale field communication among computational systems

Hayden Lindley / Glyphd Labs

2026-08-01 / Version 0.1.0

EVIDENCE: SPECIFIED

Computation by Observation

Abstract

Modern computers contain many computational systems that can operate concurrently, but their cooperation is usually organized through function calls, queues, serialized messages, pairwise APIs, or private shared buffers. A physical display-to-camera experiment suggested a different primitive: one system may publish state into an observable field while another acquires only the region, resolution, channels, and time span it needs.

This paper specifies the Virtual Optical Compute Fabric (VOCF): a GPU-resident, transport-independent spatial runtime in which computational systems publish, observe, transform, and recursively refine shared state through programmable multiscale views. Optical names an observation-based programming model. It does not require photonic hardware, visible RGB pixels, monitors, or camera capture inside the host. A surface may be a bitplane, texture, sparse tensor, graph field, token lattice, event plane, or structured buffer.

The architecture consists of Fabrics, Surfaces, Panels, Lenses, Operators, Instances, Epochs, Commit Gates, Bridges, and Receipts. Many Lenses may sample one canonical Surface without materializing many complete copies. A single host can therefore instantiate global, dual, quad, tiled, hierarchical, or thousands of sparse views. The real limits are memory traffic, transform cost, dispatch overhead, synchronization, contention, locality, and compute rather than physical display refresh.

VOCF is a specified composition, not a performance result. The first required experiment compares multiscale field coordination with full-field and queue-based baselines under controlled workloads.

1. Origin

The immediate trigger was a browser-based optical transfer demonstration in which one phone displayed rapidly changing machine-readable frames and another reconstructed a payload through its camera [P30-S02, P30-S03]. Its deeper importance was not file transfer. It demonstrated that a display can publish state into a field and that another system can recover structured information through observation.

The first extension was a physical compute fabric: nearby screens and cameras could discover one another, advertise capacity, receive assignments, and return results. That remains a useful transport experiment, but it depends on several devices and a commodity optical channel far slower than internal memory.

The decisive move is to remove the camera while retaining the topology. One computer virtualizes the entire environment. Surfaces behave like screens; Lenses behave like cameras, microscopes, viewports, or foveated observers. Computational systems update their own fields with whatever representation is useful. A location may contain one bit, a color, a vector, a token, a graph edge, a timestamp, uncertainty, provenance, or several synchronized channels.

The question is therefore not how quickly a computer can scan a fake monitor. Internally, the reader samples the backing resource directly. The question is:

> *What new computational organizations become possible when systems communicate by publishing observable fields and receiving programmable views of those fields?*

2. Thesis

> *Independent computational systems can be composed through a shared spatial field in which observation is a first-class, programmable, multiscale operation rather than an incidental memory read.*

Current GPU APIs already expose the necessary low-level substrate: storage buffers, storage textures or images, resource views, compute pipelines, explicit bindings, synchronization, and memory control [P30-S04, P30-S05, P30-S06]. Dataflow systems already map computation across heterogeneous devices [P30-S07]. Process networks formalize communicating concurrent processes [P30-S08]. Blackboard systems coordinate independent knowledge sources through a common problem space [P30-S09, P30-S10].

VOCF does not claim to replace those systems. Its proposed contribution is to make observation, scale, locality, projection, time, and recursive system instantiation explicit runtime objects.

3. Runtime objects

3.1 Fabric

A Fabric contains Surfaces, Operators, Instances, schedules, clocks, authority rules, Bridges, and Receipts. It may run inside a browser, native application, game engine, research harness, or larger operating environment.

3.2 Surface

A Surface is a spatially addressable state substrate. It may be dense or sparse, finite or logically unbounded, flat or hierarchical. Its backing representation may be a bitplane, texture, buffer, sparse tensor, voxel field, token lattice, graph projection, event field, or content-addressed tile hierarchy.

The screen metaphor is a contract for location and observation, not a requirement that canonical state be stored as RGB pixels.

3.3 Panel

A Panel is a named workspace, projection, or partition on a Surface. One Surface may contain a full-field Panel, four quadrants, thirty-two specialist regions, or thousands of sparse work areas.

3.4 Lens

A Lens specifies how an observer acquires a view:

A Lens may select a crop, mip level, channel subset, sparse address set, semantic mask, temporal interval, coordinate transform, or derived representation. It combines ideas normally divided among camera, query, subscription, decoder, viewport, and access grant.

3.5 Operator

An Operator reads one or more observations and proposes writes to one or more Surfaces. Operators may be shaders, neural networks, language models, physics solvers, databases, search procedures, symbolic systems, software agents, ordinary functions, emulated processors, or virtual machines.

An Operator may retain private state. The Fabric governs only its observable inputs, proposed outputs, resource budget, permissions, and receipts.

3.6 Instance

An Instance packages Surfaces, Panels, Lenses, Operators, and policies into one virtual computational system. It may represent a model pipeline, multi-agent team, simulated processor, rendering cluster, control system, or nested Fabric.

3.7 Epoch and Commit Gate

An Epoch identifies a coherent version of Fabric-visible state. Operators observe a declared epoch and return proposed patches. The Commit Gate validates authority, resolves conflicts, applies accepted writes, and advances the Fabric. An Operator does not directly rewrite accepted canonical state merely because it generated an output.

3.8 Bridge

A Bridge maps a Surface or Lens contract across a boundary: GPU to CPU, process to process, virtual machine to virtual machine, network to network, or eventually display to camera. The abstraction does not make those transports equally fast. It preserves the higher-level observation contract while permitting transport-specific negotiation.

3.9 Receipt

A Receipt records an accepted transition: source epoch, operator identity and version, observed inputs, proposed writes, conflict decisions, output epoch, exact hashes where applicable, and limitations.

4. Canonical state versus visible projection

A naive implementation would encode every internal object as visible pixels and then decode it. That would add precision loss, conversion cost, synchronization, and unnecessary copies.

VOCF separates three layers:

- 1. Canonical computational representation -- the buffer, texture, tensor, bitset, graph, or other form suited to the workload.
- 2. Observation view -- a typed Lens over that state, sampled directly or materialized temporarily.
- 3. Human or physical projection -- a visible rendering produced only when a person, camera, recorder, or external optical device needs it.

The architecture is inspired by optics without forcing computation through an image codec.

5. One surface, many views

A view count is not necessarily a screen count. One Surface can support:

- one global view;
- a global view plus one high-resolution fovea;
- four fixed quadrants;
- sixteen or thirty-two regional workers;
- hundreds of adaptive tiles;
- or thousands of sparse windows.

A thousand Lenses should normally be descriptors over one allocation, not a thousand copied 4K images. Some Lenses may alias existing subresources; others may sample a lower-resolution hierarchy or materialize a transformed result only when reuse justifies it.

This is the central distinction between view multiplicity and data duplication.

6. Hierarchical and fractal geometry

Hierarchical geometry is useful because every region can have a stable address, parent, children, neighbors, scale, and refinement path. A quadtree, octree, space-filling curve, adaptive partition, or other FOVEA-compatible address system may support:

- foveated processing;
- sparse updates;
- coarse-to-fine analysis;
- locality-aware scheduling;
- hierarchical memory;
- and logically large surfaces whose detail is materialized only where needed.

Fractal geometry is optional. It may organize recursive locality or self-similar topology, but this paper does not claim that fractals compress arbitrary data. The useful property is addressable multiscale structure, not magical storage reduction.

7. Computation cycle

A bounded cycle is:

Different Operators may observe different resolutions, channels, or time spans. A global coordinator may see a low-resolution overview. Regional workers may inspect high-detail areas. A verifier may see proposed changes and evidence but not private working state. A resource controller may observe cost and thermal channels rather than semantic content.

Observation therefore becomes part of scheduling and authority, not merely data access.

8. Recursive computational worlds

Because an Instance may contain another Fabric, one host can instantiate and compare different computational organizations against the same task:

- a central coordinator with sixteen workers;
- a decentralized cellular field;
- a recursive hierarchy of local controllers;
- three expert models plus one verifier;
- a simulated bitplane processor;
- or several competing architectures sharing one measurement harness.

These Instances may cooperate, compete, inspect one another through approved Lenses, or remain isolated. This makes VOCF a possible laboratory for computer architectures as well as an application runtime.

Virtual success does not prove physical performance. Nested scheduling, shared caches, and common host resources may distort a later distributed deployment. Those differences must be measured.

9. Self-observation and feedback

An Instance may receive a Lens over its own previous outputs, uncertainty, error field, resource use, or the way other systems interpreted its work. That supports explicit feedback loops:

Self-observation is not automatically self-understanding. It is a controlled way to expose selected behavior back to the system.

10. Relationship to Glyphd architecture

VOCF does not replace canonical truth or authority.

- LCSM / Backboard remains the truth, memory, provenance, journal, and receipt plane.
- FOVEA Address Fabric supplies address, projection, navigation, locality, and attention-scaled retrieval [P30-S11].
- SurfaceMind compiles governed state into operational surfaces [P30-S12].
- Addressed-state agents observe bounded state and return proposals rather than silently rewriting canon [P30-S13, P30-S14].
- Human acceptance and revocation remain above model output [P30-S15].

A live Surface may be transient, recomputable, speculative, or cached. Only an accepted and receipted transition changes canonical project state.

11. Resource model and real limits

Inside one host, monitor refresh and camera frame rate are irrelevant. The live constraints are:

- bytes read and written;
- memory capacity and residency;

- cache locality and coalescing;
- transform and sampling cost;
- dispatch count and occupancy;
- barriers and synchronization;
- write contention and atomics;
- CPU-GPU transfers;
- persistence and replay cost;
- and scheduling overhead.

For intuition, a 4K one-bit field is about one megabyte. An RGBA8 field is about thirty-three megabytes. Four 16-bit floating-point channels are about sixty-six megabytes. Repeatedly scanning full fields can quickly dominate traffic, while small sparse Lenses may add their own indirection and dispatch costs.

The relevant metric is not nominal panel resolution. It is read amplification:

The architecture is valuable only when the work avoided exceeds the cost of Lens management, synchronization, and transformation.

12. Why this is not merely shared memory

Shared memory already permits many systems to read and write common bytes. VOCF proposes additional semantics:

- stable spatial addresses;
- typed multiscale views;
- explicit temporal windows;
- independent channel permissions;
- derived projections;
- declared update policies;
- versioned epochs;
- conflict-gated writes;
- and receipted transitions.

A plain pointer says where bytes are. A Lens says what part of a field an observer may see, at what scale, through which transform, for which interval, under what authority, and relative to which epoch.

That additional structure may prove useful--or may prove too expensive. The distinction must be tested rather than assumed.

13. Security and authority

A visual metaphor must not become ambient authority. The minimum rules are:

- 1. Lenses are explicit capabilities, not implied access.
- 2. Read and write grants are separate.
- 3. Derived channels inherit disclosure restrictions.
- 4. Operator identity and version are recorded.

- 5. Child Instances receive hard compute, memory, and recursion budgets.
- 6. Cross-Instance Bridges default to deny.
- 7. Proposed writes are not accepted writes.
- 8. Timing, update frequency, and spatial location are treated as possible side channels.
- 9. Receipts preserve what was observed and accepted without claiming that all private working state is canonical.

14. First implementation

A bounded prototype can be built with WebGPU for portability or Vulkan/CUDA-class APIs for lower-level control [P30-S04, P30-S05, P30-S06]. It requires:

- a Surface registry;
- hierarchical address allocator;
- Lens descriptor table;
- Operator registry;
- Epoch scheduler;
- Commit Gate;
- resource budgeter;
- instrumentation;
- visual inspector;
- receipt writer;
- and replay harness.

A minimal Operator interface is:

GPU-native Operators may use a lower-overhead path while preserving the same logical contract.

15. VOCF-1 experiment

The first experiment should test the architecture rather than market it.

Objective

Determine whether multiscale Lens scheduling reduces unnecessary observation work while preserving exact output correctness on a sparse, spatially structured task.

Workload

Use a deterministic synthetic field containing regions of varying complexity, change rate, and uncertainty. Operators must identify, classify, or reconstruct target regions.

Topologies

Compare:

- 1. one full-field Operator;
- 2. four fixed quadrant Operators;
- 3. sixteen fixed regional Operators;
- 4. one global Operator plus dynamic foveated workers;
- 5. sixty-four dynamic sparse workers;
- 6. two hundred fifty-six sparse Lenses;
- 7. one thousand twenty-four small sparse Lenses when supported.

Baselines

Compare VOCF with repeated full-field scanning, explicit queue-based tile assignment, and a direct shared-buffer implementation without Lens abstraction.

Measurements

Record exact correctness, wall-clock latency, bytes read and written, unique bytes required, read amplification, dispatch count, barrier count, memory residency, CPU-GPU transfers, Lens churn, conflict rate, replay consistency, and inspection overhead.

Failure criteria

The performance hypothesis is not supported if Lens scheduling costs exceed saved work, read amplification does not improve on genuinely sparse workloads, synchronization dominates, correctness differs from the pinned baseline, or the implementation depends on hidden workload-specific shortcuts.

A failed result remains valuable because it identifies where the abstraction is too expensive or too general.

16. Additional experiments

Subsequent experiments should test:

- heterogeneous Operators: shader, vision model, symbolic verifier, and language model;
- recursive Instances competing on the same task;
- self-observation through error and uncertainty channels;
- moving one Instance across a process or machine Bridge;
- and expert inspection of live machine state versus ordinary logs.

No interoperability, human-factor, or performance result is claimed before those experiments are specified and receipted.

17. Potential product forms

Possible product surfaces include:

- a compound-AI runtime connecting models, tools, simulations, verifiers, and agents;
- an architecture laboratory for designing compute topologies inside one host;

- a GPU-native agent workspace with bounded spatial context;
- an adaptive simulation runtime that applies detail only where needed;
- and a single-host personal compute fabric that later extends across devices through Bridges.

The product becomes important only if it materially improves latency, attainable workload size, privacy, energy, implementation complexity, observability, or cost. A beautiful field with no measurable advantage is an instrument, not a platform.

18. Failure modes

The architecture may fail because:

- it merely renames ordinary buffers and kernels;
- a workload has no useful spatial locality;
- Lens allocation and transforms dominate execution;
- semantic incompatibility still requires extensive adapters;
- shared fields create write contention;
- recursive Instances exhaust resources or become unstable;
- human-visible projections hide important state;
- Lens timing and location leak information;
- or the contract does not cross process and network boundaries without complete redesign.

These risks are part of the specification, not footnotes to be removed later.

19. Explicit non-claims

This paper does not claim:

- a working VOCF runtime;
- a performance or energy improvement;
- a new compression theorem;
- a photonic computer;
- replacement of message passing, shared memory, or dataflow;
- universal semantic interoperability;
- deterministic behavior for arbitrary Operators;
- safe multi-tenant isolation;
- market adoption;
- or production readiness.

The architecture is SPECIFIED. Performance, transport continuity, inspectability, and market significance remain HYPOTHESIS.

20. Falsification boundaries

The broad architecture should be revised or rejected if controlled implementations show that:

- 1. Lens semantics add persistent overhead without reusable scheduling, isolation, or inspection value;

- 2. multiscale observation does not reduce bytes touched on workloads with genuine locality and sparsity;
- 3. equivalent queue or dataflow systems express the same topologies more simply and efficiently;
- 4. recursive Instances cannot be bounded or replayed reliably;
- 5. the contract cannot cross at least one meaningful runtime boundary without complete redesign;
- 6. human inspectability adds no operational value;
- 7. or the spatial model repeatedly forces unnatural representations.

A narrower result may survive as a sparse GPU scheduler, agent blackboard, architecture simulator, or visualization instrument.

21. Research questions

The next work should answer:

- What is the smallest useful Surface and Lens type system?
- Which workloads naturally benefit from multiscale observation?
- When should a Lens alias existing storage versus materialize a view?
- How should conflicts be resolved without global serialization?
- Which address hierarchy best preserves locality on current GPUs?
- Can Operators cross CPU, GPU, process, and machine boundaries without application redesign?
- What receipt granularity is affordable?
- How should nondeterministic outputs be replayed?
- Can a human-visible projection remain faithful to machine state?
- Does a common field reduce adapter count in real compound systems?
- Which topology changes can occur safely at runtime?
- What is the relationship between Lens scheduling and learned attention?

22. Conclusion

The physical optical-transfer demonstration supplied the clue, not the final system.

The deeper architecture is a computer in which computational systems publish state into shared, addressable fields. Other systems observe those fields through bounded, programmable Lenses. Some see the whole at low resolution. Others inspect one region in detail. Some watch change, uncertainty, provenance, error, or resource use. They transform what they observe, propose writes, and advance the Fabric through explicit commits.

One host can contain many such systems. Views may be single, dual, quad, sixteen-way, thirty-two-way, hierarchical, or thousands of sparse windows. Geometry may be static, adaptive, fractal, or dynamically allocated. The limits are memory traffic, compute, dispatch, synchronization, locality, contention, and authority--not the refresh rate of a physical monitor.

The concise proposition is:

> *Instead of programs only sending messages to one another, they may show one another what they know.*

The Virtual Optical Compute Fabric is the proposed runtime for that proposition. It is now preserved as a bounded, falsifiable architecture rather than an idea that must survive through conversational memory alone.