Foveal Playground — Plan C: Map-driven frontier exploration

Nov 8, 2026 · @Luiz Velho

Question and scope

Question. If the gaze policy reads the persistent 3-D map, can the explorer reach the same reconstruction with fewer fixations, shorter saccades or a better-timed stop? TR-26-11 (Section 13) lists "map-driven frontier exploration, closing the loop from the map to the policy"; Figure 7 draws that loop as the one dashed arrow not in the baseline.

The rows changed. Local gaze policy (8 neighbours with edge support × unseen gain → map-driven 3-D frontiers or information gain), then the global policy and the stop rule, each as its own step. Renderer, oracle matcher, triangulation, fusion, coverage and the lattice stay fixed.

In scope. A read-only map summary the policy may use; frontier and gain definitions on it; local, global and stop variants; an exact offline simulator so that policies can be developed without the GPU.

Out of scope. Gazes off the 356-gaze lattice (possible, but they need rendering and break the simulator); moving the head; natural stereo.

A note on the baseline. TR-26-11 describes v0 (8 tangent-frame neighbours, a jump to the deepest cell of the largest unexplored region; 542 fixations). The repository ships Playground-2 (lattice neighbours within 16°, a jump to the unvisited lattice gaze of largest unseen gain; 356 fixations, 315 local, 40 global). This plan starts from Playground-2.

Starting point

`explorer.decide(state, core_valid, yaw, pitch, visited)` is a pure function of the 1° coverage sphere, the current fixation's 256 × 256 valid-geometry mask, the current gaze and the visited list. The map is written every fixation and never read during control.

- **Local:** unvisited lattice gazes within 16°; eligible if `EDGE_SUPPORT ⩾ 0.25` and `UNSEEN_GAIN > 0`; highest product wins. Edge support looks only at the *current* fovea's outer 64-px band, so it forgets everything earlier fixations showed about where surfaces continue.
- **Global:** the unvisited lattice gaze with the largest unseen gain, wherever it is.

- **Stop:** when no lattice gaze adds unseen area (`SEEN_STOP` = 2.0 is unreachable on purpose).

Why the fixation count cannot move under the current stop. The 356 lattice footprints are non-redundant: each covers a cell no other one covers. So the stop fires only after every lattice gaze has been fixated, whatever the policy. Any lattice policy with this stop makes exactly 356 fixations. A map-driven policy can change only:

1. the *order* of fixations (how fast DEPTH and first-hit coverage grow);
2. the *path* (total saccade angle, number and length of global jumps — the "late global tail" the report discusses);
3. the *stop*, once the stop rule may also read the map. This is where fewer fixations can come from: the last lattice gazes often add only slivers of unseen solid angle, or point at sky or at depth holes no gaze can fill.

Depth holes the head cannot fill. The 0.74 % of the sphere that is SEEN but not DEPTH probably comes mostly from binocular half-occlusion next to depth edges (a hypothesis M2 checks). With a fixed head the two eyes' viewpoints never change, so re-fixating such a cell cannot help. The map can recognise these cells (a range jump next to them) and stop spending fixations on them.

Design

An exact offline simulator comes first. With a fixed head and vergence fixed at 2.1 m along the gaze, what a fixation measures depends only on its gaze, not on the order of fixations. The baseline run fixates all 356 lattice gazes once, so one baseline run is a complete *observation cache*: for every lattice gaze its float64 points, its `core_valid` mask (saved in `policy/state-NNNN.npz`) and its display RGB. `tools/greedy_foveal/simulate.py --cache BASE --policy P` runs the whole control loop — fusion, coverage, decision, stop — from the cache, on a laptop, in minutes. It is exact for any policy that stays on the lattice. (The cache needs the float64 points: `points.npz` stores float32. Use Plan B's `--save-points-f64` , or add it here.)

The policy reads a map summary, not the map. Querying 17 M surfels each decision is slow, and replaying a decision would need the whole map at that moment. Instead, after each fusion, compute fixed-size spherical layers from the map and pass them to `decide(…, summary=S)` :

layer (per 1° cell, or 0.5°)	meaning
`count`	surfels whose direction from the H0 origin falls in the cell
`range_min`, `range_med`, `range_max`	ranges of those surfels, m; min ≠ max flags a depth edge inside the cell
`support_med`	median support: how often the cell's surface has been re-observed
`normal`	mean surface normal from a local plane fit (or from the range gradient on the sphere)
`frontier`	a DEPTH cell next to an UNSEEN cell whose range field continues smoothly toward it
`hole_kind`	for SEEN-not-DEPTH cells: next to a depth edge (half-occlusion), no hit at all, or unknown

The summary is saved with each decision in `policy/state-NNNN.npz` as extra arrays, so `run.checks` can still re-decide every state. The existing keys stay unchanged, so the baseline's signature still holds.

The truth firewall. The summary builder sees a read-only map view exposing `xyz`, `support`, `first_fix` and `n` only. It never sees `inst` (the oracle label). It never sees `rgb` either: RGB is not reproducible run to run under Cycles OPTIX, and a policy that read it would lose bitwise determinism.

Policy and stop variants, each tried alone first:

id	row	change
L1	local	replace `EDGE_SUPPORT` with map frontier support: frontier cells inside the candidate's footprint, from all fixations so far
L2	local	information gain: predicted new surface in the footprint, extrapolating planes fitted to bordering DEPTH cells; skip cells whose `hole_kind` is half-occlusion

id	row	change
G1	global	nearest unvisited lattice gaze with frontier gain above a threshold (shorter jumps)
G2	global	gain ÷ (1 + saccade angle ÷ θ_0): trades gain against travel
S1	stop	stop when no unvisited gaze has predicted gain (L2) above τ; sweep τ

The main hypothesis is that S1 can stop well before 356 fixations at a small loss in first-hit coverage. L1 and G1 should mainly shorten the path and the global tail.

Work plan

Six milestones on branch `project/map-frontier`. Only M0's cache run and M5 need the GPU.

1. **M0 — Observation cache and simulator.**
   - One full baseline run with float64 points saved; `simulate.py` builds the cache from it, keyed by lattice index.
   - *Check that can fail:* simulating the baseline policy reproduces the real run bitwise — every gaze, decision, the SEEN / DEPTH histories and the final map.
   - *Negative control:* zeroing one gaze's `core_valid` in the cache must change the simulated trajectory.
2. **M1 — Evaluation and headroom, before any new policy.**
   - First-hit evaluation (exercise E3, shared with Plan A), computed after every fixation of a simulated run to give a coverage curve.
   - Lower bounds: random lattice order (10 seeds) and the baseline. Upper bound: a REFERENCE-greedy order that picks the gaze adding the most first-hit coverage. It reads the reference, so it is labelled REFERENCE and is never a policy.
   - The gap between the baseline and the REFERENCE curve is the most any policy can win. Measure it first.
3. **M2 — Map summary, firewall, hole classification.**
   - `summary.py` builds the layers in Design from the map view; `decide` takes `summary=None`; the summary is saved in the policy state.

- *Firewall test:* a map view whose `inst` and `rgb` raise on access; the builder must run.
- *Known answer for `hole_kind`:* a plane in front of a wall must give a half-occlusion strip whose angular width matches the baseline × (1/Z_near − 1/Z_far).
- Report how much of Classroom's SEEN-not-DEPTH area each `hole_kind` explains.
- *Check:* with the summary computed but unused, a GPU smoke run still PASSES `compare.py check`.

4. **M3 — Local variants L1, L2** in the simulator; their growth curves against M1's bounds.
5. **M4 — Global variants G1, G2 and stop S1.** Sweep τ and the travel weight θ_0; plot first-hit coverage at the stop against fixations used.
6. **M5 — Confirmation.** The best one or two variants as real GPU runs. *Check:* the simulator's prediction matches the real run bitwise on the control path. Then `checks` with the new policy's known answers, and the report.

Evaluation

Every policy is judged on the same cache, against the same three references: random order, the Playground-2 baseline and the REFERENCE-greedy upper bound.

measure	definition
first-hit coverage curve	% of first-hit solid angle with a surfel within 12 mm, after each fixation
area under that curve	over fixations 1–356, normalised so the upper bound is 1
fixations to 95 / 99 %	of the baseline's final first-hit coverage
DEPTH curve	from `depth_history`, as a control-side cross-check
total saccade angle	sum of angles between consecutive gazes, degrees
global jumps	count, and mean angle
stop (S1 only)	fixation of the stop, and first-hit coverage there
decision time	mean and maximum seconds per decision, including the summary

Figures. Coverage curves of all variants and the three references on one axis. For S1: coverage at the stop against fixations used, one point per τ, with the baseline's (356, final coverage) point marked. And the trajectory on the sphere (`visualize`) for the best variant beside the baseline's.

What would count as a result. L1 or G1 beating the baseline's area under the curve and its total saccade angle; S1 reaching 99 % of the baseline's final first-hit coverage in clearly fewer than 356 fixations. Write the thresholds into the branch README before M3.

Risks and open questions

- **Small headroom.** The baseline already ends at 100 % SEEN and 99.26 % DEPTH. If M1 shows the REFERENCE curve close to the baseline's, the win must come from the stop (S1) and the path, not the order. That is still a result; report it as such.
- **Overfitting to Classroom.** Thresholds (τ, θ_0, the frontier smoothness test) tuned on one scene may not transfer. Re-run the final variants on at least one of Plan A's scenes.

- **Simulator drift.** The simulator is exact only while gazes stay on the lattice and the head and vergence stay fixed. Any variant that leaves the lattice must render, and M5′s simulator-vs-real check is the guard.
- **The upper bound must not leak.** REFERENCE-greedy reads the first-hit reference. Keep it in its own module, outside `explorer.py`, and make `simulate.py` refuse to import it for a policy run.
- **Summary cost.** A per-fixation plane fit over many cells could cost more than the render. Build the layers incrementally from each fused patch′s surfels, not from the whole map.
- **Interaction with Plan B.** De-duplicated or voxel maps change `count` and `support`. Develop against `GlobalMap` (or the bitwise-equal hashed map) and state which map a result used.
- *Open question:* should information gain count solid angle (what the coverage sphere counts) or metric area (what the map holds)? They rank near and far surfaces differently. Try both in L2.

Deliverables and effort

About 7 student-weeks, with 3–4 GPU runs; policy work itself runs on a laptop (ESTIMATE).

milestone	deliverable	effort (ESTIMATE)
M0	observation cache, `simulate.py`, bitwise reproduction of the baseline	1.5 weeks
M1	per-fixation first-hit curves; random and REFERENCE bounds; headroom figure	1 week (less if Plan A′s M1 exists)
M2	`summary.py`, firewall test, `hole_kind` known answer, hole breakdown	1.5 weeks
M3	L1, L2 with curves	1 week
M4	G1, G2, S1 with sweeps	1 week
M5	GPU confirmation, simulator-vs-real check, report	1 week

The observation cache and simulator are also the fastest way for Plan D to test object-aware attention, so it is worth merging M0 to `main` early.