Cursor-Ready Project Spec

Discovering and verifying the 6-point MHV gravity form via positive geometry (n=6 DCP wonderful model / channel arrangement).

Date: 2025-12-29

Inputs provided

- Latest Sage script: **54.sage** (to be refactored into modular stages).
- Progress summary PDF: key findings and current bottlenecks.

Purpose of this document

Give Cursor a precise, runnable plan: what to build, how to save/resume work, what constitutes success (Hodges), and how to maximize the probability of real physics progress per compute-hour.

1. Executive summary

You have a working exact pipeline that repeatedly finds large nested-set charts and performs boundary-intersection scans. Under strict S6 symmetry the invariant space is small but factorization fails massively; under relaxed symmetry (e.g., S3 x S3) you have at least one nontrivial factorization HIT. The fastest path is: (A) generate candidates in an enlarged space, (B) project/reduce them toward S6 using additional physical constraints, and (C) run a hard verification suite against Hodges (up to scale).

Key principle: scanning is not discovery; verification is discovery.

2. Current progress snapshot (from the attached report)

- Built full M6 matroid / channel arrangement (25 channels, rank 10), OS3 space (dim 2008), and computed S6-invariant vectors (dim = 2).
- Building set contains 3008 connected flats with ~882k incompatible pairs; randomized nested-set search finds charts of size ~407 flats.
- Exact QQ chart scans enforcing strict S6 symmetry consistently fail factorization with ~77k-78k bad constraints per chart.
- Optimizations added: sign-table precompute, NumPy-optimized loops, multiprocessing, caching of invariants/flats/incompatibility (no chart caching).
- Breakthrough: in relaxed-symmetry nullspace scan (dim ~58), a chart of size ~407 produced a factorization HIT (nontrivial alpha satisfies constraints).
- Independent n=4 unit test confirmed permutation action / representation logic.

Interpretation: full S6 may be overconstraining at the construction stage; build in a larger space, then impose the 'right' additional constraints to recover the unique S6 object (Hodges) if it exists.

3. Goal, deliverables, and definition of success

Goal: Construct a canonical top log-form ("gravity scattering form") on the n=6 DCP wonderful model with correct poles and factorization, reproducing (or uniquely determining) the Hodges determinant for 6-pt MHV gravity.

Deliverables (machine-usable):

- A reproducible pipeline with three executable stages: scan, reduce/project, verify.
- Persistent cache artifacts (flats, incompatibility, group action, invariant bases, precomputed sign tables).
- Run artifacts per experiment: state.json, charts.jsonl, candidates.jsonl, verify.jsonl, report.md (and STOP.json on success).
- A verification routine that matches Hodges up to scale on many test points, across charts, and across boundary residues.

Success criteria ("T3 / Hodges-level"):

- Candidate passes the full boundary/factorization suite (target boundary set specified in config).
- Candidate reduces to a unique object up to scale under S6 (or a minimal residual ambiguity that is eliminated by documented extra physical constraints).
- Candidate matches Hodges det'Phi up to overall scale on N >= 20 independent rational kinematic points, with out-of-sample checks.
- Cross-chart consistency: independent charts yield the same result.
- All results are reproducible from a single run_id and config file.

4. Architecture: scan -> reduce -> verify

This project must be run as a two-phase (actually three-stage) pipeline. Cursor should never declare success from scan output alone.

Stage A - SCAN (fast candidate generation)

- Sample/build nested-set charts; compute chart descriptors (flat masks) and stable chart_id.
- Assemble constraint matrices efficiently (allow modular prefilters to reject hopeless charts quickly).
- Solve for nullspace directions / candidate coefficient vectors (exact QQ for final candidates; mod-p for filtering).
- Write candidates.jsonl continuously; update state.json after each chart/boundary.

Stage B - REDUCE/PROJECT (toward S6)

- Load candidate vectors from enlarged symmetry space (e.g., S3 x S3).
- Project onto S6-invariant basis (dim 2 known); record coordinates and whether projection is nonzero.
- Optionally apply additional constraints (see Section 7) to reduce to dim 1 up to scale.

Stage C - VERIFY (hard gate)

- Evaluate candidate and Hodges det'Phi on random rational kinematic points; match up to scale.
- Cross-chart checks: same candidate expressed in different charts yields consistent results.
- Residue/factorization checks across all specified boundaries; emit VERIFY_PASS only on full pass.

5. Persistence: caching + resume (do not rerun the same experiments)

Cursor must make the pipeline crash-safe and resumable. The compute server can die; the run must resume without recomputation.

Persistent cache directory (shared across runs)

- flats.sobj: connected flats for the building set (keyed by n, arrangement spec).
- incompat.sobj: incompatibility pairs (or compressed bitmap representation).
- group_action_*.sobj: permutation action matrices / representations.
- invariant_basis_S6.sobj (and others): basis vectors for invariant subspaces.
- signtables_*.npz: precomputed sign tables / orientation data used in fast scanning.

Run directory (append-only artifacts)

- state.json: which (boundary, chart_id) jobs are complete; best candidates so far; verification status.
- charts.jsonl: each line is a chart descriptor (chart_id, size, seed, trial index, score).
- candidates.jsonl: each line is a candidate record (candidate_id, chart_id, boundary set passed, alpha vector).
- verify.jsonl: verification attempts and outcomes, including fitted scale and residuals.
- report.md: human-readable summary written at end of run and after any VERIFY_PASS.

Cache keys

Key every expensive artifact by a stable hash of (n, symmetry mode, boundary set spec, seed, trial count, chart_id, and a code-version hash). Before computing: check for existing artifact; otherwise compute and write atomically (temp file then rename).

6. Interfaces: logs, JSONL schemas, and configs

Cursor should make outputs machine-readable so the agent can decide what to do next without guessing.

Log line conventions (human + machine)

```
[YYYY-MM-DD HH:MM:SS] STAGE=SCAN boundary=(1,4,5) chart_id=... trial=... workers=...
[... ] HIT level=2 chart_id=... candidate_id=... null_dim=... bad=0 passed=...
[... ] VERIFY_PASS candidate_id=... scale=... points=20 charts=2 boundaries=10
[... ] STOP reason="Hodges match" run_id=...
```

JSONL record sketches

```
charts.jsonl:
  {"chart_id":"<sha256>", "size":407, "seed":42, "trial_index":12345, "score":401, "flats_digest":"

candidates.jsonl:
  {"candidate_id":"<sha256>", "space":"S3xS3", "chart_id":"<sha256>", "boundaries_passed":[...],
   "alpha":{"type":"QQ_vector","num":[...],"den":[...]}, "metrics":{"null_dim":8,"bad":0,"terms":..

verify.jsonl:
  {"candidate_id":"<sha256>", "status":"PASS|FAIL", "scale":{"num":...,"den":...},
   "points_tested":20, "max_rel_error":"1e-30", "charts_tested":2, "details":{...}}
```

Config (YAML) example

```yaml
n: 6
symmetry_mode: "S3xS3"      # or "S6"
trials_total: 150000
seed: 42
max_charts: 50
workers: 44
chart_keep_cap: 300
boundary_set:
  type: "3|3_all"           # 10 boundaries for n=6
scan:
  modp_prefilter: true
  modp_primes: [1000003, 1000033]
  exact_verify_each_hit: true
verify:
  hodges_points: 25
  require_cross_chart: 2
  require_all_boundaries: true
paths:
  cache_dir: "cache"
  runs_dir: "runs"
```

7. Verification suite (turn candidates into physics)

Most false positives come from insufficient verification. The verification suite must be deterministic, multi-point, and cross-chart.

7.1 Hodges matching (up to scale)

- Generate rational kinematic points that satisfy any imposed kinematic constraints (record them to disk).
- Evaluate candidate object at each point; evaluate Hodges det'Phi at the same points.
- Fit overall scale using first 1-2 points; validate on remaining points (out-of-sample).
- Require tight consistency threshold and stability across independent point sets.

7.2 Residue/factorization checks

- For each boundary s_S=0 in the boundary_set: compute Res_{s_S=0} Omega_cand.
- Compare to expected wedge/product structure (lower-point forms) or to the project's current factorization constraints.
- Record per-boundary pass/fail with diagnostics (which rows fail, which terms dominate).

7.3 Cross-chart consistency

- Re-express the same candidate in at least 2 unrelated charts (different nested sets) and re-run evaluation.
- Reject candidates that depend on chart artifacts (coordinate singularities, degenerate u choices).

Optional additional constraints to break degeneracy (use only if needed)

- Gram-locus / 4D restriction: impose the 4D Gram determinant conditions if your target kinematics is 4D.
- Soft/collinear limits: enforce known soft behavior; check scaling as a momentum becomes soft.
- CHY pushforward check: compare to known CHY representation for MHV gravity, if available in your setup.
- Boundary recursion: require compatibility of residues with a recursion relation across multiple boundaries.

8. Performance playbook (maximize physics progress per hour)

This section is tuned for CPU-heavy exact algebra + combinatorial search on a multi-core server.

- **Mod-p prefiltering:** quickly reject charts that cannot satisfy constraints by solving modulo a few large primes before any QQ work.
- **Incremental constraints:** add constraints in batches and early-stop as soon as inconsistency is detected (avoid building full matrices unnecessarily).
- **Chunked trials:** run trials in fixed-size batches and checkpoint state after each batch to avoid losing progress.
- **Work stealing:** assign independent (boundary, chart) jobs to workers; keep them busy and avoid global locks.
- **Memory discipline:** store large sparse structures in compressed form; avoid duplicating flats/incompatibility in each worker.
- **Deterministic seeding:** derive per-worker seeds from (global_seed, worker_id, boundary_id, chart_id) so reruns reproduce behavior.
- **One source of truth:** all expensive artifacts live in cache/; runs/ only contains lightweight descriptors and results.

Hit triage levels (so the agent knows what to verify first)

- **Level 1:** partial pass or small bad-count; record but do not verify immediately.
- **Level 2:** passes the full current boundary subset on one chart/seed; schedule verification on additional points/charts.
- **Level 3:** passes full boundary set and survives projection toward S6; run full Hodges verification. On Level 3 PASS: stop and write STOP.json.

Auto-tuning (adaptive search heuristics)

- Track features of charts that yield hits (size, boundary mask patterns, closure stats) and bias sampling toward those regions.
- Use a multi-armed bandit style allocation: spend more trials on chart generators that produced Level 2 hits recently.
- If a symmetry choice produces only failures for many runs, automatically switch to a broader symmetry mode and re-run reduce/projection later.

Stop rules (prevent burning compute)

- Stop Stage A if you have K (e.g., 10) distinct Level 2 candidates and none survive Stage C verification after M attempts; then auto-adjust search heuristics.
- Stop any chart early if mod-p rejects it or if bad-constraint count exceeds a configured threshold after partial constraint batches.
- Stop the whole run immediately on VERIFY_PASS and write STOP.json + report.md.

9. Cursor agent operating procedure

Cursor should follow this loop without manual babysitting.

```
1) Load config and state.json (resume).
2) Ensure cache artifacts exist (flats, incompat, invariant bases, sign tables). Build missing arti
3) SCAN until:
   - chart budget reached, or
   - candidate threshold reached, or
   - time budget reached.
4) For each new candidate:
   - run REDUCE/PROJECT,
   - then run VERIFY.
5) Write report.md after each major milestone and at run end.
6) If VERIFY_PASS: write STOP.json and terminate cleanly.
```

What the agent must never do

- Never rerun an identical (boundary, chart_id) job if it is already recorded as complete in state.json.
- Never claim a physics result without Stage C verification.
- Never silently change definitions (basis ordering, sign conventions, boundary set) without bumping the code-version hash.

10. Runbook: local vs compute server vs CoCalc

You can run the same repo locally or on a remote server; the key is preserving cache/ and runs/.

Recommended workflow

- Develop and refactor in Cursor locally (or via remote SSH).
- Run compute on the server; keep cache_dir on fast disk; rsync cache_dir between machines only when needed.
- Use git for code; do not store huge binary caches in git.
- If using CoCalc: explicitly sync project files to the compute server and keep cache_dir in a known path.

Environment checklist

- SageMath installed and runnable via `sage` (same major version across machines if possible).
- Python packages used by the pipeline available (NumPy, etc.).
- Sufficient file descriptor limits for many worker processes (ulimit -n).
- Enough RAM to avoid swap; if swap occurs, reduce workers or switch to mod-p prefilter earlier.

11. Troubleshooting and sanity checks

- If full S6 fails with huge bad counts: confirm constraint rows are correct; then treat as overconstraint and continue in enlarged space.
- If results vary wildly across seeds: check deterministic seeding and that workers are not sharing mutable global state.
- If OOM occurs: reduce workers, avoid duplicating large structures per worker, and checkpoint more frequently.
- If you see frequent near-HITs: prioritize verifying the best candidates across multiple charts before increasing trials.
- If you get HITs that do not reproduce: treat them as artifacts; require reproduction on a second chart and second seed before promoting.

Minimal next actions for Cursor (first commit)

- Refactor 54.sage into src/scan.sage, src/reduce_symmetry.sage, src/verify_hodges.sage with shared utils.
- Implement chart_id + candidate_id hashing and JSONL writers.
- Implement state.json resume skipping for (boundary, chart_id) and (candidate_id) verification.
- Create configs/quick.yaml for fast iteration and configs/deep.yaml for overnight runs.
- Write report.md generator that summarizes best hits and verification outcomes.

Appendix A: paths to provided inputs

- Sage script: /mnt/data/54.sage
- Progress summary: /mnt/data/positive_geometry_gravity_progress_summary.pdf

End of spec.