Tagma-Geo

Spatial Query Layer on Structural Coordinate Spaces

Taeho Lee*
SSCCS Foundation
ssccs.org

July, 2026

Abstract

CoordPath[1] treats coordinate space as a tree: each path is a sequence of Coords through N levels of 11,172-slot arrays. This provides collision-free O(k) point lookup and pointwise mutation, but spatial queries – proximity, bounding box enumeration, distance metrics, compound axis filters – fall outside its direct expression. CoordCube (from `tagma-geo`) reinterprets the same CoordPath keys as D-dimensional coordinates. Spatial operations – proximity, bounding box, distance metrics – are provided by `tagma-geo` via the `SpatialOps` and `DistanceMetrics` traits. Construction costs 0.96 ns; axis extraction costs the same as raw path access (319 ps). The design targets workloads where point lookup is necessary but insufficient: geometric queries over coordinate-indexed data where the coordinate itself carries spatial meaning. This document is a draft work-in-progress as the CoordCube layer continues to mature.

1 Introduction

A CoordPath answers one question: "does this exact path exist?" Many workloads require a different class of question: "what paths are near this point?", "what paths lie within this bounding box?", "what is the distance between these two points?" CoordPath alone cannot answer these without external iteration over manually specified paths, secondary index structures, or fallback to hash-based spatial indexing.

CoordCube fills this gap. It reinterprets an existing CoordPath storage key as a D-dimensional coordinate grid, where the N bytes of the path are partitioned into D dimensions of R bytes each (N = D x R). From this interpretation, three capabilities emerge:

- Proximity queries: enumerate all paths within L-infinity radius of a center point
- Bounding box enumeration: list all paths whose coordinates fall within a rectangular region
- Distance metrics: compute Hamming distance between coordinate pairs at sub-2 ns

The reinterpretation is zero-cost: no re-encoding, no duplication, no index rebuild. The same bits that serve as a tree path now also serve as a geometric coordinate.

*Founder & Architect; Corresponding author: lee@ssccs.org
[1]Tagma Whitepaper: https://docs.ssccs.org/projects/syntagma/tagma/wp/

2 Spatial Interpretation Model

A CoordCube reinterprets a CoordPath as a D-dimensional coordinate grid by partitioning its N coords into D groups of R coords each. It does not reference any storage – CoordCube is a pure geometric interpretation of the path itself. Spatial queries (proximity, bounding box, distance) are generated from this interpretation independently of any storage backend. When used with a KV store, the CoordCubeKV trait (from `tagma-kv`) combines path generation with store lookup.

The dimensional decomposition is the only configuration: given N bytes per path and D dimensions, each dimension receives R = N / D bytes. This decomposition is declared at CoordCube construction and can be changed per query without storage overhead.

3 Query Types

3.1 Proximity

Enumerate all paths within L-infinity radius r of a center coordinate C. Generation cost is O(A(r) x D) where A(r) is the number of paths in the L-infinity annulus. Throughput ranges from 60.6 Melem/s (r=0) to 365.1 Melem/s (r=5), increasing with radius as fixed overhead amortizes. These figures include Vec allocation for the result set, which accounts for approximately 70% of the measured latency at r=1.

3.2 Bounding Box

Enumerate paths within a rectangular region defined by lower and upper bounds per dimension. Bounding box throughput is consistently higher than proximity because no distance computation is required during iteration.

3.3 Distance Metrics

Hamming distance between two coordinates is computed at 1.75 ns. No path lookup involved.

3.4 Compound Axis Filter

Fuse proximity generation or bounding box enumeration with CoordSet axis filters inside a single iteration, avoiding Vec allocation entirely. Compound queries achieve 13.5 ns vs 84 ns for collect-then-filter (6.2x).

4 Cost Model

CoordCube proximity overhead on dense stores is 127 ns over sequential lookup, dominated by Vec allocation (37 ns) and push (74 ns, 9 paths). Coordinate arithmetic accounts for 16 ns (13%). On sparse stores, CoordCube becomes faster than sequential lookup (up to 3.3x at 48.5 ns vs 158 ns) because it avoids tree lookups for nonexistent paths. On empty stores, CoordCube returns immediately at 15.7 ns (pure path generation cost), while sequential lookup still pays 158 ns for 9 tree misses. The crossover point is at approximately 55% hit rate.

CoordCube query cost is bounded by region size (path count in the result), not store size. At 10M entries, CoordCube proximity completes in 285 ns vs HashMap spatial filter at 238 ms – a million-fold advantage.

5 Comparison with Existing Systems

Aspect	CoordCube	R-tree	GeoHash	HashMap filter
Point lookup	O(k) direct	O(log N)	O(1) hash	O(1) average
Proximity	O(paths) bounded	O(log N + results)	O(N) full scan	O(N) full scan
Index maintenance	zero (view)	O(log N) insert	O(1)	zero
Storage overhead	zero (reinterpretation)	pointer-based	hash-based	none for scan
Distance metric	O(D) hamming	O(log N)	O(1)	O(1)

CoordCube is not a spatial index in the traditional sense. It is a spatial interpretation layer that adds zero storage overhead, zero index maintenance, and zero data duplication. The cost is paid only at query time and is bounded by region size, not store size.

6 Status

This document is a draft. The CoordCube layer is implemented and benchmarked within the synTagma[2] repository. Hardware design implications and topology mapping for cluster deployment are discussed in the main Tagma whitepaper[3].

References

Appendices

6.1 Core Advantage: When CoordCube Wins

The following benchmarks directly measure CoordCube's value proposition. Proximity throughput determines raw geometric query speed. The crossover point vs sequential lookup shows when CoordCube is the faster choice. Iterator fusion demonstrates the structural efficiency gain over naive collect-then-filter.

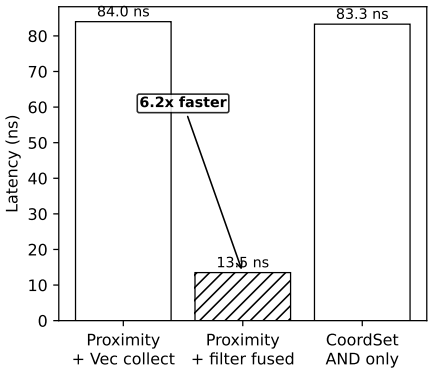

Figure 1: Compound query efficiency: iterator fusion vs collect-then-filter

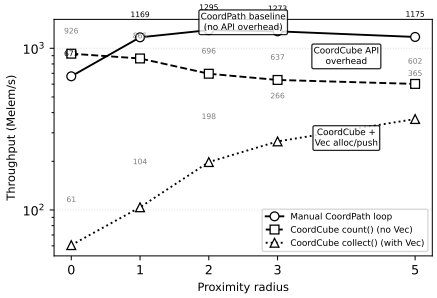

Figure 2: Proximity path generation throughput: manual CoordPath vs CoordCube

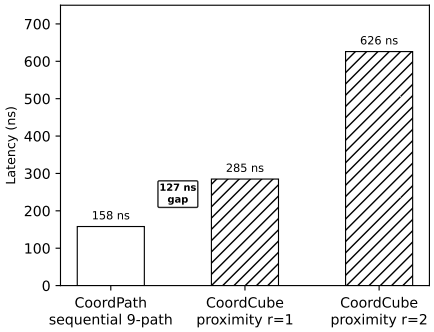

Figure 3: CoordCube vs CoordPath on KV store

[2]synTagma repository: Github (Pre-release, Apache 2.0)
[3]Tagma Whitepaper: https://docs.ssccs.org/projects/syntagma/tagma/wp/

6.2 Behavioral Characteristics

The following charts detail CoordCube's cost structure, density sensitivity, and scaling behavior. These explain the tradeoffs behind the core advantage figures above.

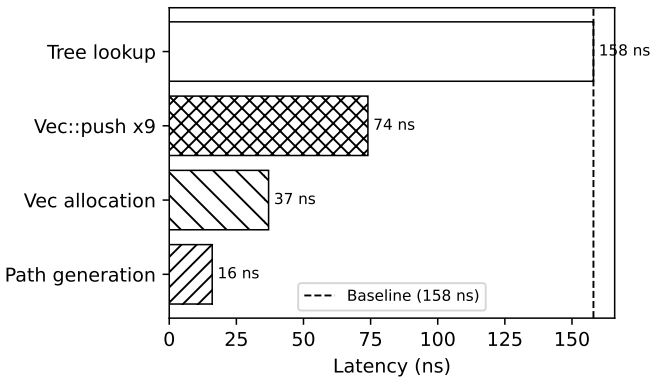

Figure 4: Where does the 127ns CoordCube overhead go?

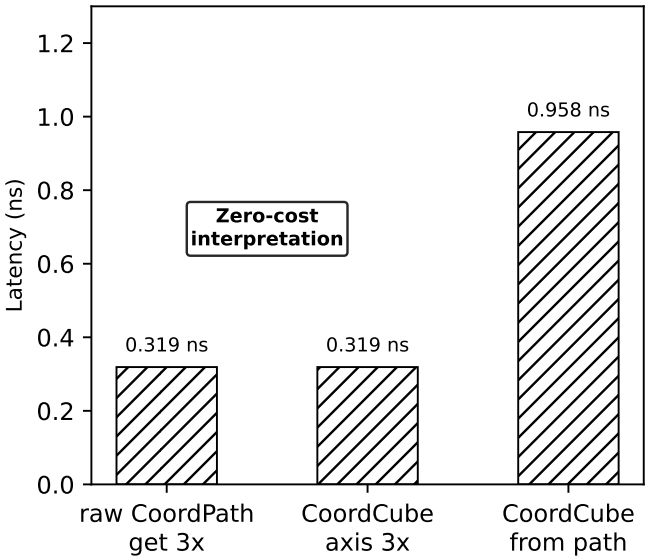

Figure 5: CoordCube overhead vs raw CoordPath

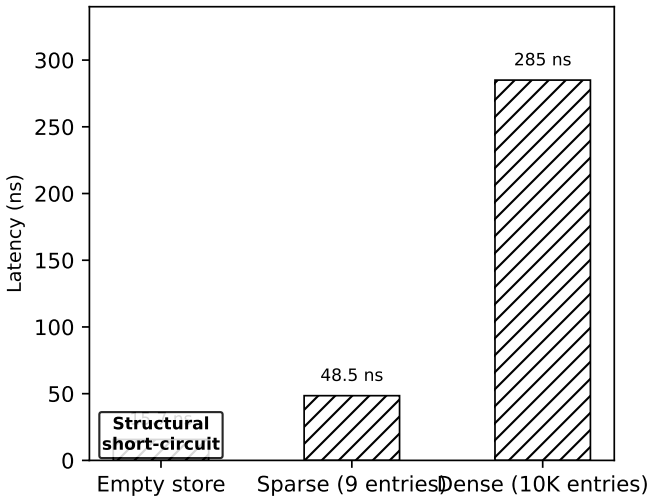

Figure 6: Store density effect on CoordCube proximity

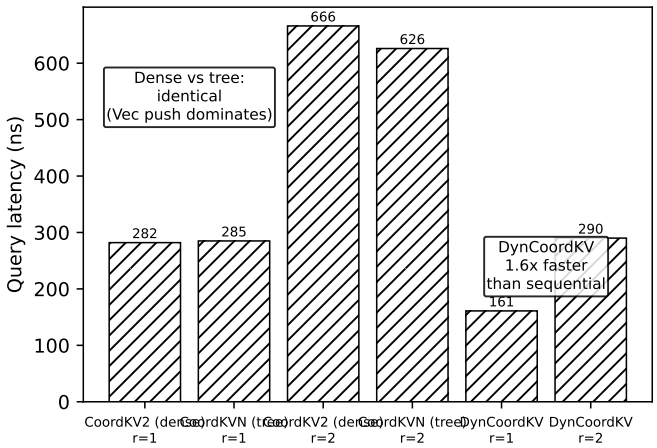

Figure 7: CoordCube on different backends: same query, same cost

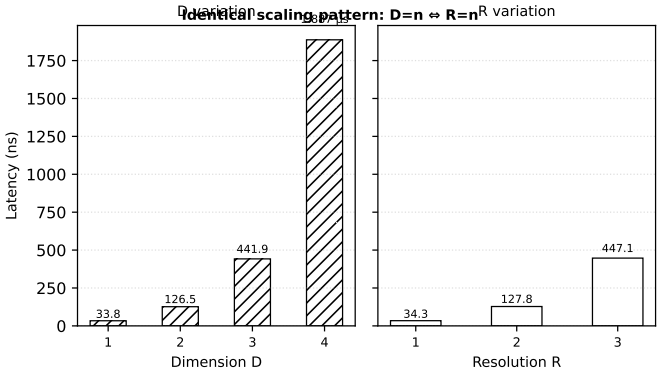

Figure 8: Dimensional vs Resolution scaling (proximity r=2)

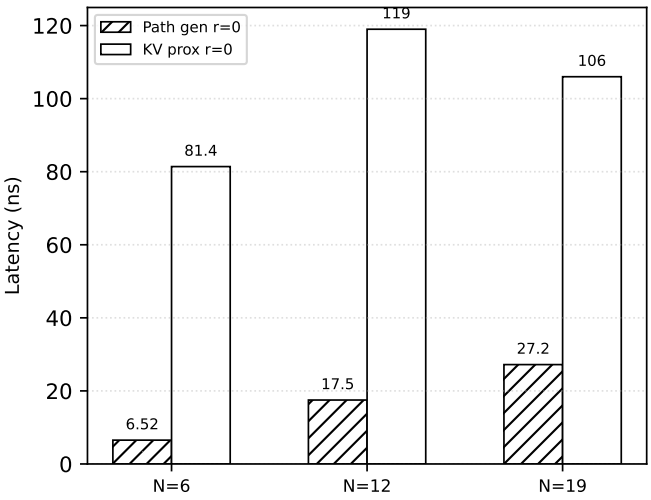

Figure 9: Large-N CoordCube capability

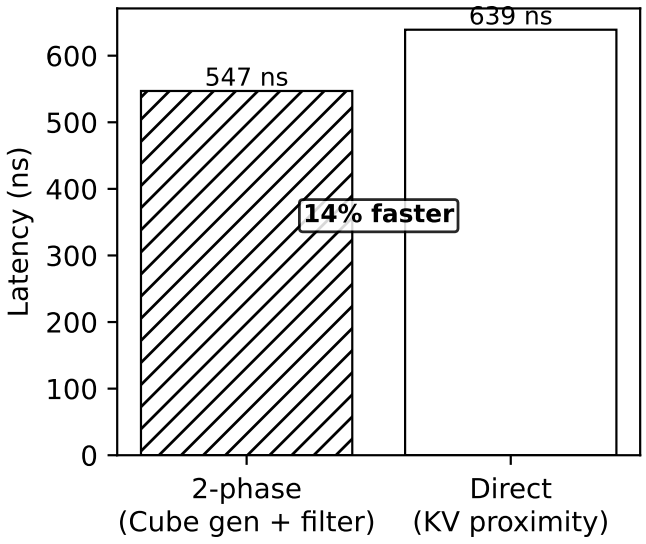

Figure 10: Hierarchical query: two-phase vs direct

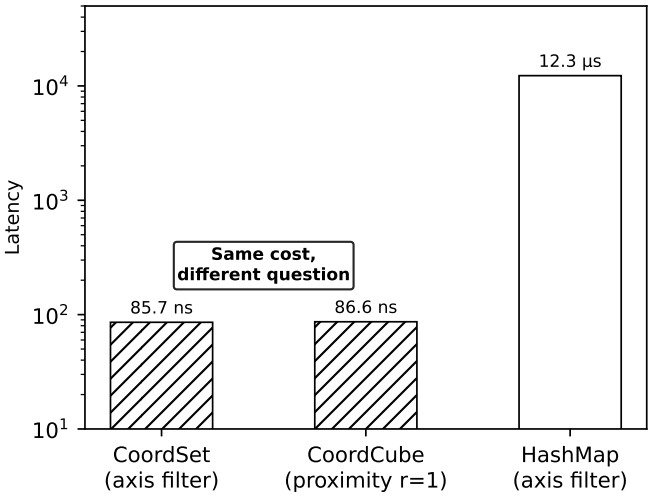

Figure 11: Spatial query strategies: CoordSet vs CoordCube