Adaptive Caching of Spatial Queries in PostGIS Using H3-Based Indexing

Mohammed Shakir
mohsha-0@student.ltu.se
Department of Computer Science, Electrical and Space Engineering

October 20, 2025

Abstract

x

Acknowledgements

x

Contents

Abstract i

Acknowledgements ii

Abbreviations & Acronyms v

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Motivation . . . 1
- 1.3 Problem Statement . . . 2
- 1.4 Research Questions . . . 2
- 1.5 Scope . . . 2
- 1.6 Contributions . . . 2
- 1.7 Thesis Structure . . . 3

2 Related Work 4
- 2.1 Hierarchical spatial indexing and pre-aggregation . . . 4
- 2.2 Adaptive caching for spatial workloads . . . 4
- 2.3 Cache consistency and spatially scoped invalidation . . . 5
- 2.4 Hexagonal grids and H3 . . . 5
- 2.5 Gap analysis . . . 5

3 Theory 6
- 3.1 Conceptual Frame . . . 6
  - 3.1.1 Query Models and Spatial Footprints . . . 6
  - 3.1.2 Hotspots and Workload Skew . . . 6
- 3.2 Spatial Indexing and Partitioning . . . 7
  - 3.2.1 Spatial Indexes in PostGIS . . . 7
  - 3.2.2 Grid-Based Bucketing vs. Native Indexing . . . 7
  - 3.2.3 Hexagonal Grids and H3 Basics . . . 7
  - 3.2.4 H3 Resolutions and Cell Geometry . . . 8
  - 3.2.5 Mapping BBoxes/Polygons to Cells . . . 8
  - 3.2.6 Spatial Approximation Limits of H3 . . . 8
- 3.3 Caching Theory for Spatial Workloads . . . 8
  - 3.3.1 Caching Objectives and Cost Model . . . 8
  - 3.3.2 Key Design for Spatial Queries . . . 8
  - 3.3.3 Admission and Replacement . . . 8
  - 3.3.4 TTL-Based Freshness Control . . . 8
  - 3.3.5 Event-Driven Invalidation . . . 8
- 3.4 Consistency and Freshness Semantics . . . 8
  - 3.4.1 Definitions: Consistency, Coherence, and Staleness . . . 8
  - 3.4.2 Spatial Validity Scopes . . . 8
  - 3.4.3 Combining TTL with Event-Driven Updates . . . 8
  - 3.4.4 Delivery Semantics and Ordering . . . 8
- 3.5 Performance and Latency Modeling . . . 8
  - 3.5.1 End-to-End Latency Decomposition . . . 8
  - 3.5.2 Queueing Basics and Little's Law . . . 8

3.5.3 Tail Latency and Percentiles 8
3.5.4 Hit Ratio vs. Latency/Throughput Trade-off 8
3.5.5 Granularity vs. Memory Cost (H3 Resolution) 8
3.5.6 Workload Skew and Hotspot Dynamics 8
3.5.7 Golden Signals and Cache Metrics 8

4 Method **9**

5 Results **10**

6 Discussion **11**

7 Conclusions **12**

References **13**

Abbreviations & Acronyms

x

1 Introduction

1.1 Background

Operational web mapping stacks at geospatial service providers typically combine GeoServer for OGC-compliant services with a PostGIS spatial database. Raster map tiles can be served via WMS/WMTS endpoints, which predefine tile matrices to improve the delivery of rendered images [11]. In contrast, vector feature access through WFS exposes fine-grained queries over feature properties and geometries, enabling filtering, editing, and analytics workflows that do not align cleanly with fixed tile boundaries [10]. On the database side, PostGIS provides spatial types and functions with GiST-based indexes that are important for performance [14]; yet when traffic is highly skewed toward a few urban "hotspot" regions, many requests repeatedly overlap in space and time, re-triggering similar queries and I/O.

Conventional server-side caching (e.g., tile caches) mostly targets rasterized responses; it is less effective for vector queries with complex geometries, attribute filters, or dynamic layers. A better approach is to partition the query footprint into a spatial grid and reuse results for frequently accessed cells [2]. Hexagonal hierarchical grids such as H3 provide near-uniform cell areas across resolutions, consistent neighbors, and identifiers that are convenient as cache keys and for aggregation across multiple zoom levels.

A spatial cache must also stay fresh because upstream data changes. Time-to-live (TTL) expiration gives a simple baseline, but event-driven invalidation, common in change-data-capture (CDC) pipelines built around streaming platforms, can invalidate only the affected spatial cells when features are updated. Combining TTL with event-driven invalidation improves freshness without discarding useful cache entries. Finally, an in-memory store such as Redis provides the fast access needed to reduce end-to-end latency on cache hits while keeping implementation complexity manageable within a middleware layer. This thesis builds on these observations to investigate an H3-keyed, Redis-backed cache between GeoServer and PostGIS, coupled with TTL and Kafka-driven invalidation to handle freshness under skewed spatial workloads.

1.2 Motivation

Workloads in operational web mapping services frequently show spatial skew: a small set of geographic areas attracts the majority of traffic, while large regions are queried infrequently. Recomputing overlapping vector queries in such hotspots increases latency and worsens database load even when spatial indexes are present. An H3-guided middleware cache provides three benefits:

1. Lower latency by serving popular regions from memory.
2. Lower PostGIS CPU and I/O by avoiding redundant evaluations.
3. Tunable trade-offs between hit ratio, memory footprint, and freshness by adjusting H3 resolution and invalidation policy.

From a research perspective, this work bridges ideas from hierarchical spatial indexing, adaptive caching, and spatially scoped invalidation into a deployable component that complements, rather than replaces, existing OGC services. The outcome should be practical recommendations on when adaptive caching is worthwhile, which resolutions to use, how to size memory, and how to integrate event-driven invalidation. The study provides an end-to-end evaluation of H3-keyed caching for dynamic vector workloads, a setting that is less well served by conventional tile caches and has few publicly documented, reproducible benchmarks.

1.3 Problem Statement

Modern spatial data platforms serve diverse read workloads through GeoServer and PostGIS. In practice, user traffic is highly skewed; a small set of geographic areas (e.g., city centers) receives a disproportionate share of requests, while large regions are rarely accessed [12]. Under this skew, repeated evaluation of overlapping spatial queries increases latency, drives unnecessary CPU and I/O load in PostGIS, and may slow down downstream map rendering. Conventional tile oriented caching helps for pre-rendered map tiles [11], but is less effective for dynamic vector queries, highly detailed filters [10], and requests that do not align with fixed tile boundaries [17]. As a result, the platform needs a cache that adapts to *where* and *how often* users query, while remaining correct and fresh when data changes.

The problem addressed in this thesis is to design, implement, and evaluate an *adaptive, spatially aware* caching layer that sits between GeoServer and PostGIS and uses an H3 hexagonal grid to detect hotspot regions from live query traffic, cache corresponding query results in Redis for low latency reuse, and keep cache entries acceptably fresh using a combination of time-to-live (TTL) and event driven invalidation from Kafka. The solution must reduce end-to-end query latency and database load under skewed workloads without introducing unacceptable staleness or excessive memory overhead. The study aims not only to derive actionable guidance on grid resolution, cache sizing, and invalidation policy, but also to deliver a prototype middleware that can be deployed in real production environments.

1.4 Research Questions

To evaluate the proposed approach, this thesis investigates the following research questions:

1. **Effectiveness:** To what extent does H3-based adaptive caching reduce latency and offload query pressure from PostGIS compared to no caching or a simple geometry/TTL cache baseline under skewed workloads?

2. **Granularity:** Which H3 resolution(s) give the best balance between cache hit ratio, memory footprint, and data freshness, given uneven access to different areas?

3. **Invalidation Policies:** What combination of time-to-live (TTL) and event-driven invalidation (triggered by upstream data change notifications) provides acceptable staleness bounds while keeping high performance?

1.5 Scope

The thesis develops and evaluates a middleware layer between GeoServer and PostGIS that encodes request footprints as H3 cells, caches vector query results in Redis, tracks hotness, and applies TTL and Kafka based invalidation. This includes building a working prototype that intercepts spatial requests and serves frequently accessed regions directly from Redis. Experiments will use containerized components and synthetic, skewed workloads representative of urban hotspots. The evaluation focuses on latency percentiles, database load, hit/miss ratios, staleness bounds, and cache memory usage across H3 resolutions, with an emphasis on quantifying the trade-offs among H3 resolution, cache size, hit ratio, and freshness.

1.6 Contributions

x

1.7 Thesis Structure

The thesis is organized as follows:

- **Introduction** outlines the background, motivation, problem statement, research questions, scope, contributions, and thesis structure.
- **Related Work** surveys prior research on spatial indexing, adaptive caching, and cache consistency for dynamic geodata.
- **Theory** introduces the theoretical background needed to understand H3 indexing, caching principles, and consistency mechanisms.
- **Method** describes the H3-based request bucketing, hotness tracking, cache invalidation policies, and the middleware design.
- **Results** present the experimental setup and report findings across baselines and different H3 resolutions.
- **Discussion** reflects on the results, limitations, and threats to validity and includes considerations of sustainability.
- **Conclusions** summarize the contributions of the thesis and outline directions for future improvements and deployment.

2 Related Work

This section frames the thesis in four strands of prior work:

- Hierarchical spatial indexing and pre-aggregation.
- Adaptive caching for spatial workloads.
- Cache consistency and invalidation with spatial scope.
- Hexagonal grid indexing (H3).

The goal is to motivate the chosen H3-guided middleware design and clarify how it differs from existing systems.

2.1 Hierarchical spatial indexing and pre-aggregation

A central idea behind the proposed middleware is to divide query footprints into grid cells so that repeated queries over similar regions can be identified and reused. Work on *GeoBlocks* pre-aggregates point data into fine-grained cells and approximates query polygons by unions of those cells; the approximation error is bounded by the chosen cell size, and a trie-like cache reuses previous results for frequently queried regions, adapting to skew over time [17]. Similarly, *Adaptive Cell Trie (ACT)* speeds up point-in-polygon joins by combining multiresolution (quadtree-style) approximations with a radix-tree over cell identifiers; the design avoids most expensive geometric tests while offering user controlled precision [3]. Earlier systems such as *Nanocubes* and the *aR-tree* introduced in-memory spatio-temporal aggregation over hierarchical rectangles; they offer interactive performance but rely on rectangular partitions that can cause unbounded error for arbitrary polygons unless cell size is carefully managed [4, 13]. Together, these works support a grid based approach that supports multiresolution refinement near boundaries, exposes compact cell identifiers for fast lookup and reuse, and adapts well under workload skew properties leveraged with H3 based bucketing.

2.2 Adaptive caching for spatial workloads

Beyond indexing, several systems adapt caching to hotspot access patterns. *STASH* is a distributed, in-memory middleware cache for hierarchical aggregations that chooses replicas based on query frequency and freshness, leading to large latency reductions and a higher query processing rate under skew [8]. *GeoBlocks* cache also adapts automatically as regions get popular [17]. At the opposite end of the stack, *SAC* (Semantic Adaptive Caching) predicts future query windows on the client using partitioned spatial history and prefetches answers to improve responsiveness [5]. Finally, *Vecstra* proposes an OGC-compliant in-memory geospatial cache that integrates caching with standard WFS/WCS interfaces, which is useful when the server side must remain standards based, as in GeoServer/PostGIS deployments [16]. Although not spatial per se, *MinMaxCache* shows how an adaptive in-memory cache can trade level of aggregation for interactive visualization latency with error, an idea that inspires the evaluation of H3 resolution choices and composition overheads in this thesis [7]. The common thread is that caching should be driven by query semantics, popularity, and acceptable approximation, rather than being a static tile store; the middleware follows this line by caching vector results keyed by H3 cells and adapting to observed hotness.

2.3 Cache consistency and spatially scoped invalidation

Keeping cached results fresh is not easy when data and query scopes are spatial. Classic mobile computing studies introduced *location dependent* invalidation: cached answers can become stale not only because the data change but also because the user moves outside a valid geographic scope [18]. This led to approaches like *Bit-Vector* with *Compression and Implicit Scope Information*, which link cached items directly to their spatial validity regions [18]. Related work on *location dependent semantic caching* also organizes cache segments with explicit spatial scopes and uses them in replacement and validation [6]. The practical takeaway for a server side cache is to tie each cached entry to an explicit spatial footprint and to invalidate it on time based expiry (TTL) and spatial overlap with changed features. The proposed design does exactly this: H3 cell keys define validity scopes, while TTL and Kafka-driven events provide temporal and event based invalidation hooks.

2.4 Hexagonal grids and H3

H3 is a hierarchical, global grid of hexagonal cells. Its properties include near uniform area per resolution, consistent adjacency, and a clean hierarchical relation between resolutions that make it attractive for bucketing and composing spatial answers [15]. Empirical reports outside pure research also suggest that hexagons reduce visual and aggregation artifacts compared to rectangular geohashes for many analytics tasks [1]. For our use case, H3 gives a compact key that is independent of programming language (suitable for Redis), deterministic up/down-sampling across resolutions for composition, and a natural way to map data change events to affected cells for invalidation.

2.5 Gap analysis

To the best of current knowledge, none of the above combines, in a single end-to-end system, real time, *schema agnostic* hotspot detection external to the database via H3 bucketing over arbitrary WFS/WMS requests, *adaptive* caching of vector query results at the middleware layer with measured trade-offs across grid resolutions, and *event driven* freshness via streaming updates (Kafka) that target exactly the overlapping H3 cells. *Index and cache* systems either pre-computes specific analytics (GeoBlocks, STASH), focus on client centric prediction (SAC), or discuss standards compliant caching without fine-grained spatial invalidation (Vecstra). This thesis fills that gap by implementing and evaluating an H3-keyed Redis cache between GeoServer and PostGIS, and by quantifying how resolution, hit ratio, memory footprint, and invalidation policy interact under skewed workloads.

3 Theory

This section introduces the concepts that the middleware relies on. Things like how vector queries are expressed and scoped, why real workloads concentrate on a few "hot" places, and how spatial indexes in PostGIS accelerate predicates yet leave room for complementary caching. The aim is to fix terminology and assumptions so later design and evaluation choices (e.g., H3 cell resolution, cache keys, and freshness policies) can be understood and compared consistently.

3.1 Conceptual Frame

3.1.1 Query Models and Spatial Footprints

Vector web services expose two distinct dimensions of query semantics: a *spatial filter* over geometries and an optional *attribute filter* over feature properties. In OGC-style services, the spatial part ranges from coarse bounding boxes (BBOX) to polygonal predicates such as `Intersects`, `Within`, or `Contains`, while the attribute part uses comparison and logical operators (e.g., `POP_DENSITY > 5000 AND TYPE = 'residential'`). Filter Encoding (FES) defines how these spatial, comparison, and logical operators are combined into a single query expression, in either XML or ECQL form [9].

We use spatial footprint to denote the minimal geometry that defines the spatial filter as sent by the client (e.g., the BBOX or a user-drawn polygon). For caching theory, the footprint is mapped to a distinct set of spatial buckets (H3 cells) that act as reusable containers. A *spatial response* is the set (or stream) of features matching both filters within the query area, together with any server-side transformations. Reuse then follows two patterns:

- Full reuse: When a new request's cell set and attribute filter exactly match a cached entry.
- Composed reuse: When the new request spans multiple cached cell entries whose combination can be merged and then filtered.

In both cases, the cache key must encode (layer, cell or cell-set, filter hash, SRID) to preserve identical query meaning with the database result. Freshness is controlled independently via TTL or event-driven invalidation (discussed later), but the scope of what to invalidate remains the same, which is the cells touched by the updated features.

3.1.2 Hotspots and Workload Skew

Interactive map traffic is rarely uniform across space. Request popularity typically follows a heavy-tailed (Zipf-like) distribution, where a small number of locations attract a large fraction of requests, while most locations are cold. This pattern has been observed broadly in networked systems and motivates cache designs that selectively retain popular items. In spatial systems, the same pattern appears as *spatial skew*: urban centres, transport corridors, and touristic areas become hotspots, and the set of hotspots may evolve over time with events and daily cycles [12].

Skew has two practical consequences. First, repeated evaluation of overlapping queries in hotspots can dominate query latency and backend load, even when individual queries are optimized with spatial indexes [17]. Second, any effective cache should be *spatially selective*, meaning that it should spend memory on the small subset of cells where reuse is high, rather than caching uniformly across the entire map. Dividing footprints into H3 cells makes it possible to track per-cell popularity, admit only "hot" cells, and combine results for larger queries without caching redundant geometry everywhere.

3.2 Spatial Indexing and Partitioning

3.2.1 Spatial Indexes in PostGIS

PostGIS optimizes spatial queries using GiST-based indexes that implement an R-tree-like search over geometry bounding boxes. The index performs a fast filter step (bounding-box test) to filter out candidates, followed by an exact geometry check (`ST_Intersects`, `ST_Within`, etc.) on the survivors [14]. Correct use therefore requires both creating the index with `USING GIST` and using index-aware filters so the planner can apply the filter step.

While GiST dramatically reduces the cost of single queries and spatial joins, it does not take advantage of *temporal locality* or *overlap reuse*. If many users repeatedly query the same small region, the database still performs planning, index descent, and exact checks for each request; buffers, caches, and prepared geometries help, but they are not keyed by *where* reuse occurs. An external, cell-keyed cache complements GiST by avoiding the entire evaluation pipeline for popular regions. Once a cell's result is computed, later requests that map to the same (cell, filter) can be answered from memory. In short, GiST reduces the work to find candidates; spatial caching reduces how often the database must do that work at all.

3.2.2 Grid-Based Bucketing vs. Native Indexing

Native database indexing (e.g., GiST-backed R-trees in PostGIS) speeds up single queries by filtering out candidates with a fast bounding-box filter and then applying exact geometry filters to the survivors. This *two-pass* evaluation is highly effective per request, but it does not directly capture *temporal locality* across requests that repeatedly target the same small area: the planner, index descent, and exact checks still run for every query.

An external, grid-based bucketing strategy divides space into fixed cells and uses those cells as cache keys for query answers. In this model, a request's footprint (BBOX or polygon) is mapped to a set of cells; if the (layer, cell-set, filter) tuple is already cached, the middleware composes the response from memory, bypassing the database. Overlapping requests then naturally reuse prior work because they share cells. The database remains responsible for correctness and misses, while the middleware focuses on detecting and serving hotspots. In short, native indexing reduces the amount of work within a query, while grid bucketing reduces how often the database must do that work at all by enabling answer reuse across partially overlapping requests. The two techniques are complementary, GiST handles precise geometry tests and complex joins, while grid bucketing takes advantage of spatial clustering to minimize repeated work in hotspot regions.

3.2.3 Hexagonal Grids and H3 Basics

H3 is a hierarchical discrete global grid that represents locations as 64-bit cell identifiers. The grid is organized in resolutions ranging from 0 to 15, where each cell has exactly one parent at any lower resolution and up to seven children at the next higher resolution; parent/child navigation is provided by API functions such as `cellToParent` and `cellToChildren`. Adjacency is based on shared edges; neighborhood and vicinity queries use grid traversal operations (e.g., `kRing`, `hexRange`) to list cells within a given graph distance.

At resolution 0, the grid contains 122 base cells (derived from an icosahedral projection); refinements add hexagonal children, with pentagons appearing to maintain topology. These properties (stable adjacency, clean up/down sampling, and compact integer keys) make H3 a practical foundation for bucketing requests and composing answers across neighboring cells in a middleware cache.

3.2.4 H3 Resolutions and Cell Geometry

Cell size decreases exponentially with resolution. Empirically, average hexagon areas are on the order of $5.16\,\mathrm{km}^2$ at res 7, $0.737\,\mathrm{km}^2$ at res 8, and $0.105\,\mathrm{km}^2$ at res 9 (global averages; exact area varies with position relative to the icosahedron). The minimum/maximum area ranges per resolution reflect distortion near icosahedron vertices and the presence of pentagons.

These statistics drive practical trade-offs. Higher resolutions improve cache specificity (higher chance that an incoming query is covered by already-hot cells and less overfetch when composing answers) but increase (i) the number of keys per query, (ii) metadata/memory overhead per cached entry, and (iii) composition cost when merging many cell payloads. Coarser resolutions reduce key churn and memory but risk overestimation (returning many features outside the exact footprint) and lower hit probability for small, detailed queries. Our evaluation therefore sweeps a small set of candidate resolutions and reports hit ratio, composition overhead, and memory per resolution, grounded in the documented cell-area curves.

3.2.5 Mapping BBoxes/Polygons to Cells

3.2.6 Spatial Approximation Limits of H3

3.3 Caching Theory for Spatial Workloads

3.3.1 Caching Objectives and Cost Model

3.3.2 Key Design for Spatial Queries

3.3.3 Admission and Replacement

3.3.4 TTL-Based Freshness Control

3.3.5 Event-Driven Invalidation

3.4 Consistency and Freshness Semantics

3.4.1 Definitions: Consistency, Coherence, and Staleness

3.4.2 Spatial Validity Scopes

3.4.3 Combining TTL with Event-Driven Updates

3.4.4 Delivery Semantics and Ordering

3.5 Performance and Latency Modeling

3.5.1 End-to-End Latency Decomposition

3.5.2 Queueing Basics and Little's Law

3.5.3 Tail Latency and Percentiles

3.5.4 Hit Ratio vs. Latency/Throughput Trade-off

3.5.5 Granularity vs. Memory Cost (H3 Resolution)

3.5.6 Workload Skew and Hotspot Dynamics

3.5.7 Golden Signals and Cache Metrics

4 Method

x

5 Results

x

6 Discussion

x

7 Conclusions

x

References

[1] Nicolas Drapier, Aladine Chetouani, and Aurélien Chateigner. "Enhancing Maritime Trajectory Forecasting via H3 Index and Causal Language Modelling (CLM)". In: *Transactions on Machine Learning Research* (2024). DOI: 10.48550/ARXIV.2405.09596. URL: https://arxiv.org/abs/2405.09596.

[2] GeoSolutions. *Tile Caching with GeoWebCache*. GeoServer Training (official documentation). 2020. URL: https://docs.geoserver.geo-solutions.it/edu/en/enterprise/gwc.html.

[3] Andreas Kipf et al. *Adaptive Main-Memory Indexing for High-Performance Point-Polygon Joins*. en. 2020. DOI: 10.5441/002/EDBT.2020.31. URL: https://openproceedings.org/2020/conf/edbt/paper_190.pdf.

[4] Lauro Lins, James T. Klosowski, and Carlos Scheidegger. "Nanocubes for Real-Time Exploration of Spatiotemporal Datasets". In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (Dec. 2013), pp. 2456–2465. ISSN: 1077-2626. DOI: 10.1109/tvcg.2013.179. URL: http://dx.doi.org/10.1109/TVCG.2013.179.

[5] Chang Liu, Brendan C. Fruin, and Hanan Samet. "SAC: semantic adaptive caching for spatial mobile applications". In: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL'13. ACM, Nov. 2013, pp. 174–183. DOI: 10.1145/2525314.2525366. URL: http://dx.doi.org/10.1145/2525314.2525366.

[6] Heloíse Manica, M. A. R. Dantas, and Murilo S. de Camargo. "An Architecture for Location-Dependent Semantic Cache Management". In: *Proceedings of the Seventh International Conference on Enterprise Information Systems*. SciTePress - Science, 2005, pp. 320–325. DOI: 10.5220/0002524903200325. URL: http://dx.doi.org/10.5220/0002524903200325.

[7] Stavros Maroulis et al. "Visualization-Aware Time Series Min-Max Caching with Error Bound Guarantees". In: *Proceedings of the VLDB Endowment* 17.8 (Apr. 2024), pp. 2091–2103. ISSN: 2150-8097. DOI: 10.14778/3659437.3659460. URL: http://dx.doi.org/10.14778/3659437.3659460.

[8] Saptashwa Mitra et al. "STASH: Fast Hierarchical Aggregation Queries for Effective Visual Spatiotemporal Explorations". In: *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Sept. 2019, pp. 1–11. DOI: 10.1109/cluster.2019.8891029. URL: http://dx.doi.org/10.1109/CLUSTER.2019.8891029.

[9] Open Geospatial Consortium. *OGC Filter Encoding 2.0 Encoding Standard*. Tech. rep. OGC 09-026r2. Version 2.0, Accessed: 2025-09-28. Open Geospatial Consortium, 2011.

[10] Open Geospatial Consortium. *Web Feature Service (WFS) Standard*. https://www.ogc.org/standards/wfs/. Accessed: 2025-09-28. 2025.

[11] Open Geospatial Consortium. *Web Map Tile Service (WMTS) Standard*. https://www.ogc.org/standards/wmts/. Accessed: 2025-09-28. 2025.

[12] Shaoming Pan et al. "A Global User-Driven Model for Tile Prefetching in Web Geographical Information Systems". In: *PLOS ONE* 12.1 (Jan. 2017). Ed. by Houbing Song, e0170195. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0170195. URL: http://dx.doi.org/10.1371/journal.pone.0170195.

[13] Dimitris Papadias et al. “Efficient OLAP Operations in Spatial Data Warehouses”. In: *Advances in Spatial and Temporal Databases*. Springer Berlin Heidelberg, 2001, pp. 443–459. ISBN: 9783540477242. DOI: 10.1007/3-540-47724-1_23. URL: http://dx.doi.org/10.1007/3-540-47724-1_23.

[14] Paul Ramsey, Mark Leslie, and PostGIS contributors. *Spatial Indexing — Introduction to PostGIS*. https://postgis.net/workshops/postgis-intro/indexing.html. Accessed: 2025-09-29. 2012–2023.

[15] Uber Technologies, Inc. *H3: A Hexagonal Hierarchical Spatial Index*. https://h3geo.org/docs/. Accessed: 2025-09-29. 2025.

[16] Yiwen Wang. “Vecstra: An Efficient and Scalable Geo-spatial In-Memory Cache”. In: *Proceedings of the VLDB 2018 Ph.D. Workshop, August 27, 2018, Rio de Janeiro, Brazil*. CEUR Workshop Proceedings, 2018. URL: https://ceur-ws.org/Vol-2175/paper06.pdf.

[17] Christian Winter et al. “GeoBlocks: A Query-Cache Accelerated Data Structure for Spatial Aggregation over Polygons”. en. In: *Proceedings of the 24th International Conference on Extending Database Technology (EDBT)*. OpenProceedings.org, 2021. DOI: 10.5441/002/EDBT.2021.16. URL: https://openproceedings.org/2021/conf/edbt/p78.pdf.

[18] Jianliang Xu, Xueyan Tang, and Dik Lun Lee. “Performance analysis of location-dependent cache invalidation schemes for mobile environments”. In: *IEEE Transactions on Knowledge and Data Engineering* 15.2 (Mar. 2003), pp. 474–488. ISSN: 1041-4347. DOI: 10.1109/tkde.2003.1185846. URL: http://dx.doi.org/10.1109/TKDE.2003.1185846.