# Why Are Geospatial Databases So Hard To Build?

02 Mar 2015

In 2005, I decided to design my first geospatial database engine. PostGIS was unable to keep up with the modest volumes of sensor data I was working with and while I was new to geospatial data, I had skill designing database engines. It seemed simple enough to get the scale and performance I required. Nothing could have been further from the truth and the reasons were rarely obvious.

Most software engineers assume today, as I did a decade ago, that the design of scalable geospatial databases is a straightforward task. As it turned out, in 2005 the computer science required to design such things did not even exist. Typical methods for representing dynamic geospatial data, from R-trees to "hyperdimensional hashing" to space-filling curves were all invented in the 1980s or earlier. If you could build scalable geospatial databases that way people would have; I incorrectly thought no one had tried.

Geospatial databases, at a basic computer science and implementation level, are unrelated to more conventional databases. The surface similarities hide myriad design challenges that are specific to spatial data models. All of the architectural differences below are lessons I learned the hard way, manifested as critical defects in real-world applications. You can think of it as a checklist for "is my geospatial database going to fail me at an inconvenient moment".

## Computer Science Does Not Understand Interval Data Types

Algorithms in computer science, with rare exception, leverage properties unique to one-dimensional scalar data models. In other words, data types you can abstractly represent as an integer. Even when scalar data types are multidimensional, they can often be mapped to one dimension. This works well, as the majority of data people care about can be represented with scalar types.

If your data model is inherently non-scalar, you enter an algorithm wasteland in the computer science literature. Paths, vectors, polygons, and other elementary

aggregations of scalar coordinates used in spatial analysis are non-scalar data types. Computational relationships are topological instead of graph-like.

Spatial data types, among a few other common data types, are *interval data types*. An interval data type cannot be represented with less than two scalar values of arbitrary dimensionality, like the boundary of a hyper-rectangle. These differ from scalar types in two important ways: sets have no meaningful linearization and intersection relationships are not equivalent to equality relationships. The algorithms that do exist in literature for interval data are poor.

- **There are no dynamic sharding algorithms that produce uniform distributions of interval data**. In fact, you can show that a general partitioning function that uniformly distributes *n*-dimensional interval data in *n*-dimensional space does not even exist. Hash and range partitioning, which in this context is essentially a distributed Quad-Tree, are demonstrably poor choices but still popular in naive implementations.

- **Interval indexing algorithms in literature are either not scalable or not general**. Ironically, the literature describes hundreds of algorithms. The R-Tree family of algorithms, the traditional choice for small data sets, cannot scale even in theory. The Quad-Tree family of algorithms are pathological for interval data; R-Trees were invented to replace Quad-Trees for this reason. Sophisticated variants of grid indexing ("tiling") can scale for static interval data but fail for dynamic data and most software engineers use even more naive variants in practice.

- **Interval data sets have no exploitable order**. The assumption that data is sortable is so pervasive in computer science that many design patterns either have no benefits or are incorrect when applied to interval data. For example, what is the best storage format for an analytical geospatial database? The answer is not obvious. Many storage formats, such as those in columnar databases, make tradeoffs that assume the sortability of scalar data types.

## Database Engines Cannot Handle Real-Time Geospatial

Most geospatial databases were built for creating maps. As in, geospatial data models that can be rendered as image tiles or paper products. Mapping databases evolved in an environment where the data sets were small, rarely changed, and production of a finished output could take days to complete.

Modern spatial applications are increasingly built around real-time spatial analysis and contextualization of data from IoT, sensor networks, and mobile platforms. These workloads look nothing like making maps. In fact, these workloads are unlike any workload studied in database literature. There is no existing design to copy that can support this use case. A database engine optimized for a modern geospatial workload must be designed and implemented from first principles, which requires unusual levels of skill.

- **Many spatial data sources continuously generate data at extremely high rates**. The Twitter firehose is a trickle compared to many high-value data sources that must be spatially organized. Not only do you need to parse, index, and store complex spatial data at rates of petabytes per day, but this must be concurrent with low-latency queries against incoming and historical data that cannot be summarized. In-memory databases are useless here; the volume of online data is too large. While technically feasible, virtually no databases were designed for this workload.

- **Traditional storage, execution, and I/O scheduler designs assume a simple ordered log or tree structure**. This reflects the typical organization of scalar data models. Complex multidimensional data traversals appear irregular and semi-random to traditional database engines, leading to suboptimal operation scheduling that greatly reduces throughput. Sophisticated scheduler designs for spatial relationship traversals do not exist like they do for relational databases.

- **Geospatial workloads tend to be highly skewed and constantly shifting over the data model in unpredictable ways**. Traditional skew and hotspot mitigation strategies based on *a priori* assumptions about workload distribution that can be hardcoded into software, such as those used to deal with time-series data, are largely useless for geospatial. Adaptive mitigation of severe and unpredictable hotspotting is a novel architectural requirement.

## Correct, Fast Computational Geometry Is Really Hard

Geospatial operators are inherently built around computational geometry primitives. Looking up a description of the Vincenty algorithm is easy. Implementing non-Euclidean operators that are correct, precise, and fast in the general case is beyond most programmers' ken (and mine). Implementations designed for cartographic and GIS use cases typically lack the performance and precision required for analytics.

Analytics requires producing answers in a timeframe that matters with tiny and precisely characterized error bounds. In practice, performance is often so poor that commercial vendors tout analytic results that are delivered in weeks for data sets that could fit in memory. Reliably producing correct and precise computational results from complex geometry operations is notoriously difficult to achieve, particularly at scale. This may be good enough for making maps but is nearly useless for modern sensor analysis applications.

- **The physical world is non-Euclidean**. The curvature of Earth's gravity field is approximately 8 centimeters per kilometer. This produces horizon effects even within a single large building. The simplest geometric surface that still applies to most calculations is a geodetic ellipsoid. The computationally simpler approximations used under the hood by many popular platforms work for maps but not for spatial analytics.

- **Geospatial analytics requires the ability to do polygon intersections quickly**. Polygon intersection algorithms, like relational joins, are quadratic in nature. Naively intersecting polygons with thousands of vertices, which are common in some industries, may require millions of high-precision geodetic operations to evaluate one record. Multiply that by billions of records and your query may not complete this month. This often shows up as an unintentional denial-of-service attack on poorly engineered geospatial databases.

- **Computational geometry has to be precise if you care about analysis**. For trivial algebraic geometry it is not difficult to guarantee the error bounds on floating point computations. Most programmers do not realize that the transcendental functions implemented in their CPUs exhibit significant losses of precision in a variety of edge cases that must be accounted for. At the scale of these data sets, the edge cases are tested quickly and frequently.

## Difficult But Not Impossible

Any discussion of how to solve one of these problems is a long blog post on its own but suffice it to say for now that these are all solvable with sufficiently clever computer science and competent implementation. However, these problems are not trivial and some have solutions that are not in the public computer science literature.

With the emergence of mobile, IoT, and sensor analytics as high-value markets, every database platform is scrambling to “enable” spatial analytics. It is extremely difficult to add adequate support for high-performance spatial analytics to databases that were not purpose-built for that use case. Many database companies have tried and failed.

From a database architecture standpoint, the root of the problem is that the internals of a database engine designed for traditional “text and numbers” data models, which is virtually all of them, has very little in common with a database engine designed for real-time spatial data models. No amount of software duct tape circumvents this impedance mismatch yet this is how most people attempt to build a geospatial database. That is how I tried to build a geospatial database and why it took me years to actually build one.

Geospatial databases are challenging to build in the best case. They are nigh impossible to build if you naively assume they are essentially the same as more conventional databases with a little bit of geospatial sprinkled on top.

## Related Posts

### SpaceCurve: An Utterly Unique And Absurdly Fast Geospatial Database 08 Oct 2015

### MetroHash: Faster, Better Hash Functions 27 May 2015