Efficient 3D and 4D Geospatial Indexing in RDF Stores with a Focus on Moving Objects

Jans Aasman[1], Steve Haflich[1]

[1] Franz Inc., 2201 Broadway, Suite 715, Oakland, CA 94612
ja@franz.com, smh@franz.com

Abstract: We first explain AllegroGraph's existing capabilities for 2D geospatial indexing for proximity queries. We explain our general position on geospatial encoding and serialization. We describe our current work extending to 3D and 4D to support moving objects (MOBs) with efficient cursors on MOB tracks, and posit five classes of MOB queries.

Keywords: RDF Database, Geospatial 2D 3D 4D, moving objects, scalability

1 Introduction

Combining geospatial and temporal reasoning in a single query framework seems an important capability for the Semantic Web in general and for the intelligence community in particular (see Aasman,[1]). If you want to reason about events that happen in time and space you need capabilities that can deal with them together.

We see an active interest in geospatial computing in the Semantic Web community. A prominent example is the Linked Data initiative that has the Geonames database as one of the central data sources pointed to by many other data sources. The geospatial community also shows a growing interest in RDF and semantic technologies. The National Map project being one of the examples where we see how the experts in the field are working to develop ontologies that will allow better use of the information in various data sources.

But curiously, there is hardly any interest in dealing with time in the Semantic Web community. The W3C site shows no activity and the temporal reasoning community rarely considers RDF.

It follows that there is also not much interest apparent in the combination of geospatial and temporal reasoning in one computational framework. Recently in January of 2009, the National Science Foundation held a symposium, with leaders in the field, to gage whether there is interest in the academic arena to develop research projects in this area.

The AllegroGraph RDF database has supported basic temporal reasoning and 2D geospatial querying for two years, but recently we engaged projects that pushed the representation requirements in unexpected ways. One project deals with AIS vesseltracking data and the other deals with GPS tracking data for cell phone users. The current separate facilities for 2D geospatial indexing and temporal indexing could not in combination provide sufficient efficiency to deal with moving objects (MOBs) on a large scale. Consequently, we started a research project for direct indexing of 3D and 4D data. This paper reports some of our findings and ongoing development.

2 Some Observations About Data Representation

We have several observations about data representation. It is useful to separate the issues of internal representation from external representation(s), i.e. serialization formats.

It is frequently the practice, probably inherited from relational databases (RDB), of storing longitude, latitude, optionally altitude and time, each as a separate triple. These correspond to the separate columns in a RDB. We show below that this is grossly inefficient for locality queries, and can result in O(n^2) or worse performance. Our experience is that longitude/latitude and longitude/latitude/time (etc.) should be encoded and indexed internally as a single RDF quantity. (It is essentially never the case that one wants to retrieve or search any one of these data without retrieving them all together.) We have devised and will describe an indexing scheme that supports locality search with near-linear O(n) time in the number of entries in the result set.

If geo or geo/time data should be combined as a single datum internally, they should also remain a single datum when externalized, e.g. in N-triples format. RDF serialization places no requirements on triple ordering, and if a latitude and longitude pair were to became hugely separated in a serialization it could be pernicious to have to recombine them during deserialization. Therefore we make the following position statement: There should be a standard RDF type for externalized geo data that allows geo and geo/time entry to be represented as a single lexical datum. As a specific straw proposal, externalization for geo position should be something like ISO6709, and externalization of a MOB datum should be something like the concatenation of an ISO6709 string with an ISO8601 string (details to be worked out).

3 How We Do Geospatial in AllegroGraph

This current section will explain the principles that underlie our 2D, 3D, and 4D indexing. The section following will discuss a series of increasingly difficult queries on an RDF based MOB database. This taxonomy of queries is useful both in query implementation and in predicting performance.

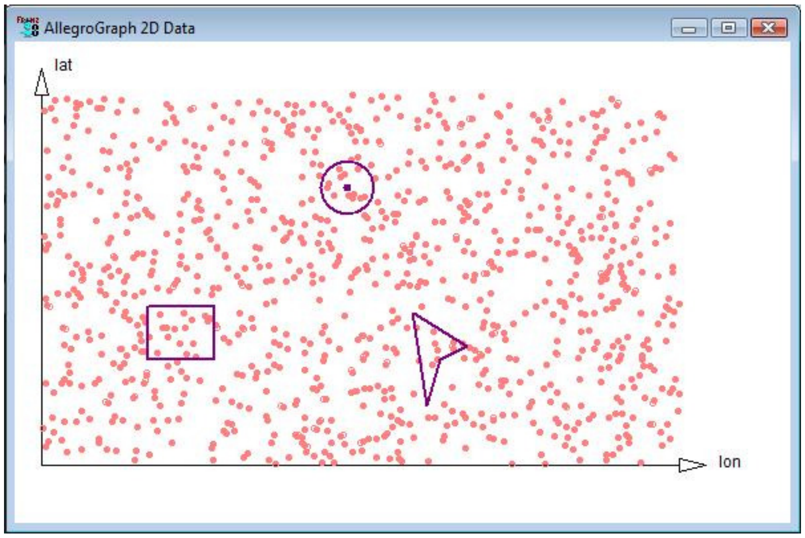
AllegroGraph is both an RDF triple store and a quad graph store. It was designed to be hugely scalable (beyond core size). The four parts of each triple [sic] SPOG can hold any kind of data. All parts are efficiently linearly sortable. Along with string resources and literals, efficient specialized part encodings are supported: machine numerical types (fixed and float) as well as other specialized types. The encoded types generally sort in the natural order of the encoding.

Computer main memory and disk are linearly addressable vectors. It is well known that a vector of length n can be sorted in O(n log n) time. Searched in O(log n) time. AllegroGraph is designed to exploit machine speed, despite scaling requirements, by keeping everything linear. AllegroGraph maintains multiple sorted indexes (e.g. SPOG, POSG, GOSP) and by selecting the proper index, triples variously related to others can be retrieved from a local region of that index.

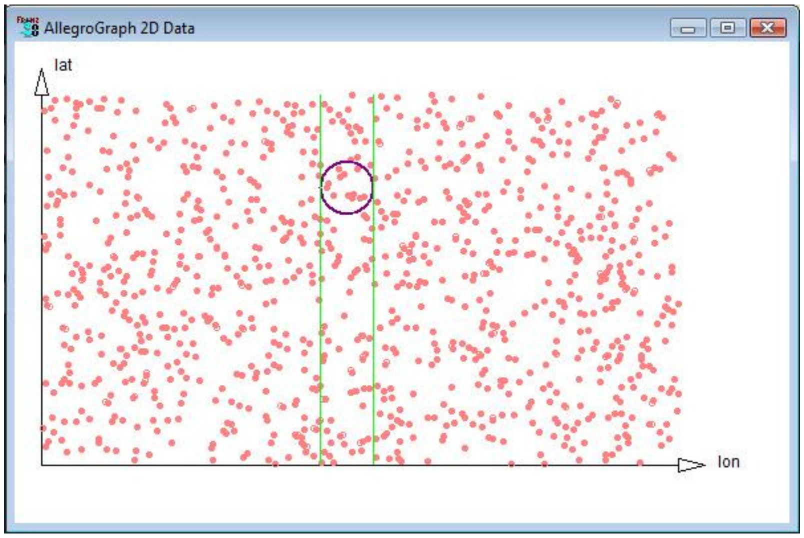
If a user wants to retrieve everything about <http://franz.com/employees#Jans>, all triples with this Subject are sorted together in the SPOG index. All triples with Jans as the Object are together in the OSPG index. All triples with a particular Predicate e.g. <http://franz.com/employees#isSupervisorOf> are grouped together in the POSG index, sorted secondarily on Object. And so on...

Age, date and/or time, currency, phone numbers, stock prices, license-plate numbers, and barometric pressure are all linearly orderable. Cartesian and spherical (e.g. geospatial) coordinates in two or higher dimensions are not immediately orderable and sortable. How to integrate these into the AllegroGraph model?

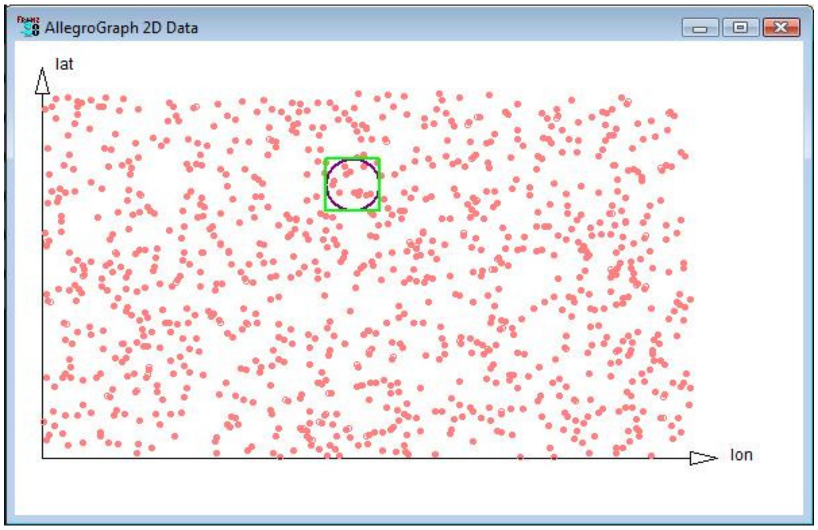
The important problem is proximity search. We want to optimize speed retrieving all triples with coordinates in a certain locality.



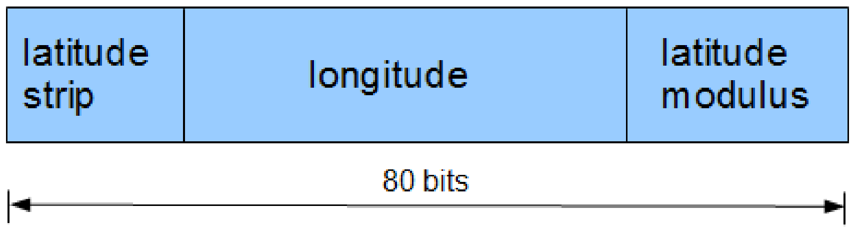
Data in two dimensions could be sorted on the two separate dimensions in the obvious way, either first on Y/latitude or X/longitude, or the reverse. But this causes search time over a locality to increase linearly with the size of the data set. Locality search increases with the product of the search width and the number of items in the dataset, i.e. O(n^2).

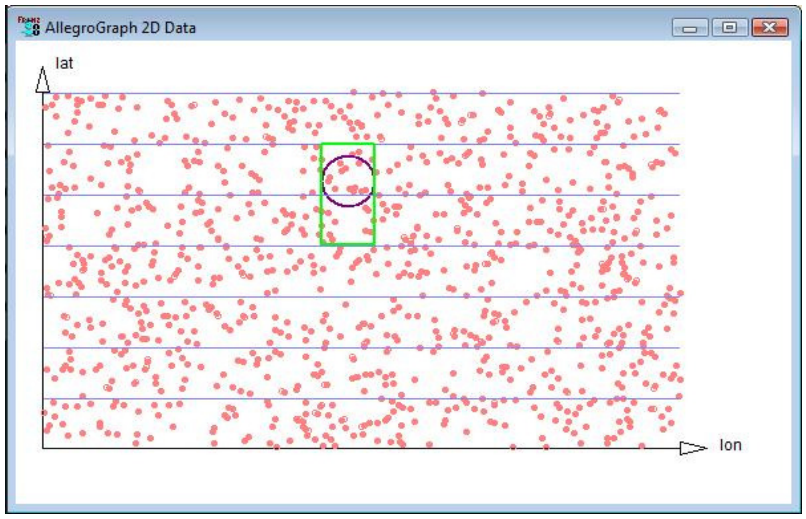


We'd much rather just search the region of interest, reasonably bounded in two dimensions instead of just one.



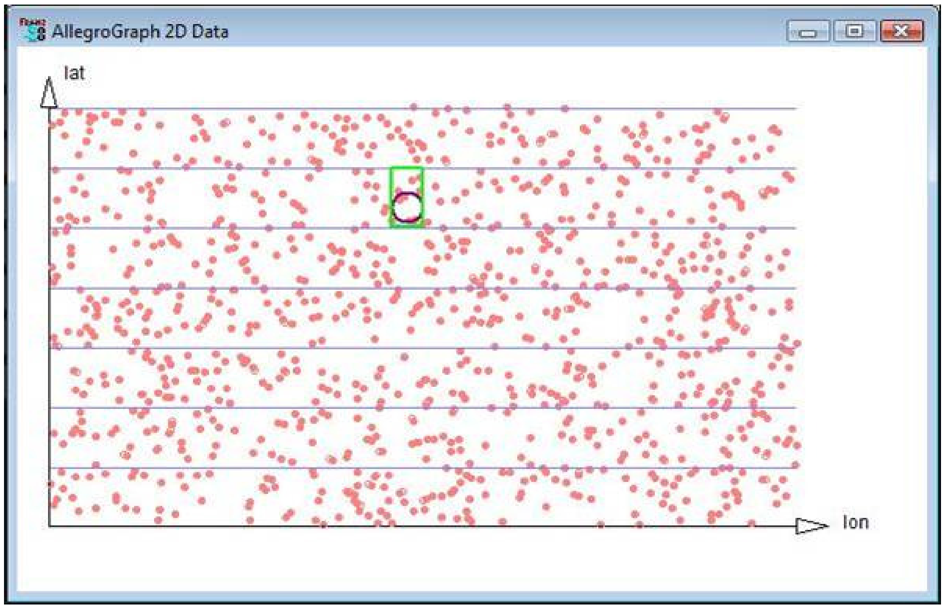
Within the requirements of AllegroGraph's fundamentally linear indexing, how can we avoid this unfortunate scaling? R-trees and numerous other schemes support very efficient search of localities. But if all the data won't fit in memory, paging performance can be unpredictable and data management can be convoluted. There is no obviously efficient way to reconcile 2-D and higher R-trees with the AllegroGraph linear indexing. But suppose we knew a little more about how we will use our data, specifically: The approximate size of typical regions to be searched. We can sort the data into strips.



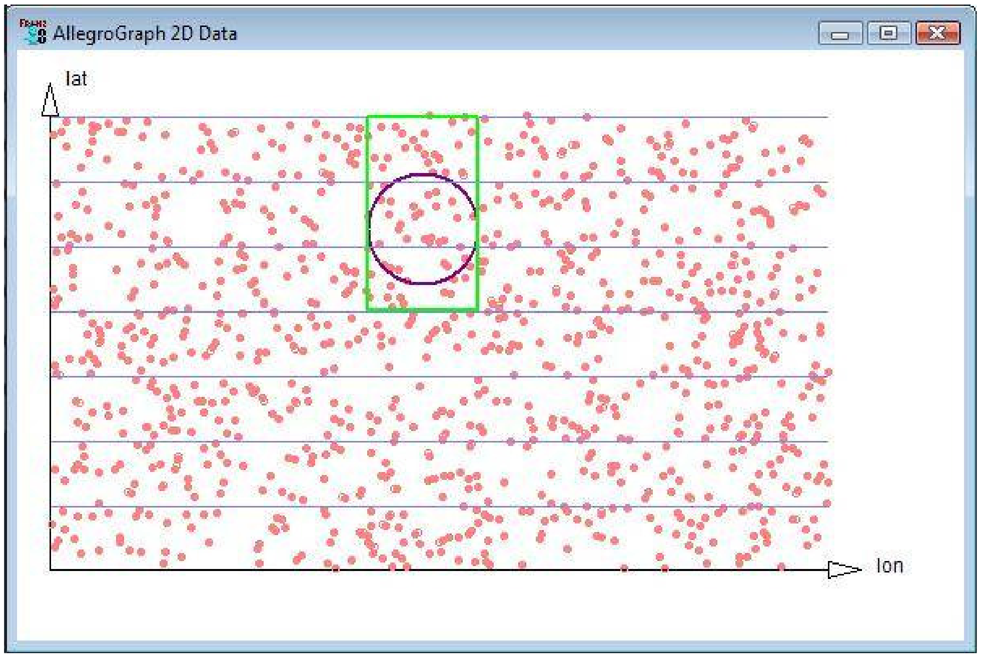


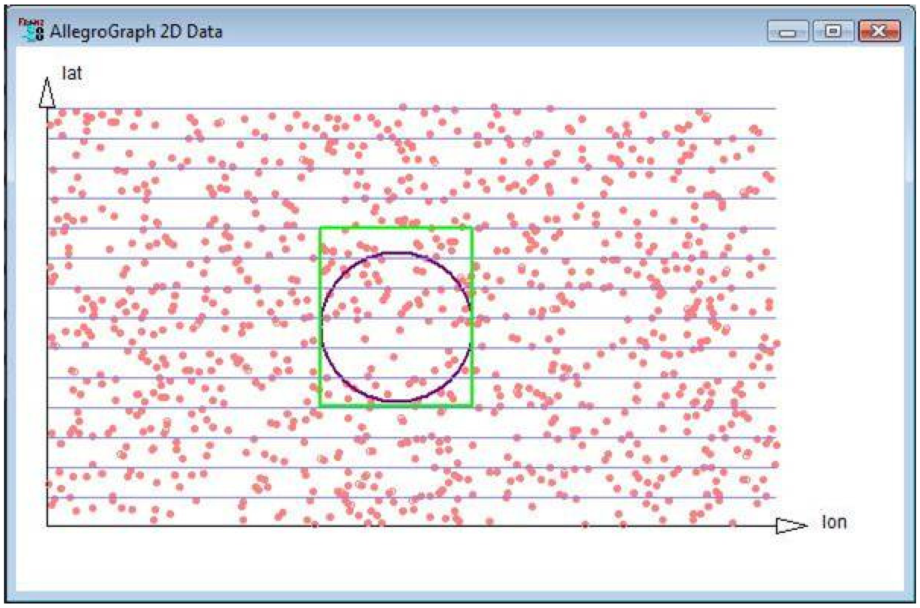
In detail, the coordinates are converted to unsigned integers. The major sort ordinate is split into strip and modulus within the strip. This is merely integer division with remainder. (All this can be thought of as a variation on the technique of space-filling curves.) If the search diameter is exactly the same as the strip width, we need traverse short linear regions of just two strips. For circular region, the number of data traversed is only 4/pi the size of the result set. AllegroGraph implements cursors as the mechanism for stepping through a range of data. A specialized class of cursor called a concatenated cursor can step though a set of linear segments of the data, e.g. the regions of the two strips above.

If the search diameter is smaller than the strip width, short regions of only one or two strips need be traversed. Efficiency stays high, falling off roughly linearly in the size of the mis-estimate in the estimated search diameter.



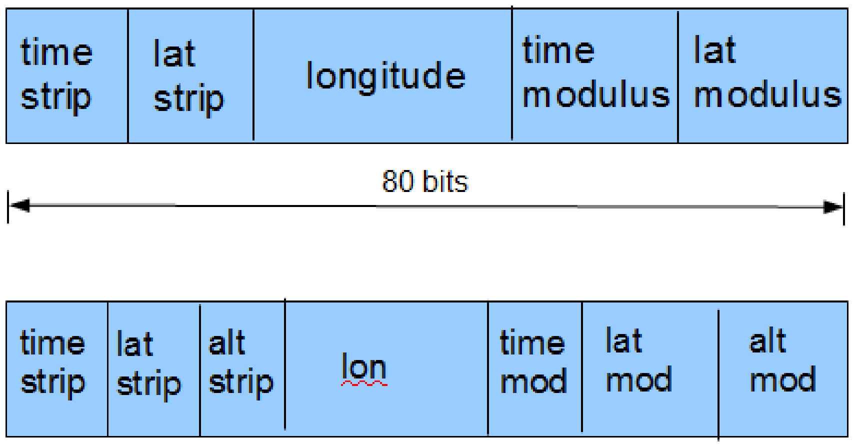
If the search diameter is somewhat larger than the strip width, the ratio between the number of data that need be traversed and the size of the result set stays fairly constant, but the number of separate linear strips that must be addressed and seeked increases about linearly with the error in estimate. With an increasing in the number of separate linear regions that must be traversed there is of course an additional cost, but performance stays reasonable even with fairly large estimation errors.

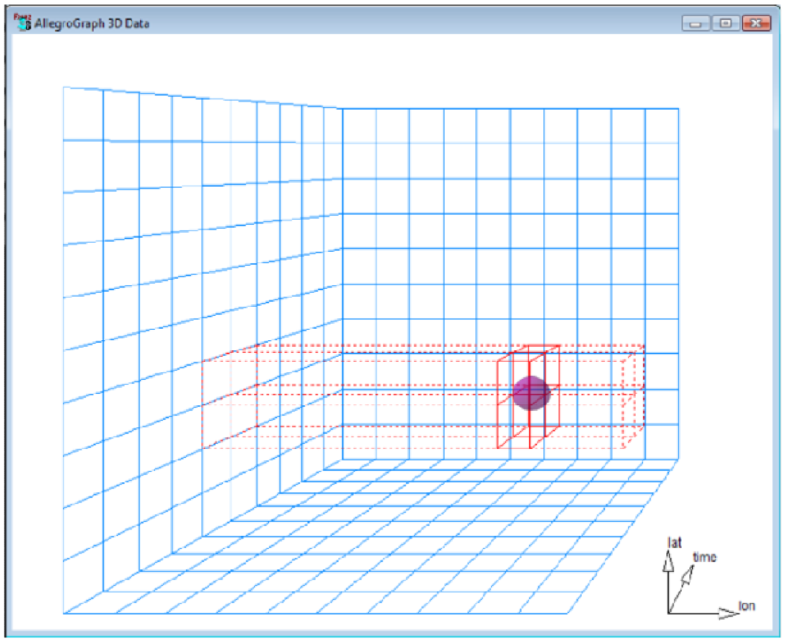




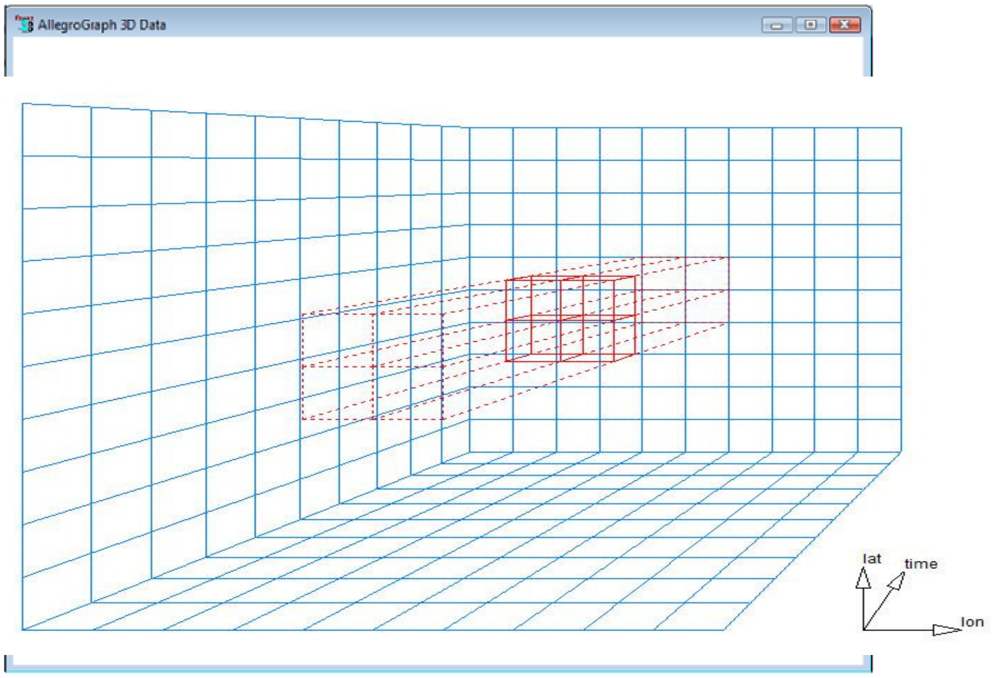
To summarize our 2D implementation, it may be inconvenient to need to specify strip width in advance. But performance is still reasonable even with an order of magnitude error. In addition, if extremely different strip sizes are needed, the data can be stored twice with different strip sizes. The two coordinates can be anything: pressure and temperature distance and time. This last possibility suggests extension to 3-D and beyond, particularly MOB data encoding paths in latitude/longitude/time.

The 2D approach extends naturally to 3D and 4D, except that instead of string data in strips we store it in prisms.





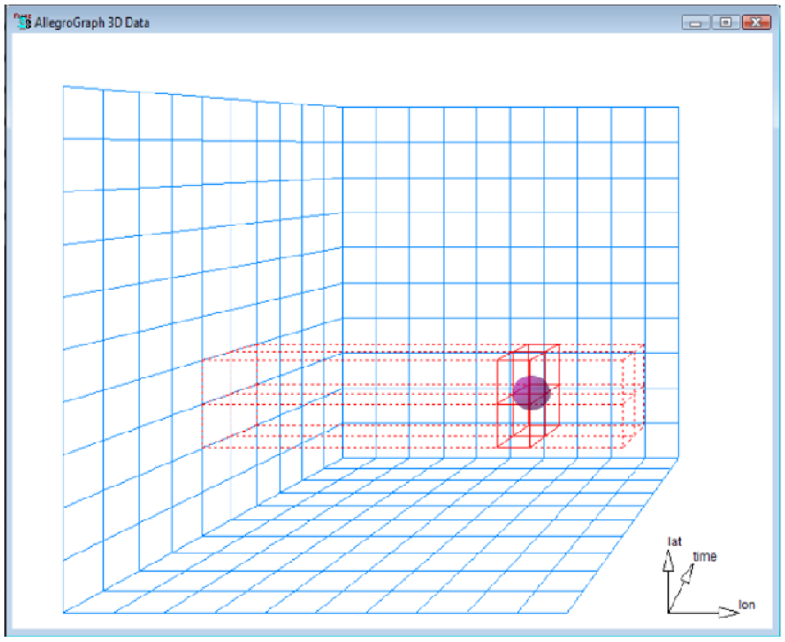
Erratum: The prisms in the above and all remaining 3D drawings are mis-oriented and should have been drawn with the prisms aligned with the time axis, as in this single corrected sample below. They will be redrawn for the final camera copy!



4 On the Difficulty of Various Classes of Moving Object Queries

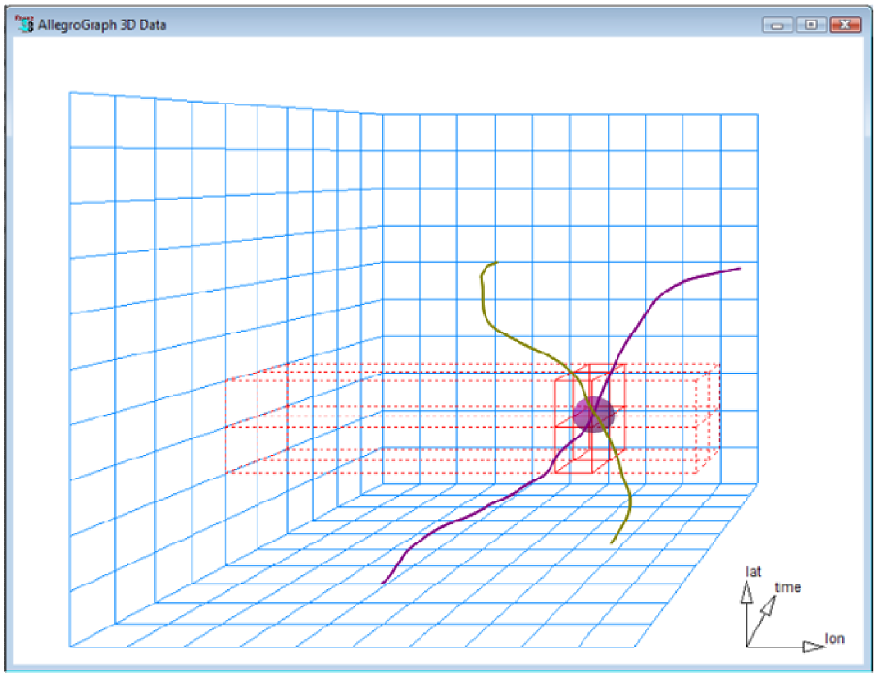
This paper concludes with an informal illustration of five classes of MOB queries. These different kinds of queries have different degrees of difficulty.

I. Simple proximity search.
This has already been covered above. Find all 3D data with a certain proximity (or bounding box or other solid) of a given point.



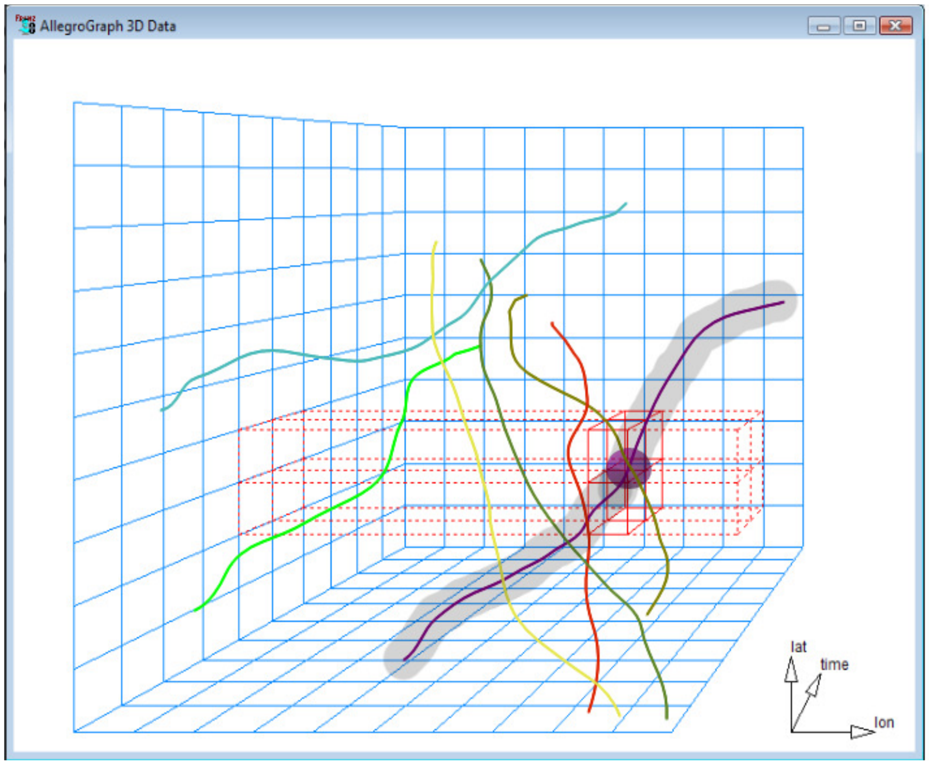
II. Given two particular MOB tracks, determine if and when they were ever within a certain lat/lon/time distance.

This requires following the two tracks in parallel through time, repeatedly checking distance. The several prisms containing the regions around each track occur in local linear regions of the SPOG index. A special kind of cursor is implemented which can traverse through time, adding or removing prisms as the path shifts in longitude and latitude. But the two cursors need traverse only the data for the two given MOBs. Any proximate data will necessarily be adjacent in one of several adjacent prisms.


AllegroGraph 3D Data
lat
time
lon

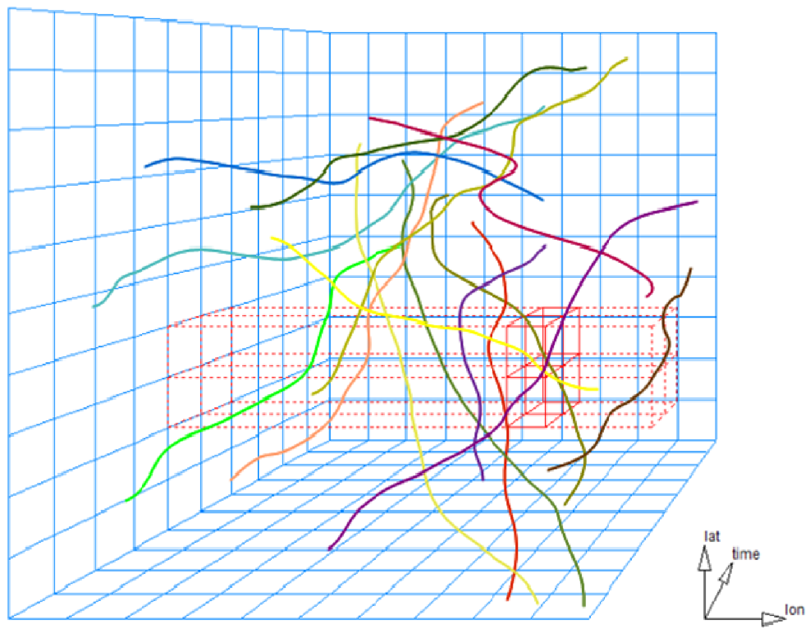
III. Given a single MB, detect any other MOBs that ever come within a certain proximity.

This is somewhat similar to the above, requiring traversal of that single MOB path and detecting other MOBs in proximity. Any such MOB data will be in located within the same several adjacent prisms in the track being followed. Therefore, this search still requires only traversing the portion of the total data set along a single track.

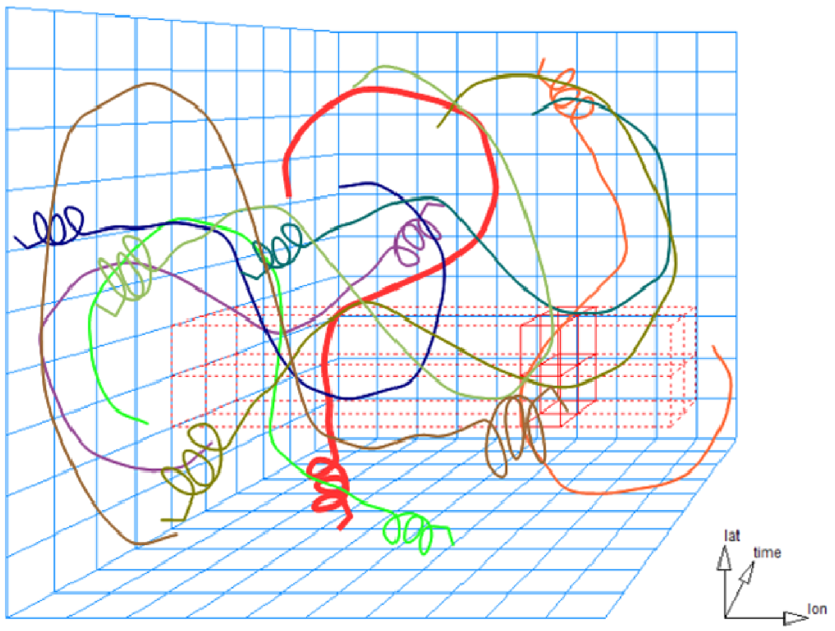


IV. Find all occurrences of any two MOBs within a certain proximity.

This clearly requires a traversal through the entirety of the MOB data using a slightly different kind of cursor. However, since MOB positions within a given proximity must be within a local region proximate within one of several adjacent prisms, the scan can be done with a single traversal through the data, stepping in parallel through some small number of adjacent prisms.



V. Detect potential <u>Social Network Cliques</u> between unknown MOBs, e.g. as evidenced by MOBs repeatedly being proximate in Place and Time, or being repeatedly suspiciously proximate in Place at different Times.



We do not know how to solve this, because we do not quite know what we would be looking for. It is a real research problem, hence (for now) the visual pun.

References

1. Aasman, J.: Unification of Geospatial Reasoning, Temporal Logic, & Social Network Analysis in Event-based Systems, Distributed Event Based Systems (DEBS 2008) http://portal.acm.org/citation.cfm?id=1386007