The Changing Problems, Databases, and Tools in Spatial Search

W. RANDOLPH FRANKLIN
Professor, Department of Electrical, Computing, and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY
Email: mail@wrfranklin.org

With

MARCUS V. A. ANDRADE
Federal University of Viçosa
Brazil
Email: marcus.ufv@gmail.com

This paper presents a view of the changing nature of search occasioned by larger databases and new software and hardware tools.

The problems that we want to solve are changing. Simple queries of a database to find those records r satisfying some predicate P(r) are easy. The more interesting queries involve coincidences. Find all pairs of records r,s such that P(r,s) holds. For example, search a set of aircraft track logs for near collisions.

Searching algorithms' data access patterns haven't changed much. Read operations are common, while write operations are rare, except when building a search structure in a preprocessing phase. Also, except while preprocessing, there is little interaction between the database elements.

What is new is that searching larger databases requires greater logical rigor during any computations. Unlikely errors from a sloppy algorithm design are now not so unlikely as the number of operations grows. An example would be a numerical roundoff error causing a topological error such as a point to be on the wrong side of a line.

The available hardware is changing. For all hardware, whether CPU or GPU, computation is becoming much cheaper than reading or writing. This applies to data in memory, in addition to data on secondary storage. Data compression becomes more useful. Transforming and computing with the data is often free.

Two complementary classes of hardware have recently become affordable: general multicore Intel CPUs and NVidia GPU accelerators. (We cite Intel and NVidia because they are leaders this year, but in a few years they might not even exist.) Problems that required servers are now doable on workstations.

Intel CPU workstations can now support very large main memory and multiple processors and cores. Even laptops can have 32GB of main memory. Imagine having a workstation with 1TB of main memory available for your database. Any record from that database is retrievable in roughly constant time (ignoring caches). That changes the appropriate search data structure.

Binary trees, quadtrees and octrees are silly. A B-tree with the root having 1,000,000 children, or a 1000x1000 uniform grid, is much faster. A massive hash table might be optimal.

That massive main memory is accessible by multiple CPU cores, each with multithreading. The author's 2012-vintage lab workstation is a dual 8-core Intel Xeon, with each core supporting 2 threads, for a total of 32 threads, and 128GB of main memory (expandable to 256GB). That is programmable by adding OpenMP directives to a properly structured C++ program. The directives instruct the system as to what operations can be performed in parallel. Loops where the different iterations do not depend on each other are common candidates. Reduction operations, such as summing over an array, also parallelize in OpenMP.

An example of a well parallelizable search operation is out Union3 algorithm and implementation. The input is a set of overlapping congruent cubes in E3. The output is the volume of their union. One search operation is to find all sets of three faces that intersect. Another is to determine which points from a set are not inside any of the input cubes. When processing 100,000,000 cubes, Union3 executes 10 times faster on 32 threads than on one. (In that case, of the 600M-choose-3 triples of faces, 395M intersected.)

Running a CUDA program on an NVidia GPU accelerator such as their newest model, the K40, is a more adventuresome solution, but with plusses and minuses. Massive parallelism is possible with the K40's 2880 CUDA cores. However: efficiently CUDA programming is complicated. The amount of available memory is very small—only 12GB. That memory has 3 levels of caching. The register level, about 100x faster than most of the K40's memory, has only 256KB per thread block. Data that has to be retrieved from the CPU's memory is even slower. The result is that one CUDA core is perhaps only 5% as fast as one Intel CPU core.

GPU programming also strongly prefers simple regular data structures, and code where each thread follows the same execution path on data that is adjacent to the data accessed by the previous thread. Complicated algorithms and structures with recursion and pointers are deprecated. Elevation DEMS are better than TINs. This theme of the optimal solution being simple has occurred many times before, e.g., with hash functions and page replacement algorithms.

Because the physics of the current technology limits processor speeds to about what they are now, whether with CPUs or with GPUs, parallel programming is necessary for searching large databases.

(Switching a transistor is like charging a capacitor. Reducing the switching time requires increasing the voltage or making the transistor smaller. The former is infeasible; instead voltages get smaller to reduce power. The latter (counterintuitively) increases the transistor's resistance, which also requires more power. That runs up against the limits of how quickly heat can be removed from a small circuit.)

The available software is changing. Exactness may seem to be a pointless luxury in GIS applications, since GIS data is by its nature approximate, and approximate results are sufficient for all practical purposes. However, most geometric algorithms used in GIS, such as point location and map overlay, become much more complex and prone to failure if their elementary

operations are subject to rounding errors, no matter how small. Consider for example a distributed application that cuts a map into smaller submaps, handles each piece to a separate processor, and combines the partial results into a single map. If the cutting step is exact, the final step needs only to identify common boundary edges between the partial results, and remove them. The task becomes much harder if the cutting step is affected by rounding errors: the partial results may overlap, or may be separated by gaps. The pasting operation is then almost impossible to specify, let alone to implement.

The well-founded solution to roundoff errors in geometric computing is to compute in a number domain that does not suffer roundoff. That would be the rational numbers. Each number is composed of a numerator and a denominator, each of which is an integer with as many digits as necessary (implemented as an array of C++ ints). gmp++ is one library that implements rational computation. Computing with rationals is slower than computing with floats; however see the earlier comments about computation cost.

Searching and working with curves as curves, instead of as a polyline approximation, is starting to look like perhaps becoming feasible. CGAL (Computational Geometry Algorithms Library) is an example of a very large package for more complete geometric computing. The intersection point of two curves can be determined as the root of a polynomial equation. (Just as common variables can be eliminated from a set of linear equations, so can a set of polynomial equations be solved by techniques such as resultants.) The problem is that this polynomial is of a very high degree. A bicubic parametric patch, the lowest degree that allows matching curvature when two patches join, has degree 18 when considered as an implicit function. Intersecting two produces a curve of degree 324. This is an area of continued research.

These powerful software tools are now feasible only because of the newly affordable powerful hardware. They come at the right time to facilitate higher-order searches in the new massive databases.