Map Overlay Area Animation and Parallel Simulation

Wm Randolph Franklin

ECSE Dept., 6026 JEC
Rensselaer Polytechnic Institute
Troy, New York 12180–3590, USA
+1 (518) 276–6077
Internet: wrf@ecse.rpi.edu, Fax: (518) 276–6261

January 20, 1993

1 Introduction

A map, for purposes of this video, is a planar graph with *vertices*, *edges*, and *faces* or *polygons*. Each vertex is given by its location in E^2, and each edge is given by the names of the two vertices and two adjacent polygons (or faces). All the consecutive 2-vertices and edges between two 3-vertices may be called a *chain*. Examples of maps include the coterminous states of the USA, and 10° Fahrenheit isotherms.

Overlaying two maps to produce a new map whose polygons are all the intersections of polygons of either input map is one of the most difficult problems in GIS. There are large databases with numerical degeneracies. Minor numerical roundoffs can cause global topological errors in the output.

One major application of map overlay is to interpolate data from one map to another. Let polygon $\mathcal{P}_i$ of map 1 have population p_i. Assume that we want the population of polygon $\mathcal{Q}_j$ of map 2. Let the area of $\mathcal{Q}_j$ intersected with each p_i be a_{ij}, and let the area of each $\mathcal{P}_i$ be $a_{i.}$. Then a reasonable approximation to the population of $\mathcal{Q}_j$ is

$$\sum_i \frac{a_{ij}}{a_{i.}} p_i$$

Note that here we need only the areas of the overlay polygons, not the polygons themselves.

In [1] I presented `overlay.c`, an implementation of my algorithm for finding the overlay areas, which is faster and more robust than finding the overlay polygons themselves. It is about 3000 lines of C code. Wu[7] has done overlaying with rational numbers in Prolog, and Sun[6] and Sivaswami[5] have looked at related issues. One parallel implementation is Hopkins[3].

This video animates and illustrates that algorithm. The video recorded the realtime results of a `xanim.c`, a 1600 line C++ program running on a 25 MHz Sun IPC. All the graphics was low-level `Xlib` calls. The recording was from the video-out of the monitor to a 3/4 inch Umatic VTR. Much of the displayed data, such as the intersection locations, and numbers of edges in each cell, were obtained by running `overlay.c`, instrumented to write the data for `xanim.c` to read.

The edge length histogram shows that edges of widely varying lengths are handled. The unit is pixels. Indeed, some edges are so much longer than average that they must be truncated to see the main part of the histogram. The edges-per-cell histogram, while not perfectly even, is more even than the original data. This is true at several different grid resolutions. The algorithm can handle this unevenness.

The parallel data are partly simulated and partly real. The Single Instruction Multiple Data Stream (SIMD) model was chosen for much of this since it is so restrictive, and any algorithm doing well on a SIMD will do well anywhere. Each of ten processores takes time proportional to the number of relevant

cells to calculate which cells that edge passes through. When all processors are finished, each takes another edge. Thus the isolated long edges are bad, but still the overall estimated efficiency is good.

Simulated parallel determination of edge intersection in cells is similar. Each processor takes a cell, and spends time proportional to the number of pairs of edges to test them. When all processors are done, each takes another cell. Under this restrictive model, the efficiency is somewhat slower. Therefore another model was also simulated, where each processor takes 10% of the cells and processes them. A processor starts another cell as soon as it finishes the last. This is easier to do on a Multiple Instruction Multiple Data Stream (MIMD) machine.

Real parallel implementations on a CM-2 Connection Machine, Intel 32 processor hypercomputer, and 16 processor Sequent Balance, of finding all edge intersections, and similar problems, are described in Chandrasekhar[4] and Kankanhalli[2].

The humongous example at the end of the video processed 2 maps: US counties and hydrographic polygons. The former has 55068 vertices, 46116 edges, 2985 faces, and 8941 chains. The latter has 76215 vertices, 69835 edges, 2075 faces, and 6380 chains. The execution time in CPU seconds of each part of `overlay.c` is as follows.

Read map	99.32	Intersect edges	6.50
Scale vertices	1.03	Locate map 0 points in map 1	5.83
Extract edges from chains	2.38	Locate map 1 points in map 0	8.17
Calculate input polygon areas	3.65	Accumulate output areas	14.35
Make grid	1.28	Print areas	17.23
Add map to grid	8.60	TOTAL TIME	168.35

That is, reading the (ASCII formatted) data was slower than everything else combined. This is not surprising since each database, stored as an ASCII file, has at least 30K lines, or 2M bytes.

References

[1] W. R. Franklin. Calculating map overlay polygon' areas without explicitly calculating the polygons — implementation. In *4th International Symposium on Spatial Data Handling*, pages 151–160, Zürich, 23-27 July 1990.

[2] W. R. Franklin and M. Kankanhalli. Parallel object-space hidden surface removal. In *Proceedings of SIGGRAPH'90 (Dallas, Texas) in Computer Graphics*, volume 24, August 1990.

[3] S. Hopkins and R. G. Healey. A parallel implementation of Franklin's uniform grid technique for line intersection detection on a large transputer array. In K. Brassel and H. Kishimoto, editors, *4th International Symposium on Spatial Data Handling*, pages 95–104, Zürich, 23-27 July 1990.

[4] C. Narayanaswami and W. R. Franklin. Determination of mass properties of polygonal CSG objects in parallel. *International Journal on Computational Geometry and Applications*, 1(4), 1992.

[5] V. Sivaswami. Point inclusion testing in polygons and point location in planar graphs using the uniform grid technique. Master's thesis, Rensselaer Polytechnic Institute, Electrical, Computer, and Systems Engineering Dept., May 1990.

[6] D. Sun. Implementation of a fast map overlay system in c. Master's thesis, Rensselaer Polytechnic Institute, Electrical, Computer, and Systems Engineering Dept., May 1989.

[7] P. Y. Wu and W. R. Franklin. A logic programming approach to cartographic map overlay. *Canadian Computational Intelligence Journal*, 6(2):61–70, 1990.

January 20, 1993 ~/c/animation/p.tex