AUTO CARTO 9
Ninth International Symposium on
Computer-Assisted Cartography
Baltimore, Maryland
April 2 — 7 1989

UNIFORM GRIDS: A TECHNIQUE FOR INTERSECTION DETECTION ON SERIAL AND PARALLEL MACHINES

Wm. Randolph Franklin
Chandrasekhar Narayanaswami
Mohan Kankanhalli
David Sun
Meng-Chu Zhou
Peter YF Wu

Electrical, Computer, and Systems Engineering Dept.,
6026 J.E.C.,
Rensselaer Polytechnic Institute,
Troy, NY, 12180, USA,
(518) 276-6077,
Internet: Franklin@CS.RPI.EDU, Bitnet: WRFRANKL@RPITSMTS,
Fax: (518) 276-6003, Telex: 6716050 RPI TROU.

ABSTRACT

Data structures which accurately determine spatial and topological relationships in large databases are crucial to future developments in automated cartography. The uniform grid technique presented here offers an efficient solution for intersection detection, which is the key issue in many problems including map overlay. Databases from cartography, VLSI, and graphics with up to 1 million edges are used. 1,819,064 edges were processed to find 6,941,110 intersections in 178 seconds on a Sun 4/280 workstation. This data structure is also ideally suited for implementation on a parallel machine. When executing on a 16 processor Sequent Balance 21000, total times averaged ten times faster than when using only one processor. Finding all 81,373 intersections in a 62,045 edge database took only 28 seconds elapsed time. These techniques also appear applicable to massively parallel SIMD (Single Instruction, Multiple Data Stream) computers. We have also used these techniques to implement a prototype map overlay system and performed preliminary tests on overlaying 2 copies of US state boundaries, with 3660 edges in total. Finding all the intersections, given the edges in memory, took only 1.73 seconds on a Sun 4/280. We estimate that the complete overlay would take under 20 seconds.

INTRODUCTION

Algorithms specific to polyline intersection are particularly important for cartographic purposes. The classic problem of map overlay is a good example where edge intersection forms the core of the algorithm. The results of this paper are also useful in diverse disciplines such as graphics and VLSI design.

We are given thousands or millions of small edges, very few of which intersect, and must determine the pairs of them that do intersect. Clearly, a quadratic algorithm comparing all $\binom{N}{2}$ pairs is not acceptable.

Useful line intersection algorithms often use sweep line techniques, such as in Nievergelt and Preparata (1982), and Preparata and Shamos (1985). Chazelle and Edelsbrunner (1988) have an algorithm that finds all K intersections of N edges in time $T = \theta(K+N\log N)$. This method is optimal in the worst case, and is so fast that it

pages 100-109

cannot even sort the output intersections. However, this method has some limitations. First, it cannot find all the red-blue intersections in a set of red and blue edges without finding (or already knowing) all the red-red and blue-blue intersections. Second, it is inherently sequential.

Alternative data structures, based on hierarchical methods such as quadtrees, have also been used extensively, Samet (1984). They are intuitively reasonable data structures to use since they subdivide to spend more time on the complicated regions of the scene. An informal criticism of their overuse in Geographic Information Systems in given in Waugh (1986). A good general reference on cartographic data structures is Peucker (1975).

Since cartographers deal with vast amounts of data, the speed and efficiency of the algorithms are of utmost importance. With the advent of parallel and supercomputers, efficient parallel algorithms which are simple enough to implement, are gaining importance. Since this field is relatively new, few implementable algorithms exist. Some of the related parallel algorithms in computational geometry are as follows. Akl (1985) describes some parallel convex hull algorithms. Evans and Mai (1985) and Stojmenovic and Evans (1987) present parallel algorithms for convex hulls; however they require a MIMD machine, and have tested on only a few processors. Aggarwal et al (1985) give parallel algorithms for several problems, such as convex hulls and Voronoi diagrams. They assume a CREW PRAM (concurrent read exclusive write, parallel random access machine). This is a MIMD model. No mention is made of implementation. Although it is not mentioned in those papers, randomized algorithms, such as described by Clarkson (1988a), and Clarkson and Shor (1988b) appear to lend themselves to parallelization sometimes. Yap (1987) considers general questions of parallelism and computational geometry. Hu and Foley (1985), Reif and Sen (1988), and Kaplan and Greenberg (1979) consider hidden surface removal. Scan conversion is considered by Fiume, Fournier, and Rudolph (1983). For realistic image synthesis see Dippe and Swensen (1984).

This paper concentrates on an alternative data structure, *the uniform grid*. Here, a flat, non-hierarchical grid is superimposed on the data. The grid adapts to the data since the number of grid cells, or resolution, is a function of some statistic of the input data, such as average edge length. Each edge is entered into a list for each cell that it passes through. Then, in each cell, the edges in that cell are tested against each other for intersection. The grid is completely regular and is not finer in the denser regions of the data.

The uniform grid (in our use) was first presented in Franklin (1978) and was later expanded by Franklin, Akman, and Wu (1980), (1981), (1982), (1983), (1984), (1985), (1987), and Wu(1988) . The latter two papers used extended precision rational numbers and Prolog to implement map overlay. Geometric entities and relationships are represented in Prolog facts and algorithms are encoded in Prolog rules to perform data processing. Multiple precision rational arithmetic is used to calculate geometric intersections exactly and therefore properly identify all special cases of tangent conditions for proper handling. Thus topological consistency is guaranteed and complete stability in the computation of overlay is achieved.

In these papers the uniform grid was called an *adaptive* grid. However, there is another, independent and unrelated, use of the term adaptive grid in numerical analysis in the iterative solution of partial differential equations. Our papers present an expected linear time object space hidden surface algorithm that processed 10,000 random spheres packed ten deep in 383 seconds on a Prime 500. The idea was extended to a fast haloed line algorithm that was tested on 11,000 edges. The concept was applied to other problems such as point containment in polygon testing. Finally it was used, in Prolog and with multiple precision rational numbers in the map overlay problem in cartography.

This present paper presents experimental evidence that the uniform grid is an efficient means of finding intersections between edges in real world data. The uniform grid is similar to a quadtree is the same sense that a relational database schema is similar to a

hierarchical schema. The power of relational databases, derived from their simplicity and regularity, is also becoming apparent.

The uniform grid data structure is also ideally suited to execution on a parallel machine because of the simple data structures. Also, it is more numerically robust than sweep-line algorithms that have problems. This is of importance in the cartographic domain because numerical instability can easily introduce topological inconsistencies which tend to be difficult to rectify.

The uniform grid technique is fairly general and can be used on a variety of geometric problems such as computing Voronoi diagrams, convex hull determination, Boolean combinations of polygons, etc.

INTERSECTION ALGORITHM

Assume that we have N edges of length L independently and identically distributed (i.i.d.) in a 1×1 screen. We place a $G \times G$ grid over the screen. Thus each grid cell is of size $\frac{1}{G} \times \frac{1}{G}$. The grid cells partition the screen without any overlaps or omissions. The intersection algorithm proceeds as follows.

1. For each edge, determine which cells it passes through and write ordered pairs *(cell number, edge number)*.

2. Sort the list of ordered pairs by the cell number and collect the numbers of all the edges that pass through each cell.

3. For each cell, compare all the edges in it, pair by pair, to test for intersections. If the edges are *a priori* known to be either *vertical* or *horizontal*, the vertical edges are compared with the horizontal edges only. To determine if a pair of edges intersect, we test each edge's endpoints against the equation of the other edge. We ignore calculated intersections that fall outside the current cell. This handles the case of some pair of edges occurring together in more than one cell.

Fig 1(a). USA Map - Shifted and Overlaid on itself

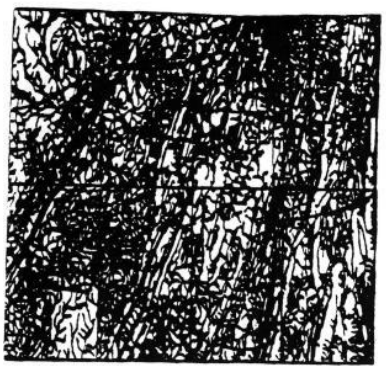

Fig 1(b). Chickamauga Area - All 4 overlays

THEORETICAL ANALYSIS

Let $N_{c/e}$ be the number of cells that an average edge passes through. The determination of $N_{c/e}$ is similar to the Buffon's Needle Problem, McCord (1964). A simple analysis shows that,

$$N_{c/e} = (1+\frac{4}{\pi}LG) \quad (Eq.1)$$

Then N_p, the total number of (cell, edge) pairs is

$$N_p = N(1+\frac{4}{\pi}LG) \quad (Eq.2)$$

The average number of edges per cell is

$$N_{e/c} = \frac{N_p}{G^2} \quad (Eq.3)$$

$$= \frac{N}{G^2}(1+\frac{4}{\pi}LG) \quad (Eq.4)$$

The time to calculate the (cell, edge) pairs is

$$T_1 = \alpha N_p \quad (Eq.5)$$

where α is a constant. The time to test the edges for intersections is about

$$T_2 = \beta G^2 N_{e/c}(N_{c/e} - 1) \quad (Eq.6)$$

where β is a constant. The overhead for processing the cells is

$$T_3 = \gamma G^2 \quad (Eq.7)$$

where γ is a constant. and the total time is

$$T = T_1 + T_2 + T_3$$

$$= N\left[(\alpha-\beta)+\frac{4}{\pi}(\alpha-\beta)LG\right] + \beta N^2\left[\frac{1}{G^2}+\frac{8}{\pi}\frac{L}{G}+\frac{16}{\pi^2}L^2\right] + \gamma G^2 \quad (Eq.8)$$

This is minimized if the 2 fastest terms in the sum grow at the same speed, which occurs when $G = \min\left(\delta\sqrt{N}, \frac{\pi}{4L}\right)$ for some δ.

What about some cells being denser since the edges are randomly distributed? Since the time to process a cell depends on the square of the number of edges in that cell, an uneven distribution might increase the total time. However, since the edges are assumed independent, the number of edges per cell is Poisson distributed, and the expected value of the square of the number of edges equals the square of the expected number of edges. Therefore the expected time doesn't increase.

RESULTS

Edge Intersection

For each data set we tried many values of G to learn the variation of time with G. Table 1 shows the results from intersecting the 116896 edges in all the 4 overlays of the Chikamauga DLG (Figure 1). There are 144,666 intersections in all, and the best time is 37 seconds with a 325×325 grid. The time is within 50% of this for grids from 175×175 up to 1000×1000, which shows the extreme insensitivity of the time to the grid size. This is why real scenes with dense and sparse areas can be accommodated efficiently.

For the USA state boundaries shifted and overlaid on themselves, the execution time is within 20% of the optimum from about $G = 40$ to $G = 400$ and is within a factor of two of the optimum from about $G = 20$ to $G = 700$. Outside these limits, the execution time starts to rise quickly.

The economy of the grid structure is shown by the fact that the number of comparisons between pairs of edges needed to isolate the intersections is about twice the number of the edges when using the optimal grid resolution. This behavior was also observed in hidden surface algorithm described in earlier publications. There is not much room for

No. of edges	116896
Avg. edge length	0.00231
Standard deviation	0.0081
Xsects. by end pt. coincidence	135875
Xsects. by actual equation soln	8791
Total intersections	144666

Grid Size	Pairs	P/Cell	P/Edge	Grid Time	Sort Time	Xsect Time	Total Time
50	131462	52.585	1.125	4.33	3.67	182.04	190.04
80	140407	21.939	1.201	4.50	3.93	90.75	99.18
100	146389	14.639	1.252	4.72	4.22	67.31	76.25
125	153492	9.823	1.313	4.88	4.32	51.36	60.56
175	168341	5.497	1.440	5.43	4.82	36.22	46.46
200	175791	4.395	1.504	6.70	6.13	35.18	48.01
275	197815	2.616	1.692	8.45	7.68	31.78	47.91
325	212282	2.010	1.816	7.37	6.18	23.60	37.15
400	234372	1.465	2.005	8.37	7.15	21.82	37.33
500	263646	1.055	2.255	10.18	7.78	20.62	38.58
625	300413	0.769	2.570	11.72	8.92	20.22	40.85
800	351891	0.550	3.010	14.52	10.77	21.37	46.65
1000	410589	0.411	3.512	17.72	12.93	23.05	53.70
2000	704147	0.176	6.024	31.05	22.92	29.57	83.53

Table 1: Intersecting 116,896 Edges of the Chikamauga DLG

further improvement by a hierarchical method.

The largest cartographic database was the 116,896 edges of the Chikamauga Digital Line Graph (DLG) from the USGS sampler tape. The average edge length was 0.0022 and the standard deviation 0.0115, so the edges were quite variable. We used a 325×325 grid to find all 144,666 intersections in 37.15 seconds on a Sun 4/280. Other results are listed in Franklin, Chandrasekhar, Kankanhalli, Seshan, Akman (1988).

One of our examples consisted of 1,819,064 edges, with an average length of 0.0012, forming a complete VLSI chip design. We found all 6,941,110 intersections in 178 seconds. In this case, the program was optimized to use the orthogonality of the edges. The edges' lengths were quite variable, with the standard deviation being over 30 times the mean. This example illustrates the generality of this method and its applicability to other areas besides cartography.

Execution in Parallel

The uniform grid method is ideally suited to execution on a parallel machine since it mostly consists of two types of operations that run well in parallel: applying a function independently to each element of a set to generate a new set, and sorting. Determining which cells each edge passes through is an example of the former operation.

We implemented several versions of the algorithm on a Sequent Balance 21000 computer, which contains 16 National Semiconductor 32000 processors, Sequent (1986), Kallstrom(1988) and compared the elapsed time when up to 15 processors were used to

the time for only one processor, Kankanhalli (1988). We used the 'data partitioning' paradigm of parallel programming which involves creating multiple, identical processes and assigning a portion of the data to each process. The edges are distributed among the processors to determine the grid cells to which each edge belongs and then the cells are distributed among the processors to compute the intersections. Since the Sequent Balance 21000 is a shared memory parallel computer, shared data structures is the communication mechanism for the processors. The synchronization of the processors is achieved by using atomic locks. Basically, the concept of 'local processing' has been adopted in this algorithm to achieve parallelism.

There were several different ways of implementing the uniform grid data structure. First, we had a G^2MP array of cells, where G is the grid size, M is the maximum number of edges per cell per processor and P is the number of processors. However this implementation took up a lot of memory space though it obviated the use of locks. Then, G^2 array of linked lists was used. This also did not require locking but it was slow because of the dynamic allocation of shared global memory. Then it was implemented using a linked list of (*cell*,*edge*) pairs but this also was slow because of dynamic memory allocation. Finally a G^2M array implementation was made which used atomic locks. This implementation gave the best results.

The speedup ratios range from 8 to 13. Figure 2 shows the results from processing 3 overlays of the United State Geological Survey Digital Line Graph, totaling 62,045 edges. 81,373 intersections were found. The time for one processor was 273 seconds, and for 15 processors was 28 seconds, for a speedup of about 10. This is a rate of 7.9 million edges and 10.5 million intersections per hour. For other data sets, these extrapolated times would depend on those data sets' number of intersections per edge.

The speedup achieved for any parallel algorithm is dependent on the amount of inherently sequential computation in the algorithm, the hardware contention imposed by the competing processors, the overhead in creating multiple processes and the overhead in synchronization & communication among the multiple processes. We believe that the first factor is not dominant when using the uniform grid technique. The large speedups achieved show that the other three factors also do not affect the performance significantly. Finally, the speedup, as a function of the number of processors, was still rising smoothly at 15 processors. This means that we should achieve an even bigger speedup on a more parallel machine.

Map Overlay

We are implementing a complete map overlay package in C on a Sun workstation. The input and output are in a simplified form of the Harvard Odyssey cartographic database format. The preliminary version emphasizes clarity at the expense of speed by representing the process as a pipeline of several sequential processes. Each process writes its output to a temporary ASCII file for the next process to read, thus incurring repeated I/O costs. The stages are as follows.

1. *Deform*: In this stage chains are broken into edges.
2. *Intersect*: This stage finds all intersection points between edges.
3. *Connect*: This stage breaks up original chains into new chains, additional break points being made at the intersection points.
4. *Link*: This stage calculates and sorts the angles between each of the chains at each node and the x-axis.
5. *Form*: This stage recognizes all polygons formed by the new chains.

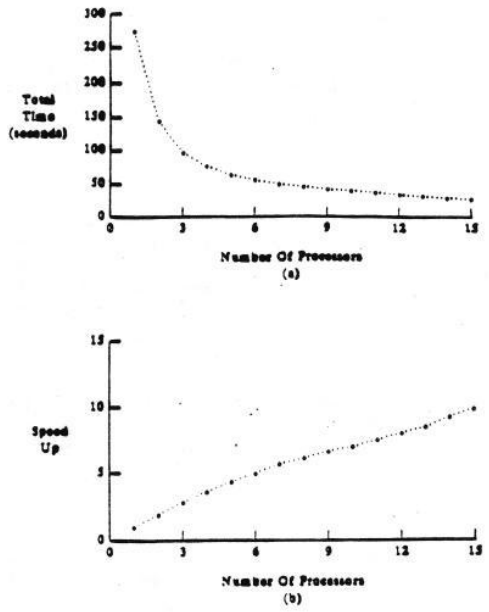

Fig 2. Time and Speedup when intersecting the 62045 edges in the Roads & Trails, Railroads and Pipes and Transmission Lines Overlays of The Chickamauga DLG in Parallel on 1 to 15 Processors. Grid size = 250. 81,373 intersections found.

6. *Display*: This displays the resulting overlaid map along with labels for each recognizable polygon.

7. *Timer*: This sums up the time each of the first 5 modules takes to complete each individual task.

One, possibly controversial, decision, was to split the chains into the individual edges at the start. This makes the data more voluminous, but much simpler, since now the elements have a fixed length. After we have intersected all the edges, and split them into pieces which are the edges of the result, it is easy to reform the output chains.

Another advantage of using individual edges is that the algorithm will be easier to

implement on a parallel machine for even greater speed.

We have implemented the algorithm partly on a Sun 3/50 and part on a Sun 4/280. Testing all 3660 edges in both input maps to find intersections takes only 1.73 seconds on the Sun 4.

CONCLUSION

Our technique has been successfully used for the important problem of map overlay which occurs in cartography. The results indicate that this is a very robust general technique which is fast and simple. It is evident from this research that simple solutions are often faster than theoretically efficient but convoluted and complicated methods. Also, the power of randomized techniques in algorithm design for real world problems is now being appreciated. Our algorithm is parallelizable and shows very good speedup with minimal auxiliary data structures.

As mentioned before, we are investigating other problems where the uniform grid technique may be applied for inventing parallel algorithms. We feel that the uniform grid technique is a good technique for parallel geometric computation in the future.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under PYI grant no. CCR-8351942. It used the facilities of the Computer Science Department funded by a DoD equipment grant, and the Rensselaer Design Research Center.

REFERENCES

Aggarwal, Alok, Chazelle, Bernard, Guibas, Leo, O'Dunlaing, Colm, and Yap, Chee (1985) "Parallel Computational Geometry," *Foundations of Computer Science - 25th Annual Symposium*, pp. 468-477 (1985).

Akl, Selim G. (1985) "Optimal Parallel Algorithms for Selection, Sorting, and Computing Convex Hulls," pp. 1-22 in *Computational Geometry*, ed. Godfried T. Toussaint (1985), pp. 1-22.

Chazelle, Bernard and Edelsbrunner, Herbert (1988) "An Optimal Algorithm for Intersecting Line Segments in the Plane," *Foundations of Computer Science - 29th Annual Symposium*, White Plains (October 1988).

Chrisman, T.K. Peucker, and N. (1975) "Cartographic Data Structures," *The American Cartographer* 2(1), pp. 55-69 (1975).

Clarkson, Kenneth L. (1988a) "Applications of Random Sampling in Computational Geometry, II," *Proc 4th Annual Symposium on Computational Geometry*, Urbana-Champagne, Illinois, pp. 1-11, ACM (June 6-8, 1988).

Clarkson, Kenneth L. and Shor, Peter W. (1988b) "Algorithms for Diametrical Pairs and Convex Hulls that are Optimal, Randomized, and Incremental," *Proc 4th Annual Symposium on Computational Geometry*, Urbana-Champagne, Illinois, pp. 12-17, ACM (June 6-8, 1988).

Dippe, Mark and Swensen, John (1984) "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics* 18(3), pp. 149-158 (July 1984).

Evans, D.J. and Mai, Shao-wen (1985) "Two parallel algorithms for the convex hull problem in a two dimensional space," *Parallel Computing* 2, pp. 313-326 (1985).

Fiume, Eugene, Fournier, Alan, and Rudolph, Larry (1983) "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer," *Computer Graphics* 17(3), pp. 141-150 (July 1983).

Franklin, W. Randolph (1978) *Combinatorics of Hidden Surface Algorithms*, Center for Research in Computing Technology, Harvard University (June 1978). Ph.D. thesis

Franklin, Wm. Randolph (1980) "A Linear Time Exact Hidden Surface Algorithm," *ACM Computer Graphics* 14(3), pp. 117-123, Proceedings of SIGGRAPH'80 (July 1980).

Franklin, Wm. Randolph (1981) "An Exact Hidden Sphere Algorithm That Operates In Linear Time," *Computer Graphics and Image Processing* 15(4), pp. 364-379 (April 1981).

Franklin, Wm. Randolph (1982) "Efficient Polyhedron Intersection and Union," *Proc. Graphics Interface'82*, Toronto, pp. 73-80 (19-21 May 1982).

Franklin, Wm. Randolph (1983) "A Simplified Map Overlay Algorithm," *Harvard Computer Graphics Conference*, Cambridge, MA (31 July - 4 August 1983). sponsored by the Lab for Computer Graphics and Spatial Analysis, Graduate School of Design, Harvard University.

Franklin, Wm. Randolph (1984) "Adaptive Grids for Geometric Operations," *Cartographica* 21(2 & 3) (1984).

Franklin, Wm. Randolph and Akman, Varol (1985) *A Simple and Efficient Haloed Line Algorithm for Hidden Line Elimination*, Univ. of Utrecht, CS Dept., Utrecht (October 1985). report number RUU-CS-85-28

Franklin, Wm. Randolph and Wu, Peter YF (1987) "A Polygon Overlay System in Prolog," *Autocarto 8: Proceedings of the Eighth International Symposium on Computer-Assisted Cartography*, Baltimore, pp. 97-106 (March 29– April 3, 1987).

Franklin, Wm. Randolph, Chandrasekhar, Narayanaswami, Kankanhalli, Mohan, Seshan, Manoj, and Akman, Varol (1988) "Efficiency of Uniform Grids for Intersection Detection on Serial and Parallel Machines," *Computer Graphics International*, Geneva (May 1988).

Hu, Mei-Cheng and Foley, James D. (1985) "Parallel Processing Approaches to Hidden Surface Removal in Image Space," *Computers & Graphics* 9(3), pp. 303-317 (1985).

Kallstrom, Marta and Thakkar, Shreekant (1988) "Programming Three Parallel Computers," *IEEE Software* 5(1), pp. 11-22 (January 1988).

Kankanhalli, Mohan "The Uniform Grid Technique for Fast Line Intersection on Parallel Machines," (1988) M.S. Thesis, Electrical,Computer & Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY (April 1988).

Kaplan, Michael and Greenberg, Donald P. (1979) "Parallel Processing Techniques For Hidden Surface Removal," *Computer Graphics* 13, pp. 300-309 (1979).

McCord, J. and Moroney, R (1964) pp. 4-8 in *Probability Theory*, The Macmillan Company, New York (1964), pp. 4-8.

Nievergelt, J. and Preparata, F.P. (1982) "Plane-Sweep Algorithms for Intersecting Geometric Figures," *Comm. ACM* 25(10), pp. 739-747 (October 1982).

Preparata, Franco P. and Shamos, Michael Ian (1985) *Computational Geometry An Introduction*, Springer-Verlag (1985).

Reif, John H. and Sen, Sandeep (1988) "An Efficient Output-sensitive Hidden-Surface Removal Algorithm and its Parallelization," *Proc. Fourth Annual Symposium on Computational Geometry*, pp. 193-200 (June 1988).

Samet, H. (1984) "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys* 16(2), pp. 187-260 (June 1984).

1986. Sequent Computer Systems Inc.,, *Balance Technical Summary*, 1986.

Stojmenovic, Ivan and Evans, David J. (1987) "Comments on two parallel algorithms for the planar covex hull problem," *Parallel Computing* 5, pp. 373-375 (1987).

Waugh, T.C. (1986) "A Response to Recent Papers and Articles on the Use of Quadtrees for Geographic Information Systems," *Proceedings of the Second International Symposium on Geographic Information Systems*, Seattle, Wash. USA, pp. 33-37 (5-10 July 1986).

Wu, Peter Y.F. and Franklin, Wm. Randolph (1988) "A Logic Programming Approach to Cartographic Map Overlay," *International Computer Science Conference*, Hong Kong (December 1988).

Yap, Chee-Keng (1987) "What can be Parallelized in Computational Geometry?," *International Workshop on Parallel Algorithms and Architectures* (May 1987).