EFFICIENT INTERSECTION CALCULATION IN LARGE DATABASES

Wm. Randolph Franklin,
Narayanaswami Chandrasekhar
Mohan Kankanhalli

Electrical, Computer, and Systems Engineering Dept., *and* Computer Science Department, 6026 J.E.C., Rensselaer Polytechnic Institute, Troy, NY, 12180, USA. (518) 276-6077, *Internet:* WRF@ECSE.RPI.EDU, *Bitnet:* WRFRANKL@RPITSMTS, *Fax:* (518) 276-6261, *Telex:* 6716050 RPI TROU.

Varol Akman

Dept. of Computer Engineering and Information Sciences, Bilkent University, PO Box 8, 06572 Maltepe, Ankara, TURKEY. *Bitnet:* AKMAN@TRBILUN.

Peter YF Wu

IBM Thomas J. Watson Research Center, PO Box 704, Yorktown Heights, NY, 10598, USA. *Internet:* PWU@IBM.COM

Efficient cartographic operations on large databases are possible using techniques such as uniform grids and local topological data structures. Primitive operations, such as determining all the intersections among a large set of small edges, can be extended upward to cartographic overlay operations. The uniform grid is a regular grid overlaying the data whose resolution depends on the number and length of the data's edges. Hierarchical data structures such as quadtrees are not necessary for these operations. The grid resolution can be varied a factor of three either way from the optimum without increasing execution time over 65%. The local data structures represent the object as a set of fractional neighborhoods of the vertices. The edges and faces are stored only implicitly. With this, properties such as the areas of all the polygons resulting from the overlaying of two maps can be calculated without calculating the polygons themselves explicitly. The edge intersection has been implemented and tested. For example two million edges were tested to find all seven million intersections in three minutes on a Sun 4/280. The complete USGS Digital Line Graph sampler tape of 116,896 edges was intersected in 37 seconds.

1. INTRODUCTION

Overlaying two maps is one of the most difficult problems in computer assisted cartography [White77a, White89a]. It is related to geometric problems in separate fields, such as interference testing in robotics, VLSI circuit extraction, hidden surface removal in computer graphics, in that they all have common low level operations, such as testing a point for inclusion in a polygon, and finding which of a large set of small edges intersect. These operations then integrate upward to applications such as boolean operations on thousands of polyhedra, visible surface calculations for large scenes, determining mass properties of the union of many rectangles without determining the union polygon itself, and determining the area of the intersection of several polygons each defined as the union of many rectangles.

This paper presents algorithms to solve some of these problems using the techniques of uniform grids, local topological data structures, and multiple precision rational fractions in Prolog. An efficient algorithm for finding the intersections among large numbers of edges is extended to finding the area and perimeter of the union of many rectangles. These use the uniform grid and the local data structures. These mass properties can be found more quickly than finding the resulting polygon itself. A map overlay algorithm using rationals in Prolog is also presented.

High speed on large databases is a major advantage of these techniques, and has been demonstrated experimentally. We have processed a complete integrated circuit design of 1,819,064 edges to find all 6,941,110 intersections in 178 seconds on a Sun 4/280. Processing the complete USGS Chikamauga DLG (Digital Line Graph) took only 37 CPU seconds. When implemented on a 16 processor Sequent, which has separate processors with a common bus and memory, the program ran 10 times as fast with 15 processors as with one. Finding mass properties of the union of many rectangles was also implemented. It processed a union of 100,000 rectangles in 199 seconds.

This paper will first present uniform grids, compare them to other possibilities, and describe their use to intersect edges. Then it will describe local topological data structures. Next it will apply these ideas to the map overlay problem. Finally it will discuss the advantages of these concepts.

2. UNIFORM GRIDS

We use *uniform grids* to determine spatial adjacency and intersection [Chandras87a, Franklin88a, Kankanha88a, Akman89a]. Here a uniform, regular, grid, with resolution determined by the number and average length of the edges, is laid over the data. For each object, the cells that it passes through are noted. Then the data structure is inverted and the contents of each cell are processed sequentially. This algorithm is useful for operations such as detecting intersections among large numbers of small edges, boolean combinations of polyhedra, and hidden surface calculations of complex scenes.

2.1. Comparison to Quadtrees

The obvious, but wrong, objection to uniform grids is that they cannot handle irregular scenes, and that hierarchical methods such as quadtrees must be used to give finer cells where the data is denser. However a criticism of their overuse in GIS is given in [Waugh86a]. There are three answers to the use of quadtrees.

60-2

Theoretical: If the data are distributed independently and identically then some cells will have more objects than others, with n_i, the number of objects in cell #i following a Poisson distribution, $p(n_i = k) = \lambda^k e^{-\lambda}/k!$, where λ is the average number of objects per cell. Then the time to process cell #i by comparing all objects in it against each other will be n_i^2, and its mean will be λ^2. That is, for the Poisson distribution, the mean of the square is the square of the mean. In this case, the expected time, $\overline{T} = \Theta(N+K)$†, where N is the number of input objects, and K the number of output intersections. More details of the analysis are given in [Akman89a, Franklin88a].

If the input objects are distributed unevenly, but the unevenness is "bounded", then the above result is still true. It is sufficient that the densest cell be a constant times the average density as the data sets grow to infinity. Similar restrictions are imposed on data in other fields, such as Lipschitz conditions in partial differential equations, and bounded variation in analysis.

Real data can be worse than than the randomness assumptions indicate. For example, the probability of two random edges having the same endpoint is zero, but this happens. However, such correlations are of only local importance, and become relatively more insignificant as $N \rightarrow \infty$.

Experimental: We have tested uniform grids with several algorithms on data in assorted applications, and have never observed a serious problem. These tests have included apparently quite bad input data, such as a haloed line algorithm on a wire-frame database of 11,000 edges [Franklin85a]. The data contained faceted cylinders composed of dozens of faces, whose many, close, parallel lines did not intersect each other, but did pass through the same cells several times. This caused some cells to have many more edges than the average. Also, when one cylinder's projection crossed another, all the edges of one intersected all the edges of the other. Nevertheless, calculating the haloed lines took only about 10 minutes on a Prime 500 computer.

Experimentally the actual grid size is not critical, within large variations. Factors of three in grid resolution off from the optimum usually increase execution time by under 50%. This represents a range of almost 100 in number of cells during which the execution time varies less than 50%. The optimum is so broad because the total time is the sum of two parts: putting objects into cells, which runs faster if there are fewer cells, and processing the cells, which runs slower. This insensitivity of time to grid size explains why the uniform grid can handle regions of varying data density.

Quadtree Performance: Quadtrees are useful for defining irregular regions of space in image processing and (as octrees) defining irregular objects, but they cannot handle very irregular geometric scenes. Consider a scene with N parallel edges spaced $1/N^2$ apart. A quadtree fine enough to determine that there are no intersections would require more than N^2 time to construct, so the even naive method would be faster.

The problem with hierarchical methods in general is that if N=1,000,000 and the quadtree is fine enough that on average each object is in a separate low level cell, then the tree, even if perfectly balanced, is $\log_4 1000000 = 10$ levels deep. In practice, 15

† $T = \Theta(f(N))$ means that T grows just as fast as N does. Formally, there exists N_0, $0<\alpha<\beta$, such that for all $N>N_0$, $\alpha f(N) \leq T \leq \beta f(N)$.

would be more likely. Thus accessing any object requires following ten to fifteen pointers. Pointers to random locations in memory, the usual case, defeat any hardware caching mechanism. The tree can be sorted and compressed, but this is slow and prevents dynamic updating.

The difference between flat uniform data structures and hierarchical adaptive data structures is similar to the difference between relational databases and hierarchical ones. Forcing a database to satisfy the Codd Normal Forms may increase the size, but gives increased regularity in return.

Parallelizability is the final advantage of a uniform data structure. The cells that an object passes through can be determined in parallel with each object. After the data set is inverted, each cell can be processed in parallel. In contrast, constructing a tree is inherently a more sequential operation, and accessing all elements by first traversing the root may cause collisions there (depending on the model of parallel machine) unless the top few nodes of the tree are replicated.

The problems with using trees on parallel machines have been observed by Hillis [Hillis85a, Hillis86a]: "One case where our serial intuition misled us was our expectation that parallel machines would dictate the use of binary trees. It turns that linear linked lists serve almost as well, since they can be inverted to balanced binary trees whenever necessary and are far more convenient for other purposes."

Nevertheless, a limited hierarchical data structure may prove useful in the future if the available storage is hierarchical. On a parallel machine such as a Connection Machine or Hypercube, it may be reasonable to partition the data in subsets small enough to fit onto one processor. Each processor would then apply a uniform grid. This two level data structure would increase computation costs, since it would be harder to parallelize the initial partition step, but it would lower communication costs, which are often more important.

2.2. Comparison to Theoretically Optimal Methods

Chazelle and Edelsbrunner's line intersection algorithm [Chazelle88a], has a worst case time of $T = \Theta(N\log N + K)$. However, the algorithm is much more complicated to implement and appears to be quite difficult to parallelize. There are simpler "scan line" algorithms with $T = \Theta((N+K)\log N)$. However they also appear inherently sequential. One could start P separate scanlines in parallel, but that would incur an overhead in splitting the edges between the processors. The scan line methods also take up to $T = \Theta(N^2 \log N)$, that is, worse than the naive method, for very bad data.

Both of these theoretically optimal methods have a $\log N$ factor in the time because they work with a theoretical machine model which is weaker than real computers. Thus they assume that sorting takes $\Theta(N\log N)$ expected and maximum time, which is true only for pair-comparison models of machines. If we assume that in constant (not log) time we can write to any one of N words of memory, then sorting i.i.d. keys takes $\overline{T} = \Theta(N)$ time with the address calculation sort, [Knuth73a]. A radix sort, similar to a bucket sort, is

60-4

also much faster than the theory on a pair-comparison machine would suggest.

2.3. Application to Edge Intersections

Finding all the intersections among a large set of small edges was the first application of uniform grids. The algorithm, presented in [Chandras87a, Akman89a, Franklin88a, Kankanha88a, Franklin89a, Franklin89b], has expected time equal to the sum of the sizes of the input and output. The sophistication of the algorithm rests in its simplicity, as well as in the appropriate implementation of the abstract data structure for the cell–edge information. Various possibilities for serial and parallel implementation are examined in[Franklin88b]

The largest example used so far is a complete design of WaRP1, a VLSI chip by Jim Guilford. It has 1,819,064 edges, and we found all 6,941,110 intersections in 178 seconds on a Sun 4/280 with 32 MB of real memory. The program took advantage of the large flat address space by storing all the intermediate and final data in 110 MB of virtual memory. The virtual memory was not thrashed. The algorithm was implemented as a C program using the Sun-supplied compiler, which doesn't optimize. Using commercial quality compilers and assembler modules for inner loops would presumably reduce this time considerably. This version did take advantage of the fact that all the edges were horizontal or vertical, which gave us an estimated factor of three in speed, and simplified the code.

We also implemented a version for general (non-rectilinear) edges, and tested it on the complete USGS (United State Geological Survey) DLG (Digital Line Graph) sampler tape, of Chikamauga, Tennessee, figure 1. This database contains all surface features, such as roads, railroads, rivers, streams, power lines, and pipelines, but not the contour lines. Finding all 144,666 intersections among the 116,896 edges took only 37 CPU seconds. The unevenness of the data can be seen and its irregularity quantified: for the edge lengths, $\mu = 0.00231$, while $\sigma = 0.0081$.

The uniform grid structure is remarkably insensitive to the actual grid spacing. Figure 2 shows the effect of processing the Chikamauga data and varying the grid resolution over a range of from 50×50 to 2000×2000. Over a factor of eight in grid resolution, from 125 to 1000, the time is less than 65% worse than the minimum. From 175 to 800, the time is less than 30% worse than the minimum, which occurs at 325. The insensitivity is due to the time to insert the edges into cells running slower when there are more cells, while the time to process the cells runs faster (within broad limits) because the cells are emptier. Contrary to our expectations, at the optimal grid resolution, most of the time is spent in processing the cells.

When implemented on a 16 processor Sequent Balance 21000, which has separate processors with a common bus and memory, the program ran 10 times as fast with 15 processors in use as with one processor. Finding all 81,373 intersections in a 62,045 edge database (half the Chikamauga data) took only 28 seconds elapsed time. Unfortunately, the Balance with 16 National Semiconductor 32000 processors is an older machine which is slower than the Sun 4. Figure 3 shows the relative parallel efficiency, that is (time on one processor) / P / (time on P processors). It is 66% for $P = 15$, which is quite good.

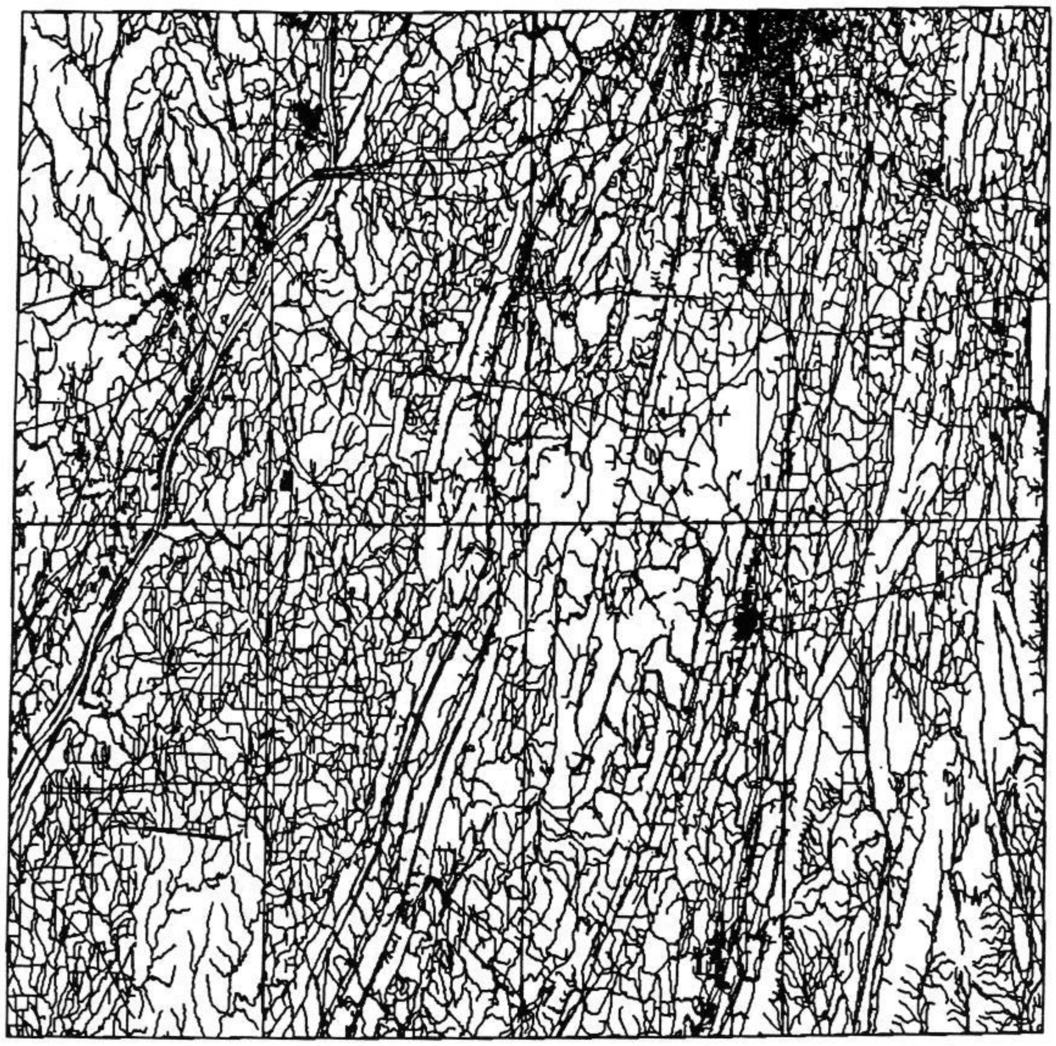

Figure 1: Chikamauga DLG

3. LOCAL TOPOLOGY DATA STRUCTURES

It is possible to determine mass properties, such as area and perimeter, of a polygon using only local topological information about fractions of the neighborhoods of the vertices. All that is required is the set of edge endpoints. For each endpoint, we need its coordinates, a unit vector pointing along the edge, and a normal vector to the edge pointing inside the polygon. Note that we do not explicitly have in one place either any complete edge or any complete vertex, although we could determine them by linking up the data.

Assume the following notation and definitions. O is the coordinate origin. P is a vertex. When it is necessary to refer to individual components, we will use $P = (P_x, P_y)$. T is a unit tangent vector from P along an edge. N is a unit vector normal to T, pointing

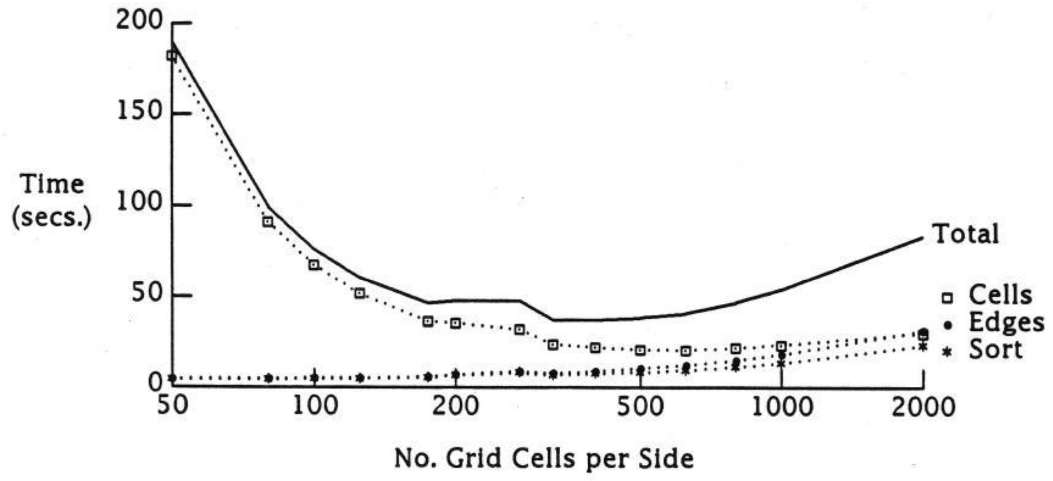

Figure 2: Effect of Grid Size on Edge Intersection Time

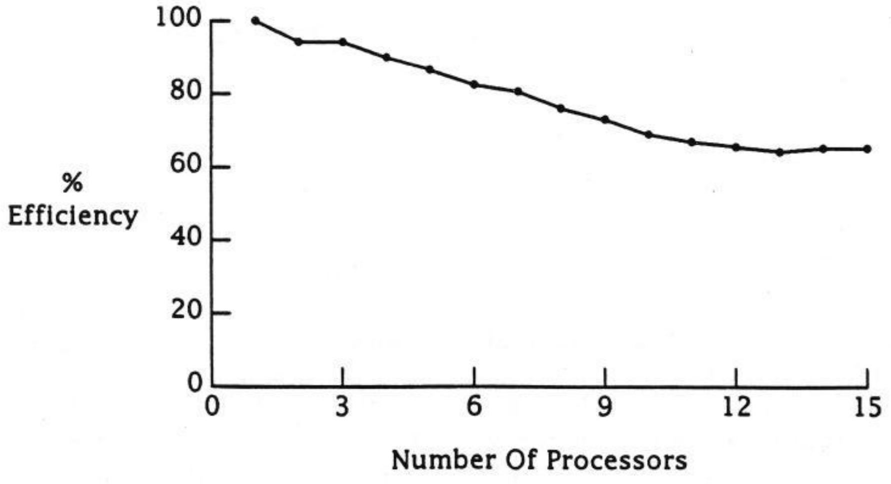

Figure 3: Parallel Efficiency for Intersecting 62,045 Edges on Sequent

towards the polygon's interior. L is the polygon's perimeter. A is its area. $\hat{P}$ is a unit vector parallel to P.

σ is the sign function, $\sigma(x) = \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{cases}$

60-7

Consider the polygon vertex, and its neighborhood, shown in figure 4.

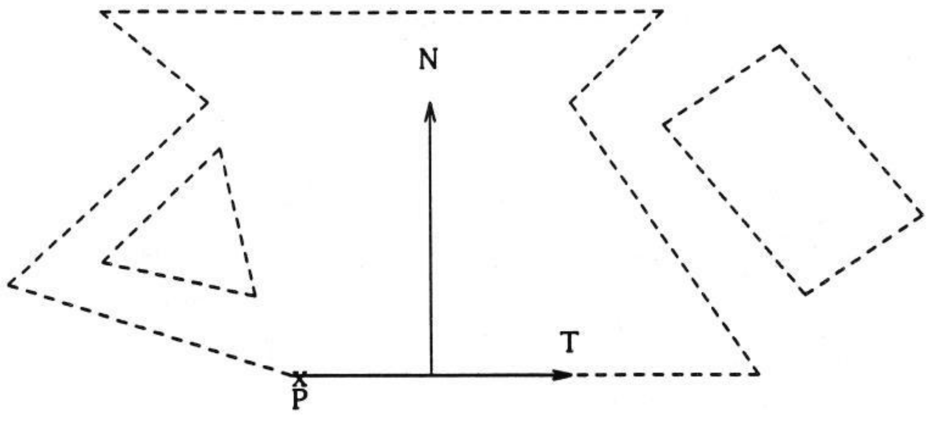

Figure 4: Polygon Vertex - Edge Adjacency

The polygon is represented as the set *{(P, T, N)}*, where there is one element for each occurrence of an adjacency of a vertex and edge.

Given this, the following theorems are true; the proofs are in [Franklin88c].

$$L = -\sum P \cdot T$$

$$A = \frac{1}{2} \sum P \cdot T \, P \cdot N$$

$$\text{if } O \text{ is } \left\{ \begin{matrix} \text{outside the polyhedron} \\ \text{inside the polyhedron} \\ \text{on the boundary} \end{matrix} \right\} \text{ then } \sum \sigma\,(P_x T_x)\,\sigma(P \cdot N) = \left\{ \begin{matrix} 0 \\ 4 \\ 0, 2, \textit{ or } 4 \end{matrix} \right\} \text{resp.}$$

If O is on the boundary, then its value is the average of the value for points in the neighborhood above and those below. There is no loss of generality in testing O since any arbitrary point may be transformed to O. Although there are cases, such as raster display, where the complete topology is required, these formulae show that there are also many cases where it is unnecessary, and that then the calculations are both faster and parallelizable.

Other local topological methods are also possible. The *vertex neighborhood* method [Franklin87a], defines the polygon as a set of the neighborhoods of the vertices, that is as $\{(P, \hat{A}, \hat{B})\}$, where P is a vertex's position, and $\hat{A}$ and $\hat{B}$ are unit vectors along the two adjacent edges. $\hat{A}$ and $\hat{B}$ are oriented so that sweeping from $\hat{A}$ to $\hat{B}$ traverses inside the polygon. For each neighborhood, we can define a weight function assigning a value to each point in the plane depending on its relation to the neighborhood. The sum of these weight functions is a characteristic function for the polygon, which can be used for point inclusion testing and mass property calculation. This also extends to E^3.

60-8

Alternatively, one may split each edge into two semi-infinite *rays* [Franklin83a]. Again, point inclusion testing and mass property calculation are a parallelizable vector reduction operation on the set of data, regardless of the global topology.

4. MAP OVERLAY

4.1. Multiple Precision Rationals in Prolog

The problem of arithmetic roundoff errors that occur when the intersection of two lines is not exactly representable is well-known. Cartography simplifies the problem compared to general CAD databases since cartography uses only straight lines in E^2. Therefore the intersections are exactly representable in rational numbers. Although the numerators and denominators may be multiple precision, this is a problem only if the computation tree for the operation is deep. After considering various representational issues, we combined this with a test of the usefulness of Prolog as an implementation tool by implementing a multiple precision rational number map overlay package in Prolog [Wu88a]. As would be expected, the program ran quite slowly, although this would be improved by the Prolog hardware boards now in existence. On the other hand there were no roundoff errors.

4.2. Determining Areas of Resulting Polygons

The areas of all the polygons resulting from overlaying several maps can be determined without finding the complete overlaid map itself. It is sufficient to find all the output vertices, and which input polygons they are in. The intersections can be found by the fast edge intersection algorithm. Then the grid structure can be used to determine which other input polygons all the input vertices are in. This is sufficient information for the local data structure formula to calculate the output areas without calculating any connectivity information about the output map. We are now implementing this method.

5. SUMMARY

By synthesizing techniques such as uniform grids and local topological data structures, it is possible to derive geometric algorithms which can process large data sets with the following properties.

- *Simple data structures*, that is, sets, instead of trees. This obviates the need to tree balance, which is messy, but essential for optimal searching. Hierarchical methods such as quadtrees are not necessary.
- *Ease of implementation*, because of the above.
- *Parallelizability*, because the operations are mapped over sets of elements. The algorithms can be implemented even on a SIMD or a vector machine. Scheduling the cooperating processes becomes easier because of the simple set-based data structures. This results in lesser overhead, and hence gives better speed-ups.

60-9

- *Speed of execution* on all the real data seen so far, including regions of widely differing density.

- *Dynamizability*, since the data structures could be extended to allow us to add and delete objects on-line, and quickly know the changes in intersections and mass properties.

ACKNOWLEDGEMENTS

Partial support for this work was provided by the Directorate for Computer and Information Science and Engineering of the National Science Foundation under Institutional Infrastructure Grant No. CDA-8805910. It was also supported by NSF under PYI grant CCR-8351942. It was implemented on equipment provided by Sun Microsystems at a discount in the Computer Science Department and Rensselaer Design Research Center at RPI.

REFERENCES

Akman89a.
Akman, Varol and Wm. Randolph Franklin, "Geometric Computing and the Uniform Grid Data Structure," *Computer Aided Design*, 1989. (to appear)

Chandras87a.
Chandrasekhar, Narayanaswami and Manoj Seshan, "The Efficiency of the Uniform Grid for Computing Intersections," M.S. thesis, Electrical, Computer, and Systems Engineering Dept., Rensselaer Polytechnic Institute, December 1987.

Chazelle88a.
Chazelle, Bernard and Herbert Edelsbrunner, "An Optimal Algorithm for Intersecting Line Segments in the Plane," *Foundations of Computer Science - 29th Annual Symposium*, White Plains, October 1988.

Franklin83a.
Franklin, Wm. Randolph, "Rays - New Representation for Polygons and Polyhedra," *Computer Graphics and Image Processing*, vol. 22, pp. 327-338, 1983.

Franklin85a.
Franklin, Wm. Randolph and Varol Akman, *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

Franklin87a.
Franklin, Wm. Randolph, "Polygon Properties Calculated from the Vertex Neighborhoods," *Proceedings of the Third Annual Symposium on Computational Geometry*, pp. 110-118, Waterloo, Canada, 8-10 June 1987.

60-10

Franklin88b.
Franklin, Wm. Randolph, Narayanaswami Chandrasekhar, Mohan Kankanhalli, Varol Akman, and Peter YF Wu, "Efficient Geometric Algorithms for CAD," *IFIP TC 5/WG 5.2 Second Workshop on Geometric Modelling*, Elsevier, September 1988.

Franklin88c.
Franklin, Wm. Randolph, *Vertex Based Polyhedron Formulae*, Rensselaer Polytechnic Institute, November 1988. (submitted for publication)

Franklin88a.
Franklin, Wm. Randolph, Narayanaswami Chandrasekhar, Mohan Kankanhalli, Manoj Seshan, and Varol Akman, "Efficiency of Uniform Grids for Intersection Detection on Serial and Parallel Machines," *Computer Graphics International*, Geneva, May 1988.

Franklin89a.
Franklin, Wm. Randolph, Mohan Kankanhalli, and Chandrasekhar Narayanaswami, "Efficient Primitive Geometric Operations on Large Databases," *GIS National Conference*, Ottawa, March 1989.

Franklin89b.
Franklin, Wm. Randolph, Chandrasekhar Narayanaswami, Mohan Kankanhalli, David Sun, Meng-Chu Zhou, and Peter YF Wu, "Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines," *Autocarto 9: Proceedings Ninth International Symposium on Computer-Assisted Cartography*, pp. 100-109, Baltimore, MD, USA, April 1989.

Hillis85a.
Hillis, W. Daniel, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

Hillis86a.
Hillis, W. Daniel and Guy J. Steele, "Data Parallel Algorithms," *Communications of the ACM*, vol. 29, no. 12, pp. 1170-1183, December 1986.

Kankanha88a.
Kankanhalli, Mohan, "The Uniform Grid Technique for Fast Line Intersection on Parallel Machines," *M.S. Thesis*, Electrical,Computer & Systems Eng. Dept., Rensselaer Polytechnic Institute, Troy, NY, April 1988.

Knuth73a.
Knuth, D.E., *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley, 1973.

Waugh86a.
Waugh, T.C., "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*, pp. 33-37, Seattle, Wash. USA, 5-10 July 1986.

White77a.
White, Denis, "A New Method of Polygon Overlay," *An Advanced Study Symposium on*

60-11

Topological Data Structures for Geographic Information Systems, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, MA, USA, 02138, Oct. 16-21, 1977.

White89a.
White, Denix, Margaret Maizel, Kelley Chan, and Jonathan Corson-Rikert, "Polygon Overlay to SUpport Point Sample Mapping: The National Resources Inventory," *Autocarto 9: Proceedings Ninth International Symposium on Computer-Assisted Cartography,* pp. 384-390, Baltimore, MD, USA, April 1989.

Wu87a.
Wu, Peter Y.F., *Polygon Overlay in Prolog,* ECSE Dept., Rensselaer Polytechnic Institute, Ph.D. Thesis, Troy, NY, August 1987.

Wu88a.
Wu, Peter Y.F. and Wm. Randolph Franklin, "A Logic Programming Approach to Cartographic Map Overlay," *International Computer Science Conference,* Hong Kong, December 1988.

60-12