A POLYGON OVERLAY SYSTEM IN PROLOG

Wm. Randolph Franklin
Peter Y.F. Wu
Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY 12180-3590
U.S.A.
(email: franklin@csv.rpi.edu,wup@csv.rpi.edu)

ABSTRACT

A system for polygon overlay is developed using Prolog. The algorithm expands the concept of local processing and decomposes the process of polygon overlay into a number of stages, resulting in much simplified data structures. The system in Prolog adopts a relational approach to data structuring. Geometric entities are defined as Prolog facts, and the Prolog rules encoding geometry algorithms perform data processing. Processing follows a paradigm of set-based operations using iterative search and backtracking. Calculation of geometric intersection is done using rational arithmetic. Numerical accuracy is therefore preserved, and hence the topological consistency is guaranteed. Special cases of chain intersection, that is, touching and partially overlapping chains, are handled properly. A strategy to remove sliver polygons due to coincidental input is outlined.

INTRODUCTION

This paper presents a polygon overlay system developed in Prolog. We decompose the process of polygon overlay into a number of stages, each of which performs certain local operations. Our strategy simplifies data structuring. Furthermore, we achieve stability using rational arithmetic to compute geometric intersections.

Prolog offers several advantages as a programming tool for geometry algorithms. A fundamental problem in dealing with geometry on the computer is the primitive nature of conventional programming languages. Conceptually simple ideas often are unexpectedly difficult to implement. The descriptive nature of Prolog provides a much more intuitive programming environment, and hence fosters more readable programs. More specifically, Prolog rules are well suited to coding geometry algorithms for set-based operations (Swinson 82,83; Gonzalez 84; Franklin 86). Inherent to the problem of implementation is the design of data structures. Decomposing the complicated process of polygon overlay, we have simplified the data structures. Furthermore, Prolog provides a built-in relational database which makes data structuring even easier. Another problem in the implementation of geometry algorithms is numerical inaccuracy which results in topological

inconsistency. The problem stems from discretization errors in finite precision computer arithmetic (Franklin 84). Prolog has the flexibility of operator overloading to allow modular installation of other arithmetic domains, such as rational numbers, in existing programs. Our system for polygon overlay makes use of this feature.

The purpose of this paper is therefore two fold: to demonstrate that Prolog is a viable programming tool for geometric and geographic data processing, and to present rational arithmetic as a practicable solution to the problems stemming from discretization errors, in polygon overlay. In this paper, we will first briefly survey the previous works on the polygon overlay problem. Then, we will describe our polygon overlay system and the design of our data structures. We will discuss further on the issue of using rational arithmetic to calculate geometric intersections, and will outline an approach to removing slivers due to coincidental input data. We have implemented our system on a SUN 2/170 machine running C-Prolog version 1.5 (Pereira 86) and we are gathering more results at the time of this writing.

A BRIEF SURVEY

Polygon overlay encompasses a number of geometric and topological computation problems. During the 70's when geographic information systems were first developed, geographers thought of polygon overlay as "the most complex problem of geographic data structuring..." (Chrisman 76). Reports studying the problem at further length also had similar remarks (Goodchild 78; White 78). Guevara presented formal treatment and an analysis of several solutions in his thesis (Guevara 83).

Fundamental to the development of a solution during the 70's was the research effort in computational geometry. Solutions to basic problems of polygon intersection, and point-in-polygon inclusion were reported in (Eastman 72; Franklin 72) and (Ferguson 73), respectively. Shamos and Bentley in 1976 developed a number of algorithms to efficiently solve many geometry problems (Shamos 76). Preparata and Shamos organized most of the work in the design and analysis of geometry algorithms in their book (Preparata 85). Algorithms in determining polyline intersections were particularly important to polygon overlay. Burton designed a data structure, called Binary Search Polygon Representation, for efficient processing of polygons and polylines (Burton 77). Study on algorithms to determine polyline intersections were reported in (Freeman 75) and (Little 79).

Systems with polygon overlay implemented were available in the late 70's. The well-known CGIS - Canadian Geographic Information System combined grid/raster based approach with vector based approach to perform map overlay (Tomlinson 76). Two systems, PIOS - Polygon Information Overlay System (DeBerry 79) and MOSS - Map Overlay and Statistical System (Reed

82), both operate on two polygons at a time in pairwise comparison. By far a much more advanced algorithm due to White introduced the concept of local processing in WHIRLPOOL - a program in the system ODYSSEY (White 78). Franklin described an adaptive grid for efficient determination of intersecting objects (Franklin 80,83a). We implemented this idea in Prolog as presented in this paper. Teng reported a system taking a topological approach which quite likely is based on the concept of local processing (Teng 86).

DESCRIPTION OF THE SYSTEM

Polygon overlay is the process of superimposing two maps: Given two maps A and B, the polygon overlay process produces an output map C which comprises all the information of each input map, as well as the spatial correlation information. Map C contains all the chains of A and B; intersecting chains are split at the intersection points. Thus C contains all the nodes of A and B, and the new nodes generated at the intersection points. Each polygon in C is the intersection of two polygons, one in each of A and B.

Input/Output File Structures

We assume data consistency in our input maps. An input map is a file of variable length records. Each record is a Prolog "fact" in the following format:

chain(C,N1,N2,[[X,Y],..],P1,P2)

Each record is uniquely identified by name *C*; *N1,N2* are the names of the beginning and ending nodes; [[*X,Y*],...] is the list of (x,y) coordinates for the vertices along the polyline structure from *N1* to *N2*; and *P1,P2* are the polygons to the left and right of chain in the direction from *N1* to *N2*. The output map is a set of chain records in the same format, with each polygon identified as the intersection of two input polygons.

An Overview of The System

Our system divides the polygon overlay process into three major stages. Each is further subdivided into a number of steps. The following presents a brief description of the three stages. We will then discuss each stage in further detail.

1. *Chain Intersection.*
   Determine intersecting chains and split them at the intersection points.

2. *Polygon Formation.*
   Link the chains to form the output polygons.

3. *Overlay Identification.*
   Identify each output polygon as the intersection of two input polygons.

Chain Intersection Determining chain intersections is inevitably the performance bottle-neck in the polygon overlay process. To speed it up, we cast an adaptive grid over the edge segments (Franklin 83b). The intention is to isolate cases of intersection to within those elements that occupy the same grid cell. Implemented in our polygon overlay system, it involves the following steps:

1.1 Compute an appropriate grid size to form the grid.

1.2 Cast each edge element into each of the grid cells occupies. For each edge segment E, enter a fact *edge_in_grid(E,G)* for each grid cell G occupied by E.

1.3 Collect the edges in each grid cell for pairwise comparison. For each grid cell with potentially intersecting edges *E1,E2,...*, enter a fact *edges_in_same_grid(G,[E1,E2,...])*.

1.4 For each grid cell, test all pairs of edges in it to determine intersecting pairs. Split the edges and form a new node at the intersection point.

Figure 1 depicts the casting of a grid to isolate the intersection cases. The process splits the intersecting chains at the point of intersection. Hence, we have all the chains of the output map: the chains do not intersect each other except at the nodes.

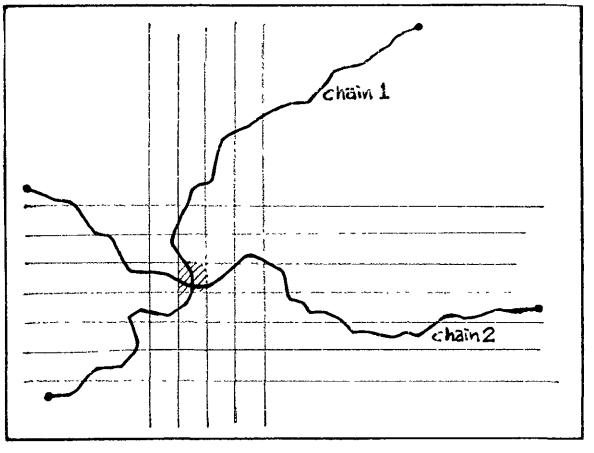

Figure 1. The grid isolates potentially intersecting edge segments.

Polygon Formation. Here we connect the chains to form polygons of the output map. The steps involved are the following:

2.1 For each chain, identify the incident nodes at both head and tail, and calculate the incident angles. Enter the facts as *incidence(node,chain_incidence,angle).*

2.2 At each node, sort the incident chains into proper cyclic order. For each node *N*, enter the facts as *node(N,[C1,C2,...])* for chain incidences *C1,C2,...* sorted in order.

2.3 Each consecutive pair of incident chains identifies a corner of an output polygon; enter the facts as *linkage([C1,C2]), linkage([C2,C3]), . ., linkage([Cn,C1])*, one for each pair of adjacent chains.

2.4 Link up the linkage facts in proper cyclic order. For example, connect *linkage([C1,C2])* and *linkage([C2,C3,C4])* to form *linkage([C1,C2,C3,C4])*. Each complete cycle, such as *linkage([C1,C2, . ,C1])* identifies a polygon.

Figure 2 illustrates these steps. We form all the polygons with only local operations.

Overlay Identification. For each output polygon, we need to determine the two polygons, one from each input map, which intersect to form it. We observe that there are two kinds of output polygons: If the boundary chains of output polygon C involve the chains of two different polygons A and B, one from each map, C is $A \cap B$. Otherwise, all the boundary chains around C must come from the same polygon, A which is completely contained in a polygon B in the other input map. Then C is $A \cap B$. To determine B, we can search the neighbors of C and their neighbors, and so on. The search fails only when the two input maps are not involved in any chain intersection.

EXACT RATIONAL ARITHMETIC

Rational arithmetic has been in use in many symbolic mathematics computation systems, most notably MACSYMA (Macsyma 83), which is in Lisp. The Unix system also provides a library of multiple precision integer arithmetic in C (Sun 86a), and serveral tools are available for calculations using rational arithmetic (Sun 86b). We developed a package for exact rational arithmetic in Prolog (Wu 86). In exact rational arithmetic, we evaluate an expression to a fraction, of both denominator and numerator as integers with virtually no overflow limit. Since Prolog allows operator overloading, the syntax for arithmetic expression remains unchanged. Installation of the package in an existing program is relatively simple.

Stability and Special Cases

Numerical inaccuracy has long been a problem with geometric computation, since it leads to topological inconsistency. With rational arithmetic we are able to circumvent problems of arithmetic inaccuracy arising from

(1) polygon overlay example
(2) nodes and chains after splitting intersecting chains
(3) form linkage record for each polygon corner at every node
(4) connect linkages to form polygons

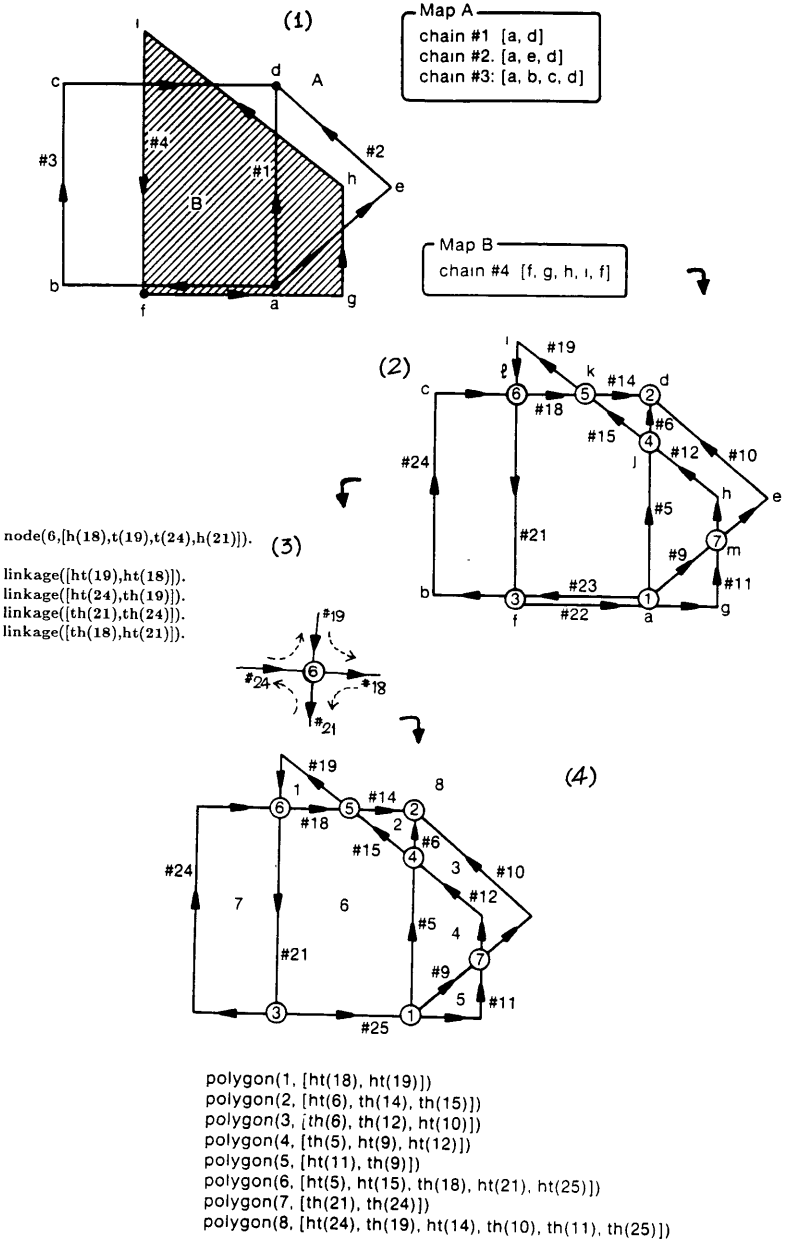

Figure 2. The process of linking chains to form polygons.

discretization errors. Given the coordinates in rational numbers, the coordinates of the intersection point are always rational, since the intersection point between two line segments is the solution to a linear equation with rational coefficients. Hence we can guarantee stability in the numerical computation for geometric intersections in the polygon overlay process.

Since we can preserve numerical accuracy in our calculations, we can also properly identify the special cases of chain intersections including touching and partially overlapping chains. We limit our handling of special cases to only the primitive operations, which in our case is in intersecting edge segments. We check absolute equality instead of setting a tolerance to identify cases of the end point of an edge segment lying exactly on another edge segment, as well as intersection between colinear edge segments. Thus we consider two edge segments not intersecting if they overlap exactly, and we can identify overlapping chains when forming polygons since they have the same incident angle at their beginning and ending nodes.

Coincidental Input Data and Slivers

Although rational arithmetic offers total accuracy, a realistic problem with map data is coincidental input. Two input maps may have data values of the same feature only approximately equal. As a result, the polygon overlay process generates an output map with sliver polygons which have to be removed. Goodchild studied the problem of slivers and established a measure of the number of sliver polygons related to number of edge segments in the coincidental input data (Goodchild 77). We are developing rules to automatically recognize and remove these sliver polygons. We outline our approach below:

Recognizing Slivers A sliver polygon has a small area and has few bounding edges. We identify three kinds of sliver shapes: a rounded polygon, elongated strip, and a crooked strip. Figure 3 illustrates the different kinds of slivers. They can be recognized by their small area, a small minimum diameter, or a small ratio of area to that of its convex hull.

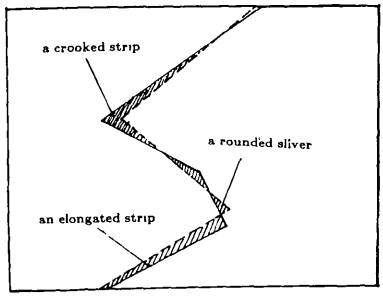

Figure 3. Three kinds of slivers.

Sliver Removal. We remove a sliver by coalescing it with one of its neighboring polygons which is not a sliver. This would avoid coalescing slivers to form a non-sliver polygon. A polygon is coalesced to its neighbor by removing the boundary chain in common, and updating the adjacent polygon fields in the remaining boundary chains.

SOME PRELIMINARY RESULTS

The polygon overlay system is implemented using C-Prolog version 1.5, an interpreter written in C. The system runs on SUN 2/170 machine running Unix 4.2 bsd from SUN microsystems release 3.0. Figure 4 shows a test run of two reduced maps with a total of 1720 vertices and 1826 edge segments in 328 chains. The adaptive grid system casted a grid of 116×100 cells onto the scene. 714 pairs of edges were examined and 118 pairs actually intersected. The system uses approximately 11 CPU hours to complete the entire overlay process.

Figure 4. Test example with USA vs USA.

CONCLUSION

We have presented a polygon overlay system developed in Prolog. The system decomposes the complicated process of polygon overlay into various stages. This decomposition allows us to use only simple data structures and mostly local processing operations. Prolog offers a relational database for geometric entities stored as Prolog facts, and a logic programming approach to the encoding of geometry algorithms for data processing. An exact rational arithmetic package enables us to preserve numerical accuracy in calculating intersections. We can then guarantee topological consistency, and properly identify and handle special cases such as touching and overlapping chains. We have also outlined our strategy to remove sliver polygons due to coincidental input data.

ACKNOWLEDGEMENT

This work is supported by the National Science Foundation under grant No. ECS-8351942.

REFERENCES

Chrisman, N.R 1976, "Local versus Global the Scope of Memory Required For Geographic Information Processing," *Internal Report 76-14*, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, Massachusetts.

DeBerry, T 1979, *Polygon Information Overlay System User's Manual*, Environmental Systems Research, Inc., Redlands, California.

Eastman, C.M. and Yessios, C.I. 1972, "An Efficient Algorithm for Finding the Union, Intersection, and Differences of Spatial Domains," Dept of Computer Science, Carnegie-Mellon University.

Ferguson, H R 1973, "Point in Polygon Algorithms," *Urban Data Center Technical Report*, University of Washington, Seattle

Franklin, W R 1972, "ANOTB Routine to Overlay Two Polygons," *Collected Algorithms*, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, Massachusetts

Franklin, W.R 1980, "A Linear Time Exact Hidden Surface Algorithm," *ACM Computer Graphics*, Vol 14, No 3, pp.117-123.

Franklin, W.R. 1983, "A Simplified Map Overlay Algorithm," *Proc. Harvard Computer Graphics Conference*, Cambridge, Massachusetts.

Franklin, W R. 1983, "Adaptive Grids for Geometric Operations," *Proc. 6th International Symposium on Automation in Cartography*, Ottawa, Ontario, pp.230-239.

Franklin, W.R. 1984, "Cartographic Errors Symptomatic of Underlying Algebra Problems," *Proc of 1st International Symposium on Spatial Data Handling*, Zurich, Switzerland, pp 190-208

Franklin, W.R., Wu, P Y F., Samaddar, S and Nichols, M 1986, "Prolog and Geometry Projects," *IEEE Computer Graphics & Applications*, Vol 6, No.11, pp 46-55

Freeman, H. and Shapira, R 1975, "Determining the Minimum Area Encasing Rectangle for An Arbitrary Closed Curve," *Communications of ACM*, Vol 18, No 7, pp 409-413.

Gonzalez, J C , Williams, M H and Aitchison, I E 1984, "Evaluation of the Effectiveness of Prolog for a CAD Application," *IEEE Computer Graphics & Applications*, Vol 4, No 3, pp 67-75

Goodchild, M F. 1978, "Statistical Aspects of the Polygon Overlay Problem," *An Advanced Study Symposium on Topological Data Structures and Geographic Information Systems*, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, Massachusetts.

Guevara, J A. 1983, *A Framework for the Analysis of Geographic Information System Procedures The Polygon Overlay Problem, Computational Complexity and Polyline Intersection*, Ph.D. Thesis, State University of New York at Buffalo, Buffalo, New York

Little, J.J. and Peucker, T.K. 1979, "A Recursive Procedure for Finding the Intersection of Two Digital Curves," *Computer Graphics and Image Processing*, Vol.10, pp 159-171

Macsyma Group. 1983, *MACSYMA Reference Manual*, Version 10, Vol.II, MIT Press, Cambridge, Massachusetts.

Pereira, F. (ed) 1986, *C-Prolog User's Manual*, Department of Architecture, University of Edinburgh, Edinburgh, U.K

Preparata, F.P. and Shamos, M I. 1985, *Computational Geometry*, Springer-Verlag, New York.

Reed, C.W. 1982, *Map Overlay and Statistical System User's Manual*, Western Energy and Land Use Team, U.S. Fish and Wildlife Services, Fort Collins, Colorado.

Shamos, M.I. and Bentley, J.L 1976, "Optimal Algorithms For Structuring Geographic Data," Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania

Sun Microsystems 1986, *Commands Reference Manual*, Revision G of 17, Part No: 800-1295-02, Sun Microsystems, Inc., Mountain View, California, pp.18-19,82-83.

Sun Microsystems. 1986, *UNIX Interface Reference Manual*, Revision A of 17, Part No: 800-1341-02, Sun Microsystems, Inc , Mountain View, California, pp.270-271.

Swinson, P.S.G 1982, "Logic Programming: A computing Tool for the Architect of the Future," *Computer Aided Design*, Vol.14, No 2, pp 97-104.

Swinson, P.S.G. 1983, "Prolog A Prelude to a New Generation of CAAD," *Computer Aided Design*, Vol 15, No.6, pp 335-343.

Teng, A.T., Joseph, S.A. and Shojaee, A.R. 1986, "Polygon Overlay Processing: A Comparison of Pure Geometric Manipulation and Topological Overlay Processing," *Proc 2nd International Symposium on Spatial Data Handling*, Seattle, pp.102-119.

Tomlinson, R.F , Calkins, H W. and Marble, D.F. 1976, *Computer Handling of Geographical Data*, UNESCO Press, Paris.

White, D. 1978, "A New Method of Polygon Overlay," *Harvard Papers on Geographic Information Systems*, Vol.6, Harvard University, Cambridge, Massachusetts

Wu, P.Y.F. 1986, "Two Arithmetic Packages in Prolog: Infinite Precision Fixed Point and Exact Rational Numbers," *Technical Report IPL-TR-082*, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York.