

## COMPUTATIONAL GEOMETRY AND PROLOG

Wm. Randolph Franklin  
Electrical, Computer, and Systems Engineering Dept.  
Rensselaer Polytechnic Institute  
Troy, New York, USA, 12180.

Telephone: (518) 266-6077  
Telex: 646542 RPI TRO  
Arpanet: WRF%RPI-MTS.Mailnet@MIT-Multics

## ABSTRACT

Prolog is a good tool for implementing computational geometry algorithms. This paper discusses its advantages and disadvantages, and gives examples, including linking chains together, boolean operations on polygons, and cartographic map overlay and reduction. An implementation of polygon intersection proves Prolog to be much more compact and easier to use than Fortran.

This material is based upon work supported by the National Science Foundation under grant no. ECS-8351942, the Rome Air Development Center, contract number, F30602-85-C-0008, subcontract 353-9023-7, and by the Data System Division of the International Business Machines Corp.

## INTRODUCTION TO PROLOG

Prolog is a so-called fifth generation logic programming language based on Horn clauses. The standard Prolog reference is Clocksin and Mellish [8], accompanied by the exercises in Perreira [9]. Its chief characteristics are as follows.

a) Prolog is a declarative, rather than a procedural, language. Essentially, a formula that the solution must satisfy is stated, and the system generates possible solutions and tests them until a good one is found. Nevertheless, in real Prolog programs, there are some procedural steps, such as the assertion of facts, and I/O.

b) There are no assignments of values to variables, except the binding (unifying) of two formulae. If one formula is a free variable, and the other a known quantity, then the effect is similar to an assignment, except that the bound variable cannot have its value changed, except as described below.

c) Prolog proceeds by finding facts in its database that satisfy its current formula. If part of the formula fails, then the system backtracks, undoes bindings, and matches another fact.

d) There is only one data structure, the list, as in Lisp. The major control structure is the recursive procedure. This effects the same result as iteration in most other languages.

e) Prolog is often combined with other languages such as Lisp so that assignment, iteration, and other features can be used. Of course a Lisp assignment executed from Prolog does not get undone on backtracking. Another major limitation of Prolog is that it is not self-referential in the same sense as Lisp: it is difficult for a Prolog procedure to create another Prolog procedure.

Some other implications of automated deduction are given in Goos [17, 18]. Swinson [22, 23, 24] introduces the use of Prolog in architectural design.

The examples in this paper were implemented in Salford Prolog [21] on a Prime 750 computer in the Image Processing Lab at Rensselaer Polytechnic Institute.

## COMPUTATIONAL GEOMETRY

Intuition versus implementation

Computational geometry is concerned with spatial relationships and geometric coincidences, such as which pairs of a set of objects intersect. There is frequently a wide gap between intuition and implementation. Consider for example

performing boolean operations on two polygons, A and B.

When the intuitive algorithm given in the section Boolean Combinations of Polygons is implemented in Fortran by a typical graphics programmer, it will require 500 to 1000 lines of code, although it can be explained to another person in a few minutes. While this is true throughout computer science - people joke about the need for a DWIM (Do What I Mean) machine - nowhere is it more true than for geometry algorithms.

Algorithms which are difficult to understand, unlike the above, can be close to impossible to implement. An example is the recursive construction of a Voronoi diagram.

### Special Cases

Special cases form another problem with graphics implementations. When all the special cases for the polygon combination problem are considered, the number of lines of code rises to 1000 to 2000. A simple example shows the problem. We wish to determine whether two finite line segments intersect. What do we do if the endpoint of one line lies on the other line? Note that in software engineering terms, this is a functional specification problem, not a design or implementation one. A failure to realize this is common and serious.

How this special case is handled does not matter much except that the line intersection routine will be incorporated into other routines, such as one to determine whether two chains of line segments intersect. The hope is that deciding the low level special case properly will mean that the high level algorithm will automatically work correctly. For example, we might define the half plane on one side of each line segment positive, and assume that a line lying on a line lies on the positive side. Then if an endpoint of one chain lies on an edge of another, then the chain intersection will automatically be correct.

However, the above fails if a vertex of one chain coincides with a vertex of another. It can be proved by exhaustion that there is no way to handle the special case of coincident vertices when two lines intersect so that two intersecting chains will be processed properly. The only solution is for the chain intersection algorithm to consider the line intersection special cases itself, at a considerable increase in complexity.

The above simple problem has been considered in some detail since it illustrates quite well the problem of special cases. Special cases in nontrivial problems are that much worse, and they occur everywhere. The cases that occur in polygon combination include coincident vertices, collinear edges, a vertex of A on an edge of B, and multiple components that may or may not intersect the other polygon.

## SIMPLE PROLOG EXAMPLE - LINKING CHAINS

The thesis of this paper is the Prolog is an excellent tool for implementing computational geometry algorithms. Its power will be illustrated by a simple problem, that of linking isolated pieces of a into a chain. The data structure is a set of facts, one per chain piece:

```
chain(end1, end2, list)
```

'end1' and 'end2' are the names of the vertices at the two ends of the chain. 'list' is a list of all the vertices in the chain after end1, up to and including end2. This definition of list simplifies the algorithm. If the chain is only one edge, then list = [end2]. The complete Prolog program to combine chains segments into longer chains as long as this is possible is this:

```
join1 :-      /* Combine 2 chains
               chain(A,B,L1),
               chain(B,C,L2),
               retract(chain(A,B,L1)),
               retract(chain(B,C,L2)),
               append(L1,L2,L3),
               assert(chain(A,C,L3)).

joinall :- join1, fail.
joinall.

chain(a,b,[b]).          /* Sample data
chain(c,d,[d]).
chain(b,c,[c]).
```

Join1 searches for two chains where the ending vertex of the vertex equals the starting vertex of the second. This search is built into Prolog. The lists of vertices for the chains are combined, then these two chains are deleted from the database, and the new chain added.

Joinall repeats join1 as long as join1 can find two chains to combine. After it is finished, the only remaining fact concerning chains is

```
chain(a,d,[b,c,d]).
```

This program, which is complete, compares quite favorably with, a Fortran program to accomplish the same goal. We will now consider in detail some of Prolog's advantages and disadvantages for computational geometry.

## DETAILS OF PROLOG ADVANTAGES AND DISADVANTAGES

### High Level Language

Prolog is a good high level language. It has dynamic data structures, lists, and the code and data can be interchanged.

Another HLL advantage is that Prolog is extensible: we can define new data structures and operators. This allows the creation of more powerful virtual machines. This can improve implementations of geometry algorithms that are bedeviled by numerical inaccuracies. For example, if a point is slightly inside a polygon, and you rotate the scene including the point and the polygon, then the point might now be outside. Also, although floating point numbers are a model for the real number system, almost none of the real field axioms are satisfied. There are computers, for example, where addition is not commutative. This lack of correspondence between what the user expects and what actually happens, is in SWE terms, a FAILURE. Using Prolog, a rational number package can be implemented with which to perform the arithmetic.

### Pattern Matching

Much of geometry consists of processing patterns. If you split a line where another line intersects it, then you are pattern matching for the intersection of two lines. This matches the intuitive way people think about geometry.

In the chain linking example given above, we wish to search for the pattern which is two chains that can be linked, and then link them.

### Unification

This is equivalent to strong connectivity in graph theory. This operation can be used in graphics to determine the connected components of a graph as follows.

- a) Associate a different free variable with each vertex of the graph.
- b) Process each edge of the graph by unifying the free variables corresponding to the vertices at the ends of the edge.

After all the edges have been processed, there remains exactly one free variable for each separate component of the graph. When the free variable corresponding to some vertex is bound to a value, such as that vertex's name, then all the variables corresponding to vertices in the same component are simultaneously bound. If this is repeated until there are no more free variables, then the graph components are determined.

## Data Structures

Pure Prolog, at least, lacks the random addressing of local data. Thus the only efficient way to simulate an array is to implement balanced trees. This adds a log factor to the execution time.

## Efficiency

There are two considerations here. First, a primitive operation in Prolog, such as  $A=B$ , can take an arbitrarily large time to execute, depending on the size of the two expressions being unified. This makes it more difficult to determine bounds for the execution time.

Second, defining algorithms declaratively, which Prolog executes with an exhaustive generate and test procedure, clearly hurts efficiency. Although it is simple to write a procedure that is satisfied for all intersections of two edges in the database, this will require time  $= O(N^2)$  where  $N$  = number of edges in the database.

There are two solutions to this, at least in the geometric case. The first is to impose a second data structure such as an adaptive grid. Franklin has implemented algorithms where this can be quite efficient for geometric intersection problems. [11] gives a hidden surface algorithm that when implemented on a Prime 500, a midicomputer, could determine the visible arcs of 10,000 random circles, packed ten deep, in 383 seconds. These techniques would transfer to Prolog. An even more robust data structure is a k-d tree invented by Bentley, [4]. However, these techniques are retreating somewhat from the declarative form of Prolog since they mean giving a detailed procedure.

The more abstract answer is to notice that Prolog implementations can be good at using hashing to test for equality of the arguments of isolated facts. That is, given a database with facts of the form

person(height, weight).

they can return all facts matching

person(H, 200).

in constant time per fact returned. It is possible to imagine how searches such as

person(H1,W), person(H2,W)

might be performed in time proportional to the size of the data returned. However, what is really needed for geometry, and what no Prolog implementation can do is to execute the following search

in  
req

amo  
wil  
rul  
hav

Sof

dif  
The  
the  
sep  
hea

con

His

bo  
di

co  
in  
or  
re

po  
Ho  
is

ge  
ir

in  
"  
re  
a



person(H,W),  $H > 5$ ,  $H < 6$ ,  $W < 200$ .

in time linear in the number of facts returned. This would require that k-d trees be used in the search mechanism.

Nevertheless, with Prolog, the user has the choice of the amount of efficiency desired. A purely declarative procedure will be easy to code, but slow to execute, while as procedural rules are added, the execution becomes faster. The user does not have this choice in many other languages.

#### Software Engineering Considerations

Prolog lacks certain nesting facilities that may make it difficult to properly modularize and structure large programs. The definition of a procedure is global; it is not restricted to the domain of another procedure. It is impossible to have separate domains or subdomains of facts apart from the global heap.

### BOOLEAN COMBINATIONS OF POLYGONS

In this paper, our big example will be a polygon boolean combination algorithm, so a review of this problem is in order.

#### History


There have been many solutions to the polygon and polyhedron boolean combination problem, each with various advantages and disadvantages. Some of them are:

Eastman and Yessios [10] give a general algorithm for combining 2-D polygons. It works by finding all the intersections between the polygons' edges, and then "threading" or traversing around the pieces of edges to determine the resulting algorithm.

Maruyama [19] gives a procedure for determining whether two polyhedra intersect by comparing the faces pair by pair. However, he does not determine the intersection, only whether it is null.

Baer, Eastman, and Henrion [1] give a good summary of geometric modelling systems, which "shape operations" (i.e. intersection etc.) they perform, and how they do it.

Tilove [25] considers important questions of what intersection and so on mean in the abstract, and introduces "regularized set operators" to answer them. He also gives recursive methods of intersection and union of objects defined as as combinations from a small family of primitives. These methods



are used in P.A.D.L., one of the best known geometric processors. P.A.D.L. also contains a non-CSG polyhedron combination algorithm that has been presented in short courses.

Boyse [5] gives an algorithm for determining whether two objects interfere, where one of the objects can be moving along a straight or circular trajectory. His objects are composed of vertices, straight edges, and flat faces, so for two objects to interfere it is necessary and sufficient for an edge of one object to pass through a face of the other object. This he tests for. However, this does not extend to producing the intersection.

If one of the objects is a convex polyhedron, then we can use the fact that it is the intersection of a number of semi-infinite half-spaces, one for each face, by intersecting them against the other polyhedron, one by one. This is easier if the other polyhedron is also convex. However, the only way that this method generalizes to non-convex objects is to partition them into convex pieces, which increases the complexity.

Baumgart [2, 3] has a good geometric manipulation system with polyhedron combination operators, using a different algorithm. Braid [6] has another polyhedron combination algorithm. Parent [20] describes another one.

Another independent similar algorithm is described in Turner [26]. Again, it does not include a means of handling complex objects. It has been implemented, but cannot handle two objects with a common face, or an edge of one lying in a face of the other.

A similar polyhedron boolean combination algorithm is a part of the design of the Kepler geometric manipulation system described in [12, 13]. An earlier algorithm for polygons was implemented in 1973. A similar planar graph overlay algorithm is described in [14].

Finally, Weiler [27] gives an excellent polygon comparison method that clips two polygons against each other. Its use of a graph data representation simplifies matters, and can compare concave polygons with holes.

#### Prolog Algorithm

A set theoretic definition of  $C = A \text{ intersect } B$  is:

$$C = \{p \mid p \text{ in } A \ \& \ p \text{ in } B\}$$

This is not directly implementable if the universe has an infinite measure. A CSG (Constructive Solid Geometry) representation merely avoids the problem until it must determine whether the resulting object is empty, at which time it must perform some approximation to the algorithms described below.



We give an intuitive definition of C that includes all special cases and is implementable. It handles collinear edges, many edges incident on the same vertex, a vertex in the middle of another edge, a polygon with multiple separate components and holes with islands, and an infinite polygon that includes the whole plane except for a finite region. It is as follows.

a) Determine all the intersections of edges of A with edges of B.

b) Split all the edges at these intersections into smaller segments that do not intersect. If an endpoint of one edge lies on another, then split the other edge. This includes the case of two collinear edges; if either contains an endpoint of the other in its interior, then it will be split there.

c) Determine the relation of each segment to the other polygon. There are six cases:

i) A segment from an edge of polygon A may be inside polygon B.

ii) It may be outside B.

iii and iv) Ditto for segments of polygon B.

v) The segment results from an edge of A being collinear with an edge of B, and is part of an edge of both A and B. Both polygons are on the same side of the segment.

vi) As before, except that A and B are on opposite sides of the segment.

d) Use a decision table to select the segments appropriate to the boolean operation desired. For intersection, we want the segments from only cases (i), (iii), and (v).

This produces the set of segments in the output polygon. A planar graph traversal may be done to link them up, but this is not necessary since most desired operations, such as point inclusion testing, area measurement, cross-hatching, and further boolean operations, do not require the global topology.

#### Status

The program is implemented and working except for the collinear edges. In contrast to Fortran, the Prolog version has only 151 executable lines of Prolog. The lines are of a natural length, not packed to column 80.

U

## CARTOGRAPHIC MAP OVERLAY AND REDUCTION

Franklin [15] describes the problem of map overlay in cartography. Here a difficult algorithm combines with numerical inaccuracies. Peter Wu is implementing a solution in Prolog where the two difficulties are decoupled. The algorithm is being implemented with an infinite precision rational number package in Prolog. Next the output will be reduced to floating point format with an expert system.

This reduction process involves moving the vertices to legal coordinate values one by one. After each vertex is moved, the topology of the map around it is checked for correctness and consistency. If an error has been produced, then the vertex must be moved to another legal place, or something else must be moved to correct the topology. In extreme cases, the topology must be changed while keeping its consistency. For example, a very small polygon might be combined with a neighbor.

## SUMMARY

The pattern matching, backtracking, capabilities of fifth generation languages such as Prolog are particularly useful in geometry. Their correlation with the way that people think about geometry and their abstraction of unnecessary details allows us to spend more time on the creative aspects of the algorithms, which makes larger systems and more difficult algorithms practical to implement.

## REFERENCES

- [1] A. Baer, C. Eastman, and M. Henrion, "Geometric Modelling: A Survey", Computer Aided Design 11 (5), Sept. 1979.
- [2] B.G. Baumgart. GEOMED: Geometric Editor, Stanford University STAN-CS-74-414, Also available as NTIS AD-780 452, (May 1974),
- [3] B.G. Baumgart. Geometric Modelling for Computer Vision, Stanford University Artificial Intelligence Memo AIM-249, (Oct. 1974),
- [4] J.L. Bentley and M.I. Shamos. "Divide And Conquer in Multidimensional Space", Proc. 16th Annual IEEE Symposium on the Foundations of Computer Science, (1975), pp. 220-230

[5] J.W. Boyse. "Interference Detection Among Solids and Surfaces", Comm. ACM 22 (1), Jan. 1979, pp. 3-9.

[6] I.C. Braid. "The Synthesis of Solids Bounded by Many Faces", Comm. ACM, (1975).

[7] K.L. Clark and S.-A. Tarnlund. Logic Programming, (1982), APIC Studies in Data Processing No 16, Academic Press.

[8] W.F. Clocksin and C.S. Mellish. Programming In Prolog, (1981), Springer-Verlag, New York.

[9] H. Coelho, J.C. Cotta, and L.M. Pereira. How to Solve it With Prolog, 2nd edition, Ministerio da Habitacao e Obras Publicas, Laboratorio Nacional de Engenharia Civil, Lisboa, (1980).

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

[11] W.R. Franklin. "An Exact Hidden Sphere Algorithm That Operates In Linear Time", Computer Graphics and Image Processing 15, 4, (April 1981), pp. 364-379.

[12] W.R. Franklin. "3-D Geometric Databases Using Hierarchies of Inscribing Boxes", Proceedings of the 7th Canadian Man-Computer Conference, (10-12 June 1981), Waterloo, Ontario, pp. 173-180.

[13] W.R. Franklin. "Efficient Polyhedron Intersection and Union", Proc. Graphics Interface'82, Toronto, (19-21 May 1982), pp. 73-80.

[14] W.R. Franklin. "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,

[15] W.R. Franklin. "Cartographic Errors Symptomatic of Underlying Algebra Problems", Proc. International Symposium on Spatial Data Handling, vol. 1, (20-24 August 1984), Zurich, Switzerland, pp. 190-208.

- [16] J.C. Gonzalez, M.H. Williams, and I.E. Aitchison. "Evaluation of the Effectiveness of Prolog for a CAD Application", IEEE Computer Graphics and Applications, (March 1984), pp. 67-75.
- [17] G. Goos and J. Hartmanis. Lecture Notes in Computer Science 87: 5th Conference on Automated Deduction, (1980), Springer-Verlag, New York.
- [18] G. Goos and J. Hartmanis. Lecture Notes in Computer Science 138: 6th Conference on Automated Deduction, (1982), Springer-Verlag, New York.
- [19] K. Maruyama. "A Procedure to Determine Intersections Between Polyhedral Objects", Int. J. Comput. Infor. Sc. 1 (3), 1972, pp. 255-266.
- [20] R.E. Parent. "A System for Sculpting 3-D Data", Computer Graphics (ACM) 11 (2), (Summer 1977), pp. 138-147.
- [21] University of Salford. LISP/PROLOG Reference Manual, (March 1984).
- [22] P.S.G. Swinson. "Logic Programming: A Computing Tool for the Architect of the Future", Computer Aided Design 14, (2), (March 1982), pp. 97-104.
- [23] P.S.G. Swinson, F.C.N. Periera, and A. Bijl. "A Fact Dependency System for the Logic Programmer", Computer Aided Design 15, (4), (July 1983), pp. 235-243.
- [24] P.S.G. Swinson. "Prolog: A Prelude to a New Generation of CAAD", Computer Aided Design 15, (6), (November 1983), pp. 335-343.
- [25] R.B. Tilove. "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", IEEE Trans. Comput. C-29 (10), 874-883, (October 1980).
- [26] J.A. Turner. An Efficient Algorithm for Doing Set Operations on Two- and Three- Dimensional Spatial Objects, Architectural Research Laboratory, University of Michigan.

[27] K. Weiler. "Polygon Comparison Using a Graph Representation", ACM Computer Graphics ACM Computer Graphics 14 (3), (Proc. SIGGRAPH'80), (July 1980), pp. 10-18.

[28] F. Yamaguchi and T. Tokieda. "A Unified Algorithm for Boolean Shape Operations", IEEE Computer Graphics and Applications 4, (6), (June 1984), pp. 24-37.

U