In Advanced Computer Graphics
Edited by Tosiyasu Kunii

Geometry in Prolog

Wm. Randolph Franklin*, Margaret Nichols**, Sumitro Samaddar and Peter Wu
Rensselaer Polytechnic Institute, Troy, NY 12180, USA

ABSTRACT

The Prolog language is a useful tool for geometric and graphics implementations because its primitives, such as unification, match the requirements of many geometric algorithms. We have implemented several problems in Prolog including a subset of the Graphics Kernel Standard, convex hull finding, planar graph traversal, recognizing groupings of objects, and boolean combinations of polygons using multiple precision rational numbers. Certain paradigms, or standard forms, of geometric programming in Prolog are becoming evident. They include applying a function to every element of a set, executing a procedure so long as a certain geometric pattern exists, and using unification to propagate a transitive function. Certain strengths and weaknesses of Prolog for these applications are now apparent.

Keywords: Prolog, Geometry, Graphics Kernel Standard

INTRODUCTION

The fifth generation logic programming language Prolog, Clocksin (1981), Pereira (1980) appears appropriate for research in geometry and graphics. Some examples of its use in architectural design are given in Swinson (1982, 1983a, 1983b). Its use in CAD has been evaluated in Gonzalez (1984). Constructing geometric objects from certain constraints is described in Brüderlin (1985). Over the past two years, the authors of this present paper have implemented several geometric and graphic problems in Prolog using assorted machines. This paper describes the experiences, including some paradigms of programming that have appeared useful, and finally listing the advantages and disadvantages of Prolog that we have experienced.

IMPLEMENTATIONS

Over the last two years we have implemented several graphics and geometric algorithms in Prolog, totally a few thousand lines of code, using four different Prolog interpreters on four different computers. The systems include:

Machine	Operating System	Prolog Version
IBM 3081	Michigan Terminal System	York (U.K.)
IBM 4341	CMS	Waterlog
Prime 750	Primos	Salford
VAX 780	Unix bsd 4.3	UNSW

*Until June 1986: Computer Science Division, 543 Evans, University of California, Berkeley, CA 94720, USA. Arpanet: wrf@ernie.Berkeley.EDU
**Current address: North American Philips Lighting Co., Bloomfield NJ.

The implementations include:

- Graphics Kernel Standard subset
- Convex Hull
- Planar Graph Traversal
- Big Rational Numbers
- Polygon Intersection
- Recognizing Groupings of Objects

Graphics Kernel Standard Subset

This graphics addition to Prolog was implemented by Nichols (1985) on an IBM 4341 using Waterlog as described in Roberts (1984), under the CMS operating system. This allowed us to draw lines and so on on the 3270 graphics CRT from a Prolog program. The major problems and design decisions were as follows.

1. Waterlog, like most Prologs, lacks floating point numbers, and even four byte integers. (The latter was undocumented; large integers just didn't work.) However it has the powerful capability to be linked to programs in other languages such as Fortran. Thus we implemented a real number in Prolog as a data structure of the form real(A,B) where A and B are Prolog integers holding the upper and lower halfword, respectively, of the integer. The user never looks at A and B, but accesses the real numbers via the following Prolog procedures:

```
add(A,B,C)
dev(A,B,C)
exp(A,B,C)
itor(I,A)
mult(A,B,C)
rtoi(A,I)
subt(A,B,C)
```

   that perform the obvious operations on real numbers in the stated form. Because of this modularity, the user never uses the actual form of the real number in Prolog.

2. We used the available Fortran graphics package, GSP, as little as possible, preferring to do scaling and so on in Prolog for portability reasons. GSP's limitations also prevented us from implementing variable beam intensities and the full GKS text orientation capabilities.

3. Both the long GKS names and the six character Fortran bindings are available to the user. Thus he can say either open_gks(1,1) or gopks(1,1).

4. We implemented two classes of lines: *permanent* and *backtrackable*. If the Prolog procedure that drew a backtrackable line was backtracked over, then the line would be erased. This used a feature of the graphics package GSP.

5. The lack of a *current cursor* concept in GKS eased our task since the only way to store global information such as that in Prolog would be to retract and assert it after every cursor movement. This would have been slow.

6. This is a sample Prolog routine to draw a line between two points.

```
draw_to(Point_x.Point_y) :-
    erase_on_backtrack(yes),
    ifelse(deferral_mode(1, asti),
        draw1(Point_x, Point_y, 1, Key),
        draw1(Point_x, Point_y, 0, Key)),
    (True; (erase(Key), !, fail)).
```

```
draw_to(Point_x.Point_y) :-
    ifelse(deferral_mode(1, asti),
        draw(Point_x, Point_y, 1),
        draw(Point_x, Point_y, 0)).
```

Draw1 is a Fortran subroutine which draws a line and returns a key which can be used to erase that line at a later time. When it is called with zero as its third parameter, the line is immediately drawn on the screen. A one in the third position enters the line segment into the data set, to be drawn when the workstation is updated.

Draw is a Fortran subroutine which draws a line but does not return a key. This means that the line segment cannot be erased.

The clause (true; (erase(Key), !, fail)) has no effect on the rule unless backtracking occurs, because the true goal is the only one that is encountered. However, should the Prolog proof fail somewhere after draw_to is satisfied, causing the rule to be backtracked through, then the clause (erase(Key), !, fail) will be encountered. Its effect is to call erase, a Fortran subroutine which will erase the line segment, and then fail, which will cause the backtracking to continue back through the proof. The cut symbol allows the rule to fail without trying to satisfy itself with the second clause.

7. Some clipping and transformations are also implemented.

8. Problems with Waterlog's not freeing trail space were never fully solved.

9. All the *ma*-level commands were implemented except some inquiry and set commands, the escape command, text rotation, and the inquire text extent command. Here are the 70 Prolog procedures that were implemented.

```
activate_workstation
adjust
allocate_and_initialize_workstation_state_list
bothin
change_state
clip1
clip2
close_gks
close_workstation
deactivate_workstation
deallocate_workstation_state_list
draw_to
error
errormsg
fill_area
find
go_to_workstation
help
initialize_gks_description_table
initialize_gks_state_list
initialize_other_state_lists
initialize_workstation_description_table
inquire_asf
inquire_clipping_indicator
inquire_current_normalization_transformation_number
inquire_current_primitive_attribute_values
inquire_current_individual_attribute_values
inquire_level_of_gks
inside
intersect
line
listadd
listdelete
```

make_adjustments
make_decision
make_transformation
member
move_to
normalize
not_both_in
open_gks
opengks0
open_workstation
plot_point
plot_point2
plot_text
point
points
polyline
polyline_
polymarker
putout
sameside
set_character_height
set_deferral_state
set_erase
set_marker_type
set_workstation_window
set_workstation_viewport
suggest
test_midpoint
test_point
test_points
text
update_workstation
wipe_out_gks_description_table
wipe_out_gks_state_list
wipe_out_other_state_lists
wipe_out_workstation_description_table

These Prolog procedures were supported by several Fortran routines and an Assembler interface.

Convex Hull

This Graham Scan algorithm was implemented by Wu, described in (Franklin and Wu, 1985) on both the IBM 4341, and on the Prime in Salford Prolog (1984). The Salford system allows both real numbers and dynamic linking to Fortran routines. We also tested York Prolog (Spivey, 1983), which is written in Pascal. The York system has the advantage that it is portable to any machine that can compile a thousand line Pascal program that uses four byte integers. Unfortunately this did not include the official Pascal compiler available from Prime. (We have not evaluated third-party Pascal compilers for Prime computers.) We also tested York Prolog on an IBM 3081 running the Michigan Terminal System, but found the other computers' operating systems more flexible and cheaper to use.

Boolean Combinations of Polygons

A program to perform operations such as intersection, union, and difference on two planar polygons was implemented by Franklin and Wu (Franklin and Wu, 1985) on the Prime and IBM 4341. The algorithm was by Franklin (1985). Wu first implemented a package to perform arithmetic using rational numbers in multiple precision. Each number, in life a quotient of an integer numerator and denominator, is implemented as a list of the numerator and denominator. Each of them is a list of groups of the digits of the number. For example, 123456789/987654321 is represented as [[56789, 1234], [54321, 9876]]. Rational numbers are used to avoid roundoff errors, as part of an ongoing investigation into their utility in geometry and the map overlay problem in cartography (Franklin, 1984).

Object Grouping in Photoreconnaissance

Assume that we recognize certain objects in a photograph, and know that they tend to be grouped in clusters. However we might have failed to recognize all of the objects, and some of them might be absent. We have built an experimental system in Prolog to help (Samaddar, 1985).

Planar Graph Traversal

At some point during an object space hidden surface algorithm, Franklin (1980), we have the set of the visible edges and must join them to find the visible polygons. This requires a planar graph traversal, sometimes called a tesselation. For example, in figure 1,

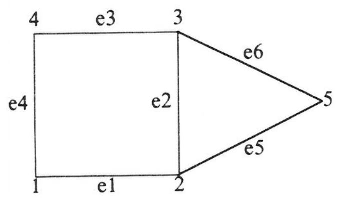


Figure 1: Finding the Faces of a Planar Graph

we are given the vertices and edges in the form

```
vert(vert-name, x-coord, y-coord)
edge(edge-name, first vertex, second vertex, angle)
```

for example

```
vert(v1, 0, 0)
edge(e1, v1, v2, 0)
```

The angle of the edge is supplied because of the difficulty of computing arctangents using only integers. The output is a set of facts of the form

```
polygon([v1, v2, v3, v4])
```

This was implemented in UNSW Prolog (Sammut, 1983) on a Vax.

STRENGTHS AND WEAKNESSES OF PROLOG

Certain advantages and disadvantages of Prolog for graphics and geometric applications are becoming evident from these implementations.

Advantages Of Prolog

- Prolog has same high level advantages of Lisp, as the equivalence of code and data and dynamic data allocation.
- There are the specific advantages of Prolog. Unification makes determining graph connectivity a primitive operation and in general is useful for propagating transitive properties such which occur frequently.

- The pattern matching fits with the form of expression of many algorithms. For example, our polygon combination algorithm proceeds as follows. Whenever the pattern of two edges intersecting, or one edge ends on the interior of another edge, occurs, then retract those edges and assert new smaller edges. When this pattern no longer exists, then we have a superset of the edges in the output polygon.

- Although many of the above features could be implemented in any language that is Turing equivalent, Prolog is somewhat standard so that different researchers can understand and use each others' extensions.

Disadvantages Of Prolog

However, there are some problems with using Prolog for geometry.

- There are software engineering problems with using Prolog for a large project because of its lack of nesting in the program and databases.

- Many geometry algorithms are more natural to a forward reasoning system than a backward reasoning system. That is, we are more likely to want the output from some given input than the reverse.

- The natural way of expressing pattern matching algorithms requires us to modify a database as we are searching through it. Thus in polygon overlay, whenever we find the pattern of two edges crossing, we retract them and assert four new edges. Backtracking and redoing a database that we are modifying does not work on all Prologs.

- In general Prolog in completely unstandardized around the fringes as some tests of cuts in (Moss, 1985) show.

PARADIGMS OF PROGRAMMING

Certain techniques have proven to be generally useful in our implementations, and may be useful to others also. They include the following paradigms.

Set Based Algorithms

Many algorithms such as polyhedron intersection and hidden surface algorithms, Franklin (1982, 1980), are the alternation of two types of steps:

- Applying function to every element of a set, and

- Combining all the elements having a common key.

This is clearly easy in Prolog.

Pattern Matching

The second paradigm uses pattern matching to propagate certain properties. For example, in the planar graph traversal algorithm, the edges around each vertex are found and sorted by the angle at which they leave it. Then the edges around each vertex are paired to form *corners*. These corners can be considered to be fragments of the output polygons. Whenever two fragments exist such that the last edge of one is the same as the first edge of another, then these two fragments are retracted and a single longer fragment asserted. When such a pattern no longer exists, then we have the output polygons.

Unification

Frequently we wish to determine the closure of some transitive property, such as when we are given a set of graph edges edge(u, v), and wish to determine the connected components. We have implemented the following short algorithm that uses unification and the set processing paradigm.

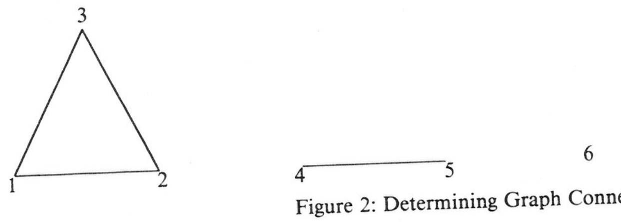


Figure 2: Determining Graph Connectivity

- Create a property list (*plist*) with one record per vertex, and the property of each vertex a free variable. For example in figure 2 we would have [[1,_],[2,_],[3,_],[4,_],[5,_],[6,_]].
- Process the set of edges and for each edge unify the free variable properties of the endpoints. After this we will have [[1,_1],[2,_1],[3,_1],[4,_2],[5,_2],[6,_3]] with one unique free variable per graph component.
- Bind a name identifying each component to the free variables in the list to give something like [[1,first],[2,first],[3,first],[4,second],[5,second],[6,third]].

A longer example of a simple hidden surface algorithm would go as follows.

- Wherever the pattern of two edges' projections' intersecting occurs, split the edges into four smaller edge segments.
- For each edge segment find the set of faces hiding its midpoint. Iff it is empty then the edge segment is visible. Draw them.
- Use a planar graph traversal algorithm such as described above to link the visible edges into polygons.
- For each polygon, find a point inside it and then find the set of faces whose projections contain the projection of that point. Find the closest such face; the polygon came from it. Color the polygon accordingly.

This illustrates all of the paradigms operating together.

ACKNOWLEDGEMENT

This work was supported by the National Science Foundation under grant no. ECS-8351942, the Data Systems Division of the International Business Machines Corporation, and by the Rome Air Development Center under the postdoctoral development program via Syracuse University.

REFERENCES

Bruderlin, Beat (1985) "Using Prolog for Constructing Geometric Objects Defined by Constraints," *Eurocal 85, Conference Proceedings*, Linz, Austria. Institut fü Informatik, ETH Zürich, CH-8092, Zürich, Switzerland

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

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

Franklin, Wm. Randolph (July 1980) "A Linear Time Exact Hidden Surface Algorithm," *ACM Computer Graphics*, vol. 14, no. 3, pp. 117-123. Proceedings of SIGGRAPH'80

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

Franklin, Wm. Randolph (20-24 August 1984) "Cartographic Errors Symptomatic of Underlying Algebra Problems," *Proc. International Symposium on Spatial Data Handling*, vol. 1, pp. 190-208, Zürich, Switzerland.

Franklin, Wm. Randolph and Wu, Peter Y.F. (July 1985) *Convex Hull and Polygon Intersection Implemented in Prolog,* Rensselaer Polytechnic Institute, Troy, NY.

Gonzalez, J.C., Williams, M.H., and Aitchison, I.E. (March 1984) "Evaluation of the Effectiveness of Prolog for a CAD Application," *IEEE Computer Graphics and Applications*, pp. 67-75.

Moss, Chris and Fogel, Earl (June 1985) *Tests to Distinguish Various Implementations of Cut in Prolog,* Imperial College and Logicware Inc.. Reported on Usenet in Net.lang.Prolog, message-id <1742@utecfa.UUCP>.

Nichols, Margaret (August 1985) *The Graphic Kernal System in Prolog,* ECSE Dept., Rensselaer Polytechnic Institute, Masters Thesis, Troy, NY.

Roberts, Grant (1984) *Waterloo Core Prolog Users Manual (version 1.5),* Intralogic Inc., Waterloo, Ont, Canada.

Salford, University of (March 1984) , *LISP/PROLOG Reference Manual.*

Samaddar, Sumitro (August 1985) *An Expert System for Photo Interpretation,* ECSE Dept., Rensselaer Polytechnic Institute, Masters Thesis, Troy, NY.

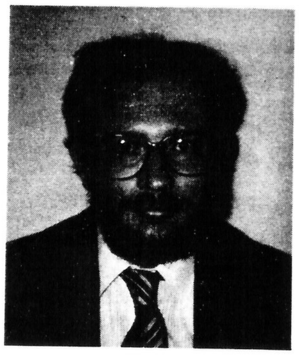
Sammut, Claude (1983) *UNSW Prolog User Manual,* University of New South Wales (Australia).

Spivey, J. M. (March 1983) , *University of York Portable Prolog System (Release 1) User's Guide*, York, U.K..

Swinson, P.S.G. (March 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. (November 1983) "Prolog: A Prelude to a New Generation of CAAD," *Computer Aided Design*, vol. 15, no. 6, pp. 335-343.

Swinson, P.S.G., Periera, F.C.N., and Bijl, A. (July 1983) "A Fact Dependency System for the Logic Programmer," *Computer Aided Design*, vol. 15, no. 4, pp. 235-243.



Wm. Randolph Franklin is an associate professor with the Electrical, Computer, and Systems Engineering Department at Rensselaer Polytechnic Institute. For the 1985-86 academic year he is on sabbatical with the Electrical Engineering and Computer Science Department, University of California at Berkeley. His research interests include graphics, geometry, and artificial intelligence. Franklin received the BSc in computer science from the University of Toronto, and the AM and PhD in applied mathematics (computer science) from Harvard University. He is a member of IEEE, ACM, and SIAM.

Franklin's address until May 1986 is Computer Science Division, Electrical Engineering and Computer Science Department, 543 Evans Hall, University of California, Berkeley, CA 94720.