Computers & Geosciences, Vol. 4, pp. 173–178

0098-3004/78/0601-0173/$02.00/0

AN EFFICIENT POINT-IN-POLYGON ALGORITHM

KENNETH B. SALOMON[1]
Department of Mathematics, California State University, Hayward, California 94542, U.S.A.

(*Received* 14 *September* 1977)

Abstract—An algorithm, and its FORTRAN implementation, are presented which efficiently determine whether a point is interior to a polygonal boundary. As such algorithms form the heart of many geoscience programs concerned with spatial information, it is important that they be efficient in their use of storage and time. A comparison of the algorithm with others obtained from the literature shows that it represents a significant improvement in run-time. An important feature of the algorithm is a user-specifiable function which can be tailored to any particular application so as to maximize run-time efficiency.

Key Words: Algorithm, Subroutine, FORTRAN, Mathematics, Geometry, Polygons.

INTRODUCTION

The need for an efficient algorithm which determines if an arbitrary point is within a polygonal boundary arises frequently in the geological sciences (and in many other applications concerned with spatial information). Such algorithms therefore have been studied extensively (e.g. Aldred, 1972; Anderson, 1976; Burton, 1977; Hall, 1975). Most of the algorithms have two drawbacks: (1) their execution time is proportional directly to the number of polygon sides, and (2) each interrogation is independent of all preceding interrogations. Drawback (1) indicates that although simple boundaries run quickly, complex boundaries absorb large amounts of time; in some applications becoming the single largest run-time factor. (2) indicates that information about the location of a point must be recreated when, in fact, it could be deduced from the outcome of a preceding interrogation.

The algorithm presented here addresses with these two difficulties by (a) preprocessing the boundary polygon so that it is represented by "swaths" rather than in its raw vertex-side form. In most situations this will significantly reduce the number-of-sides dependency of (1). And by (b) providing a user-specifiable swath-selection function which, if judiciously chosen, makes maximum use of the information developed from a preceding interrogation.

PRINCIPLES OF THE ALGORITHM

By this author's empirical evaluation of the various point-in-polygon algorithms obtained from the literature (and confirmed independently by Aldred, 1972), the Jordan line-crossing technique (Spanier, 1966, p. 198) is most efficient on both a run-time and storage-requirement basis. Thus it forms the heart of the algorithm. Anderson (1976) gives a FORTRAN implementation of this algorithm applicable to all polygons not having horizontal sides. This program, termed RAWJORDAN, is modified suitably to handle all polygons.

Our algorithm is not concerned with the original polygon, but with a processed version of it whereby it is represented by a set of horizontal swaths. Each swath contains those sides spanning the interval of the y-axis between two successive vertices when they are ordered by decreasing y-endpoints. For example, the polygon of Figure 1 describes eight swaths. $SWATH_1$ contains sides 5 and 6; $Y\text{-}INTERVAL_1 = [4, 3)$. $SWATH_2$ contains sides 3 and 6; $Y\text{-}INTERVAL_2 = [3, 2.5)$. $SWATH_3$ contains sides 1, 2, 3 and 6; $Y\text{-}INTERVAL_3 = [2.5, 1)$. ... $SWATH_7$ contains sides 10, 9, 8 and 7; $Y\text{-}INTERVAL_7 = [-1, -2)$. $SWATH_8$ contains sides 8 and 7; $Y\text{-}INTERVAL_8 = [-2, -2.5)$.

This representation has several advantages; first, the number of swaths described by a polygon is less than the number of sides; second, horizontal sides, which typically are treated on a special-case basis in line-crossing algorithms, are omitted completely from consideration; lastly and most importantly, whether a point is interior or exterior to the polygon now can be determined simply by

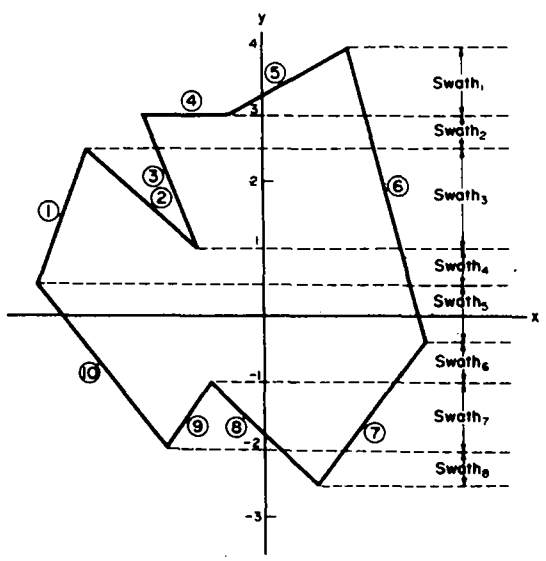

Figure 1. Example of polygon and swaths and y-intervals it describes.

[1]Present address: Mining Technical Service, Utah International Inc. 550 California St., San Francisco, California 94104, U.S.A.

counting the number of polygon segments *within the appropriate swath* intersecting a right-directed ray emanating from the point.

Furthermore, if the polygon segments within a swath are ordered from left-to-right (as in our example), we simply need find the *first* segment producing an intersection in order to resolve the point's location with respect to the polygon by the following

Claim

Let j be the index of the first segment within an ordered swath producing an intersection. Point is in polygon if and only if $j \equiv 0 \pmod 2$.

Proof

Let n be the total number of segments within the swath.

point-in-polygon iff number of line-crossings $\equiv 1 \pmod 2$ by Jordan separation theorem.

$$\text{iff } n - j \equiv 0 \pmod 2$$

since $n - j$ is the number of line-crossings remaining in swath after first determined.

$$\text{iff } j \equiv 0 \pmod 2$$

because $n \equiv 0 \pmod 2$; that is any swath contains an even number of segments by another application of the Jordan separation theorem. Q.E.D.

Hence, only in the worst situation, when the point in question is to the right of all segments in its swath, will a true line-crossing count be necessary. Thus by incurring a (one-time only) overhead cost of preprocessing the polygon, run-time improves in two respects compared to RAWJORDAN; not all sides need be searched for line-crossings, only those within a swath, and not all segments within a swath need be tested, only until the first intersection is determined.

In those applications involving many-sided polygons, the dominant factor in run-time now becomes the swath selection function rather than the polygon's complexity. Thus a particular function can be chosen by the user to exploit any feature of the distribution of the points to be tested against a given polygon. The function shown in the program listing (Appendix) is suitable for a "random" distribution, in which a point is equally likely to fall into any particular swath. If the points to be tested are drawn from a grid and have a predominantly left-to-right testing order, say, then a function which first tests if the current point falls into the same swath as the one on the preceding call, would be an appropriate choice. If the points are drawn from a grid but have a predominantly top-to-bottom order, a function which first tests if the point falls into the next lower swath from the preceding call might be appropriate.

A FORTRAN IMPLEMENTATION AND A COMPARATIVE PERFORMANCE EVALUATION

The Appendix provides a listing of a FORTRAN-IV package implementing the algorithm. These are four subroutines to preprocess the polygon: PREPLY which calls SORT, INCLUD and ORDER; and the modified version of RAWJORDAN, INOUT. The preprocessing establishes the number of y-intervals described by the polygon, INTVLS; the endpoints of these intervals, [YINTVL(I), YINTVL(I + 1)), I = 1, INTVLS; the matrix SWATH, where for I = 1, INTVLS

SWATH(I, 1) = number of polygon segments within the I-th swath

SWATH(I, J + 1) = index of the J-th side from the left contained within the I-th swath, J = 1, SWATH(I, 1);

and the reciprocal of the slope of the nonhorizontal sides, RSLOPE(I). The implementation of the Claim by INOUT is straightforward. However, note that as the first entry in a row of SWATH contains the total of segments within that swath, the running index I then starts at 2. Hence, in terms of the Claim, the value of I upon exit from the intersection-finding loop equals $j + 1$. Hence, point-in-polygon iff $j \equiv 0 \pmod 2$, by Claim, iff $I \equiv 1 \pmod 2$, because $I = j + 1$.

The package, hereafter called FASTJORDAN, was compiled and run against several polygon-point populations on an IBM-370/145 operating under OS/VSI using the G1 (release 2.0) compiler. For comparison, timings of RAWJORDAN and, in one situation, the timing of a line-integral algorithm (Hall, 1975) was obtained. Three polygons were used; 4-sided (P_4), 17-sided (P_{17}) and 84-sided (P_{84}). Two point populations were used; 5000 pseudorandom points (R_{5000}) and a 300×400 grid mesh ($G_{300\times400}$).

All algorithms gave identical results, the only differences were in run-times (Table 1). Three times are given for each example; the overhead time required to prepare the polygon, in the situation of RAWJORDAN this involves determining the clockwise-counterclockwise orientation of the polygon (Salomon, 1977), in the situation of FASTJORDAN, the establishment of matrix SWATH and arrays YINTVL and RSLOPE. Secondly, the total execution time from start to stop. Finally, the net time spent executing the routine determining the location of the points in the population. This last factor is, of course, the best basis for comparison.

We see that the line-integral routine is several orders of magnitude slower than either of the line-crossing techniques and thus is dismissed from further consideration. For the pseudorandom point population, FASTJORDAN is faster, the difference growing as the number of polygon sides increases. This reflects the efficiency of avoiding the need to consider all sides when searching for line-crossings. Finally, on the 300×400 grid, we see that only a slight improvement results from the use of an appropriately biased swath-selection function for the 17-sided polygon, but a significant improvement occurs for the 84-sided polygon. This is occasioned by the fact that the 17-sided polygon describes only 7 swaths whereas the 84-sided polygon describes 69 swaths. Hence the unbiased function is relatively unimportant in the former situation but exacts a large penalty in the latter situation. As the grid points were tested in a predominantly left-to-right order, the following two lines inserted at the top of

Table 1. Run-times of various point-in-polygon routines. Times are minutes-seconds

RAWJORDAN	FASTJORDAN	FASTJORDAN with biased swath selection function	Line integral	Case
0– 0.82	0– 0.94			P_4 vs R_{5000}–prep.
0– 6.37	0– 5.37			P_4 vs R_{5000}–total exec.
0– 2.93	0– 2.10			P_4 vs R_{5000}–net exec.
0– 1.19	0– 1.04		0– 0.00	P_{17} vs R_{5000}–prep.
0–12.73	0– 5.74		1–30.49	P_{17} vs R_{5000}–total exec.
0– 9.23	0– 2.22		1–26.99	P_{17} vs R_{5000}–net exec.
0– 1.69	0– 2.45			P_{84} vs R_{5000}–prep.
0–45.97	0–10.97			P_{84} vs R_{5000}–total exec.
0–41.90	0– 6.11			P_{84} vs R_{5000}–net exec.
0– 1.19	0– 1.04	0– 1.04		P_{17} vs $G_{300\times400}$–prep.
3– 2.98	0–55.58	0–49.72		P_{17} vs $G_{300\times400}$–total exec.
2–49.54	0–41.00	0–35.14		P_{17} vs $G_{300\times400}$–net exec.
0– 1.89	0– 3.76	0– 3.76		P_{84} vs $G_{300\times400}$–prep.
17–36.64	2–52.94	1– 5.55		P_{84} vs $G_{300\times400}$–total exec.
17–21.44	2–36.10	0–48.71		P_{84} vs $G_{300\times400}$–net exec.

the code enclosed in dashes in INOUT would be appropriate:

```
IF(INTVAL.EQ.0. .AND. YP.GT.YINTVL(1))
RETURN
IF(YINTVAL(INTVAL).GE.YP .AND. YP.GE
YINTVAL(INTVAL + 1)) GOTO 300
```

A final comment can be drawn from these results. The preparation time for a many-sided polygon may require several seconds. If in a particular application, boundary polygons will become active and subsequently be replaced by another, the overhead imposed by their preparation may become significant. In extreme situations a mixed-strategy algorithm may be indicated (see Aldred, 1972).

USER INSTRUCTIONS

The package conforms to ANSI FORTRAN-IV standards except in the use of general integer-valued arithmetic expressions for subscripts and the use of string quotes in the two FORMAT statements used to write diagnostic error messages. The envoking program is expected to place the polygon's vertices in X and Y and the number of vertices, not including return to the starting vertex, in NUVERT prior to calling PREPLY. In order to minimize linkage overhead, all parameters for INOUT are passed in COMMON except the coordinates of the point being tested (XP, YP). Thus a typical calling sequence would be

```
COMMON X(100), Y(100), YINTVL(100), INTVLS,
SWATH(100, 25), RSLOPE(100)
LOGICAL INOUT
INTEGER SWATH

establish (X(I), Y(I)), I = 1, NUVERT

CALL PREPLY(X, Y, NUVERT, YINTVL, INTVLS,
SWATH, RSLOPE)

establish (XP, YP)

IF(INOUT(XP, YP)) N = N + 1
```

The current dimensioning allows for a maximum of 100 polygon vertices and 25 segments per swath. It should be clear how larger polygons can be accomodated by redimensioning the arrays. Also, the constant -10^{75} occurring in subroutine SORT is the largest negative number representable in IBM–370 FORTRAN. Users of other equipment will need to make necessary modifications.

REFERENCES

Aldred, B. K., 1972, Points in polygon algorithms: UKSC–0025, UK Scientific Centre, IBM United Kingdom Ltd., Peterlee, County Durham, England 31p.

Anderson, K. P., 1976, Simple algorithm for positioning a point close to a boundary: Jour. Math. Geology, v. 8, no. 1, p. 105–106.

Burton, W., 1977, Representation of many-sided polygons and polygonal lines for rapid processing: Comm. ACM, v. 20, no. 3, p. 166–171.

Hall, J. K., 1975, PTLOC–a FORTRAN subroutine for determining the position of a point relative to a closed boundary: Jour. Math. Geology, v. 7, no. 1, p. 75–79.

Salomon, K. B., 1977, Algorithm for determining the orientation of a boundary: Computers & Geosciences, v. 3, no. 2, p. 383–384.

Spanier, E. H., 1966, Algebraic topology: McGraw-Hill Book Co., New York, 212 p.

APPENDIX

Program Listing

```
      SUBROUTINE PREPLY(X,Y,NUVERT,YINTVL,INTVLS,SWATH,RSLOPE)
C
C*********************************************************************
C
C   THIS ROUTINE PREPARES THE POLYGON CONSISTING OF THE NUVERT VERTICES
C   (X(I),Y(I)) BY FIRST SORTING THE SEGMENT Y-END POINTS INTO
C   DECREASING ORDER AND FORMING AN INTERVAL FOR EACH CONSECUTIVE PAIR:
C   (YINTVL(I),YINTVL(I+1)), I=1,INTVLS. THIS IS PERFORMED BY CALLING
C   SORT.
C        THE CODE CONSISTING OF THE DO 100 AND DO 200 LOOPS CONSTRUCTS,
C   FOR EACH INTERVAL I, THE LIST OF SEGMENTS TO BE TESTED BY INOUT.
C   THIS LIST IS PLACED INTO THE I-TH ROW OF SWATH. THE FIRST ENTRY,
C   SWATH(I,1), WILL BE SET TO THE NUMBER OF SEGMENTS IN THE ROW. NOTE
C   THAT AS YINTVL CONTAINS NO REDUNDANCIES, I.E. YINTVL(I) IS STRICTLY
C   GREATER THAN YINTVL(I+1), NO HORIZONTAL SEGMENTS WILL BE PLACED IN
C   THE LIST.
C        THE CODE CONSISTING OF THE DO 300 LOOP ESTABLISHES THE
C   RECIPROCAL SLOPE FOR EACH NON-HORIZONTAL SEGMENT. THIS IS TO BE
C   USED BY INOUT. FINALLY, THE SEGMENTS WITHIN A ROW OF SWATH ARE
C   ORDERED FROM LEFT-TO-RIGHT.
C
C*********************************************************************
C
      INTEGER SWATH(100,25)
      REAL X(100),Y(100),YINTVL(100),RSLOPE(100)
      CALL SORT(Y,NUVERT,YINTVL,INTVLS)
      IF(INTVLS .LE. 0) GOTO 400
      X(NUVERT+1)=X(1)
      Y(NUVERT+1)=Y(1)
      DO 100 I=1,INTVLS
  100 SWATH(I,1)=0
      DO 200 I=1,INTVLS
         DO 200 J=1,NUVERT
         IF(Y(J).GE.YINTVL(I) .AND. YINTVL(I+1).GE.Y(J+1) .OR.
     *      Y(J+1).GE.YINTVL(I) .AND. YINTVL(I+1).GE.Y(J))
     *                                  CALL INCLUD(SWATH,I,J)
  200    CONTINUE
      DO 300 I=1,NUVERT
         IF(Y(I).EQ.Y(I+1)) GOTO 300
         RSLOPE(I)=(X(I+1)-X(I))/(Y(I+1)-Y(I))
  300    CONTINUE
      CALL ORDER(X,Y,YINTVL,INTVLS,SWATH,RSLOPE)
      RETURN
  400 WRITE(6,401)
  401 FORMAT(' **** ERROR ****    PREP OF POLYGON ABORTED SINCE NO INTER
     *VALS WERE CONSTRUCTED')
      STOP
      END
      SUBROUTINE SORT(Y,NUVERT,YINTVL,INTVLS)

C
C*********************************************************************
C
C   ROUTINE ESTABLISHES THE INTERVALS OF THE Y-AXIS DEFINED BY THE
C   ENDPOINTS OF THE POLYGON'S SEGMENTS. THE DO 100 LOOP INITIALIZES
C   YSORT FROM THE SEGMENT Y-END POINTS. THE DO 200 LOOPS SORT YSORT
C   INTO DESCENDING ORDER. THE DO 300 LOOP ELIMINATES REDUNDANCIES IN
C   YSORT AND PLACES IRREDUNDANT SORTED Y'S INTO YINTVL. IT ALSO SETS
C   INTVLS TO THE TRUE NUMBER OF Y INTERVALS. JUST PRIOR TO RETURNING
C   A FINAL INTERVAL EXTENDING TO '-INFINITY' IS ESTABLISHED.
C
C*********************************************************************
C
      REAL Y(100),YINTVL(100),YSORT(100)
      INTEGER UPPER
      DO 100 I=1,NUVERT
  100 YSORT(I)=Y(I)
      UPPER=NUVERT-1
      DO 200 I=1,UPPER
         IPLS1=I+1
         DO 200 J=IPLS1,NUVERT
         IF(YSORT(I).GE.YSORT(J)) GOTO 200
            TEMP=YSORT(I)
            YSORT(I)=YSORT(J)
            YSORT(J)=TEMP
```

```
  200     CONTINUE
          YINTVL(1)=YSORT(1)
          INTVLS=0
          UPPER=NUVERT-1
          DO 300 I=1,UPPER
             IF(YSORT(I).EQ.YSORT(I+1)) GOTO 300
             INTVLS=INTVLS+1
             YINTVL(INTVLS+1)=YSORT(I+1)
  300     CONTINUE
          YINTVL(INTVLS+2)=-1.0E75
          RETURN
          END
          SUBROUTINE INCLUD(SWATH,I,J)
C
C*************************************************************************
C
C    ROUTINE PLACES THE J-TH POLYGON SEGMENT INTO THE NEXT AVAILABLE
C    LOCATION IN ROW I OF SWATH.
C
C*************************************************************************
C
          INTEGER SWATH(100,25),POINTR
          SWATH(I,1)=SWATH(I,1)+1
          POINTR=SWATH(I,1)
          SWATH(I,POINTR+1)=J
          RETURN
          END
          SUBROUTINE ORDER(X,Y,YINTVL,INTVLS,SWATH,RSLOPE)
C
C*************************************************************************
C
C    FOR EACH INTERVAL, A HORIZONTAL LINE IS PASSED THROUGH THE MIDDLE
C    (YMID) OF THE INTERVAL.  THE DO 100 LOOP PLACES THE X-INTERSECTION
C    OF EACH SEGMENT IN THIS SWATH SO THAT THESE INTERSECTIONS OCCUR
C    FROM LEFT-TO-RIGHT.
C
C*************************************************************************
C
          REAL X(100),Y(100),YINTVL(100),RSLOPE(100),XINTSC(25)
          INTEGER SWATH(100,25),POINTR,SEGNO,UPPER
          LOGICAL VERTSG
          DO 200 INTVAL=1,INTVLS
             NMBSEG=SWATH(INTVAL,1)
             YMID=(YINTVL(INTVAL)+YINTVL(INTVAL+1))/2.0
             DO 100 POINTR=1,NMBSEG
                SEGNO=SWATH(INTVAL,POINTR+1)
                VERTSG=ABS(X(SEGNO+1)-X(SEGNO)) .LT. 1.0E-5
                IF(VERTSG) XINTSC(POINTR)=X(SEGNO)
                IF(.NOT.VERTSG) XINTSC(POINTR)=X(SEGNO)+
     *                                         RSLOPE(SEGNO)*(YMID-Y(SEGNO))
  100        CONTINUE
             IF(NMBSEG.LT.2 .OR. MOD(NMBSEG,2).NE.0) GOTO 300
             UPPER=NMBSEG-1
             DO 200 I=1,UPPER
                IPLS1=I+1
                DO 200 J=IPLS1,NMBSEG
                   IF(XINTSC(I).LE.XINTSC(J)) GOTO 200
                   TEMP=XINTSC(I)
                   XINTSC(I)=XINTSC(J)
                   XINTSC(J)=TEMP
                   ITEMP=SWATH(INTVAL,I+1)
                   SWATH(INTVAL,I+1)=SWATH(INTVAL,J+1)
                   SWATH(INTVAL,J+1)=ITEMP
  200     CONTINUE
          RETURN
  300     WRITE(6,301) INTVAL
  301     FORMAT(' **** ERROR ****   PREP OF POLYGON ABORTED. INTERVAL ',I5,
     *    ' HAS EITHER LESS THAN TWO SEGMENTS OR AN ODD NUMBER OF THEM')
          STOP
          END
          LOGICAL FUNCTION INOUT(XP,YP)
C
C*************************************************************************
C
C    THE FOUR LINES ENCLOSED IN DASHES DETERMINE THE INTERVAL CONTAINING
C    YP. THE DO 400 LOOP CONTINUES UNTIL THE FIRST SEGMENT WITHIN THE
C    INTERVAL FALLS TO THE LEFT OF (XP,YP). IN THIS EVENT, INOUT IS SET
C    .TRUE. IFF AN EVEN NUMBER OF SEGMENTS HAD BEEN TESTED.
C
C*************************************************************************
```

```
C
      COMMON X(100),Y(100),YINTVL(100),INTVLS,SWATH(100,25),RSLOPE(100)
      INTEGER SWATH,SEGNO
      INOUT=.FALSE.
C-----------------------------------------------
      INTVAL=0
  100 INTVAL=INTVAL+1
      IF(YINTVL(INTVAL).GT.YP) GOTO 100
      INTVAL=INTVAL-1
C-----------------------------------------------
  300 IF(INTVAL.LT.1 .OR. INTVAL.GT.INTVLS) RETURN
      NMBSEG=SWATH(INTVAL,1) + 1
      DO 400 I=2,NMBSEG
         SEGNO=SWATH(INTVAL,I)
         IF(XP-X(SEGNO).LE.(YP-Y(SEGNO))*RSLOPE(SEGNO)) GOTO 500
  400    CONTINUE
      RETURN
  500 INOUT= MOD(I,2) .EQ. 1
      RETURN
      END
```