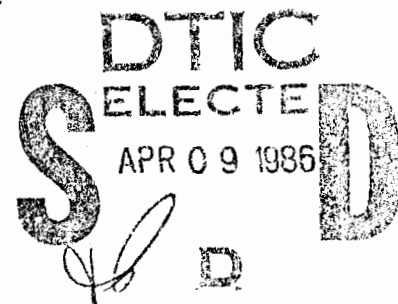


AD-A166 509

U.S. Army Intelligence Center and School
Software Analysis and Management System

A Collection of Area of Interest
(AOI) Algorithms



July 1985

National Aeronautics and
Space Administration



Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

AD-A166 509

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER JPL D-171	2. GOVT ACCESSION NO. ADA 166 509	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A COLLECTION OF AREA OF INTEREST (AOI) ALGORITHMS		5. TYPE OF REPORT & PERIOD COVERED Final
		6. PERFORMING ORG. REPORT NUMBER D-171
7. AUTHOR(s) A. Griesel, J. Gillis, E. Drell, R. J. Gardner, F. Lesh, N. Covella, B. Pardo		8. CONTRACT OR GRANT NUMBER(s) NAS7-918
9. PERFORMING ORGANIZATION NAME AND ADDRESS Jet Propulsion Laboratory 4800 Oak Grove Drive ATTN: 126-200 Pasadena CA 91109		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS RE 182 AMEND # 187
11. CONTROLLING OFFICE NAME AND ADDRESS Commander, USAICS ATTN: ATSI-CD-SF Ft. Huachuca, AZ 85613-7000		12. REPORT DATE July, 1985
		13. NUMBER OF PAGES 98
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Dissemination		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Prepared by JPL for the U. S. Army Intelligence Center and School's Combat Developer's Support Facility		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) AREA OF INTEREST (AOI), CONNECTEDNESS, WINDING NUMBER, CAUCHY'S THEOREM, POLYGONS, CONVEX HULL, ALGORITHMS, FORTRAN, PASCAL		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is one of a series of algorithm analysis reports performed for the US Army Intelligence Center and School covering selected algorithms in existing or planned Intelligence and Electronic Warfare (IEW) systems. This report describes the mathematical development and computer implementation of several algorithms designed to determine whether a given point in an arbitrary co-ordinate system is inside, on the boundary of, or outside a po-		

lygon specified as a series of points in the same coordinate system. The code and test results are given for each algorithm.

UAA011

①

U.S. Army Intelligence Center and School
Software Analysis and Management System

A Collection of Area of Interest
(AOI) Algorithms

Ann Griesel
James Gillis
Editors

July 1985

National Aeronautics and
Space Administration

JPL

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL D-171

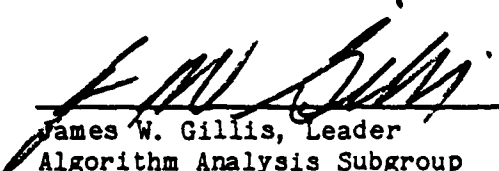
86 4 8 065

U.S. ARMY INTELLIGENCE CENTER AND SCHOOL
Software Analysis and Management System

A Collection of Area of Interest (AOI) Algorithms

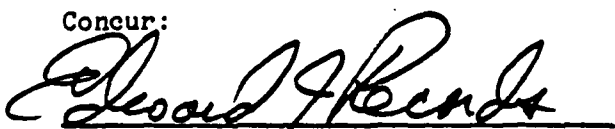
July 1985

Editors:

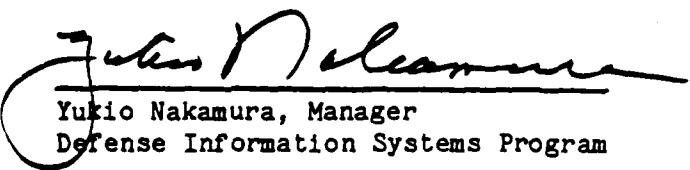

James W. Gillis, Leader
Algorithm Analysis Subgroup


Martha A. Griesel, Consultant
Algorithm Analysis Subgroup

Concur:


E. J. Records, Manager
USAMS Development Task


J. P. McClure, Manager
Ground Data Systems Section


Yukio Nakamura, Manager
Defense Information Systems Program

JET PROPULSION LABORATORY
California Institute of Technology
Pasadena, California



Accession For	
NTIS	CRA&I ☒
DTIC	TAB ☐
Unannounced ☐	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

CONTENTS

1.	INTRODUCTION	1-1
1.1	UNDERLYING MATHEMATICAL CONCEPTS	1-2
1.2	ORDER DEPENDENCE IN SPECIFYING VERTICES	1-5
2.	ALGORITHMS	2-1
2.1	A CONVEXITY ALGORITHM (QUADTEST)	2-2
2.2	AN ANGLE-SUMMING ALGORITHM (R_J_G)	2-5
2.3	THE QUADRANGLE METHOD (TEST_IT)	2-8
2.4	HANDEDNESS ALGORITHM	2-11
3.	TEST AND EVALUATION	3-1

APPENDIXES

A.	USAMS ALGORITHM ANALYSIS SERIES	A-1
B.	REFERENCES	B-1
C.	IMPLEMENTATION AND TESTING	C-1
D.	SOME MATHEMATICAL PROOFS	D-1

Figures

1-1.	Specifying a Polygonal AOI	1-7
1-2.	Star-like Polygons	1-8
1-3.	A General Non-degenerate Closed Curve	1-8
1-4.	Degenerate Quadrilaterals	1-9
1-5.	An $n(O,P)=1$ Example	1-9
1-6.	Vertex Ordering Examples	1-10
2-1.	Triangulation of Quadrilaterals	2-17
2-2.	Quadrangle Summation	2-18
2-3.	Triangulation of a Concave Quadrilateral	2-19

CONTENTS (Continued)

Figures (continued)

2-4. All Possible Quadrilaterals	2-20
D-1. Proof of Handedness Theorem	D-4

Tables

3-1. Geometric Robustness	3-2
3-2. Efficiency Considerations	3-3

ABBREVIATIONS AND ACRONYMS

AOI	Area of Interest
I/EW	Intelligence and Electronic Warfare
CDSF	Combat Developers Support Facility
USAICS	United States Army Intelligence Center and School
USAMS	USAICS Software Analysis and Management Systems Task

SECTION 1

INTRODUCTION

This report presents several independently developed and tested algorithms implemented in Pascal and FORTRAN 77 for determining if a point falls within a specified quadrilateral Area of Interest (AOI). These algorithms represent several ways of mathematically solving the AOI point inclusion problem, and provide a set of unclassified benchmarks against which existing system algorithms can be evaluated. These algorithm analyses and evaluations are being conducted for the Combat Developers Support Facility (CDSF) at the United States Army Intelligence Center and School (USAICS) by the Algorithm Analysis Subgroup of the USAICS Software Analysis and Management Systems (USAMS) Development Task. It is the ninth in a series of reports examining algorithms used in Intelligence and Electronic Warfare (I/EW) systems. Each of the other reports, listed in Appendix A, analyzes algorithms from several existing I/EW systems that perform a single function and examine their underlying mathematics. The main functional areas studied have been correlation, geographic transformations, and direction finding and location (fix) estimation.

In this introductory section, the basic mathematical and geometric considerations on which the AOI algorithms presented later are based will be introduced, and the potential order dependence inherent in defining a polygon by the location of its vertices discussed. The second section presents the algorithms, and the third section, an overview of testing results and algorithm evaluations.

Detailed test procedures and results are given in Appendix C. The algorithms presented here were developed by several different individuals. The subsection describing each algorithm is authored by the principal developer of that algorithm, with acknowledgments to associates as appropriate.

1.1 UNDERLYING MATHEMATICAL CONCEPTS

There are two basic mathematical approaches to determining if a point is included within the area bounded by a closed polygon. The topological approach relies on some notions of connectedness and the Wandering Cow Theorem. The analytic approach uses the concept of winding number and is based on Cauchy's Theorem. These approaches will be discussed in this section, and algorithms presented in Section 2 based on both methods.

Consider the polygon P formed by "connecting the dots" in Figure 1-1(a) in the order specified, giving Figure 1-1(b). P is a closed curve (since the last and first points are the same) and the region R it encloses is convex, that is, any two points (e.g. points a and b) within it can be joined by a straight line which also lies within the curve. Also, the line connecting any point inside the closed curve P with a point outside it (e.g. points a and c) must cross P exactly once. This is the basic idea behind the topological approach to determining if a point lies inside or outside the closed polygon.

There are formally two ways of viewing this intuitive result. The polygon P is the boundary of the region R , and the set $P+R$ is closed (" $+$ "

being used to represent the union operator). Then any connected set (i.e. any line segment is a connected set) that does not cross the boundary P of R lies either in R or outside $P+R$ (Moore, Theorem 33). Or, viewed another way, given the point a in R and c outside $P+R$, then P cuts between a and c (Hocking and Young, Theorem 3-6). This means P intersects every path (actually, every closed, connected subset) containing both a and c . Thus the cow can not wander from the pasture onto the road without crossing the fence.

Neither theorem cited above requires R to be convex. These theorems do place restrictions on the topological space containing P and R , but in this case P and R are so embedded in the Euclidean plane that the restrictions are easily satisfied. Thus, at first glance, these results would seem to apply to any polygon. However, there is a vacuous case, the straight line (Figure 1-1(c)) where they do not help, because R is empty so contains no point a and certainly can not contain a connected set. But in all other cases the theorems apply, although for polygons such as that illustrated in Figure 1-1(d), implementing this approach becomes tedious and tricky.

The analytic approach is based on Cauchy's Theorem which says that the value of the integral of an analytic function $f(z)$ (that is, a differentiable complex function) around a regular closed curve is zero. There are some restrictions on the polygon P hidden in the adjectives "analytic" and "regular." Certainly it is closed by definition, since the last point is connected back to the first point. The other conditions are

certainly satisfied by any convex polygon with three or more vertices, such as that developed in Figure 1-1(a and b). They are also satisfied if the polygon is merely star-like (Figure 1-2). In fact, any finite number of smooth curves joined at their endpoints and enclosing an open region will work (Figure 1-3). However, polygons such as those illustrated in Figure 1-4 may cause trouble due to segments which enclose no area. Carefully handled, (a) can be viewed as two disjoint closed curves, for which the theorem holds, but (b) can not.

Now using Cauchy's Theorem it can be shown that, for a point a not on the closed curve P , the value of the integral around P of $dz/(z-a)$ is a multiple of $(360^\circ)i$. This can be seen (but not proved) by observing that

$$\begin{aligned} dz/(z-a) &= d \log(z-a) \\ &= d \log|z-a| + i[d \arg(z-a)]; \end{aligned}$$

the integral around a closed curve of the first term is zero and $\arg(z-a)$ increases or decreases by a multiple of 360° .

Define the winding number $n(a,P)$ as this integral divided by $(360^\circ)i$. Then n is positive if P is traversed counterclockwise, and

$$n(a,-P) = -n(a,P).$$

Since $R+P$ is a bounded, connected, closed set,

$$n(a,P) = 0$$

for all a not in $R+P$. Furthermore,

$$n(a,P) = n(b,P)$$

for all a and b in R . The only place the winding number is not defined is on P itself. This once again excludes Figures 1-1(c) and 1-4(b) from consideration. However, it also means an algorithm based on this principle must check if the point lies on P for any polygon.

One final result is useful when designing AOI point inclusion algorithms. It pertains to cases where $n = 1$.

Let a and b be two points on the polygon, and let P_{ab} be the arc from a to b , P_{ba} the arc from b to a (Figure 1-5). Suppose that a lies in the lower half plane and b in the upper half plane. If P_{ab} does not meet the negative real axis, and P_{ba} does not meet the positive real axis, then $n(0, P) = 1$.

Since the winding number is translation and rotation invariant, the conditions on a and b are satisfied without loss of generality. This gives a simple criteria which covers the n equals 0 and 1 cases.

1.2 ORDER DEPENDENCE IN SPECIFYING VERTICES

Throughout the above, the vertices were assumed specified in the order in which they were to be connected. Can this restriction be relaxed? Unfortunately not.

Given four points in the plane, if one of them is in the interior of the convex hull of (that is, the smallest convex set containing the other

three (Figure 1-6 (a)), the four points do not define a unique quadrilateral. In general, they will define three different ones (Figure 1-6(b), (c), and (d)) having only the line segments AD, BD, and CD common to all three. Thus any point in the convex hull that does not fall on one of these three line segments (the shaded area in 1-6(e)) is or is not in "the" quadrilateral depending on which quadrilateral is chosen.

Therefore, to be able to determine if a fifth point falls within a quadrilateral defined by four given corner points, either the order (clockwise or counterclockwise) of the points must be specified, or the resulting quadrilateral must be convex and the boundary of the convex hull taken to be the quadrilateral. Any polygon with four or more vertices can be made concave by reordering the defining points (see Figure 1-6(f)); remember that these self-intersecting polygons are mathematically the hardest to deal with. To avoid constraining the problem by considering only convex quadrilaterals, and it is not always obvious that a figure is not convex, nor that easy to check for convexity, the order of the vertices must be specified.

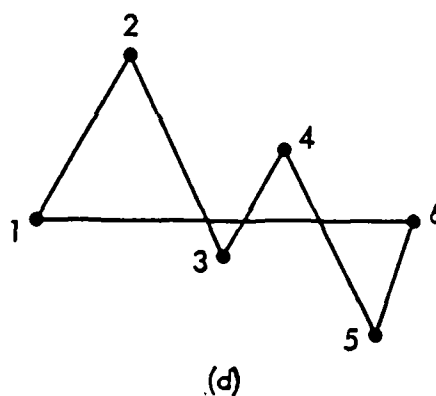
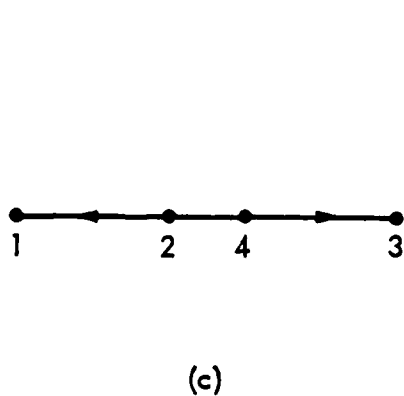
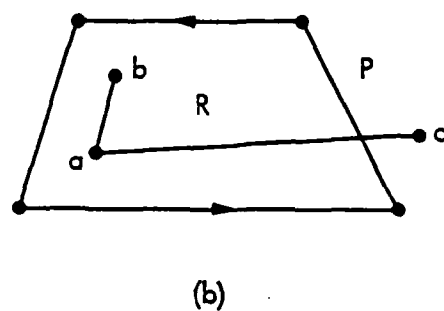
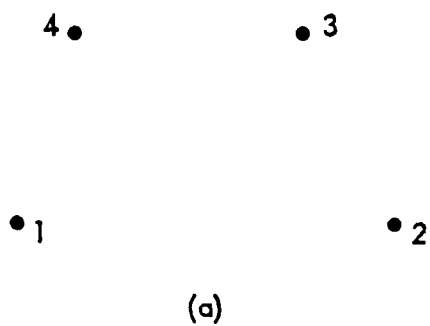
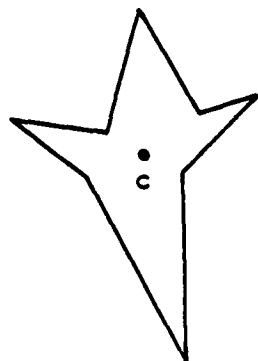
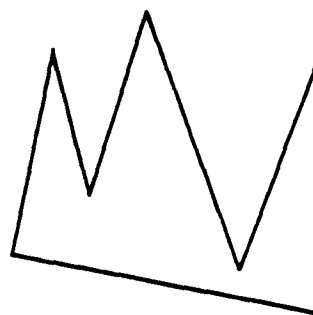


Figure 1-1. Specifying a Polygonal AOI



(a) Star-Like with
Star Center C



(b) Not Star-Like

Figure 1-2. Star-like Polygons

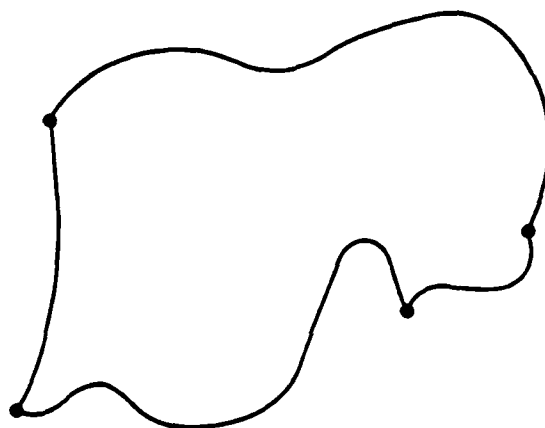


Figure 1-3. A General Non-degenerate Closed Curve

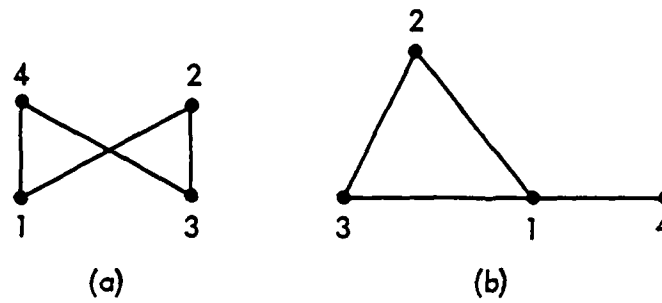


Figure 1-4. Degenerate Quadrilaterals

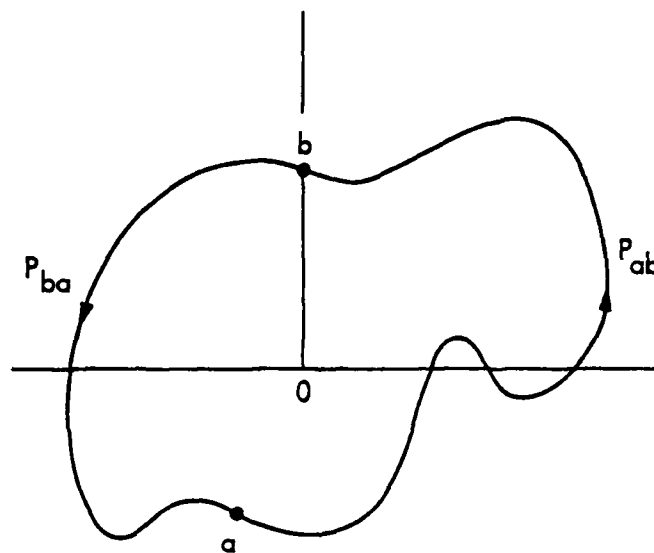


Figure 1-5. An $n(O, P) = 1$ Example

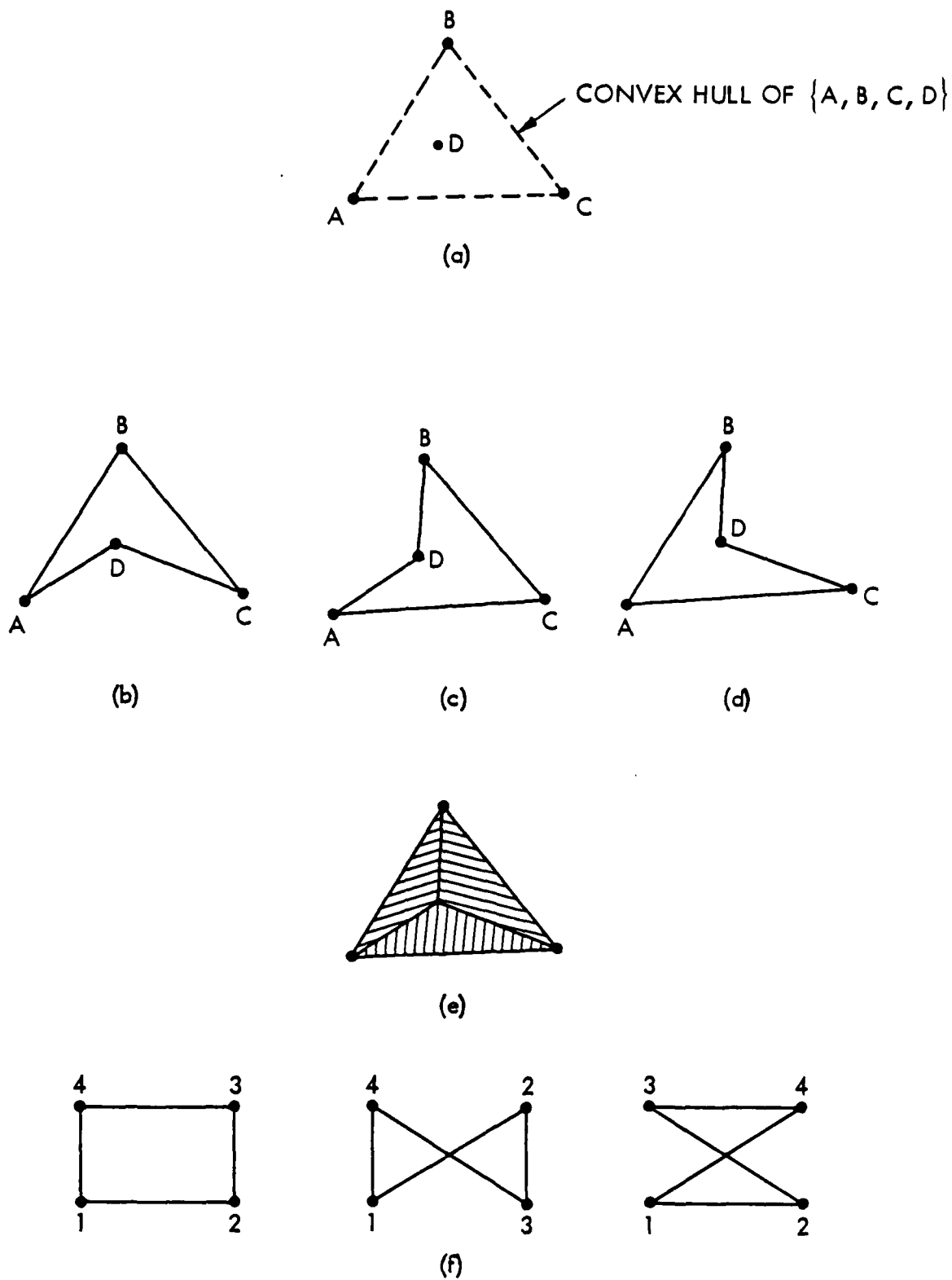


Figure 1-6. Vertex Ordering Examples

SECTION 2

ALGORITHMS

Before presenting the algorithms themselves, a formal statement of the problem is needed.

Statement of Problem:

Given an ordered set of n points $P_1(x_1, y_1)$, $P_2(x_2, y_2), \dots, P_n(x_n, y_n)$ where n may be any finite positive integer, determine if a specified point $P(x, y)$ is enclosed by the polygon formed by connecting the points in the given order by straight line segments.

This polygon may be concave or convex and may loop in any fashion. The point P may be multiply enclosed by the polygon, in which case some of the algorithms will determine the number of enclosures.

2.1 A CONVEXITY ALGORITHM (QUADTEST)

Author: E. Drell

Acknowledgments: R. Henderson, CSUN
E. Schmidt, JPL

Principle of the Algorithm:

This algorithm was designed specifically for quadrilaterals. It is based on three observations.

- (1) Any path connecting a point inside and a point outside a closed curve intersects the curve (the Wandering Cow Theorem discussed above).
- (2) Any quadrilateral has at most one point of concavity.
- (3) Any quadrilateral (Figure 2-1(a)) can be divided into two triangles (Figure 2-1(b)).

The last point allows the first to be applied. Although in general it would be hard to establish that any path did or did not cross the boundary, in the case of a triangle this reduces to showing that the lines passing through AP, BP, and CP intersect BC, CA, and AB, respectively (see proof in

Note: This algorithm produces valid results for test cases C-2, C-5, and C-7, and is not applicable to the other test cases.

Appendix D). Thus the following algorithm will divide the quadrilateral into two triangles, then use this test to determine if the point lies within either triangle.

The Algorithm: (implemented in QUADTEST in Appendix C)

1. Test for vertices and output message and stop if point is on a vertex.
2. If not, determine the concavity of the figure.

To do this determine if either the second or fourth points of the figure lie within the triangle defined by the other 3 points. Recall this can only happen for one point. If this condition is true, then assign the second vertex as the point of inflection or concavity. If the condition is false then assign the first vertex as the point of inflection. The concavity only determines the way the quadrilateral is divided.

3. Break the quadrilateral into two triangles based on the inflection point (Figure 2-1 (b)).

The default is taking the first and third vertices as the center line. If so indicated by the inflection point, the second and fourth are used instead.

4. Test the two triangles to see if the point is interior to either.
 - A. For each side construct a line from the vertex opposite the side to the point in question.
 - B. If this line intersects the triangle side within the bounds of the two endpoints of the side, then the test is successful. For any point to be interior to the triangle, each vertex/point line must intersect its respective edge. If less than three intersections are determined then the point is outside the triangle. To test if a vertex/point line and triangle side intersect, subtract the equation of the line defining the triangle side from the equation of the line constructed in (A). Evaluate the resulting equation at the two endpoints of the side yielding two values. For a solution equal to zero to exist within the interval between the two points, the value on one side of the interval must be greater than zero while on the other side less than zero (by the Intermediate Value Theorem). Thus by multiplying the two values together, if a number greater than zero results, the solution does not fall within the interval forcing a failure of the triangle test.

2.2 AN ANGLE-SUMMING ALGORITHM (R_J_G)

Author: R.J. Gardner

Acknowledgment: F. Lesh

Principle of the Algorithm

Use the point in question, $P(x,y)$, as the vertex of all angles. Calculate the angles P_1PP_2 , $P_2PP_3, \dots, P_{n-1}PP_n$, and P_nPP_1 where $P_n = P_1$. Call these angles A_i where $i=1,2,3,\dots,n$ (i.e., A_1 is P_1PP_2). The order and sign of the angles must be carefully preserved, and the value of each angle must be properly determined in the interval $-180^\circ \leq A_i \leq 180^\circ$. If the algebraic sum of these n angles (the winding number) is zero, the point in question is not enclosed by this polygon as defined by the ordering of the points. If the absolute value of this sum is $360^\circ n$, then the point has been enclosed n times by the polygon.

The Algorithm: (implemented in R_J_G in Appendix C)

Step 1. Read in the polygon points $P_1(x_1,y_1)$, $P_2(x_2,y_2), \dots, P_n(x_n,y_n)$, n and R (the resolution to consider two points coincident).

Step 2. Read in the test point.

Step 3. Initialize sum S to zero.

$S = 0$

Step 4. For i equals 1 through n do the following; Translate the origin of the coordinates to the test point, $P(x,y)$.

$$X_i = x_i - x,$$

$$Y_i = y_i - y.$$

NOTE: $P(X,Y) = (0,0)$.

Calculate;

$$M_i = \sqrt{X_i^2 + Y_i^2}.$$

If $M_i \leq R$ output message "Point of interest is on the vertex."

Go to Step 8. This branch is used to prevent division overflow.

R was chosen as 0.0001 for the purpose of this test (1 km in 10,000 km).

Step 5. Do this step for each of the i equal to 1 through n with $n + 1 = 1$.

$$A_i = [\text{sign}(X_i Y_{i+1} - X_{i+1} Y_i)] \arccos[(X_i X_{i+1} + Y_i Y_{i+1}) / (M_i M_{i+1})]$$

As A_i approaches $\pm 180^\circ$, the testpoint approaches the side. For the purpose of this test, $\pm 179.88^\circ$ was used to ensure the capture of boundary cases in spite of round off errors. (i.e. If $|A_i| > 179.88$ then output the message, "The point is ON THE LINE.")

$$S = S + A_i$$

Step 6.

$$E = \text{Round}[\text{abs}(S/360^\circ)] \text{ to the nearest integer.}$$

Step 7.

If $E = 0$, point is outside polygon.

Output message, "Point not enclosed."

Otherwise output, "Point is enclosed."

Note: The value of E yields the number of times the point is enclosed.

Step 8. More points to check? If yes go to Step 2.

Step 9. Another polygon? If yes go to Step 1.

Step 10. EXIT

2.3 THE QUADRANGLE METHOD (TEST_IT)

Author: F. Lesh

Principle of the Algorithm:

Trigonometry may be thought of as the study of the mapping of spatial points in two dimensions onto a unit circle at the origin of the coordinate system. The angles are measured by their arc length on the unit circle (see Figure 2-2 (a)), and are positive in the counter-clockwise sense. This quadrilateral algorithm is similar to Section 2.2, R_J_G, which is based on the unit circle. However, rather than angles, the points are mapped onto a unit square defined by (1,1), (-1,1), (-1,-1), (1,-1) in that order. Analogous to arc lengths as angles, call these displacements along the unit square "quadrangles." The measure of these "quadrangles" is the distance along the perimeter between the ordered mapped points starting at point (1,1). In Figure 2-2 (b) the values of the quadrangles at critical points are given as the eight values for S. These critical points correspond to multiples of $\pi/4$ on the unit circle. The origin of the coordinate system is translated to the point in question and the zero quadrangle is defined by the point (1,1) Figure 2-3 (b). Starting with the first point of the AOI, calculate the quadrangle between consecutive points through the last corner on to the first corner. The algebraic sum (Figure 2-2 (c)) of these quadrangles divided by 8 gives the number of times the point has been enclosed by the polygon. In an analogous manner, the sums of the angles divided by 2π gives the number of enclosures of the test

polygon. This algorithm has the advantage over angle summing with trigonometry in that it uses no transcendental functions, uses less memory to execute, and executes more rapidly.

The Algorithm: (implemented in TEST_IT in Appendix C)

Given n points on a plane, $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$, and a test point (h, k) ; determine if the test point lies inside, outside, or on the boundary of the area bounded by the lines connecting (x_0, y_0) to (x_1, y_1) to $(x_2, y_2), \dots, (x_{n-1}, y_{n-1})$ to (x_0, y_0) . Note that $x_n = x_0$, and $y_n = y_0$.

Two functions must be defined.

$S(u, v) \equiv$ the distance around the unit square (Figure 2-2 (b) from (1,1) counter clockwise.

If $|v| \geq |u|$ AND $v > 0$, THEN $S(u, v) = 1 - (u/v)$.

If $|u| \geq |v|$ AND $u < 0$, THEN $S(u, v) = 3 + (v/u)$.

If $|v| \geq |u|$ AND $v < 0$, THEN $S(u, v) = 5 - (u/v)$.

If $|u| \geq |v|$ AND $u > 0$, THEN $S(u, v) = 7 + (v/u)$.

$R(w) \equiv$ a function used to calculate the difference between two points as seen from the test point. A small variable ϵ , is introduced to ensure the boundary conditions against round off errors.*

*Note: An arbitrary value for ϵ of 0.0001 was chosen for the "Test and Input" in Appendix C.

If $-4+\epsilon < w < 4-\epsilon$ $R(w) = w$.

If $w < -4-\epsilon$ $R(w) = w+8$.

If $w > 4+\epsilon$ $R(w) = w-8$.

Otherwise test point is on a boundary, and go to next case.

Test points on boundaries (corners and sides) are considered to be inside the polygon.

Proceed as follows: first calculate

$$u_0 = x_0 - h, \quad v_0 = y_0 - k.$$

$$S_0 = S(u_0, v_0).$$

$$Z = 0.$$

Then, for each i from 1 to n perform the following sequence of steps:

(1) $u_i = x_i - h, \quad v_i = y_i - k.$

(2) If $((u_i = 0) \text{ AND } L(v_i = 0))$ set $z = 8$ and go to step six.

(3) $S_i = S(u_i, v_i).$

(4) $D_i = R(S_i - S_0).$

(5) $S_0 = S_i.$

(6) $Z = Z + D_i.$

Finally $Z/8 =$ the number of times the test point has been enclosed.

2.4 HANDEDNESS ALGORITHM

Author: R. Gardner

Acknowledgment: F. Lesh

Principle of Algorithm:

To determine if a point is interior to a convex polygon, progress along the sides and ask for each side (pair of points); "is the point in question on the left or right of the extended line segment of these two points?" If the point is always on the same side of each segment for a complete circuit around the convex polygon, the point is interior to the polygon.

To determine if a point $P(x,y)$ is to the left (or right) of a directed segment $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$ evaluate the determinant;

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{vmatrix} \equiv h \text{ (handedness)}$$

If the determinant is 0 the points are colinear. If h is >0 the P is a left turn. If $h < 0$ the turn is right. See Appendix D for proofs.

This does not work for a concave polygon. However in this case one can consider the quadrilateral as consisting of two overlapping triangles as illustrated, (Figure 2-3). The first triangle (b) is formed by excluding the concave vertex. If the h test shows the point is definitely

outside this triangle (h of different signs and none zero) then the point is outside the AOI. However, in any other case the second triangle still must be checked. The second triangle c is formed by the concave point and the two points adjacent to it in the ordered sequence. If the point is definitely inside this triangle ($h_i=0$ and of the same sign), the point is definitely outside the AOI. For the boundary cases, if both tests had one $h = 0$, the point is outside the area of interest. If one h_i of one of the two triangles was zero, then the unknown point is on the boundary of the AOI.

It should be noted at this point that the handedness check is used to determine the nature of a quadrilateral (see Figure 2-4 for the quadrilateral case). The given h_i are $h_i((x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1}))$. For a convex quadrilateral the h_i will all have the same sign. For a concave quadrilateral one sign will be different. For the degenerate case of a simple triangle, one h is zero and the three remaining h are of the same sign. For the triangle with a ray there is one zero h and one which differs in sign with the remaining pair. The "bow tie" has two adjacent h_i greater than zero and two less than zero.

The Algorithm:

Write a routine;

$HAND(x_1, y_1, x_2, y_2, x_3, y_3, h),$

which determines h for the three points P_1, P_2, P_3 in that order.

Step 1. Input $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, $P_3(x_3, y_3)$, and $P_4(x_4, y_4)$.

Step 2.

Calculate:

$$\text{sign } h(P_4, P_1, P_2) = h_1$$

$$\text{sign } h(P_1, P_2, P_3) = h_2$$

$$\text{sign } h(P_2, P_3, P_4) = h_3$$

$$\text{sign } h(P_3, P_4, P_1) = h_4$$

Where;

$$\text{sign } h = +1 \text{ if } h > 0,$$

$$\text{sign } h = 0 \text{ if } h = 0,$$

$$\text{sign } h = -1 \text{ if } h < 0.$$

This set of $\{h_i\}$ is used to determine the type of quadrilateral.

Step 3. Input point $P(x, y)$.

Step 4. If all of the $\{h_i\}$ are of the same sign and non zero, it is a convex quadrilateral.

Calculate:

$$\text{sign } h(P_1, P_2, P) = H_1$$

$$\text{sign } h(P_2, P_3, P) = H_2$$

$$\text{sign } h(P_3, P_4, P) = H_3$$

$$\text{sign } h(P_4, P_1, P) = H_4$$

If the set $\{H_i\}$ is of the same sign but non zero, output the message, "Point is in the AOI!", and go to Step 8.

If the set $\{H_i\}$ is of the same sign but with one zero, output the message, "The point is on the boundary of the AOI!", and go to Step 8.

If the set $\{H_i\}$ contains two zeros and the other two H_i are of the same sign output the message "Point is on a vertex of AOI!", and go to Step 8.

Otherwise output the message "Point is outside the AOI!", and go to Step 8.

Step 5. If no h_i are zero and one of the h , say h_j , is of a different sign than the other two, it is a concave quadrilateral. From the triangle, formed by excluding this P_j , calculate the following:

$$\text{sign } h(P_{j+1}, P_{j+2}, P) = H'_1$$

$$\text{sign } h(P_{j+2}, P_{j+3}, P) = H'_2$$

$$\text{sign } h(P_{j+3}, P_{j+1}, P) = H'_3$$

If any two of these H'_i are different in sign (excluding a zero) output the message "Point is outside AOI!", and go to Step 8.

If two of the H'_i are zero output the message, "Point is on a vertex of AOI!", and go to Step 8.

Otherwise, for the triangle formed by the P_j (concaved point) and its adjacent points (P_{j-1} and P_{j+1}) calculate;

$$\text{sign } h(P_{j-1}, P_j, P) = H''_1$$

$$\text{sign } h(P_j, P_{j+1}, P) = H''_2$$

$$\text{sign } h(P_{j+1}, P_{j-1}, P) = H''_3$$

If the H''_i are all of the same sign and not zero, or if both $\{H'\}$ and $\{H''\}$ contain a zero, output the message, "Point is exterior to the AOI!", then go to Step 8.

If two H''_i are zero, output the message, "The point is on a vertex of the AOI!", and go to Step 8.

If either set $\{H'_i\}$ or $\{H''_i\}$ contains a zero, but a zero is not contained in both, and $\{H''\}$ contains no change in sign, output the message, "The point is on the boundary of the AOI!", then go to Step 8.

Otherwise, output the message, "The point is inside the AOI!", and go to Step 8.

Step 6.

If only one $h(h_j)$ is zero, (Figure 2-4(c) or (d)) form a triangle by excluding the point P_j ; i.e. $P_{j+1}, P_{j+2}, P_{j+3}$ where $j + n$ is cyclic in the range 1,2,3,4. Read in the $P(x,y)$.

Then calculate;

$$\text{sign } h(P_{j+1}, P_{j+2}, P) = H'_1,$$

$$\text{sign } h(P_{j+2}, P_{j+3}, P) = H'_2.$$

$$\text{sign } h(P_{j+3}, P_{j+1}, P) = H'_3.$$

Where the H'_i are non zero and of the same sign output the message "The point is in the AOI!".

Where there is one zero H'_i and the other two H_i are of the same sign output the message "Point is on the boundary of the AOI!".

Where there are two zero H'_i are non zero and of the same sign output the message "Point is on vertex!", and go to Step 8.

Where none of these conditions exist output the message, "Point is outside the AOI!".

Go to Step 8.

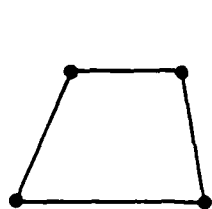
Step 7.

Otherwise output the message, "Illegal input!" and go to next step 9.

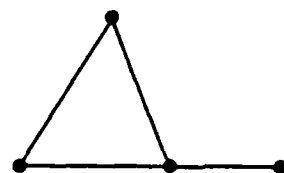
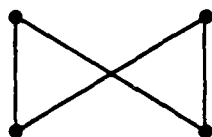
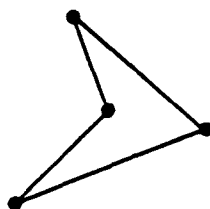
Step 8. Output the message "Are there more points to Input?". If answer is yes go to Step 4.

Step 9. Output the message, "Are there more areas of interest so specify?". If answer is yes go to Step 1.

Step 10. Exit.

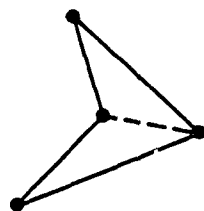
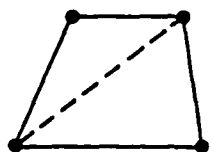


REGULAR



DEGENERATE

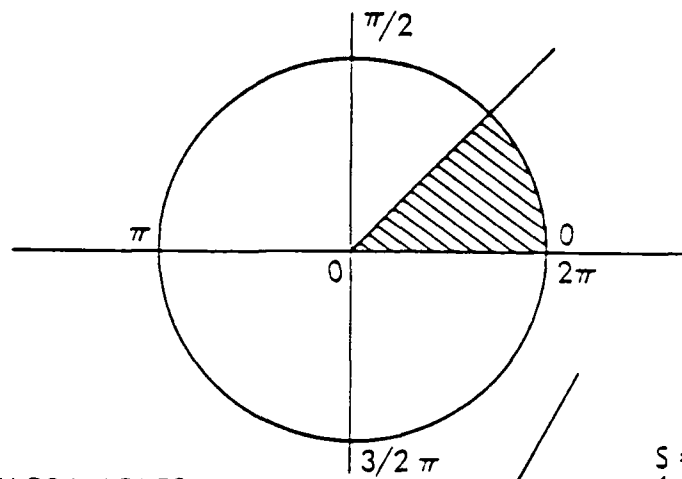
(a)



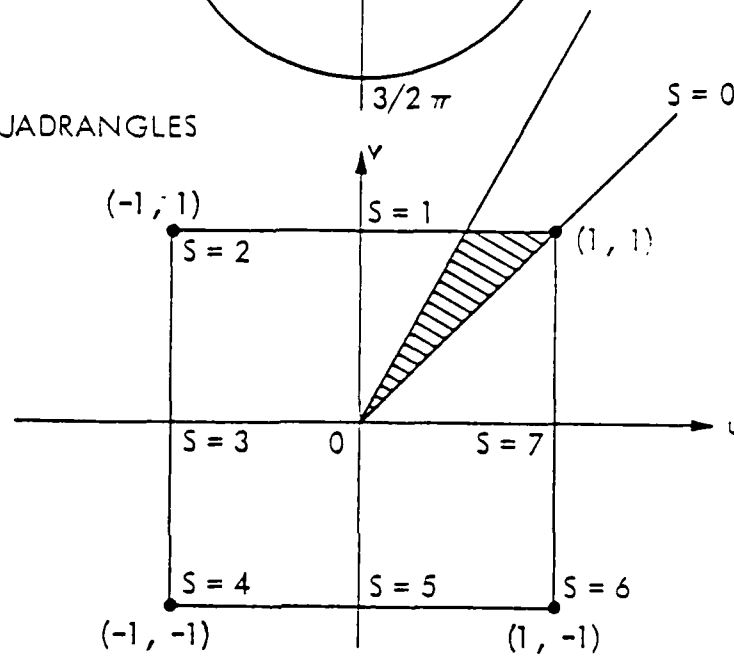
(b)

Figure 2-1. Triangulation of Quadrilaterals

(a) CIRCULAR ARCS



(b) QUADRANGLES



(c) SAMPLE ENCLOSURE

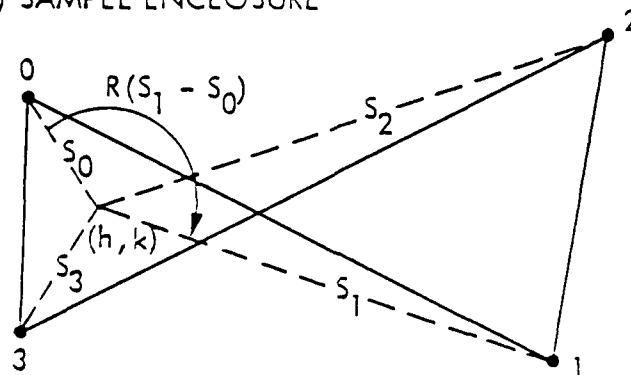
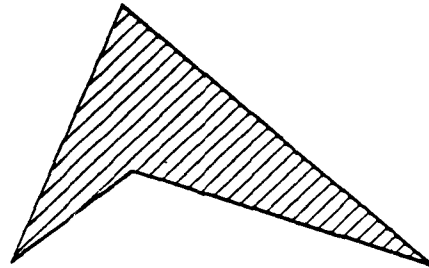
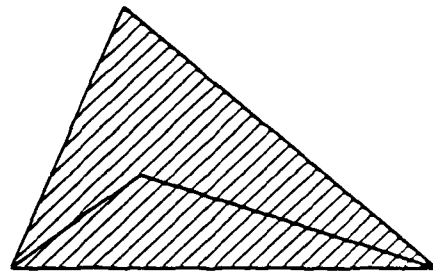


Figure 2-2. Quadrangle Summation

(a) CONCAVE QUADRILATERAL



(b) FIRST TRIANGLE



(c) SECOND TRIANGLE

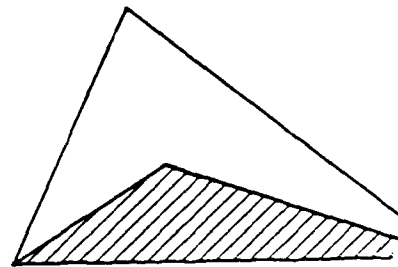
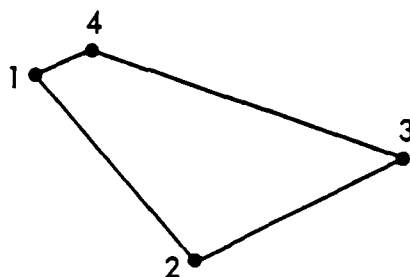


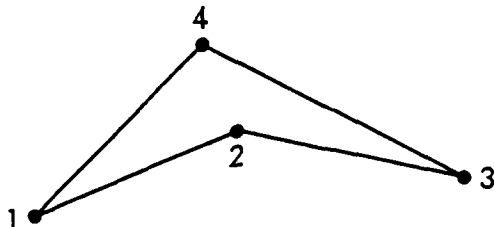
Figure 2-3. Triangulation of a Concave Quadrilateral

(a) A CONVEX QUADRILATERAL



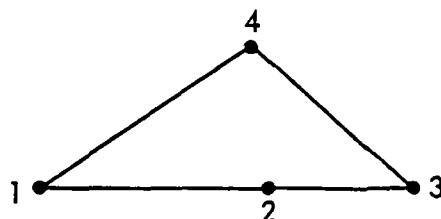
$$\begin{aligned} h_1 &= + \\ h_2 &= + \\ h_3 &= + \\ h_4 &= + \end{aligned}$$

(b) A CONCAVE QUADRILATERAL



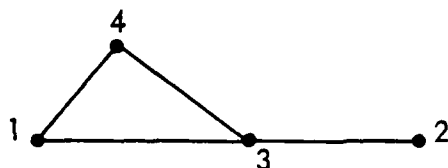
$$\begin{aligned} h_1 &= - \\ h_2 &= + \\ h_3 &= + \\ h_4 &= + \end{aligned}$$

(c) A SIMPLE TRIANGLE (DEGENERATE QUADRILATERAL)



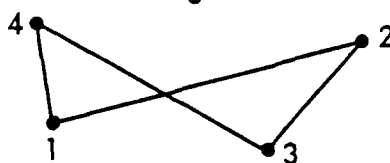
$$\begin{aligned} h_1 &= 0 \\ h_2 &= + \\ h_3 &= + \\ h_4 &= + \end{aligned}$$

(d) A TRIANGLE WITH A RAY (DEGENERATE QUADRILATERAL)



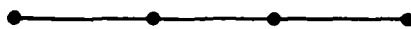
$$\begin{aligned} h_1 &= 0 & h_3 &= + \\ h_2 &= - & h_4 &= + \end{aligned}$$

(e) A "BOW TIE"



$$\begin{aligned} h_1 &= - & h_3 &= + \\ h_2 &= - & h_4 &= + \end{aligned}$$

(f) A LINE SEGMENT (DEGENERATE QUADRILATERAL)



$$h_1 = h_2 = h_3 = h_4 = 0$$

Figure 2-4. All Possible Quadrilaterals

SECTION 3

TEST AND EVALUATION

The algorithms presented in Sections 2.1, 2.2, 2.3, and 2.4 were implemented and tested on a VAX-11/780 using Pascal and FORTRAN. The code, test quadrilaterals and points, test results, and specific comments on the performance of each algorithm are given in Appendix C. The following is a survey of those findings.

Two main categories of performance emerged in designing and conducting the tests:

- (1) Geometric robustness.
- (2) Timing and efficiency.

The first is a descriptive as well as evaluative category. For example, in some applications points lying on the polygon itself may be considered outside the enclosed region and for other applications inside it. Table 3-1 presents geometric robustness for the four algorithms. Note that, although in theory some of the algorithms work for any polygons with three or more sides, only quadrilaterals were used in the testing. The three classes tested were convex, concave, and degenerate. Both design and analysis of the algorithms indicate that for polygons with more sides the algorithms presented in Sections 2-2 and 2-3 should work in all but (possibly) the degenerate cases.

Questions of timing and efficiency arose primarily in five ways. Three of these are fairly independent of the specific application:

- (1) How many special cases had to be checked.
- (2) How many subproblems were solved.
- (3) Whether trigonometric functions were used.

Table 3-2 addresses these. The other major efficiency issues are quite application dependent. The order in which the various tests are performed in an algorithm depend on the characteristics of the most likely quadrilaterals and input points. Thus the following two considerations can significantly affect efficiency:

- (4) Whether to test for wild inputs.
- (5) When to test for degenerate quadrilaterals.

Table 3-1. Geometric Robustness

Algorithm from Section:	2.1	2.2	2.3	2.4
Name	QUADTEST	R_J_G	TEST_IT	HANDEDNESS
Convex Interiors	+	+	+	+
Concave Interiors	+	+	+	+
Boundaries (Non Degenerate)	+	+	+	+
Degenerate Quadrilaterals	-	+	+	-
Quadrilaterals	+	+	+	+
N-sided Polygons*	-	+	+	-
+ handles				
- does not handle				
* has not been proved in this report				

These questions are also covered in Table 3-2. For any applications the trade-off must be made, whether there are enough wild inputs to take time testing for them, and just how often the quadrilaterals will be degenerate. The table is meant only as a guide for applications.

Table 3-2. Efficiency Considerations

Algorithm from Section:	2.1	2.2	2.3	2.4
Name	QUADTEST	R_J_G	TEST_IT	HANDEDNESS
Subproblems	+	-	-	-
Trigometric Functions	-	+	-	-
Wild Inputs	-	-	-	-
Degenerate Polygons	+	-	-	+
Language	P	F	F	P
- does not have				
0 comes late				
+ comes early				
F FORTRAN				
P PASCAL				

APPENDIX A

USAMS ALGORITHM ANALYSIS SERIES

Gillis, J.W., Griesel, M.A., and Radbill, J.R., Analysis of Geographic Transformation Algorithms. JPL Report D-181, July, 1982.

Gillis, J.W., Griesel, M.A., and Radbill, J.R., Correlation Algorithm Report. JPL Report D-182, September, 1982.

Gillis, J.W., Griesel, M.A., and Radbill, J.R., and Kuo, T.J., Intelligence Algorithm Methodology I. JPL Report D-183, August, 1983.

Gillis, J.W., Griesel, M.A., Radbill, J.R., and Sizemore, N.L., Cross-Correlation: Statistics, Templating, and Doctrine. JPL Report D-184, February, 1984.

Gillis, J.W., and Griesel, M.A., I/EW Direction-Finding and Fix Estimation Analysis Report. JPL Report D-180, to be published.

Mathematics Clinic, Claremont Graduate School, Applications of Correlation Techniques to Battlefield Identification I. JPL Report D-179, June 1984.

Institute of Decision Sciences, Claremont McKenna College, Power of Statistical Tests Used in Correlation Techniques for Battlefield Identification II. JPL Report to be published.

Mathematics Clinic, Claremont Graduate School, A Non-Standard Probabilistic Position-Fixing Model. JPL Report D-186, June 1985.

Carson, G.S., Gillis, J.W., and Griesel, M.A., Intelligence Algorithm Methodology II: A Tactical Sensors Model. JPL Report D-185, to be published.

APPENDIX B

REFERENCES

SECTION 1.1

Wandering Cow Theorem:

Hocking, J.G., and Young, G.S. Topology. Massachusetts: Addison-Wesley, 1961 [p.109].

Moore, R.L. Foundations of Point Set Theory. Rhode Island: American Mathematical Society, 1962 [p.12].

Winding Number and Cauchy's Theorem:

Ahlfors, L.V. Complex Analysis. New York: McGraw-Hill, 1966 [pp.109-117].

Goodstein, R.L. Complex Functions. New York: McGraw-Hill, 1965 [pp.91-94].

Veech, W.A. A Second Course in Complex Analysis. New York: W.A. Benjamin, 1967 [pp.23-27].

SECTION 2.1

Intermediate Value Theorem:

Buck, R.C. Advanced Calculus. New York: McGraw-Hill, 1965 [p.75].

APPENDIX C
IMPLEMENTATION AND TESTING

by

Nick Covella and Bruce Pardo

For each of the following programs and test results, refer to Figures C-1 through C-7. The solid lines reflect the input quadrilaterals. The X's indicate the locations of the input test points. The handling of points on boundaries (corners and sides) is described in Section 2.

There are four coded programs (2 in Pascal and 2 in FORTRAN) that have been designed and tested by the individuals who took part in the design of the respective programs. The four programs are:

- | | | | |
|----|------------|-----------|-----|
| 1. | QUADTEST | (PASCAL) | 2.1 |
| 2. | R_J_G | (FORTRAN) | 2.2 |
| 3. | TEST_IT | (FORTRAN) | 2.3 |
| 4. | HANDEDNESS | (PASCAL) | 2.4 |

Each of these algorithms used different mathematical methods to determine whether a point lies inside, outside, on a corner, or on a boundary of the input figure. These methods are discussed in Section 2. The analyses of the generated results appear below.

PROGRAM QUAD: (SECTION 2.1)

This program used the idea of triangles. If the line connecting the test point with each vertex intersects the line of the opposite side of the triangle at a point within the confines of the triangle, and this condition holds true for all three cases, then the test point is in the interior of the triangle in question.

For non-degenerate quadrilaterals, this algorithm performed well. It does not properly handle the bow tie or degenerate cases for which three points fall on the same line.

PROGRAM RJG: (SECTION 2.2)

This program used the idea of angle summing. The point in question is used as the reference point, then the angles formed by this point and the points of the figure are summed. Direction of counter-clockwise or clockwise makes no difference so long as the points tested are in sequence. The value of the final summation is either 0 or a multiple of 360. If the value is 0, then the point in question is outside the figure, otherwise it is inside the figure.

Based on the testing here, this algorithm seems very robust.

PROGRAM TEST_IT: (SECTION 2.3)

This algorithm used the idea of a unit square. Function Scale calculates the distance around the input figure starting at (0,0) and travelling counter clockwise. Rcalc calculates the angular difference between two points as seen from the test point. The result gives the number of times

the test point was circled. For a closed quadrilateral that had no internal intersections, if the test point was inside the figure then the enclosed value was 1; otherwise, it is zero.

Overall, this algorithm seems very robust. It tested each input point correctly and quickly and could be used in a real-time environment.

PROGRAM HANDEDNESS: (SECTION 2.4)

This algorithm's basic principle only does not apply to "bow-tie" quadrilaterals. One should progress along the perimeter in the direction specified by the ordered points. With each pair of the perimeter points, follow along the direction specified by their ordering, connect two consecutive points, form a segment, then check to determine if the test point is on the right or left of the segment. If the answer is sometimes "left" and with other point pairs "right", the point is exterior to the convex quadrilateral. If the answer is always the same (left or right), the point

is interior to the convex polygons. In the case of the concave quadrilateral, the convexity limitation is avoided by dividing the quadrilateral into two triangles and applying this principle to each of the triangles.

Though this algorithm appears very robust, the logic is very complicated when handling the concave case. Thus it becomes impractical to implement in the field, particularly when compared to the other algorithms. The difficulty of expanding this algorithm increases geometrically with the number of sides.

In the following seven diagrams (Figures C-1 through C-7), the lines represent the outline of the input figure, the crosses (X's) represent the input test points, and the encircled crosses are special test points; the vertices of the input figure.

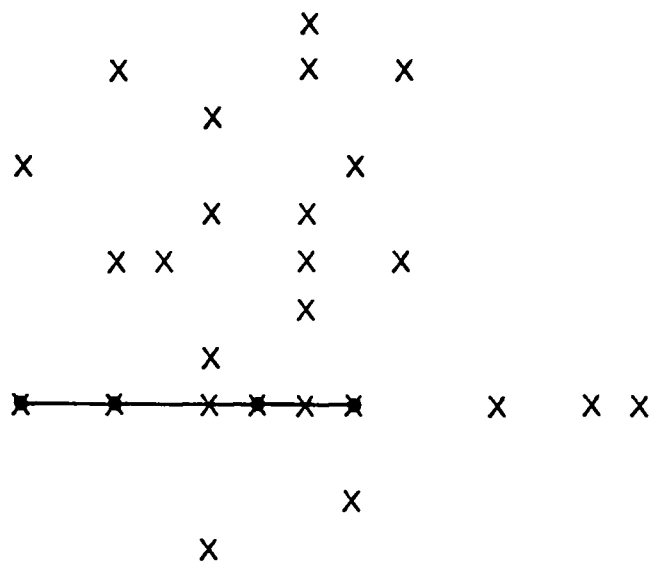


Figure C-1.

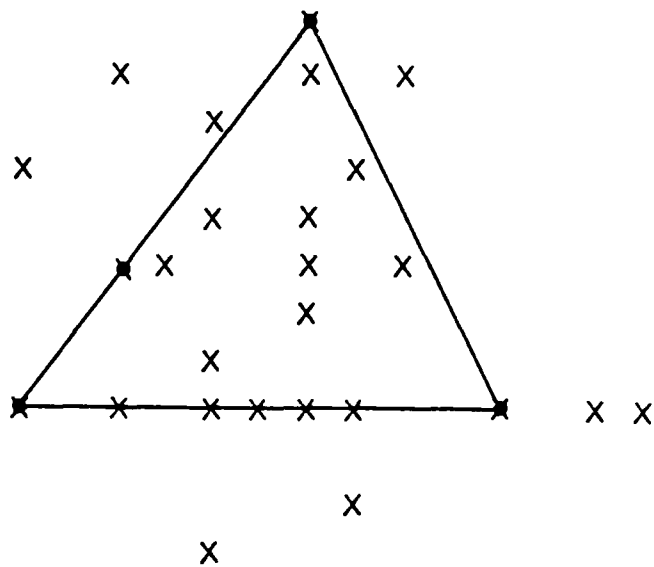


Figure C-2.

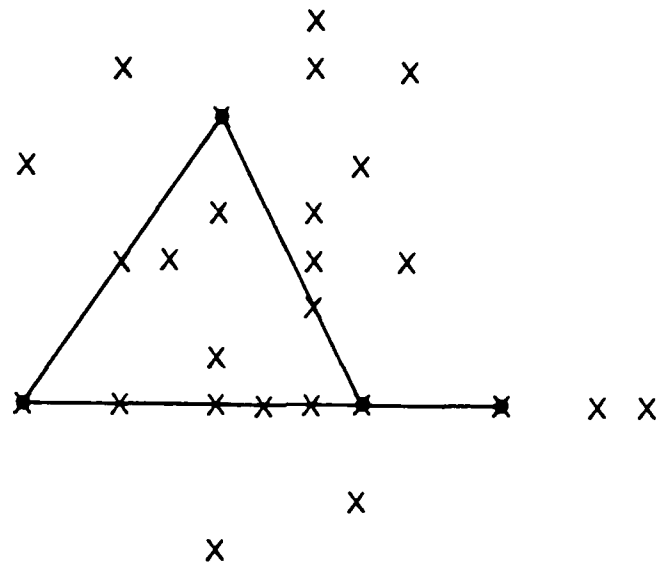


Figure C-3.

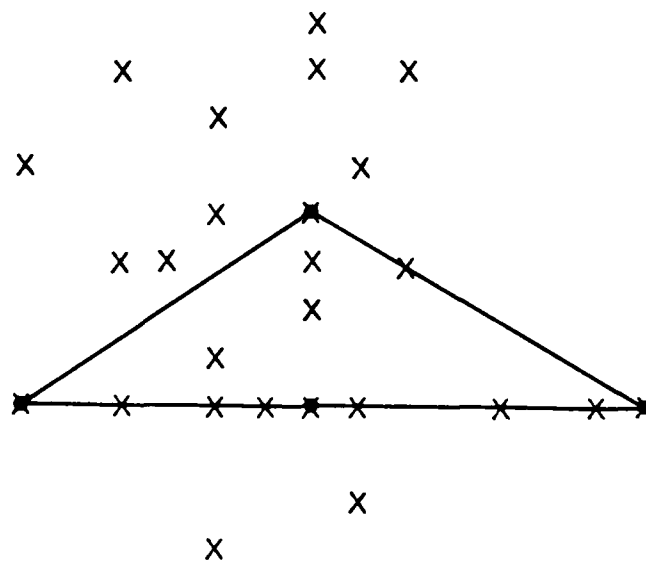


Figure C-4.

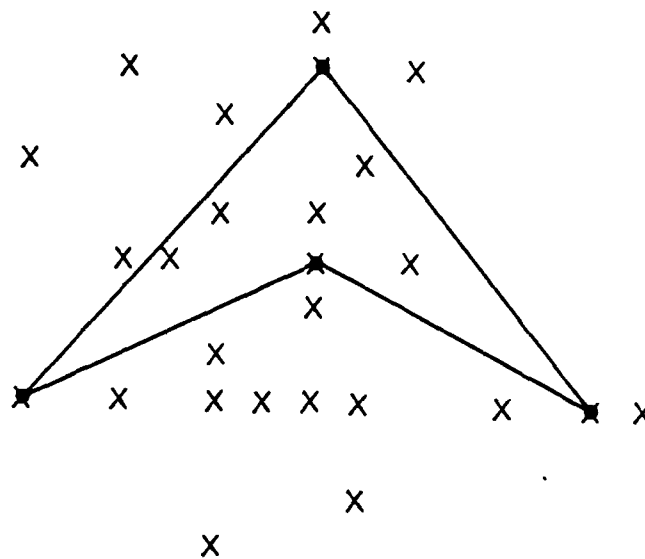


Figure C-5.

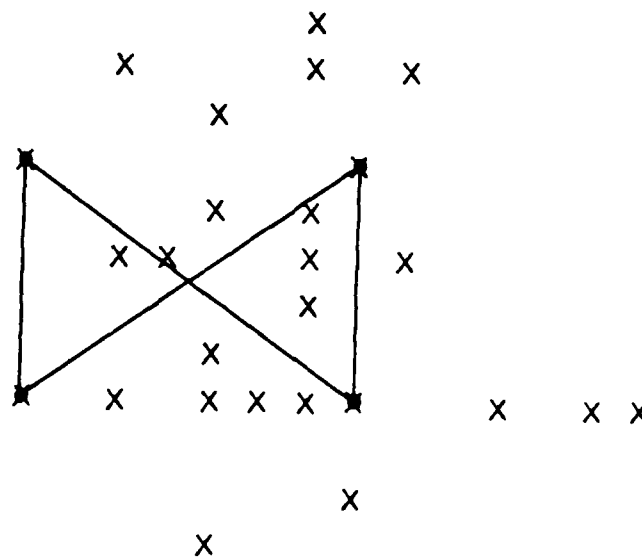


Figure C-6.

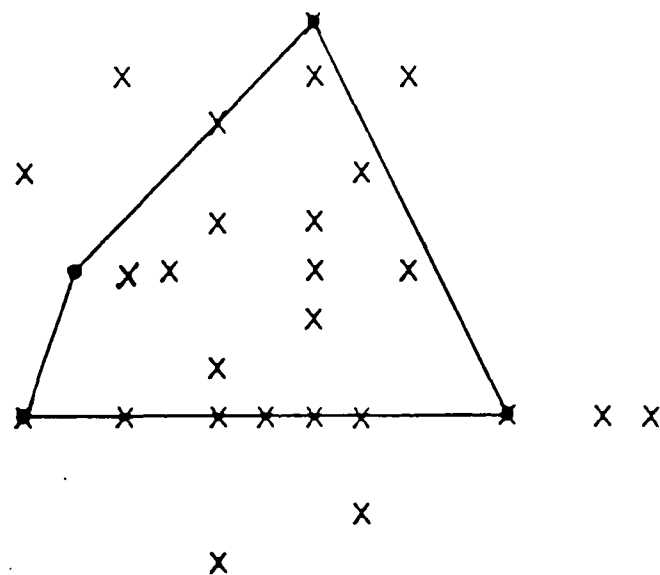


Figure C-7.

IMPLEMENTED ALGORITHMS AND TEST RESULTS

PROGRAM QUADTEST (INPUT, OUTPUT);

(*THIS PROGRAM TESTS TO SEE IF A POINT FALLS WITHIN THE CURVE OF
A 4 SIDED POLYGON GIVEN THE 4 POINTS IN ORDER AND THE COORDINATES
OF THE POINT IN QUESTION. IT IS DESIGNED TO HANDLE CASES OF BOTH
CONVEX AND CONCAVE FIGURES.

WRITTEN BY: ELLEN DRELL
DATED: 12/26/84

ACKNOWLEDGMENTS: PROF. ROBERT HENDERSON, C. S. U. N.
MS. ELAINE SCHMIDT, J. P. L.

*)

TYPE

POINT = ARRAY [1..2] OF REAL;
TRIANGLE = ARRAY [1..3] OF INTEGER;

VAR

A : ARRAY [1..4] OF POINT;
P : POINT;
T1, T2 : TRIANGLE;
CENTER : INTEGER;
LFLAG, I, J, K : INTEGER;
INFLECTION : INTEGER;
RESPONSE, REPLY : CHAR;

(*-----*)
(*-----*)

FUNCTION TRIANGLETEST (V1, V2, V3 : INTEGER; P : POINT) : BOOLEAN;

(* THIS ROUTINE TESTS IF THE LINE CONNECTING THE POINT P WITH EACH
VERTEX INTERSECTS THE LINE OF OPPOSITE SIDE OF THE TRIANGLE AT A
POINT WITHIN THE CONFINES OF THE TRIANGLE. IF THIS CONDITION
HOLDS TRUE FOR ALL 3 CASES THEN THE POINT IS IN THE INTERIOR OF
THE TRIANGLE DEFINED.

*)

VAR
VAL1, VAL2, LX, RX, M : REAL;
I, COUNT : INTEGER;
V, S1, S2, TEMP : POINT;

BEGIN

COUNT := 0;
V := A[V1];
S1 := A[V2];
S2 := A[V3];

FOR I = 1 TO 3 DO BEGIN

(*FOR THE THREE SIDES TEST IF THE INTERSECTION OF THE 2 LINES
OCCURS AT A POINT BETWEEN THE LEFT AND RIGHT ENDPOINTS OF THE
SIDE. USE THE INTERMEDIATE VALUE THEORM TO DETERMINE IF THE
SOLUTION EXISTS IN THE INTERVAL. C-10

*)

```
IF ( S1[1] < S2[1] )
  THEN M := ((S1[2]-S2[2])/(S1[1]-S2[1])) (*SLOPE*)
  ELSE M := 99999; (*SLOPE IS INFINITY SINCE VERTICAL*)
```

(*THE CASE WHERE THE VERTEX POINT AND THE POINT IN QUESTION
HAVE THE SAME X VALUE ONLY THE Y COORDINATE NEEDS TO BE
TESTED. A LINEAR FORMULA SIMILIAR TO THAT IN THE LINEARITY
PROCEDURE IS USED AND A COMPARISON IS DONE USING THE TWO
DIFFERENT Y VALUES WHILE HOLDING THE X CONSTANT.

*)

```
IF V[1]=P[1] (*TEST FOR SLOPE OF INFINITY*)
  THEN BEGIN
    VAL1:=(V[2]-S1[2]) - (M*(V[1]-S1[1]));
    VAL2:=(P[2]-S1[2]) - (M*(V[1]-S1[1]));
    IF (ABS(VAL1) > ABS(VAL2)) THEN COUNT :=COUNT +1;
    END (*THEN*)
  ELSE BEGIN
    LX:=S1[1]; RX:=S2[1];
    VAL1:=(V[2]-S2[2])-(M*(LX-S2[1]))+ (((V[2]-P[2])/(V[1]-P[1]))*(LX-V[1]));
    VAL2:=(V[2]-S1[2])-(M*(RX-S1[1]))+ (((V[2]-P[2])/(V[1]-P[1]))*(RX-V[1]));

    IF (VAL1 * VAL2) <= 0 THEN COUNT :=COUNT + 1;
  END (*ELSE*)
```

```
    TEMP := V;
    V := S1; (*SWITCH TO THE NEXT EDGE OF TRIANGLE*)
    S1:= S2;
    S2:=TEMP;
```

```
END (*FOR*);
```

```
IF COUNT = 3 THEN TRIANGLETEST := TRUE (*POINT WITHIN*)
  ELSE TRIANGLETEST := FALSE;
```

```
END;
```

(*-----*)

```
FUNCTION CONCAVITY : INTEGER;
```

(*THIS ROUTINE TEST TO SEE IF THE FIGURE IS CONCAVE OR CONVEX. IT
TESTS BY USE OF PERMUTATIONS, IF ANY OF THE VERTEX POINTS LIES
WITHIN THE TRIANGLE DEFINED BY THE OTHER 3 POINTS. IF CONCAVITY
IS DETERMINED THEN THE FUNCTION RETURNS THE INTERNAL POINT AS THE
POINT OF INFLECTION. IF THE FIGURE IS DETERMINED TO BE CONVEX
VERTEX 1 IS RETURNED BY DEFAULT.

*)

```
BEGIN
```

```
IF ((TRIANGLETEST(1,2,3,AP[1])=TRUE) OR (TRIANGLETEST(3,4,1,AP[2])=TRUE))
  THEN CONCAVITY :=2;
  ELSE CONCAVITY :=1;
```

```
END;
```

(*-----*)
PROCEDURE ENTERVALS;

(*THIS PROCEDURE IS THE PROMPT ROUTINE TO ENTER THE VALUES INTO THE ARRAY*)
(*OF POINTS*)

```

VAR
  LFLAG : INTEGER;

BEGIN
  WRITELN;
  WRITELN ('ENTER EACH OF THE 4 POINTS IN ORDER ');
  WRITELN ('ORDER IS DETERMINED BY THE CONSECUTIVE POINTS BEING JOINED BY A SIDE');
  WRITELN ('ENTER EACH OF THE 4 POINTS AS AN ORDERED PAIR');
  WRITELN;
  FOR I:= 1 TO 4 DO BEGIN
    WRITE ('ENTER THE X COORDINATE OF VERTEX ', I:2, ' ');
    READLN (A[I,1]);
    WRITE ('ENTER THE Y COORDINATE OF VERTEX ', I:2, ' ');
    READLN(A[I,2]);
    WRITELN;
  END;
  WRITELN;

END;

(*-----*)
FUNCTION NEWPT: INTEGER;
(*THIS ROUTINE ENTERS THE POINT TO BE CONSIDERED AND DOES *)
(*VERIFICATION CHECKING ON EXISTING COORDINATES*)
BEGIN
  NEWPT:=0;
  WRITE ('ENTER THE X COORDINATE OF THE POINT P ');
  READLN (P[1]);
  WRITE ('ENTER THE Y COORDINATE OF THE POINT P ');
  READLN (P[2]);
  WRITELN;
  FOR I:= 1 TO 4 DO
    IF ((A[I,1]=P[1]) AND (A[I,2]=P[2])) (*CHECK FOR A VERTEX POINT*)
      THEN NEWPT:=1;
  END;

(*-----*)
(*                               BEGIN MAIN ROUTINE                               *)
(*-----*)

BEGIN
  REPEAT BEGIN

    INTERVALS;
    WRITELN ('ENTER THE POINT UNDER CONSIDERATION AS AN ORDERED PAIR ');
    WRITELN;

    REPEAT BEGIN
      LFLAG:=NEWPT;
      IF LFLAG < 1 THEN BEGIN
        END (*END IF*);

      CASE LFLAG OF
        0: BEGIN (*NON VERTEX CASE*)

          INFLECTION := CONCAVITY;
          T1[1]:=INFLECTION; (*DEFINE TRIANGLES BASED ON INFLECTION *)
          T1[2]:=INFLECTION+1; (*POINTS *)
          T1[3]:=INFLECTION+2; C-12

```

```

T2C11:=INFLECTION;
T2C21:=INFLECTION+2;
IF INFLECTION = 2
  THEN T2C31:=INFLECTION-1
  ELSE T2C31:=INFLECTION + 3;

IF (TRIANGLETEST(T1C11,T1C21,T1C31,P) = TRIANGLETEST(T2C11,T2C21,T2C31,P)) THEN
  IF TRIANGLETEST(T1C11,T1C21,T1C31,P) THEN
    WRITELN('THE POINT IS AN INTERIOR POINT OF THE QUADRALATERAL')
  ELSE
    WRITELN('THE POINT IS AN EXTERIOR POINT OF THE QUADRALATERAL')
  ELSE
    WRITELN('THE POINT IS AN INTERIOR POINT OF THE QUADRALATERAL')

END (*O CONDITION OF CASE*);
1 . WRITELN ('THE POINT WAS A VERTEX OF THE QUADRALATERAL');
END (*CASE*);

WRITELN;
WRITE ('ANOTHER POINT TO TEST WITHIN THIS QUADRALATERAL? Y/N ');
READLN (REPLY);
WRITELN;
END (*REPLY REPEAT*);
UNTIL REPLY IS 'Y';
WRITELN;
WRITE ('ANOTHER QUADRALATERAL TO TEST? Y/N ');
READLN (RESPONSE);
WRITELN;
END (*RESPONSE REPEAT*);
UNTIL RESPONSE IS 'Y';
END.

```

QUADRILATERAL	X-COORDINATE	Y-COORDINATE
C-1	0.0000	0.0000
	2.0000	0.0000
	4.0000	0.0000
	6.0000	0.0000

X-COORDINATE

Y-COORDINATE

CHECKING THE INPUT (X, Y) POINTS

0.0000
2.0000
4.0000
6.0000

0.0000
0.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT Y

GENERATED RESULT

ACTUAL RESULT

4.0000	4.0000
4.0000	1.0000
2.0000	3.0000
6.0000	3.0000
8.0000	3.0000
5.0000	0.0000
2.0000	7.0000
8.0000	7.0000
6.0000	2.0000
-3.0000	4.0000
-2.0000	7.0000
0.0000	0.0000
2.0000	0.0000
4.0000	0.0000
6.0000	0.0000
10.0000	0.0000
6.0000	8.0000
3.0000	3.0000
4.0000	6.0000
7.0000	0.0000
6.0000	4.0000
13.0000	0.0000
6.0000	7.0000
12.0000	0.0000
0.0000	5.0000
7.0000	5.0000

OUT
OUT
OUT
OUT
OUT
ON THE LINE
OUT
OUT
OUT
OUT
OUT
CORNER
CORNER
CORNER
CORNER
OUT
OUT
OUT
OUT
OUT
OUT
OUT
OUT
OUT
OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-2

0.0000
10.0000
6.0000
3.0000

0.0000
0.0000
8.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	CORNER	CORNER
4.0000	6.0000	OUT	OUT
7.0000	0.0000	IN	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-3

0.0000
4.0000
7.0000
10.0000

0.0000
6.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000		IN
4.0000	1.0000		IN
2.0000	3.0000		ON THE LINE
6.0000	3.0000		OUT
8.0000	3.0000		OUT
5.0000	0.0000		ON THE LINE
2.0000	7.0000		OUT
8.0000	7.0000		OUT
6.0000	2.0000		ON THE LINE
-3.0000	4.0000		OUT
-2.0000	7.0000		OUT
0.0000	0.0000		CORNER
2.0000	0.0000		ON THE LINE
4.0000	0.0000		ON THE LINE
6.0000	0.0000		ON THE LINE
10.0000	0.0000		CORNER
6.0000	8.0000		OUT
3.0000	3.0000		IN
4.0000	6.0000		CORNER
7.0000	0.0000		CORNER
6.0000	4.0000		OUT
13.0000	0.0000		OUT
6.0000	7.0000		OUT
12.0000	0.0000		OUT
0.0000	5.0000		OUT
7.0000	5.0000		OUT

QUADRILATERAL	X-COORDINATE	Y-COORDINATE
C-4	0.0000	0.0000
	6.0000	4.0000
	13.0000	0.0000
	6.0000	0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000		OUT
4.0000	1.0000		IN
2.0000	3.0000		OUT
6.0000	3.0000		IN
8.0000	3.0000		OUT
5.0000	0.0000		ON THE LINE
2.0000	7.0000		OUT
8.0000	7.0000		OUT
6.0000	2.0000		IN
-3.0000	4.0000		OUT
-2.0000	7.0000		OUT
0.0000	0.0000		CORNER
2.0000	0.0000		ON THE LINE
4.0000	0.0000		ON THE LINE
6.0000	0.0000		CORNER
10.0000	0.0000		ON THE LINE
6.0000	8.0000		OUT
3.0000	3.0000		OUT
4.0000	6.0000		OUT
7.0000	0.0000		ON THE LINE
6.0000	4.0000		CORNER
13.0000	0.0000		CORNER
6.0000	7.0000		OUT
12.0000	0.0000		ON THE LINE
0.0000	5.0000		OUT
7.0000	5.0000		OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-5

0.0000
6.0000
12.0000
6.0000

0.0000
7.0000
0.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X

INPUT Y

GENERATED RESULT

ACTUAL RESULT

4.0000	4.0000	IN	IN
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	CORNER	CORNER
8.0000	3.0000	IN	IN
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	CORNER	CORNER
12.0000	0.0000	CORNER	CORNER
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

 QUADRILATERAL

C-6

 X-COORDINATE

0.0000
 0.0000
 7.0000
 7.0000

 Y-COORDINATE

0.0000
 5.0000
 0.0000
 5.0000

CHECKING THE INPUT (X, Y) POINTS

----- INPUT X -----	----- INPUT Y -----	----- GENERATED RESULT -----	----- ACTUAL RESULT -----
4.0000	4.0000		OUT
4.0000	1.0000		OUT
2.0000	3.0000		IN
6.0000	3.0000		IN
8.0000	3.0000		OUT
5.0000	0.0000		OUT
2.0000	7.0000		OUT
8.0000	7.0000		OUT
6.0000	2.0000		IN
-3.0000	4.0000		OUT
-2.0000	7.0000		OUT
0.0000	0.0000		CORNER
2.0000	0.0000		OUT
4.0000	0.0000		OUT
6.0000	0.0000		OUT
10.0000	0.0000		OUT
6.0000	8.0000		OUT
3.0000	3.0000		OUT
4.0000	6.0000		OUT
7.0000	0.0000		CORNER
6.0000	4.0000		IN
13.0000	0.0000		OUT
6.0000	7.0000		OUT
12.0000	0.0000		OUT
0.0000	5.0000		CORNER
7.0000	5.0000		CORNER

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-7

0.0000
10.0000
6.0000
2.0000

0.0000
0.0000
8.0000
4.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	IN	IN
4.0000	6.0000	IN	ON THE LINE
7.0000	0.0000	IN	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

Program R_J_G

This program reads in (x,y) values a three quadrilateral having 4 corners. A test point (x,y) is checked to determine if the inputted point falls in the quadrilaterals

Algorithm designed by: Robert J. Gardner, J.P.L.
Implemented by: Nick Covella, J.P.L.
Dated: March, 1985

```

Real*8 XArray(1,4),YArray(1,4)
Real*8 NewXArray(1,4),NewYArray(1,4)
Real*8 InXcoord, InYcoord
Character*1 Resp1,Resp2,Flag
Integer Quadcntr

```

```

C *****
C ***** M A I N ** P R O G R A M *****
C *****

```

```

Open(Unit = 1,File='Rjg.Dat', Status = 'New')
Open(Unit = 2,File='Pts.Dat', Status = 'Old')
Call Skip(1)
Call Skip(2)
Call Prompt1(Resp1)
Quadcntr = 0
Do While ((Resp1 .EQ. 'Y') .OR. (Resp1 .EQ. 'y'))
  Quadcntr = Quadcntr + 1
  Open(Unit = 3,File='Test.Dat',Status = 'Old')
  Call Initialize(XArray,YArray,NewXArray,NewYArray)
  Call LoadArrays(XArray,YArray,Quadcntr)
  Call UnloadArrays(XArray,YArray,Quadcntr)
  Call Prompt2(Resp2,InXcoord,InYcoord)
  Do While ((Resp2 .EQ. 'Y') .OR. (Resp2 .EQ. 'y'))
    Call TestPt(InXcoord,InYcoord,XArray,YArray,Flag)
    Call Transform(InXcoord,InYcoord,XArray,YArray,
      NewXArray,NewYArray)

    Call Test_It(NewXArray,NewYArray,InXcoord,InYcoord)
    GO TO 29
  End Do
  Call Prompt2(Resp2,InXcoord,InYcoord)
  If ((Resp2 .EQ. 'N') .OR. (Resp2 .EQ. 'n')) Then
    Close(Unit=3)
  End If
End Do
Call Prompt1(Resp1)
End Do
Write(5,55)

```

```

Close(Unit=1)
Close(Unit=2)
55 Format(T2,'Normal Completion of Program by N. Covella')
Stop
End

```

```

C *****
C *****
Subroutine TestPt(InXcoord,InYcoord,XArray,YArray,Flag)

```

This subroutine checks if the point is one of the corners

C

```

Real*8 XArray(1,4),YArray(1,4)
Real*8 InXcoord,InYcoord
Character*1 Flag

```

```

Flag = 'F'
Do 10 I = 1,4
  IF (InXcoord .EQ. XArray(1,I)) Then
    IF (InYcoord .EQ. YArray(1,I)) Then
      Flag = 'T'
    End If
  End If
End Do

```

10 Continue

```

Return
End

```

```

C*****
C*****

```

```

Subroutine Initialize(XArray,YArray,NewXArray,NewYArray)

```

C

C

C

```

This subroutine initializes the two arrays (i.e. quadrilateral
and Analyst) to zero so that any unknown values may be detected

```

```

Real*8 XArray(1,4),YArray(1,4)
Real*8 NewXArray(1,4),NewYArray(1,4)
Integer Index

```

```

Do 10 Index = 1,4
  Xarray(1,Index) = 0
  Yarray(1,Index) = 0
  NewXarray(1,Index) = 0
  NewYarray(1,Index) = 0

```

10 Continue

```

Return
End

```

```

C*****
C*****

```

```

Subroutine LoadArrays(XArray,YArray,Kount)

```

C

C

C

C

```

This subroutine loads the 3 quadrilaterals with (x,y) coordinates
for each corner of the corresponding quadrilaterals.

```

```

Real*8 XArray(1,4),YArray(1,4)
Integer Kount

```

```

Write(5,01)
Write(1,01)
Do 10 Index = 1,4
  Write(5,97) Index,Kount
  Read (2,99) XArray(1,Index)
  Write(5,98) Index,Kount
  Read (2,99) YArray(1,Index)

```

10 Continue

```

01      Format(' ')
97      Format(T1, ' Input X_coordinate #', I1, T23,
      ' for Quadrilateral #', I1)
98      Format(T1, ' Input Y_coordinate #', I1, T23,
      ' for Quadrilateral #', I1)
99      Format(F9.4)

```

```

      Return
      End

```

```

C*****
C*****

```

```

      Subroutine UnLoadArrays(XArray, YArray, QuadNo)

```

```

C
C      This subroutine displays the inputted values of the
C      (x,y) coordinates for each quadrilateral and the
C      (x,y) coordinate that is to be checked whether or not it
C      is in any of the three quadrilaterals.
C

```

```

      Real*8 XArray(1,4), YArray(1,4)
      Integer QuadNo, Index

```

```

      Write(5,96)
      Write(5,97)
      Write(1,96)
      Write(1,97)

```

```

      Do 10 Index = 1,4

```

```

          Write(5,98) QuadNo, XArray(1, Index), YArray(1, Index)
          Write(1,98) QuadNo, XArray(1, Index), YArray(1, Index)

```

```

10      Continue

```

```

      Write(1,97)

```

```

96      Format(T3, 'Quadrilateral', T18, 'X-Coord', T30,
      'Y-Coord')

```

```

97      Format(T3, '-----', T18, '-----', T30,
      '-----')

```

```

98      Format(T9, I1, T15, F9.4, T27, F9.4)

```

```

      Return
      End

```

```

C*****
C*****

```

```

      Subroutine Transform(InXcoord, InYcoord, XArray, YArray,
      NewXArray, NewYArray)

```

```

C
C      This subroutine normalizes the inputted quadrilaterals to
C      coordinates (0,0) for easy calculations based on the origin.
C

```

```

      Real*8 XArray(1,4), YArray(1,4)
      Real*8 NewXArray(1,4), NewYArray(1,4)
      Real*8 InXcoord, InYcoord
      Integer Index

```

```

      Do 10 Index = 1,4

```

```

          NewXArray(1, Index) = XArray(1, Index) - InXcoord

```

```

          NewYArray(1, Index) = YArray(1, Index) - InYcoord

```

```

10      Continue

```

Return
End

C*****
C*****

Subroutine Test_It(NewXArray, NewYArray, X, Y)

Integer EValue, Index, J
Real*8 NewXArray(1, 4), NewYArray(1, 4)
Real*8 X, Y, A(4), M(4), Sum, R, Tmp1, Tmp2
Character*1 Loop, Exit

R = 0.0001

Do 10 Index = 1, 4

M(Index) = DSQRT((NewXArray(1, Index) ** 2) +
+ (NewYArray(1, Index) ** 2))

If (M(Index) .LE. R) Then

Prevention of division by zero.

Write(5, 99)

Write(1, 99)

GOTO 13

End If

10 Continue

Index = 1

J = 2

Sum = 0

Do While (Index .LE. 4)

If (Index .EQ. 4) Then

J = 1

End If

Tmp1 = ((NewXArray(1, Index) * NewYArray(1, J)) -
+ (NewXArray(1, J) * NewYArray(1, Index)))

4 Tmp2 = ((NewXArray(1, Index) * NewXArray(1, J)) +
+ (NewYArray(1, Index) * NewYArray(1, J)))

A(Index) = DACOSD(Tmp2 / (M(Index) * M(J)))

If (A(Index) .GT. 179.88) Then

Write(5, 30)

30 Format(T2, 'The point is ON THE LINE.')

Write(1, 31)

31 Format(T2, 'The point is ON THE LINE.')

GO TO 13

Endif

If (Tmp1 .LT. 0.0) Then

A(Index) = -A(Index)

Endif

Sum = Sum + A(Index)

Index = Index + 1

J = J + 1

End Do

EValue = JIDNNT(ABS(Sum / 360.0))

If (EValue .EQ. 0) Then

Write(5, 27)

Write(1, 27)

Else

Write(5, 28)

Write(1, 28)

Endif

```

27      Format(T2, 'Point not enclosed. ')
28      Format(T2, 'Point is enclosed. ')
99      Format(T2, 'Point of interest is on the VERTEX')
10      Continue
      Return
      End

```

```

C*****
C*****

```

```

      Subroutine Skip(Lines)
C
C      This subroutine skips the indicating number of lines by
C      the place where skip has been called from (i. e. many places)
C
      Integer I, Lines
      Do 10 I = 1, Lines
        Write(5, 11)
10      Continue
11      Format(' ')
      Return
      End

```

```

C*****
C*****

```

```

      Subroutine Prompt1(Response)
C
C      This subroutine Prompts the user if he/she wants to input
C      data to determine if the point of interest falls within
C      any one of three quadrilaterals.
C
      Character*1 Response
C
      Call Skip(1)
      Write(5, 55)
      Write(5, 56)
      Write(5, 57)
      Call Skip(1)
      Read(5, 99) Response
11      Format(' ')
55      Format(T2, 'Would you like to input some coordinates to ')
56      Format(T2, 'determine if a point falls in an inputted ')
57      Format(T2, 'quadrilateral?? Y/N ')
99      Format(A1)
      Return
      End

```

```

C*****
C*****

```

```

      Subroutine Prompt2(Response, InXcoord, InYcoord)
C
C      This subroutine Prompts the user if he/she wants to input
C      data to determine if the point of interest falls within
C      any one of three quadrilaterals.

```

Real*8 InXcoord, InYcoord
Character*1 Response

```
Call Skip(1)
write(5, 55)
Read(5, 89) Response
Call Skip(1)
If ((Response . EQ. `Y`) . OR. (Response . EQ. `y`)) Then
    write(1, 01)
    Write(1, 01)
    Write(5, 97)
    Read(3, 99) InXcoord
    Write(1, 95) InXcoord
    Write(1, 01)
    Write(5, 98)
    Read(3, 99) InYcoord
    Write(1, 96) InYcoord
    write(1, 01)
    Write(1, 01)
Endif
```

```
01  Format(` `)
55  Format(T2, ` Input a point to test?? Y/N `)
89  format(A1)
95  Format(T2, `Inputted X-Coordinate ==> `, F9.6)
96  Format(T2, `Inputted Y-Coordinate ==> `, F9.6)
97  Format(T2, `Input X-coord: `)
98  Format(T2, `Input Y-coord: `)
99  Format(F9.4)
```

Return
End

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-1

0.0000
2.0000
4.0000
6.0000

0.0000
0.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X

INPUT Y

GENERATED RESULT

ACTUAL RESULT

4.0000	4.0000	OUT	OUT
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	OUT	OUT
8.0000	3.0000	OUT	OUT
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	CORNER	CORNER
4.0000	0.0000	CORNER	CORNER
6.0000	0.0000	CORNER	CORNER
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	OUT	OUT
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-2

0.0000
10.0000
6.0000
3.0000

0.0000
0.0000
8.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	CORNER	CORNER
4.0000	6.0000	OUT	OUT
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-3

0.0000
4.0000
7.0000
10.0000

0.0000
6.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	ON THE LINE	ON THE LINE
6.0000	3.0000	OUT	OUT
8.0000	3.0000	OUT	OUT
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	ON THE LINE	ON THE LINE
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	CORNER	CORNER
7.0000	0.0000	CORNER	CORNER
6.0000	4.0000	OUT	OUT
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-4

0.0000
6.0000
13.0000
6.0000

0.0000
4.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	OUT	OUT
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	OUT	OUT
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	CORNER	CORNER
10.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	CORNER	CORNER
13.0000	0.0000	CORNER	CORNER
6.0000	7.0000	OUT	OUT
12.0000	0.0000	ON THE LINE	ON THE LINE
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-5

0.0000
6.0000
12.0000
6.0000

0.0000
7.0000
0.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	CORNER	CORNER
8.0000	3.0000	IN	IN
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	CORNER	CORNER
12.0000	0.0000	CORNER	CORNER
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-6

0.0000
0.0000
7.0000
7.0000

0.0000
5.0000
0.0000
5.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	OUT	OUT
4.0000	1.0000	OUT	OUT
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	OUT	OUT
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	CORNER	CORNER
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	CORNER	CORNER
7.0000	5.0000	CORNER	CORNER

 QUADRILATERAL

 X-COORDINATE

 Y-COORDINATE

C-7

0.0000
 10.0000
 6.0000
 2.0000

0.0000
 0.0000
 8.0000
 4.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	IN	IN
4.0000	6.0000	ON THE LINE	ON THE LINE
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

Program Test_it

This program accepts the vertices of a 4-sided quadrilateral and determines whether or not a given point lies inside or outside the figure.

Algorithm designed by: Fred Lesh, J.P.L.

Implemented by: Nick Covella, J.P.L.

Dated: January, 1985

```
Integer Quadcntr, Index, Flag, NoOfSides
Real*8 Sigma, Temp, X, Y, U(8), V(8), NoOfEnclosures
Real*8 S(8), InX(8, 8), InY(8, 8), Tmp(8), Delta(8), Epsilon
Character*1 Answ, Resp
Data Quadcntr, Epsilon, NoOfSides /0, 0.0001, 4/
```

```
Open(Unit=1, File='Freds. Dat' , Status='New')
Open(Unit=2, File='Pts. Dat' , Status='Old')
Open(Unit=4, File='FredRes. Dat', Status='Old')
Open(Unit=3, File='Test. Dat' , Status='Old')
```

```
Quadcntr = Quadcntr + 1
Index = 1
Do While (Index .LE. NoOfSides)
  Write(1,01)
  Write(5,99) Index, Quadcntr
  Write(1,99) Index, Quadcntr
  Read(2,97) InX(Quadcntr, Index)
  Write(1,97) InX(Quadcntr, Index)
  Write(1,01)
  Write(5,98) Index, Quadcntr
  Write(1,98) Index, Quadcntr
  Read(2,97) InY(Quadcntr, Index)
  Write(1,97) InY(Quadcntr, Index)
  Index = Index + 1
```

```
End do
NoOfEnclosures = 0
Write(1,01)
Write(1,96)
Read(3,97) X
Write(1,97) X
Write(1,01)
Write(1,95)
Read(3,97) Y
Write(1,97) Y
Index = 1
U(Quadcntr) = InX(Quadcntr, Index) - X
V(Quadcntr) = InY(Quadcntr, Index) - Y
Sigma = 0
Tmp(Quadcntr) = Scale(U(Quadcntr), V(Quadcntr))
Flag = 1
Index = 2
Write(5,31) X, Y
```

```
Format(T2, 'Testing point [ ', F5.2, ' ', F5.2, ' ] *****')
Do While ((Index .LE. 5) .AND. (Flag .EQ. 1))
  If (Index .EQ. 5) Then
    Index = 1
    Flag = 0
  Endif
  U(Index) = InX(Quadcntr, Index) - X
  V(Index) = InY(Quadcntr, Index) - Y
  If ((U(Index) .EQ. 0) .AND. (V(Index) .EQ. 0)) Then
```

```

        Sigma = 8
        GO TO 11
    Endif
    S(Index) = Scale(U(Index),V(Index))
    Temp = S(Index) - Tmp(Quadcntr)
    IF (Abs(Temp) .LT. (4 - Epsilon)) Then
        Delta(Index) = Temp
    Else IF (Temp .LT. (-4 - Epsilon)) Then
        Delta(Index) = Temp + 8
    Else IF (Temp .GT. (4 + Epsilon)) Then
        Delta(Index) = Temp - 8
    Else
        Sigma = 8
        GO TO 11
    End if

    Tmp(Quadcntr) = S(Index)
    Sigma = Sigma + Delta(Index)
    Index = Index + 1
End do
11 NoOfEnclosures = NINT(ABS(Sigma/8))
   Write(3,94) NoOfEnclosures
   Write(1,01)
   Write(1,94) NoOfEnclosures
   IF (NoOfEnclosures .LT. 1) Then
       Write(5,43) Quadcntr
       Write(1,01)
       Write(1,43) Quadcntr
   Else
       Write(5,44) Quadcntr
       Write(1,01)
       Write(1,44) Quadcntr
   Endif
   Write(3,01)
   Write(3,93)
   Write(1,01)
   Write(1,93)
   Read(4,92) Answ
   Write(1,92) Answ
   IF ((Answ .EQ. 'y') .OR. (Answ .EQ. 'Y')) Then
       GO TO 20
   Else
       Close(Unit=3)
       Write(5,91)
       Write(1,01)
       Write(1,91)
       Read(4,90) Resp
       Write(1,90) Resp
       IF ((Resp .EQ. 'y') .OR. (Resp .EQ. 'Y')) Then
           GO TO 30
       Endif
   Endif
Endif
Print *, 'Normal completion of program '

01 Format(T2,' ')
44 Format(T2,'Point is IN quadrilateral #',I1)
43 Format(T2,'Point is OUT of quadrilateral #',I1)
90 Format(A1)
91 Format(T2,'More quadrilaterals??')
92 Format(A1)
93 Format(T2,'More test points??')C-35

```

```

94 Format(T2, 'Value of Enclosures ==> ', F15.13)
95 Format(T2, 'Input Y value for the point in question. ')
96 Format(T2, 'Input X value for the point in question. ')
97 Format(F15.4)
98 Format(T2, 'Input y-coord #', I1, T19, 'of quad #', I1)
99 Format(T2, 'Input x-coord #', I1, T19, 'of quad #', I1)
    Stop
    End

```

```

C*****
C*****

```

```

Real*8 Function Scale(U,V)

```

```

C
C This function returns the distance around the unit square (i.e
C the size being a 1 x 1 square) in a counter clockwise manner.
C

```

```

Real*8 U,V

```

```

If ((ABS(V) .GE. (ABS(U))) .AND. (V .GT. 0)) Then
    Scale = 1 - (U/V)
Else If ((ABS(U) .GE. (ABS(V))) .AND. (U .LT. 0)) Then
    Scale = 3 + (V/U)
Else If ((ABS(V) .GE. (ABS(U))) .AND. (V .LT. 0)) Then
    Scale = 5 - (U/V)
Else If ((ABS(U) .GE. (ABS(V))) .AND. (U .GT. 0)) Then
    Scale = 7 + (V/U)
End if

```

```

Return
End

```

```

C*****
C*****

```

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-1

0.0000
2.0000
4.0000
6.0000

0.0000
0.0000
0.0000
0.0000

CHECKING THE INPUT (X,Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	OUT	OUT
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	OUT	OUT
8.0000	3.0000	OUT	OUT
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	IN	CORNER
4.0000	0.0000	IN	CORNER
6.0000	0.0000	IN	CORNER
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	OUT	OUT
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-2

0.0000
10.0000
6.0000
3.0000

0.0000
0.0000
8.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	ON THE LINE
10.0000	0.0000	IN	CORNER
6.0000	8.0000	IN	CORNER
3.0000	3.0000	IN	CORNER
4.0000	6.0000	OUT	OUT
7.0000	0.0000	IN	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

QUADRILATERAL	X-COORDINATE	Y-COORDINATE
C-3	0.0000	0.0000
	4.0000	6.0000
	7.0000	0.0000
	10.0000	0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	IN	ON THE LINE
6.0000	3.0000	OUT	OUT
8.0000	3.0000	OUT	OUT
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	ON THE LINE
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	ON THE LINE
10.0000	0.0000	IN	CORNER
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	IN	CORNER
7.0000	0.0000	IN	CORNER
6.0000	4.0000	OUT	OUT
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-4

0.0000
6.0000
13.0000
6.0000

0.0000
4.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	OUT	OUT
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	OUT	OUT
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	CORNER
10.0000	0.0000	IN	ON THE LINE
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	IN	ON THE LINE
6.0000	4.0000	IN	CORNER
13.0000	0.0000	IN	CORNER
6.0000	7.0000	OUT	OUT
12.0000	0.0000	IN	ON THE LINE
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-5

0.0000
6.0000
12.0000
6.0000

0.0000
7.0000
0.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

----- INPUT X -----	----- INPUT Y -----	----- GENERATED RESULT -----	----- ACTUAL RESULT -----
4.0000	4.0000	IN	IN
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	CORNER
8.0000	3.0000	IN	IN
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	CORNER
12.0000	0.0000	IN	CORNER
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

 QUADRILATERAL

 X-COORDINATE

 Y-COORDINATE

C-6

0.0000
 0.0000
 7.0000
 7.0000

0.0000
 5.0000
 0.0000
 5.0000

CHECKING THE INPUT (X, Y) POINTS

 INPUT X

 INPUT Y

 GENERATED RESULT

 ACTUAL RESULT

4.0000	4.0000	OUT	OUT
4.0000	1.0000	OUT	OUT
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	OUT	OUT
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	IN	CORNER
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	IN	CORNER
7.0000	5.0000	IN	CORNER

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-7

0.0000
10.0000
6.0000
2.0000

0.0000
0.0000
8.0000
4.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	IN	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	IN	CORNER
2.0000	0.0000	IN	ON THE LINE
4.0000	0.0000	IN	ON THE LINE
6.0000	0.0000	IN	ON THE LINE
10.0000	0.0000	IN	CORNER
6.0000	8.0000	IN	CORNER
3.0000	3.0000	IN	IN
4.0000	6.0000	IN	ON THE LINE
7.0000	0.0000	IN	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

PROGRAM HANDEDNESS (INPUT, OUTPUT, PTS, TESTPTS, MESS, RESPONSES);

LABEL 69, 70;

TYPE

SETTINGS = RECORD

ZERO : INTEGER;

POS : INTEGER;

NEG : INTEGER;

NA : INTEGER;

END;

POINT = RECORD

X, Y : REAL;

END;

FLAG = (NEG, POS, ZERO, NOTAPP);

PNTARRAY = ARRAY[1..4] OF POINT;

HFARRAY = ARRAY[1..4] OF FLAG;

FIG = (CONVEX, CONCAVE, TRIANG, TRIRAY, BOWTIE, INVALID, LINE);

VAR

I, J, K : INTEGER;

CASECONDITION : INTEGER;

WRONGSIGN, GC, PC : INTEGER;

INDEX : INTEGER;

POINT1, POINT2 : INTEGER;

DETERMINANTVALUE : REAL;

MOREWORK : BOOLEAN;

RESP, ANSW : CHAR;

SETCHARS : SETTINGS;

SETCHARSGOLD : SETTINGS;

APPOINT : POINT;

P : PNTARRAY;

HFLAG : HFARRAY;

FIGURE, TESTFIG : FIG;

PTS, TESTPTS, MESS : TEXT;

RESPONSES : TEXT;

PROCEDURE DUMP (HF : HFARRAY; PNT : PNTARRAY; SETCHARS : SETTINGS; PT : POINT);

VAR

I : INTEGER;

BEGIN

Writeln('IN DUMP');

Writeln('-----');

Writeln(' POINT # X-COORD Y-COORD HFLAG');

Writeln('-----');

FOR I := 1 TO 4 DO

BEGIN

WRITE(' ', I:2, ' ', P[I].X:6:3, ' ', P[I].Y:6:3, ' ');

Writeln(' ', HFLAG[I]);

END;

Writeln('-----');

Writeln('INPUT X-COORD ==> ', PT.X:6:3, ' INPUT Y-COORD ==> ', PT.Y:6:3);

Writeln('-----');

Writeln('ZERO SETTING ==> ', SETCHARS.ZERO:2);

Writeln('NEGATIVE SETTING ==> ', SETCHARS.NEG:2);

Writeln('POSITIVE SETTING ==> ', SETCHARS.POS:2);

Writeln('NOT APP SETTING ==> ', SETCHARS.NA:2);

END;

PROCEDURE INITIALIZE (VAR PNT : PNTARRAY);

VAR

I : INTEGER;

BEGIN


```

      IF ((SETCHARS.NEG = 2) OR (SETCHARS.POS = 2)) THEN
        SPECIFYFIG := BOWTIE
      ELSE
        IF (SETCHARS.ZERO = 4) THEN
          SPECIFYFIG := LINE
        ELSE
          SPECIFYFIG := INVALID;

```

```

END;

```

```

FUNCTION DETERMINANT(P1,P2,P3 : POINT) : REAL;

```

```

VAR

```

```

  TMP1,TMP2,TMP3 ,TMP4 : REAL;

```

```

BEGIN

```

```

  TMP1 := P1.X * (P2.Y - P3.Y);

```

```

  TMP2 := P2.X * (P1.Y - P3.Y);

```

```

  TMP3 := P3.X * (P1.Y - P2.Y);

```

```

  TMP4 := TMP1 - TMP2 + TMP3;

```

```

  DETERMINANT := TMP4;

```

```

END;

```

```

PROCEDURE LOADPOINT(VAR TEST : POINT);

```

```

BEGIN

```

```

  WRITELN( 'ENTERING LOADPOINT' );

```

```

  READLN( TESTPTS, TEST.X );

```

```

  WRITELN(MESS, 'ENTER X-COORD TO BE TESTED ==> ', TEST.X:6:3);

```

```

  READLN( TESTPTS, TEST.Y );

```

```

  WRITELN(MESS, 'ENTER Y-COORD TO BE TESTED ==> ', TEST.Y:6:3);

```

```

  WRITELN( 'LEAVING LOADPOINT' );

```

```

END;

```

```

PROCEDURE LOADFIGURE(VAR PNTS : PNTARRAY );

```

```

VAR

```

```

  I : INTEGER;

```

```

BEGIN

```

```

  FOR I := 1 TO 4 DO

```

```

    BEGIN

```

```

      READLN( PTS, PNTS[I].X );

```

```

      WRITELN( MESS, 'ENTER X-COORD OF POINT #',I:2,' ==> ', PNTS[I].X:6:3);

```

```

      READLN( PTS, PNTS[I].Y );

```

```

      WRITELN( MESS, 'ENTER Y-COORD OF POINT #',I:2,' ==> ', PNTS[I].Y:6:3);

```

```

    END;

```

```

END;

```

```

PROCEDURE LOADHFLAGARRAY( I : INTEGER; DVAL: REAL; VAR HF : HFARRAY );

```

```

BEGIN

```

```

  IF DVAL < 0.0 THEN

```

```

    HFC[I] := NEG

```

```

  ELSE

```

```

    IF DVAL = 0.0 THEN

```

```

      HFC[I] := ZERO

```

```

    ELSE

```

```

      IF DVAL > 0.0 THEN

```

```

        HFC[I] := POS

```

```

      ELSE

```

```

        HFC[I] := NOTAPP;

```

```

END;

```

```

PROCEDURE CALCULATESETTINGS( J : INTEGER; HF : HFARRAY; VAR SC : SETTINGS);

```

```

BEGIN

```

```

  CASE HFC[J] OF

```

```

    POS      : SC.POS      := SC.POS + 1;

```

```

NEG      : SC.NEG      := SC.NEG + 1;
ZERO     : SC.ZERO     := SC.ZERO + 1;
NOTAPP   : SC.NA       := SC.NA + 1;

```

END;

END;

FUNCTION POINTINCONVEX(SC : SETTINGS) : INTEGER;

VAR

F1, F2, F3 : BOOLEAN;

BEGIN

```

F1 := FALSE;
F2 := FALSE;
F3 := FALSE;
IF ((SC.POS = 4) OR (SC.NEG = 4)) THEN
  F1 := TRUE;
IF ((SC.POS = 3) OR (SC.NEG = 3)) THEN
  F2 := TRUE;
IF ((SC.POS = 2) OR (SC.NEG = 2)) THEN
  F3 := TRUE;

```

CASE SC.ZERO OF

```

0      : IF F1 THEN
          POINTINCONVEX := 1;
        ELSE
          POINTINCONVEX := 4;
1      : IF F2 THEN
          POINTINCONVEX := 2;
        ELSE
          POINTINCONVEX := 4;
2      : IF F3 THEN
          POINTINCONVEX := 3;
        ELSE
          POINTINCONVEX := 4;
OTHERWISE POINTINCONVEX := 4;

```

END;

END;

FUNCTION POINTINOTHER(VAR I : INTEGER; SC : SETTINGS) : BOOLEAN;

VAR

TMP : BOOLEAN;

BEGIN

```

TMP := FALSE;
IF ((SC.POS > 0) AND (SC.NEG > 0)) THEN
  I := 4;
ELSE
  IF SC.ZERO = 2 THEN
    I := 3;
  ELSE
    TMP := TRUE;
POINTINOTHER := TMP;

```

END;

PROCEDURE COMPARE(A, B : SETTINGS; VAR J : INTEGER);

BEGIN

```

IF (((A.ZERO = 0) AND ((A.POS = 3) OR (A.NEG = 3)))
    OR ((A.ZERO = 1) AND (B.ZERO = 1))) THEN
  INDEX := 4;
ELSE
  IF A.ZERO = 2 THEN
    INDEX := 3;
  ELSE

```

```

      IF ((A.ZERO = 1) AND
          ((A.POS = 2) OR (A.NEG = 2))) THEN
        INDEX := 2
      ELSE
        INDEX := 1;
END;

FUNCTION POINTINTRIANGS(SC : SETTINGS) : INTEGER;
BEGIN
  IF ((SC.ZERO = 1) AND ((SC.POS = 2) OR (SC.NEG = 2))) THEN
    POINTINTRIANGS := 2
  ELSE
    IF (SC.ZERO = 2) THEN
      POINTINTRIANGS := 3
    ELSE
      IF (((SC.POS = 3) OR (SC.NEG = 3)) AND (SC.ZERO = 0)) THEN
        POINTINTRIANGS := 1
      ELSE
        POINTINTRIANGS := 4;
      END;
    END;
  END;

PROCEDURE OUTPUTMESSAGE(INDEX : INTEGER; P : POINT);
BEGIN
  CASE INDEX OF
    1 : BEGIN
      Writeln('THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS IN THE FIGURE. ');
      Write(MESS, 'THE POINT [', P.X:6:3, ', ', P.Y:6:3, '];
      Writeln(MESS, ' IS IN THE FIGURE. ');
    END;
    2 : BEGIN
      Write('THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS ON');
      Writeln(' THE BOUNDARY OF THE FIGURE. ');
      Write(MESS, 'THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS ON');
      Writeln(MESS, ' THE BOUNDARY OF THE FIGURE. ');
    END;
    3 : BEGIN
      Write('THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS ON');
      Writeln(' THE VERTEX OF THE FIGURE. ');
      Write(MESS, 'THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS ON');
      Writeln(MESS, ' THE VERTEX OF THE FIGURE. ');
    END;
    4 : BEGIN
      Write('THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS OUT ');
      Writeln(' OF THIS FIGURE. ');
      Write(MESS, 'THE POINT [', P.X:6:3, ', ', P.Y:6:3, '] IS OUT ');
      Writeln(MESS, ' OF THIS FIGURE. ');
    END;
    OTHERWISE Writeln(MESS, 'ERROR');
  END;
END;

PROCEDURE CALCDETERMINANT(VAR HF : HFARRAY; VAR SC : SETTINGS; I : INTEGER;
  P1, P2, P3 : POINT; VAR DVAL : REAL);
BEGIN
  DVAL := DETERMINANT(P1, P2, P3);
  LOADHFLAGARRAY(I, DVAL, HF);
  CALCULATESETTINGS(I, HF, SC);
END;

BEGIN (* MAIN *)
  RESP := 'Y';

```

```

ANSW := 'Y';
QC := 0;
REWRITE(MESS);
RESET(PTS);
RESET(RESPONSES);
REPEAT
  QC := QC + 1;
  RESET(TESTPTS);
  PC := 0;
  INIT(ALIZE(P));
  LOADFIGURE(P);
  REPEAT
    PC := PC + 1;
    WRITELN('-----'),
    WRITELN(MESS);
    WRITELN(MESS, 'CHECKING QUADRILATERAL # ', QC:2);
    WRITELN('CHECKING QUADRILATERAL # ', QC:2);
    WRITELN('CHECKING POINT # ', PC:2);
    WRITELN('-----');
    INIT(HFLAG, SETCHARS);
    LOADPOINT(APOINT);
    (* DUMP(HFLAG, P, SETCHARS, APOINT); *)
    FOR I := 1 TO 4 DO
      BEGIN
        J := I + 1;
        IF J = 5 THEN
          J := 1;
        K := J + 1;
        IF K = 5 THEN
          K := 1;
        CALCDETERMINANT(HFLAG, SETCHARS, I, P[I], P[J], P[K],
          DETERMINANTVALUE);
      END;
    (*DUMP(HFLAG, P, SETCHARS, APOINT); *)
    WRONGSIGN := 55;
    TESTFIG := SPECIFYF(G(HFLAG, SETCHARS, WRONGSIGN),
    WRONGSIGN := WRONGSIGN + 1;
    POINT1 := WRONGSIGN;
    IF WRONGSIGN >= 3 THEN
      POINT2 := WRONGSIGN - 2
    ELSE
      POINT2 := WRONGSIGN + 2;
    CASE TESTFIG OF
      CONVEX : BEGIN
        WRITELN('CONVEX CASE');
        FOR I := 1 TO 4 DO
          BEGIN
            J := I + 1;
            IF J = 5 THEN
              J := 1;
            CALCDETERMINANT(HFLAG, SETCHARS, I, P[I], P[J],
              APOINT, DETERMINANTVALUE);
          END;
        OUTPUTMESSAGE(POINTINCONVEX(SETCHARS), APOINT);
      END;
      CONCAVE : BEGIN
        WRITELN('CONCAVE CASE');
        IF WRONGSIGN = 5 THEN
          WRONGSIGN := 1;
        DUMP(HFLAG, P, SETCHARS, APOINT);
      END;
    END;
  END;

```

```

INIT(HFLAG, SETCHARS);
FOR I := 1 TO 3 DO
  BEGIN
    WRONGSIGN := WRONGSIGN + 1;
    IF WRONGSIGN = 5 THEN
      WRONGSIGN := 1;
    K := (WRONGSIGN MOD 4) + 1;
    IF I = 3 THEN
      K := (K MOD 4) + 1;
    (*DUMP(HFLAG, P, SETCHARS, APOINT), *)
    CALCDETERMINANT(HFLAG, SETCHARS, I,
                    P[WRONGSIGN], P[K], APOINT,
                    DETERMINANTVALUE);
  END;
DUMP(HFLAG, P, SETCHARS, APOINT);
MOREWORK := POINTINOTHER(INDEX, SETCHARS);
WRITE('BEFORE MOREWORK. MOREWORK ==> ', MOREWORK);
WRITELN(' INDEX ==> ', INDEX:2);
IF NOT MOREWORK THEN
  OUTPUTMESSAGE(INDEX, APOINT)
ELSE
  BEGIN
    WRONGSIGN := POINTI;
    IF WRONGSIGN = 5 THEN
      WRONGSIGN := 1;
    SETCHARSOLD := SETCHARS;
    DUMP(HFLAG, P, SETCHARS, APOINT);
    INIT(HFLAG, SETCHARS);
    K := WRONGSIGN;
    FOR I := 1 TO 3 DO
      BEGIN
        IF I = 1 THEN
          WRONGSIGN := WRONGSIGN - 1;
        IF WRONGSIGN = 0 THEN
          WRONGSIGN := 4;
        K := (WRONGSIGN MOD 4) + 1;
        IF I = 3 THEN
          K := (K MOD 4) + 1;
        (*DUMP(HFLAG, P, SETCHARS, APOINT), *)
        CALCDETERMINANT(HFLAG, SETCHARS, I,
                        P[WRONGSIGN], P[K], APOINT,
                        DETERMINANTVALUE);
        WRONGSIGN := WRONGSIGN + 1;
        IF WRONGSIGN = 5 THEN
          WRONGSIGN := 1;
      END;
    DUMP(HFLAG, P, SETCHARS, APOINT);
    MOREWORK := POINTINOTHER(INDEX, SETCHARS);
    COMPARE(SETCHARS, SETCHARSOLD, INDEX);
    OUTPUTMESSAGE(INDEX, APOINT);
  END;
END;

TRIRAY.
TRIANG BEGIN
  WRITELN('TRI CASE');
  INIT(HFLAG, SETCHARS);
  FOR I := 1 TO 3 DO
    BEGIN
      WRONGSIGN := WRONGSIGN + 1;
      IF WRONGSIGN = 5 THEN

```

```

        WRONGSIGN := 1;
        K := (WRONGSIGN MOD 4) + 1;
        IF I = 3 THEN
            K := (K MOD 4) + 1;
            (*DUMP(HFLAG, P, SETCHARS, APOINT); *)
            CALCDETERMINANT(HFLAG, SETCHARS, I,
                            P[WRONGSIGN], P[K], APOINT,
                            DETERMINANTVALUE);
        END;
        DUMP(HFLAG, P, SETCHARS, APOINT);
        OUTPUTMESSAGE(POINTINTRIANGS(SETCHARS), APOINT);
    END;
INVALID,
BOWTIE,
LINE      BEGIN
            WRITELN('INVALID CASE');
            GOTO 69;
        END;
    END;
    READLN(RESPONSES, ANSW);
    WRITELN('ANOTHER POINT?? Y/N', ANSW);
    GOTO 70;
69:    ANSW := 'N';
    WRITELN('THIS FIGURE "', TESTFIG, '" IS AN ILLEGAL FIGURE. ');
    WRITELN('PLEASE INPUT ANOTHER FIGURE OR TERMINATE ');
70:    UNTIL ((ANSW = 'N') OR (ANSW = 'n'));
    CLOSE(TESTPTS);
    READLN(RESPONSES, RESP);
    WRITELN('ANOTHER QUAD?? Y/N', RESP);
    UNTIL ((RESP = 'N') OR (RESP = 'n'));
    WRITELN('THAT'S ALL FOLKS!!!');
    CLOSE(PTS);
    CLOSE(RESPONSES);
    CLOSE(MESS);
END.

```

QUADRILATERAL	X-COORDINATE	Y-COORDINATE
C-1	0.0000	0.0000
	2.0000	0.0000
	4.0000	0.0000
	6.0000	0.0000

CHECKING THE INPUT (X, Y) POINTS
 LINE IS ILLEGAL FIGURE, NO POINT TESTED

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000		OUT
4.0000	1.0000		OUT
2.0000	3.0000		OUT
6.0000	3.0000		OUT
8.0000	3.0000		OUT
5.0000	0.0000		ON THE LINE
2.0000	7.0000		OUT
8.0000	7.0000		OUT
6.0000	2.0000		OUT
-3.0000	4.0000		OUT
-2.0000	7.0000		OUT
0.0000	0.0000		CORNER
2.0000	0.0000		CORNER
4.0000	0.0000		CORNER
6.0000	0.0000		CORNER
10.0000	0.0000		OUT
6.0000	8.0000		OUT
3.0000	3.0000		OUT
4.0000	6.0000		OUT
7.0000	0.0000		OUT
6.0000	4.0000		OUT
13.0000	0.0000		OUT
6.0000	7.0000		OUT
12.0000	0.0000		OUT
0.0000	5.0000		OUT
7.0000	5.0000		OUT

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-2

0.0000
10.0000
6.0000
3.0000

0.0000
0.0000
8.0000
3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	CORNER	CORNER
4.0000	6.0000	OUT	OUT
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

 QUADRILATERAL

 X-COORDINATE

 Y-COORDINATE

C-3

0.0000
 4.0000
 7.0000
 10.0000

0.0000
 6.0000
 0.0000
 0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	ON THE LINE	ON THE LINE
6.0000	3.0000	OUT	OUT
8.0000	3.0000	OUT	OUT
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	ON THE LINE	ON THE LINE
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	OUT*	CORNER
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	CORNER	CORNER
7.0000	0.0000	CORNER	CORNER
6.0000	4.0000	OUT	OUT
13.0000	0.0000	OUT	OUT
6.0000	7.0000	OUT	OUT
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

*THIS CORNER IS DEGENERATED TO ZERO ANGLE. THIS ANSWER DEPENDS ON DEFINITION.

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-4

0.0000
6.0000
13.0000
6.0000

0.0000
4.0000
0.0000
0.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	OUT	OUT
4.0000	1.0000	IN	IN
2.0000	3.0000	OUT	OUT
6.0000	3.0000	IN	IN
8.0000	3.0000	OUT	OUT
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	CN THE LINE*	CORNER
10.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	8.0000	OUT	OUT
3.0000	3.0000	OUT	OUT
4.0000	6.0000	OUT	OUT
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	CORNER	CORNER
13.0000	0.0000	CORNER	CORNER
6.0000	7.0000	OUT	OUT
12.0000	0.0000	ON THE LINE	ON THE LINE
0.0000	5.0000	OUT	OUT
7.0000	5.0000	OUT	OUT

*THIS CORNER IS DEGENERATE AND MAY BE CONSIDERED A LINE. THIS ANSWER DEPENDS ON DEFINITION.

QUADRILATERAL	X-COORDINATE	Y-COORDINATE
C-5	0.0000	0.0000
	6.0000	7.0000
	12.0000	0.0000
	6.0000	3.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	OUT	OUT
2.0000	3.0000	OUT	OUT
6.0000	3.0000	CORNER	CORNER
8.0000	3.0000	IN	IN
5.0000	0.0000	OUT	OUT
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	OUT	OUT
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	OUT	OUT
4.0000	0.0000	OUT	OUT
6.0000	0.0000	OUT	OUT
10.0000	0.0000	OUT	OUT
6.0000	8.0000	OUT	OUT
3.0000	3.0000	IN	IN
4.0000	6.0000	OUT	OUT
7.0000	0.0000	OUT	OUT
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	CORNER	CORNER
12.0000	0.0000	CORNER	CORNER
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

 QUADRILATERAL

 X-COORDINATE

 Y-COORDINATE

C-6

0.0000
 0.0000
 7.0000
 7.0000

0.0000
 5.0000
 0.0000
 5.0000

CHECKING THE INPUT (X, Y) POINTS
 BOWTIE IS AN ILLEGAL FIGURE. NO POINTS TESTED

----- INPUT X -----	----- INPUT Y -----	----- GENERATED RESULT -----	----- ACTUAL RESULT -----
4.0000	4.0000		OUT
4.0000	1.0000		OUT
2.0000	3.0000		IN
6.0000	3.0000		IN
8.0000	3.0000		OUT
5.0000	0.0000		OUT
2.0000	7.0000		OUT
8.0000	7.0000		OUT
6.0000	2.0000		IN
-3.0000	4.0000		OUT
-2.0000	7.0000		OUT
0.0000	0.0000		CORNER
2.0000	0.0000		OUT
4.0000	0.0000		OUT
6.0000	0.0000		OUT
10.0000	0.0000		OUT
6.0000	8.0000		OUT
3.0000	3.0000		OUT
4.0000	6.0000		OUT
7.0000	0.0000		CORNER
6.0000	4.0000		IN
13.0000	0.0000		OUT
6.0000	7.0000		OUT
12.0000	0.0000		OUT
0.0000	5.0000		CORNER
7.0000	5.0000		CORNER

QUADRILATERAL

X-COORDINATE

Y-COORDINATE

C-7

0.0000
10.0000
6.0000
2.0000

0.0000
0.0000
8.0000
4.0000

CHECKING THE INPUT (X, Y) POINTS

INPUT X	INPUT Y	GENERATED RESULT	ACTUAL RESULT
4.0000	4.0000	IN	IN
4.0000	1.0000	IN	IN
2.0000	3.0000	IN	IN
6.0000	3.0000	IN	IN
8.0000	3.0000	IN	IN
5.0000	0.0000	ON THE LINE	ON THE LINE
2.0000	7.0000	OUT	OUT
8.0000	7.0000	OUT	OUT
6.0000	2.0000	IN	IN
-3.0000	4.0000	OUT	OUT
-2.0000	7.0000	OUT	OUT
0.0000	0.0000	CORNER	CORNER
2.0000	0.0000	ON THE LINE	ON THE LINE
4.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	0.0000	ON THE LINE	ON THE LINE
10.0000	0.0000	CORNER	CORNER
6.0000	8.0000	CORNER	CORNER
3.0000	3.0000	IN	IN
4.0000	6.0000	ON THE LINE	ON THE LINE
7.0000	0.0000	ON THE LINE	ON THE LINE
6.0000	4.0000	IN	IN
13.0000	0.0000	OUT	OUT
6.0000	7.0000	IN	IN
12.0000	0.0000	OUT	OUT
0.0000	5.0000	OUT	OUT
7.0000	5.0000	IN	IN

APPENDIX D
SOME MATHEMATICAL PROOFS

Proposition: The point P lies within the triangle ABC if, and only if, the extension of the line segments AP , BP , and CP intersect BC , CA , and AB , respectively.

Proof: First note that for any two points X and Y , $XY \setminus X$, and $XY \setminus Y$ are connected sets (Moore, Theorem 98). Suppose P lies within ABC , then so do AP , BP , and CP (by Theorem 33, Moore applied to $AP \setminus A$, etc). Since AP divides the angle CAB , its extension intersects BC , and similarly for BP and CP .

Suppose one of the line extensions does not intersect the appropriate side. Without loss of generality, suppose the line passing through A and P does not intersect BC . Since it does not pass through C , PAC are not colinear, and similarly PAB are not colinear. Thus it intersects ABC at the endpoint A only. Thus the connected set $AP \setminus A$ does not intersect ABC , and hence falls entirely outside ABC (again by Moore, Theorem 33). Therefore P is not enclosed by ABC .

Rotation of coordinates:

See Figure D-1 (a)

$$x' = (x + y \tan \theta) \cos \theta$$

$$= x \cos \theta + y \sin \theta$$

$$y' = y / \cos \theta - (x + y \tan \theta) \sin \theta$$

$$= -x \sin \theta + y (1 / \cos \theta - \sin^2 \theta / \cos \theta)$$

$$= -x \sin \theta + y \cos \theta$$

$$x' = (-x \cos \theta + y \sin \theta)$$

$$y' = (-x \sin \theta + y \cos \theta)$$

ANOTHER PROOF

Given; three ordered points $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ and $P_3(x_3, y_3)$.

$$\det \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \equiv h.$$

Prove:

$h = 0$ if P_3 is on the extended line $P_1 P_2$.

$h > 0$ if P_3 is on the left side of the directed line segment.

$h < 0$ if P_3 is on the right of the directed line segment.

Proof:

Translate origin to P_1 (see Figure D-1 (b))

$$X_i = (x_i - x_1) \text{ where } (X_1, Y_1) = (0, 0).$$

$$Y_i = (y_i - y_1).$$

Rotate the axis so that the Y_2 lies on the axis. (see Figure D-1 (c))

$$\sin \theta = -Y_2 / \sqrt{X_2^2 + Y_2^2}$$

$$\cos \theta = X_2 / \sqrt{X_2^2 + Y_2^2}.$$

$$Y_3' = -X_3 \sin \theta + Y_3 \cos \theta$$

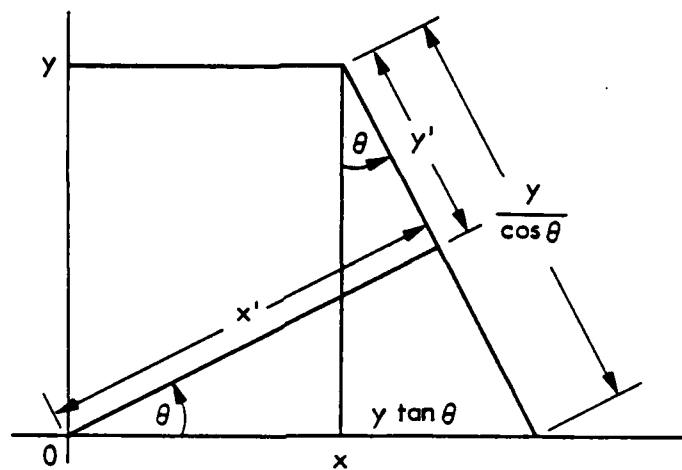
Turn in transition along P_1, P_2, P_3 , is left if Y_3' is > 0 , along a line,

there is no turn if $Y'_3 = 0$, and turn is right if $Y'_3 < 0$. If this function is called sign then:

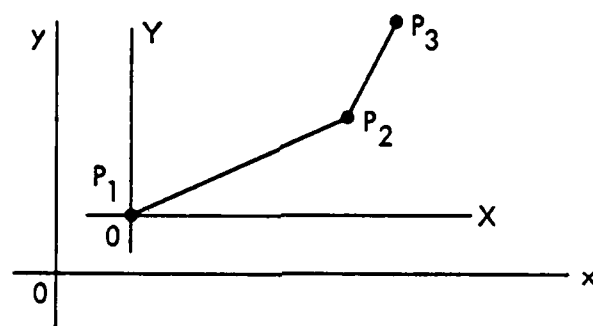
$$\begin{aligned}
 Y'_3 &= -X_3 \sin\theta + Y_3 \cos\theta \\
 &= -X_3 X_2 / \sqrt{X_2^2 + Y_2^2} + Y_3 X_2 / \sqrt{X_2^2 + Y_2^2} \\
 &= [-X_3 Y_2 + X_2 Y_3] / \sqrt{X_2^2 + Y_2^2} \\
 &= [-(x_3 - x_1)(y_2 - y_1) + (x_2 - x_1)(y_3 - y_1)] / \sqrt{X_2^2 + Y_2^2} \\
 &= [x_3 y_1 + x_1 y_2 + x_2 y_3 + x_1 y_1 - x_3 y_2 - x_1 y_1 - x_2 y_1 - x_1 y_3] / \sqrt{X_2^2 + Y_2^2}.
 \end{aligned}$$

$$\text{sign} \left(\det \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \right) \equiv \text{sign } h \quad \text{q.e.d.}$$

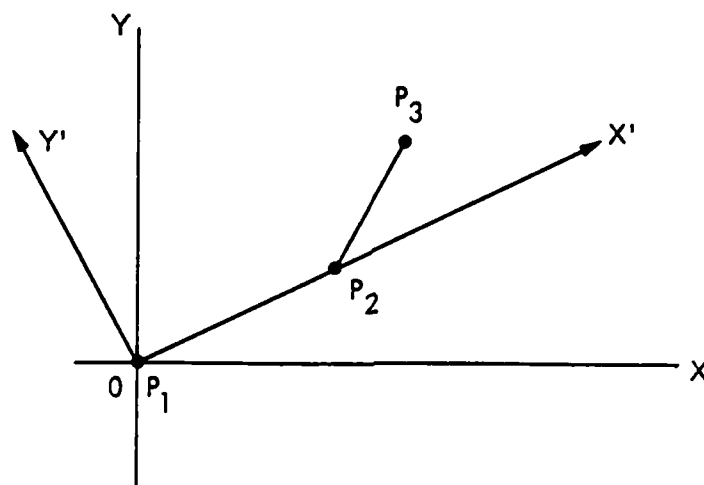
since the square root is always positive.



(a) ROTATION OF COORDINATES IN GENERAL



(b) TRANSLATION OF COORDINATES



(c) RELATION BETWEEN X , Y AND X' , Y' COORDINATES

Figure D-1. Proof of Handedness Theorem

END