Pergamon

*Computers & Geosciences* Vol. 23, No. 1, pp. 109–118, 1997

PII: S0098-3004(96)00071-4
0098-3004/97 $17.00 + 0.00

# ON THE COMPLEXITY OF POINT-IN-POLYGON ALGORITHMS

CHONG-WEI HUANG and TIAN-YUAN SHIH
Department of Civil Engineering, National Chiao-Tung University, Hsin-Chu, Taiwan

*(Received 4 March 1996; revised 4 July 1996)*

**Abstract**—Point-in-polygon is one of the fundamental operations of Geographic Information Systems. A number of algorithms can be applied. Different algorithms lead to different running efficiencies. In the study, the complexities of eight point-in-polygon algorithms were analyzed. General and specific examples are studied. In the general example, an unlimited number of nodes are assumed; whereas in the second example, eight nodes are specified. For convex polygons, the sum of area method, the sign of offset method, and the orientation method is well suited for a single point query. For possibly concave polygons, the ray intersection method and the swath method should be selected. For eight node polygons, the ray intersection method with bounding rectangles is faster. © 1997 Elsevier Science Ltd. All rights reserved

*Key Words*: Point-in-polygon, Complexity, Ray intersection, Sum of angles method, Swath method, Sign of offset method.

## INTRODUCTION

In this article, the "point-in-polygon" problem is defined as: "With a given polygon P and an arbitrary point q (Fig. 1), determine whether point q is enclosed by the edges of the polygon". This question does not appear too difficult to solve. However, for circumstances as in Figure 2, in the event that the polygon is concave and composed of many vertices, an efficient and reliable algorithm is necessary.

An algorithm is understood as a series of procedures implemented to solve a particular mathematical problem with a computer. Manber (1989) stated that an efficient algorithm is more valuable than a fast computer. In Geographical Information Systems (GIS), the node–arc–polygon model functions as one of the fundamental representations to describe accurately the topological relation between spatial objects. Determining whether a certain position is located in a particular district or not is a point-in-polygon problem. Point-in-polygon is generally a subject of geometric searching; in addition, geometric searching is one of the six major topics of Computational Geometry (Preparata and Shamos, 1985).

The efficiency of an algorithm can be evaluated by the following four cost measures (Preparata and Shamos, 1985):

(1) query time: the time required to respond to a single query;
(2) storage: the size of memory required for the data structure;
(3) pre-processing time: the time required to arrange the data for searching; and
(4) update time: the time required to renew the data structure.

In this article, the complexity of algorithms and its representation is described first. Next, eight algorithms for point-in-polygon are described, and their complexities are analyzed. The applications of each algorithm are discussed in the final conclusion.

## COMPLEXITY

Assuming that a procedure takes 1 cpu sec to process and then repeating this procedure ten times, requires 10 sec. Restated, the required cpu time is linearly proportional to the number of procedures. Correspondingly, the order of growth for the required computer time, $T$, with respect to the number of repetitions, $N$, can be modeled with a simple relation, $T = \mathbf{f}(N)$, e.g. "$c \cdot \log N$". When $N$ is reasonably large, $c$ can be neglected. In most instances, algorithm complexity is evaluated with the degree of the complexity function. A notation is then defined (Cormen, Leiserson, and Rivest, 1991).

$\mathbf{O}(\mathbf{f}(N)) = \{\mathbf{g}(N)$: $\exists$ positive constants $c$ and $n_0$, such that $0 \leq \mathbf{g}(N) \leq c \cdot \mathbf{f}(N)$, $\forall\ N \geq n_0\}$ Besides **O**-notation, other indices are used to describe/measure the complexity, such as $\Omega(\mathbf{f}(N))$. In this article, **O**-notation is used (Fig. 3).

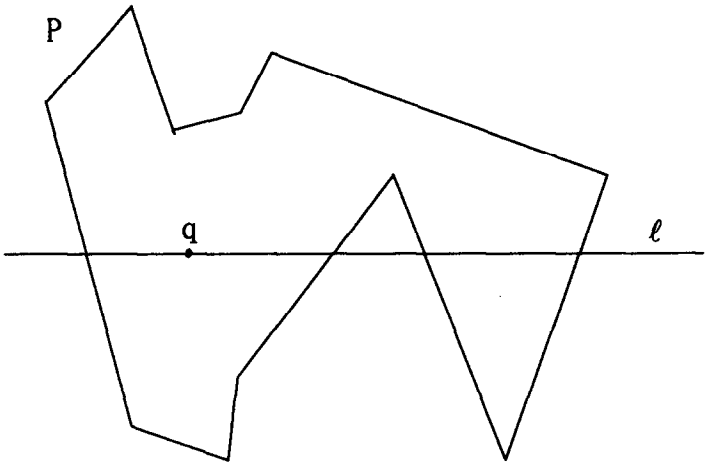


Figure 1. Point-in-polygon, example.

## ALGORITHMS FOR POINT-IN-POLYGON

A number of algorithms can be applied to the point-in-polygon problem.

Besides the kernel procedures, the complexity differs with the pre-processing method. The conventional procedure for reducing the search entails using the bounding rectangle of polygon P to examine point q. If q is outside this bounding rectangle, q will not be inside P, and the problem is solved. Assuming that the probability of point q located inside is 50%, this filtering can reduce the expected execution time by half. Because the coordinates defining the bounding rectangle of a polygon are regular items stored in a vector-based GIS, this filtering takes no more than four Boolean operations. This pre-processing can be applied for any of the point-in-polygon algorithms described in this article.

In the next section, the kernel procedures of each algorithm are described together with the complexity analysis with an asymptotically large $N$. The analysis with a limited $N$ follows.

## TIME COMPLEXITY WITH ASYMPTOTICALLY LARGE N

### *Grid method*

In this algorithm, the polygon P is represented as a group of grid cells. To determine whether a given point q is inside the polygon, the coordinates of point q are compared with the coordinates of each grid cell of P. Figure 4 provides an illustrative

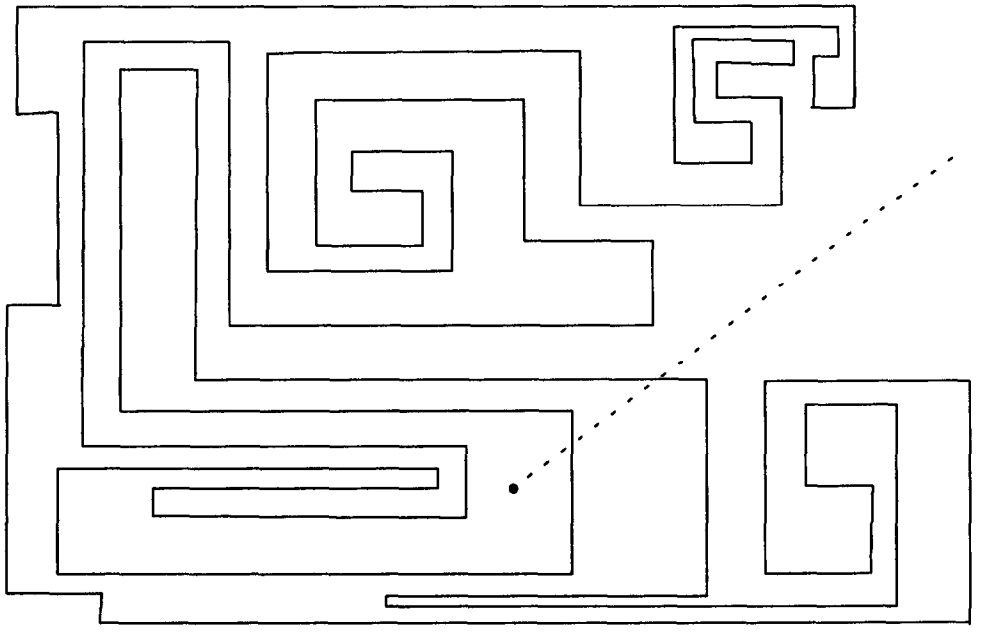

Figure 2. Point-in-polygon, a complex example, from Manber (1989).

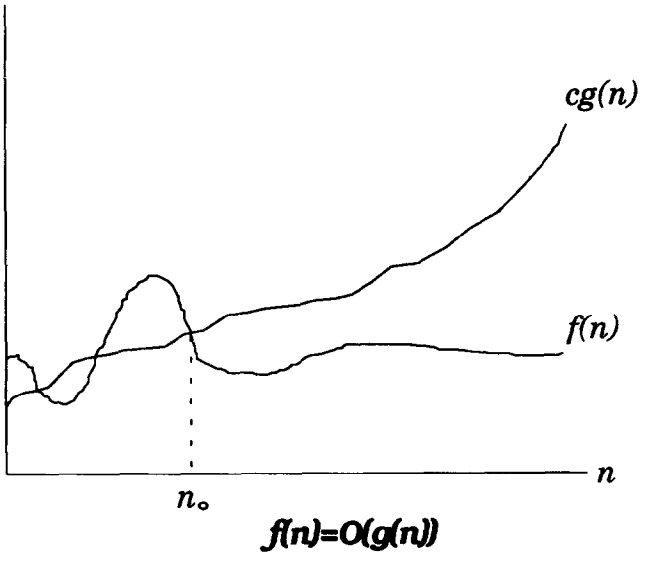


Figure 3. Illustration for O-notation (Cormen, Leiserson, and Rivest, 1991).

example. This algorithm is the procedure NORK stated in Nordbeck and Rystedt (1967).

```
Algorithm Grid_Method(G,P,q):
Input: G ( a grid mesh with d = (x0, y0) being its left-lowest point, and h being its
       spacing ), P ( a polygon area composed of N grid cells p_i in G, for each cell,
       (Xid, Yid) are its x y directional identifiers), and q = (x, y)( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
  i:=⌈(x - x0)/h⌉;   { x-directional identifier for q in G }
  j:=⌈(y - y0)/h⌉;   { y-directional identifier for q in G }
  Inside :=false ;
  for all cells p_k of the polygon do
    if i=Xid[k] and j=Yid[k] then
      Inside :=true ;
      return
end
```

Determining whether the point is within a cell requires two steps. Because point q is compared with each grid cell, the time complexity is $O(N)$. The space complexity is $O(N)$ also because each grid cell requires a unit to store. This algorithm is well adapted to raster-based situations. For a vector-based situation, a vector-to-raster procedure must be performed in advance. Because point-in-polygon is actually applied in the polygon rasterization process, this method is not practical for vector-based situations.

*Ray intersection method*

Draw a line passing point q, and count the number of intersections made by this line and the edges of polygon. If the number of intersections on either side of point q is an odd number, point q is inside polygon P. Frequently, a line parallel to one of the coordinate axes is used, such as that shown in Figure 1 (Nordbeck and Rystedt, 1967; Manber, 1989).

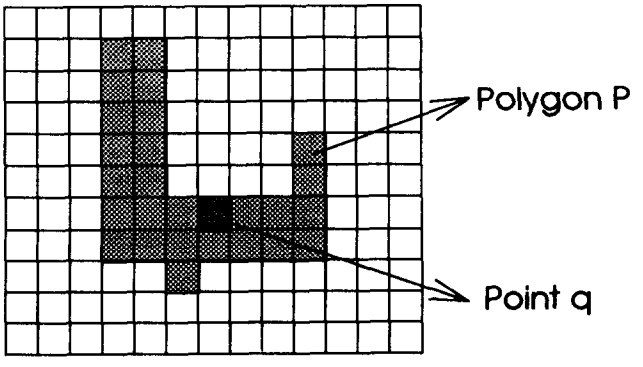


Figure 4. Grid method.

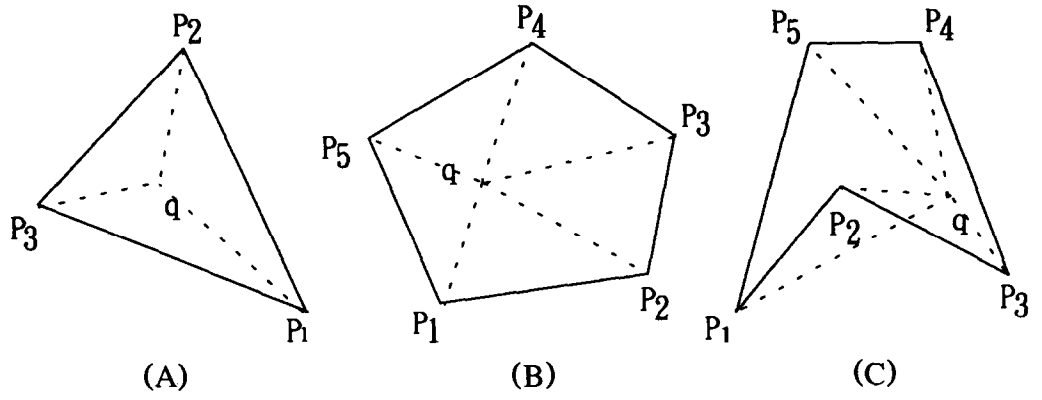


Figure 5. Sum of angles method.

```
Algorithm Ray_Intersection_Method(P,q);
Input: P ( a simple polygon with vertices p1,p2,....,pn ,and edges e1,e2,....,en ),and
       q = (x0, y0)( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   count :=0;
   for all edges ei of the polygon do
      if the line x = x0 intersects ei then
      { The intersection is assumed here to not be at a vertex nor is the line x = x0
        overlapping with ei }
        Let yi be the y coordinates of the intersection between line x = x0 and ei
        if yi < y0 then {the intersection is below q }
           increment count;
   if count is odd then Inside :=true
   else Inside :=false
end
```

For each polygon's edges, an intersection analysis is performed. The time complexity for each intersection analysis is $O(1)$. Because the number of edges is the same as the number of nodes, the time complexity of point-in-polygon determination is $\mathbf{O}(N)$.

This algorithm can be applied for both convex- and concave-shaped polygons. It is well suited for vector-based applications. For implementation, some special situations must be considered. These include situations such as a point q lying on one of the edges, or the ray intersecting the polygon at a node.

*Sum of angles method*

As shown in Figure 5, the angles formed by point q are computed as the vertex and the node-pairs as the sides in sequence. If the sum of these angles is 360°, point q is inside the polygon (Nordbeck and Rystedt, 1967). The angles could be positive, negative, or zero. For instance, case (C) in Figure 5 has a negative angle. This algorithm is applicable to both convex and concave polygons. The time complexity is $\mathbf{O}(N)$. However, the primary limitation of the algorithm is that it is slow because the time required to compute an angle is always greater than the time required for computing the determinant of a $3 \times 3$ matrix (Preparata and Shamos, 1985). Several algorithms are available for computing an angle. However, the efficiency of angle computation contributes to the constant $c$ here. The $\mathbf{O}(N)$ time complexity remains unaffected. The other disadvantage is that this algorithm is affected significantly by the rounding errors (Nordbeck and Rystedt, 1967).

```
Algorithm Sum_of_Angles(P,q);
Input: P ( a simple polygon with vertices p1,p2,....,pn ,and edges e1,e2,....,en ),and
       q ( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   sum_angle:=0;
   for all edges ei of the polygon do
      calculate ∠ pi q pi+1;
      if pi − q − pi+1 is left-turn then assign negative sign to ∠ pi q pi+1;
      accumulate ∠ pi q pi+1 to sum_angle;
   if sum_angle= 360° then Inside:=true
   else Inside:=false
end
```

*Swath method (Salomon, 1978)*

This algorithm uses the ray intersection algorithm as its kernel. However, several pre-processing procedures are added.

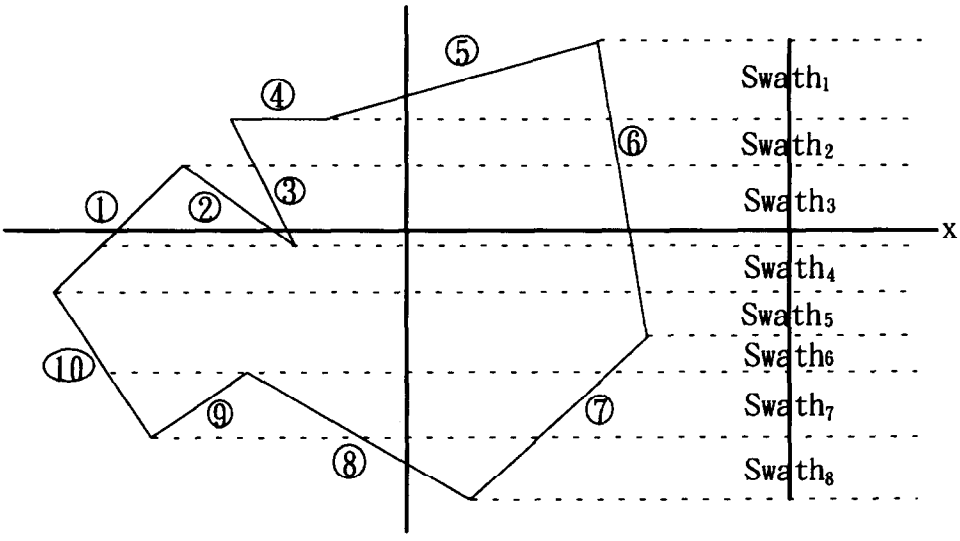


Figure 6. Swath method.

***Algorithm Swath_Method(P,q);***
**Input:** *P* ( a simple polygon with vertices $p_1, p_2, \ldots, p_n$ ,and edges $e_1, e_2, \ldots, e_n$ ),and
*q* ( a point ).
**Output:** *Inside* ( a Boolean variable that is set to true if *q* is inside *P* and false otherwise ).
***begin***
*{ preprocessing }*
*Normalize Y-coordinates of vertices into integer type;*
*Sort Y-coordinates of vertices in P in a descending order,*
*each consecutive Y-value decides a swath;*
*Build a binary tree for Y intervals;*
***for*** *all edges $e_i$ of the polygon* ***do***
*determine how many swaths $e_i$ belongs to;*
*Decides q in which swath from the binary tree;*
*Determine q if it is inside of the polygon from the corresponding swath*
*using* ***Algorithm Ray_Intersection_Method***
***end***

This algorithm divides the polygon into swaths according to the $y$-coordinates of its nodes (Fig. 6). The time complexity of this dividing process is $\mathbf{O}(N\log N)$. There are $N-1$ swaths to the maximum, that is an $\mathbf{O}(N)$, and each swath has at least two edges and $N-1$ edges to the maximum, that is another $\mathbf{O}(N)$. The total space complexity is then $\mathbf{O}(N) \cdot \mathbf{O}(N) = \mathbf{O}(N^2)$.

Regarding the time complexity, three major stages are involved. The first stage records the number of edges in each swath. It can be achieved by looping over all edges in $\mathbf{O}(N)$ time. The second stage finds the swath in which point q is located. This search can be performed in $\mathbf{O}(\log N)$ by a balanced binary tree. The final stage counts the number of intersections of the ray and edges in the swath. The time complexity of this stage is $\mathbf{O}(N)$.

When the number of edges in each swath are limited, the space complexity can be reduced to $\mathbf{O}(N)$, while the time required for the intersection counting can be reduced to $\mathbf{O}(1)$.

*Sign of offset method*

As shown in Figure 7, the nodes of polygon P are arranged in a counter-clockwise manner. If the distance between point q and the edge $p_ip_{i+1}$ has the same sign as the distance between the vertex $p_{i+2}$ and the edge $p_ip_{i+1}$, for all edges of P, point q is inside the polygon (Nordbeck and Rystedt, 1967). This is equivalent to evaluating whether point q is

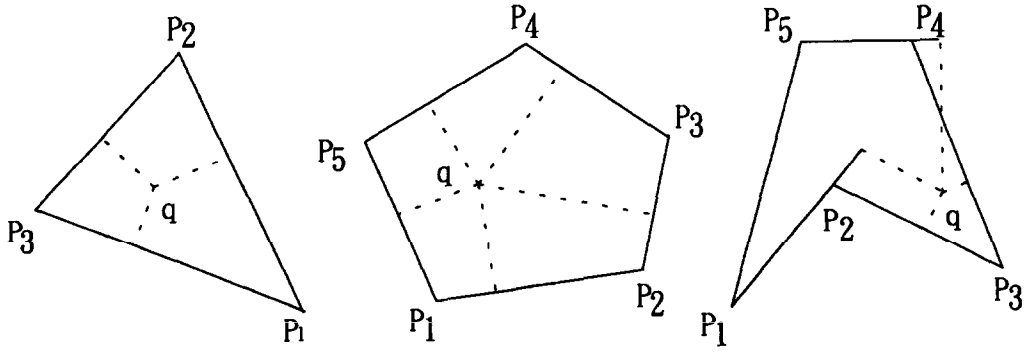


Figure 7. Sign of offset method.

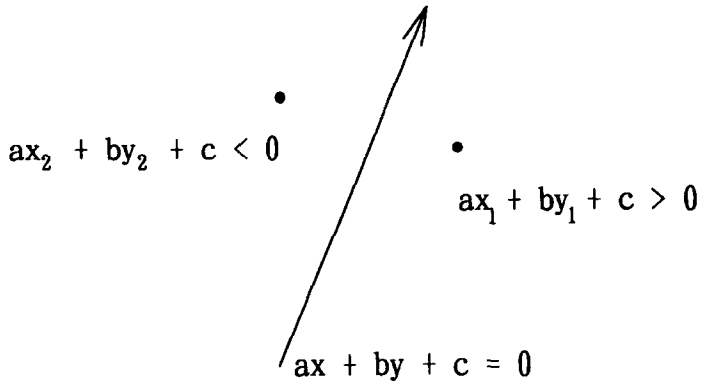


Figure 8. Sign and sides.

at the same side of any two consecutive edges, $e_i$, $e_{i+1}$. If yes, point q is inside polygon P (Fig. 7). This algorithm is not valid for concave polygons. The time and space complexities of this algorithm are $O(N)$.

Regarding the sign evaluation, a sub-algorithm, as illustrated in Figure 8, could be applied. However, as far as the time complexity is concerned, area computation with coordinates is a better scheme. For a triangle formed by ($p_i$, $p_{i+1}$, q) its area can be represented as a third-order determinant (Preparata and Shamos, 1985)

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_q & y_q & 1 \end{vmatrix} = (x_2 - x_1)(y_q - y_1) - (x_q - x_1)(y_2 - y_1)$$

From the sign of this determinant, it can be determined whether q is located at the left or right side. Sloan (1987) revised this algorithm into two multiplications, four subtractions, and one Boolean operation. This operation is considered as one of the primitive operations in Computational Geometry (Guibas and Stolfi, 1985). The time complexity is **O(1)**. Evaluating the intersection of two lines, as applied in the ray intersection algorithm, can be performed with four triangle area computations (Sedgewick, 1988). That is, eight multiplications, sixteen subtractions, and five Boolean operations are included in Sloan's (1987) algorithm. Taylor (1994) reported an algorithm modified from Saalfield (1987), which reduced the number of operations to seven multiplications, nine subtractions and three Boolean operations. If point q is inside polygon P, all edges must be examined. If q is outside P, then on average, only half of the edges must be processed. In either instance, the time complexity is **O**($N$).

```
Algorithm Sign_of_Offset(P, q);
Input: P ( a convex polygon with vertices p1, p2, ..., pn, and edges e1, e2, ..., en ), and
       q ( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   for each consecutive pair of edges ei and ei+1 of the polygon do
      if (q - pi - pi+1 is left-turn ) and (q - pi+1 - pi+2 is left-turn) then Inside := true
      else Inside := false;
           return
end
```

*Sumz of area method*

This algorithm states that if point q is inside polygon P, then connecting q with each node of P will subdivide P into a number of triangles, and the sum of the area of these triangles is equivalent to the area of polygon P (Nordbeck and Rystedt, 1967). The geometrical figure is the same as in Figure 5; however, instead of the angles, the area is computed.

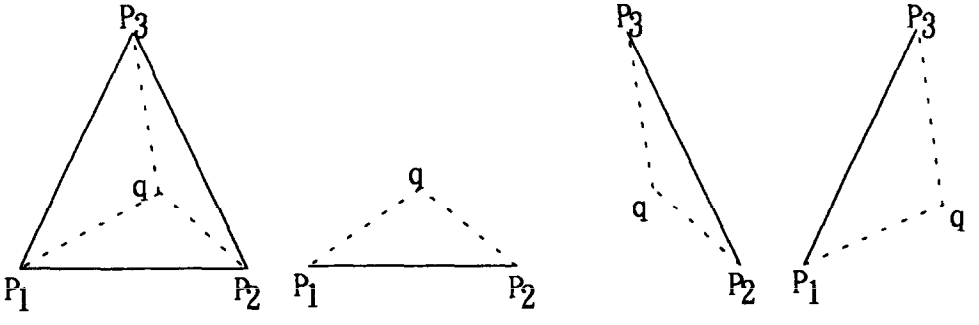


Figure 9. Orientation method.

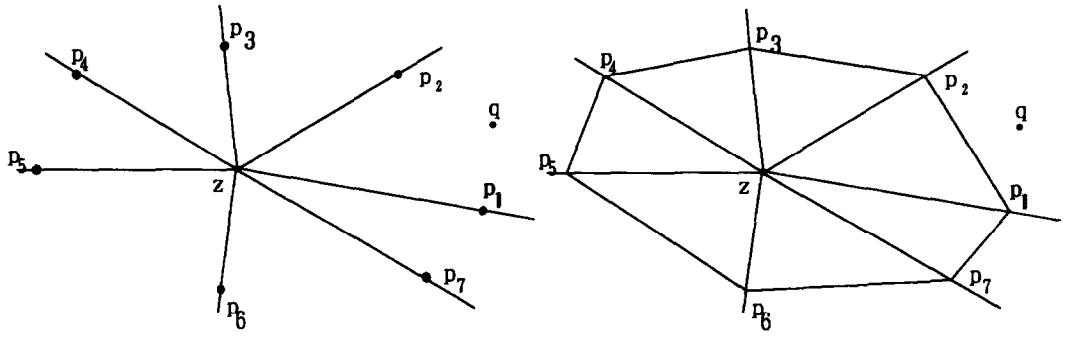


Figure 10. Wedge method.

```
Algorithm Sum_of_Area(P,q);
Input: P ( a simple polygon with vertices p1,p2,....,pn ,and edges e1,e2,....,en ),and
       q ( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   Calculate the total area T of the polygon;
   sum_area:=0;
   for all edges ei of the polygon do
      accumulate the area of triangle ( Pi Pi+1 q ) to sum_area;;
   if sum_area= T then Inside:=true
   else Inside:=false
end
```

Because the **for**-loop is executed $N$ times, the time complexity is $\mathbf{O}(N)$. This algorithm is applicable only to convex polygons.

*The orientation method*

As shown in Figure 9, if the nodes of a convex polygon are arranged in a counter-clockwise direction, then for a point q inside polygon P, q must be located at the left side of each edge (Nordbeck and Rystedt, 1967).

```
Algorithm Orientation_Method(P, q);
Input: P ( a simple polygon with vertices p1,p2,....,pn ,and edges e1,e2,....,en ),and
       q ( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   Inside:=true;
   for all edges ei of the polygon do
      if Pi - Pi+1 - q is right-turn then
         Inside:=false
         return
end
```

Comparing this algorithm with the sum of area method reveals that both of them perform area computations, and both complexities are of $\mathbf{O}(N)$. The advantage of this method is that no value for the area has to be known—only the sign of the area is sought. Therefore, no accumulation is necessary. Once a sign change is detected, the evaluation ends. This indicates that this method is better than the sum of area method. Compared with the sign of offset method, most procedures are the same. However, the sign of offset method takes two Boolean operations for each edge, while the orientation method takes one.

*Wedge method*

For point z inside a convex polygon, $N$ wedges can be formed with z and all nodes, where $N$ is the number of nodes (Fig. 10). Similar to the swath method, the wedge which contains point q is searched first. The time complexity of this search can be performed in $\mathbf{O}(\log N)$ with a tree structure. Next, whether point q is located in the sub-triangle, that is, inside the polygon, is verified by a third-order determinant-value computation. If point q is located on the same side of each triangle edge, q is inside this triangle (Preparata and Shamos, 1985).

```
Algorithm Wedge_Method(P,q);
Input: P ( a convex polygon with vertices p1,p2,....,pn , and edges e1,e2,....,en ), and
       q ( a point ).
Output: Inside ( a Boolean variable that is set to true if q is inside P and false otherwise ).
begin
   { preprocessing }
   Find an internal point z;
   Divide the polygon into wedges using z as the origin of coordinates;
   Build a binary tree for these wedges;
   { searching begins }
   Decide q in which wedge (wi) from the binary tree:
   if Pi - Pi+1 - q is right-turn then Inside:=false
   else Inside:=true
end
```

This algorithm is applicable only to convex polygons, and is well suited for multiple point queries. Among all other convex-only algorithms, this algorithm requires the least computational time in the query stage. The complexity of forming wedges is $\mathbf{O}(N)$.

## TIME COMPLEXITY WITH LIMITED $N$

An asymptotically large $N$ usually is assumed for the complexity analysis of algorithms. However, for a specific application, $N$ is limited frequently. According to the statistics obtained from the Land Survey Bureau of Taiwan Provincial Government,

Table 1. Computer times required for five million repetitions

| | Integer*2 | Real*6 | Boolean |
|---|---|---|---|
| assign | 0.00 | 0.22 | 0.00 |
| + | 0.17 | 7.64 | — |
| − | 0.16 | 9.94 | — |
| * | 0.82 | 43.12 | — |
| div | 2.42 | — | — |
| / | — | 62.12 | — |
| cos | — | 495.43 | — |
| sin | — | 538.65 | — |
| arctan | — | 374.65 | — |
| sqrt | — | 293.08 | — |
| < | 0.44 | 4.67 | — |
| = | 0.38 | 4.73 | — |
| > | 0.27 | 4.77 | — |
| or | — | — | 0.44 |
| and | — | — | 0.49 |

the average number of nodes for each land parcel in Taiwan is 7.5 (E. -S. Lu, pers. comm., 1995). Under this circumstance, the $c$ constant in O-notation becomes important.

Manber (1989) indicated that when $\mathbf{N} < 16$, bubble-sort, of which the time complexity is $O(N^2)$, is faster than quick-sort, of which the time complexity is $O(N \log N)$. Among algorithms for matrix multiplication, the time complexity of Strassen's algorithm is $O(N^{2.81})$; however, when $\mathbf{N} < 100$, it is slower than regular $O(N^3)$ algorithms. Hence, the number of basic operations required for each algorithm is evaluated for $N = 8$. Table 1 lists the execution times for five million repetitions of some basic operations. The test is performed on an Intel Pentium-66-based personal computer with Turbo-PASCAL. The execution time for the loops has been deducted.

In the following analysis, $A$ indicates a unit addition (+) or subtraction (−), $B$ is a Boolean operation, $M$ is a unit multiplication, and $T$ is a unit arc-tangent computation. To compute the area with a third-order determinant, the number of operations is $(2M + 5A)$. To obtain the sign of the area in order to determine the orientation of the triangle, $(2M + 4A + 1B)$ operations are needed. To verify whether two lines intersect with the algorithm from Taylor (1994), $(7M + 9A + 3B)$ operations are necessary. The times required for the looping and the variable assignment are not included.

*Ray intersection method*

For a polygon with eight edges, eight line intersections must be evaluated. If q is outside, the number of intersections would likely be zero or two. If q is inside, the number of intersections is likely to be one. Therefore, on average, one addition is needed to count the intersections. There is also an odd/even number verification, which is a Boolean operation. The basic operations required are

$$8(7M + 9A + 3B) + 1A + 1B = 56M + 73A + 25B.$$

*Sum of angles method*

The sum of all internal angles of a polygon is $(n-2)\pi$. This requires $(1M + 1A)$ operations. Assuming that the angles are computed from the azimuth of each edge,

$$\varphi_i = \tan^{-1}\left(\frac{x_i - x_o}{y_i - y_o}\right).$$

$(1T + 1M + 2A)$ operations are required. To rep-

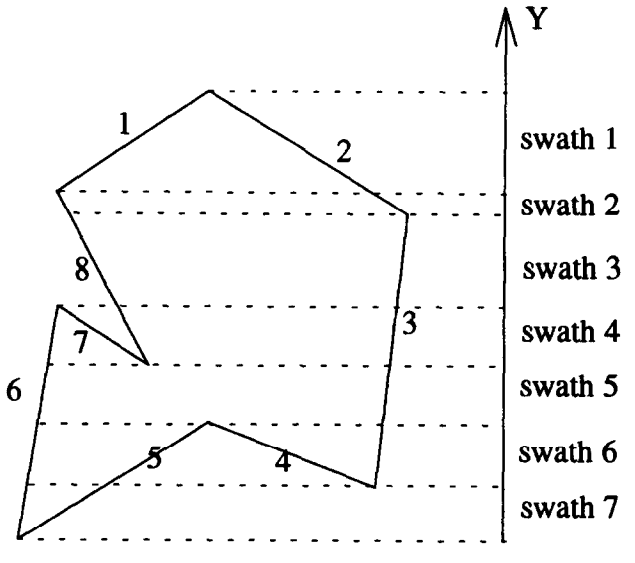


Figure 11. 8-gon example for swath method.

Table 2. Complexity for point-in-polygon algorithms

| Algorithm | Pre-processing | Storage | Query | Polygon condition | Suited query mode | Number of operations for $N = 8$ |
|---|---|---|---|---|---|---|
| grid | $O(1)$ | $O(N)$ | $O(N)$ | raster, any shape | single, multiple | $2M + 2A + 16B$ |
| ray intersection | $O(1)$ | $O(N)$ | $O(N)$ | vector, any shape | single | $56M + 73A + 25B$ |
| sum of angle | $O(1)$ | $O(N)$ | $O(N)$ | vector, any shape | single | $8T + 17M + 48A + 17B$ |
| swath | $O(N \log N)$ | $O(N^2)$ usually $O(N)$ | $O(N)$ usually $O(\log N)$ | vector, any shape | multiple | $26M + 95A + 77.5B$ |
| sign of offset | $O(1)$ | $O(N)$ | $O(N)$ | convex | single | $9M + 18A + 9B$ |
| sum of area | $O(1)$ | $O(N)$ | $O(N)$ | convex | single | $24M + 55A + 1B$ |
| orientation | $O(1)$ | $O(N)$ | $O(N)$ | convex | single | $9M + 18A + 5.5B$ |
| wedge | $O(N)$ | $O(N)$ | $O(\log N)$ | convex | multiple | $10M + 20A + 4B$ |

resent $\phi_i$ with a value between 0 and $2\pi$, extra $(1M + 1A + 1B)$ operations are required to determine in which quadrant $\phi_i$ is located. Therefore, for eight nodes, there are $8(1T + 2M + 3A + 1B)$ operations for azimuth, $8(2A + 1B)$ to compute angles from azimuth, $7A$ for the sum of the angles, and $1B$ for the evaluation of the sum. There is a total of $(8T + 17M + 48A + 17B)$ operations.

*Swath method*

During preprocessing, the normalization on the $y$-coordinate for each vertex takes $N$ multiplications. If a bubble-sort is applied, 17.5 Boolean operations are required for eight nodes.

$$\frac{1}{2}\left(\sum_{i=1}^{n-1} 1 + \sum_{i=1}^{n-1} i\right) = \frac{1}{2}(7 + 28) = 17.5.$$

No arithmetic operations are required to form the balanced binary tree for $(N-1)$ swaths, if the tree is represented as an array. To relate the edges to each swath, the number of operations changes with the shape of the polygon. For the polygon shown in Figure 11, the total number of operations required is $(73A + 46B)$. To find the swath in which point q is located, $(3A + 4M + 7B)$ operations are required on average. However, if the $y$-coordinate of q is outside the $y$-coordinate range of the polygon, no more than two Boolean operations are needed. Because the average number of edges in each swath is about two, to determine whether q is inside the polygon,

$$2(7M + 9A + 3B) + 1A + 1B = 14M + 19A + 7B$$

operations are needed in this procedure. There is a total of $(26M + 95A + 77.5B)$ operations.

*Sign of offset method*

The evaluation of two consecutive edges for the sign, requires two area computations and one Boolean operation. Once different signs are detected, the looping is curtailed. Therefore, for an eight-sided polygon, the expected number of repetitions in the looping is 4.5. The total number of operations is:

$$4.5(2M + 4A + 1B + 1B) = 9M + 18A + 9B.$$

*Sum of area method*

During pre-processing, the polygon's area is computed. With the coordinate method, $(8M + 8A)$ operations are required, not including the operation of dividing the sum by two. In the loop, eight triangle areas are computed. These need $[8(2M + 5A)]$, that is, $(16M + 40A)$. Accumulating these areas takes $7A$. After all, one Boolean operation is need to verify the area, $1B$. There is a total of $(24M + 55A + 1B)$ operations.

*Orientation method*

Once the area of a sub-triangle is detected to be negative, the looping is terminated. Therefore, the number of repetitions can be assumed to be 4.5. Then, the total number of operations is:

$$4.5(2M + 4A + 1B) = 9M + 18A + 5.5B.$$

*Wedge method*

For the pre-processing, to find the internal point z, one can set

$$x_z = \frac{x_1 + x_2 + x_3}{3};$$

$$y_z = \frac{y_1 + y_2 + y_3}{3}.$$

This requires $(2M + 4A)$ operations. Because the nodes of the polygon are already in sequence, no arithmetic operation is required to establish eight wedges and the binary tree. To find the wedge which contains point q, three $(\ln_2 8)$ repetitions of orientation determination are necessary. To determine whether q is inside P, another orientation determination must be performed. There is a total of

$$2M + 4A + 4(2M + 4A + 1B) = 10M + 20A + 4B$$

operations.

Table 2 summarizes each of the algorithm's complexities.

## CONCLUDING REMARKS

(1) When the polygon is represented in raster format, the grid method is well suited. The grid method is also the most efficient because extremely few mathematical computations are needed. However, the number of grid cells is normally larger than the number of nodes. Although other efficient algorithms exist, the execution time increases with the increase of resolution.

(2) For a polygon stored in vector format, the swath method is well suited for multiple point queries. An $\mathbf{O}(N\log N)$ pre-processing can reduce the execution time significantly. The drawback is the increase in space complexity. This, however, may be improved with proper data structures.

(3) If the polygon is known to be convex, the sum of area method, the sign of offset method, and the orientation method, all can be applied for a single point query. The time complexity for all three is $\mathbf{O}(N)$. The wedge method requires $\mathbf{O}(N)$ pre-processing, and the complexity for the query is reduced to $\mathbf{O}(\log N)$. For multiple point queries, the wedge method is well suited.

(4) In practical applications, the number of nodes is usually limited, and polygons are frequently concave. The ray intersection method and the swath method are well suited. When $N$ equals 8, the ray intersection method with bounding rectangles is slightly faster than the swath method. However, the swath method is expected to become more efficient when $N$ increases. For a convex polygon with a single point query, the orientation method is the most efficient algorithm when $N$ equals 8. The efficiency of the wedge method increases when $N$ increases.

(5) All algorithms discussed in this paper are designed for single polygons. For multiple polygons with multiple point queries, the situation would be different. This topic is to be explored in future work.

*Acknowledgments*—The authors wish to thank the anonymous reviewers for their constructive suggestions that improved the paper.

## REFERENCES

Cormen, T. H., Leiserson, C. E., and Rivest, R. L., 1991, Introduction to algorithms: MIT Press, New York, 1028 p.

Guibas, L., and Stolfi, J., 1985, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams: ACM Trans. Computer Graphics, v. 4, no. 2, p. 74–123.

Manber, U., 1989, Introduction to algorithms—a creative approach: Addison-Wesley Publ. Co., Reading, Massachusetts, 478 p.

Nordbeck, S., and Rystedt, B., 1967, Computer cartography point in polygons programs: National Swedish Institute for Building Research Rept., Sweden, 19 p.

Preparata, F. P., and Shamos, M. I., 1985, Computational geometry—an introduction: Springer-Verlag, New York, 398 p.

Saalfield, A., 1987, It doesn't make me nearly as CROSS—some advantages of the point-vector representation of line segments in automated cartography: Intern. Jour. Geographical Information Systems, v. 1, no. 4, p. 379–386.

Salomon, K. B., 1978, An efficient point-in-polygon algorithm: Computers & Geosciences, v. 4, no. 2, p. 173–175.

Sedgewick, R., 1988, Algorithms (2nd ed.): Addison-Wesley Publ. Co., Reading, Massachusetts, 657 p.

Sloan, S. W., 1987, A fast algorithm for constructing Delaunay triangulation in the plane: Adv. Engineering Software and Workstations, v. 9, no. 1, p. 34–55.

Taylor, G., 1994, Point in polygon test: Survey Review, v. 32, p. 479–484.