0-315-08977-6

National Library of Canada — Bibliothèque nationale du Canada

Canadian Theses Division — Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

56894

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

MICHAEL SETH NORTEY

Date of Birth — Date de naissance 6 SEPTEMBER 1948	Country of Birth — Lieu de naissance GHANA

Permanent Address — Résidence fixe

C/o DR. P. A. NORTEY
FACULTY OF ARCHITECTURE
UNIVERSITY OF SCIENCE & TECHNOLOGY
KUMASI, GHANA

Title of Thesis — Titre de la thèse

Point-in-Polygon Algorithms for Geographic Information Systems

University — Université

UNIVERSITY OF ALBERTA

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M.SC - COMPUTING SCIENCE

Year this degree conferred — Année d'obtention de ce grade 1982 (Spring)	Name of Supervisor — Nom du directeur de thèse DR. STAN CABAY

Date Dec 23, 1981	Signature [illegible]

NL-91 (4-77)

THE UNIVERSITY OF ALBERTA

Point-in-Polygon Algorithms for Geographic Information Systems

by

© Michael Seth Nortey

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING, 1982

[illegible]

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR	Michael Seth Nortey
TITLE OF THESIS	Point-in-Polygon Algorithms for Geographic Information Systems

DEGREE FOR WHICH THESIS WAS PRESENTED: MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED: 1982

SIGNED ..[illegible]......................

(PERMANENT ADDRESS)

C/O Dr. P. A. Nortey,

Faculty of Architecture,

University of Science & Technology,

Kumasi, GHANA.

DATED ....December 17, 1981

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Point-in-Polygon Algorithms for Geographic Information Systems** submitted by Michael Seth Nortey in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE.

[illegible]

Supervisor

[illegible]

[illegible]

[illegible]

Date December 17, 1981

ABSTRACT

Algorithms for resolving the point-in-polygon proble
are surveyed. The scope of the survey includes only t
algorithms, and consequently those applications
polygonal vertices are represented using vector rath
raster notation.

Two classes of algorithms are described. Th
methods sequentially process the n vertices of the
and require O(n) operations to resolve the point-
problem. The pre-conditioned methods operate
hierarchical preprocessed representation of the poly d
are of complexity $O(\log n)$, $O(\log^2 n)$, or $O(n)$. However, the
pre-conditioned algorithm have associated with them
additional preprocessing and storage costs.

ACKNOWLEDGEMENTS

To my supervisor, Dr. Stan Cabay, sincere thanks for the patience and guidance he provided me during the research. The thesis committee members Drs. Wayne Davis, Jean Claude Muller and Len Schubert read and commented on all chapters of this thesis.

I must mention my good friend, Kofi Marfo whose invaluable help enabled the generation of numerous diagrams.

My sponsor, the Department of Computing Science, provided the essential financial assistance.

Dr. Narh Omaboe of Trenton, Ontario, gave all the moral support needed to bring about this complete research document.

My sincere thanks to them all.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1 Background

The concept of a geographic information system is essentially that of storage and retrieval of information about specific locations or regions, usually on the earth's surface. In such a system, descriptive data (such as housing information, rainfall, etc.) is structured according to these geographic locations or regions. Because of this structure, rapid retrievals of descriptive information about a geographic location or region are made possible.

The boundary of a geographic region in geographic information systems is frequently represented discretely by a finite sequence of points. By drawing imaginary cords between successive points, the resultant polygon serves as an approximation to the geographic region. The discrete boundary points are frequently obtained from their source (e.g. maps) by means of computerized digitizers; and in practice, the number of points required for an accurate approximation is large.

In storing descriptive data associated with some geographic point, it often becomes necessary to identify a region (polygon) which contains that point. The cost of resolving the identification problem, more typically called the point-in-polygon problem, dominates the cost of data entry in most geographic information systems. It is this problem that is addressed in this thesis.

The point-in-polygon problem arises in a variety of other applications including such important ones as image processing and pattern recognition. Whereas the size of the polygons (i.e., the number of sides or vertices) may be small in some of these applications, in this thesis the focus shall be on large polygons (e.g. 1000 sides).

1.2 Statement of the Problem

There are a large number of computer representations for regions and polygons [5,15,17,18,21,22,24,26,28,36,38]. Grids (raster), curves, points and polygonal lines are some of the forms used. For the purpose of this thesis, we choose to define a polygon as follows and in so doing we restrict the scope of this thesis to those applications where this definition is valid.

DEF'N: Let $P=(V_1,V_2,...,V_{n+1})$ be a sequence of points in 2- dimensional Euclidean space E_2, where $V_{n+1}=V_1$. Denote the coordinates of V_i by $V_i=(x_i,y_i)$ and let (V_i, V_{i+1}) be the line segment joining the point V_i and V_{i+1} for $i=1,2,....,n$. Then P describes a <u>polygon</u> if (V_i, V_{i+1}) does not intersect (V_k, V_{k+1}) whenever $i \neq k$.

DEF'N: The polygon $P=(V_1,......,V_{n+1})$ is convex if for any two points contained in P, the line segment joining these two points is contained in P.

DEF'N: A test point Z is a set of coordinates (x,y) in the cartesian coordinate system.

DEF'N: The point-in-polygon problem is defined as follows: Given the closed polygon P, determine whether or not the test point Z is contained in P.

For solving this problem, there are four possible relations between the given point and the region. The point can lie outside, inside, on the boundary, or coincide with a boundary point of the region as illustrated in Figure I.1.

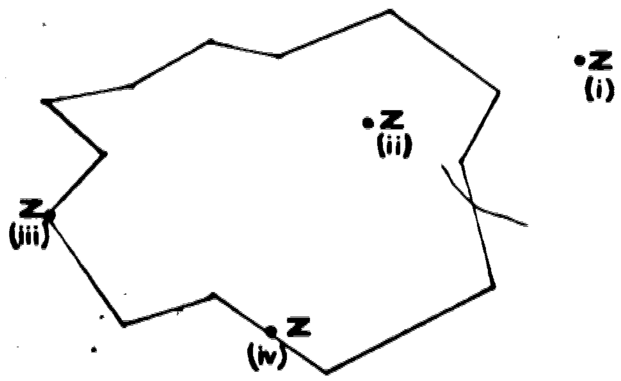


FIGURE I.1: Positions of a given Point Z=(x,y) Relative to a Polygon

As illustrated ir Figure I.1, we say

Case i. $Z \not\subset P$ (Z is not contained in region P)

Case ii. $Z \subset P$ (Z is contained in region P)

Case iii. $Z \subset P$ (Z is contained in region P)

Case iv. $Z \subset P$ (Z is contained in region P)

The cases that Z is a boundary point are distinguished above because some algorithms described within give such cases special consideration.

1.3 Outline of Thesis

The purpose of this thesis is to survey methods for the solution of the point-in-polygon problem. However, no attempt is made to include in the discussion all works concerned with this problem. Instead, this thesis describes only those methods which are considered to be significantly different in their approach or performance. Since such considerations are necessarily biased, we apologize to those authors whose work we may not have referenced.

In the description of all methods herein the intention is always to present only those notions essential to understanding the essence of the material. For details and for the handling of exceptional cases, the reader may need to refer to the original source.

Chapter II reviews direct methods where no initial processing of the sides of the polygon is required. Extensions to some of the algorithms are discussed. Typically, these methods require O(n) operations to resolve the point-in-polygon problem.

Chapter III deals with methods that require pre-processing of the polygon in order to speed up the search time. Typically these methods require O(n), O($\log^2$n), or O(log n) operations to resolve the point-in-polygon problem,but in exchange they require the construction and the storage of a more complex data structure.

Chapter IV reviews methods which quickly eliminate the areas which do not contain the given point. Examples are presented and implementation factors are considered.

Chapter V summarizes the previous chapters. Conclusions are drawn and recommendations for further research proposed.

CHAPTER II

DIRECT METHODS

2.1 Introduction

This chapter describes those algorithms, for resolving the point-in-polygon problem, which can be applied directly to the polygon represented as a sequence of the oriented vertices $V_1, \ldots, V_n$ (see Chapter I). These methods are to be distinguished from those of Chapter III which require the polygon first to be transformed into different representations before the methods are applied.

The methods described in this chapter are therefore applicable to those applications where a polygon is subjected to frequent changes. No preprocessing of the polygon representation is required before the point-in-polygon algorithms of this chapter are used.

Two of the methods are restricted to convex polygons. Polygonal regions which are non-convex, however, can be split into the union of convex polygons. Refer to Nordbeck and Rystedt[23] where a non-convex polygon is made convex by adding to it an appropriate number of triangles. In those cases where many new triangles must be added, the two methods affected may become very slow.

2.2 The Directed Line Method

This method, described by Aldred[1] and by Nordbeck et al[23], may be applied to the point-in-polygon problem only if the polygon is convex.

Proceeding sequentially around the polygon in a counter- clockwise direction, at the i'th step, the method determines whether or not the given point $Z=(x,y)$ lies to the left or to the right of the line directed through the vertices $V_i=(x_i,y_i)$ and $V_{i+1}=(x_{i+1},y_{i+1})$ of the polygon P. (see Figure II.1).

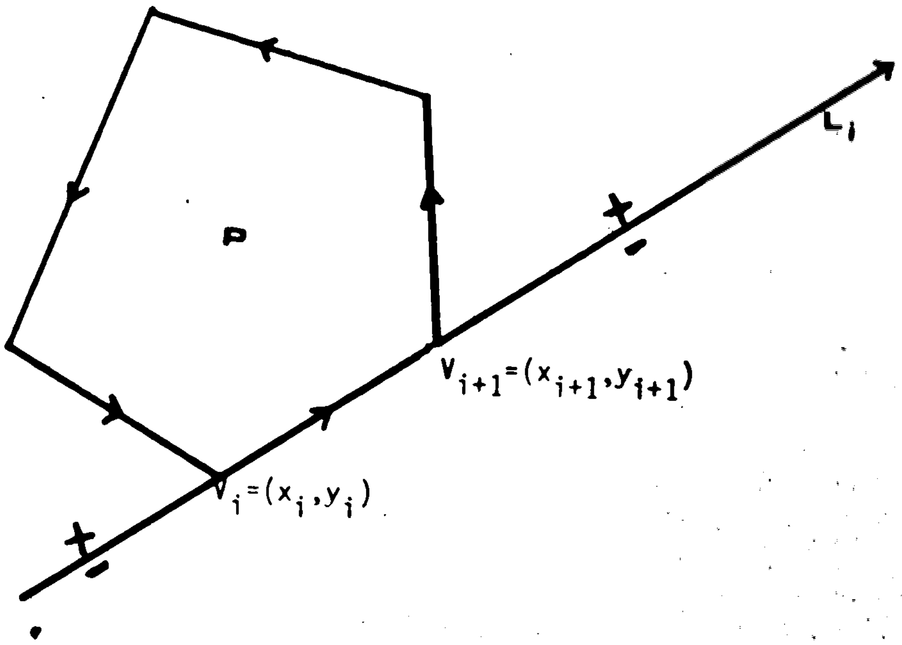


FIGURE II.1: Directed Line Method

Let

$$F_i(x,y) = (x_{i+1} - x_i)y - (x-x_i)y_{i+1} + (x-x_{i+1})y_i.$$

Then

$$F_i(x,y) \begin{cases} < 0, \text{ if } Z \text{ lies to the right of } L_i, \\ = 0, \text{ if } Z \text{ lies on } L_i, \\ > 0, \text{ if } Z \text{ lies to the left of } L_i. \end{cases}$$

It is clear that for the polygon $P=V_1,V_2,......,V_{n+1}$ (with $V_{n+1}=V_1$),

$$Z \subset P, \text{ if } F_i(x,y) > 0, \text{ for all } i=1,2,.....,n ,$$

$$Z \subset \partial(P), \text{ if } F_k(x,y) = 0, \text{ for some } k,$$
$$\text{and } F_{k-1}(x,y) > 0,\ F_{k+1}(x,y) > 0$$

$$Z \not\subset P, \text{ if } F_k(x,y) < 0, \text{ for some } k.$$

In the worst case when $Z \subset P$, the cost of the method is

$$C = n(3C_m + 5C_a + C_c)$$

(where C_m is the cost of a multiplication, C_a is the cost of an addition or subtraction, and C_c is the cost of a comparison).

If $Z \not\subset P$, however, the cost of the method may be significantly less, since the method terminates with the first occurrence of $F_k(x,y) < 0$.

2.3 The Area of the Polygon Method

The area of the polygon method described by Aldred[1], like the directed line method, is restricted to polygons that are convex.

Proceeding sequentially around the polygon in a counter- clockwise direction, at the i'th step, the method determines the area A_i of the triangle determined by the given point $Z=(x,y)$ and the two vertices $V_i=(x_i,y_i)$ and $V_{i+1}=(x_{i+1},y_{i+1})$ of the polygon P (see Figure II.2).

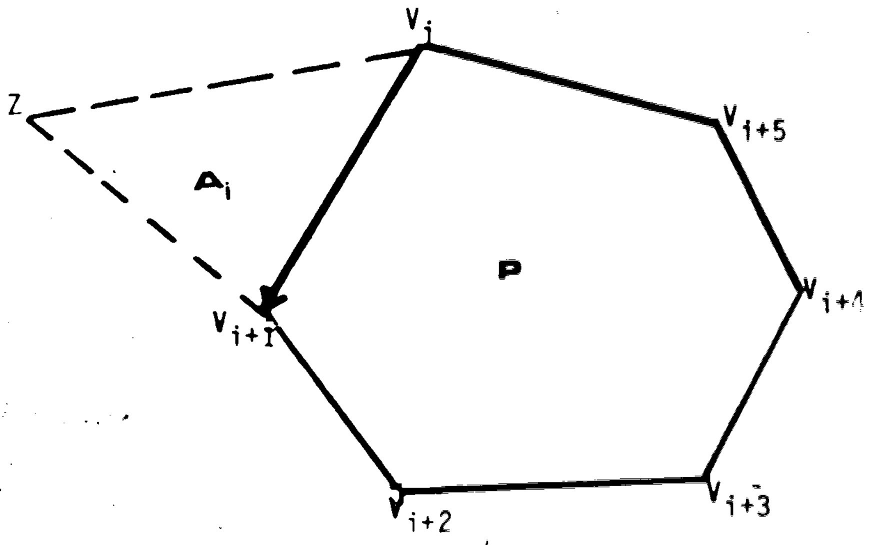


FIGURE II.2: Polygonal Area Method

The area A_i is expressed using the determinant notation as

$$2A_i = \begin{vmatrix} x & x_i & x_{i+1} & x \\ y & y_i & y_{i+1} & y \end{vmatrix}$$

The determinant is to be interpreted as pairwise multiplication of elements in the two rows and then subsequent additions as follows:

$$2A_i = [(xy_i+x_iy_{i+1}+x_{i+1}y) - (yx_i+y_ix_{i+1}+y_{i+1}x)]$$
$$= [x(y_i-y_{i+1})+x_i(y_{i+1}-y)+x_{i+1}(y-y_i)]$$

It is easy to show that the area A of the polygon P satisfies

$$2A = \sum_{i=1}^{n} 2A_i$$

The summation above may be simplified as follows (but note that this simplification is not germane to the point-in-polygon problem):

$$2A = \begin{vmatrix} x_1 & x_2 & \cdots\cdots x_n & x_{n+1} \\ y_1 & y_2 & \cdots\cdots y_n & y_{n+1} \end{vmatrix}$$
$$= \sum_{i=1}^{n} (x_iy_{i+1} - y_ix_{i+1})$$

Let

$$2S = \sum_{i=1}^{n} | 2A_i |.$$

Since $A_i \geq 0$, $i=1,2,\ldots\ldots,n$, if $Z \in P$ or $Z \in \partial(P)$, and since all of the A_i's are not of the same sign if $Z \notin P$, it follows that

$Z \in P$, if $S=A$,

$Z \in \partial(P)$, if $A_i=0$ for some i,

$Z \notin P$, if $S \neq A$.

The cost of Aldred's method is

$$C = n(3C_m + 7C_a + C_c)$$

(where C_m is the cost of multiplication, C_a is the cost of addition or subtraction, and C_c is cost of comparison).

As an extension to the above method, it is clear that

$$Z \subset P \text{ if } A_i > 0,\ i=1,2,\ldots\ldots,n$$

$$Z \subset \partial(P) \text{ if } A_i = 0, \text{ for some } i,$$

$$Z \not\subset P \text{ if } A_i < 0, \text{ for some } i.$$

Hence, the cost of the method is reduced to

$$C = n(3C_m + 5C_a + C_c)$$

in the worst case.

2.4 The Sum of Angles Method

This method, due to Nordbeck and Rystedts[23] proceeds sequentially around the polygon, determining at the i'th step the angle θ_i subtended by lines from the points $V_i = (x_i, y_i)$ and $V_{i+1} = (x_{i+1}, y_{i+1})$ to the point $Z=(x,y)$. (See Figure II.3).

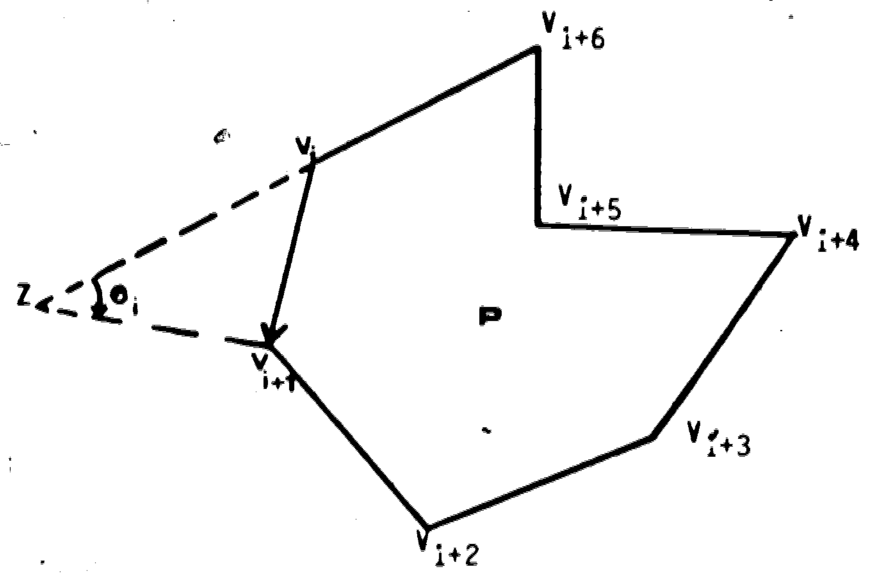


FIGURE II.3: Sum of Angles Method

The angle θ_i is given by

$$\theta_i = \tan^{-1}(y_{i+1}-y)/(x_{i+1}-x) - \tan^{-1}(y_i-y)/(x_i-x),$$

where $-180^0 \leq \theta_i \leq 180^0$.

Let $\theta = \sum_{i=1}^{n} \theta_i$.

Then

$$Z \in P \text{ if } \theta=360^0,$$

$$Z \in \partial(P) \text{ if } \theta_i = 180^0 \text{ for any } i,$$

$$Z \notin P \text{ if } \theta=0.$$

Special attention must be given to any vertex $V_i=(x_i,y_i)$ where x_i-x is "small" in order to avoid division by zero and to account for large round-off errors. Having determined that $Z \notin \partial(P)$, however, the effect of round-off errors is not otherwise critical to the algorithm; that is, $Z \in P$ if θ is "close" to 360^0, and $Z \notin P$ if θ is "close" to 0^0.

A considerable improvement to the methods has been suggested by Hall[14]. Rather than accumulating the partial sum $\sum_{i=1}^{k} \theta_i$, the sine and cosine of $\sum_{i=1}^{k} \theta_i$ are accumulated instead. Then, at the vertex V_{k+1},

$$\mathrm{Sin}(\theta_{k+1} + \sum_{i=1}^{k} \theta_i) = \mathrm{Sin}(\sum_{i=1}^{k} \theta)\mathrm{Cos}(\theta_{k+1})$$

$$+ \mathrm{Cos}(\sum_{i=1}^{k} \theta_i)\mathrm{Sin}(\theta_{k+1}).$$

$$\mathrm{Cos}(\theta_{k+1} + \sum_{i=1}^{k} \theta_i) = \mathrm{Cos}(\sum_{i=1}^{k} \theta_i)\mathrm{Cos}(\theta_{k+1})$$

$$- \mathrm{Sin}(\sum_{i=1}^{k} \theta_i)\mathrm{Sin}(\theta_{k+1})$$

The improvement in this approach arises from the observation that $\mathrm{Sin}(\theta_{k+1})$ and $\mathrm{Cos}(\theta_{k+1})$ may be computed using vector dot and cross products.

The vector dot product satisfies

$$(V_k-Z).(V_{k+1}-Z) = |V_k-Z|.|V_{k+1}-Z|.\text{Cos } \theta_{k+1}.$$

The vector cross product satisfies

$$(V_k-Z)x(V_{k+1}-Z) = |V_k-Z|.|V_{k+1}-Z|.\text{Sin } \theta_{k+1}.$$

Only at the final step does a trigonometric routine need to be invoked in order to compute

$$\theta = \tan^{-1}(\text{Sin } \theta/\text{Cos } \theta) \text{ , where } \theta = \sum_{i=1}^{n} \theta_i.$$

The computation of $\sin(\sum_{i=1}^{k+1} \theta_i)$, $\cos(\sum_{i=1}^{k+1} \theta_i)$,given $\text{Sin}(\sum_{i=1}^{k}\theta_i)$, $\text{Cos}(\sum_{i=1}^{k}\theta_i)$ requires 13 multiplications, 10 additions, 2 divisions and 1 square root evaluation. Ignoring the cost of the final $\tan^{-1}$ computation, the cost of Hall's method is given by

$$C = n(13C_m + 10C_a + 2C_d + C_s).$$

(where C_m is the cost of multiplication, C_a is the cost of addition or subtraction, C_d is the cost of division, and C_s is the cost of a square root evaluation).

2.5 Line Crossing Method

The method has been described from a variety of perspectives by a number of different authors [1,2,3,4,5,6,9, 10,13,14,15,22,32]. It can be applied directly to polygonal regions which are convex, non-convex, and even to those which are not simply connected.

Basically, the method consists first of passing a line through the point Z=(x,y) and parallel to the x-axis. (Although any line through Z will do, a horizontal or vertical line is computationally best.) The number of intersection points of this line with the polygon boundaries is then tabulated (see Figure II.4).

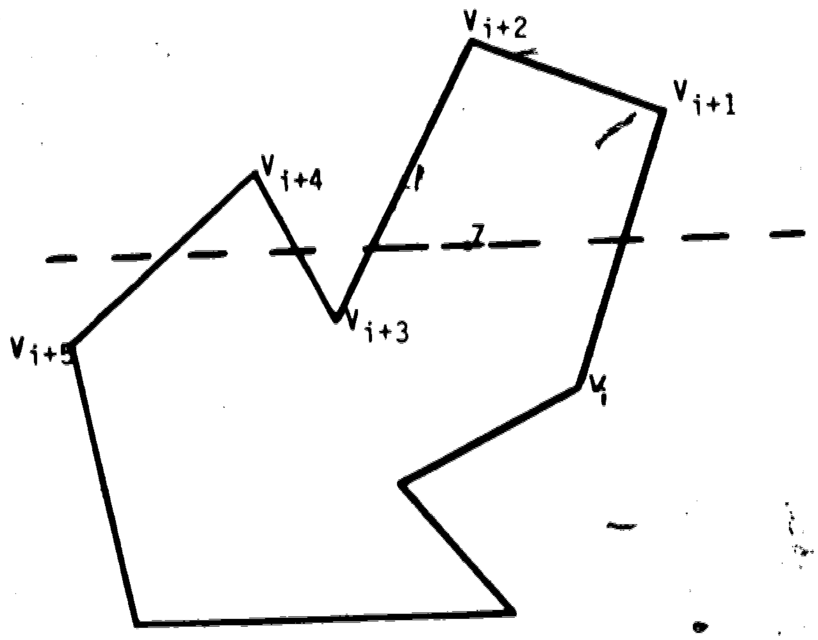


FIGURE II.4: Line-Crossing Method

Computationally, at the i'th step, i=1,2,.....,n, the sign of $(y_{i+1}-y)$ is determined and compared with sign $(y_i - y)$. If the signs are different, the polygon boundary has likely crossed the given line, and x-coordinate of Z is then compared with the x-coordinates of V_i, V_{i+1} in order to determine the position of Z relative to the segment V_i, V_{i+1}. If Z is contained between the extreme x-coordinates of V_i, V_{i+1}, then the vertices V_i and V_{i+1} are linearly interpolated to determine if Z lies on or left of the polygonal segment (V_i,V_{i+1}). Special attention must be given to the case that the sign of $(y_{i+1}-y)$ is 0.

The algorithm in greater detail is given below. The algorithm returns the value CONTAIN. If CONTAIN = 1, then Z is contained in the polygon; if CONTAIN = 0, then Z lies on the boundary of the polygon; and finally, if CONTAIN = -1, then Z lies outside the polygon.

```
1.   (Comment: Initialization)

     Set CONTAIN = -1
     Compute $S_2$ = sign of $(y_1-y)$
     Set i = 1
```

```
2.  (Comment:  Process next point.  $S_1$ and $S_2$ record the
    vertical positions of Z relative to the vertices $V_{i-1}$
    and $V_i$ respectively.)
    Set $S_1 = S_2$
    Set $i = i+1$
    Compute $S_2$ = Sign of $(y_i - y)$
3.  If $S_1 \neq S_2$ or $S_2 = 0$
    then
        (Comment:  Vertical cross-over found)
        go to step 4
    else
        (Comment:  Process next point)
        go to step 9.
4.  If $x < \min(x_{i-1}, x_i)$
    then
        (Comment:  Z lies left of segment $(V_{i-1}, V_i)$
        set CONTAIN = -CONTAIN
        go to step 9
    else go to step 5
5.  If $x > \max(x_{i-1}, x_i)$
    then
        (Comment:  Z lies right of segment $(V_{i-1}, V_i)$
        go to step 9
    else go to step 6
```

```
6.   (Comment:  Z lies in rectangle containing segment
     $(V_{i-1},V_i)$, and therefore interpolate)
     F = sign of $[y.(x_i - x_{i-1}) - y_i.(x-x_{i-1}) + y_{i-1}.(x-x_i)]$
     If ($S_2$ = -1) then set F = -F
7.   If F = 0
     then
          (Comment:  Z lies on segment $(V_{i-1}, V_i)$
          return (CONTAIN = 0)
     else go to step 8.
8.   If F > 0
     then
          (Comment:  Z lies left of segment $(V_{i-1}, V_i)$
          set CONTAIN = -CONTAIN
9.   If i < n + 1
     then go to step 2
     else return (CONTAIN).
```

Since interpolation is required infrequently (at most twice, if P is convex), the cost of the method is approximately

$$C = n(C_a + 3C_c)$$

(where C_a is the cost of an addition or subtraction, C_c is the cost of a comparison).

2.6 Summary

In this chapter, ~~we~~ have discussed four methods of determining whether or not a point lies in a polygon. These methods are summarized in terms of their worst case computation cost and in terms of restrictions in Table I.

Method	Point Processing Cost	Restrictions
Directed Line	$n(3C_m+5C_a+C_c)$	Convex Polygons
Area of Polygon	$n(3C_m+5C_a+C_c)$	Convex Polygons
Sum of Angles	$n(13C_m+10C_a+2C_d+C_s)$	None
Line Crossing	$n(C_a+3C_c)$	None

TABLE 1: Summary of Direct Methods

It is clear from the table that the Line Crossing method, in the worst case, when the point is within the polygon, is the best of all the 'Direct Methods' discussed. In addition, it is not restricted to convex polygons as are the first two methods and it is significantly faster than the third method.

When a point is outside the polygon, however, each of the first three methods may terminate early, whereas the line crossing methods continues to process all points of the polygon. Thus, if the majority of points tested happen to lie outside the polygon, it may appear that any of the first three methods may be superior, on the average, to the line crossing method. However, as we shall see in Chapter IV, initial inexpensive tests can be applied to resolve the point-in-polygon problem [illegible] most points which lie outside a polygon. The superiority of the line crossing method, therefore will prevail in all cases.

CHAPTER III

PRE-CONDITIONED METHODS

3.1 Introduction

All the methods described in the previous chapter require O(n) operations to resolve the point-in-polygon problem. Asymptotically, better methods are available, and these methods are the subject of this chapter.

There is, however, a price to pay for this speed. All the methods require that the polygon be represented by a certain, often complex structure. The storage requirements for this structure are often exorbitant ($O(n^2)$) and costs of achieving this structure (preprocessing) can be high ($O(n^2 \log n)$). Therefore, these methods can be practical only in a static environment, that is, in an environment where the polygonal boundaries seldom change.

There is one additional drawback with these asymptotically fast methods. Because of the complex structure by which a polygon is represented, access and other overhead costs reduce the practicability of fast methods to problems where n is "large". The threshold for n, as n increases, wherein fast methods become superior to classical methods (e.g., the line-crossing method) is an important implementation consideration. This matter, however, is beyond the scope of the current investigation, and is not addressed in the following pages.

Five methods, namely, Swath, Chain, Triangle, Rectangle, and Strip, are described in the following sections.

Each of the sections includes a description of the required preprocessing of a polygon, followed by a description of the point-in-polygon algorithm. Costs and storage requirements are also given.

3.2 SWATH Method

This method is due to Salomon[29,30] and to Shamos[31]. The preprocessing step proceeds as follows:

Preprocessing

Swaths are formed by constructing horizontal lines through each vertex of the polygon[1]. The swaths are then ordered by decreasing y-coordinates.

To illustrate the formation of swaths, consider the eight-sided polygon given in Figure III.1. This polygon is described by six swaths. SWATH(1) contains edges (5) and (6) within the y-interval [3.7, 2.2), SWATH(2) contains edges (3), (4) and (6) within the y-interval [2.2, 1.9), and finally SWATH(6) contains edges (1) and (8) within the y-interval [-2.6,-5][2]. Having obtained the swath coordinates, the swaths are ordered. This ordering is possible since the swaths are mutually exclusive.

[1] A swath is defined as the region between two such horizontal lines.

[2] SWATH(i) denotes the i-th swath.

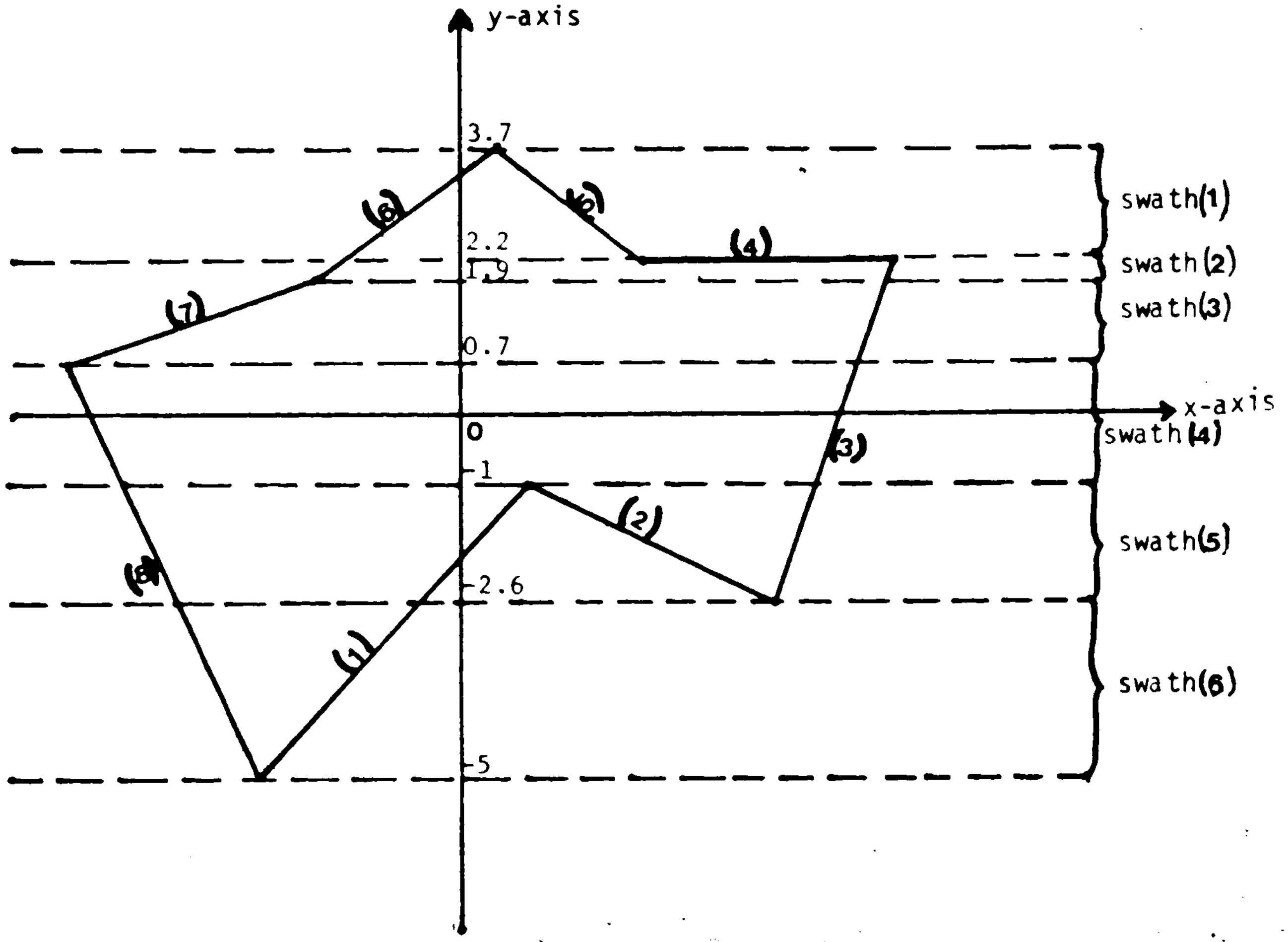


FIGURE III.1: A Polygon and its swaths definition.

Finally, the polygon segments within each swath are ordered from left to right and indexed.

The complete swath method is summarized by the following two algorithms.

Preprocessing Algorithm

1. Sort the vertices $V_i = (x_i, y_i)$, $i=1,\ldots\ldots n$ according to the decreasing values of y-coordinates y_i. Denote the ordered list by $Y(i)$, $i=1,\ldots\ldots,m$, from which all equal entries have been deleted so that $m \leq n$.
2. Denote each consecutive y pair of endpoints $[Y(i),Y(i+1)]$, for $i=1,2,\ldots\ldots,m-1$, (with $Y(i) > Y(i+1)$) by SWATH(i).
3. Within SWATH(i), establish a left-to-right index of all line segments which intersect it and store the index in SWATH(i,j+1), j=1,2,...

   (comment: SWATH(i,1) is the number of segments in the SWATH(i)).

Point-in-polygon Algorithm

The algorithm accepts the matrix SWATH(i,j) and a test point $Z=(x,y)$ as input; and returns as output the value 'true' if Z is contained in the polygon, and 'false' otherwise.

[3] SWATH(i,j+1) denotes the jth line segment within the ith swath.

1. Compare y against Y(i), i=1,....,m to determine the SWATH(i) containing y. If no swath contains y, then return the value 'false'; otherwise go to step 2.
2. Scan line segments in SWATH(i) according to the order dictated by SWATH(i,j) j=2,3,.... Let SWATH(i,Index) reference the first line segment which lies on the right of Z. If no such index exists, then return the value 'false'; otherwise go to step 3.
3. If Index =1 modulo 2, then return 'true'.
   Else return 'false'.

Analysis

[Preprocessing]

Assuming n is the number of sides of the original polygon, there are at most n-1 swaths. Thus (n log n) operations[4] are sufficient to sort the y-coordinates in the decreasing order for defining the swaths. Within each swath there are at most n segments (2 if the polygon is convex). It therefore takes at most (n log n) operations for the left-to-right ordering of segments within each swath. Since there can be O(n) Swaths, the total cost of preprocessing is $O(n^2 \log n)$ operations

[4] log n denotes the logarithm of n to the base 2

[Storage]

From the above algorithm, there are at most n-1 swaths to be stored away. In each swath, there are at most n segments. Subsequently, the total computer storage required by the (n by n) swath matrix is $O(n^2)$ units.

[Search]

Finding the appropriate swath containing Z can be done in O(log n) steps (using binary search techniques on n elements). In the selected swath, there are two possibilities.

If the polygon is convex, four additional comparisons are sufficient to determine the position of the point with respect to the quadrangle formed by two line segments and the swath. The total search in this case is (log n + 4) operations and the cost is therefore O(log n).

If the polygon is not convex, Z is compared against each line segment in the swath until a segment on the right of Z is found. This is essentially the line-crossing method (described in section 2.5) applied to the polygonal portions in the swaths. Since these polygonal portions can have as many as n-1 sides in the worst case, the cost of this step in Salomon's method and Shamos' method can attain a complexity of O(n) operations (however, see refinements below). But, in practice, the number of line segments in a swath will be much smaller than n-1 (e.g. two if the polygon is convex), and the cost of the Salomon[29,30] and Shamos[31] algorithms will be dominated by the cost of searching for the appropriate swath containing Z. Summarizing the cost of Salomon and Shamos algorithms is O(n) operations in the worst case, but O(log n) operations for most polygons.

<u>Comments - Refinement</u>

An obvious refinement of Salomon and Shamos algorithm is available. Having determined the swath containing Z in O(log n) operations using binary search techniques, the same techniques can be applied to the ordered line segments within the swath to determine that first segment lying on the right side of Z. Since both steps are now of complexity O(log n), the refined algorithm is of the same complexity. With this refinement, the algorithm becomes essentially that of Dopkin and Lipton's[8].

Note, however, that Dobkin and Lipton present their method in a more general setting. The problem considered by them is that of identifying that polygon (if any) which contains a given point from the set of all polygons formed by the intersections of n lines in a plane. But any prescribed polygon is determined by a set of ordered lines.

and the intersections of each line with its predecessor and successor only. Thus, the point-in-polygon problem is a special case of the problem considered by Dobkin and Lipton.

3.3 CHAIN Method

This method, discussed in terms of planar graphs, is due to Lee and Preparata[18] who utilize certain geometric properties of a chain for solving the point-in-polygon problem. The concepts underlying this method lend themselves to many different applications. For example, their method can be used to determine if a given point is contained in a convex region in 3-dimensional space (see [18,20,27]).

Before describing this method, some preliminary definitions are required.

DEF'N: Given a polygon P with vertices V_i, $i=1,2,\ldots\ldots,n$, a chain (with respect to P) is defined to be an ordered subset $(V_{i_1},V_{i_2},\ldots\ldots\ldots,V_{i_k})$ of vertices of P. Implicit in this definition is the existence of an edge $(V_{i_j},V_{i_{j+1}})$ joining the vertices V_{i_j} and $V_{i_{j+1}}$.

DEF'N: A chain is monotone with respect to the y-axis (or an arbitrary line L) if the orthogonal projections of the vertices of the chain onto the y-axis (i.e. the y coordinate) are ordered.

An example of a monotone chain is illustrated in Figure III.2.

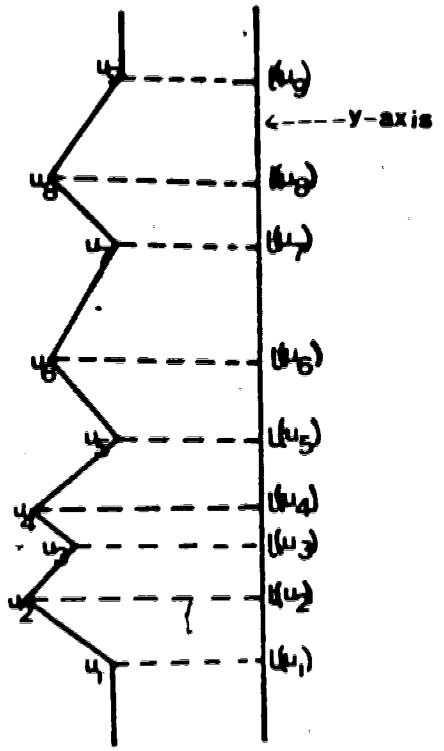


FIGURE III.2: A Chain Monotone with the y-axis

DEF'N: The set $C=(C_1,C_2,\ldots,C_k)$ of chains with respect to the polygon P is <u>complete</u> if it satisfies the following conditions:

1. Each edge of the polygon P belongs to at least one chain C_i;
2. Each chain C_i, $i=1,2,\ldots,k$ is monotone with respect to the y-axis;
3. The chains can be ordered such that $C_1<C_2<\ldots<C_k$, where $C_i<C_j$ means C_i lies on the left of C_j.

This definition ensures that the edges of C_i do not cross those of C_j, $i\neq j$, and that a total ordering of the chains is maintained.

Lee and Preparata's method begins by constructing for a polygon a complete set of monotone chains (the preprocessing step). This construction then permits the fast resolution of the point-in-polygon problem.

Only convex polygons lend themselves naturally to the construction of a complete set of monotone chains. For other polygons, it is convenient first to transform the polygon into a suitable regular form.

DEF'N: Let $(V_1^*,\ldots V_n^*)$ be a sorting of the vertices $(V_1,\ldots,V_n)$ of a polygon P so that $y(V_1^*)\le y(V_2^*)\le\ldots\le y(V_n^*)$. A vertex V_j^* is said to be regular if there are integers $i<j<k$ such that edges (V_i^*,V_j^*) and (V_j^*, V_k^*) are in P. If all of the vertices with the exception of V_1^* and V_n^* are regular then P is said to be a regular graph.

For example, the vertex N of the illustrated polygon in Figure III.3a is not regular, whereas G is.

The transformation of a polygon into a regular graph consists of drawing a minimum number of additional edges between vertices of P so that for each original edge of P there exists a monotone chain in the graph passing through V_1^* and V_n^* and containing this edge. An example of a polygon and its associated regular graph are given in Figures III.3a and III.3b.

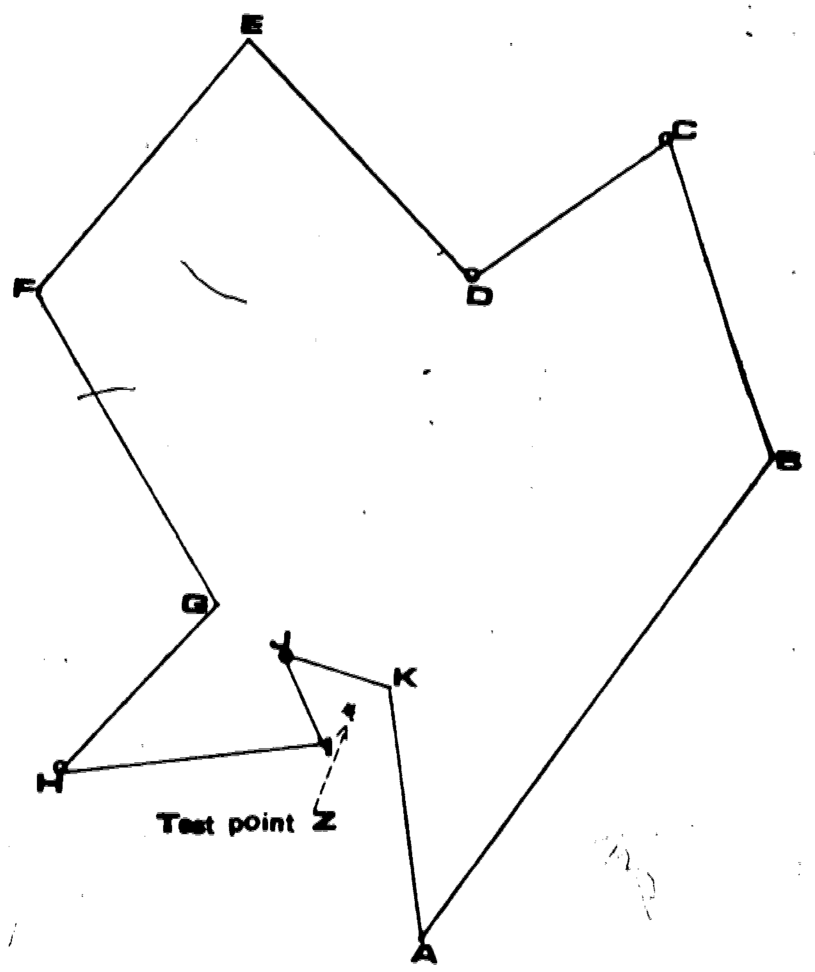


FIGURE III.3a: Non-Convex Polygon

NOTE: A vertex marked 'o' indicates non-regular vertex.

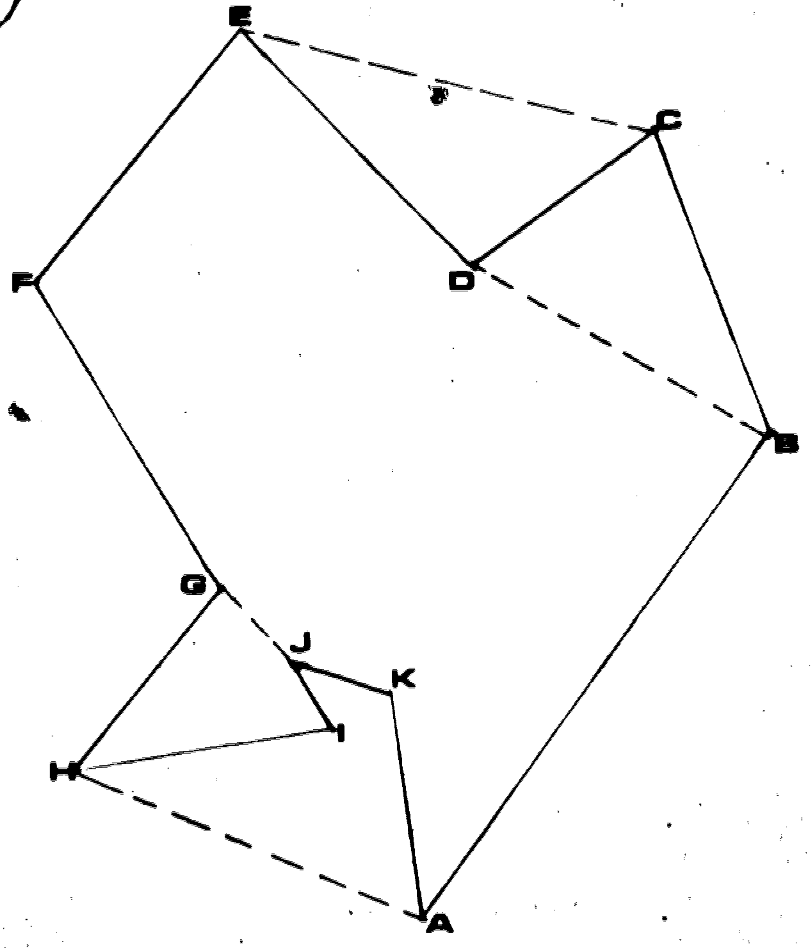


FIGURE III.3b The Associated Minimal Regular Graph

The regularization scheme is described as follows:

```
1.  Sort the vertices in non-decreasing order of
    y-coordinates as $V_1^*, V_2^*, \ldots\ldots, V_n^*$
    (Comment: $V_i^*$ is the vertex being processed)
2.  Set $j = 2$
3.  Set $i = j - 1$
        $k = j + 1$
4.  (Comment: Descending Pass)
    4.1 If there is an edge $(V_i^*, V_j^*)$
        then go to step 5
        else set $i = i - 1$
    4.2 If $i > 0$
        then go to step 4.1
        else
            set $i = j - 1$
            go to step 4.3
    4.3 (Comment:  Draw a line segment)
        If $(V_i^*, V_j^*)$ does not intersect any other segment
        then
            (Comment:  such a segment must exist for some i)
            draw segment $(V_i^*, V_j^*)$
            go to step 5
        else
            set $i = i - 1$
            go to step 4.3
```

```
5.   (Comment:  Ascending Pass)
     5.1 If there is edge ($V_j^*$, $V_k^*$)
         then go to 6
         else set $k = k + 1$
     5.2 If $k < n$
         then go to step 5.1
         else
             set $k = j + 1$
             go to step 5.3
     5.3 (Comment:  Draw a line segment)
         If ($V_j^*$, $V_k^*$) does not intersect with any other
         segment
         then
             (Comment: such a segment exists for some i)
             draw segment ($V_i^*$, $V_j^*$)
             go to step 6
         else
             set $k = k + 1$
             go to step 5.3

6.   Set $j = j + 1$
     If $j < n$
     then go to step 3
     else (Comment:  all vertices are regular)
     Stop.
```

Once the minimal regular graph for a polygon has been constructed, all unique monotone chains $(C_1, C_2, \ldots\ldots, C_k)$ passing through V_1^* and V_n^* are ordered so that C_i lies on or on the left of C_j if $i < j$. These chains form a complete set of monotone chains. The monotone chains for the exemplified polygon are given in Figure III.3c.

The complete set of monotone chains C can be represented hierarchically in a binary search tree. The algorithm for the binary tree is described as follows:

Given a complete set of chains $C = (C_1, C_2, \ldots\ldots, C_m)$, a binary tree of m chains is constructed by selecting the k^{th} element of the set (where $k = \lfloor m/2 \rfloor + 1$) as the root of the tree. The k^{th} element partitions the set C into a left subtree $(C_1, C_2, \ldots\ldots, C_{k-1})$ with elements less than C_k, and a right subtree $(C_{k+1}, \ldots, C_m)$ with elements greater than C_k. Each subtree is further partitioned by its middle element into smaller subtrees. This process is repeated until each subtree is null. For the example at hand, the search tree is given in Figure III.4.

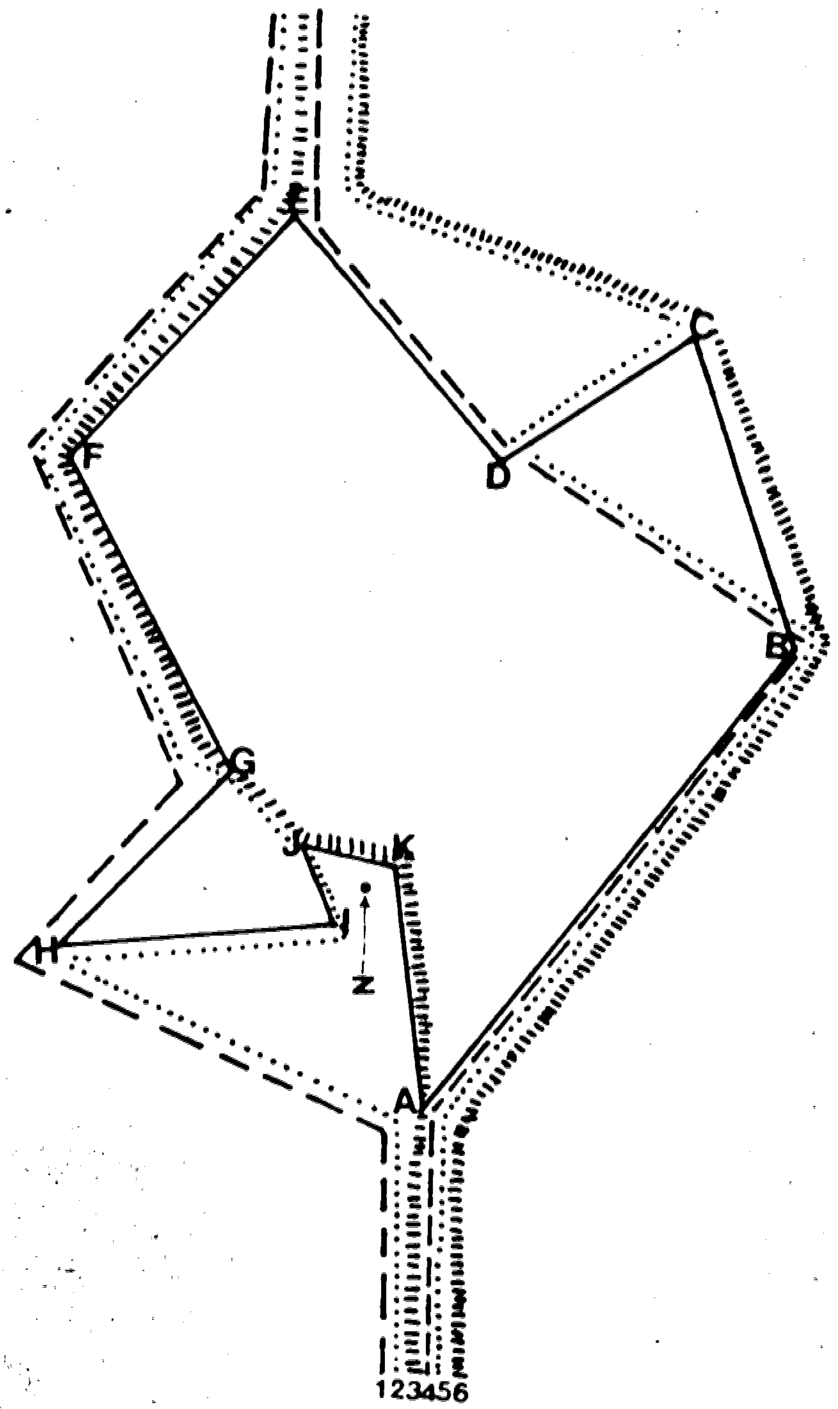
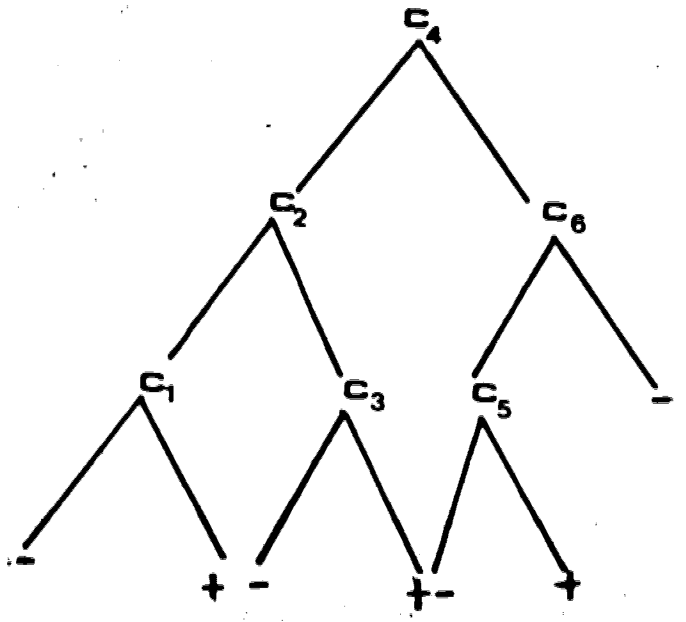


FIGURE III.3c

The Associated Completed Set of Monotone Chains

In the tree, the chains associated with the left subtree of any node (father chain) all contain smaller indices and therefore lie on the left of the father chain. A similar observation can be made about any right subtree. The leaves of the tree denote containment in (+), or exclusion from (-), the polygon.



Chain:	
C_1	AHGFE
C_2	AHIJGFE
C_3	AKJGFE
C_4	ABDE
C_5	ABDCE
C_6	ABCE

FIGURE III.4: Binary Search Tree of Monotone Chains

[Search procedure]

The search algorithm begins by checking if the y coordinate of a given test point Z=(x,y) lies within the extreme y-coordinate range (y_1^*, y_n^*) of the given plane. The processing of the point Z is terminated if Z is found to be outside the defined region. Otherwise testing commences with the root node of the search tree and proceeds downward toward a leaf of the tree, whereby the point-in-polygon problem becomes resolved.

Testing of a node at each step consists of determining if the point lies on the left or on the right of the chain associated with the node. It is interesting to observe that on termination of the search algorithm, a unique pair of consecutive chains C_i and C_{i+1} containing the test point are determined. Note that if the point Z is strictly on the left or right of the cómplete set of chains, then only one chain is identified.

Discrimination of a point Z against a monotone chain (i.e. determining if Z lies on the left or on the right of the chain) requires first the identification of the edges of the chain nearest to Z. Since the chain is monotone with respect to y-axis, binary search techniques can be applied to the ordered y-coordinates of the vertices of the chain. Once the nearest edge to Z is identified, discrimination of Z against the edge is straight-forward. Illustration of this discrimination of Z against a monotone chain is given in Figure III.2.

Analysis

[Preprocessing] The initial processing of the polygon requires the sorting of the y-coordinates of the vertices, which takes O(n log n) operations. Scanning the edges of the polygon in choosing monotone chains takes O(n) additional operations. Lastly, the construction of the complete set of monotone chains requires O(n log n) more operations. Hence, the preprocessing cost is at most O(n log n) (see Lee and Preparata[18] for additional details).

[Storage] Since at most n chains are required to form a complete set and since each chain can have at most n edges, the total storage requirements are bounded by $O(n^2)$. Lee and Preparata, however, describe a method for the storage of selected edges only, which reduces storage costs to O(n).

[Search] Suppose a complete set of chains C is given by $C=(C_1,C_2,\ldots,C_k)$. Using the binary search tree for the complete set of chains, at most log k chains need be processed to determine a unique pair of consecutive chains C_i and C_{i+1} (with $i < k$) which contain the given point Z. Let the projections of the vertices of a chain onto the y-axis form the sequence $y(V_1),y(V_2),\ldots,y(V_m)$. We can apply binary search techniques on the ordered set of vertices in order to locate that edge of the chain C_i that borders Z. This takes O(log m) time. A fixed number of arithmetic operations and a single comparison are sufficient to determine on which side of the edge the point lies. Thus, the entire search process requires O(log k . log m) operations. However, from the earlier discussion it is clear that there are at most n-1 chains in C. In addition, each chain in the complete set C has at most n vertices. Therefore, at most $O(\log^2 n)$ operations are required by the search technique to solve the point-in-polygon problem.

Note: For convex polygons, the complete set of monotone chains consists merely of two chains. The cost of searching in this case reduces to O(log n) operations.

3.4 Triangle Method

An efficient O(n log n) algorithm for partitioning a polygon P into a triangular subdivision is described by Garey et al[12]. Their algorithm first constructs for the polygon an associated regular graph, and then an associated ordered sequence of monotone chains (see section 3.3). Once this structure is available, the triangulation of the point set which determines any two consecutive chains proceeds as follows:

The triangulation algorithm is given as follows:

```
INPUT:
    Let the vertices of the polygon formed by two
    consecutive chains have k vertices:

1.  Sort the vertices into non-decreasing order of
    y-coordinates and denote them by $V_1^*$, $V_2^*$,...,$V_k^*$

2.  Place the two vertices $V_1^*$ and $V_2^*$ into a stack
    set $j = 2$

3.  (Comment:  Denote the current contents of the stack by
    $W_1$, $W_2$,...,$W_i$)
    set $j = j + 1$
    set $V^* = V_j^*$

4.  If $V^*$ is adjacent to $W_1$, but not to $W_i$
    then
        (Comment:  $V^*$ lies on the chain opposite from the
        chain associated with the stack)
        draw diagonals ($V^*$,$W_2$) ($V^*$,$W_3$),.....,($V^*$,$W_i$)
        change the stack to $W_j$, $V^*$
        go to step 3
```

```
5.   If $V^*$ is adjacent to $W_i$ but not to $W_1$
     then
          (Comment: $V^*$ lies on the same chain as the stack)
          Repeat until $i=1$ or internal angle at $W_i$ is at
          least $180^o$
          add diagonal $(V^*,W_{i-1})$
          delete $W_i$ from stack
          set $i = i-1$
     end of repeat.
     Add $V^*$ to the top of the stack
          go to step 3
6.   If $V^*$ is adjacent to $W_1$ and $W_i$
     then add diagonals $(V^*,W_2),\ldots\ldots,(V^*,W_{i-1})$
          stop.
```

The application of the triangulation algorithm on two chains is illustrated in Figure III.5.

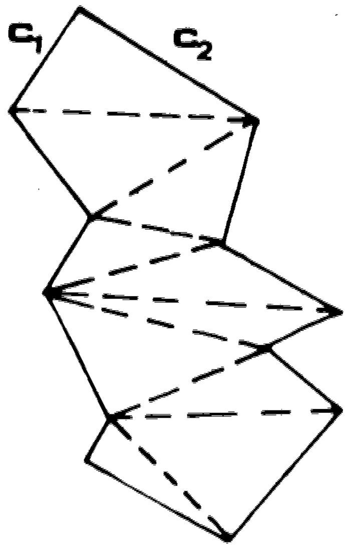


FIGURE III.5: Triangulation of Chains C_1 and C_2

Lipton and Tarjan[19] use this triangular structure to describe efficient algorithms for solving a variety of classical planar problems, including the point-in-polygon problem. A simpler and more practical adaptation of their point-in-polygon algorithm is given by Kirkpatrick[16] and it is this work that is outlined in this section.

Kirkpatrick's algorithm requires that the polygon P be enclosed within some triangle T. The polygon P as well as the region exterior to P and interior T is refined to a triangular subdivision using again the methods of Garey et al[12]. An example of a polygon and its triangular subdivision are given in Figures III.6a to III.6c. The triangles composing the subdivision of T are labelled $T^{(0)} = \{T_i^{(0)},\ i=1,2,\ldots,n_0\}$.

Given an arbitrary point Z, the problem now is to determine which triangle $T_i^{(0)}$, if any, in the triangular subdivision of T contains Z. If $Z \in T_i^{(0)}$ and if $T_i^{(0)}$ is interior to P, then Z is contained in P; otherwise it is not.

The problem of identifying that triangle $T_i^{(0)}$, if any, which contains Z is simplified by obtaining a new triangular subdivision $T^{(1)} = \{T_i^{(0)},\ i=1,\ 2,\ldots\ldots,n_1\}$, of T where $n_1 < n_0$. This new subdivision $T^{(1)}$ is obtained from $T^{(0)}$ as follows: Let V be an internal vertex of $T^{(0)}$. Then V is incident with some number of triangles of $T^{(0)}$, the union of which is called the neighbourhood of V. The strategy is to select a maximum number of such mutually disjoint (except

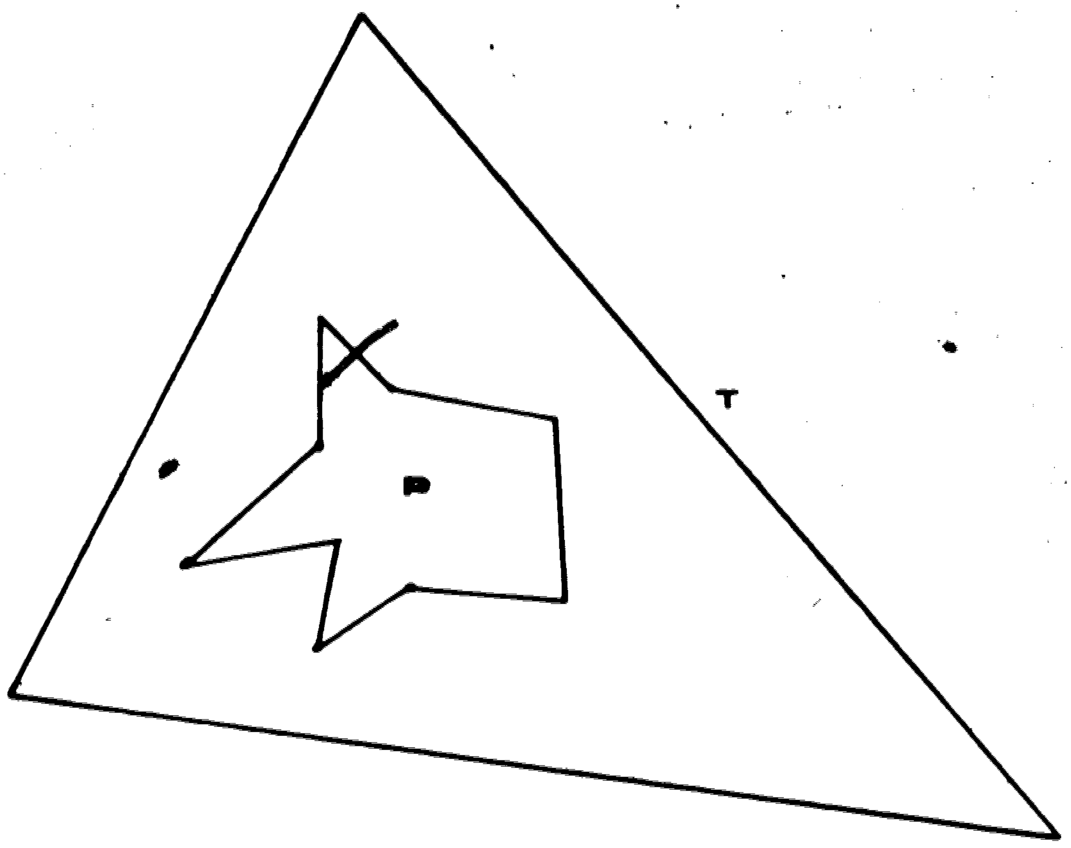


FIGURE III.6a: Polygon Enclosed in a Triangular Region

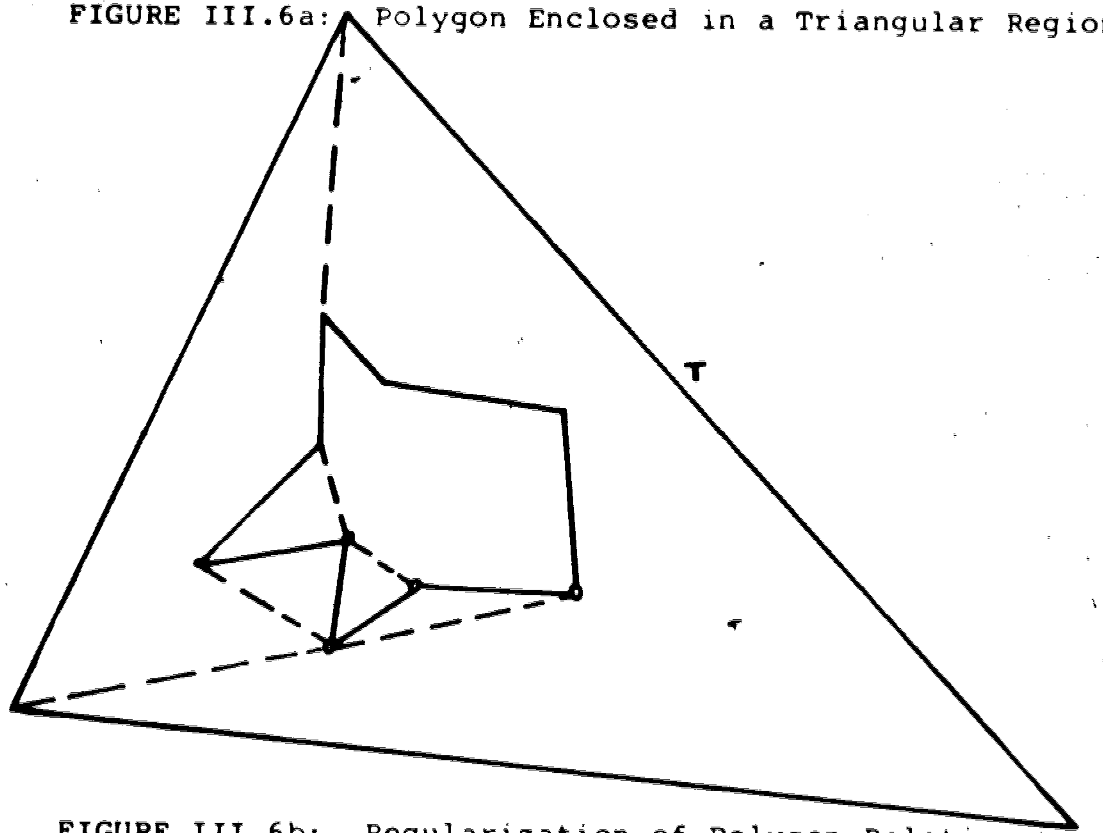


FIGURE III.6b: Regularization of Polygon Relative to a Triangle

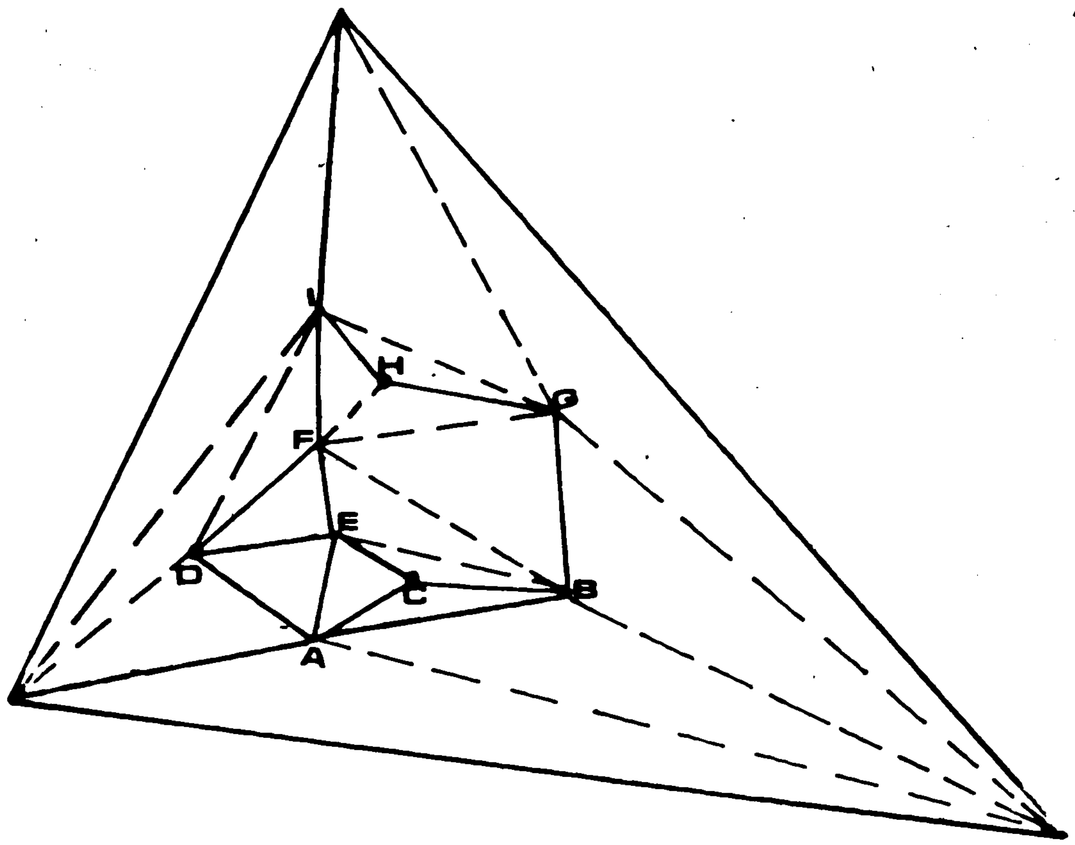


FIGURE III.6c: Triangular Subdivision

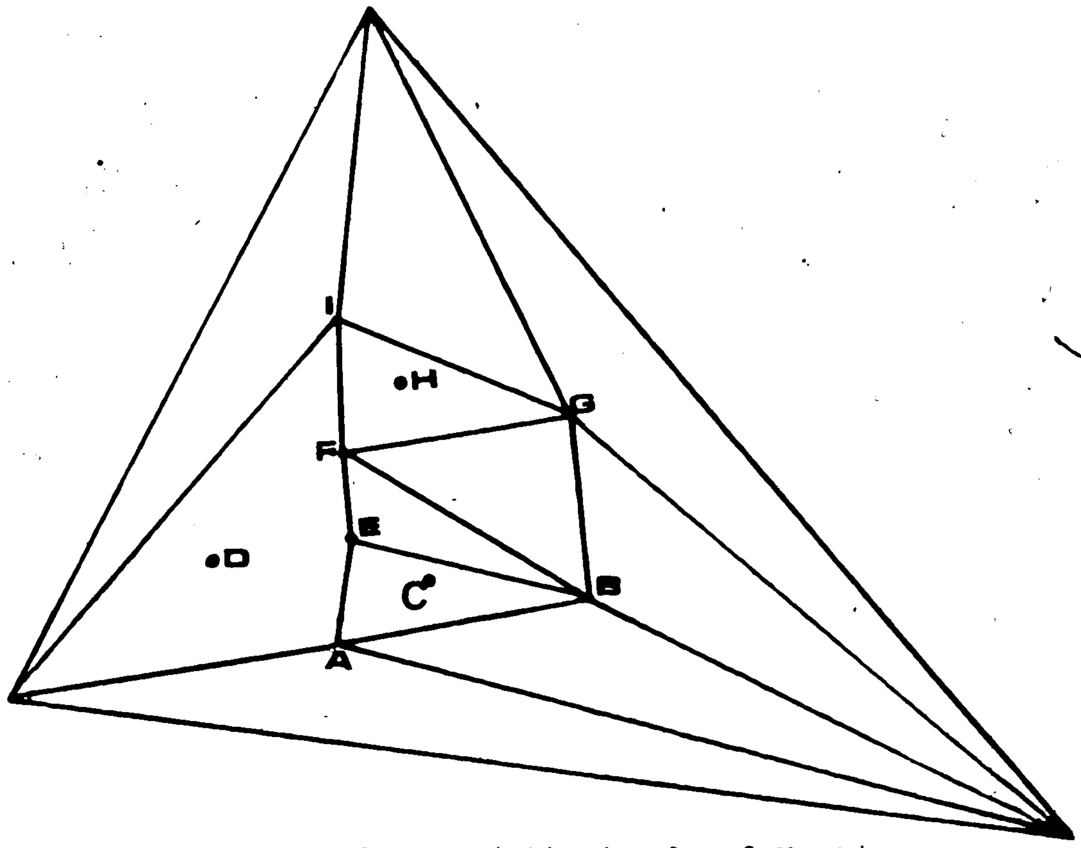


FIGURE III.6d: Neighborhoods of Vertices

possibly at boundaries) neighbourhoods in $T^{(0)}$. Such neighbourhoods are illustrated in Figure III.6d. For each neighbourhood, the associated vertex and its incident edges are dropped and the neighbourhood is re-triangularized. This new triangulation of T is called $T^{(1)} = \{T_i^{(1)}, i = 1, 2, .., n_1\}$, where n_1 must be less than n_0 since each neighbourhood now contains fewer triangles in its partition (see Figure III.6e).

This procedure is continued to produce $T^{(2)}$ from $T^{(1)}$, and so on until only the circumscribing triangle T remains. See Figures III.6f to III.6i.

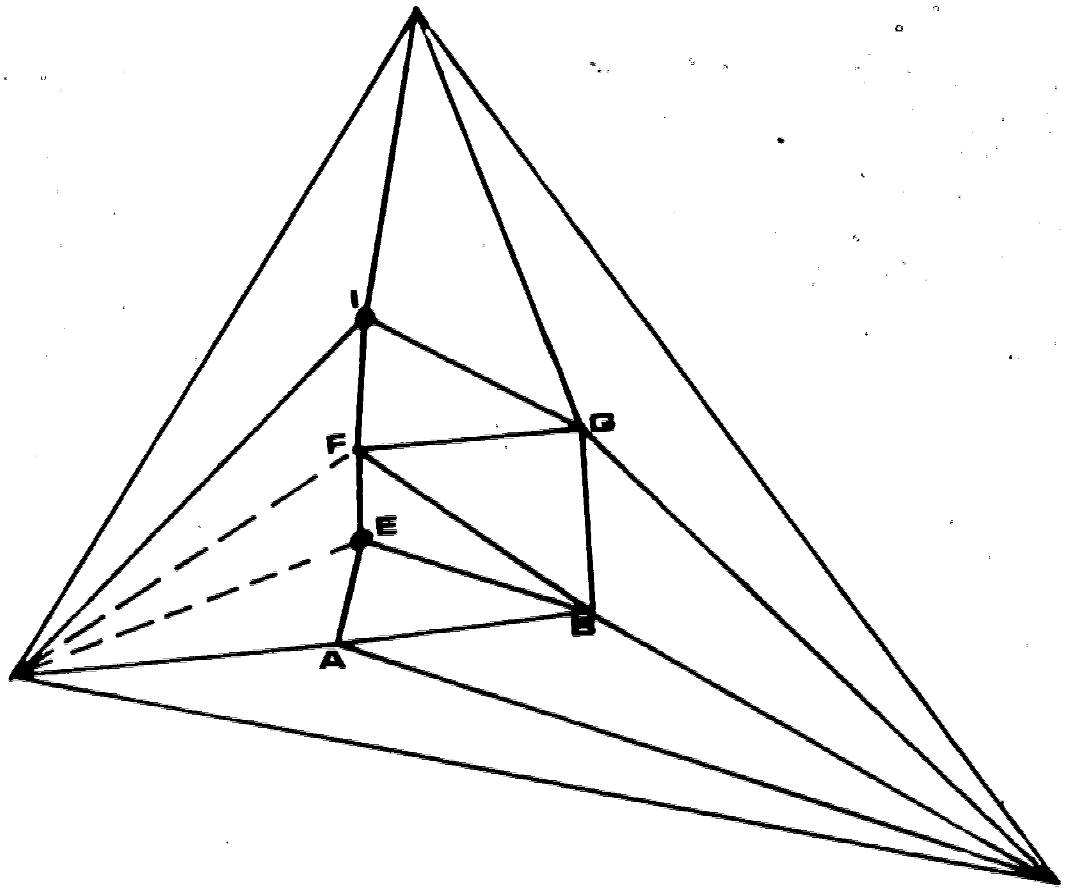


FIGURE III.6e Triangular Subdivision $T^{(1)}$

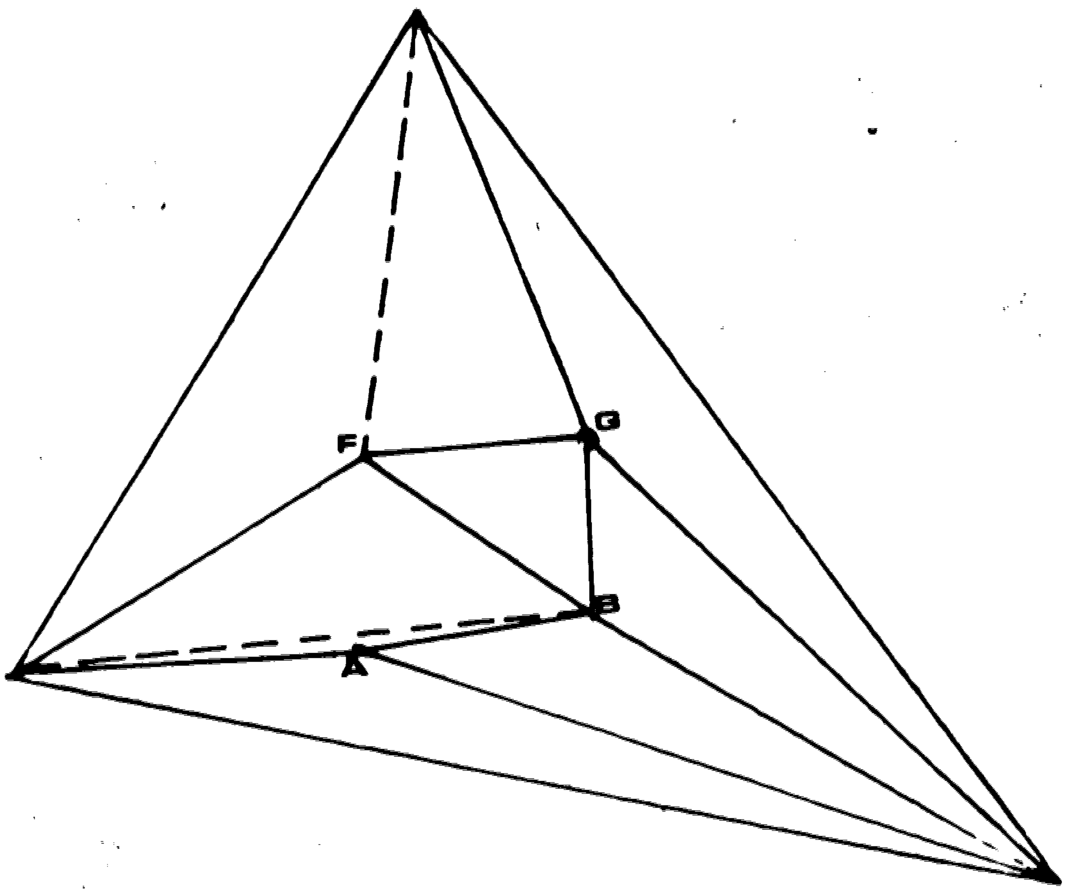


FIGURE III.6f: Triangular Subdivision $T^{(2)}$

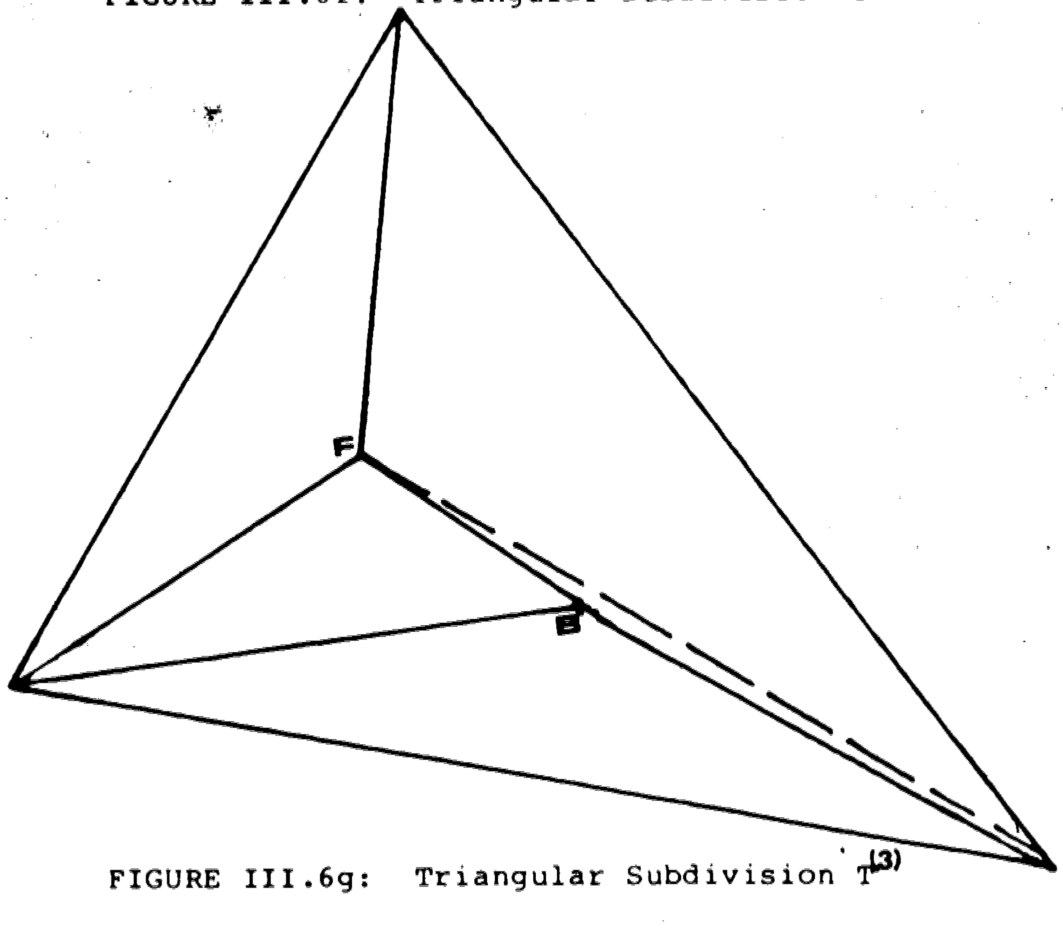


FIGURE III.6g: Triangular Subdivision $T^{(3)}$

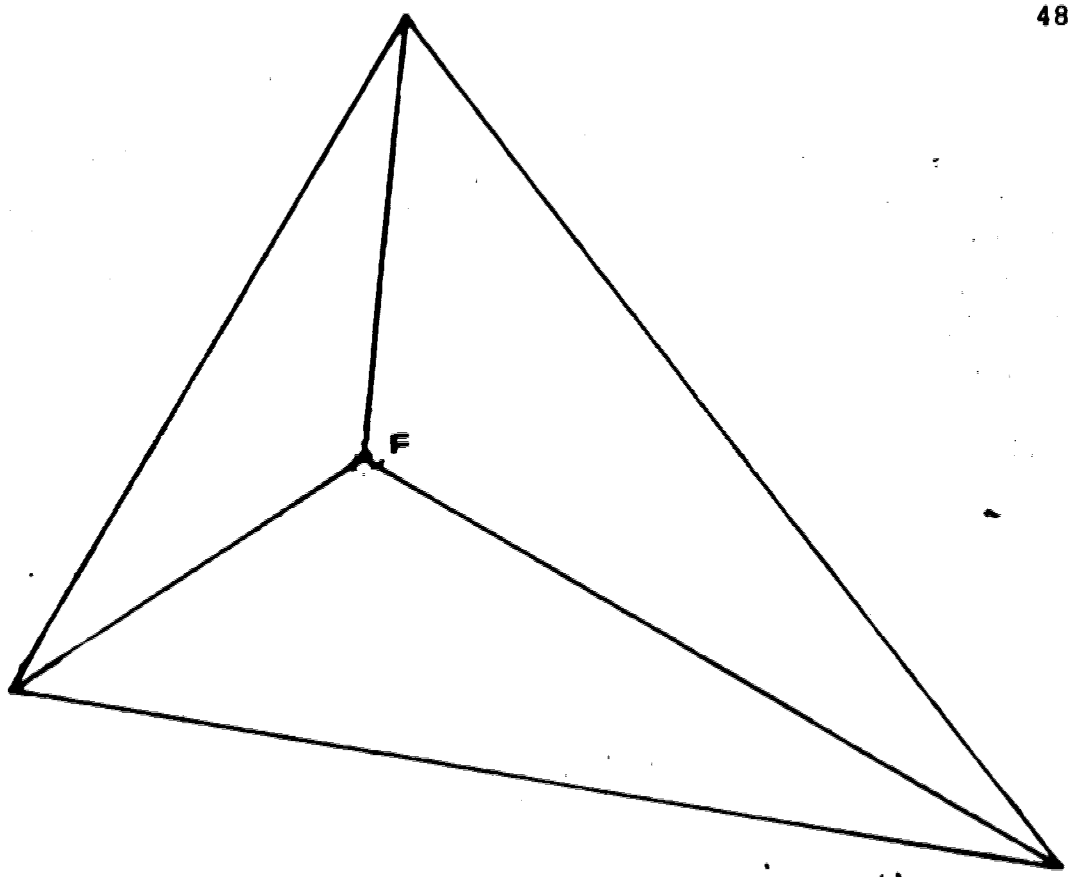


FIGURE III.6h: Triangular Subdivision $T^{(4)}$

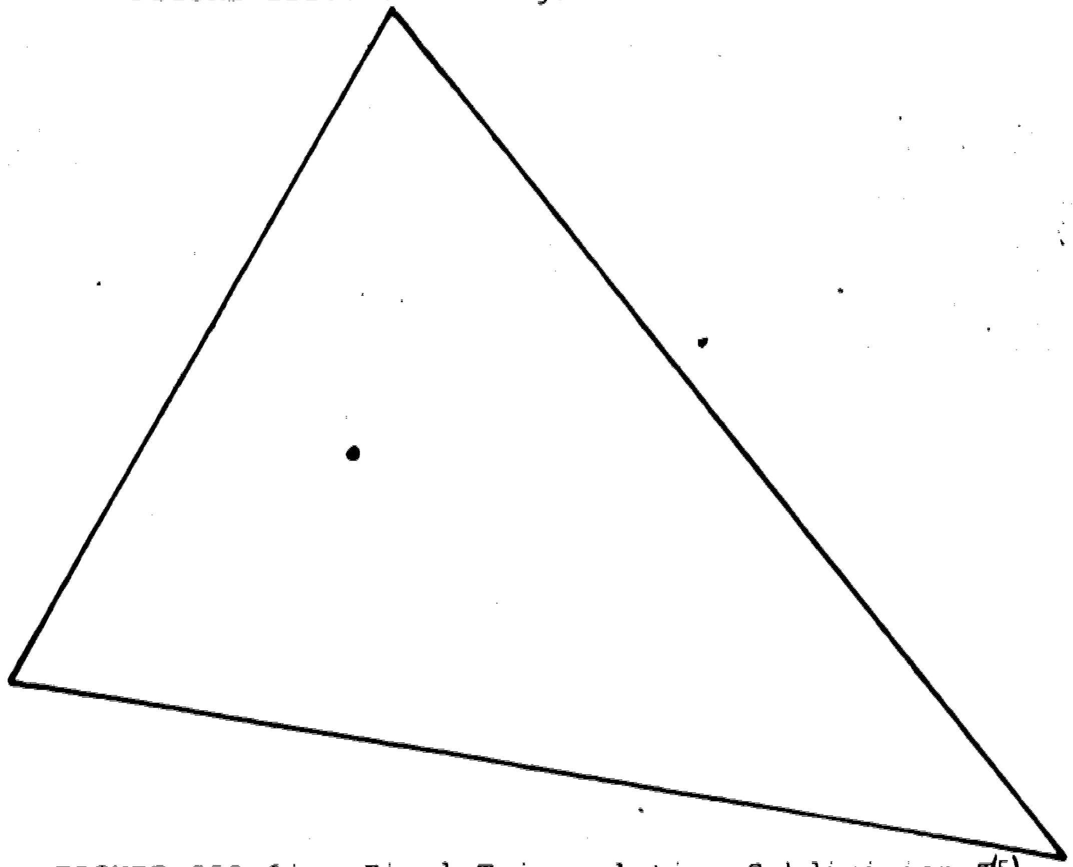


FIGURE III.6i: Final Triangulation Subdivision $T^{(5)}$

The set of triangular subdivisions may be placed in a hierarchical tree structure as follows:

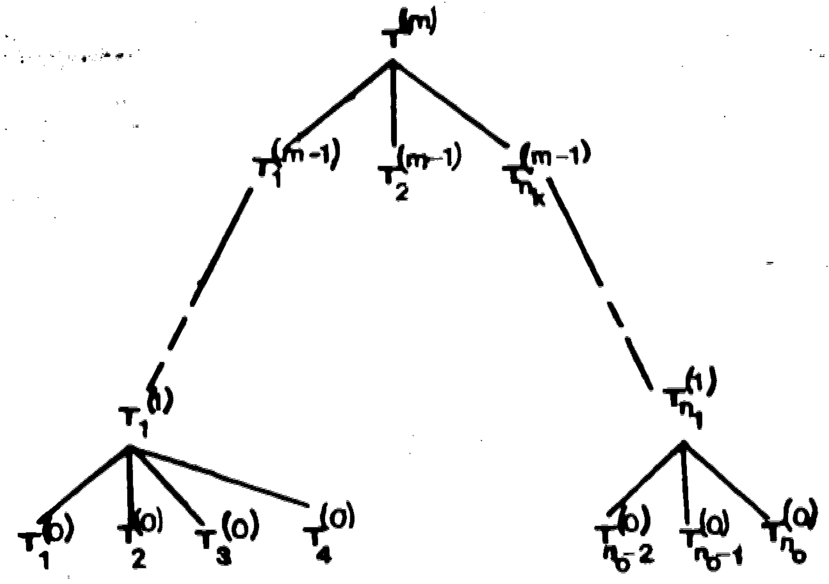


FIGURE III.7: HIERARCHICAL TREE STRUCTURE

In the tree, $T_i^{(k)}$ is a son of $T_j^{(k+1)}$ whenever their intersection is not null. That is, a son $T_i^{(k)}$ of $T_j^{(k+1)}$ need not always be totally contained in $T_j^{(k+1)}$

Analysis

[Search]

It is essential to show that the height of the above tree structure ensures logarithmic search complexity. Kirkpatrick shows that there exists positive constants c_1 and c_2 with $c_1, c_2 > 1$, such that $n_{k+1} < n_k/c_1$ and the number of sons at each node is bounded by c_2. Thus, the height m of the tree is bounded above by $\log n_o/\log c_1$. Therefore, at most, $c_2 . \log n_o/\log c_1$ triangles need be examined in order to determine that triangle $T_i^{(0)}$ which contains the point Z. Since $n_o \leq c_3 n$, where c_3 is constant and n is the number of vertices in the polygon P, it follows that Kirkpatrick's search algorithm requires O(log n) operations.

[Preprocessing]

The preprocessing costs of Kirkpatrick's algorithm are as follows: Conversion of an n-vertex polygon into a regular graph requires O(n log n) operation. The triangulation process and the hierarchical tree construction at the k'th step requires $O(n_k)$ operations. Thus, the cost of the entire triangulation and tree construction is

$$\sum_{k=0}^{m} O(n_k) \leq O(n_o) \ . \ \sum_{i=0}^{m} 1/c_1^i = O(n_o)$$

Therefore, the total number of operations required by the preprocessing phase is O(n log n).

[Storage]

The space required of the triangle method consists of storing $n/c_1{}^i$ triangles at each level of the tree structure. Since the height of the tree is bounded by log n_o/log c_1 = O(log n), requirements are bounded by

$$\sum_{i=o}^{\log n} n_o/c_1{}^i = O(n).$$

3.5 Rectangle Method

The description of this method requires some preliminary definitions.

DEF'N: A section of a polygon is defined to be a set of consecutive segments of the polygon.

DEF'N: A section which is monotonic with respect to both the x and y coordinates is called a simple section.

DEF'N: A maximal simple section is called a basic section.

By partitioning a polygon into the union of basic sections (this requires determining those vertices which are local extremes in either the x or y coordinates), the processing costs of the line-crossing method (see section

2.5) can be significantly reduced. Let the line passing through a given test point and parallel to the x-axis be called the test line. For the line-crossing method, it is required to determine the number of intersection points of the test line with the boundary (with the basic sections) of the polygon and lying left (or right) of the test point. Testing to determine if the test line intersects a basic section (we shall call this intersection test) requires simply the comparison of the y-coordinates of the section. Of course, for those basic sections where intersection does occur, additional processing (we shall call this the orientation test) of the segments which compose the section is required to determine if the intersection point lies left or right of the test point. This enhancement to the line-crossing method is the essence of the work done by Loomis[21], and in many applications can result in significant improvements.

The orientation test is simplified by first comparing the x-coordinate of the point with the extreme x-coordinates of the basic section. This, together with the intersection test, is equivalent to determining if the test point lies in the smallest rectangle encompassing the basic section. Only if the test point lies in this rectangle (called the section rectangle) does it become necessary to process the segments which compose the section. A geometric illustration of basic section rectangles $R_i^{(o)}$, i=1,2,.....,m for a polygon is given in Figure III.8a. As can be seen, the union of all the basic section rectangles of a polygon serves as an annulus approximation to the boundary of the polygon.

The basic section rectangles can be combined pairwise consecutively to yield a cruder annulus approximation as follows: Let

$$k = \lfloor \log_2 m \rfloor + 1$$

and let $R_i^{(o)}$, $(i=m+1,...,2^k)$ be a null rectangle. Define

$$R_i^{(1)} = R_{2i-1}^{(o)} \cup R_{2i}^{(o)}, \quad (i=1,2,...,2^{k-1})$$

as the smallest rectangle which encloses $R_{2i-1}^{(o)}$ and $R_{2i}^{(o)}$. Thus, $R_i^{(1)}$ is the smallest rectangle which encloses the basic sections contained in $R_{2i-1}^{(o)}$ and $R_{2i}^{(o)}$. These enclosing rectangles $R_i^{(1)}$ for the given polygon are illustrated in Figure III.8b.

The line-crossing method can be enhanced by applying it to this new structure as follows. The section rectangles $R_i^{(1)}$, $i=1,....,2^{k-1}$ are scanned consecutively for intersections with the test line. If intersection with $R_i^{(1)}$ does not occur, the algorithm proceeds directly to the next section rectangle $R_{i+1}^{(1)}$. Only in the case that intersection of the test line with $R_i^{(1)}$ does occur is processing of the basic section rectangles $R_{2i-1}^{(o)}$ and $R_{2i}^{(o)}$ required. This essentially can reduce the number of rectangles to be processed by approximately a factor of 2.

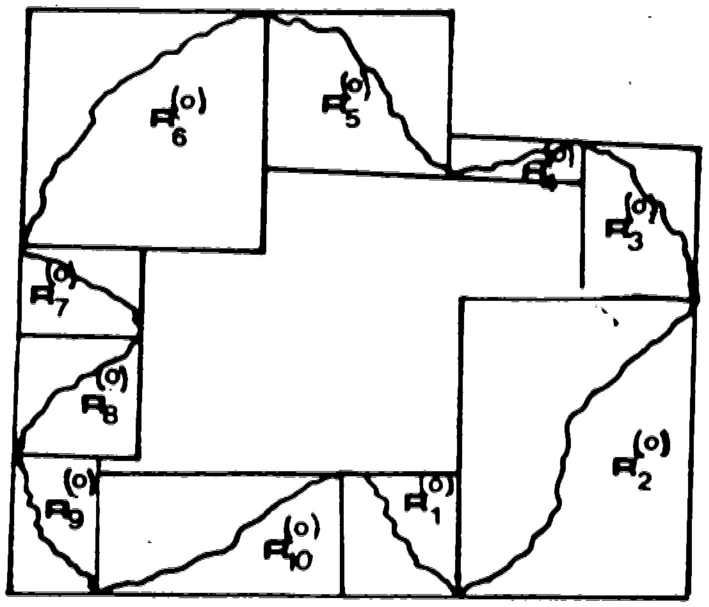


FIGURE III.8a: Basic Section Rectangles - Level 1

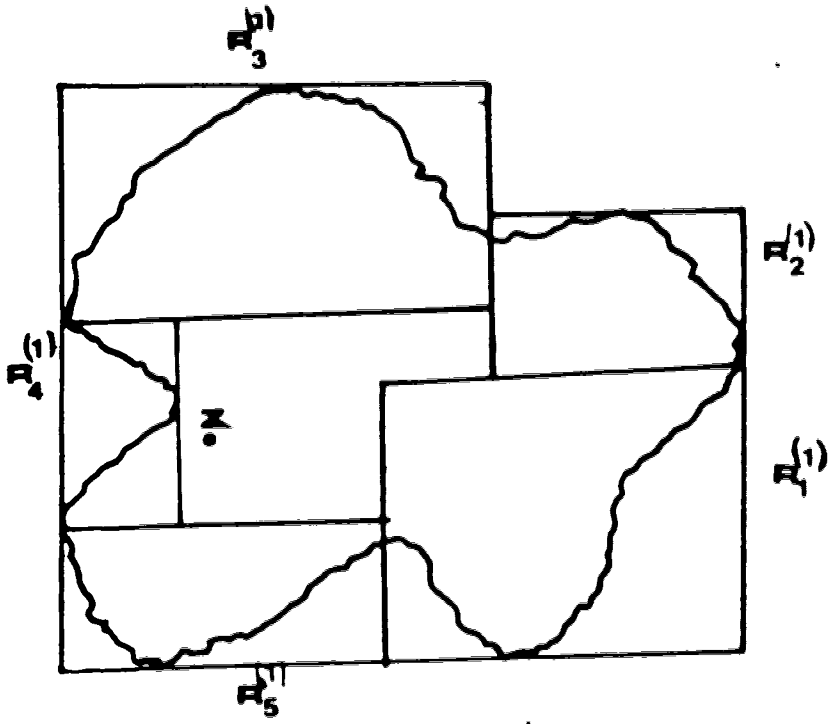


FIGURE III.8b: Section Rectangles - Level 2

What is described so far is the essence of Burton's[5] rectangle method. One additional construct is given. The pairwise merging of rectangles is again applied to the rectangles $R_i^{(1)}$ to produce the sequence of rectangles $R_i^{(2)}$, $i=1,\ldots,2^{k-2}$ and so on, until finally a section rectangle $R_1^{(k)}$ is determined which tightly encloses the polygon. (See Figures III.8c, III.8d and III.8e).

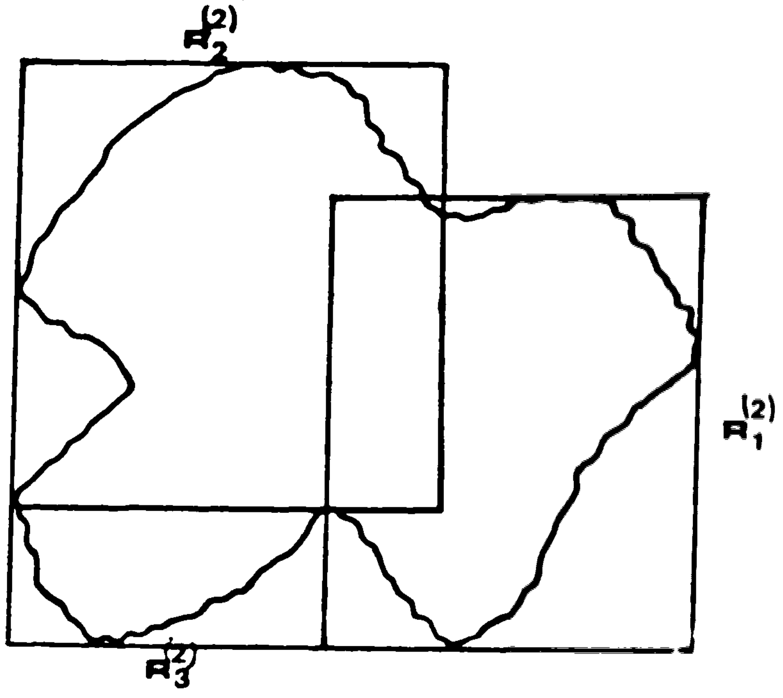


FIGURE III.8c: Section Rectangles - Level 3

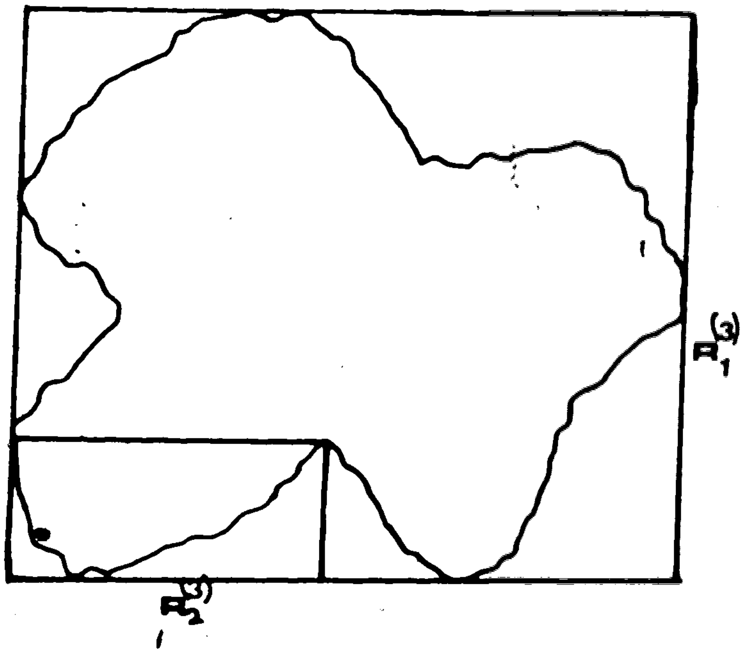


FIGURE III.8d: Section Rectangles - Level 4

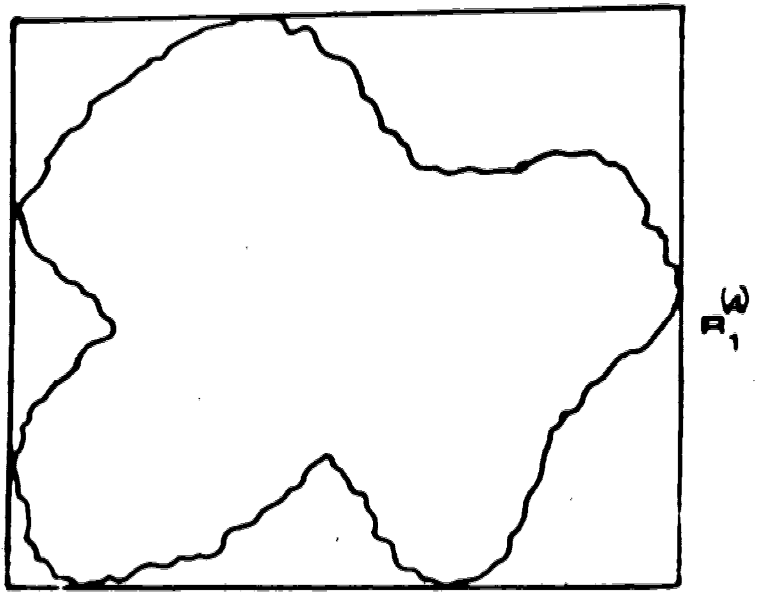


FIGURE III.8e: Final Enclosing Rectangle - Level 5

Pictorially, the hierarchical structure of section rectangles for a polygon is illustrated in Figure III.9 below.

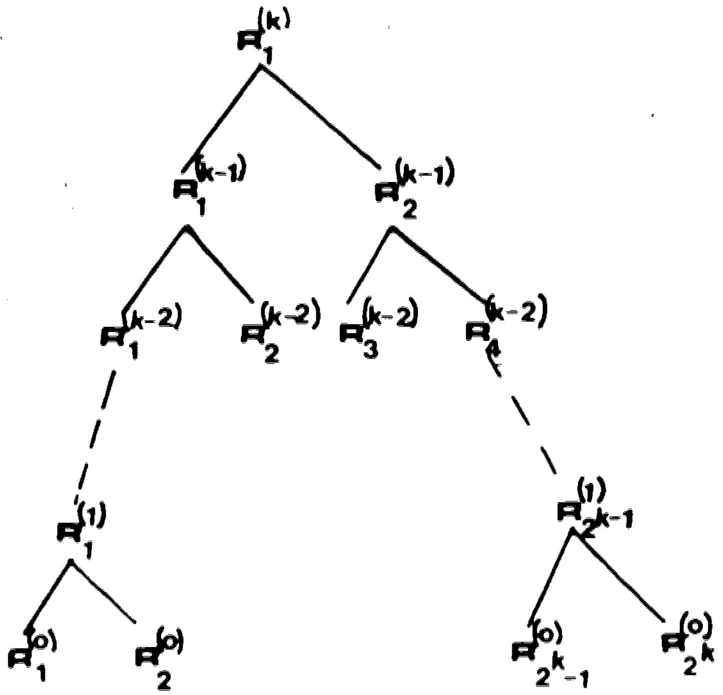


FIGURE III.9: Hierarchical Structure of Section Rectangles

Using this hierarchical tree of the section rectangles, Burton applies the line-crossing method as follows:

Beginning at the root of the hierarchy with $R_1^{(k)}$, it is determined if the test point Z is contained in $R_1^{(k)}$. If it is not, then the algorithm terminates; otherwise, the section rectangles $R_1^{(k-1)}$ and $R_2^{(k-1)}$ are processed, and so on down the hierarchy. To describe the processing of section rectangles, suppose in general that it is determined at some stage of the algorithm that $Z \subset R_i^{(j)}$, $j > 0$. It then becomes necessary to process each of the sons $R_{2i-1}^{(j-1)}$ and $R_{2i}^{(j-1)}$ of $R_i^{(j)}$.

Consider only the section rectangle $R_{2i-1}^{(j-1)}$ since the processing of $R_{2i}^{(j-1)}$ is identical. If the test line lies above or below $R_{2i-1}^{(j-1)}$, it cannot intersect any of the polygonal segments describing it. Therefore, in this case no further processing of $R_{2i-1}^{(j-1)}$, or of its descendants, is required. If the test line intersects $R_{2i-1}^{(j-1)}$, then either $Z \subset R_{2i-1}^{(j-1)}$ or it is not. If $Z \subset R_{2i-1}^{(j-1)}$, it is necessary to continue down the hierarchy by processing its sons $R_{4i-3}^{(j-2)}$ and $R_{4i-2}^{(j-2)}$. In the remaining case when the test line intersects $R_{2i-1}^{(j-1)}$ and Z is not contained in $R_{2i-1}^{(j-1)}$ (see Figure III.10a below), then the parity of the

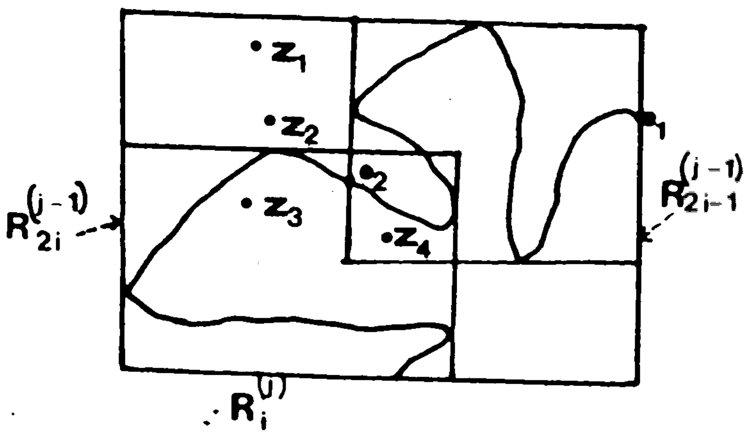


FIGURE III.10a: Containment of Point Z in More than one Section Rectangle

of the number of intersections of the test line with the polygonal segment describing $R_{2i-1}^{(j-1)}$ is determined as follows. Let e_1 and e_2 be the endpoints of the polygonal segment describing $R_{2i-1}^{(j-1)}$. Then the test line intersects the polygonal segment an odd number of times if Z lies between e_1 and e_2 (see Z_2 in Figure III.10a), and an even number of times otherwise (see Z_1 and Z_3 in Figure III.10a). Once the parity is determined no further processing of $R_{2i-1}^{(j-1)}$ or its descendants is required.

Summarizing, for $Z \subset R_i^{(j)}$, processing of the subtree for $R_i^{(j)}$

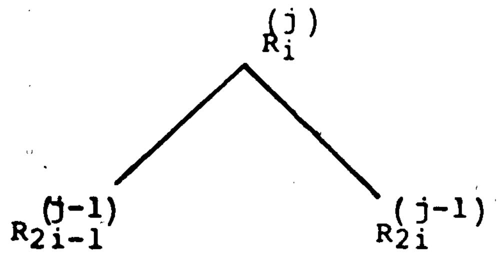


FIGURE III.10b: Subtree of Hierarchy Tree Structure

terminates at the next lower level if Z is not contained in either $R_{2i-1}^{(j-1)}$ or $R_{2i}^{(j-1)}$. If Z is contained in either $R_{2i-1}^{(j-1)}$ or $R_{2i}^{(j-1)}$ processing continues down the respective subtree. It is important to note that Z may be contained in both $R_{2i-1}^{(j-1)}$ and $R_{2i}^{(j-1)}$ (see Z_4 in Figure III.1ba)

The algorithm terminates early at level k-j, j<k whenever it is found the Z is not contained in any rectangle $R_i^{(k-j)}$, $i=1,2,\ldots,2^j$ at that level. The parity of the number of times the test line intersects the polygonal boundaries from the left then determines the solution of the point-in-polygon problem. Otherwise, Z is contained in one or more basic sections rectangles $R_i^{(o)}$ and additional processing of these rectangles is required.

The processing of a simple section rectangles is greatly simplified because the polygonal segment number which defines it is monotone with respect to y. Binary search techniques can therefore be used to discriminate quickly the point Z against the polygon segment. However, Burton suggests an even faster algorithm. The basic section is split into two simple sections, each with an equal number of vertices (within one). Each simple section is then tightly enclosed by a simple section rectangle. Therefore, as in the previous hierarchical structure for compound sections, the basic section rectangle tightly encloses the two simple section rectangles. The splitting strategy is next applied to simple sections, and so on until simple sections consisting of one polygonal edge (called primitive sections) only remains. These form the leaves of the total hierarchical tree of compound, basic, simple and primitive section rectangles. The point-in-polygon algorithm previously described can be applied directly to this total hierarchical tree. Only if the test point Z is contained in a primitive section, does it become necessary to discriminate Z against a polygonal segment.

Analysis

[Search]

Let $l = \lceil \log n \rceil$. Then total hierarchical tree contains at most 2^{l} leaves, corresponding to the primitive section rectangles. The total number of rectangles in the tree (nodes in the tree) is therefore bounded by $2^{l+1} \leq 4n$. Because processing a rectangle requires a fixed number of operations, in the worst case when the test point is contained in all the rectangles, the complexity of Burton's algorithm is O(n). Indeed, Figure III.11 provides an illustration of a polygon where Burton's method may require O(n) operations to solve the point-in- polygon problem.

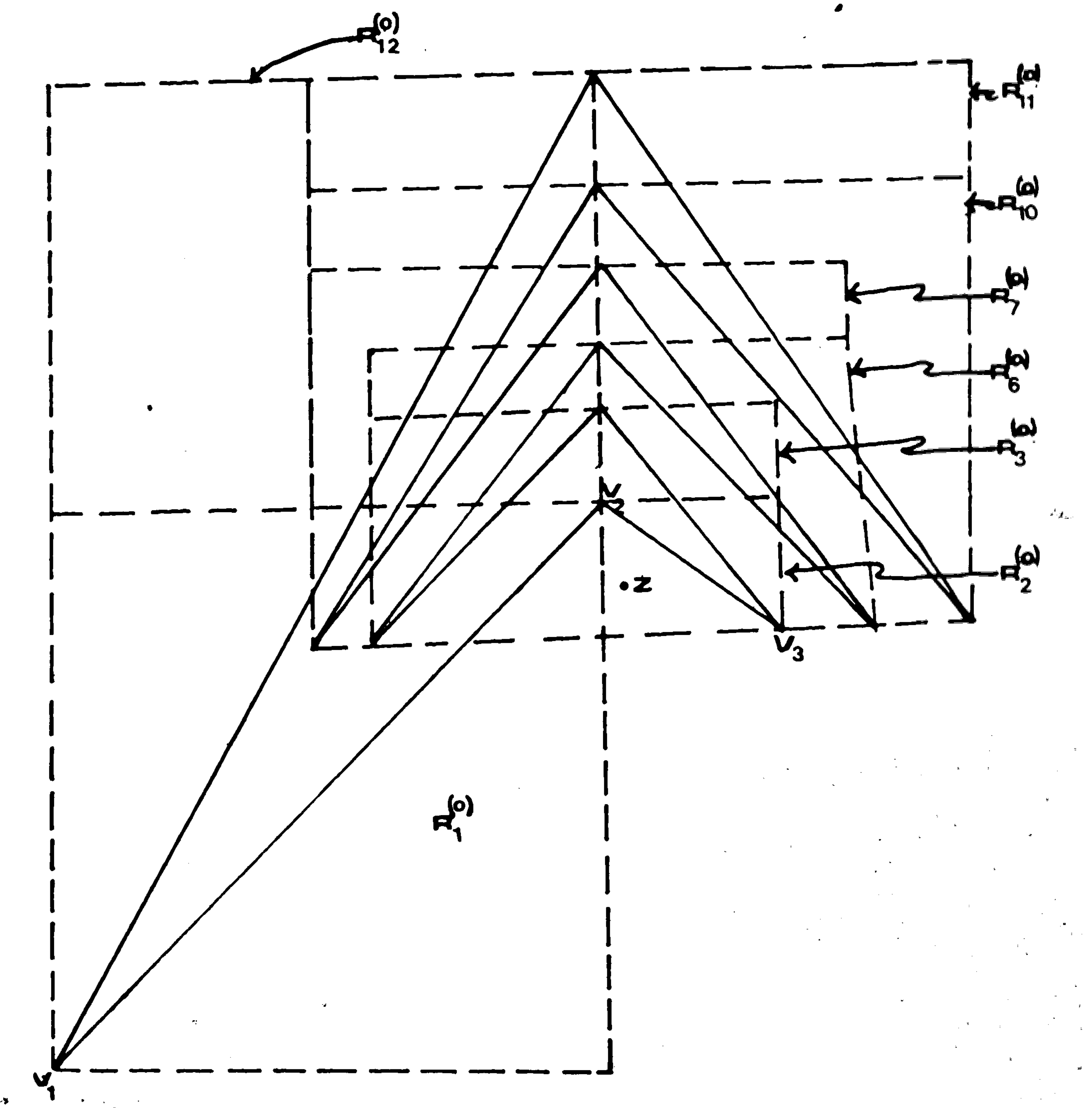


FIGURE III.11 A Worst Case Polygon (The Claw)

In practice, for "well-behaved" polygons, Burton's method performs significantly better. The algorithm terminates at that level of the hierarchical tree where none of the associated section rectangles contain the test point Z (see Figure III.8c). Furthermore, if Z is contained in at most one section rectangle at each level, the complexity of the algorithm reduces to O(log n). Indeed, because of the binary tree construct, it may be natural to erroneous assume this to be the case in general (see, for example, Fowler[10]).

[Preprocessing]

The total number of operations required to find all the turning points is O(n). Furthermore, each of O(n) section rectangles can be computed in fixed time. Hence, the total cost of preprocessing is of complexity O(n).

[Storage]

Since each of the O(n) section rectangles can be stored in a fixed number of words, the storage costs are of complexity O(n).

3.6 The Strip Method

The fundamental process common to these methods is the construction of a new polygon P* which approximates the original polygon P in the following sense. Given a tolerance or resolution e, the distance (usually Euclidean) of any point on the boundary of P to the boundary of P* is bounded by e. The polygonal approximation P* is to be selected so that it is simpler for processing purposes (fewer number of vertices), and in addition captures the character of the original polygon P. The matter of character capture is not discussed in depth here, because it has little bearing on the performance of point-in-polygon algorithms, except perhaps indirectly through its effect on the polygonal approximation P*.

One algorithm for constructing a polygonal approximation P* to a polygon $P = [V_1, V_2, \ldots\ldots, V_{n+1}]$ is given below.

Algorithm:

1. Record V_1 as a vertex of P*
   Set starter s = 1
2. Set destinator f = s+1
3. Draw a line segment L joining V_s and V_f
   Determine the maximum distance d of the points $V_s, V_{s+1}, \ldots\ldots, V_f$ from L.

```
4.  If $d < e$ and $f < n$
    then
        (comment: extend line segment)
        set $f = f+1$
        go to step 3
    else
        (comment: line segment is complete)
        record $V_{f-1}$ as a vertex of P*
        go to step 5
5.  If $d > e$
    then
        set $s = f-1$
        go to step 2
    else terminate.
```

The effect of the algorithm is illustrated by applying it to the polygon P shown in Figure III.12a. The polygon represents the boundary of a river basin in the province of Alberta obtained by means of vector digitization. It was produced from a map at a scale of 1:50,000 and at a resolution of approximately 50 meters. The polygon contains 1873 vertices, and is quite typical of those polygons which appear in many geographic database systems.

The polygonal approximation P* obtained by the algorithm at a tolerance e of 250 meters is shown in Figure III.12b. The polygon P* contains a total of 21 vertices, which is approximately one percent of the original polygon P.

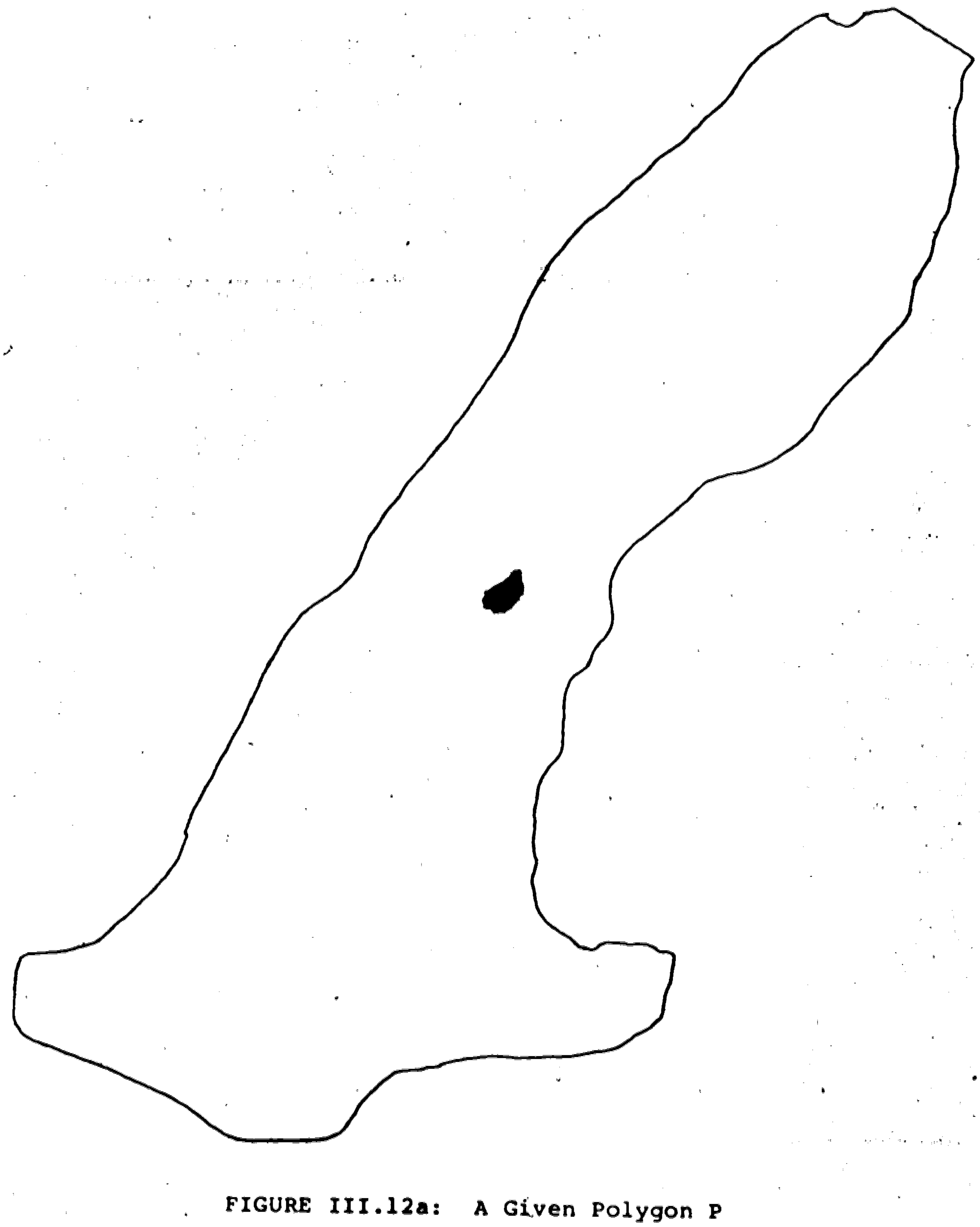


FIGURE III.12a: A Given Polygon P

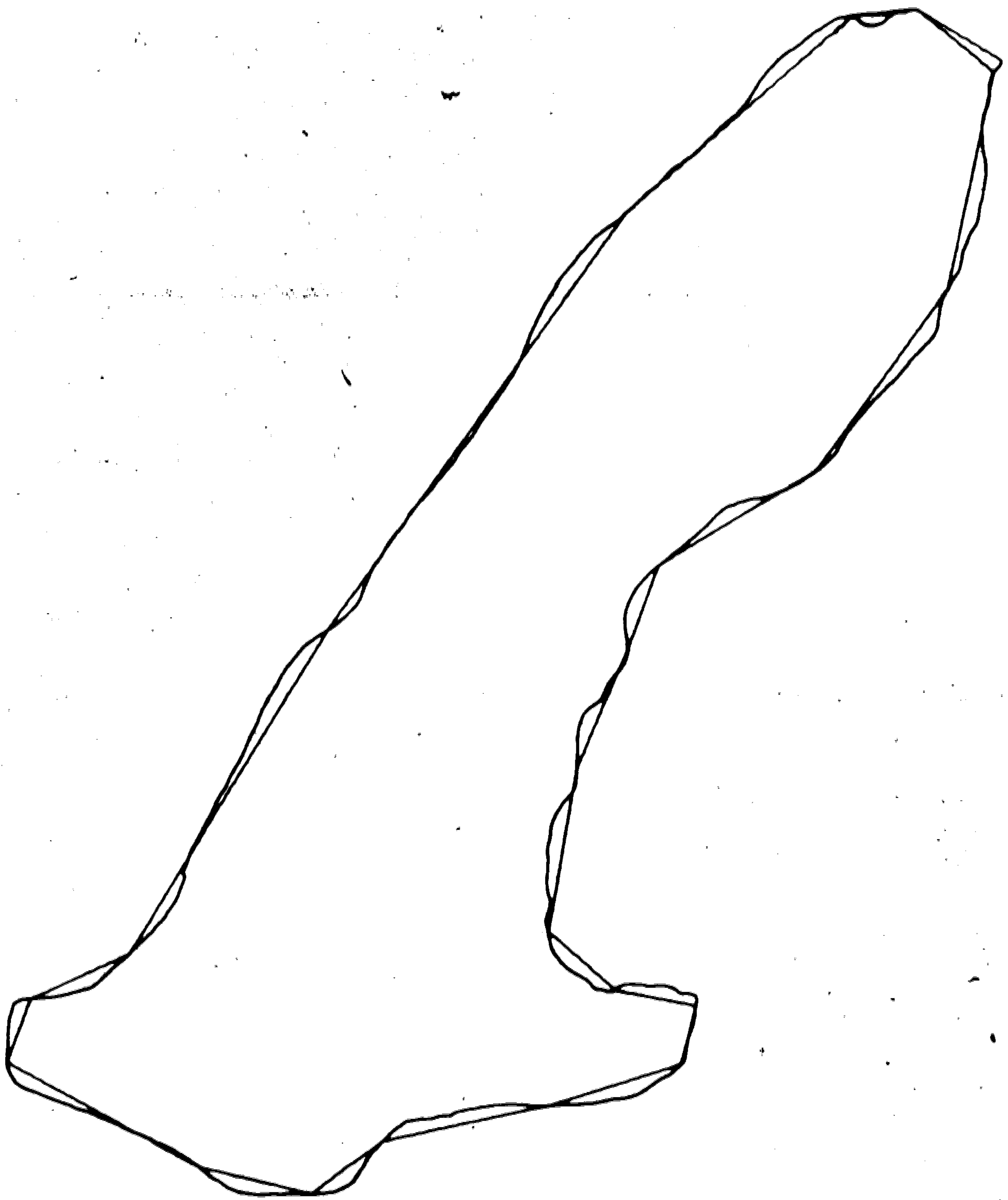


FIGURE III.12b: Polygonal Approximation of P

No claims are being made that the given algorithm is optimal in any sense. Indeed, the algorithm described by Douglas and Peucker[9] is likely to better capture the cartographic character of polygonal boundaries. An algorithm described by Ballard[3] is somewhat more efficient. And finally, the algorithm given by Kurozumi and Davis[17] yields computationally a more effective approximation in the sense that the polygon P* may have fewer vertices and yet meet the tolerance requirement. The given algorithm has the advantage of being simple and yet maintaining some similarity with all the other algorithms.

The polygonal approximation P* may be used to obtain an annulus approximation to P as follows. Move each of the line segments of P* outward by the distance e to obtain a new polygonal approximation P**. This polygon P** must enclose the polygon P. Similarly, move each of the line segments of P* inward by the distance e to obtain a polygon P*** which lies entirely inside P. The boundaries of P** and P*** describe an annulus which enclose the boundary of P. For the polygon P given in Figure III.12a, the annulus approximation is illustrated in Figure III.12c. In the figure, the annulus is partitioned into quadrangles (strips) by joining pairwise the vertices of P** and P***.

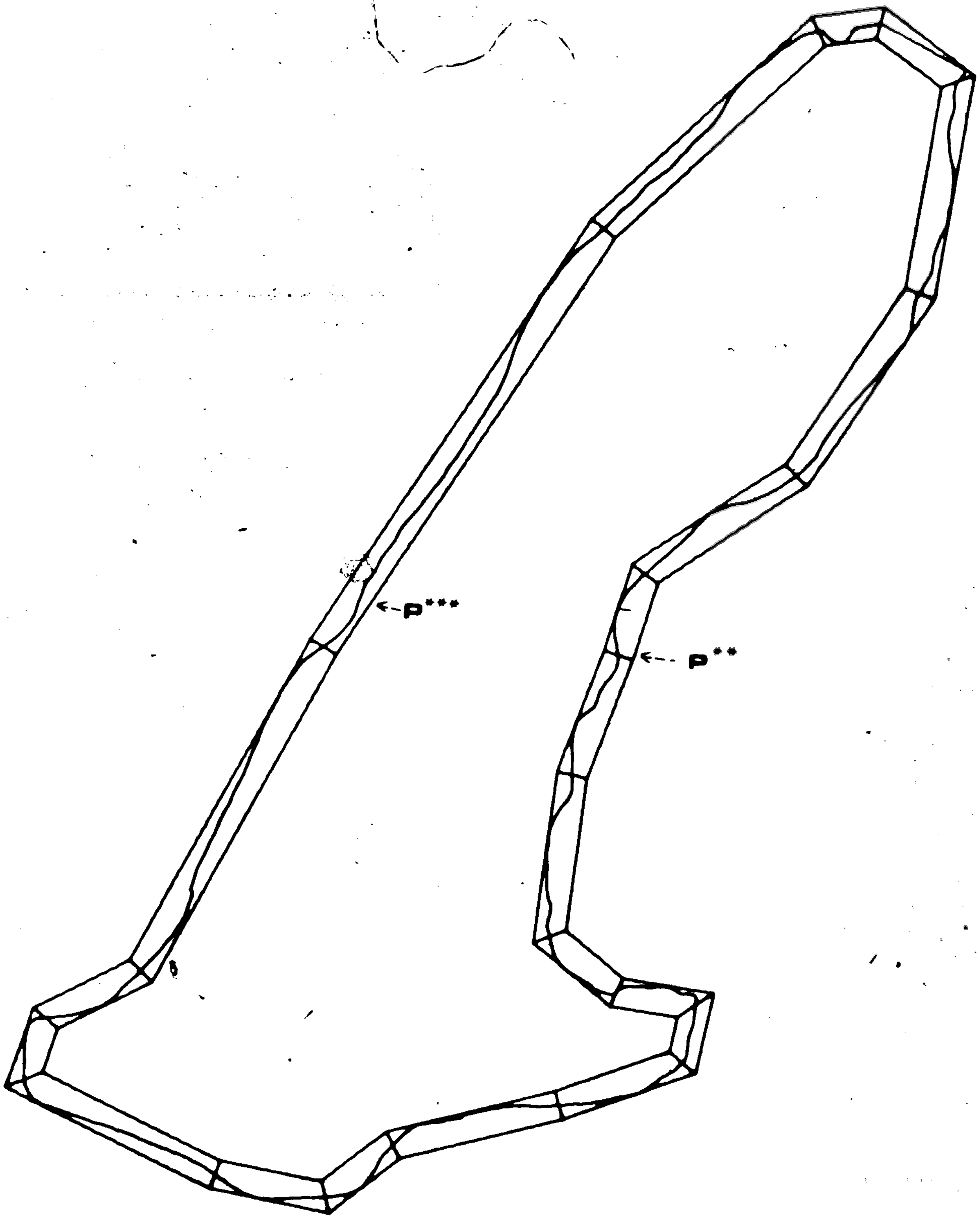


FIGURE III.12c: Annulus Approximation of Polygon P with Tolerance of 250 Meters

A cursory inspection of Figure III.12c suggests that some of the caricature of P is not captured by P* and consequently by P** and P***. An enhancement in caricature capture is obtained by reducing the tolerance e, and thereby improving the accuracy of the polygonal approximation P*. That is, there is a trade-off between obtaining an approximation P* with improved accuracy, and increasing the number of vertices required to represent it. For the example under consideration, an improved annulus approximation of P with a tolerance of 50 meters (and containing 57 vertices) is given in Figure III.12d.

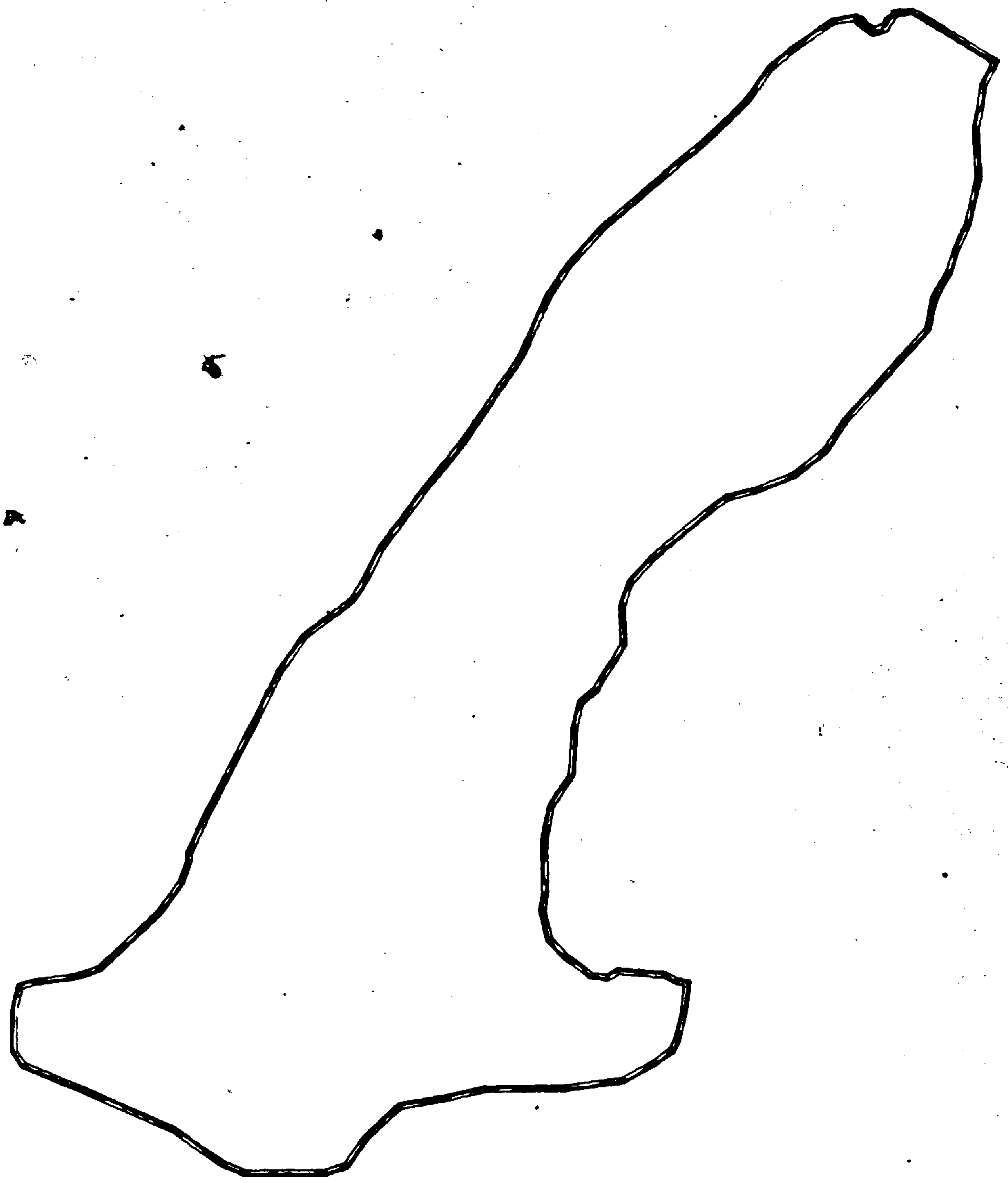


FIGURE III.12d: Annulus Approximation of Polygon P with Tolerance of 50 Meters

The question of tolerance selection is a non-trivial consideration. If the application does not predicate a choice, one possible strategy might be to remove from P those vertices which do not contribute significantly to its definition, for example, those vertices which are nearly colinear, or those which can be attributed to "noise". This can be achieved by obtaining successive approximation P* to P with ever decreasing tolerances until one is found where a further decrease would result in a dramatic increase in the number of vertices representing it.

For a polygon P obtained by digitization, this strategy will often lead to an approximating polygon P* which is very similar to P but which contains only a small portion of the number of vertices in P. For the example under consideration, in Figure III.12d see the annulus approximation of P with 50 meters and which contains only 57 vertices. For a summary of experimental results for many different tolerances which support the validity of the strategy, refer to Table II.

Table II - EXPERIMENTAL RESULTS (WITH A POLYGON OF 1873 POINTS)

Tolerance	1	2	3	4	5	6	7	8	9	10	15	20	25
Sig. Pts.	993	644	444	349	297	249	228	206	190	179	128	102	91
Timing (sec.)	0.883	0.841	0.866	0.894	0.934	0.968	1.019	1.081	1.110	1.172	1.397	1.543	1.543

Tolerance	30	35	40	45	50	55	60	65	70	75	80	85	90
Sig. Pts.	83	74	64	61	57	53	50	47	45	45	41	38	38
Timing (sec.)	1.757	2.084	2.114	2.367	2.495	2.530	2.647	2.792	2.839	2.881	3.143	3.153	3.143

Tolerance	95	100	125	150	175	200	225	250	500	750	1,000
Sig. Pts.	37	37	33	29	28	27	24	21	13	11	10
Timing (sec.)	3.155	3.139	3.876	4.398	4.447	4.751	5.496	6.130	10.028	12.522	12.909

Note: Tolerance is measured in meters.

The point-in-polygon algorithm is given as follows:

1. Apply line-crossing method to P^{***}
2. If $Z \subset P^{***}$

   Then terminate since $Z \subset P$

   else go to step 3
3. Apply line-crossing method to P^{**}
4. If $Z \not\subset P^{**}$

   then terminate since $Z \not\subset P$

   else go to step 5
5. (Comment: Z is contained in annulus)

   Apply line-crossing method to quadrangles to determine one (call it Q) which contains Z
6. Apply line-crossing method to the line segments of P contained in Q to determine whether or not $Z \subset P$.

Let m be the number of vertices in P^* and let l be the maximum number of line segments of P in any quadrangle. Then steps 1, 3 and 5 each requires $O(m)$ operations, whereas step 6 requires $O(l)$. Thus, in the worst case when the point Z lies in the annulus, the algorithm requires $O(m) + O(l)$ operations, where $m\,l \geq n$. For well-behaved polygons, however, Z likely lies outside the annulus and steps 5 and 6 of the algorithm are not executed. The cost is therefore $O(m)$ operations for the majority of points tested.

Note that the algorithm may be improved by combining steps 1 through 5. This modification however detracts from its readability, and does not effect the asymptotic cost estimates given above.

The problem of selecting an appropriate tolerance is solved by Ballard[3] by means of a data structure, a strip tree which provides a polygonal approximation to P for any required resolution e. A recursive algorithm for constructing this strip tree is given below.

<u>Algorithm</u>: Strip—tree $(V_0,V_1....V_i)$

(Comment: Let $[V_0,V_1,....V_i]$ denote a polygonal segment)

1. Draw a line L through V_0 and V_i.
2. Find the smallest rectangle with sides parallel to L which just cover V_0, V_1,....,V_i.

   (Comment: This rectangle is called the strip for the segment $[V_0,V_1,....,V_i]$).
3. Compute the width e of the strip as the euclidean distance between the two sides of the rectangle which are parallel to L.
4. If $i < 2$

   then return

   else go to step 5.
5. Find a vertex V_k which lies on one of the two sides of the rectangle that are parallel to L.

```
6.  (Comment:    Construct strips for segments
    [$V_0,V_1,\ldots,V_k$] and [$V_k, V_{k+1},\ldots,V_i$])
        subtree ($V_0,V_1,\ldots,V_k$)
        subtree ($V_k,V_{k+1},\ldots,V_i$)
        return.
```

A strip tree for a polygon with vertices $V_1,V_2,\ldots,V_n$ can be built by invoking the algorithm for the two segments ($V_1,V_2,\ldots,V_k$) and ($V_k,V_{k+1},\ldots V_n,V_1$) where V_k is selected to be approximately "opposite" of V_1 on P. The root strip of the strip tree for the polygon is defined to the smallest rectangle enclosing the strips for ($V_1,V_2,\ldots,V_k$) and ($V_k,\ldots,V_n$). Figures III.13b and III.13c illustrates the data structure for a strip tree of a typical polygonal segment in Figure III.13a.

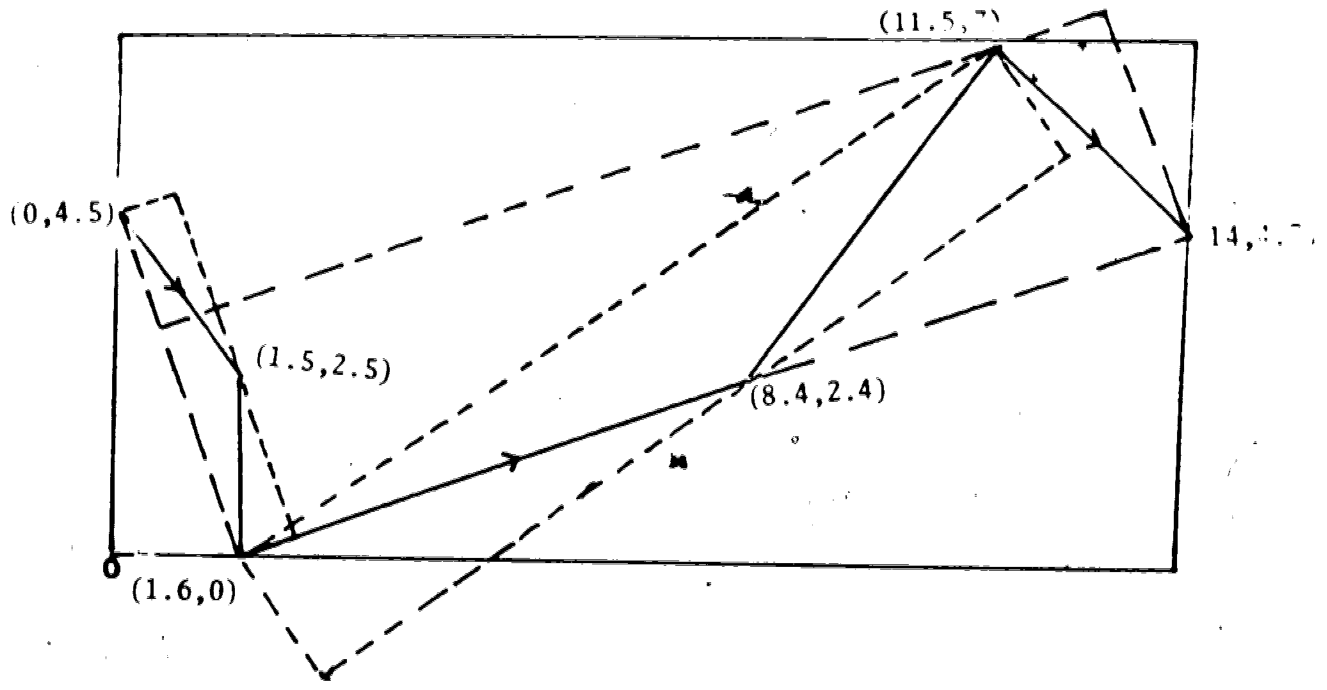


FIGURE III.13a: A Set of Strips

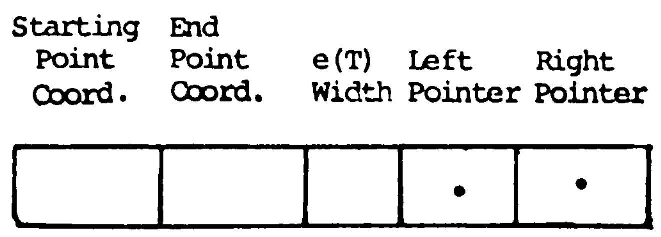


FIGURE III.13b: Format of a Node of a Strip

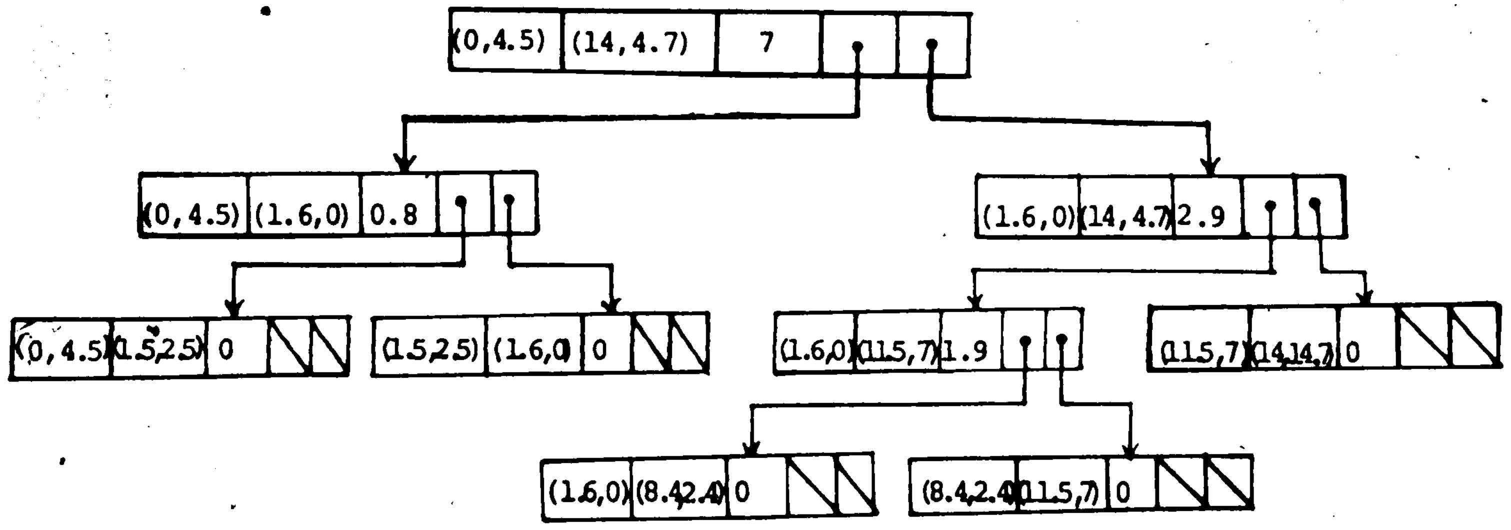


FIGURE III.13c: Data Structure for a Strip Tree

The polygon P can be displayed at any resolution e* by the following recursive algorithm described by Ballard.

```
Algorithm:  Display segment (T,e*)
    (Comment:  The segment with root node T is displayed
    with resolution at most e*)
    If e(T) < e*
    then display strip (T)
    return
    else display segment (left son (T), e*)
    display segment (right son (T), e*)
    return.
```

The point-in-polygon algorithm using the structure is described by Ballard as follows:

```
Algorithm:  Intersection (Z,T)
    (Comment:  Computed is the parity of the number of
    times a ray from Z intersects the polygonal segments
    within the strip T)
1.  If ray intersects both sides of the strip T that are
    parallel to the line L (see strip tree algorithm) then
    return (true).
2.  If ray does not intersect either side of the strip T
    that are parallel to L then return (false).
3.  Otherwise (Comment:  exclusive or)
        return [Intersection (Z, left son (T))
        or Intersection (Z, right son (T))]
```

Analysis

[Preprocessing]

Let T(i) denote the cost of constructing the strip tree for a polygonal segment with i+1 vertices $[V_0, V_1, \ldots V_i]$ using Ballard's algorithm. The cost of computing the strip which tightly encloses the polygonal segment, plus the cost of identifying the vertex $V_k (0 < k < i)$ which lies on one of the longitudinal sides of the strip, is bounded by ci, where c is a constant independant of i. The strip tree $[V_0, V_1, \ldots, V_i]$ then becomes available as soon as the strip tree for $[V_0, V_1, \ldots, V_k]$ and $[V_k, V_{k+1}, \ldots, V_i]$ have been determined. Thus the total cost of computing the strip tree for $[V_0, V_1, \ldots, V_i]$ is $T(i) = T(k) + T(i-k) + ci$.

Make the severe simplifying assumptions that i is a power of 2 ($i = 2^l$, say), and that k happens to be a mid-vertex (that is, $k = 2^{l-1}$), then

$$\begin{aligned} T(2^l) &= 2T(2^{l-1}) + c2^l \\ &= 2^l T(2^0) + cl2^l. \end{aligned}$$

Since $T(2^0) = 0$, it follows that $T(i) = ci\log i$ so that, for $i = n$, the algorithm is of complexity $O(n \log n)$. This cost is claimed by Ballard to be valid in all cases.

However, given a polygonal segment for which $k = i-1$, for this and all subsequent sub-segments (as for the segment in Figure III.14),

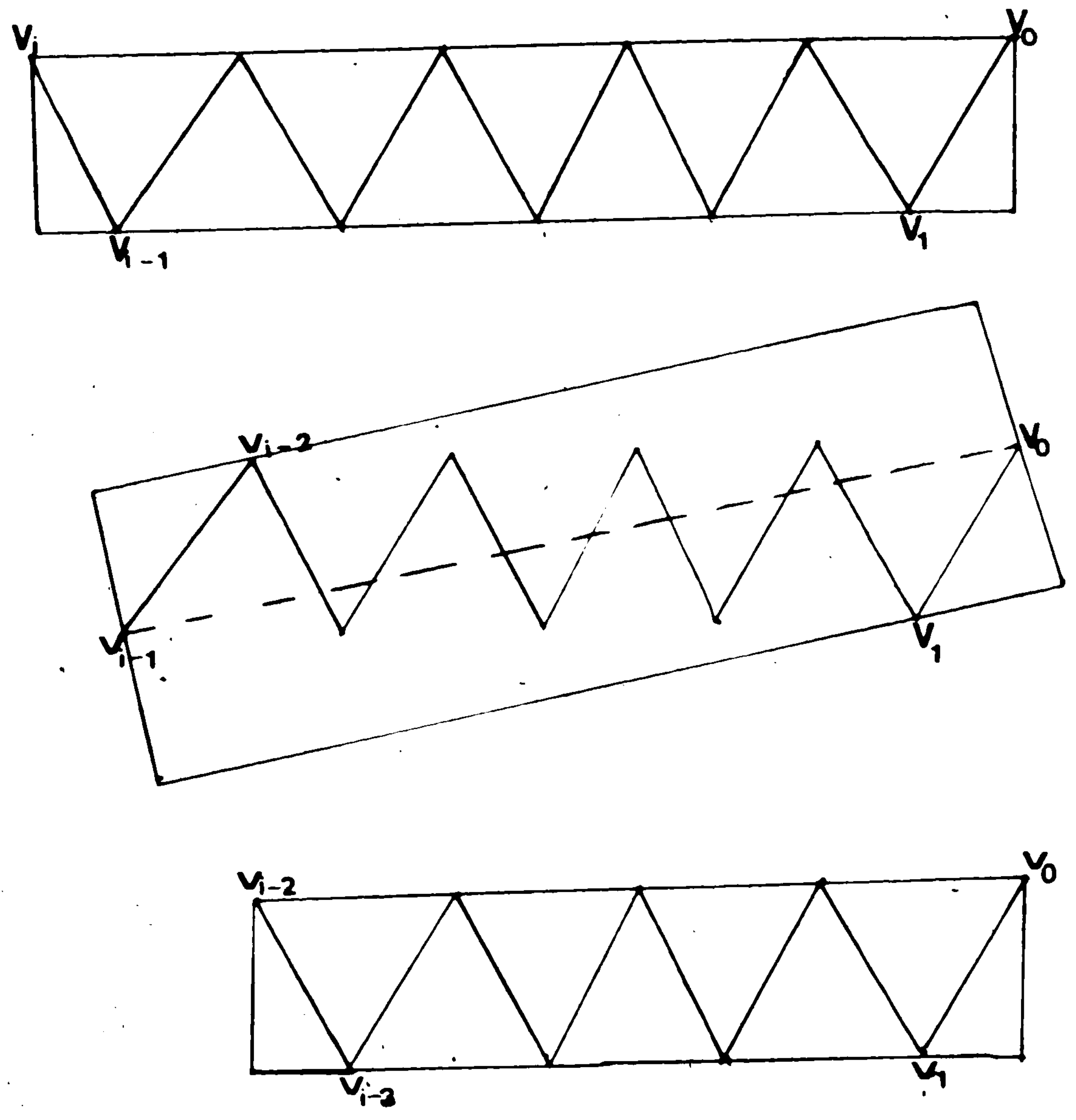


FIGURE III.14: Strip Tree Construction

$$
\begin{aligned}
T(i) &= T(i-1) + T(1) + ci \\
&= T(i-1) + ci \\
&= T(1) + ci(i-1)
\end{aligned}
$$

Thus, for $i = n$, in the worst case, $T(n) = O(n^2)$.

The reader should be made aware at this time, however, that Ballard describes an $O(n)$ algorithm for constructing a strip tree, but for which the strips enclose the polygonal segments extremely crudely. The practicability of the algorithm, therefore is severely restricted.

[Storage]

Since there are $O(n)$ strips, the total storage requirements are $O(n)$.

[Search]

Ballard asserts that for well-behaved polygons, the complexity of the point-in-polygon is $O(\log n)$. This assumes the strip tree is reasonably balanced and that one of the recursive calls which processes either the left or right sub- strips in step 3 of the algorithm exits quickly from the recursion. For the majority of polygons enclosed in geographic information systems, this assumption is indeed likely to be valid. However, the algorithm does require the processing of $n/2$ strips for the pathological example (the claw) given in Figure III.11, and therefore in the worst case, the algorithm is of complexity $O(n)$.

3.7 Conclusion

The following table summarizes the complexities of the five methods described in this chapter.

METHOD	RESULTS		
	Preprocessing	Storage	Searching
Swath	$O(n \log^2 n)$	$O(n^2)$	$O(\log n)$
Chain	$O(n \log n)$	$O(n)$	$O(\log^2 n)$
Triangle	$O(n \log n)$	$O(n)$	$O(\log n)$
Rectangle	$O(n)$	$O(n)$	$O(n)$
Strip	$O(n^2)$ *	$O(n)$	$O(n)$

TABLE II: Summary of Pre-Conditioned Methods

For n sufficiently large, the triangle method is asymptotically superior to the other methods. However, it must be stressed that these complexities are asymptotic bounds only (e.g. The Swath and Triangle methods both require C log n operations for searching, but C for the Triangle method is much larger).

For practical purposes, many other considerations will effect the selection of a method. Some of these include (1) the suitability of the data structure for other purposes (such as intersection of polygons), (2) the insensibility of the method toward noise in the polygonal data, (3) the ability of the method to rapidly capture the "character" of the polygon, and (4) average costs. Further discussion of these matters appears in the concluding chapter.

* a cruder algorithm of O(n) exists.

CHAPTER IV

PRACTICAL CONSIDERATIONS

4.1 Introduction

Techniques which answer the query of whether a given point lies in a polygon have been discussed in the last two chapters. In practice, a region may consist of a variety of subregions, for instance blocks of a town, and it is required to identify that subregion which contains a test point. Two methods, rectangle enclosure and circular enclosure which address this problem are discussed in this chapter. Such methods may be considered as an extension to the point-in-polygon algorithm. Although the discussion here is not treated in great depth, it provides an opportunity to remark on a subject that does not belong properly in the previous chapters (see Aldred[1], Nordbeck and Rystedt[23]).

4.2 Rectangular Enclosure

Let

$$X_{min} = Min(x_1, x_2, \ldots\ldots\ldots, x_n)$$

$$X_{max} = Max(x_1, x_2, \ldots\ldots\ldots, x_n)$$

$$Y_{min} = Min(y_1, y_2, \ldots\ldots\ldots, y_n)$$

$$Y_{max} = Max(y_1, y_2, \ldots\ldots\ldots, y_n)$$

Then the rectangle with vertices (X_{min}, Y_{min}), (X_{min}, Y_{max}), (X_{max}, Y_{max}) and (X_{max}, Y_{min}) is the rectangle of smallest area with sides parallel to the coordinates axis which encloses the polygon. The cost of computing this rectangle is, of course, O(n). Rectangles of smaller area which still enclose the polygon can be found (see Freeman and Shapira[11]) but these would require rotations. Such rotated rectangles however, are more costly to determine, require more storage, and adversely affect the performance of any subsequent processing.

To determine whether or not a test point $Z = (x,y)$ is contained in a given polygon, the associated rectangle is first examined for containment. If

$$x < X_{min}, \quad \text{or } x > X_{max}, \quad \text{or } y < Y_{min}, \text{ or } y > Y_{max}$$

then it is not, and Z cannot be contained in the polygon. Otherwise, the algorithms of the previous two chapters are applied to resolve the point-in-polygon problem.

4.3 Circular Enclosure

A circle which encloses the polygon is determined as follows. First, the distances among all possible pairs of vertices of the polygon are computed, and the largest such distance, d, is determined. A line segment is drawn between the two vertices V_i and V_j which achieve this maximum distance. The midpoint (x_c, y_c) of this line segment serves as the centre of circle with radius $r=d/2$. For "well-behaved" polygons, this circle tightly circumscribes the polygon. If it does not, the vertex V_k outside the circle and furthest away from the circumference is determined. The distance of V_k from the centre (x_c,y_c) can then be used as radius of a new circle with centre (x_c,y_c). This circle encompasses the polygon, but not necessarily tightly. The complexity of this algorithm is $O(n^2)$.

Alternatively, a tighter circle is obtained by passing a circumference through V_i, V_j and V_k. These three vertices can then be used as the first step in an iterative algorithm described briefly by Nordbeck and Rystedt[23] to find the circle of smallest area which encloses the polygon.

To determine whether or not a test point $Z = (x,y)$ is contained in given polygon, the associated circle with centre (x_c, y_c) and radius r is first examined for containment. If $(x-x_c)^2 + (y-y_c)^2 > r^2$ then it is not, and Z must lie outside the polygon. Otherwise, the algorithms of the previous two chapters need be applied in order to resolve the point-in-polygon problem.

Circular enclosures have advantages over rectangular ones in that

1. they require less storage, and
2. they normally produce tighter enclosures for those polygons which are relevant to most applications.

On the other hand, rectangular enclosures

1. are easier to determine ($O(n)$ versus $O(n^2)$), and
2. are usually more effective for subsequent computations (for example, the test for containment requires at most four comparisons versus three additions, two multiplications and one comparison required by a circular enclosure).

4.4 Multiple Polygon Processing

A common characteristic of most geographic information systems is a capability to partition a given region of interest into a number of smaller sub-regions. For the storage of descriptive data associated with some geographic point Z, it often becomes necessary to identify that sub-region (polygon) which contains Z. It is in this context that the enclosures of the two previous sections play a fundamental role.

All algorithms which identify that sub-region that contains Z require the processing of numerous sub-regions. Prior to invoking any particular point-in-polygon algorithm described in chapter II and III in order to determine whether or not a specific sub-region contains the point Z, it may be tremendously beneficial first to determine if Z is contained in the encompassing rectangle or circle. If it is not, then Z cannot be contained in the polygon and the much more costly point-in-polygon algorithm need not be involved. Processing then skips to the next sub-region.

It is assumed in the above paragraph that the sub-regions are listed sequentially and are processed according to this list until one is found which contains the point Z. To increase the likelihood of finding the sub-region which contains Z early in the list, a partial ordering of the sub-regions can first be defined. For example, the sub-regions can be ordered according to the distances of the centres of their encompassing rectangles (or, circles) from a point Z_0 external to the total region of interest. The sub-regions closest to the point Z_0, in the sense implied by the ordering above, would then be processed first. For general discussion of related implementation matters the reader is referred to Aldred [1].

CHAPTER V

CONCLUDING REMARKS

In this thesis, a variety of vector-based data structures for representing a polygon have been discussed and various algorithms for solving the point-in-polygon problem with respect to each of these structures have been presented and analysed. Analysis throughout has been confined primarily to asymptotic, worst-case analysis. The algorithms have been classified into three categories, namely, direct methods, pre-conditioned methods and multiple region processing methods.

Direct methods for solving the point-in-polygon problem are defined to be those which operate directly on the polygon represented as an ordered sequence of n vertices. The cost of each of the direct methods described is O(n). Pre-conditioned methods require that the polygon first be transformed into suitable data structures before search algorithms are applied. These structures permit the development of asymptotically faster algorithms [as fast as O(log n)] for solving the point-in-polygon problem. However, because of the additional preprocessing costs [typically O(n log n)] in constructing these structures and because of the additional storage costs in representing them, the benefits of fast search algorithms from a practical sense can be realized only in a static environment. In a dynamic environment where the boundaries of the polygon are subject to frequent changes, but depending on the frequency of these changes relative to the frequency of searches, the use of direct methods can turn out to be superior.

Of the direct methods, it is found that the line-crossing method for solving the point-in-polygon problem is superior in all respects. It is simpler, more efficient, and applicable to arbitrary polygons.

Of the pre-conditioned methods, the triangle method with preprocessing costs of O(n log n), storage costs of O(n) and search costs of O(log n) is asymptotically superior to the methods described. Indeed, Kirkpatrick shows that the method is asymptotically optimal.

From a practical point of view, however, asymptotic worst-case analysis of costs is inadequate for the following reasons:

1. It is justified only for n sufficiently large. Thus, it is possible that the chain method whose search costs are O($\log^2 n$) may be faster than the triangle method whose search costs are O(log n) for all n of practical interest (say, $n < 10^6$).

2. It may have no relationship to expected costs, since it must be valued for all cases including pathological ones. Thus, although the costs of the rectangle and strip methods are O(n) for a certain polygon, it is quite possible (we suspect it is likely) that the average costs of these methods for random polygons and for polygons of "practical" import is O(log n).

To answer the questions implied in the above two points requires the following undertaking:

1. The careful unbiased implementation of each of the algorithms.
2. The selection of appropriate test data ("interesting" classes and sub-classes of polygons).
3. The selection of effective comparative evaluation criteria.

These matters we leave as our primary suggestion for future research.

In selecting a method for a specific application, factors other than the comparative performance of the point-in-polygon algorithms can influence the final choice. The data structure underlying each of the methods may be particularly suitable or adaptable for solving other problems relevant to that application. Thus, the swath and chain methods are easily adaptable to the problem of spatial convex inclusion and to the problem of locating a point in a planar subdivision induced by m straight lines. The chain and triangle methods are both suitable for solving the nearest neighbour (post-office) problem, which is useful in a variety of applications including surveillance and tracking. In addition, the triangle method is appropriate in applications involving triangular finite element models (eg. some terrain models). The rectangle and strip methods, are conducive to solving problems of spacial analysis, including the union and intersection of polygons. Furthermore, because these last two methods yield approximations to the boundaries of a polygon, they can be used to capture the caricature of polygons at various resolutions. This is useful in numerous applications including graphic display systems.

REFERENCES

[1] ALDRED, B.K., "Points in Polygon Algorithms", UKSC-0025, IBM Scientific Centre, Neville Rd, Petterlee, County Durham, U.K., 1972.

[2] ANDERSON, K.P. "Simple algorithm for positioning a point close to a boundary", J. of Int. Assoc. for Math. Geology, Vol. 8(1), 1976, pp 105-106.

[3] BALLARD, D.H., "Strip trees: A hierarchical Representation of Map features", Comm ACM 24(5), May 1981, pp 310-321.

[4] BURTON, Warren, "Computer Processing of points, curves and regions", Ph.D. thesis - University of East Anglia, England, 1976.

[5] BURTON, W., "Representation for many-sided polygons and polygonal lines for Rapid Processong", Comm. ACM (20(3), March 1977, pp 166-171.

[6] BURTON, W. "Efficient Retrieval of Geographical Information on Basis of Location", 1st, Int. Advanced Study Symposium on Topological Data Structures, Vol 6, Fall 1977.

[7] DAVIS, M.W.D., and DAVID, M., "An algorithm for finding the position of a point relative to a fixed polygonal boundary", J. of Int. Assoc. for Math. Geology, Vol 12(1), 1980 pp 61-68.

[8] DOBKIN, D., and LIPTON, R.J., "Multidimensional Searching Problems", SIAM J. COMPUT 5(2), June 1976, pp 181-186.

[9] DOUGLAS, L., and PEUCKER, T., "Algorithm for the Reduction of the Number of points Requested to represent a digitized line or its Caricature", The Canadian Cartographer, 10(2), December, 1973, pp 112-122.

[10] FOWLER, R.J., "Approaches to Multi-dimensional Searching", 1st Int. Advanced Study Symposium On Topological Data Structures, Vol 6, Fall 1977.

[11] FREEMAN, B., and SHAPIRA, R., "Determining the minimum area Encasing Rectangle for an Arbitrary closed curve", Comm ACM 18(7), July 1975, pp 409-413.

[12] GAREY, M.R., JOHNSON, D.S., PREPARATA, R.P. and TARJAN, R.E., "Triangulating a simple Polygon", Info. Proc. Letters, Vol 7, #4 - June 1978, pp 175-179.

[13] HACKER, R., "Certification of Algorithm 112 - Position of point relative to polygon", Comm. ACM, December, 1962, p 606.

[14] HALL, J.K., "PTLOC- a FORTRAN subroutine for determining the position of a point relative to a closed boundary", J. Int. Assoc. for Math. Geology, Vol 7(1), 1975, pp 75-79.

[15] JACOBSON, J.D., "Geometric Relationship for Retrieval of Geographic Information", IBM syst. J., 1968, pp 331-341.

[16] KIRKPATRICK, D.G., "Optimal Search in Planar Subdivisions", June 1979 - Personal Communication.

[17] KUROZUMI, Y., and DAVIS, W.A., "Polygonal Approximation of Digitized Curves by a Minimax Method", ACM Computer Science Conference, St. Louis, Mo., February 1981.

[18] LEE, D.T. and PREPARATA, F.P., "Location of a point in a planar subdivision and its application", SIAM J. COMPUT, Vol 6(3), 1977, pp 594-606.

[19] LIPTON, R.J., and TARJAN, R.E., "Application of Separator Theorem", Proc. 18th Annual Symposium on Foundations of Computer Science, Oct - Nov. 1977, pp 162-170.

[20] LEE, D.T. and YANG, C.C., "Location of Multiple Points in a Planar Subdivision", INFORMATION PROCESSING LETTER, VOL 9(4), November 1979.

[21] LOOMIS, R.G., "Boundary Networks", COMM. ACM 8(1), January 1965, pp 44-48.

[22] MERRILL, R.D., "Representation of Contours and efficient computer search", COMM. ACM 16,2, February, 1973, pp 69-82.

[23] NORDBECK, S., and RYSTEDT, B., "Computer Cartography Point-in-Polygon Programs", BIT 7, 1967, pp 39-64.

[24] PAVLIDIS, T., "Analytical Description of Region Boundaries and Curves; Chapter 7 of Structural Pattern Recognition", Springer-Verlag, New York, Vol 1, 1977.

[25] PEUCKER, T.A., "A Theory of the Cartographic Line", International Yearbook of Cartography, 16, 1976.

[26] PEUQUET, D.J., "Raster Processing: An Alternate Approach to Automated Cartographic Data Handling", American Cartography Vol 6(2), April, 1979, pp 129-139.

[27] PREPARATA, F.P., "A note on locating a set of points in a planar subdivision", SIAM J. COMPUT, Vol 8(4), November, 1979.

[28] REID, J.K., "A note on the approximation of place Computer Journal, 14(3), 1971, pp 307-308.

[29] SALOMON, K.B., "An efficient point-in-polygon algorithm", Computers & Geosciences (Great Britain) Vol 4, 1978, pp 173-178.

[30] SALOMON, K.B., "Algorithm for determining the orientation of a boundary", Computers & Geosciences (Great Britain), Vol 3(2)1, 1977, pp 383-384.

[31] SHAMOS, M.I., "GEOMETRIC COMPLEXITY", Proc. on Theory of Computing, ACM, 1975, pp 224-233.

[32] SHIMRAT, M., "Position of a Point Relative to Polygon", COMM. ACM, 5, August 1962, ALGORITHM 112, p 434.

SUPPLEMENTARY REFERENCES

[33] BENTLEY, J.L., "Multidimensional Search Trees Used for Associative Searching", COMM. ACM, 18,9, September 1975, pp 509-517.

[34] BURTON, W., "Logical and Physical Data Types in Geographical Information Systems" - Geo-Processing 1, 1979, pp 167-181.

[35] KNUTH, D.B., "The Art of computer programming vol 3 - Sorting and searching", (Addison - Wesley), Reading MASS - 1973.

[36] RAGHAVAN, V. and YU, C.T., "A Note on a Multi-dimensional Searching Problem", Information Processing Letters, 6(4), August 1977, pp 133-135.

[37] SHAMOS, M.I., "Problems in Computational Geometry", Technical Report, Dept. of Comp. Sci., Yale Univ, New Haven, Conneticut, 1975.

[38] SHAMOS and BENTLEY, "Optimal algorithm Structuring Geographic Data", 1st Int. Advanced Study Symposium on Topological Data Structures - Havard Papers, Vol 6, Fall 1977.