

MODIFICATION OF FOURNIER AND MONTUNO'S
TRIANGULATION ALGORITHM FOR
SIMPLE POLYGONS

By

SOEHADI ADI

Bachelor of Science

in Electrical Engineering

Oklahoma State University

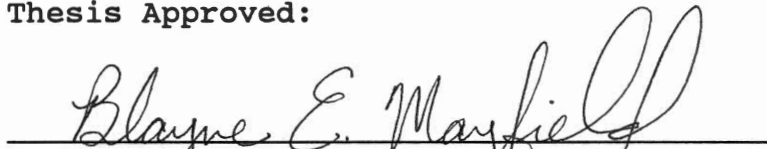
Stillwater, Oklahoma

1988

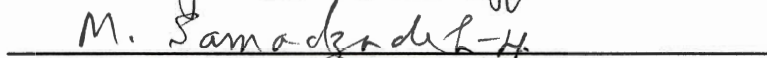
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 1991


MODIFICATION OF FOURNIER AND MONTUNO'S
TRIANGULATION ALGORITHM FOR
SIMPLE POLYGONS

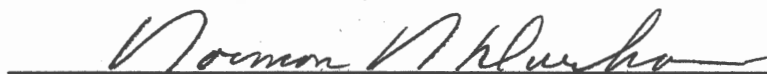
Thesis Approved:



Thesis Adviser







Dean of the Graduate College

PREFACE

Triangulation of a simple polygon is an important part of the application of geometry problems in computer graphics. A conventional triangulation algorithm runs in $O(n^2)$. Faster triangulation methods have been developed but these methods are more complicated. A simpler triangulation method was developed by Fournier and Montuno, which runs in $O(n \log n)$. The modified triangulation algorithm presented here compares favorably with the Fournier and Montuno's triangulation algorithm and is simpler in the sense of elimination of recursion. The Fournier and Montuno's triangulation algorithm and the modified triangulation algorithm are implemented using C. The performance of the two algorithms is analyzed using various data.

I wish to express my sincere gratitude first to God, second I wish to thank my main adviser, Dr. Blayne E. Mayfield, for his intelligent guidance, inspiration, and invaluable help, I am also grateful to the other committee members, Drs. Mansur Samadzadeh and David Miller, for their advice during the course of this project.

I would like to express my gratitude to my parents Mr. and Mrs. Sindu Rahardjo, my wife Selvia P. Suniman, my son Jonathan T. Adi, and my pastor Harminto Ongko, for their support and encouragement during my course work.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
1.1 Problem Background	1
1.2 Definitions.	2
II. TRIANGULATION OF A SIMPLE POLYGON.	7
2.1 Garey, Johnson, Preparata, and Tarjan's Triangulation Algorithm.	7
2.2 Fournier and Montuno's Triangulation Algorithm.	10
2.3 Tarjan and Van Wyk's Triangulation Algorithm.	11
2.4 Advantages and Disadvantages	12
III. FOURNIER AND MONTUNO'S TRIANGULATION ALGORITHM . .	13
3.1 Transform the Simple Polygon into a Trapezoidized Polygon	13
3.2 Transform the Trapezoidized Polygon into a Uni-Monotonized Polygon.	20
3.3 Transform the Uni-Monotonized Polygon into a Triangulated Polygon	22
IV. MODIFIED TRIANGULATION ALGORITHM	24
4.1 Transform the Simple Polygon into a Trapezoidized Polygon.	24
4.2 Transform the Trapezoidized Polygon into a Uni-Monotonized Polygon.	25
4.3 Transform the Uni-Monotonized Polygon into a Triangulated Polygon	29
V. COMPARISON OF FOURNIER AND MONTUNO'S AND THE MODIFIED TRIANGULATION ALGORITHMS.	30
5.1 Processing Flow of the Algorithms.	30
5.2 Simplicity	32
5.3 Run-Time Complexity.	32

Chapter	Page
VI. SUMMARY AND FUTURE WORK.	37
REFERENCES.	38
APPENDIXES.	40
APPENDIX A - DETAILS OF THE MODIFIED TRIANGULATION ALGORITHM	43
APPENDIX B - IMPLEMENTATION.	52
APPENDIX C - SAMPLE EXECUTION.	60

LIST OF FIGURES

Figure	Page
1. Complete Trapezoids.	3
2. An Example of a Monotone Polygon	4
3. An Example of a Uni-Monotone Polygon	4
4. Examples of Non-Simple Polygons and a Simple Polygon	5
5. Sequence of the Algorithm.	8
6. Interior Cusp Vertices of a Simple Polygon	9
7. Execution Sequence of the Algorithm.	10
8. Execution Sequence of Fournier and Montuno's Algorithm.	14
9. Three Phases of the Triangulation Algorithm (SP, UMP, and TP).	14
10. Different Vertex Types	14
11. Algorithm to Transform the SP into a TrP	16
12. Vertex Type I Process.	17
13. Vertex Type IIa Process.	18
14. Vertex Type IIb Process.	19
15. Vertex Type IIIa Process	19
16. Vertex Type IIIb Process	20
17. Algorithm to Transform TrP into a UMP then into a TP.	21
18. Two Classes of Trapezoids.	21
19. Algorithm to Transform the UMP into a TP	22
20. Execution Sequence of the Modified Algorithm	25

Figure	Page
21. Algorithm to Transform SP into a TrP, into a UMP, and then into a TP	26
22. Algorithm to Transform TrP into a UMP, and then into a TP	27
23. The Processing Flow of Fournier and Montuno's and the Modified Triangulation Algorithms.	31
24. The Vertices of a Simple Polygon	42
25. A Simple Polygon	42
26a. The Transformation of a Simple Polygon into a Trapezoidized Polygon	44
26b. The Transformation of a Simple Polygon into a Trapezoidized Polygon.	45
26c. The Transformation of a Simple Polygon into a Trapezoidized Polygon.	46
27. The Trapezoidized Polygon.	46
28a. The Transformation of a Trapezoidized Polygon into a Uni-Monotonized Polygon	48
28b. The Transformation of a Trapezoidized Polygon into a Uni-Monotonized Polygon	49
29. The Uni-Monotonized Polygon.	49
30. The Processing Transformation from Uni-Monotone Polygon to a Triangulated Polygon.	51
31. The Triangulated Polygon	51
32. The Correct Input Format	53
33. The Displayed Run-Time Unit Value Formula.	57
34a. The Process Sequence Flow Chart.	58
34b. The Process Sequence Flow Chart.	59
35. The Logo and Information about the Program	60
36. Input Data of Sample #1 (DATA\B.12).	61
37. The Input Prompted by the Program.	62
38. A Simple Polygon	62

39a.	The Sequence Formation of Uni-Monotone Polygons and Triangles (FM alg.)	63
39b.	The Sequence Formation of Uni-Monotone Polygons and Triangles (FM alg.)	64
40a.	The Sequence Formation of Uni-Monotone Polygons and Triangles (MO alg.)	65
40b.	The Sequence Formation of Uni-Monotone Polygons and Triangles (MO alg.)	66
41.	Input Data of Sample #2 (DATA/B.3_P)	67
42.	The Input Prompted by the Program.	68
43a.	The Sequence Formation of Uni-Monotone Polygons and Triangles (3 Vertices)	68
43b.	The Sequence Formation of Uni-Monotone Polygons and Triangles (4 Vertices)	69
43c.	The Sequence Formation of Uni-Monotone Polygons and Triangles (5 Vertices)	69
43d.	The Sequence Formation of Uni-Monotone Polygons and Triangles (6 Vertices)	69
43e.	The Sequence Formation of Uni-Monotone Polygons and Triangles (7 Vertices)	70
43f.	The Sequence Formation of Uni-Monotone Polygons and Triangles (25 Vertices)	70
43g.	The Sequence Formation of Uni-Monotone Polygons and Triangles (25 Vertices)	70
44a.	Run-Time Table	71
44b.	Run-Time Chart	71
45a.	Distribution Information on 50 Vertices of 50 Simple Polygons.	73
45b.	Distribution Information on 1000 Vertices of 50 Simple Polygons.	73
46a.	Run-Time Chart of All Simple Polygons.	74
46b.	Run-Time Chart of All Simple Polygons (Enlargement).	75

CHAPTER I

INTRODUCTION

1.1 Problem Background

In computational geometry problems, as in other contexts, it is desirable to decompose a complex structure into simpler structures. Several authors have developed algorithms as tools to simplify complex structures: the decomposition of a simple polygon into convex parts [CD79, K85, S78], the decomposition of a simple polygon into star polygons [AT81], the trapezoidization of a simple polygon [L81, W70, FM84], triangulation of a set of points [L77, S75], triangulation of a planar region [K83], and triangulation of a simple polygon [FM84, GJPT78, HM83, TV88].

The problem of triangulation of a simple polygon is the focus of this work and has the following applications in the field of computer graphics:

- speed and simplicity of hardware implementation (which is essential in shading and scan conversion) [FR82],
- two dimensional function interpolation (the result being independent of the orientation of the triangles) and evaluating functions by interpolation [FS73, M76],
- closest point problem [LP77], etc.

The resulting triangulation algorithm should be effi-

cient and as simple as possible. Several methods have been developed to triangulate a simple polygon. Some authors claim that the problem of triangulation is less complex than the problem of sorting [TV88]. The triangulation algorithm presented in this thesis is a modification of the Fournier and Montuno triangulation algorithm [FM84]. The modified triangulation algorithm is simpler in the sense that recursion is eliminated. It is also as efficient as the original Fournier and Montuno triangulation algorithm, and runs faster in implementation.

1.2 Definitions

Left Edge of the Current Vertex: the left side edge of the current vertex as viewed from the polygon interior.

Right Edge of the Current Vertex: the right side edge of the current vertex as viewed from the polygon interior.

Trapezoid (trp): a trapezoid having horizontal parallel edges. A trapezoid is said to be "complete" if it has (see Figure 1):

1. a top vertex (VT) which defines the top parallel edge
2. a bottom vertex (VB) which defines the bottom parallel edge
3. a left side edge (EL)
4. a right side edge (ER)

If the top or bottom parallel edge is zero in length, the trapezoid is actually a triangle (a triangle is considered to be a special form of a trapezoid).

Trapezoid Diagonal: an edge that connects a top vertex of a trapezoid (VT) and a bottom vertex of a trapezoid (VB) (see Figure 1).

External Diagonal: a trapezoid diagonal that has the same edge either the left side edge of a trapezoid (EL) or the right side edge of a trapezoid (ER) (see Figure 1).

Internal Diagonal: a trapezoid diagonal that has different edge from either the left side edge of a trapezoid (EL) or the right side edge of a trapezoid (ER) (see Figure 1).

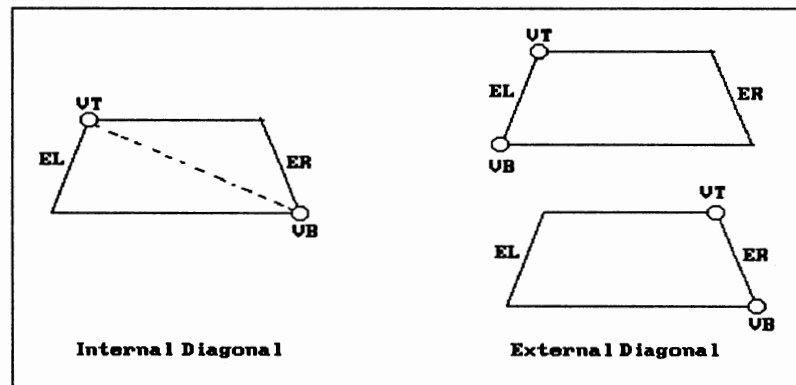


Figure 1. Complete Trapezoids.

Monotone Polygon (mp): a polygon that has n vertices labeled v_0, v_1, \dots, v_{n-1} in clockwise order, such that v_0 and v_i have the maximum and minimum y -coordinates of all vertices in the polygon, and v_0, v_1, \dots, v_i are decreasing monotonically in y -coordinates and v_i, \dots, v_{n-1}, v_0 are increasing monotonically in y -coordinates (see an example in Figure 2).

Uni-Monotone Polygon (ump): a polygon that has n vertices

labeled v_0, v_1, \dots, v_{n-1} either in clockwise or counter-clockwise order, such that v_0 and v_{n-1} are the maximum and minimum y-coordinates of all vertices in the polygon, and v_0, \dots, v_{n-1} are monotonically decreasing in y-coordinates (see an example in Figure 3).

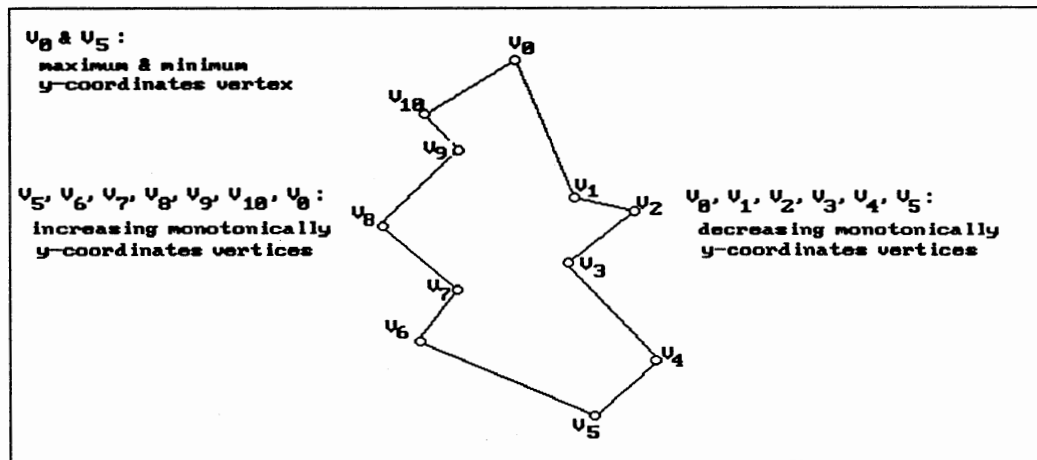


Figure 2. An Example of a Monotone Polygon.

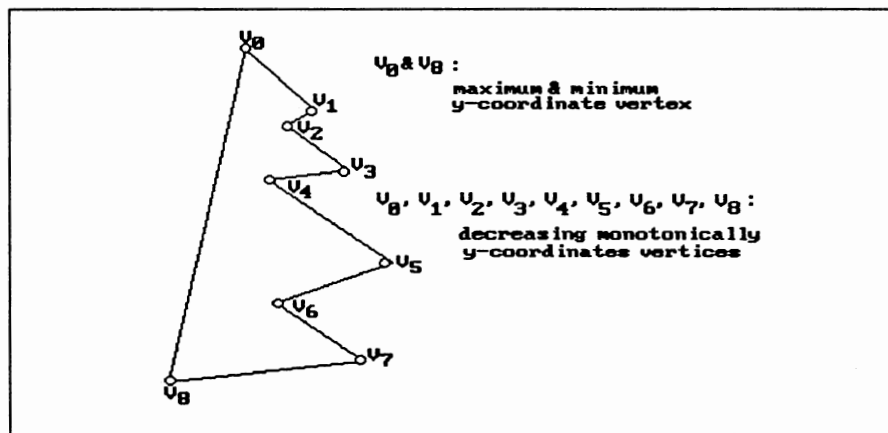


Figure 3. An Example of a Uni-Monotone Polygon.

Simple Polygon (SP): a polygon in which no edges cross each other, which has a unique y-coordinates on each vertex, and for which there are no holes inside the polygon (see

Figure 4).

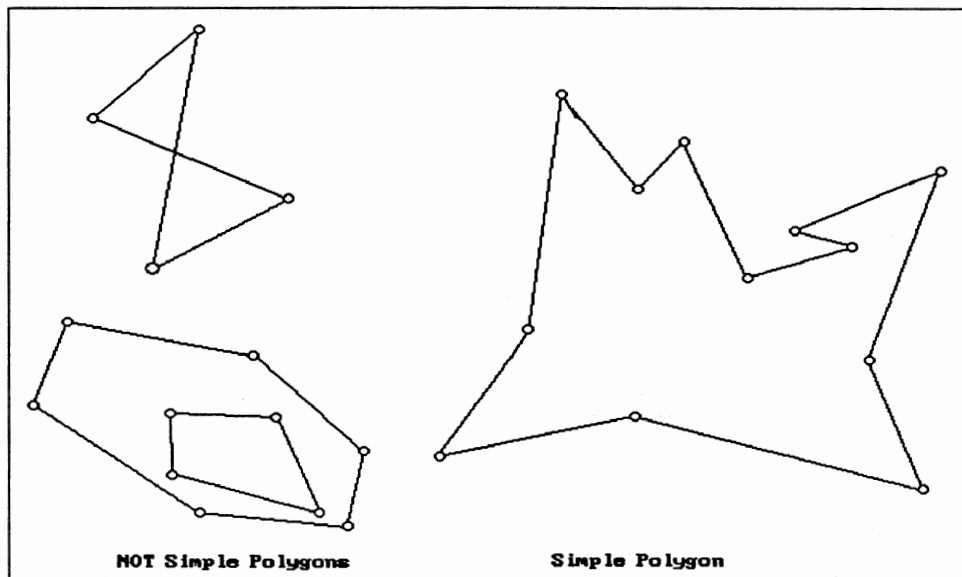


Figure 4. Examples of Non-Simple Polygons and a Simple Polygon.

Trapezoidized Polygon (TrP): a simple polygon that is divided into trapezoids by adding an internal horizontal edge originating at each vertex other than vertices of type IIb or IIIa vertices (as described on Section 3.1), so that between one vertex and another, a trapezoid will be formed (see an example in the Appendix A).

Monotonized Polygon (MP): a simple polygon that is divided into monotone polygons.

Uni-Monotonized Polygon (UMP): a simple polygon that is divided into uni-monotone polygons (see an example in appendix A).

Triangulated Polygon (TP): a simple polygon that is divided into triangles (tps) (see an example in appendix A).

Simple Polygon Diagonal: a line segment joining two non-

adjacent vertices of a simple polygon [GJPT78].

Triangulation: the process of triangulating a simple polygon of n vertices (i.e., the process of finding $n-3$ simple polygon diagonals which intersect neither each other nor the boundary of the simple polygon and which divide the interior of the simple polygon into $n-2$ triangles [GJPT78]).

SP_TrP: a procedure that transforms a simple polygon into a trapezoidized polygon.

SP_TrP_UMP_TP: a procedure that transforms a simple polygon into a trapezoidized polygon, into a uni-monotonized polygon, and then into a triangulated polygon.

TrP_UMP_TP: a procedure that transforms a trapezoidized polygon into a uni-monotonized polygon, and then into a triangulated polygon.

UMP_TP: a procedure that transforms a uni-monotonized polygon into a triangulated polygon.

CHAPTER II

TRIANGULATING A SIMPLE POLYGON

A simple polygon of n vertices can be partitioned into no fewer than $(n-2)$ triangles with $(n-3)$ diagonals [TV88]. Conventional triangulation algorithms runs in $O(n^2)$, but several more complicated methods have been developed to run faster, at $O(n \log n)$, or even at $O(n \log \log n)$.

In 1978, Garey, Johnson, Preparata, and Tarjan first proposed an $O(n \log n)$ triangulation algorithm [GJPT78]. In 1984, Fournier and Montuno presented a triangulation algorithm that runs in $O(n \log n)$ and claimed that their algorithm was simpler than other currently available algorithms [FM84]. Then in 1988, Tarjan and Van Wyk proposed a $O(n \log \log n)$ triangulation algorithm [TV88].

2.1 Garey, Johnson, Preparata, and Tarjan's Triangulation Algorithm

This algorithm is composed of two steps (see Figure 5):

1. Regularization, which transforms the simple polygon (SP) into a monotonized polygon (MP).
2. Triangulation of a monotonized polygon, which transforms the MP into a Triangulated Polygon (TP).

The following is the outline of what happens in each

step.

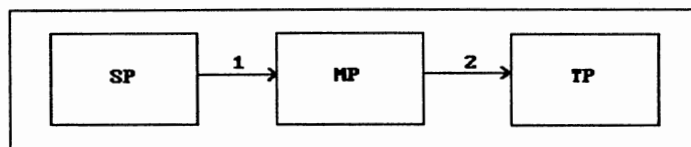


Figure 5. Sequence of the Algorithm.

Regularization

This step tries to build monotone polygons from a simple polygon; it is based on a property of monotone polygons, that a monotone polygon is a simple polygon in which no vertex is an interior cusp [GJPT78]. A vertex of a simple polygon is an interior cusp vertex if the internal angle at the vertex is more than 180° and the two vertices adjacent to the vertex on the boundary of the simple polygon either both have larger y-coordinates than the vertex or both have smaller y-coordinates than the vertex [GJPT78] (see Figure 6). This step requires a $O(n \log n)$ time [GJPT78]. The detail of this step can be seen in Lee and Preparata's work.

Triangulation of a Monotone Polygon

This step initially creates a list of sorted vertices in descending order of their y-coordinates. Then a stack is created initially containing the first two vertices from the sorted vertices list. The third vertex in the sorted list is designated as the current vertex.

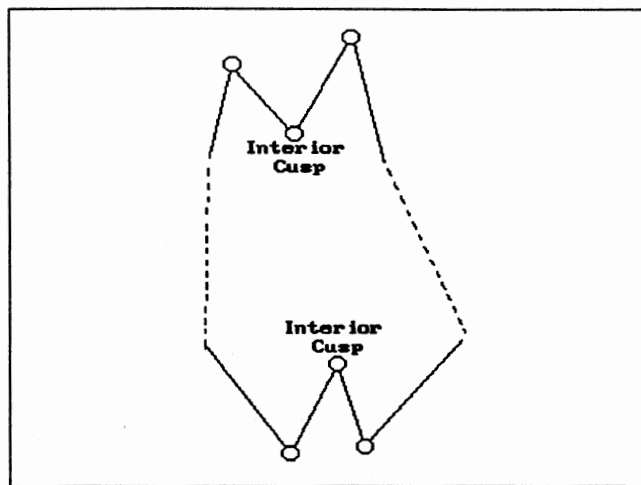


Figure 6. Interior Cusp Vertices of a Simple Polygon.

The current vertex is processed in one of the following ways:

1. If the current vertex is adjacent to the vertex at the bottom of the stack, but not to the vertex at top of the stack, then add diagonals between the current vertex and all vertices in the stack (except the vertex at the bottom of the stack). The stack contents are replaced by the vertex at the top of the stack and the current vertex. Then the next one of the sorted vertices on the list is designated as the current vertex.
2. If the current vertex is adjacent to the vertex at the top of the stack, but not to the vertex at the bottom of the stack, then repeat the following until one vertex is left in the stack or the internal angle of the vertex at the top of the stack is more than or equal to 180° :
 - add diagonals between the current vertex and the second vertex at the top of the stack; delete the

vertex at the top of the stack (pop operation).

The current vertex is added to the top of the stack (push operation). Then the next one of the sorted vertices on the list is designated as the current vertex.

3. If the current vertex is adjacent to both the vertex at the bottom of the stack and the vertex at the top of the stack, then add diagonals between the current vertex and all vertices in the stack (except the vertex at the bottom of the stack and the vertex at the top of the stack) and stop.

This step requires a $O(n)$ time. Thus, the entire algorithm runs in $O(n \log n)$ time.

2.2 Fournier and Montuno's Triangulation

Algorithm

This algorithm will be explained in detail in Chapter III; it is composed of three steps (see Figure 7):

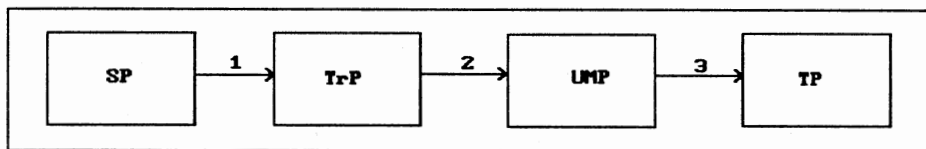


Figure 7. Execution Sequence of the Algorithm.

1. Trapezoidization of a simple polygon, which transforms the simple polygon (SP) into a trapezoidized polygon (TrP).
2. Composition of uni-monotone polygons from trapezoid structures, which transforms the TrP into a uni-monoto-

nized polygon (UMP).

3. Triangulation of a uni-monotone polygon, which transforms the UMP into a triangulated polygon (TP).

This algorithm uses 2-3 tree for searching, insertion, and deletion of the trapezoid structures.

This algorithm is significant in combining both simplicity and speed. It requires a $O(n \log n)$ time [FM84].

2.3 Tarjan and Van Wyk's Triangulation

Algorithm

The starting point for this algorithm is a reduction of the triangulation problem to the problem of computing visibility information (in the horizontal direction) [TV88]. A vertex-edge visible pair is a vertex and an edge that can be connected by an horizontal line segment that lies entirely inside the polygon [TV88]. An edge-edge visible pair is a pair of edges that can be connected by an horizontal line segment that lies entirely inside the polygon [TV88].

The second point of this algorithm relies on the intimate connection between visibility computation and the Jordan sorting problem [TV88]. Jordan sorting is a sorting of the intersection points of boundary polygon and a horizontal line by x-coordinate [TV88, HMRT86].

This algorithm uses a finger search tree for the data structure operation [TV88].

Overall this algorithm runs in $O(n \log \log n)$ [TV88], but it is complex.

2.4 Advantages and Disadvantages

A faster algorithms gives a better run-time complexity, but is usually complicated and hard to understand.

A simpler algorithm can be implemented more easily and is generally easier to understand, but is usually slower in terms of run-time complexity.

It is obviously desirable to choose an algorithm which is simple to understand and also runs fast. Fournier and Montuno claim that their triangulation algorithm has these characteristics. The modified triangulation algorithm presented here is simpler than the Fournier and Montuno's triangulation algorithm and runs as fast.

CHAPTER III

FOURNIER AND MONTUNO'S TRIANGULATION ALGORITHM

This algorithm is divided into three steps:

1. Transform a Simple Polygon (SP) into a Trapezoidized Polygon (TrP);
2. Transform the TrP into a Uni-Monotonized Polygon (UMP); and
3. Transform the UMP into a Triangulated Polygon (TP).

After the SP is divided into trapezoids (TrP), it is transformed into one or more uni-monotone polygons (UMP). After each uni-monotone polygon is completed, it is further divided directly into triangles.

Figure 8 illustrates the execution sequence of the algorithm. Figure 9 illustrates a polygon with lines indicating the three phases.

Now each of the steps of the algorithm will be described individually.

3.1 Transform the Simple Polygon into a Trapezoidized Polygon

It is assumed that a simple polygon already exists. Each vertex of the simple polygon has two edges leading from

it. These two edges are categorized as follows [FM84] (see Figure 10):

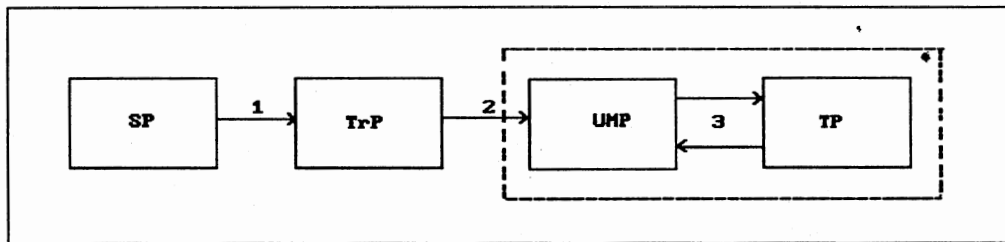


Figure 8. Execution Sequence of Fournier and Montuno's Algorithm.

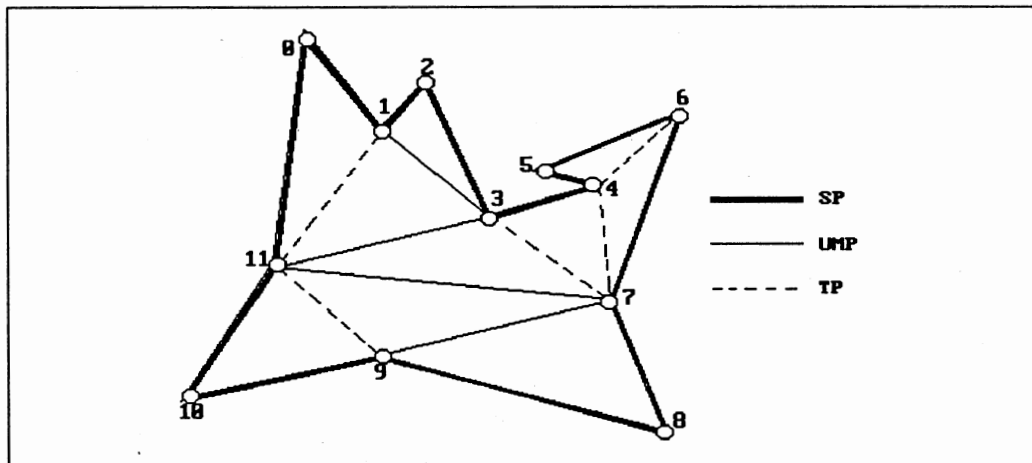


Figure 9. Three Phases of the Triangulation Algorithm (SP, UMP, and TP).

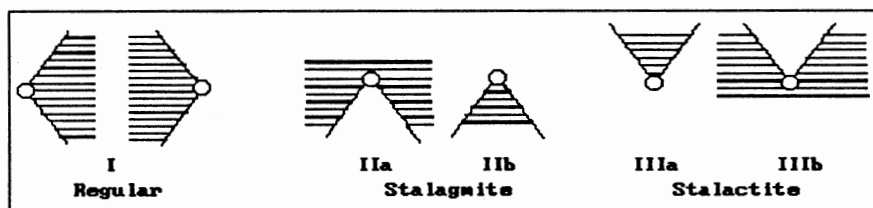


Figure 10. Different Vertex Types.

- a. Type I (Regular), where one edge slants diagonally upward and the other edge slants diagonally downward.
- b. Type II (Stalagmite), where both edges slant diagonally

downward. Vertices of this type can be more specifically classified into the following sub-categories:

- Type IIa, if the polygon interior (shaded area) is above the polygon boundary; and
- Type IIb, if the polygon interior (shaded area) is below the polygon boundary.

c. Type III (Stalactite), where both edges slant diagonally upward. Vertices of this type can be more specifically classified into the following sub-categories:

- Type IIIa, if the polygon interior (shaded area) is above the polygon boundary; and
- Type IIIb, if the polygon interior (shaded area) is below the polygon boundary.

This part of the Fournier and Montuno triangulation algorithm is shown in Figure 11.

The first step begins by sorting the vertices of the simple polygon by their y-coordinates from highest to lowest. Then each vertex (in sorted order) is processed to be transformed into trapezoids.

A trapezoid is considered complete if it has all information about VT (Top Vertex), VB (Bottom Vertex), EL (Left Edge), and ER (Right Edge). As trapezoids are built, they may have only VT, EL, and ER information, but no VB. These incomplete trapezoids are kept in a 2-3 tree. An incomplete trapezoid is completed after VB information is added, and is then removed (deleted) from the 2-3 tree. The vertex ID (i.e., v_0 , v_1 , etc.) of the top vertex of the incomplete

trapezoids is the key for searching, insertion, and deletion.

```

Input: n vertices (v0, v1, ..., vn-1) of a simple polygon
Output: k triangles
       where  $k = \sum_{i=0}^{m-1} k_i$  and m = number of umps
Note:
vi - current vertex
eL - left edge of the current vertex (vi)
eR - right edge of the current vertex (vi)
UT - top vertex of the trapezoid structure
VB - bottom vertex of the trapezoid structure
EL - left edge of the trapezoid structure
ER - right edge of the trapezoid structure
Trapz ID - ID of Trapezoid searched
Algorithms: SP_TrP()
sort all vertices (in y-coordinate, max to min)
for each vertex (in the order of sort)
{
  switch (vertex type of current vertex <vi>)
  {
    case I:
      Search_Edge of vi in 2-3 tree (&TrapzID, eL, eR)
      Complete the trapezoid with vi as bottom vertex
      (TrapzID)
      Remove trapezoid from 2-3 tree (TrapzID)
      Insert new trapezoid into 2-3 tree (vi, EL, eR)
    case II:
      Search vi location in 2-3 tree (&TrapzID)
      If vi is within an active trapezoid (type IIa)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID)
        Remove trapezoid from 2-3 tree (TrapzID)
        Insert new trapezoid into 2-3 tree (vi, EL, eL)
        Insert new Trapezoid into 2-3 tree (vi, eR, ER)
      }
      Else (type IIb)
        Insert new trapezoid into 2-3 tree (vi, eL, eR)
    case III:
      Search Edge of vi in 2-3 tree (&TrapzID, eL, eR)
      If eL and eR belong to the same trapezoid (IIIa)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID)
        Remove trapezoid from 2-3 tree (TrapzID)
      }
      Else (type IIIb)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID1)
        Complete the trapezoid with vi as bottom vertex
        (TrapzID2)
        Remove trapezoid from 2-3 tree (TrapzID1)
        Remove trapezoid from 2-3 tree (TrapzID2)
        Insert new trapezoid into 2-3 tree (vi, EL", ER")
      }
  }
}

```

Figure 11. Algorithm to Transform the SP into a TrP [FM84].

This transformation process always starts with a vertex of type IIb (the highest y-coordinate vertex) to initiate the first incomplete trapezoid. As each vertex is processed, incomplete trapezoids may be completed and new trape-

zoids may be started. An incomplete trapezoid can only be completed if the current vertex lies on one of its side edges.

The following is a detailed explanation of what happens to vertices of each type:

Type I:

This vertex type indicates the completion of one trapezoid and the beginning of a new trapezoid. The completion of one trapezoid is accomplished by completing an upper adjacent trapezoid (of this vertex) with the current vertex as its bottom vertex. The beginning of a new trapezoid is accomplished by beginning a new trapezoid with the current vertex (v_i) as its top vertex. Either the left edge of the trapezoid just completed and the right edge of the current vertex or the left edge of the current vertex and the right edge of the trapezoid just completed are the left and the right edges, respectively, of the new trapezoid (see Figure 12).

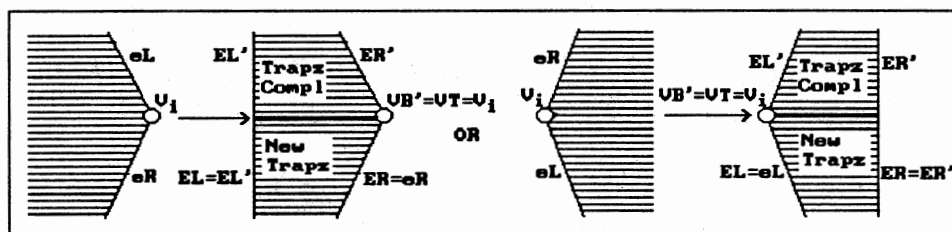


Figure 12. Vertex Type I Process.

Type IIa:

This vertex type indicates the completion of one trape-

zoid and the beginning of two new trapezoids. The completion of one trapezoid is accomplished by completing an upper adjacent trapezoid (of this vertex) with the current vertex (v_i) as its bottom vertex. The beginning of two new trapezoids are accomplished by beginning two new trapezoids with the current vertex as their top vertex. The left edge of the trapezoid just completed and the left edge of the current vertex are the left and the right edges, respectively, of one of the new trapezoids. The right edge of the current vertex and the right edge of the trapezoid just completed are the left and the right edges, respectively, of the other new trapezoid (see Figure 13).

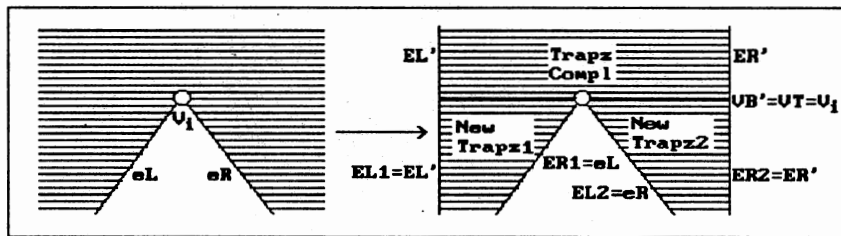


Figure 13. Vertex Type IIa Process.

Type IIb:

The vertices adjacent to a type IIb vertex must have lower y-coordinates than that vertex, and the interior of the polygon (shaded area) will be below the vertex (see Figure 14). A type IIb vertex indicates the beginning of a new trapezoid with the current vertex as its top vertex. The left and the right edges of the current vertex are the left and the right edges, re-

spectively, of the new trapezoid (see Figure 14).

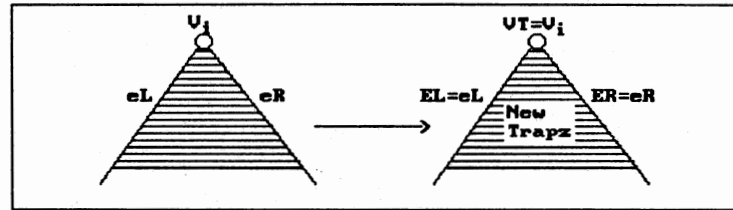


Figure 14. Vertex Type IIb Process.

Type IIIa:

This vertex type indicates the completion of one trapezoid which is accomplished by completing an upper adjacent trapezoid (of this vertex) with the current vertex (v_i) as its bottom vertex (see Figure 15).

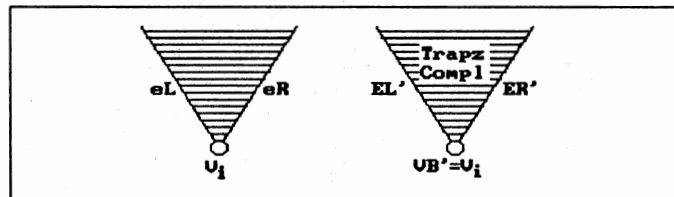


Figure 15. Vertex Type IIIa Process.

Type IIIb:

This vertex type indicates the completion of two trapezoids and beginning of one new trapezoid. The completion of two trapezoids is accomplished by completing two upper adjacent trapezoids (of this vertex) with the current vertex (v_i) as their bottom vertex. The beginning of one new trapezoid is accomplished by beginning a new trapezoid with the current vertex as its top vertex. The left edge of one of the trapezoid just

completed and the right edge of the other trapezoid just completed are the left and the right edges, respectively, of the new trapezoid (see Figure 16).

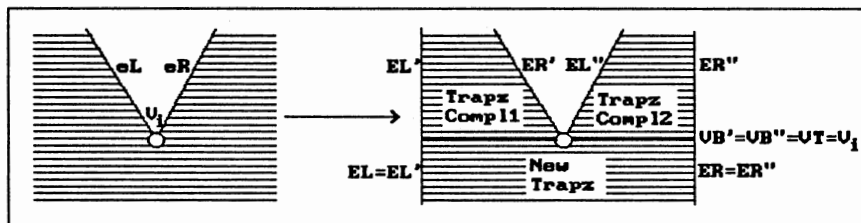


Figure 16. Vertex Type IIIb Process.

3.2 Transform the Trapezoidized Polygon into a Uni-Monotonized Polygon

When this second step begins, all trapezoids have already been completed. Trapezoids are processed to form uni-monotone polygons which will then be triangulated. This portion of the algorithm is shown in Figure 17.

This step will divide the simple polygon into uni-monotone polygons using the information provided by the trapezoidized polygon.

There are two kinds of trapezoids [FM84] (see Figure 18):

1. Class A: two vertices share an edge of a trapezoid; and
2. Class B: two vertices do not share an edge of a trapezoid.

For class B, the two vertices create one of the triangle edges (this is illustrated by the dotted lines in Figure 18). This process finally creates uni-monotone polygons.

```

Input:
  n vertices of a Polygon, where
  first is the first vertex ID of a polygon
  last is the last vertex ID of a polygon
Output:
  k triangles (TrP), where  $k = \sum_{i=0}^{n-1} k_i$  and n = number of uni-monotone polygons
Algorithm:[FM84]
  TrP_UMP_TP(first, last)
  {
    current_vertex = first
    while not current_vertex.done do
    {
      current_vertex.done = TRUE
      bottom_vertex = diagonal(current_vertex)
      if bottom_vertex not NULL then
      {
        save_next = next(current_vertex)
        save_prev = prev(bottom_vertex)
        next(current_vertex) = bottom_vertex
        prev(current_vertex) = current_vertex
        trapezoid(current_vertex) = NULL
        TrP_UMP_TP(bottom_vertex, current_vertex)
        current_vertex.done = FALSE
        bottom_vertex.done = FALSE
        next(current_vertex) = save_next
        prev(bottom_vertex) = save_prev
        next(bottom_vertex) = current_vertex
        prev(current_vertex) = bottom_vertex
        TrP_UMP_TP(current_vertex, bottom_vertex)
        return
      }
      else
        current_vertex = next(current_vertex)
    }
  }
  UMP_TP(first, last)
}

```

Figure 17. Algorithm to Transform the TrP into a UMP then into a TP [FM84].

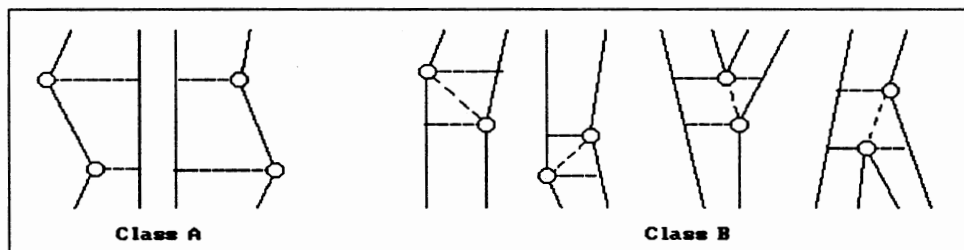


Figure 18. Two Classes of Trapezoids.

Implementation Detail

Diagonal() is a function (with a vertex input) which returns the value of:

- Bottom vertex of the trapezoid (which its top vertex is vertex input), if top and bottom vertices do not share an edge of the trapezoid.

- NULL, if top and bottom vertices share an edge of the trapezoid or if there is no trapezoid pointed at by the vertex input [FM84].

When this process calls for the TrP_UMP() function with the input of a polygon, that polygon is actually a uni-monotone polygon.

3.3 Transform the Uni-Monotonized Polygon into a Triangulated Polygon

The part of the Fournier and Montuno's triangulation algorithm that transforms the Uni-Monotonized Polygon into a Triangulated Polygon is shown in Figure 19.

```

Input: 1 UMP (contains  $n_i$  vertices), where  $0 \leq i < m-1$  and  $m =$  number of umps
Output:  $k_i$  triangles (where each triangle contains vertex1,
        vertex2, and vertex3)
Note:
vertex.prev - previous vertex
vertex.curr - current vertex
vertex.next - next vertex
vertex.stack[] - vertex on the stack
Algorithm: UMP_TP()

vertex.current = vertex (the second highest vertex in UMP)
while (number of vertex  $\geq$  3)
{
  if angle of (vertex.prev, vertex.curr, vertex.next)  $\leq$  180°
  {
    Build Triangle (vertex.prev, vertex.curr, vertex.next)
    remove vertex.curr
    decrease number of vertex by 1
    if (number of stack vertex  $>$  0)
      vertex.curr = vertex.stack[--number of stack vertex]
    else
      vertex.curr = vertex.next
  }
  else
  {
    vertex.stack[++number of stack vertex] = vertex.curr
    vertex.curr = vertex.next
  }
}

```

Figure 19. Algorithm to Transform the UMP into a TP [FM84].

A uni-monotone polygon contains n vertices labeled u_0, u_1, \dots, u_{n-1} . The uni-monotone polygon is divided into:

- the highest y-coordinate vertex (u_0) and the lowest y-coordinate vertex (u_{n-1}); the edge between this two vertices is called MainEdge.
- Intermediate vertices (all vertices except those that have the highest and lowest y-coordinates) denoted as u_i , where $1 \leq i \leq n-2$.

Only the intermediate vertices of the uni-monotone polygon are processed, so this process must start from the second highest vertex (u_1) and continue through the second lowest vertex (u_{n-2}).

The internal angle of each vertex is checked; the internal angle is defined by the adjacent edges of the current vertex (i.e., the angle that is formed by the previous vertex, the current vertex, and the next vertex). If the angle is less than or equal to 180° , then a triangle is built with those vertices (the previous vertex, the current vertex, and the next vertex), the current vertex is removed, and the next vertex is processed. If the angle is greater than 180° , then the current vertex is put in a stack to be processed later. The previous vertex, the current vertex, and the next vertex cannot build a triangle, if the internal angle is greater than 180° , because the triangle will be outside of the boundary of the uni-monotone polygon. The uni-monotone polygon is processed until it is broken up into a collection of one or more triangles.

CHAPTER IV

MODIFIED TRIANGULATION ALGORITHM

The modified triangulation algorithm is similar to the Fournier and Montuno's triangulation algorithm, with the following exceptions:

1. In the Fournier and Montuno algorithm, all trapezoids are formed before building a UMP (see Figure 8); in the modified algorithm, each completed trapezoid is immediately processed to form a UMP (see Figure 20).
2. In the Fournier and Montuno's algorithm, the step to transform the TrP into a UMP uses recursion; the modified algorithm avoids using recursion (this discussion will be explained in detail in the next chapter).

The differences between the original Fournier and Montuno's and modified triangulation algorithms are described below.

4.1 Transform the Simple Polygon into a Trapezoidized Polygon

This algorithm is almost the same as the algorithm which transforms the Simple Polygon into a Trapezoidized Polygon in the Fournier and Montuno's Triangulation algo-

rithm. The difference lies in handling the completion of a trapezoid.

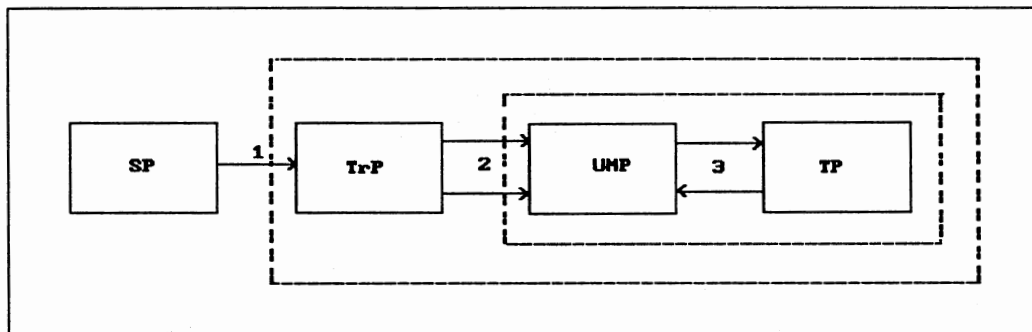


Figure 20. Execution Sequence of the Modified Algorithm.

In the Fournier Montuno's triangulation algorithm, after all trapezoids are completed, the process of transforming the Trapezoidized Polygon into a Uni-Monotonized Polygon begins (see Figure 11).

In the modified triangulation algorithm each completion of a trapezoid is directly transformed to build Uni-Monotone Polygons (see Figure 21, with sign *).

4.2 Transform the Trapezoidized Polygon into a Uni-Monotonized Polygon

Every time one trapezoid structure is completed, this step is executed by beginning a uni-monotone polygon based on the information of the completed trapezoid or appending the completed trapezoid to an existing uni-monotone polygon. The algorithm for this step is shown in Figure 22.

```

Input: n vertices (v0, v1, ..., vn-1) of a simple polygon
Output: k triangles  $\sum_{i=0}^{m-1} k_i$  and m = number of umps
       where  $k = \sum_{i=0}^{m-1} k_i$  and m = number of umps
Note:
vi - current vertex
eL - left edge of the current vertex (vi)
eR - right edge of the current vertex (vi)
VT - top vertex of the trapezoid structure
VB - bottom vertex of the trapezoid structure
EL - left edge of the trapezoid structure
ER - right edge of the trapezoid structure
Trapz ID - ID of Trapezoid searched
Algorithm: SP_TrP()
sort all vertices (in y-coordinate, max to min)
for each vertex (in the order of sort)
{
  switch (vertex type of current vertex <vi>)
  {
    case I:
      Search Edge of vi in 2-3 tree (&TrapzID, eL, eR)
      Complete the trapezoid with vi as bottom vertex
      (TrapzID)
      *-----TrP_UMP_TP (TrapzID)
      Remove trapezoid from 2-3 tree (TrapzID)
      Insert new trapezoid into 2-3 tree (vi, EL, eR)
    case II:
      Search vi location in 2-3 tree (&TrapzID)
      If vi is within an active trapezoid (type IIa)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID)
      *-----TrP_UMP_TP (TrapzID)
      Remove trapezoid from 2-3 tree (TrapzID)
      Insert new trapezoid into 2-3 tree (vi, EL, eL)
      Insert new Trapezoid into 2-3 tree (vi, eR, ER)
      }
      Else (type IIb)
        Insert new trapezoid into 2-3 tree (vi, eL, eR)
    case III:
      Search Edge of vi in 2-3 tree (&TrapzID, eL, eR)
      If eL and eR belong to the same trapezoid (IIIa)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID)
      *-----TrP_UMP_TP (TrapzID)
      Remove trapezoid from 2-3 tree (TrapzID)
      }
      Else (type IIIb)
      {
        Complete the trapezoid with vi as bottom vertex
        (TrapzID1)
      *-----TrP_UMP_TP (TrapzID1)
        Complete the trapezoid with vi as bottom vertex
        (TrapzID2)
      *-----TrP_UMP_TP (TrapzID2)
        Remove trapezoid from 2-3 tree (TrapzID1)
        Remove trapezoid from 2-3 tree (TrapzID2)
        Insert new trapezoid into 2-3 tree (vi, EL", ER")
      }
    }
  }
}

```

Figure 21. Algorithm to Transform SP into a TrP, into a UMP, and then into a TP.

The choices of beginning a uni-monotone polygon or appending to an existing uni-monotone polygon depend on the type of the top vertex (VT) of the completed trapezoid (i.e., I, IIa, IIb, IIIa, or IIIc) and the type of the diagonal of the completed trapezoid (i.e., internal diagonal or external diagonal). See Section 1.2 for the definitions

of these terms.

```

input: Trapezoid (UT, UB, EL, ER)
output:  $k_i$  triangles, where  $0 \leq i \leq m-1$  and  $m =$  number of umps
Algorithm:
  TrP_UMP_TP (TrapzID)
  {
    Switch (edge UT-UB)
    {
      case EXTERNAL_EDGE:
        if current vertex type = IIb
          Build_UMP (the longer one between EL and ER in Y-coordinate)
          Append_UMP (EL or ER, UT-UB)
      case INTERNAL_EDGE:
        if current vertex type = IIb
          {
            Build_UMP (EL)
            Build_UMP (ER)
          }
          Append_UMP (EL, UT-UB)
          Append_UMP (ER, UT-UB)
    }
  }

  Append_UMP (vertex, MainEdge)
  {
    Search_MainEdge_Location in 2-3 tree (UMP_ID, vertex)
    if UMP_ID = NULL
      {
        Build_UMP (vertex)
        Append_UMP (vertex, MainEdge)
      }
    else
      Attach vertex to UMP_ID
      if (UMP is complete)
        UMP_TP ()
  }
}

```

Figure 22. Algorithm to Transform TrP into a UMP, and then into a TP.

If the trapezoid just completed has:

1. a top vertex of type IIb and an external diagonal, begin a new uni-monotone polygon with the left or right side edge of the trapezoid as the main edge of the uni-monotone polygon.
2. a top vertex of type IIb and an internal diagonal, begin two new uni-monotone polygons with the left and right side edge of the trapezoid as the main edge of the uni-monotone polygons.
3. a top vertex of type I, IIa, IIIa, or IIIb and an external diagonal, append the trapezoid's top vertex as an intermediate vertex to an existing uni-monotone

polygon that has a main edge equal to the left or right side edge of the trapezoid. If no such uni-monotone polygon exists, begin a new uni-monotone polygon with the left or right side edge of the trapezoid as the main edge of the uni-monotone polygon.

4. a top vertex of type I, IIa, IIIa, or IIIb and an internal diagonal, append the trapezoid's top vertex as an intermediate vertex to two existing uni-monotone polygons such that:
 - a. one of the uni-monotone polygons has a main edge equal to the left side edge of the trapezoid. If no such uni-monotone polygon exists, begin a new uni-monotone polygon with the left side edge of the trapezoid as the main edge of the uni-monotone polygon.
 - b. another uni-monotone polygon has a main edge equal to the right side edge of the trapezoid. If no such uni-monotone polygon exists, begin a new uni-monotone polygon with the right side edge of the trapezoid as the main edge of the uni-monotone polygon.

If the top vertex of the trapezoid being processed is also the bottom vertex of the main edge of the uni-monotone polygon, then that uni-monotone polygon is complete and should be triangulated immediately.

Based on the definition of a uni-monotone polygon (see Section 1.2), the vertex types on the uni-monotone polygon

can be identified:

- v_0 (the highest y-coordinate vertex) must be of vertex type IIb
- v_{n-1} (the lowest y-coordinate vertex) must be of vertex type IIIa
- v_i , where $1 \leq i \leq n-2$, (the intermediate vertices) must be of vertex type I, never of type II or type III. Also the y-coordinate of v_i is always higher than y-coordinate of v_{i+1} .

To build a uni-monotone polygon, at least three vertices (two vertices as a main edge, and at least one vertex as an intermediate vertex) are needed. The main edge must have the maximum and minimum y-coordinate of that uni-monotone polygon.

4.3 Transform the Uni-Monotonized Polygon into a Triangulated Polygon

This algorithm is analogous to the algorithm which transforms the Uni-Monotonized Polygon into a Triangulated Polygon of Fournier and Montuno's Triangulation algorithm.

CHAPTER V

COMPARISON OF FOURNIER AND MONTUNO'S AND THE MODIFIED TRIANGULATION ALGORITHMS

In the previous chapters, the Fournier and Montuno's and the modified triangulation algorithms were discussed separately. This chapter examines the differences between the two algorithms by comparing the processing flows of the algorithms, their simplicities, and their run-time complexities.

5.1 Processing Flow of the Algorithms

The following is a comparison of the processing flows of the Fournier and Montuno's triangulation algorithm and the modified triangulation algorithm. Figure 23 outlines the processing flows of the two algorithms (see Section 1.2 for the explanation of the notations).

Up to the point where BuildTrapz() is invoked, both the Fournier and Montuno's and the modified triangulation algorithms are the same. After that, they take two different directions, because the Fournier and Montuno's triangulation algorithm creates uni-monotone polygons after all trapezoids are completed, while the modified triangulation algorithm creates uni-monotone polygons after each trapezoid is com-

pleted.

Fournier and Montuno's Triangulation Algorithm	Modified Triangulation Algorithm
<pre> main() { SP = read_SP() sort_SP (SP) TrP = SP_TrP (SP) TP = TrP_UMP_TP (TrP) } SP_TrP (SP) { for i = 1 to n trp = BuildTrapz (vertex) } TrP_UMP_TP (TrP1) { TrP_UMP_TP (TrP2) TrP_UMP_TP (TrP3) tps = UMP_TP (ump) } </pre>	<pre> main() { SP = read_SP() sort_SP (SP) TP = SP_TrP_UMP_TP (SP) } SP_TrP_UMP_TP (SP) { for i = 1 to n { trp = BuildTrapz (vertex) if (trp = Compl_Trapz) tps = TrP_UMP_TP (trp) } } TrP_UMP_TP (trp) { ump = BuildUmp (trp) if (ump = ComplUmp) tps = UMP_TP (ump) } </pre>
<p>Notation:</p> <p>n means number of vertices in SP</p> <p>BuildTrapz () means SP_TrP which is an algorithm to transform SP into a TrP.</p> <p>BuildUMP () means an algorithm to transform the TrP into a UMP.</p>	

Figure 23. The Processing Flow of Fournier and Montuno's and the Modified Triangulation Algorithms.

The TrP_UMP_TP() procedure of each algorithm takes a different approach in transforming a TrP to a UMP. Fournier and Montuno's triangulation algorithm uses recursion, while the modified triangulation algorithm avoids the use of recursion. The details will be explained in the next section.

5.2 Simplicity

The `TrP_UMP_TP()` procedure in Fournier and Montuno's triangulation algorithm is executed recursively. A recursive procedure (i.e., a procedure that calls itself) can be complex and difficult to understand. Moreover, in the actual algorithm this procedure calls itself twice. This factor makes the algorithm harder to understand and to follow.

The modified triangulation algorithm avoids the use of recursion; thus it is simpler than the Fournier and Montuno's triangulation algorithm.

5.3 Run-Time Complexity

In the following discussion, the run-time complexities of both algorithms are analyzed.

Both algorithms assume that the Simple Polygon is already provided, so the run-time complexity of `read_SP()` procedure is of no concern.

Fournier and Montuno's Triangulation Algorithm

This algorithm executes the `sort_SP()`, `SP_TrP()`, and `TrP_UMP_TP()` procedures.

The `sort_SP()` procedure can be done in $O(n \log n)$ [FM84].

The `SP_TrP()` procedure runs in $O(n \log n)$. The main 'for' loop is executed n times. Each of its steps processes

the BuildTrapz() procedure which uses a 2-3 tree to store (search and insert) and delete trapezoids. These operations each take $O(\log r)$, where r is the number of leaves in the tree. Because $r \leq n$, the whole loop is $O(n \log n)$ [FM84].

The TrP_UMP_TP() procedure runs in $O(n)$ [FM84]. The UMP_TP() procedure runs in $O(m)$, where m is the number of vertices of a uni-monotone polygon. Because $m \leq n$, the TrP_UMP_TP() procedure runs in $O(n)$ [FM84].

Therefore the run-time complexity is $O(n \log n + n \log n + n)$, which asymptotically is the same as $O(n \log n)$. Thus the entire Fournier and Montuno's triangulation algorithm runs in $O(n \log n)$ time [FM84].

The full analysis of this algorithm is provided by Fournier and Montuno [FM84].

The Modified Triangulation Algorithm

This algorithm executes the sort_SP() and SP_TrP_UMP_TP() procedures.

Sorting has a run-time complexity of $O(n \log n)$, so the sort_SP() procedure is $O(n \log n)$.

The SP_TrP_UMP_TP() procedure runs in $O(n \log n)$. The main 'for' loop is executed n times. Each of its steps executes the BuildTrapz() procedure and in certain condition also executes TrP_UMP_TP() procedure.

The BuildTrapz() procedure uses 2-3 tree to store (search and insert) and delete trapezoids. These operations each take $O(\log r)$, where r is the number of leaves in the

tree. Because $r \leq n$, the `BuildTrapz()` procedure runs in $O(\log n)$.

The `TrP_UMP_TP()` procedure runs in $O(\log n)$. It executes `BuildUmp()` procedure and in one condition also executes `UMP_TP()` procedure.

The `BuildUmp()` is the same run-time complexity as the `BuildTrapz()` procedure which runs in $O(\log n)$.

The `UMP_TP()` runs in $O(1)$. Assume that the run time of this procedure in a loop is z_i . Then the run time of `SP_TrP_UMP_TP()` procedure is

$$\sum_{i=0}^{n-1} (\log n + \log n + z_i)$$

The possible value of $\sum_{i=0}^{n-1} z_i =$

1. $0 + 0 + \dots + 0 + n = n$, that means the simple polygon is a uni-monotone polygon.
2. $0 + 0 + 3 + 3 + \dots + 3 + 3 = 3(n-2)$, that means all uni-monotone polygons are triangle.
3. z_i never be n more than one, so $\sum z_i \neq n^2$. On each loop (each vertex) can be used by at most three uni-monotone polygons, therefore each vertex is processed at most three times, thus $\sum z_i = O(3n)$.

The z_i is a constant, and the `UMP_TP()` runs in $O(1)$.

Therefore the `TrP_UMP_TP()` procedure runs in $O(\log n) + O(1)$, which in the worst case becomes $O(\log n)$. The `SP_TrP_UMP_TP()` procedure runs in $O(n(\log n + \log n))$, which in the worst case becomes $O(n \log n)$. Thus the entire

Modified triangulation algorithm runs in $O(n \log n)$.

Comparison

From the above run-time complexity analysis of both algorithms, the Fournier and Montuno's triangulation algorithm has the same order run-time complexity to the modified triangulation algorithm. But in implementation the modified triangulation algorithm is faster, though not significantly; they only differ by a multiplicative constant (see Appendix C). The implementation uses various data which are input to both algorithms, and the run times of both algorithms are analyzed.

There are some possible reasons about the result implementation test of run time unit of both algorithms, where the modified triangulation algorithm runs faster than the Fournier and Montuno's triangulation algorithm:

1. Like the analysis above, both algorithms runs in the same big 'Oh' run-time complexity (i.e., $O(n \log n)$), but they are different by multiplicative constant).
2. Implementation algorithm on modified triangulation algorithm is the better implementation; implementation algorithm on Fournier and Montuno's triangulation algorithm is not the good implementation.
3. The Fournier and Montuno's triangulation algorithm uses recursive function on one part of the algorithm, while the modified triangulation algorithm avoid using recursive function. Two algorithms that has the same time

complexity, but one using recursive function and the other using iteration, the one that use iteration will runs in the same order as (or slower in detail than) the one that use recursive function in analysis, but will runs faster in implementation [S81].

CHAPTER VI

SUMMARY AND FUTURE WORK

Triangulation algorithm should be simple and run fast. The modified triangulation algorithm presented here is simpler (it avoids using recursion) and runs faster (though not significantly) than Fournier and Montuno's triangulation algorithm.

Possible future work on the triangulation algorithm:

1. Try to reduce the run-time complexity. It is conjectured that the run-time complexity of the triangulation algorithm could be $O(n)$, where n is the number of vertices in a simple polygon. That means that the triangulation algorithm is faster than a typical efficient sorting algorithm [TV88].
2. Use parallel computing methods. Ideally, in a parallel environment, the algorithm should run faster by a factor of m , where m is the number of processors. If $m \geq n$, the algorithm should run faster by a factor of n .
3. Find a connection between the problem of triangulation of a simple polygon and determination of edge-vertex visibility [FM84, TV88].
4. Find the intersection of two n -gons [TV86].

REFERENCES

- [AT81] Avis, D. and G. T. Toussaint. "An Efficient Algorithm for decomposing a polygon into star-shaped polygons." Pattern Recogn., Vol. 13, No. 6 (1981), pp. 395-398.
- [CD79] Chazelle, B. and D. Dobkin. "Decomposing a Polygon into Its Convex Parts." Proceedings of the 11th Symposium on Theory of Computing, ACM, New York, 1979, pp. 38-48.
- [FM84] Fournier, A. and D. Y. Montuno. "Triangulating Simple Polygons and Equivalent Problems." ACM Trans. on Graphics, Vol. 3, No. 2 (April 1984), pp. 153-174.
- [FR82] Fussel, D. and B. D. Rathi. "A VLSI-Oriented architecture for Real-Time Raster Display of Shaded Polygons." Proceedings of Graphics Interface '82, National Research Council of Canada, Toronto, Ontario, 1982, pp. 373-380.
- [FS73] Fix, G. and G. Strang. An Analysis of the Finite Element Method Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
- [GJPT78] Garey, M. R., D. S. Johnson, F. P. Preparata, and R. E. Tarjan. "Triangulating a Simple Polygon." Info. Processing Letter, Vol. 7, No. 4 (June 1978), pp. 175-180.
- [HM83] Hertel, S. and K. Mehlhorn. "Fast Triangulation of Simple Polygons." Proceedings of the 1983 International Conference on the Foundations of Computer Science, IEEE, Los Angeles, 1983, pp. 207-218.
- [HMRT86] Hoffman, Kurt, Kurt Mehlhorn, Pierre Rosenstiehl, and R. E. Tarjan. "Sorting Jordan Sequences in Linear Time Using Level Linked Search Trees." Information and Control, Vol. 68 (1986), pp. 170-184.
- [M76] McLain, D. H. "Two-Dimensional Interpolation from Random Data." Computer Journal, Vol. 19 (1976), pp. 178-181.

- [K83] Kirkpatrick, D. G. "Optimal Search in Planar Subdivisions." SIAM J. Computing, Vol. 12, No. 1 (February 1983), pp. 28-35.
- [K85] Keil, J. Mark. "Decomposing a Polygon into Simpler Components." SIAM J. Computing, Vol. 14, No. 4 (November 1985), pp. 799-817.
- [L77] Lloyd, E. L. "On Triangulations of a Set of Points in the Plane." Proceedings of the 18th Annual Symposium on the Foundations of Computer Science, IEEE, Los Angeles, 1977, pp. 228-240.
- [L81] Lee, D. T. "Shading of Regions on Vector Display Devices." ACM Computer Graphics, Vol. 15, No. 3 (July 1981), pp. 34-44.
- [LP77] Lee, D. T. and F. P. Preparata. "Location of a Point in a Planar Subdivision and Its Applications." SIAM J. Computing, Vol. 6, No. 3 (September 1977), pp. 594-606.
- [S75] Shamos, M. I. "Geometric Complexity." Proceedings of the 7th Annual Symposium on Theory of Computing, ACM, New York, 1975, pp. 224-233.
- [S78] Schachter, B. "Decomposition of Polygons into Convex Sets." IEEE Trans. Comput., Vol. C-27, No. 11 (November 1978), pp. 1078-1082.
- [S81] Sahni, Sartaj. Concepts in Discrete Mathematics The Camelot Publishing Company, Fridley, Minnesota, 1981.
- [TV88] Tarjan, R. E. and C. J. Van Wyk. "An $O(n \log \log n)$ Time Algorithm for Triangulating a Simple Polygon." SIAM J. Computing, Vol. 17, No. 1 (February 1988), pp. 143-178.
- [W70] Watkins, G. S. "A Real-Time Visible Surface Algorithm." Technical Report UTEC-CSc-70-101, Computer Science Department, Univ. of Utah 1970, NTIS AD-762 004.

APPENDIXES

APPENDIX A

DETAILS OF THE MODIFIED TRIANGULATION ALGORITHM

This is a sample run of the modified triangulation algorithm from creating a simple polygon to modifying the simple polygon into a triangulated polygon.

Create a Simple Polygon

A simple polygon is created with 12 vertices in a batch input data file (see Appendix C for the actual batch input data file and Appendix B for explanation how to run the program). A vertex is identified by its x and y-coordinates. The type of each vertex is then identified (see Figure 24). The simple polygon is shown in Figure 25.

Vertex ID	(X,Y) coordinates	Vertex Type
0	(8,20)	II
1	(10,16)	III
2	(12,18)	II
3	(14,12)	III
4	(18,14)	I
5	(16,15)	I
6	(22,17)	II
7	(18,10)	I
8	(20, 4)	III
9	(10, 8)	II
10	(2, 6)	III
11	(6,11)	I

Figure 24. The Vertices of a Simple Polygon.

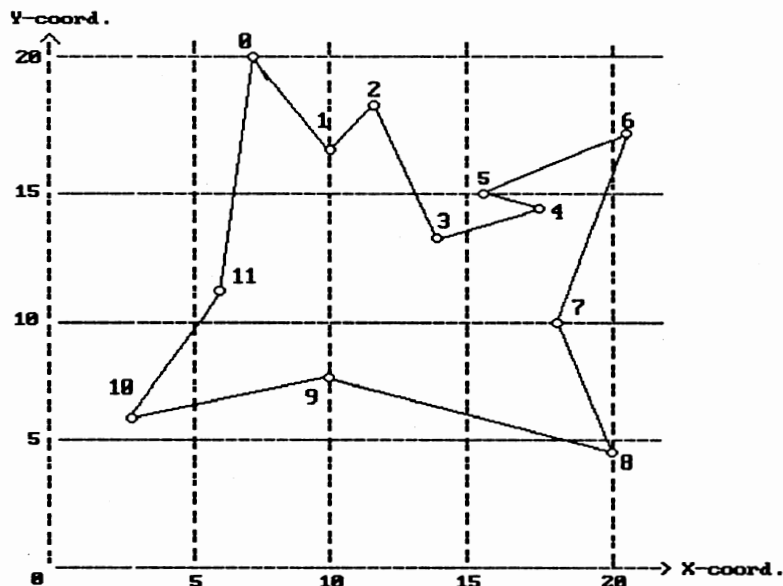


Figure 25. A Simple Polygon.

Transform the Simple Polygon into
a Trapezoidized Polygon

After a simple polygon is created, each vertex (in y-coordinate descending order) is processed to transform the

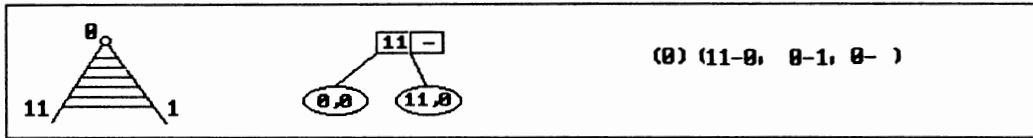
simple polygon into a trapezoidized polygon.

Notation on Figures 26a, 26b, and 26c:

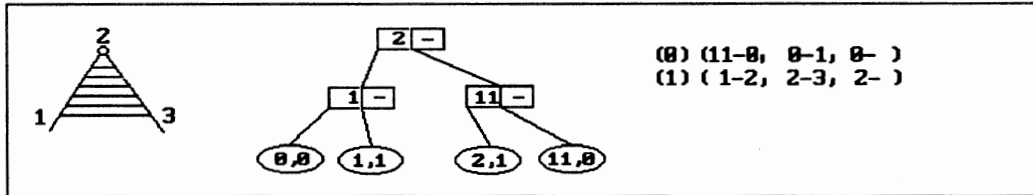
- the left part identifies the current vertex being processed (marked with a circle) with its previous and its next vertices.
- the middle part represents the 2-3 tree containing the current incomplete trapezoids. The notation on the interior node is (Diagonal Edge), where Diagonal Edge means the diagonal edge of the trapezoid. The notation on the leaf node is (Diagonal Edge, ID), where Diagonal Edge means the same as the above definition and ID means the ID of the trapezoid.
- the right part represents a list of all incomplete and complete trapezoids processed so far. Complete trapezoids are marked with an * in front of them. The notation is (ID) (EL, ER, VT_VB), where ID means the ID of the trapezoid, EL and ER mean the left side edge and the right edge of the trapezoid, respectively, VT-VB means the diagonal edge of the trapezoid that connects the top vertex and the bottom vertex of that trapezoid.

The following is the outline of what happens to each vertex (Figures 26a, 26b, and 26c).

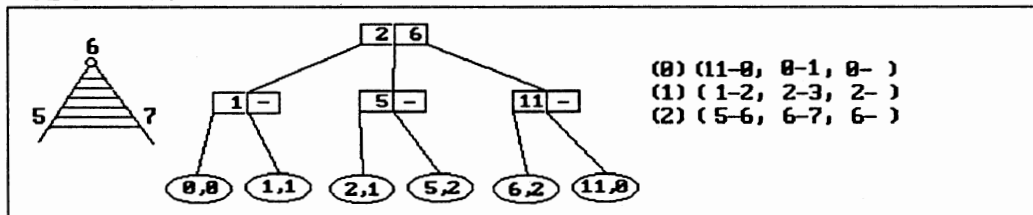
Vertex 0:



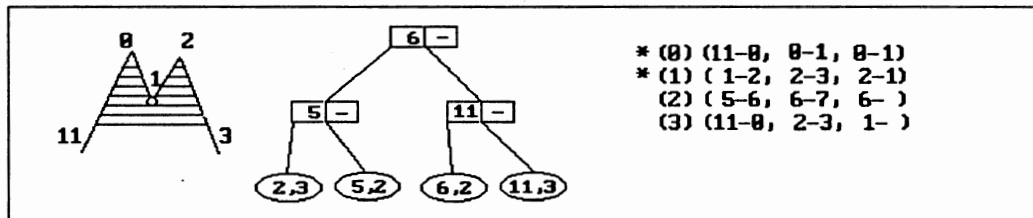
Vertex 2:



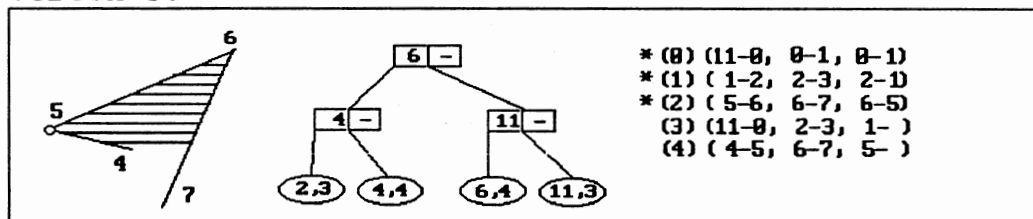
Vertex 6:



Vertex 1:



Vertex 5:



Vertex 4:

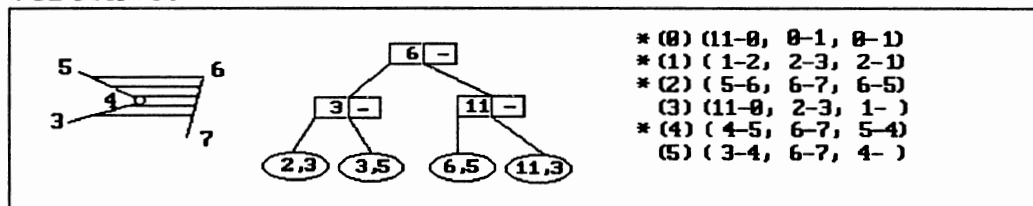



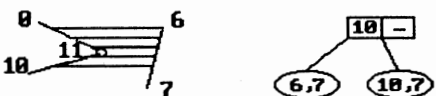
Figure 26a. The Transformation of a Simple Polygon into a Trapezoidized Polygon.

Vertex 3:




- * (0) (11-0, 0-1, 0-1)
- * (1) (1-2, 2-3, 2-1)
- * (2) (5-6, 6-7, 6-5)
- * (3) (11-0, 2-3, 1-3)
- * (4) (4-5, 6-7, 5-4)
- * (5) (3-4, 6-7, 4-3)
- * (6) (11-0, 6-7, 3-)

Vertex 11:



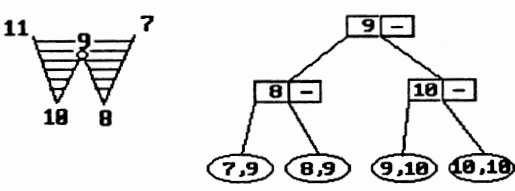
- * (0) (11-0, 0-1, 0-1)
- * (1) (1-2, 2-3, 2-1)
- * (2) (5-6, 6-7, 6-5)
- * (3) (11-0, 2-3, 1-3)
- * (4) (4-5, 6-7, 5-4)
- * (5) (3-4, 6-7, 4-3)
- * (6) (11-0, 6-7, 3-11)
- * (7) (10-11, 6-7, 11-)

Vertex 7:



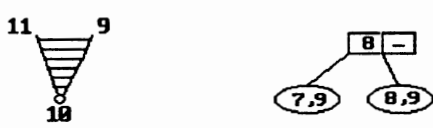
- * (0) (11-0, 0-1, 0-1)
- * (1) (1-2, 2-3, 2-1)
- * (2) (5-6, 6-7, 6-5)
- * (3) (11-0, 2-3, 1-3)
- * (4) (4-5, 6-7, 5-4)
- * (5) (3-4, 6-7, 4-3)
- * (6) (11-0, 6-7, 3-11)
- * (7) (10-11, 6-7, 11-7)
- * (8) (10-11, 7-8, 7-)

Vertex 9:



- * (0) (11-0, 0-1, 0-1)
- * (1) (1-2, 2-3, 2-1)
- * (2) (5-6, 6-7, 6-5)
- * (3) (11-0, 2-3, 1-3)
- * (4) (4-5, 6-7, 5-4)
- * (5) (3-4, 6-7, 4-3)
- * (6) (11-0, 6-7, 3-11)
- * (7) (10-11, 6-7, 11-7)
- * (8) (10-11, 7-8, 7-9)
- * (9) (7-8, 8-9, 9-)
- * (10) (9-10, 10-11, 9-)

Vertex 10:



- * (0) (11-0, 0-1, 0-1)
- * (1) (1-2, 2-3, 2-1)
- * (2) (5-6, 6-7, 6-5)
- * (3) (11-0, 2-3, 1-3)
- * (4) (4-5, 6-7, 5-4)
- * (5) (3-4, 6-7, 4-3)
- * (6) (11-0, 6-7, 3-11)
- * (7) (10-11, 6-7, 11-7)
- * (8) (10-11, 7-8, 7-9)
- * (9) (7-8, 8-9, 9-)
- * (10) (9-10, 10-11, 9-10)

Figure 26b. The Transformation of a Simple Polygon into a Trapezoidized Polygon

Vertex 8:

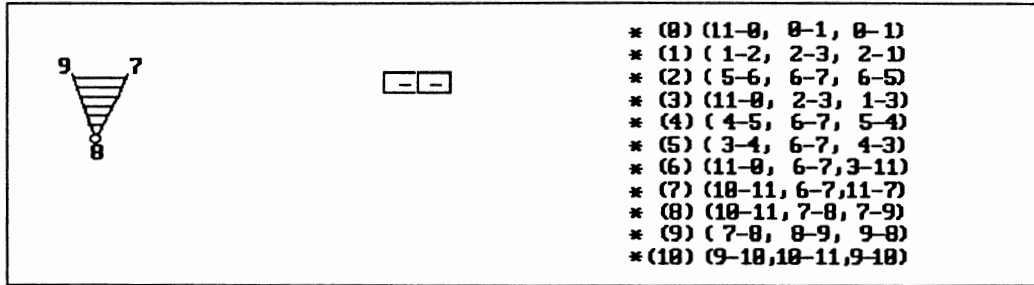


Figure 26c. The Transformation of a Simple Polygon into a Trapezoidized Polygon

In Figure 26c, vertex 8 (the last vertex to be processed, in y-coordinate sorted order) contains a list of all resulting trapezoids, as illustrated in Figure 27.

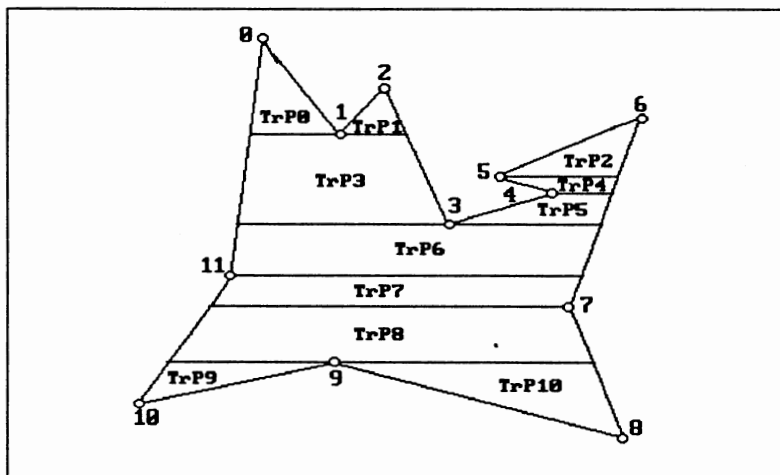


Figure 27. The Trapezoidized Polygon

Transform the Trapezoidized Polygon into
a Uni-Monotonized Polygon

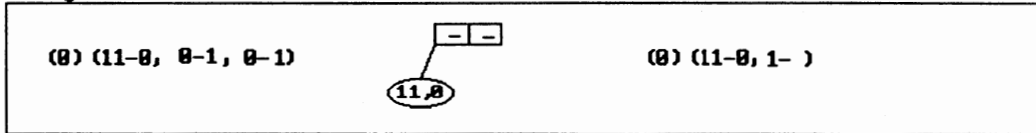
The next step is processing each completed trapezoid to transform it into uni-monotone polygons.

Notation on Figures 28a and 28b:

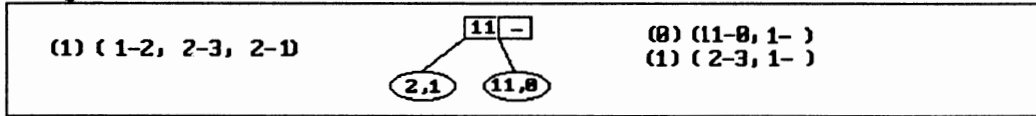
- the left part represents the current complete trapezoid now being processed. The notation is the same as the notation for the right part in the previous section.
- the middle part represents the 2-3 tree containing presently existing uni-monotone polygons. The notation on the interior node is (Main Edge), where Main Edge means the main edge of the uni-monotone polygon. The notation on the leaf node is (Main Edge, ID), where Main Edge means the same as the above definition and ID means the ID of the uni-monotone polygon.
- the right part represents a list of all incomplete and complete uni-monotone polygons processed so far. Complete uni-monotone polygons are marked with an * in front of them. The notation is (ID) (Main Edge, Intermediate vertices), where ID means the same as the above definition, Main Edge means the same as the above definition, and Intermediate Vertices means the intermediate vertices of the uni-monotone polygon.

The following is the outline of what happens to each trapezoid (Figure 28a and 28b).

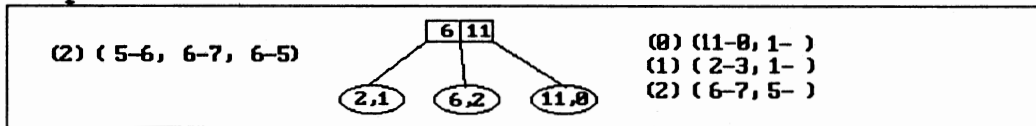
Trapezoid 0:



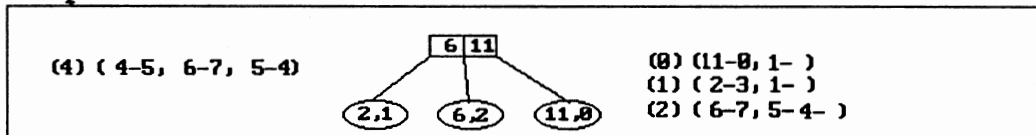
Trapezoid 1:



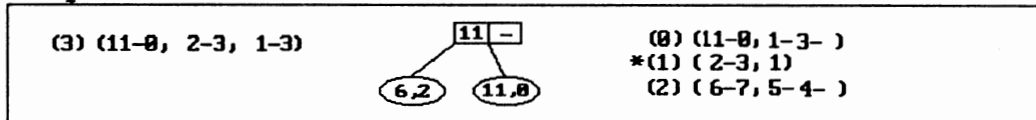
Trapezoid 2:



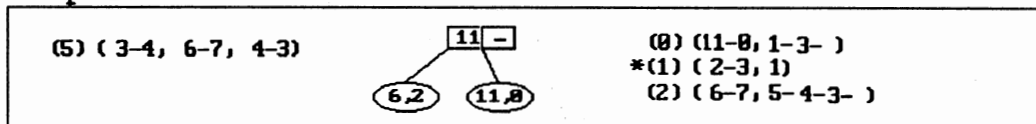
Trapezoid 4:



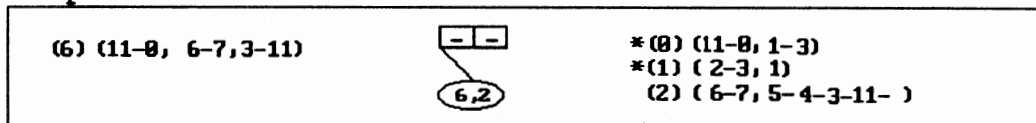
Trapezoid 3:



Trapezoid 5:



Trapezoid 6:



Trapezoid 7:

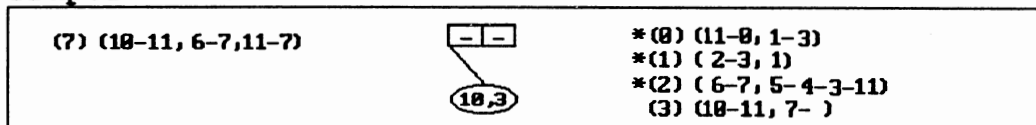
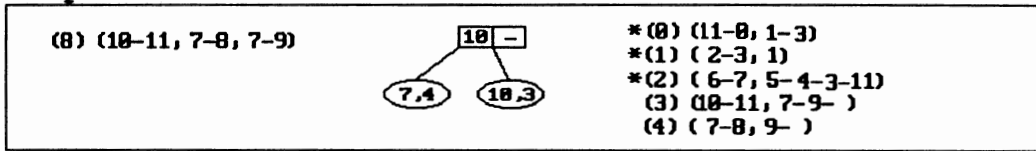
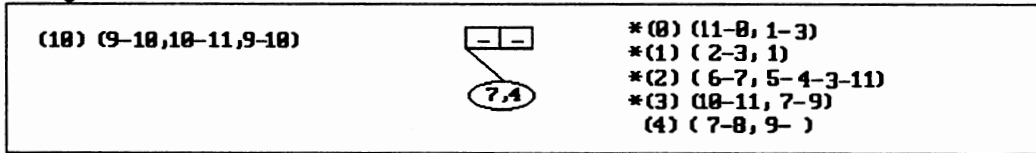


Figure 28a. The Transformation of a Trapezoidized Polygon into a Uni-Monotonized Polygon.

Trapezoid 8:



Trapezoid 10:



Trapezoid 9:

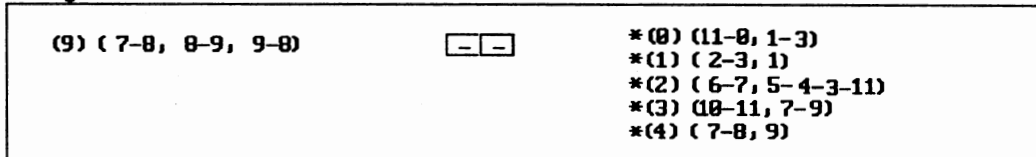


Figure 28b. The Transformation of a Trapezoidized Polygon into a Uni-Monotonized Polygon.

In Figure 28b, trapezoid 9 (the last trapezoid to be processed) contains a list of all resulting uni-monotone polygons, as illustrated in Figure 29.

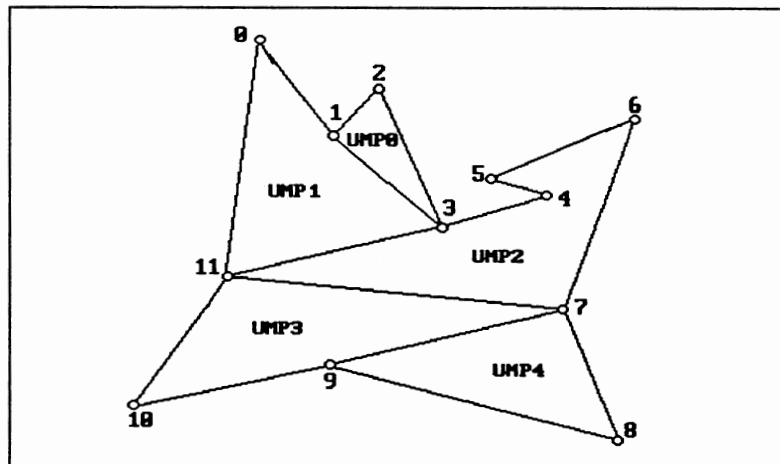


Figure 29. The Uni-Monotonized Polygon.

Transform the Uni-Monotonized Polygon
into a Triangulated Polygon

This step processes each completed uni-monotone polygon to form triangles.

Notation on Figure 30:

- the left part (Uni-Monotone Polygon column) represents the uni-monotone polygon being processed. The notation is the same as the notation for the right part in the previous section.
- the Main Edge column represents the main edge of the uni-monotone polygon being processed.
- the Available Vertex column represents a list of all vertices (of the uni-monotone polygon being processed) that are not yet deleted from that uni-monotone polygon.
- the Current Vertex column represents an intermediate vertex (of a uni-monotone polygon being processed) that are now being processed.
- the Previous Vertex column represents a vertex (of a uni-monotone polygon being processed) that is located before the current vertex (in the sorted sequence of Y-coordinates).
- the Next Vertex column represents a vertex (of a uni-monotone polygon being processed) that is located after the current vertex (in the sorted sequence of Y-coordinates).
- the " $< \pi$ " column represents a flag that indicates whether the angle between the Previous Vertex, the Current Vertex,

and the Next Vertex is less than 180° .

- the Tri ID column represents a triangle ID.
- the Stack Vertex column represents the vertices that have been processed whose interior angle between the Previous Vertex, the Current Vertex, and the Next Vertex is more than 180° . These vertices will be processed again in the next process.

The following is an outline of what happens to each uni-monotone polygon (Figure 30).

Uni-Monotone Polygon	Main Edge	Available Vertex	Vertex			< W	Tri ID	Stack Vertex
			Current	Previous	Next			
* (1) (2-3, 1)	2-3	2,1,3	1	2	3	Y	8	-
* (8) (11-8, 1-3)	11-8	8,1,3,11	1	8	3	N		1
		8,1,3,11	3	1	11	Y	1	1
		8,1,11	1	8	11	Y	2	-
* (2) (6-7, 5-4-3-11)	6-7	6,5,4,3,11,7	5	6	4	Y	3	-
		6,4,3,11,7	4	6	3	N		4
		6,4,3,11,7	3	4	11	N		3,4
		6,4,3,11,7	11	3	7	Y	4	3,4
		6,4,3,7	3	4	7	Y	5	4
		6,4,7	4	6	7	Y	6	-
* (3) (18-11, 7-9)	18-11	11,7,9,18	7	11	9	Y	7	-
		11,9,18	9	11	18	Y	8	-
* (4) (7-8, 9)	7-8	7,9,8	9	7	8	Y	9	-

Figure 30. The Processing Transformation from Uni-Monotone Polygon to a Triangulated Polygon.

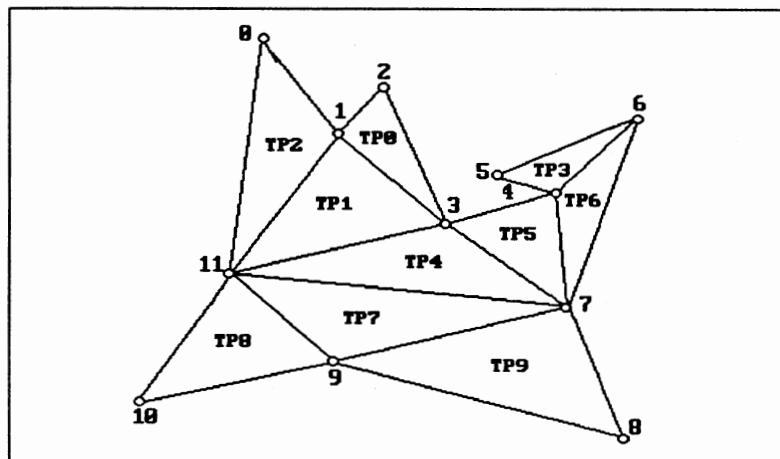


Figure 31. The Triangulated Polygon.

APPENDIX B

IMPLEMENTATION

Introduction

The program is called "TRIALG" (TRIangulation ALGo-rithm). This program implements the Fournier and Montuno's and Modified triangulation algorithms. The output of this program supports the correctness of the analysis in the previous chapters. The program is developed using Borland's Turbo C++ compiler, version 1.0, on a 16 megahertz IBM Personal System 2/30-80286 with an IBM 8512 color display monitor. The program is also developed using the C compiler on a VAXstation Model 5500 with 128 megabytes of memory, 2.5 gigabytes of disk storage, rated at 28 mips (million instructions per second) which utilize RISC (Reduces Instruction Set Computer).

The program displays the logo of TRIALG, then after any key is pressed, the program prompts the user to enter several input items.

Input

The values that the user is prompted for are:

1. Input mode:

- a. batch file input mode -- the user provides the name of a file that contains the description of a Simple Polygon. The correctness of this data will not be checked by the program, so the batch file must have the correct format and produce simple polygon(s). The batch input file can contain one or more simple polygons. The correct format is outlined in Figure 32.

```

*<name of the simple polygon> <number of the vertices>
<x-coor> <y-coor>
<x-coor> <y-coor>
...
**
*<name of the simple polygon> <number of the vertices>
<x-coor> <y-coor>
<x-coor> <y-coor>
...
**
...
...
...
*<name of the simple polygon> <number of the vertices>
<x-coor> <y-coor>
<x-coor> <y-coor>
...
***

```

Figure 32. Correct Input Format.

- b. interactive mode -- the user types the description of a Simple Polygon. The correctness of this data will not be checked by the program, so the input data must have the correct format and produce simple polygon(s). The correct input format is outlined in Figure 32 (i.e., the same as the batch file input format).
- c. random Simple Polygon generator -- the program generates the description of a Simple Polygon and checks if the vertices created are valid. The system will prompt the user for :

- the number of simple polygons,
- the maximum x and y coordinate vertex,
- the minimum x and y coordinate vertex, and
- how the number of vertices for each Simple Polygon will be specified:
 - . the user specifies them one by one,
 - . the user specifies the starting number and the incremental number, or
 - . the user specifies the range number by typing in the maximum and the minimum number (the program will generate the number of vertices randomly between minimum and maximum number specified).

2. The choice of the algorithm:

- a. Fournier and Montuno's Triangulation Algorithm,
- b. Modified Triangulation Algorithm, or
- c. Both Algorithms.

3. The choice of output files to be produced:

- a. give warnings before overwriting files? (Yes or No)
- b. the input data file (InpData.x), which contains all vertices information of simple polygons created (Yes or No)
- c. the debug file (debug.x), which contains the flow sequence of the program (Yes or No)
- d. the 2-3 tree debug operation file (debug.x), this question will appear only if the user say Yes on creating debug.x file (question c).
- e. the comparison triangles statistics file (TPstat.x),

- which contains the triangles results of the algorithms (Yes or No)
- f. the comparison time statistics file (TimeStat.x), which contains the bios run time of the algorithms (Yes or No)
 - g. the vertices result file (OutData.x), which contains all vertices of Simple Polygon, Uni-Monotone Polygon, and the Triangulated Polygon? (Yes or No)
4. The choices displayed:
- a. display the all elements polygons in detail (Yes or No), that is showing Simple Polygon, showing all uni-monotone polygons inside uni-monotonized polygon one by one, and showing triangles inside the uni-monotone polygon one by one. If the user say Yes the question b, c, d, e, and f will be skipped. This question (a) will not appear if the time complexity is to be analyzed (i.e., TimeStat.x file exists).
 - b. display the simple polygon (Yes or No).
 - c. display the uni-monotone polygons inside the uni-monotonized polygon one by one (Yes or No). This question will not appear if the time complexity is to be analyzed. If the user say Yes the question d will be skipped.
 - d. display the uni-monotonized polygon (Yes or No).
 - e. display the triangles inside the triangulated polygon one by one (Yes or No). This question will not

appear if the time complexity is to be analyzed. If the user say Yes the question f will be skipped.

- f. display the triangulated polygon (Yes or No).
- g. display run times of algorithms on a table (Yes or No). This question will appear if the time complexity is to be analyzed.
- h. display the run times of algorithms on a chart (Yes or No). This question will appear if the time complexity is to be analyzed.

After all inputs are entered, the program processes all inputs (using the Fournier and Montuno's Triangulation Algorithm or/and the Modified Triangulation Algorithm) and produce the output results, using graphics to show the triangulation process as it progresses (optional).

Output

The output of the program depends on the inputs entered, especially the part of the input on what will be displayed, either the result of the triangulated polygon and the performance data (run-time comparison) of the algorithms.

The run-time chart depicts the number of vertices of the simple polygons entered in increasing order (x-coordinate). It also averages the run times on the simple polygons that have the same number of vertices (y-coordinate). The run-time unit uses bios time (processor clock)

which is 18.2 units per second in PS-2/30-80286 IBM computer and 16.667 units per millisecond in DEC VAXStation 5500 (the bios run time of DEC VAXStation 5500 is displayed in run-time chart by multiples of 1000, so that the bios run time displayed is 16.667 units per second). Data from the clock() routine on the VAXStation 5500 are quantized in multiples of 10. The vertical scale for the run-time unit value will be displayed with tick labels numbered by the largest relevant power of 10 (i.e., 1, 10, 100, or 1000). Figure 33 shows the method how to find the displayed numbers of run-time unit.

```

degree = 10[log(max-0)]
number_displayed = max / degree
for (i = 0; i <= number_displayed; i++)
  display (i * degree)

where max = maximum number of run time unit
number_displayed = the number of value will be displayed
degree = the level number (0, 10, 100, 1000, 10000, ...)

```

Figure 33. The Displayed Run-Time Unit Value Formula.

If simple polygons in input data have the same number of vertices then the run-time chart will display the result of all simple polygon run time and also the program will display the normal distribution, the mean (average), and the standard deviation of simple polygons run time.

The flow-chart of the whole process is shown in Figures 34a and 34b.

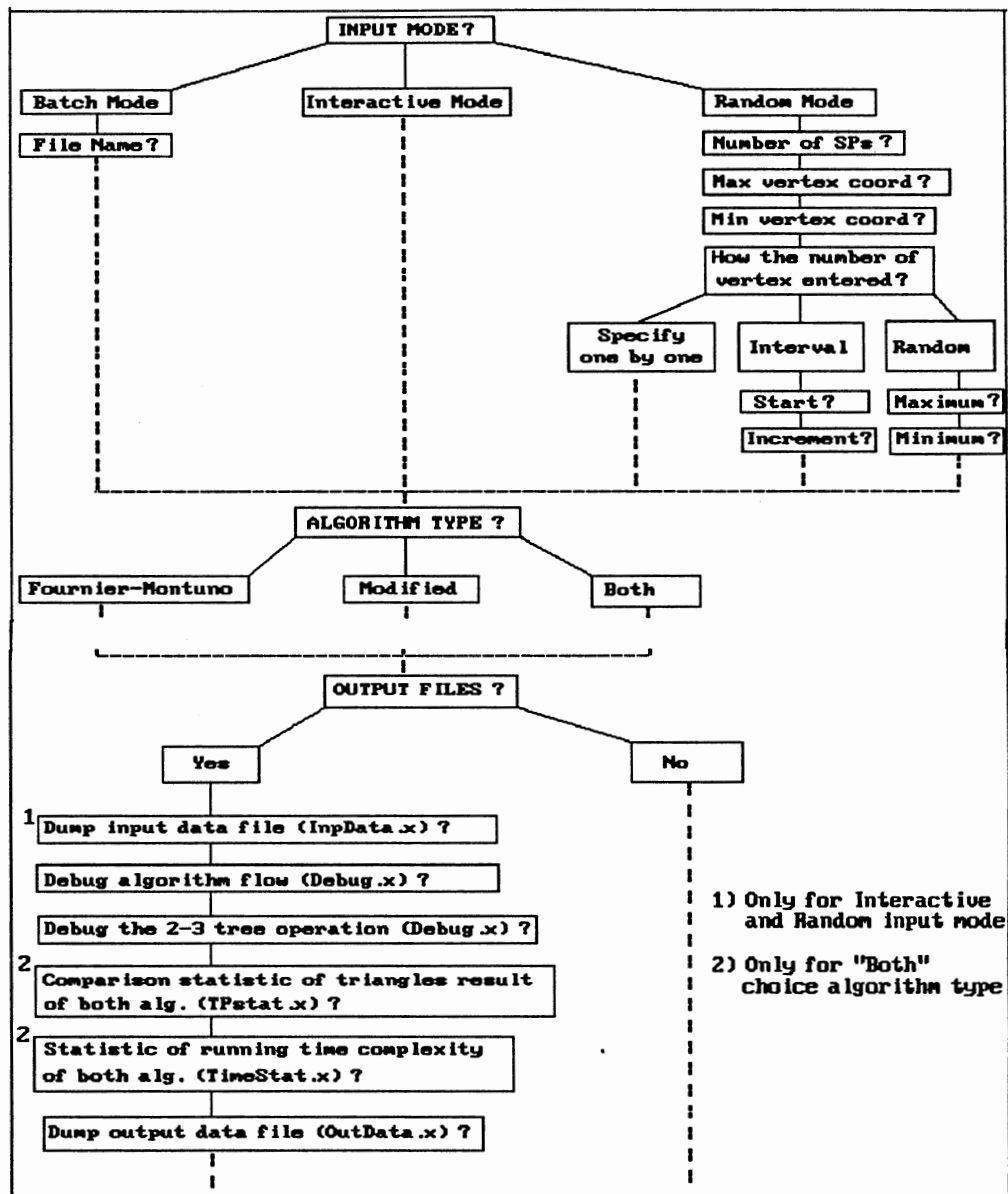


Figure 34a. The Process Sequence Flow Chart.

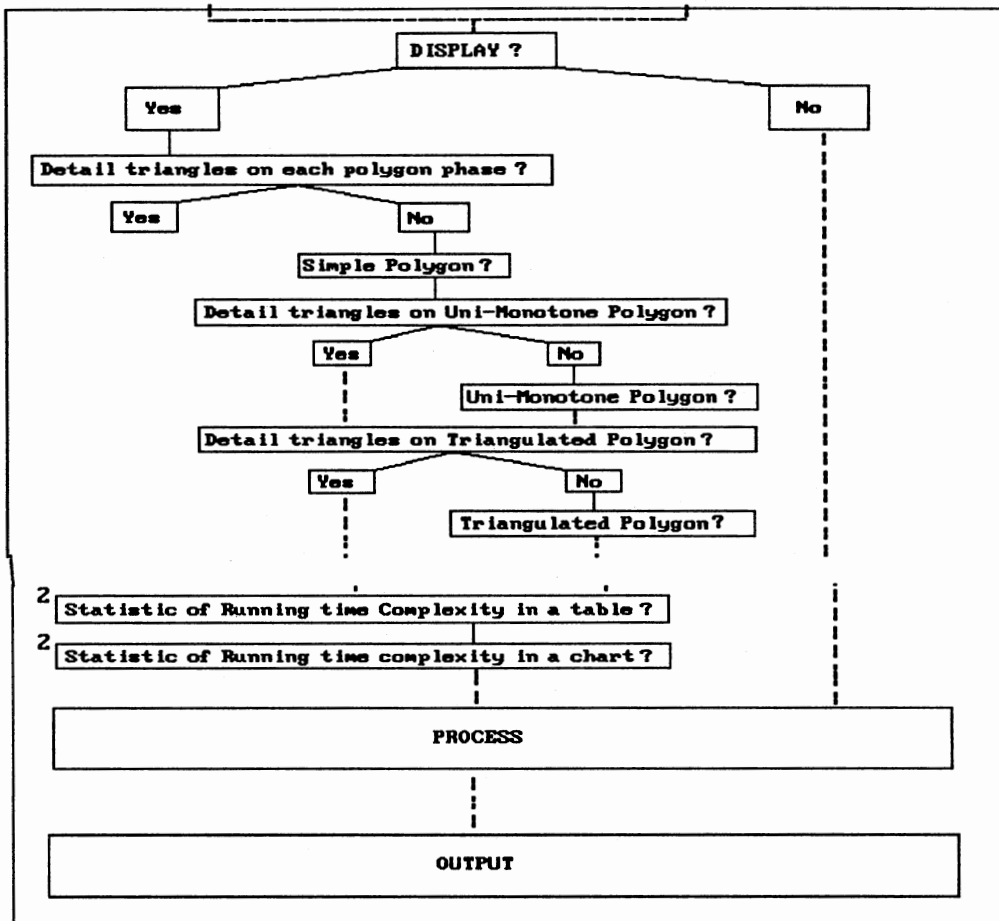
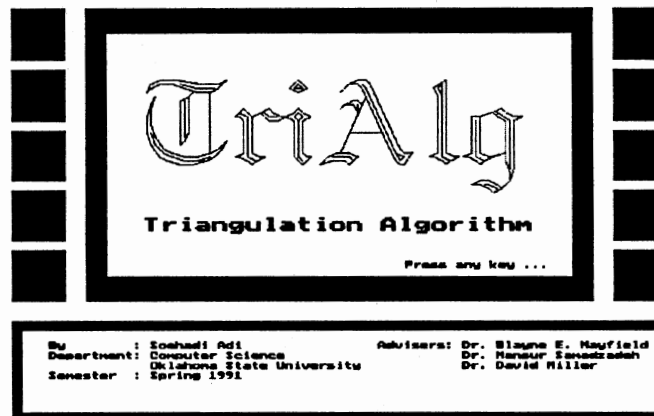


Figure 34b. The Process Sequence Flow Chart.

APPENDIX C

SAMPLE EXECUTION

The program always start with a logo and information about the program to the user (see Figure 35). There are three samples with different approaches.



This program implements:
'An Algorithm to Triangulate a Simple Polygon'
It requires inputs:

1. How input data is read.
 - Interactive
 - Batch
 - Use Random Generator
2. What algorithm is to be used.
 - Fournier & Montuno triangulation algorithm
 - Modified triangulation algorithm
3. What to display (option Yes and No)

Figure 35. The Logo and Information about the Program.

Sample #1

This sample analyzes a simple polygon with detail display of each uni-monotone polygon and each triangle formed on the Fournier and Montuno's Triangulation Algorithm and the Modified Triangulation Algorithm. This sample uses the same input data as the sample flow of the program in Appendix B. Figure 36 shows the input data (1 simple polygon with 12 vertices). Figure 37 shows the input prompted by the program. Figure 38 shows the simple polygon to be triangulated. Figure 39 and Figure 40 show the sequence of formation of uni-monotone polygons and triangles by the two algorithms (Fournier and Montuno's and Modified Triangulation Algorithms respectively).

```
*batch12 12
8         20
10        16
12        18
14        12
18        14
16        15
22        17
18        10
20         4
10         8
2          6
6         11
***
```

Figure 36. Input Data of Sample #1
(DATA\B.12).

```

How you will build the input?
1. Press 'i' or 'I' for Interactive input
2. Press 'b' or 'B' for Batch file input
3. Press 'r' or 'R' for Random input from computer
4. Press 's' or 'S' for displaying the statistic of runtime file
5. Press 'q' or 'Q' for QUIT
choice = B
Type in the batch input file name = DATA\B.12
<NEW SCREEN>
What algorithm you want to use:
1. Press 'f' or 'F'
   if you want to use FOURNIER and MONTUNO triangulation algorithm
2. Press 'a' or 'A'
   if you want to use Modified triangulation algorithm
3. Press 'b' or 'B'
   if you want to use both algorithms
Choose ... B
<NEW SCREEN>
Press 'Y'-Yes and 'N'-No for these options
=====
Analyzer files will be created?           n
Displayed results?                         y
Detail polygon on each sequence            y

```

Figure 37. The Input Prompted by the Program.

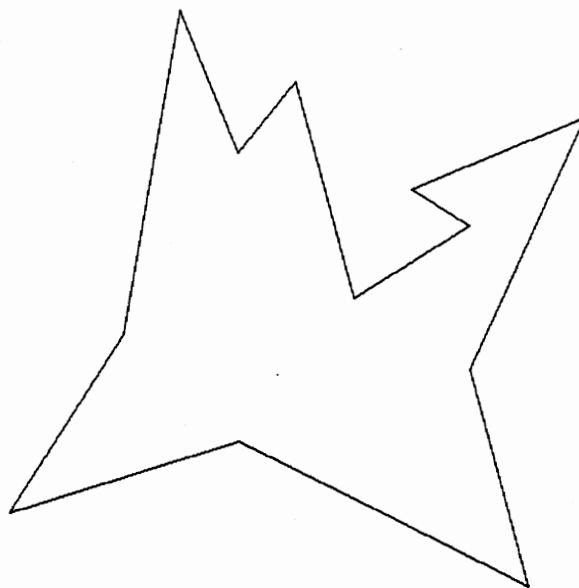


Figure 38. A Simple Polygon.

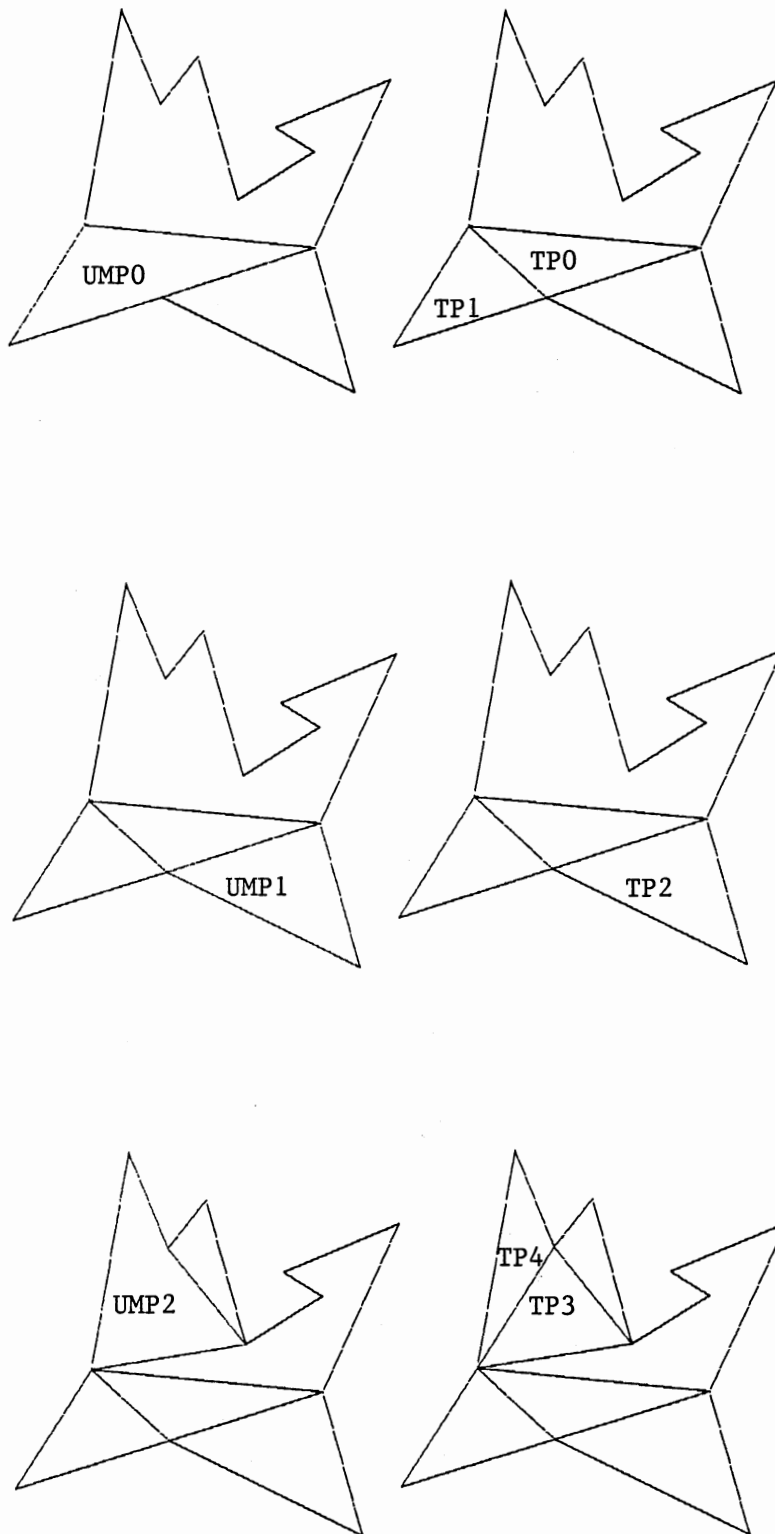


Figure 39a. The Sequence Formation of Uni-Monotone Polygons and Triangles (FM alg.).

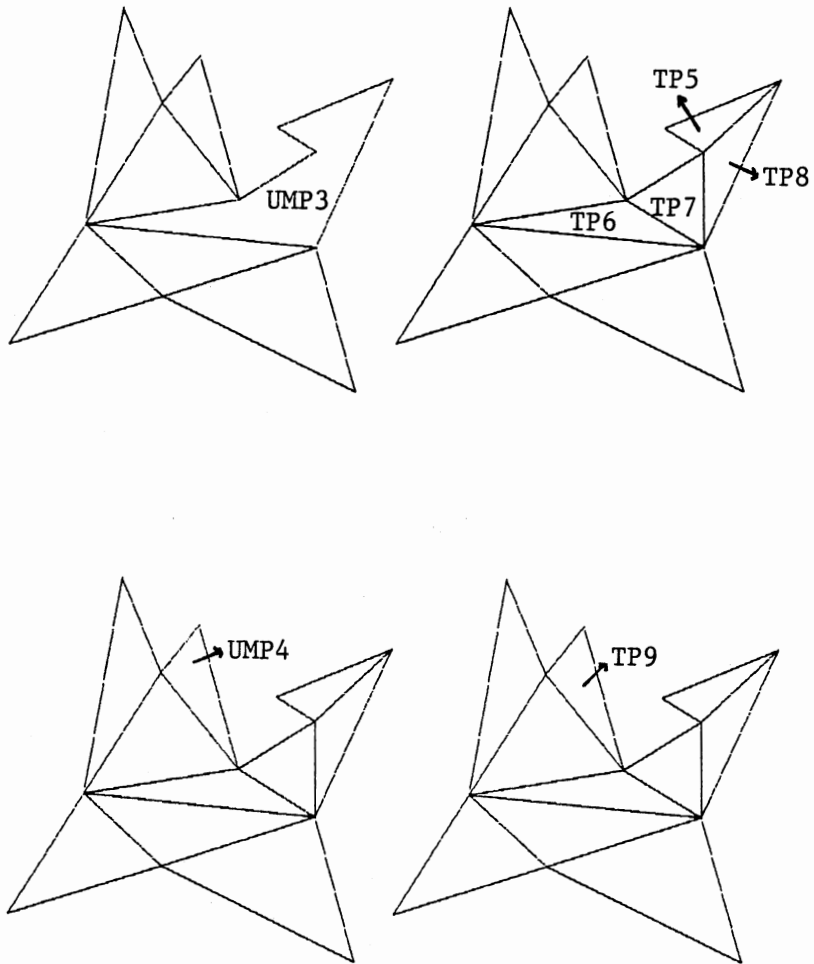


Figure 39b. The Sequence Formation of Uni-Monotone Polygons and Triangles (FM alg.).

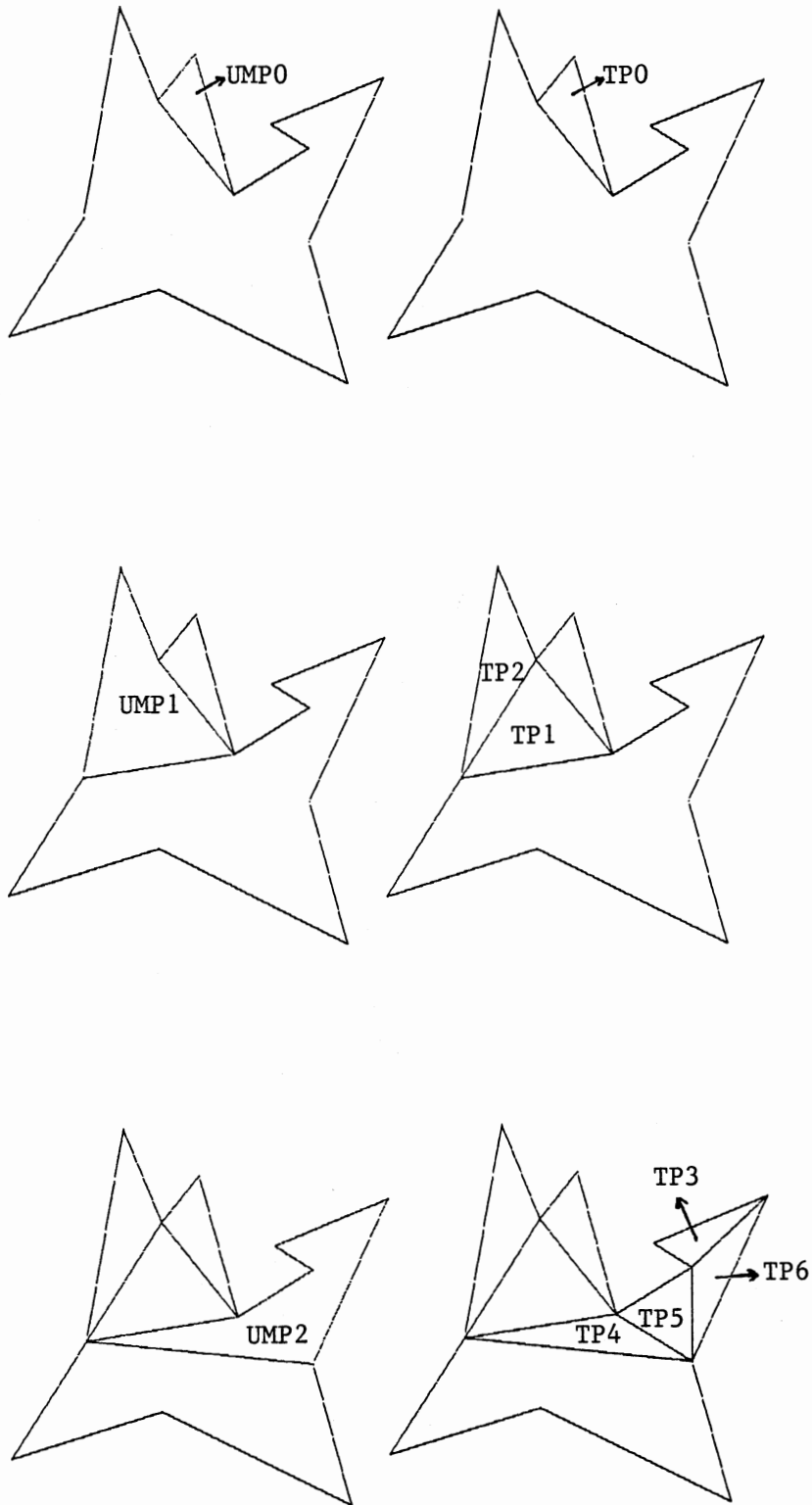


Figure 40a. The Sequence Formation of Uni-Monotone Polygons and Triangles (MO alg.).

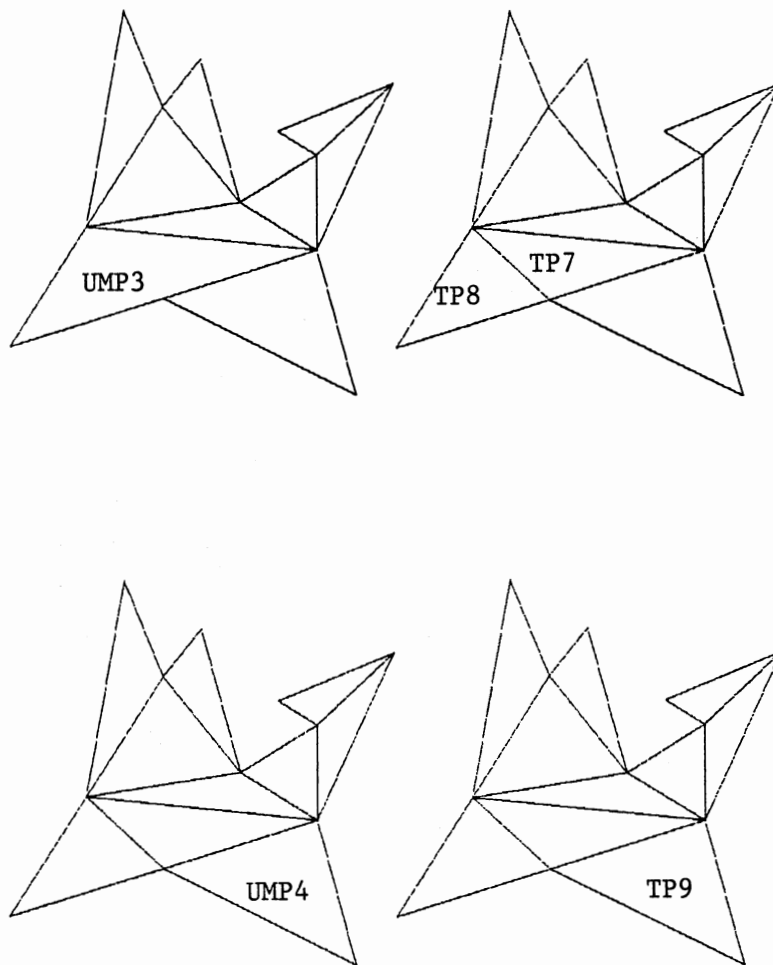


Figure 40b. The Sequence Formation of Uni-Monotone Polygons and Triangles (MO alg.).

Sample #2

This sample analyzes seven simple polygons with 3, 4, 5, 10, 15, 20, and 25 number of vertices respectively and show the run-time table and chart of the algorithms. Figure 41 shows the input data for the sample #2 (7 simple polygons with 3, 4, 5, 10, 15, 20, 25 vertices respectively). Figure 42 shows the input prompted by the program. Figure 43 shows the sequence of formation of uni-monotone polygons and triangles. Figure 44a shows the run-time table, and Figure 44b shows the run-time chart for the algorithms.

*RAND#0	3	305 2733	353 1106	7518 7497
7986 7708		632 7452	8522 7091	4400 1140
1661 2998		490 1874	6547 6442	4286 37
4938 9233		1340 7912	5939 6116	493 30
**		**	4896 4994	1779 6751
*RAND#1	4	*RAND#4	15	6818 6403
8200 7095		5500 9384		7386 6651
6527 612		1701 6523		4440 4596
5414 3051		2534 7411		4068 5682
2359 841		1037 6535		9009 7261
**		8121 5402		1195 477
*RAND#2	5	6095 5288		3809 739
5516 7068		7359 1386		1134 274
6712 6263		775 1052		1350 1227
5129 1881		6504 2082		1267 951
1018 3434		5620 4784		24 1012
1218 4584		5667 2279		796 6267
**		5518 5412		173 7132
*RAND#3	10	4588 4422		**
1914 9730		4811 1969		*RAND#6
8550 3038		642 6858		25
3374 2601		**		6108 9572
6287 2447		*RAND#5		2452 7974
7207 587		20		2181 420
602 787		1350 9364		3388 3486
		1132 5290		4143 958
				659 1116
				63 9370

Figure 41. Input Data of Sample #2
(DATA\B.U25).

```

How you will build the input?
1. Press 'i' or 'I' for Interactive input
2. Press 'b' or 'B' for Batch file input
3. Press 'r' or 'R' for Random input from computer
4. Press 's' or 'S' for displaying the statistic of run time file
5. Press 'q' or 'Q' for QUIT
choice = B
Type in the batch input file name = DATA\B.3_P
<NEW SCREEN>
What algorithm you want to use:
1. Press 'f' or 'F'
   if you want to use FOURNIER and MONTUNO triangulation algorithm
2. Press 'a' or 'A'
   if you want to use Modified triangulation algorithm
3. Press 'b' or 'B'
   if you want to use both algorithms
Choose ... B

<NEW SCREEN>
Press 'Y'-Yes or 'N'-No for these options
=====
Analyzer files will be created?
Give warning to overwrite files?           n
Debug for the flow of the algorithm (Debug.x) n
Triangles statistic of both algorithms (TPstat.x) n
Statistic of the run time unit (TimeStat.x)  y
The result of the SP, TrP, UMP, and TP vertices (OutData.x) y
Displayed results?
The Simple Polygon                          y
The detail of each Uni-Monotone Polygons     n
The Whole Uni-Monotonized Polygon           y
The detail of each Triangles                 n
The Whole Triangulated Polygon              y
The table of the run time on processing each Simple Polygon y
The chart of the duration on processing each Simple Polygon y

```

Figure 42. The Input Prompted by the Program.

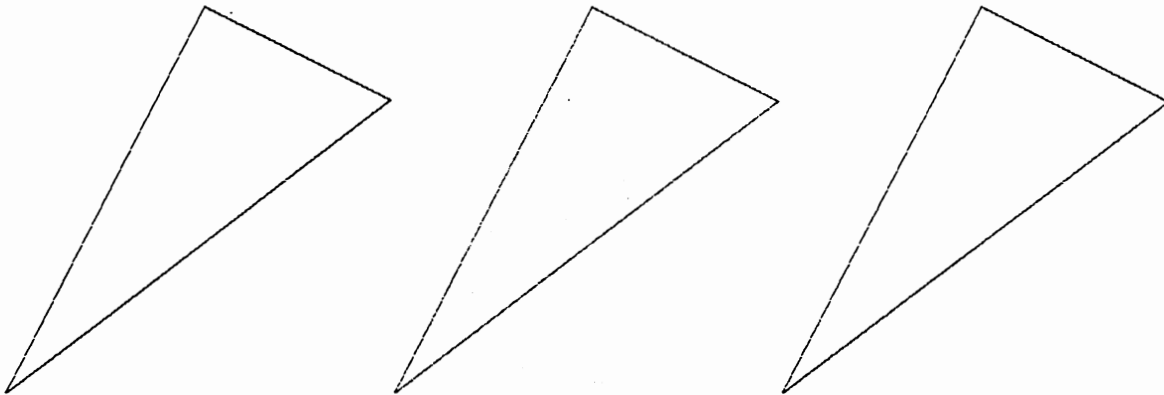


Figure 43a. The Sequence Formation of Uni-Monotone Polygons and Triangles (3 Vertices).

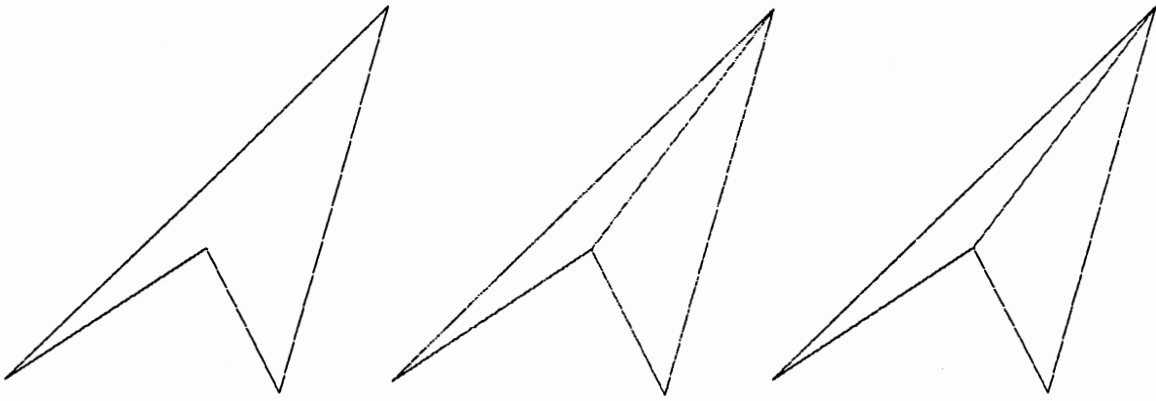


Figure 43b. The Sequence Formation of Uni-Monotone Polygons and Triangles (4 Vertices).

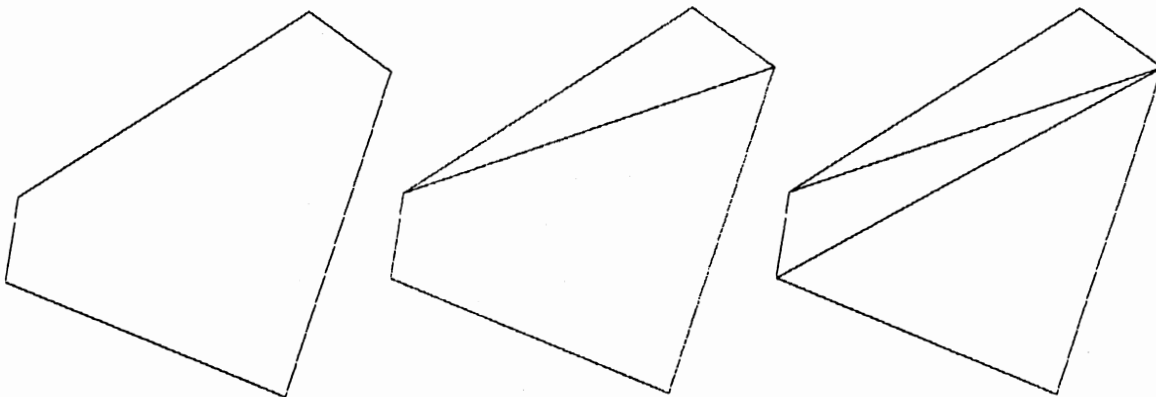


Figure 43c. The Sequence Formation of Uni-Monotone Polygons and Triangles (5 Vertices).

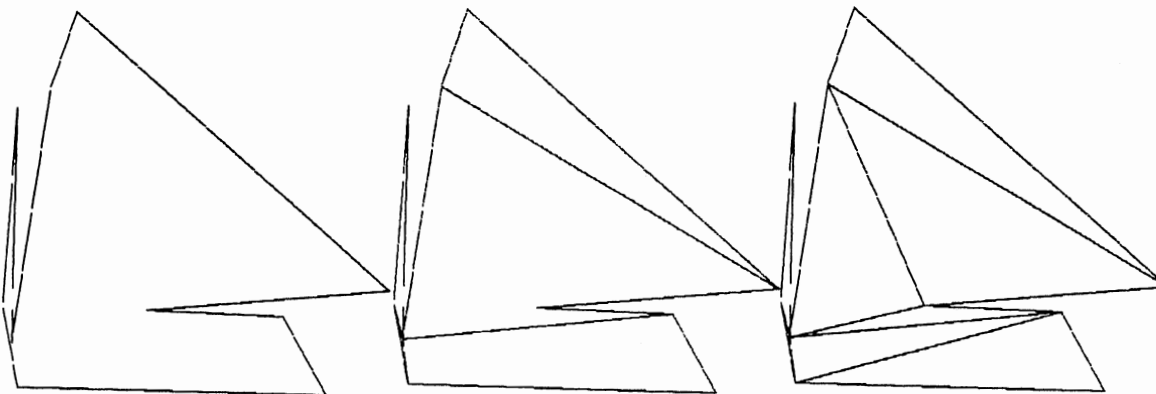


Figure 43d. The Sequence Formation of Uni-Monotone Polygons and Triangles (10 Vertices).

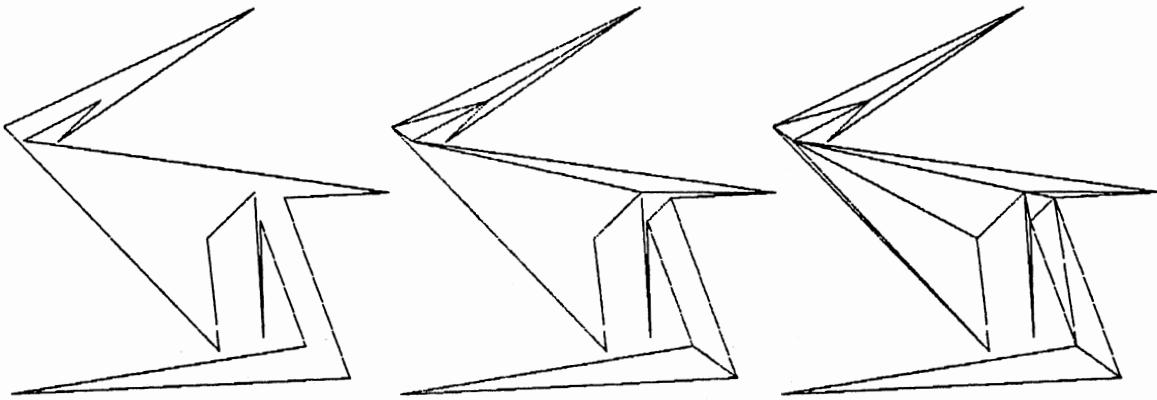


Figure 43e. The Sequence Formation of Uni-Monotone Polygons and Triangles (15 Vertices).

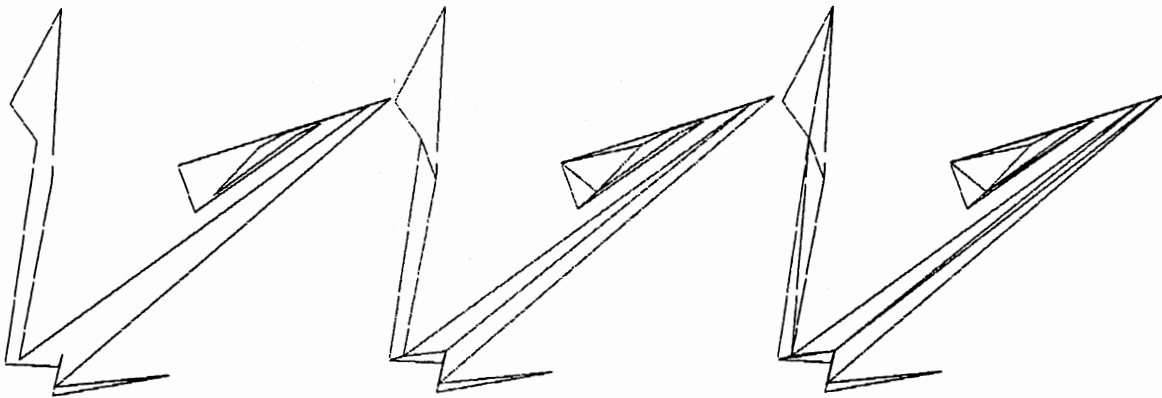


Figure 43f. The Sequence Formation of Uni-Monotone Polygons and Triangles (20 Vertices).

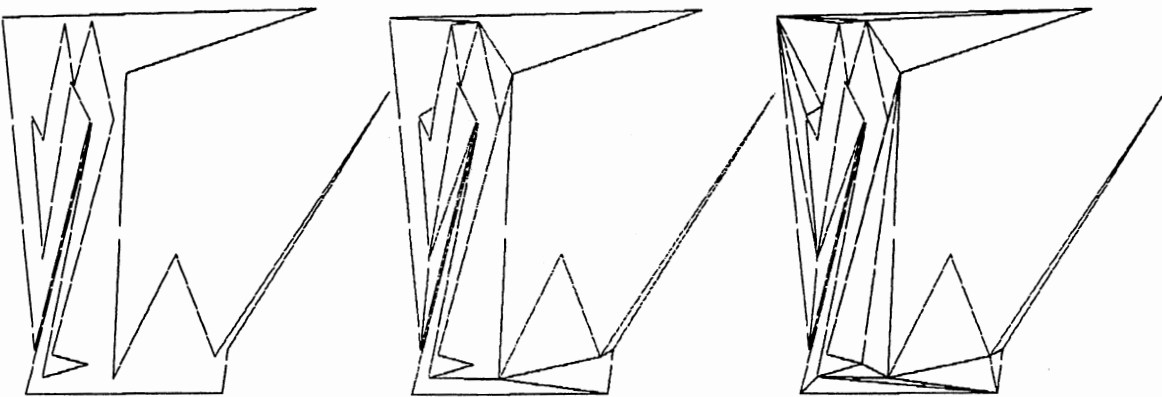


Figure 43g. The Sequence Formation of Uni-Monotone Polygons and Triangles (25 Vertices).

```

*****
*          FOURNIER-MONTUÑO ALG. *          MODIFIED ALG. *
*****
| SIMPLE | # * SP to TrP to UMP to TOTAL * SP to TrP to UMP to TOTAL *
| POL. NAME | VER * TrP UMP TP * TrP UMP TP *
*****
| RAND0  | 3 * 0 0 0 0 * 0 0 0 0 *
| RAND1  | 4 * 0 0 1 1 * 0 0 0 0 *
| RAND2  | 5 * 0 0 1 1 * 0 0 1 1 *
| RAND3  | 10 * 0 0 1 1 * 0 0 1 1 *
| RAND4  | 15 * 1 0 2 3 * 1 0 2 3 *
| RAND5  | 20 * 2 0 3 5 * 2 0 2 4 *
| RAND6  | 25 * 2 1 4 7 * 1 0 5 6 *
*****

```

Figure 44a. Run-Time Table.

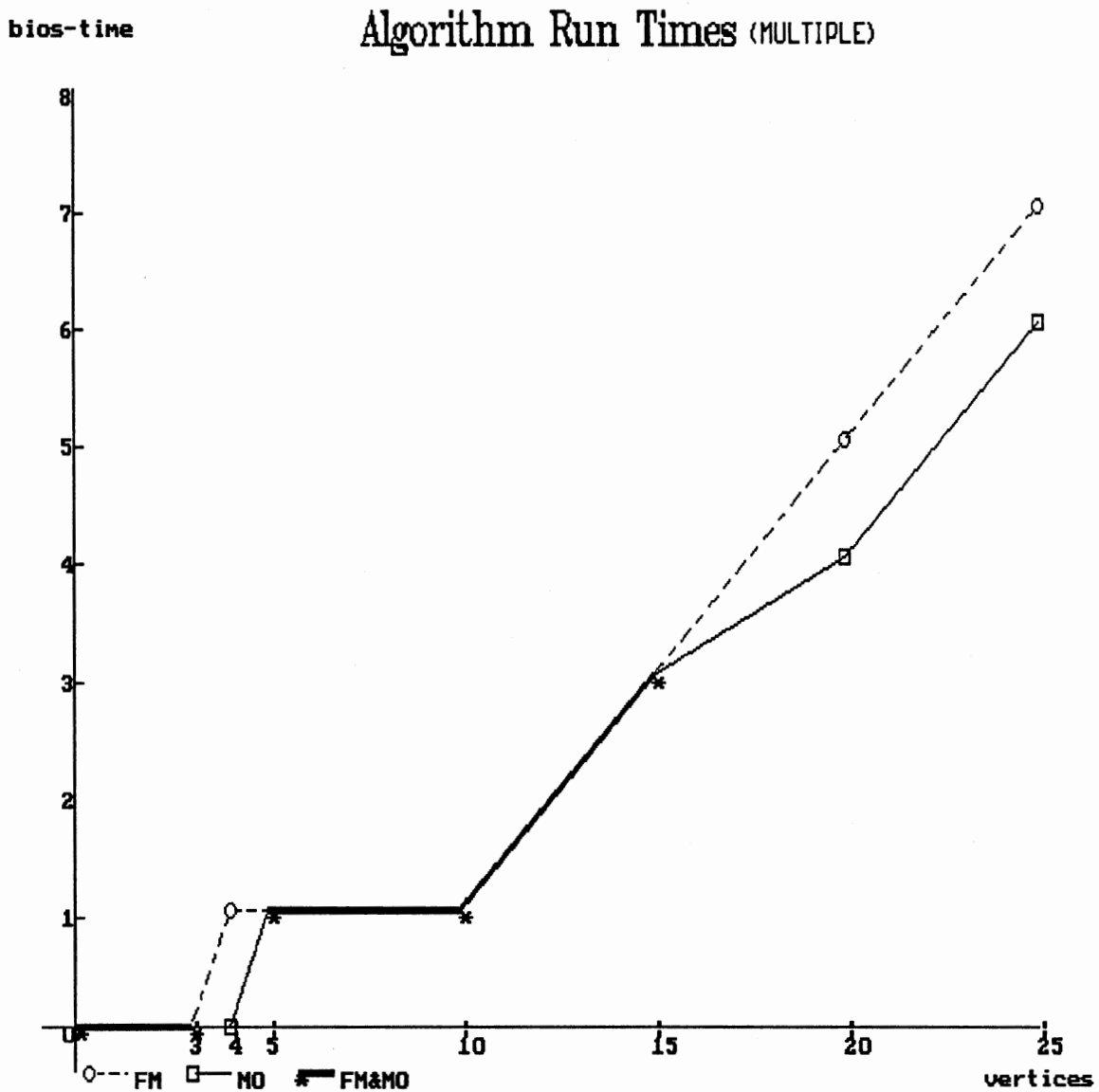


Figure 44b. Run-Time Chart.

Sample #3

This sample analyzes 50 different simple polygons on each 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500 vertices simple polygon. These simple polygons are analyzed on DEC VAXStation 5500. Figures 45a and 45b show the run-time distribution information on 50 vertices and 1000 vertices, respectively (represent the sequence on the number of vertices of the simple polygons). Figure 46a and 46b show the run-time chart of all simple polygons, with each simple polygon that has the same number of vertices are averaged.

This sample shows that the modified triangulation algorithm actually faster than the Fournier and Montuno's Triangulation Algorithm.

Run-Time Distribution

Mean[FM] = 32 StDe[FM] = 7.17
 Mean[MO] = 32 StDe[MO] = 6.08

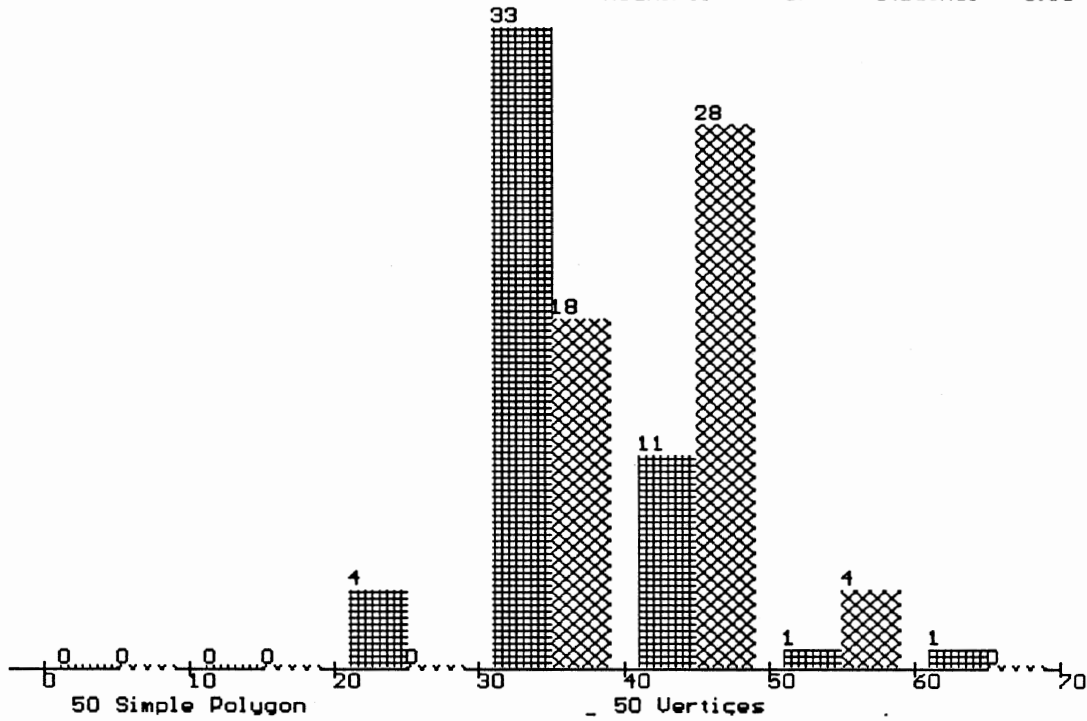


Figure 45a. Distribution Information on 50 Vertices of 50 Simple Polygons.

Run-Time Distribution

Mean[FM] = 2312 StDe[FM] = 231.29
 Mean[MO] = 1878 StDe[MO] = 557.85

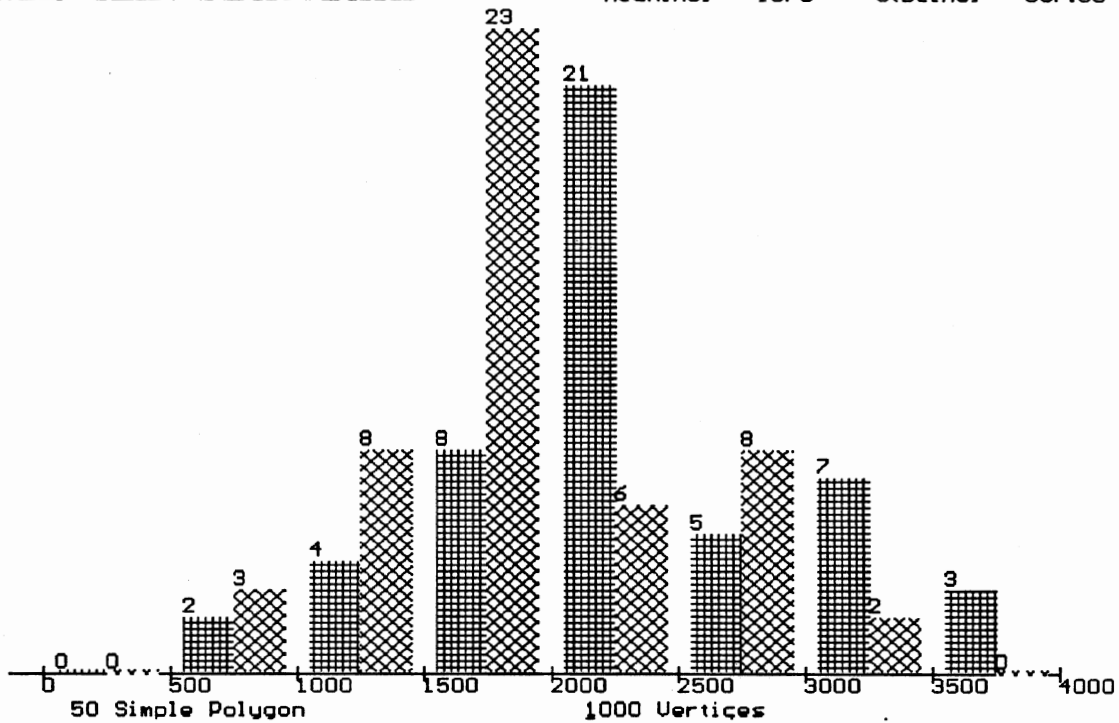


Figure 45b. Distribution Information on 1000 Vertices of 50 Simple Polygons.

bios-time

Algorithm Run Times (MULTIPLE)

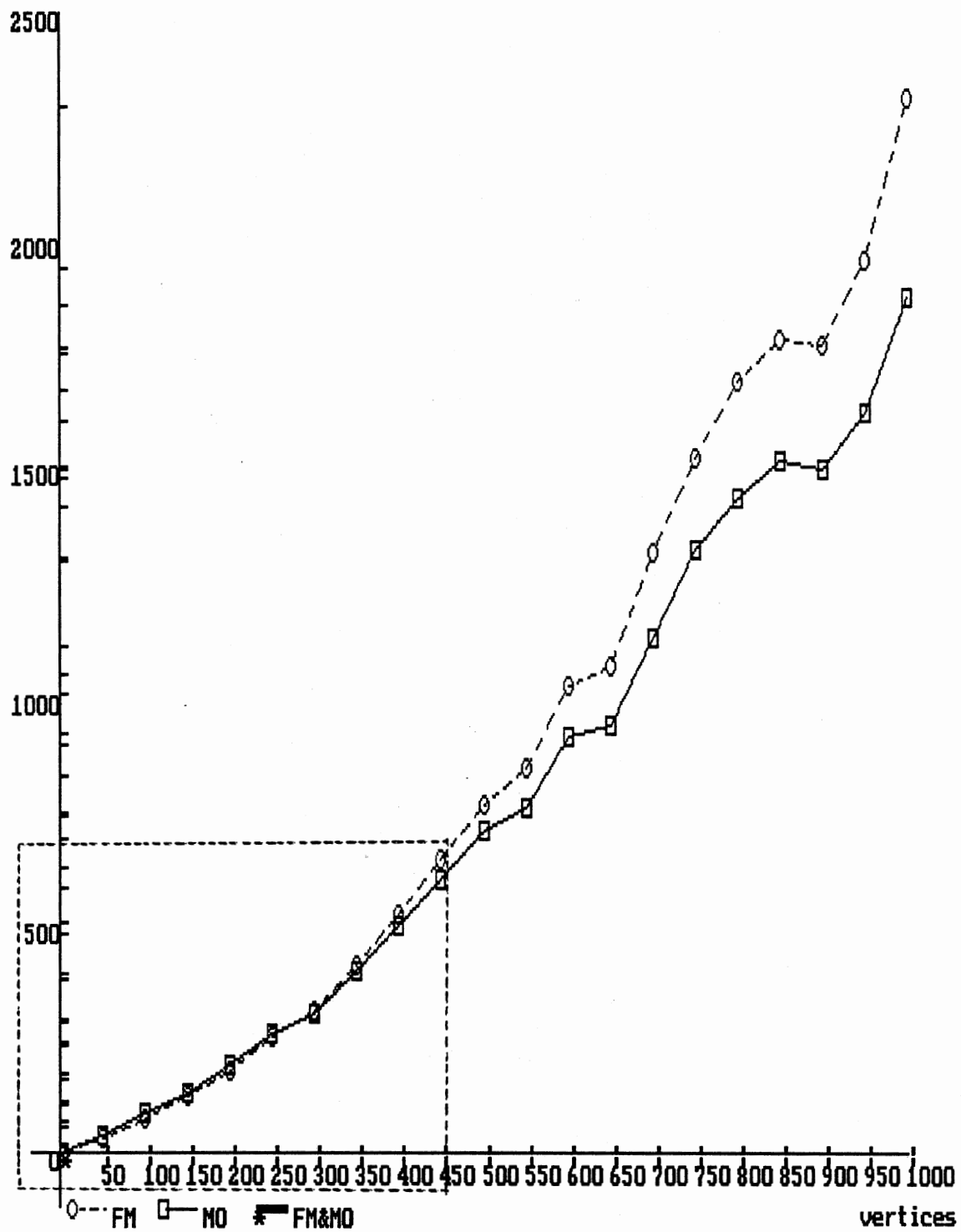


Figure 46a. Run-Time Chart of All Simple Polygons.

bios-time

Algorithm Run Times (MULTIPLE)

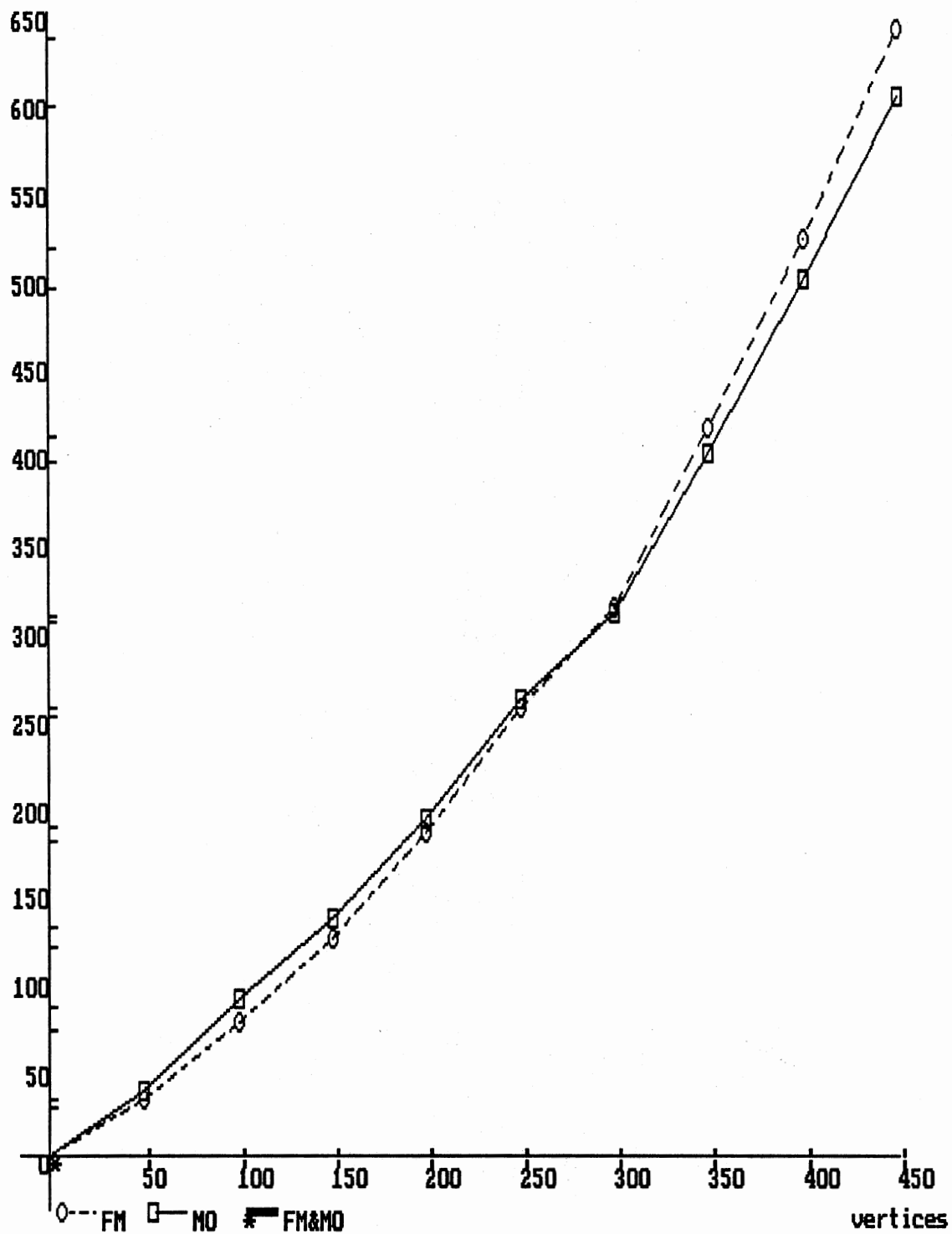


Figure 46b. Run-Time Chart of All Simple Polygons (Enlargement).

VITA

Soehadi Adi

Candidate for the Degree of
Master of Science

Thesis: MODIFICATION OF FOURNIER AND MONTUNO'S
TRIANGULATION ALGORITHM FOR SIMPLE POLYGONS

Major Field: Computer Science

Biographical:

Personal Data: Born in Bangsri, Indonesia, June 16,
1965, the son of Sindu Rahardjo and Sri Suwandini.

Education: Graduated from Kebon Dalem Senior High
School, Semarang, Indonesia, in May 1983; received
Bachelor of Science Degree in Electrical Engineer-
ing from Oklahoma State University in May 1988;
completed requirements for the Master of Science
degree in Computer Science at Oklahoma State
University in July, 1991.

Professional Experience: Programmer, Department of
Agriculture Economic Extension, Oklahoma State
University, October, 1989 to August, 1990.