To the Graduate Council:

I am submitting herewith a thesis written by Judy Chien-Hui Chang entitled "Construction of the Voronoi and Delaunay Tessellations." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Kenneth R. Kimble

Kenneth R. Kimble, Major Professor

We have read this thesis
and recommend its acceptance:

V C Reddy

Fletcher W. Donaldson

Accepted for the Council:

L Evans Peck

Vice Chancellor
Graduate Studies and Research

CONSTRUCTION OF THE VORONOI AND DELAUNAY TESSELLATIONS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Judy Chien-Hui Chang

June 1983

ACKNOWLEDGMENTS

The author wishes to take this opportunity to express gratitude to her major professor, Dr. Kenneth R. Kimble, Associate Professor of Mathematics, The University of Tennessee Space Institute, for the suggestion of the topic and for his valuable guidance, both of which have added immeasurably to the completion of this thesis. Sincere thanks are also extended to the committee Members, Dr. F. W. Donaldson, Professor of Computer Science, and Dr. K. C. Reddy, Professor of Mathematics, for reviewing and commenting on the manuscript.

The work reported herein was supported in part by the National Aeronautics and Space Administration.

ABSTRACT

The basic definitions and properties of the Voronoi tessellation and Delaunay triangulation are introduced. A computer program for generating the tessellations was developed. Related work and applications of these tessellations are discussed. Some alternatives for the algorithm and enhancements to the computer program developed in this work are identified.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

TESSELLATIONS--BASIC CONCEPT

Voronoi tessellation and Delaunay triangulation are two geometrical constructs associated with a finite set of distinct points. They are geometric duals of each other (i.e., either one can be constructed from the other one). The definitions, properties and special features of these two tessellations are discussed in this chapter.

1.1. Definitions

From a pure mathematical point of view, the definitions given below may also be applied to any non-Euclidean metric space which has a different definition of the distance function. In this work, any space referred to is a Euclidean space and any distance between points refers to Euclidean distance unless defined otherwise.

A tessellation is an aggregate of bounded convex polytopes that fit together to cover the k-dimensional space Sk without any overlapping.

The following definition of a particular tessellation applies to Euclidean space in any number of dimensions and is known as the Voronoi tessellation in k-dimensional space.[1]

---

[1]The name Dirichlet tessellation has also been used for the same structure.

Definition 1

Let {P1, P2, ..., Pn} be a finite set of distinct points in Sk (the k-dimensional space). For each point Pi(i = 1, 2, ..., n), the region of Pi is the set Ri defined by

$$Ri = \{x \in Sk: \ d(x, Pi) < d(x, Pj) \text{ for all } j \neq i \text{ and } j = 1,...,n\} ,$$

where d is the Euclidean distance.

According to the definition, the one-dimensional case is trivial. The region associated with each point is the segment of the line extending in both directions from the point halfway towards the next point, or in the boundary case, extending to infinity. In two-dimensional space, i.e., a plane, the region associated with each point is the portion of the plane nearer to that point than to any other point. Consider the half-plane that includes one point and is bounded by the perpendicular bisector of the line joining this point with any other point. The interesection of all these half-planes is the convex polygon (or the infinite region at the boundary) which is nearer to this point than to any other. Figure 1 is the Voronoi tessellation of a set of nine points in

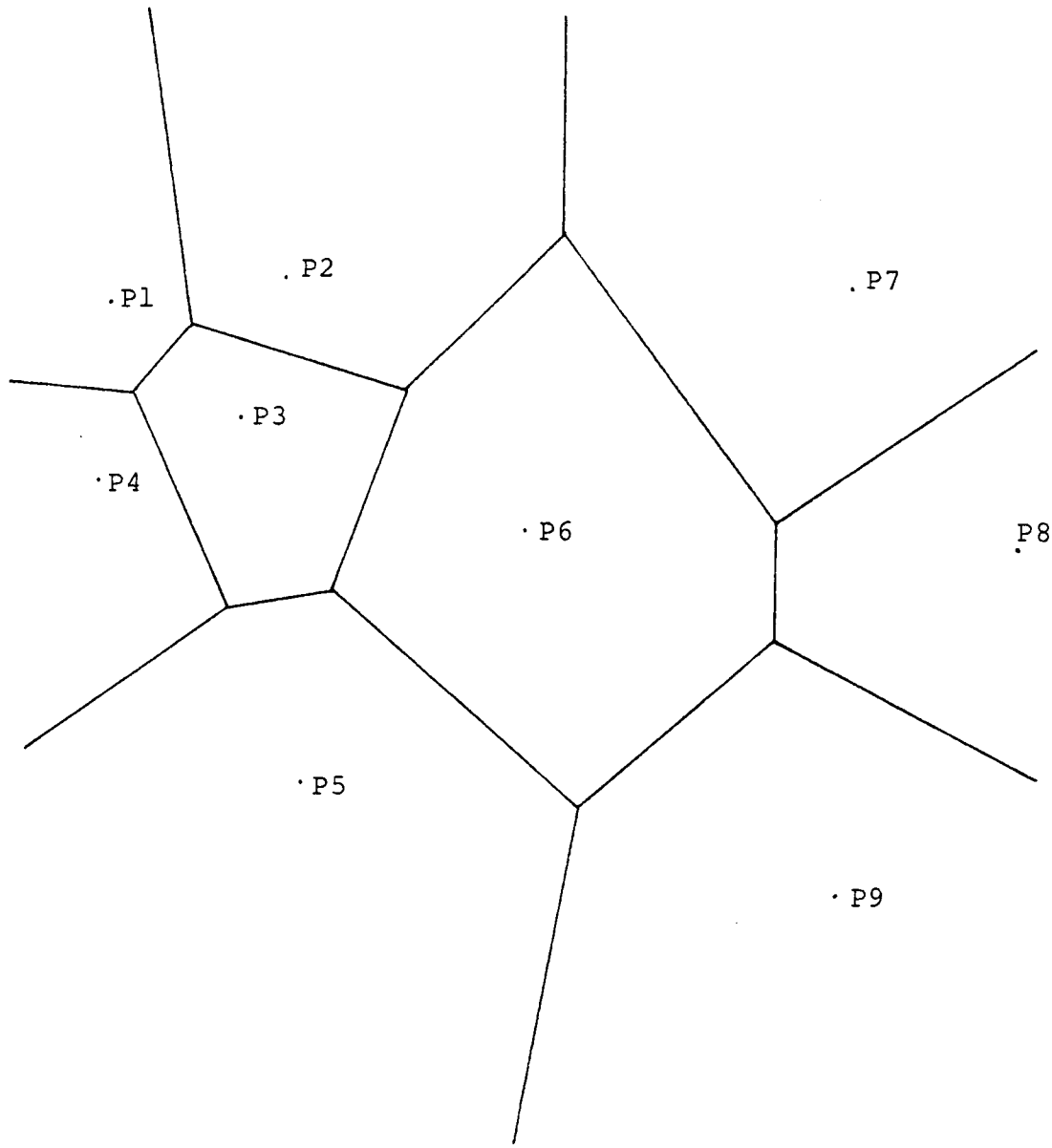


Figure 1. Voronoi Tessellation of Nine Data Points.

two-dimensional space.[2] Similarly, in three dimensions, the region of each point is a convex polyhedron; this region of space is nearer to the interior point than to any other.

Definition 2

The regions in the Voronoi tessellation which have a boundary in common are said to be contiguous. The points associated with two contiguous regions are also contiguous.

From the definition of Voronoi tessellation given above, it is evident that each segment of the region boundary is part of the perpendicular bisector of the line joining the point pair between whose regions the boundary segment lies. Therefore, Delaunay triangulation (the geometrical dual of the Voronoi tessellation) can be defined as follows.

Definition 3

If all contiguous point pairs are joined by straight lines, the result is a triangulation associated with the data points.

---

[2]Various authors refer to an individual region within the construction as "polygon," "territory," "tile," or "Thiessen polygon."

According to Definition 3, the Delaunay triangulation in the one-dimensional case is trivial. It is the sum of all the line segments which join data point pairs. The Delaunay triangulation in two-dimensional space is a pattern of packed irregular triangles covering the convex hull of the data points. Figure 2 is the Delaunay triangulation of the same set of nine points as in Fig. 1. In three dimensions, it is an aggregate of disjoint irregular tetrahedra. The term simplex is applied to an individual triangle or tetrahedron in Delaunay triangulation.

In k-dimensional Voronoi and Delaunay tessellations each vertex of Voronoi regions occurs where k+1 regions meet. The k+1 points in those k+1 regions are called the forming points of that associated vertex. When more than k+1 regions meet at a vertex, the vertex is said to be degenerate. Degeneracy will be discussed in Chapter IV.

Definition 4

The simplices in k-dimensional Delaunay triangulation which have k forming points in common are said to be contiguous. The vertices associated with two contiguous simplices are also contiguous. Two contiguous vertices are neighboring vertices.

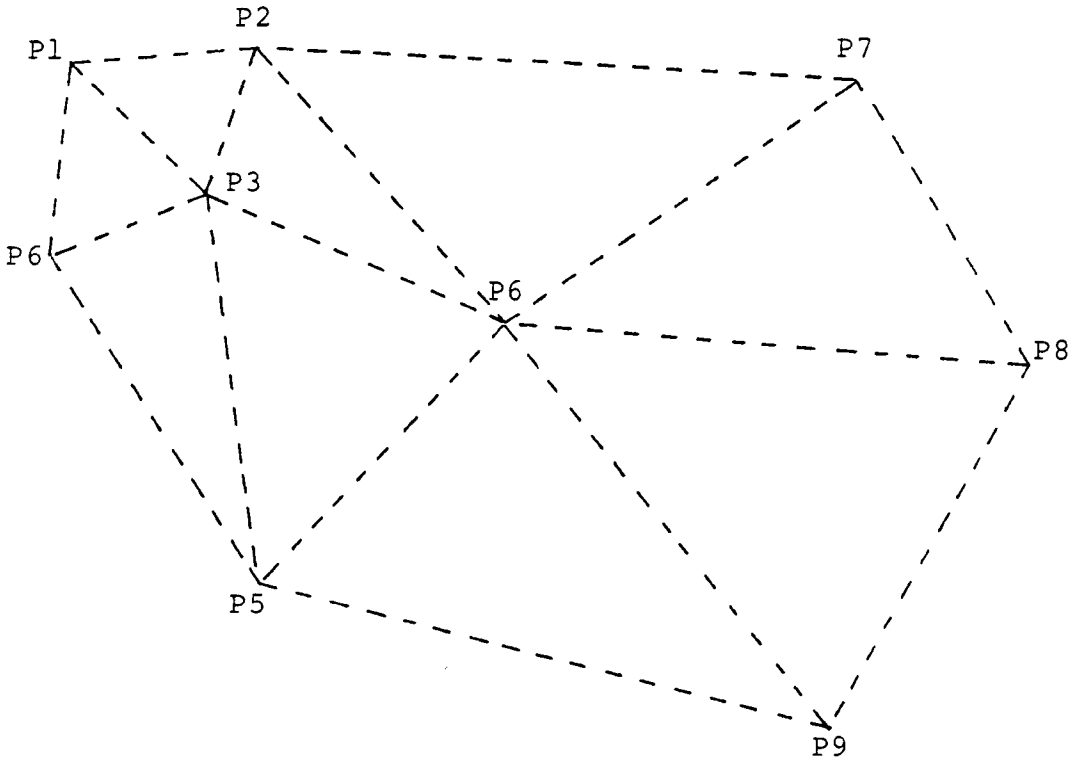


Figure 2. Delaunay Triangulation of Nine Data Points.

In conclusion, the Voronoi tessellation and Delaunay triangulation are both defined by the same finite set of points. Each Voronoi region has a unique point associated with it. Also, each Delaunay simplex has a unique vertex of a Voronoi region associated with it and vice versa. The original set of data points actually lies on the vertices of the simplices in Delaunay triangulation. In order to avoid any ambiguity, vertices will be used only for the regions in Voronoi tessellation.

1.2. Properties

Consider the Voronoi and Delaunay tessellations associated with n distinct points in a k-dimensional space.

Theorem 1

Each vertex is the geometric center of the simplex formed by its k+1 forming points (i.e., each vertex is equidistant from all k+1 of its forming points).

Proof

Let Pi, i=1,...,k+1 be the k+1 forming points of the simplex associated with vertex V, and let d(V,Pi) be the distance between V and Pi, i=1...,k+1. If there are two points Pi and Pj, i=j such that $d(V,Pi) \neq d(V,Pj)$, say $d(V,Pi) < d(V,Pj)$, then by Definition 1, $V \in Ri$ (the

region of Pi). Since the vertex lies on the boundary of the region, this is a contradiction.

Due to this property, each vertex is naturally defined by its forming points.

Theorem 2

The k+1 forming points of each k-simplex in Delaunay triangulation lie on the (k-1)-dimensional circumference of a k-dimensional hypersphere with the center at the associated vertex. No other point lies within that sphere.

Proof

This is proved by Theorem 1.

Theorem 3

The boundary shared by a contiguous point pair lies in the (k-1)-dimensional hyperplane that bisects that line joining the point pair.

Proof

Let x be a point on the boundary shared by a contiguous point pair Pi and Pj, $i \neq j$. If x does not belong to the plane that bisects the line joining Pi and Pj, then $d(V,Pi) \neq d(V,Pj)$, say $d(V,Pi) < d(V,Pj)$. By Definition 1, $x \in Ri$ (the region of Pi) and x does not lie on the boundary of Ri. This is a contradiction.

The two- and three-dimensional cases will now be examined.[3] Figure 3 shows the single Delaunay 2-simplex (i.e., Delaunay triangle) and its associated vertices. Table 1 is the topology table of this structure. The vertex V0 lies at the point where the three regions meet and is the center of the triangle formed by three points, P1, P2, and P3 (the forming points). The numbers associated with points are arbitrary identifiers. By Theorems 1 and 2, V0 is equidistant from all three forming points and the three points all lie on the circle with V0 as the center, and the distance between any point and the vertex is the radius R0. Also, by Theorem 3, a boundary (say V0-V3) shared by the contiguous point pair (P1 and P3) lies on the line which bisects the line segment joining the point pair (P1 and P3). There is a certain order of storing neighboring vertices of V0. That is, the triangle of the first neighboring vertex (N.V.) opposite to V0 and the triangle of V0 have the boundary that contains the second and third forming points (F.P.) of V0 in common. The triangle of the second N.V. of V0 and the triangle of V0 have the boundary that contains the first and third F.P. of V0 in common. And the triangle of the third N.V. of V0 and the triangle of V0 have the boundary that contains the first and second F.P. of V0 in common.

---

[3]The tessellation in k-dimensional (k > 3) space is used primarily in pure mathematical studies.

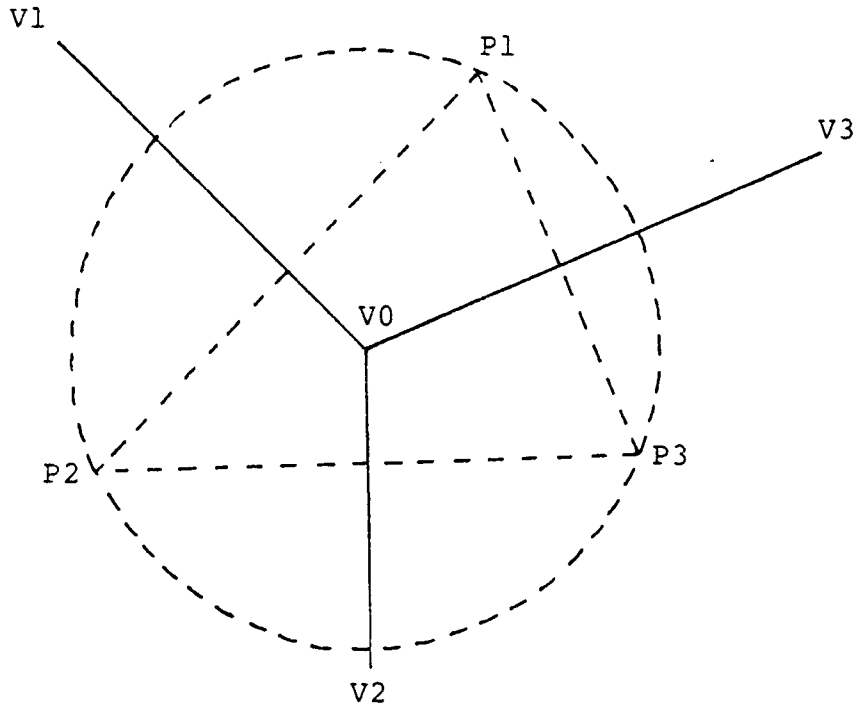


Figure 3. Two-Dimensional Delaunay Triangle and Its Associated Vertices.

Table 1

Two-Dimensional Initial Structure

V	R	F.P.			N.V.		
V0	R0	P1	P2	P3	V2	V3	V1

A slightly more complicated three-dimensional case is shown in Fig. 4 and Table 2. Instead of a triangle, the Delaunay 3-simplex is a tetrahedron; V0 is at the location where four regions meet, which is also the center of the tetrahedron formed by the points, P1, P2, P3, and P4. Therefore, the four forming points lie on the surface of a sphere with the center at V0 and the radius R0. The boundary (say defined by V3-V0-V4) lies on the plane that bisects the line segment joining the contiguous point pair (P3 and P4). Similarly, the tetrahedron of the first N.V. of V0 and the tetrahedron of V0 have the boundary containing the second, third, and fourth forming points of V0 in common, and so on.

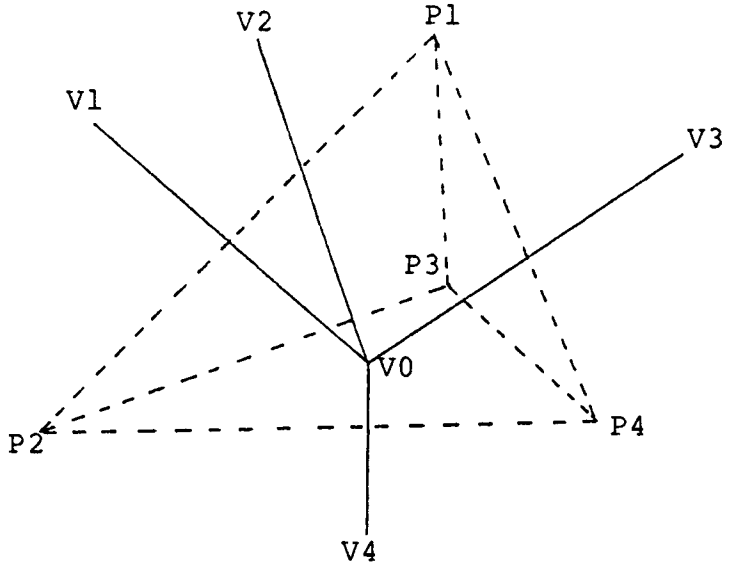


Figure 4. Three-Dimensional Delaunay Tetrahedron and Its Associated Vertices.

Table 2

Three-Dimensional Initial Structure

V	R	F.P.				N.V.			
V0	R0	P1	P2	P3	P4	V4	V3	V2	V1

CHAPTER II

ALGORITHM--DEVELOPMENT AND IMPLEMENTATION

2.1. Introduction

Various algorithms have been used to find the boundaries of the Voronoi polygons. One algorithm was given by Rhynsburger [1][1] in 1973. Knowing that the nearest neighboring points to some given points must form an interface, one may proceed by tracing a pathway, always in the same rotational direction, around each polygon. Shamons [2] gave a general discussion of a geometrical algorithm in 1975. A more sophisticated approach by Green and Sibson [3] describes a recursive algorithm tracing the boundary adjustments required as a new point with its associated new polygon fitted into an established aggregate. Brown [4] published an algorithm for construction of a planar Voronoi diagram of n points by transforming the points into a three-dimensional space, followed by the construction of a convex hull back to the plane. Further, according to Brown, the construction of Voronoi diagrams for a general n-dimensional case can be achieved by using an extension of the same rules. However, computing these

---

[1]Numbers in brackets refer to similarly numbered references listed in the Bibliography.

structures in three or higher dimensions was not accomplished until 1980. Watson [5] established an algorithm that finds the topological relationships in these tessellations and then computes the Delaunay triangulation.

The algorithms introduced in the present study are based on the algorithm originally designed by Bowyer [6]. The work of Bowyer has been modified by using the relationships between the Voronoi and Delaunay tessellations explored by Watson [5]. These algorithms compute the construct in a k-dimensional Euclidean space ($k \geq 2$) as a new point is added and an existing point is deleted. Adjusting the boundaries of the simplices and restoring the tessellations are the main tasks of these algorithms. the computer software developed here has been purposely limited to two and three dimensions in order that a graphical display of the result can be presented.

2.2. Algorithm for Adding a Forming Point

As mentioned above, this algorithm computes the tessellation to be generated after a new point is added. Hence, it must have previously stored the construct which is to be updated.

To explain the algorithm, the construct introduced in Fig. 5 will be used as a starting construct. The topology table of this structure is included in Fig. 5.

Suppose a new forming point Q is added into the construct; six steps constitute the basic algorithm for adding a point: Step 1, Fig. 6; Step 2, Fig. 7; Step 3, Fig. 8; Step 4, Table 3; Step 5, Table 4; Step 6, Fig. 9. Figure 10 and Table 5 illustrate the final structure for adding a point Q.

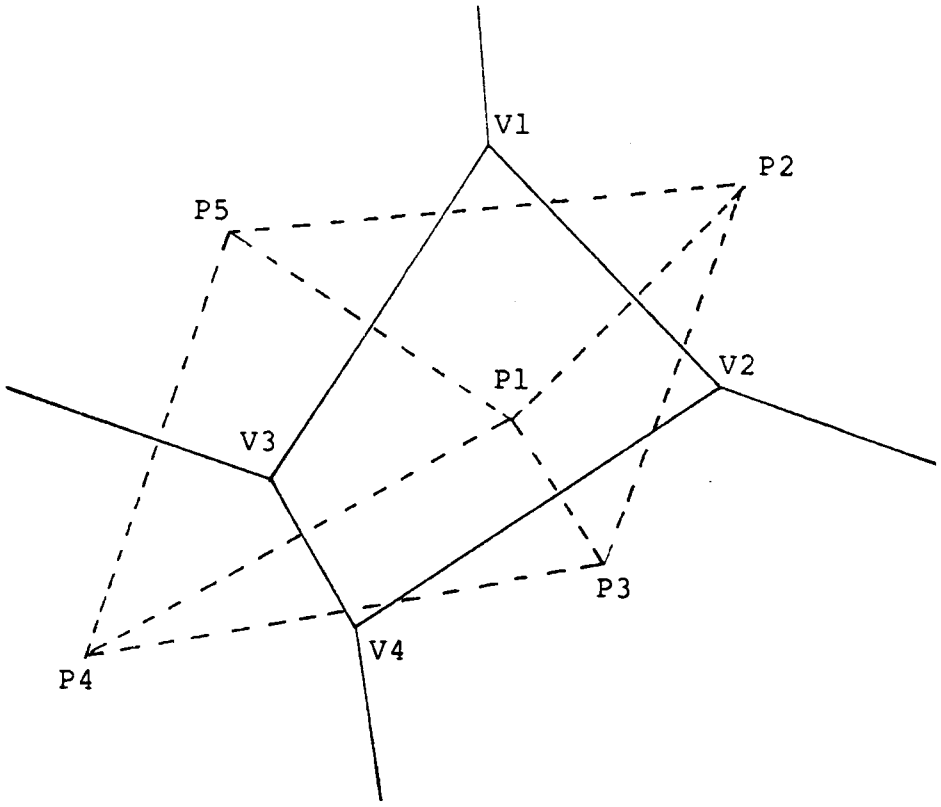
2.3. Algorithm for Deleting a Forming Point

To explain this algorithm, an example construct introduced in Fig. 11 is used. It is stored as shown in Table 6.

Suppose the point Q is deleted from the current construct; five steps constitute the algorithm for deleting a point: Step 1, Fig. 12; Step 2, Fig. 13; Step 3, Fig. 14; Step 4, Fig. 15; Step 5, Table 7. Figure 16 and Table 8 illustrate the final structure for deleting a point Q.

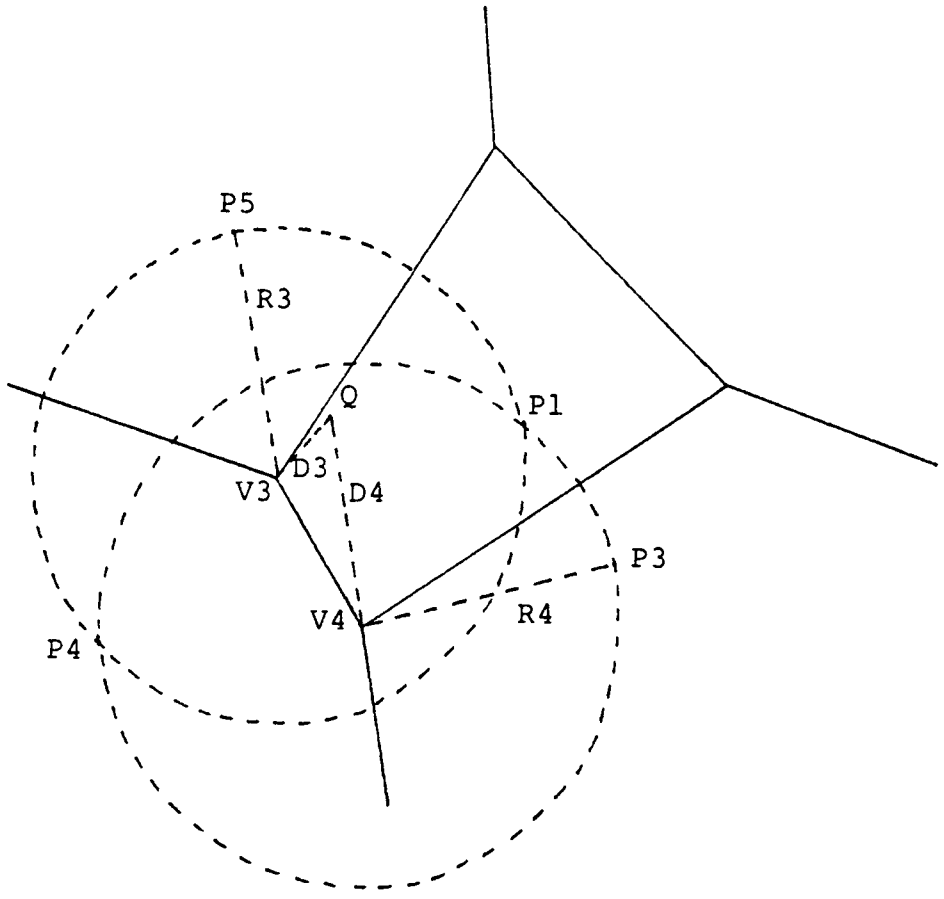
2.4. Data Structure

Before discussing the details of the program, two features of storing the information on the constructs will be presented. Recalling the topology information



V	R	F.P.			N.V		
V1	R1	P1	P2	P5	0	V3	V2
V2	R2	P1	P2	P3	0	V4	V1
V3	R3	P5	P1	P4	V4	0	V1
V4	R4	P3	P4	P1	V3	V2	0

Figure 5. Example Structure for Adding Point.

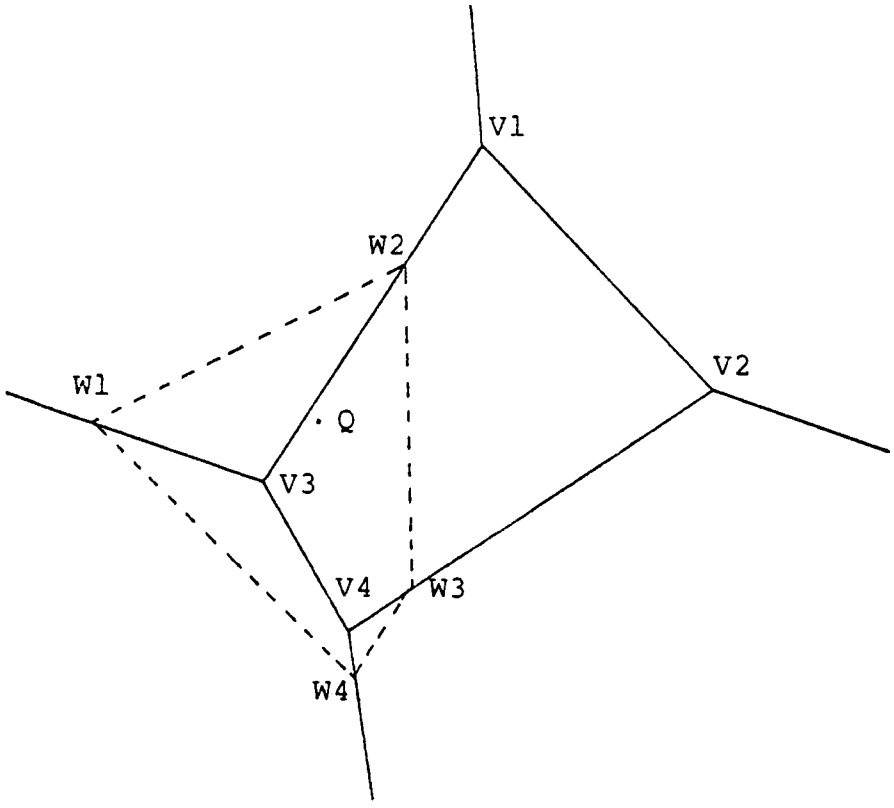


V	R	d(Q,V)
V3	R3	D3
V4	R4	D4

Figure 6. Deleted Structure when Adding a Point Q.

Step 1: Find the list of vertices that will be deleted by the new point; LISTOFV will be used for the name of this list.

If the distance (D3) between the new point (Q) and the vertex (V3) is less than the radius (R3) associated with it, then the vertex should be included in the list. The reason for this is that the vertex is closer to the new point than to any other forming point. In other words, it would lie in the region belonging to the new point. In this example, LISTOFV = {V3,V4}.

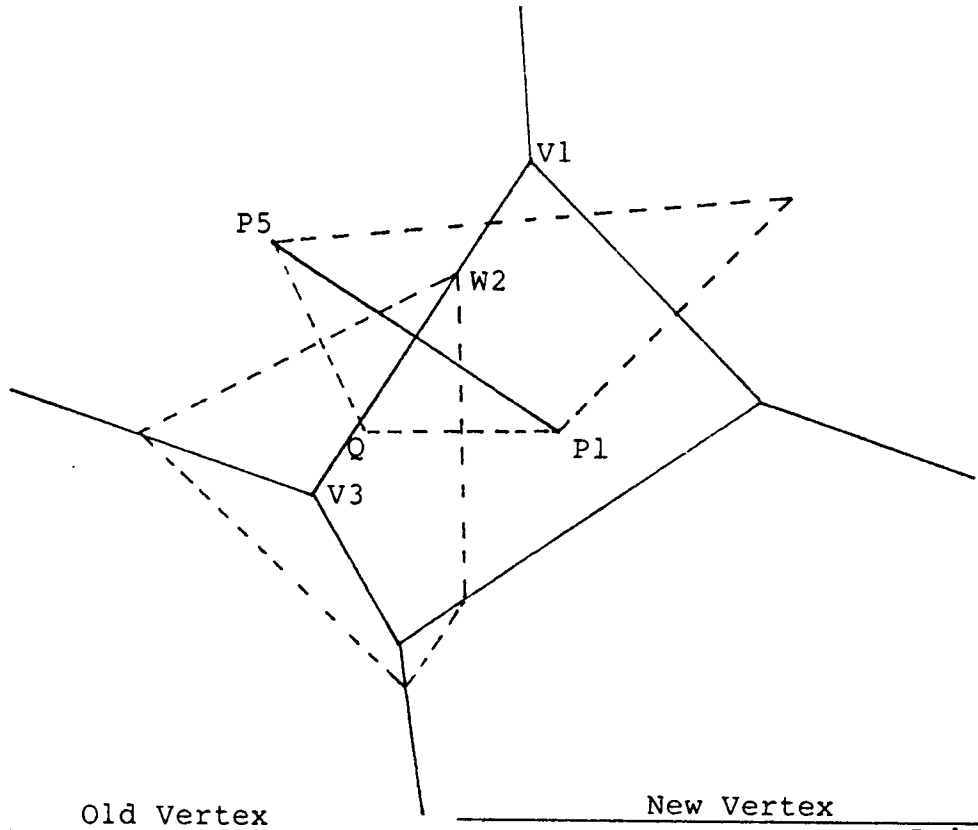


V	R	F.P.			N.V.			Deleted V
W1	?	Q	?	?	0	?	?	V3
W2	?	Q	?	?	V1	?	?	V3
W3	?	Q	?	?	V2	?	?	V4
W4	?	Q	?	?	0	?	?	V4

Figure 7. Newly Generated Structure when Adding a Point Q.

Step 2: Allocate a new set of vertices if necessary.

Since some vertices may be deleted in Step 1, the need for new vertices (W1) is established in this step. In order to do this, every neighboring vertex of the vertex in LISTOFV (obtained from Step 1) is examined. If the neighboring vertex is not in LISTOFV, i.e., the vertex (V1) opposite to a vertex (V3) in LISTOFV is not itself deleted, then a new vertex (W2) is needed. Its forming points should contain the new added point (Q), and the vertex (V1) which is not deleted should be a neighboring vertex of this newly allocated vertex (W2).



Old Vertex				New Vertex				
V	R	F.P.	N.V.	V	R	F.P.	N.V.	Deleted V
V3	R3	P5 P1 P4	V4 0 V1	W2	?	Q P5 P1	V1 ? ?	V3

Figure 8. Common Forming Points of V1 and W2.

Step 3: Find the forming points of the newly generated vertices.

According to Step 2, the new vertex (say W2) is generated because the neighboring vertex (V1) of the vertex (V3) in LISTOFV is not deleted, and the two originally contiguous triangles (triangles of V1 and V3) have two points in common (P5 and P1). Therefore, the triangle of the new vertex (W2) and the triangle of the vertex not deleted (V1) should also have the same two forming points in common.

Table 3

Step 4 in Adding Point[a]

V	R	F.P.				N.V.	
W1	RW1	Q	P5	P4	0	?	?
W2	RW2	Q	P5	P1	V1	?	?
W3	RW3	Q	P3	P1	V2	?	?
W4	RW4	Q	P3	P4	0	?	?

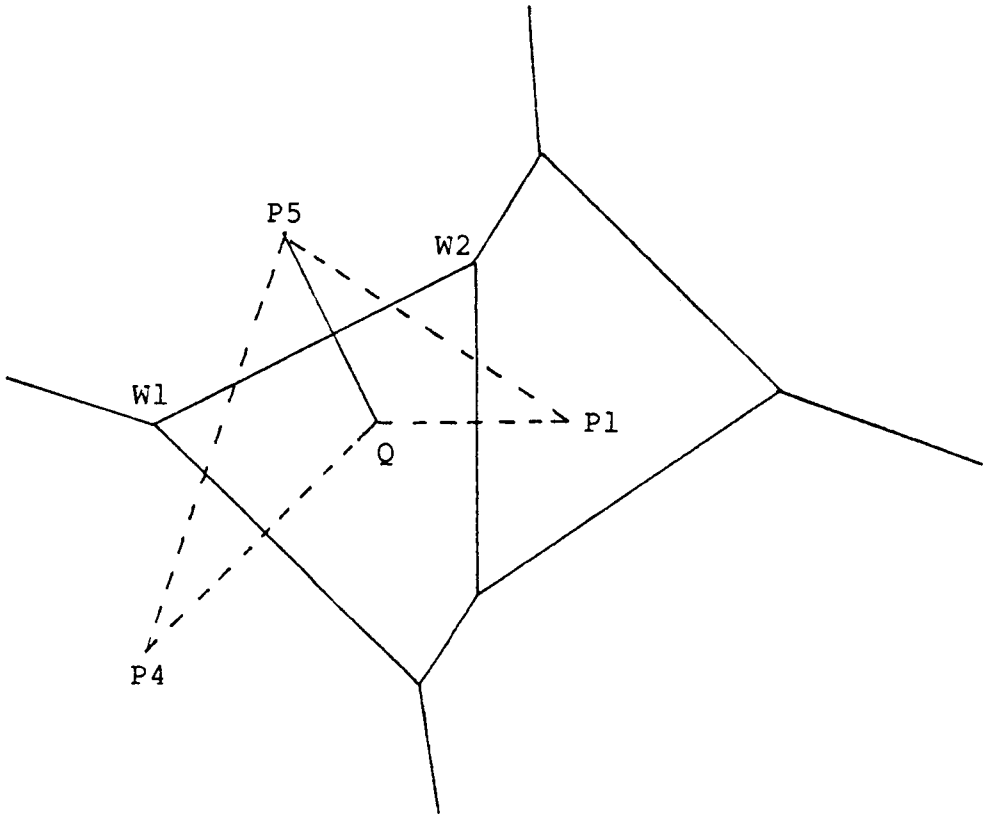
[a]Step 4: Calculate the radius of the circle defined by its forming points. By Theorem 1, the vertex is the geometric center of its forming points. Applying the same procedure to all the new vertices, the newly generated part of the structure is stored as shown.

Table 4

Updating the Neighboring Vertices[a]

	V	N.V.		
Old Status	V1	0	V3	V2
New Vertex	W2	V1	?	?
New Status	V1	0	W2	V2
	V2	0	W3	V1

[a]Step 5: Update the neighboring vertices of related vertices. Since a new vertex is generated only when a neighboring vertex of a vertex in LISTOFV is not deleted, it is efficient to update the neighboring vertices in the current structure immediately. Instead of searching the entire record to update the neighboring vertices containing vertices which are deleted at the end, doing this at this point will save operation time. In this example, records of V1 and V2 are updated as shown.



V	R	F.P.			N.V.		
W1	RW1	Q	P5	P4	0	?	W2
W2	RW2	Q	P5	P1	V1	?	W1
W3	RW3	Q	P3	P1	V2	?	?
W4	RW4	Q	P3	P4	0	?	?

Figure 9. Triangles of Two Neighboring Vertices W1 and W2.

Step 6: Find the neighboring vertices of the newly generated vertices.

If two vertices are neighbors (say W1 and W2), then their associated triangles must have two forming points in common (Q and P5) which always contain the new point. Therefore, if only one forming point is different between two triangles, the two associated vertices must be contiguous (i.e., they are neighboring vertices).

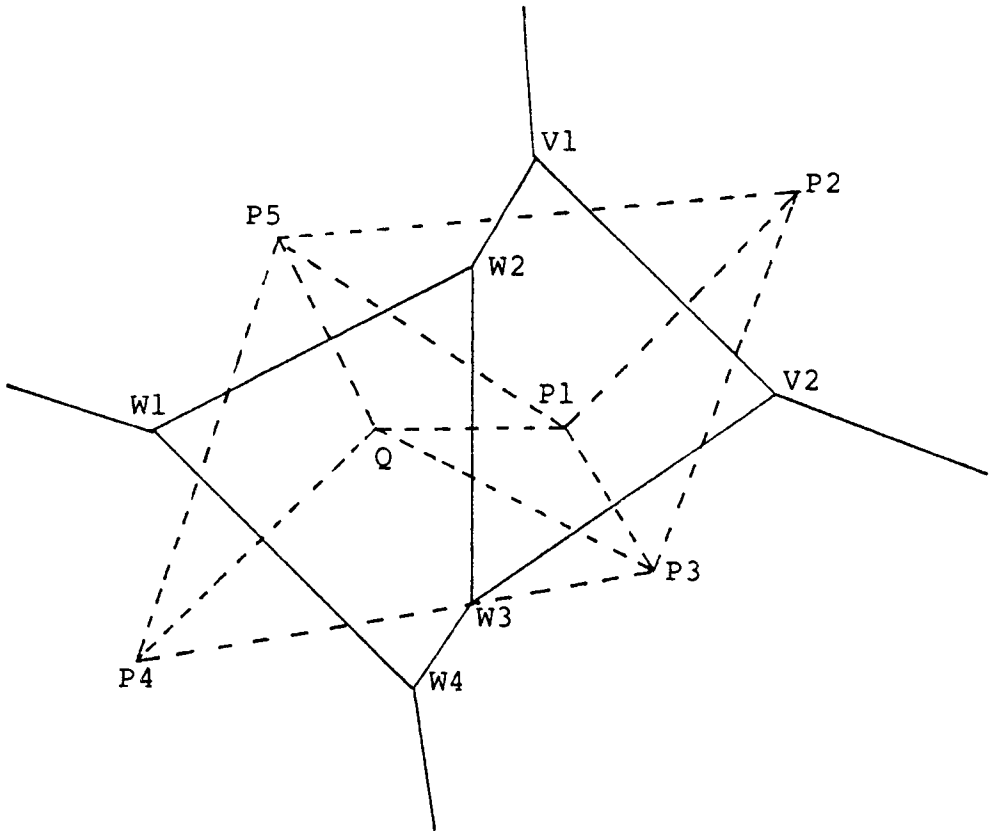


Figure 10. Final Structure After Adding a Point Q.

Table 5

Numerical Structure After Adding a Point Q

V	R	F.P.				N.V.	
V1	R1	P1	P2	P5	0	W2	V2
V2	R2	P1	P2	P3	0	W3	V1
W1	RW1	Q	P5	P4	0	W4	W2
W2	RW2	Q	P5	P1	V1	W3	W1
W3	RW3	Q	P3	P1	V2	W2	W4
W4	RW4	Q	P3	P4	0	W1	W3

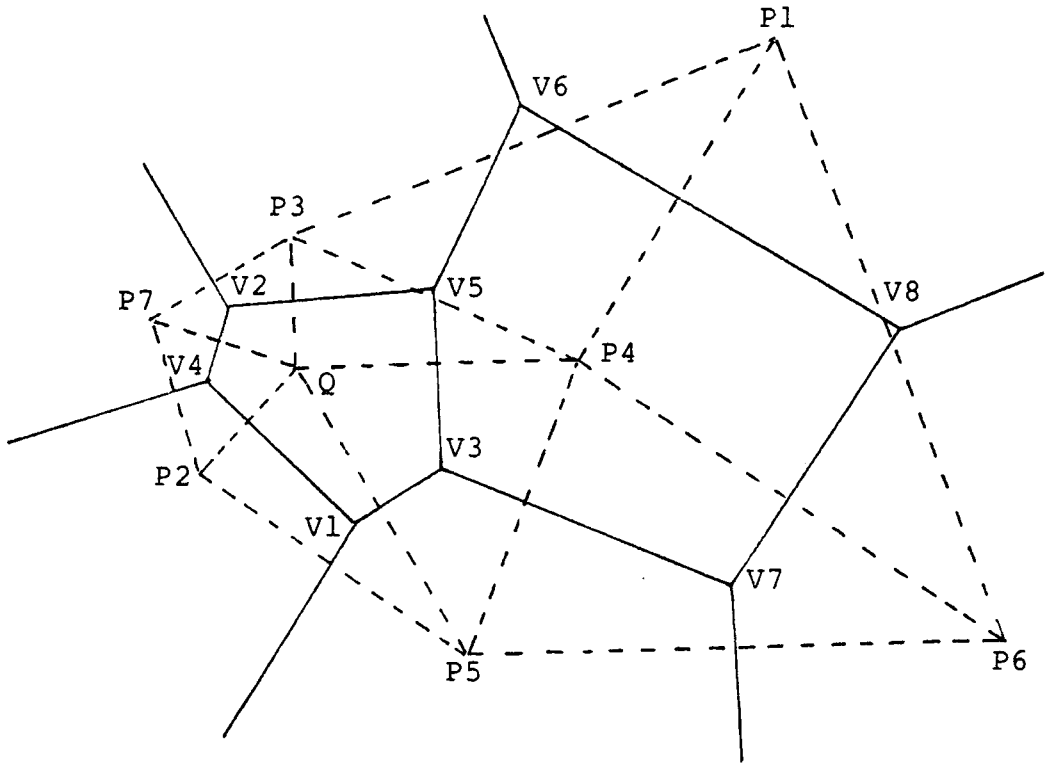
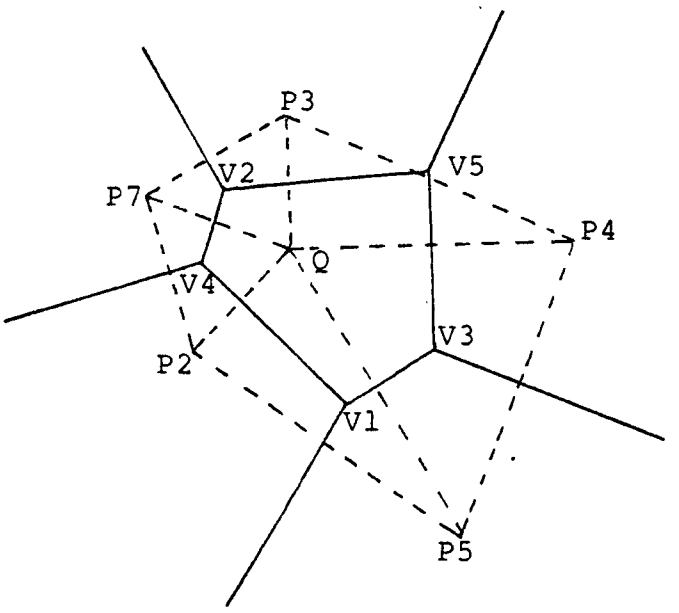


Figure 11. Example Structure for Deleting Point.

Table 6

Numerical Structure for Deleting Point

V	R	F.P.			N.V.		
V1	R1	P2	Q	P5	V3	0	V4
V2	R2	P7	P3	Q	V5	V4	0
V3	R3	P5	P4	Q	V5	V1	V7
V4	R4	Q	P2	P7	0	V2	V1
V5	R5	P3	Q	P4	V3	V6	V2
V6	R6	P3	P4	P1	V8	0	V5
V7	R7	P5	P6	P4	V8	V3	0
V8	R8	P1	P4	P6	V7	0	V6

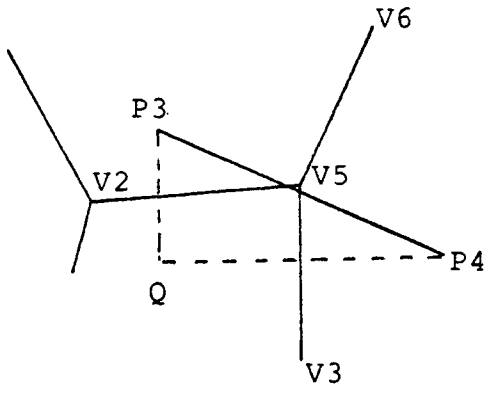


V	F.P.		
V1	P2	Q	P5
V2	P7	P3	Q
V3	P5	P4	Q
V4	Q	P2	P7
V5	P3	Q	P4

Figure 12. Related Structure for Deleting a Point Q.

Step 1: Find the list of vertices that will be deleted by deleting this point. At the same time, generate a list of their forming points not including the deleted point; LISTOFV and LISTOFP will be used for the list of vertices and points, respectively.

The vertices that should be included in this list are those which have the deleted point as one of their forming points. In this example, LISTOFV = {V1,V2,V3,V4,V5} and LISTOFP = {P2,P5,P7,P3,P4}.



V	R	F.P.			N.V.			Deleted V
V5	R5	P3	Q	P4	V3	V6	V2	
W1	?	P3	P4	?	?	?	V6	V5

Figure 13. Example of Allocating New Vertex.

Step 2: Allocate a new set of vertices from the points in the list of points LISTOFP.

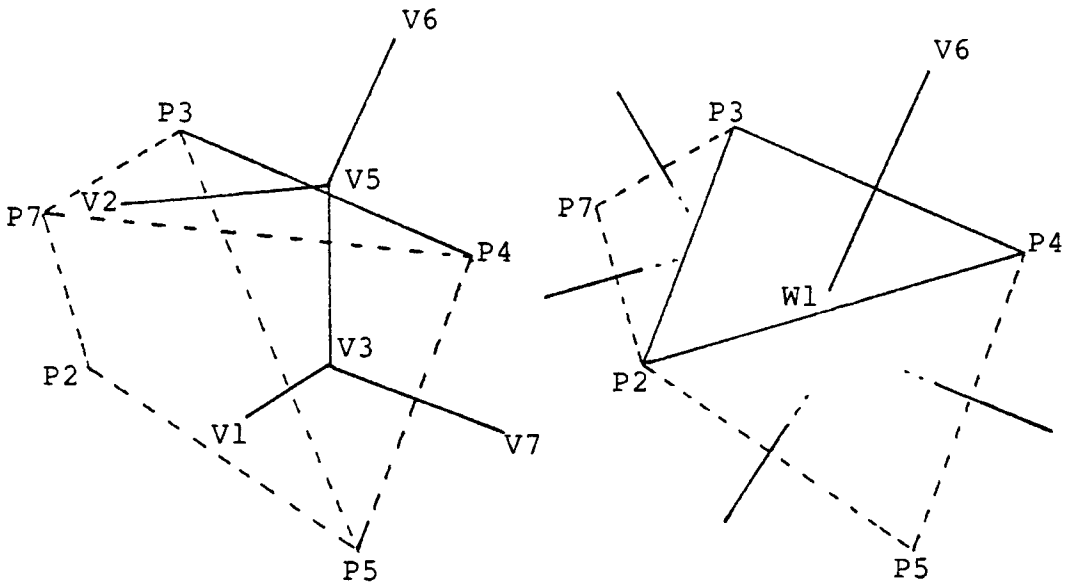
Starting from any vertex (V5) in the list of vertices obtained from Step 1, the neighboring vertices (V3,V6,V2) are examined. If the neighboring vertex (V6) is not in the list (LISTOFV), then a new vertex (W1) is generated. Its forming points should contain the common points (P3,P4) of the two associated triangles.

Figure 14. Procedure for Finding the Last Forming Point of W1.

Step 3: Find the last forming point of newly generated vertices.

For example, to find the last forming point of W1, a contiguous point of the existing forming points (P3,P4) is to be selected from LISTOFP. In this example, P5 is selected.

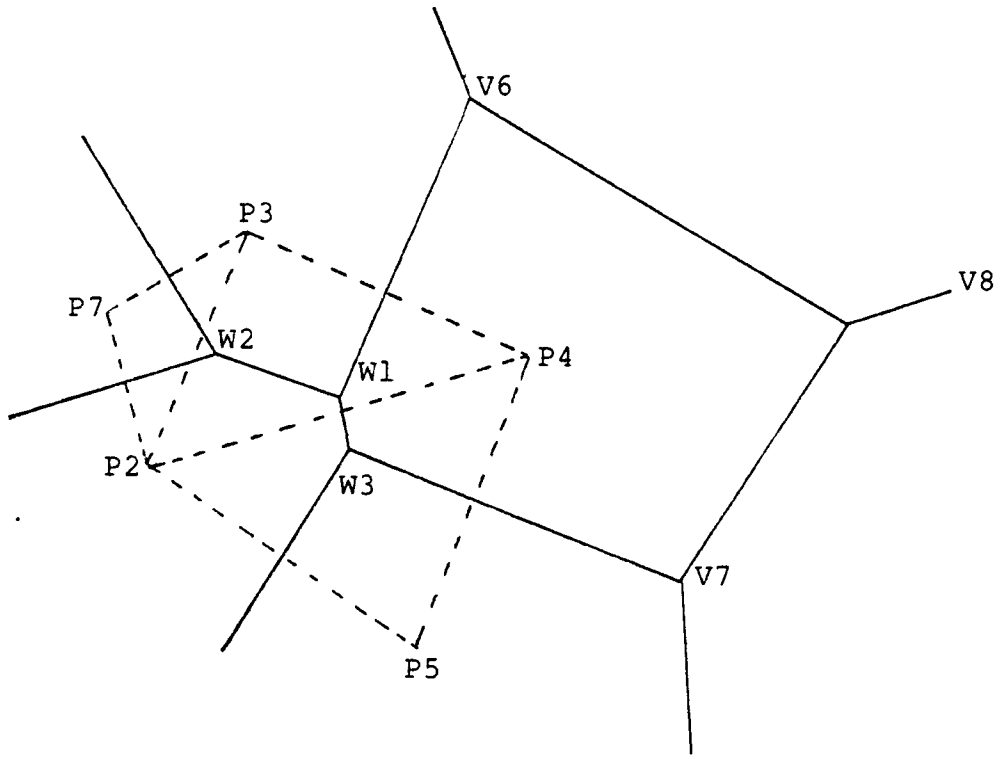
By Definition 1 and Theorem 2, the three forming points lie on the circle and no other points lie within it. All the points in LISTOFP are examined. If no other points lie within the circle defined by P3, P4, and P5, the algorithm enters Step 4. Otherwise the closest point (P2) is chosen as the last forming point.



V	F.P.			N.V.		
V2	P7	P3	Q	V5	V4	Q
V3	P5	P4	Q	V5	V1	V7
V5	P3	Q	P4	V3	V6	V2

V	R	F.P.			N.V.			Deleted V	
W1	RW1	P3	P4	P5	?	V7	V6	V5	V3
W1	RW1	P3	P4	P2	?	?	V6	V5	

Figure 14.



V	R	F.P.			N.V.		
W1	RW1	P3	P4	P2	?	?	V6
W2	RW2	P7	P3	P2	?	0	0
W3	RW3	P2	P5	P4	V7	?	0

Figure 15. Newly Generated Structure After Deleting a Point Q.

Step 4: Applying the procedure in Step 3 to the rest of the vertices in LISTOFV gives the newly generated structure as shown.

Table 7

Step 5 in Deleting Point[a]

V	N.V.		
V6	V8	0	W1
V7	V9	W3	0

[a]Step 5: Update the neighboring vertices of the related vertices. This step is similar to Step 5 shown on Table 4, page 12; V6 and V7 are updated as shown.

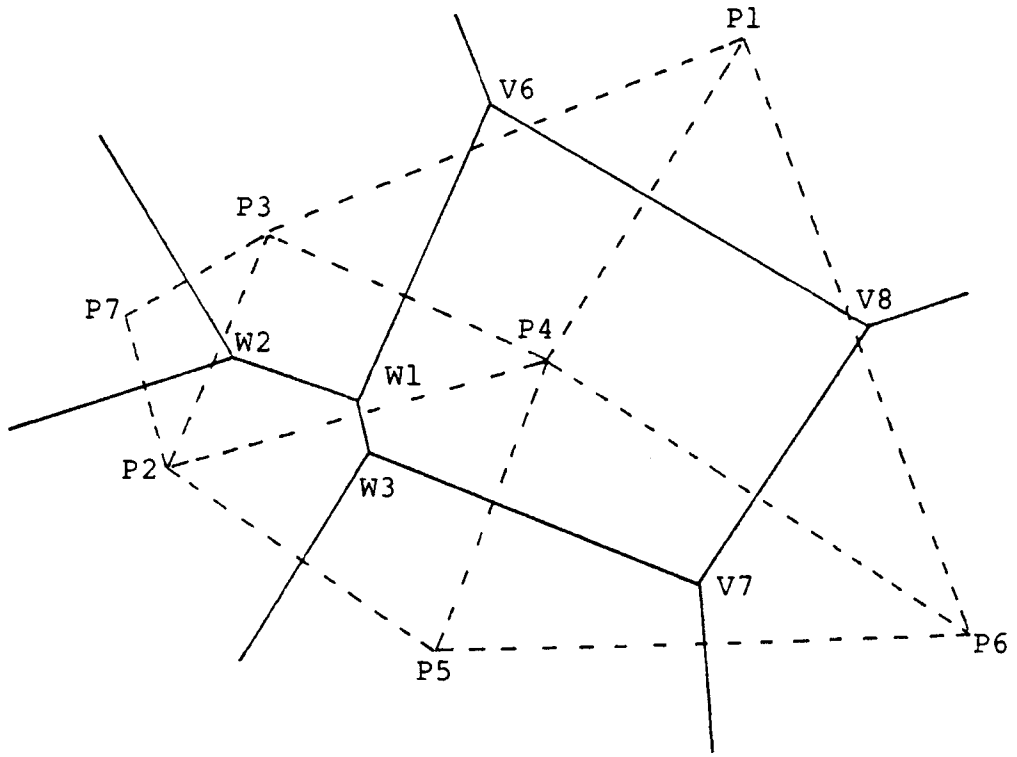


Figure 16. Final Structure After Deleting a Point Q.

This step is similar to Step 6 described in Fig. 9, page 22, in the algorithm for adding a point into the structure.

Table 8

Numerical Structure After Deleting a Point Q

V	R	F.P.				N.V.	
V6	R6	P3	P4	P1	V8	0	W1
V7	R7	P5	P6	P4	V8	W3	0
V8	R8	P1	P4	P6	V7	0	V6
W1	RW1	P3	P4	P2	W3	W2	V6
W2	RW2	P7	P3	P2	W2	0	0
W3	RW3	P2	P5	P4	V7	W1	0

introduced in Section 2.3, every Delaunay simplex can be represented by a vertex. Basically, the tessellation is an aggregate of simplices where the pieces of information of all vertices and the associated Delaunay simplices have to be linked together in order to represent the whole construct. It is thus possible to represent the construct using a double-linked list of vertices which allows for easy insertion and deletion of vertex records. Due to limited storage space for records, another linked list of available space is represented.

Suppose that 10 is the maximum number of simplices that can be stored in the reserved storage space. Two double-linked lists using all the 10 spaces are employed. One is the list of vertices for the current tessellation; FIRSTV and LASTV point to the first and last element in the list, respectively. The other linked list starts at location FIRSTA, which is the first available space in the allocated storage space.

For example, for the structure introduced in Fig. 5, page 16, several arrays are used to represent the structure, as shown in Table 9.

In the final structure of the above example (as shown in Fig. 10, page 23), space originally occupied by simplices and vertices deleted by the addition of the new point are linked to the list of available spaces

Table 9

Exact Data Stored

I	R(I)	IFP(J,I)			NV(J,I)			IPARENT(I)	ICHILD(I)	
1	R(1)	1	2	5	0	3	2	0	0	← FIRSTV
2	R(2)	1	2	3	0	4	1	1	3	
3	R(3)	5	1	4	4	0	1	2	4	
4	R(4)	3	4	1	3	2	0	3	0	← LASTV
5								0	6	← FIRSTA
6								5	7	
7								6	8	
8								7	9	
9								8	10	
10								9	0	

(see Table 10). The linked lists of data are not only a well-represented network of simplices, but are also a nice structure for "garbage collection."

Another storage arrangement for the data base concerns ordering the neighboring simplices. In a list of "n" elements, there are n! different permutations for storing elements. Therefore, the three forming points of a triangle can be stored in six different orders, although they all represent the same triangle. Likewise, three neighboring vertices opposite the center vertex can also be stored in six different orders. The aim is to store the neighboring vertices to simplify the retrieval of the construct from the data base.

This goal is achieved by using the two characteristics of the binary number system. Consider two lists (given in Table 11) of unsigned binary integers both starting with zero and having either three or four digits. First, the list of unsigned binary integers falls naturally into an ordinal sequence. Second, if n bits are used there are exactly n out of the 2^n numbers having only one bit equal to zero, while the remaining n-1 bits are equal to one. These features of the binary number system provided the consistency and unambiguity needed for the data base of this algorithm.

Table 10

Data Stored for Final Structure

I	R(I)	IFP(J,I)			NV(J,I)			IPARENT(I)	ICHILD(I)	
1	R(1)	1	2	5	0	6	2	0	2	← FIRSTV
2	R(2)	1	2	3	0	7	1	1	5	
3	R(3)	3	4	1	4	2	0	0	4	← FIRSTA
4	R(4)	5	1	3	3	0	1	3	9	
5	R(5)	6	5	4	0	8	6	2	6	
6	R(6)	6	5	1	1	7	5	5	7	
7	R(7)	6	3	1	2	6	8	6	8	
8	R(8)	6	3	4	0	5	7	7	0	← LASTV
9								4	10	
10								9	0	

Table 11

List of Unsigned Binary Integers

Bit 1	Bit 2	Bit 3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Bit 1	Bit 2	Bit 3	Bit 4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Those numbers which have only one zero bit form an n-tuple which defines a segment of the Delaunay simplex, as shown in Table 12. Each bit represents one forming point of the triangle in two-dimensional (tetrahedron in three-dimensional) space. Using the natural sequence of binary number, the neighboring vertices can be recorded in an unambiguous order. In conclusion, the forming points can be stored in an arbitrary order but the neighboring vertices are always stored as described above.

2.5. The Program

ALGORITHM + DATA STRUCTURE = PROGRAM .

An interactive computer program which is based on the data structure and algorithm discussed in Sections 2.3 and 2.4 is introduced in this section. The Appendix contains the listing of the program.

The first task of this program is to set up the initial construct. Since the computer representation of the tessellation is a linked list of vertices and their associated Delaunay simplices, the simplest construct naturally has only one simplex and one associated vertex. The question is how to build the initial simplex.

An initial simplex which is large enough to include all data points is highly desirable. Obviously, to make the first input data point valid, a non-empty list

Table 12

Relation Between the Binary Numbers and the Forming Points

	P1	P2	P3	P1	P2	P3	P4
First	0	1	1	0	1	1	1
Second	1	0	1	1	0	1	1
Third	1	1	0	1	1	0	1
Fourth				1	1	1	0

of vertices that will be deleted by this point should be found. Hence, the initial simplex should be large enough to start with. Furthermore, the boundary regions in any tessellation always extend to infinity, which can be difficult to handle. If a very large initial construct can be set up, it will not only meet the requirement mentioned above, but will also act as a virtual window for display purposes.

Figure 17 is a flow chart giving an outline of the entire program.

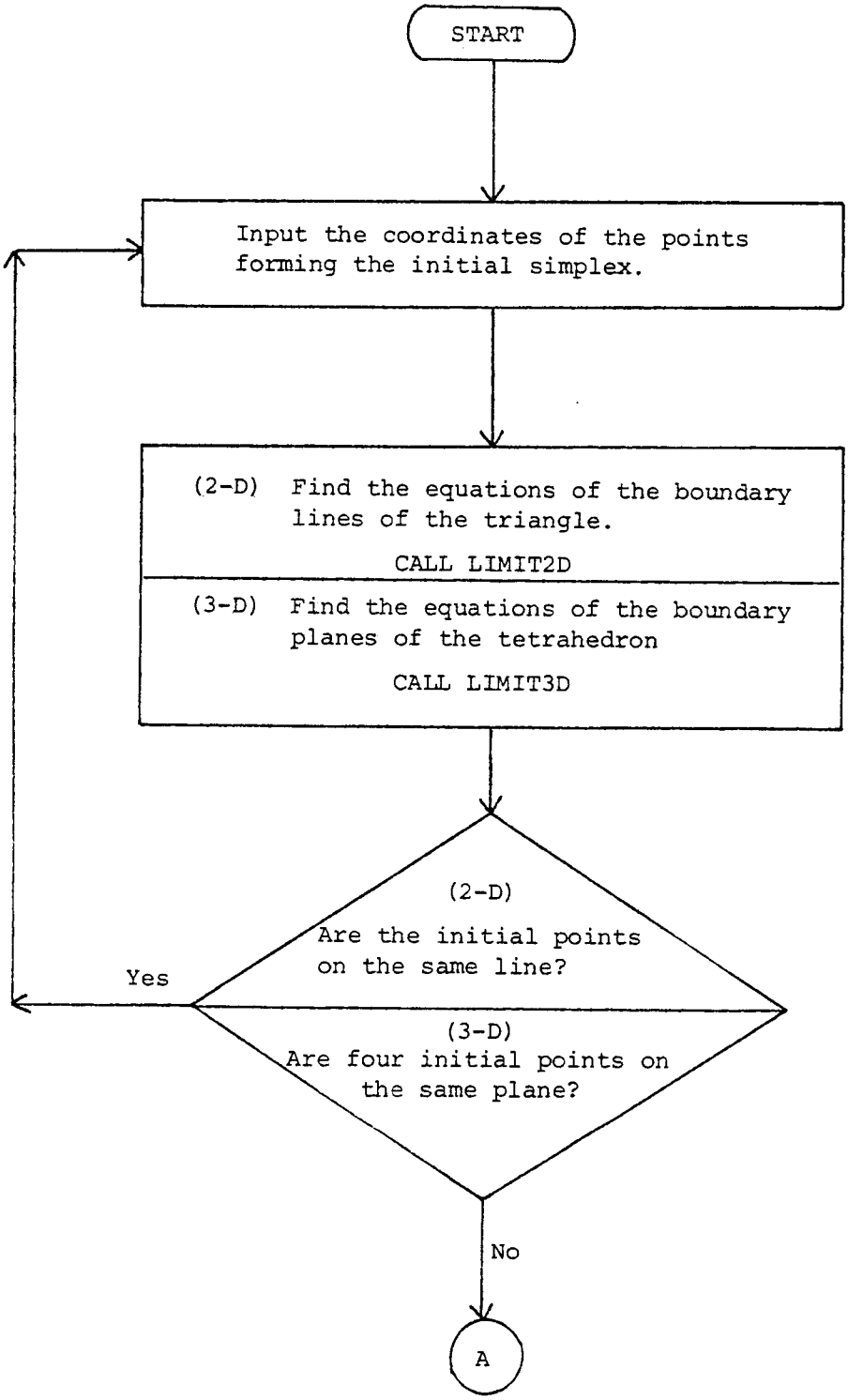


Figure 17. Flow Chart of the Algorithm.

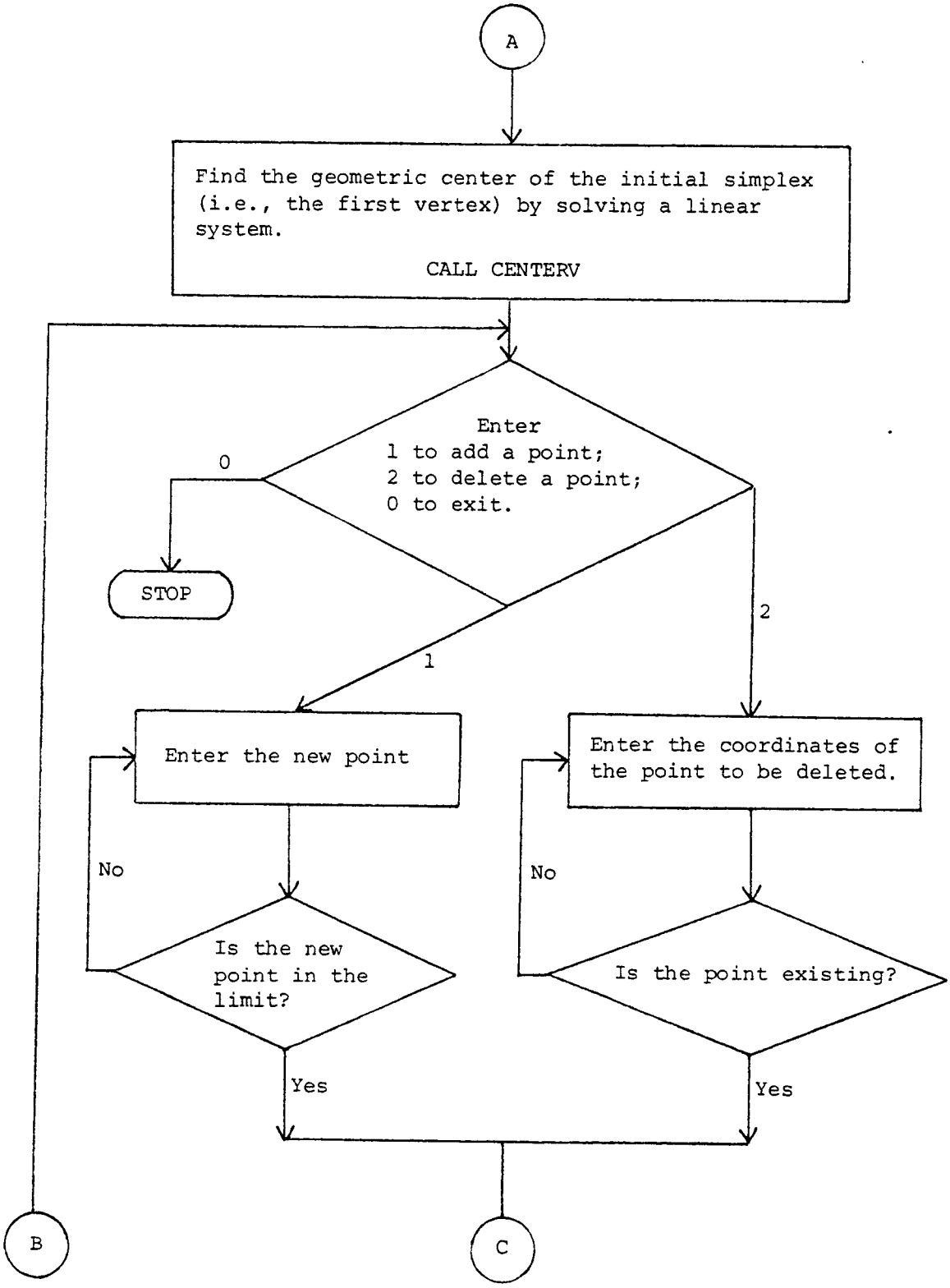


Figure 17. (Continued)

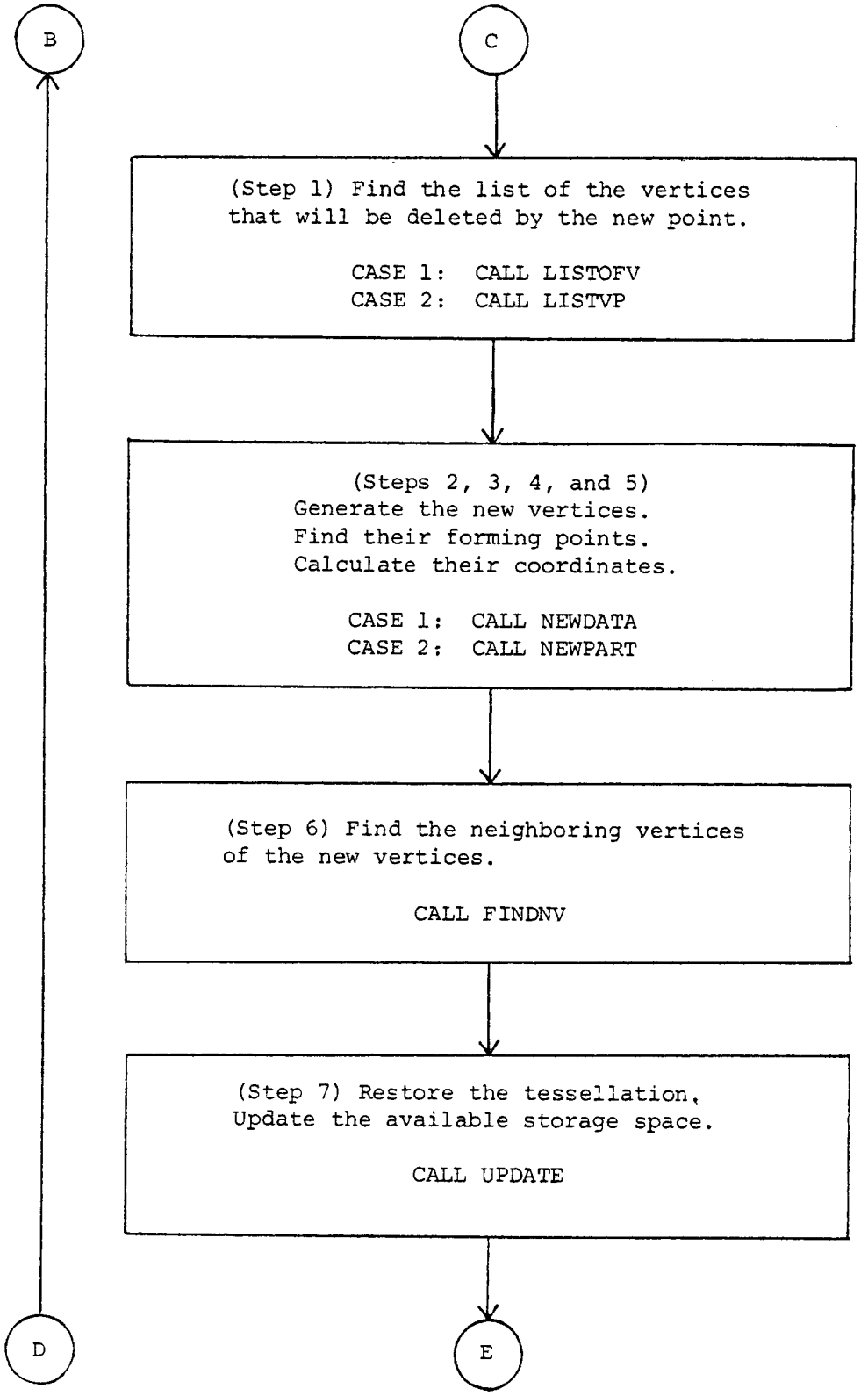


Figure 17. (Continued)

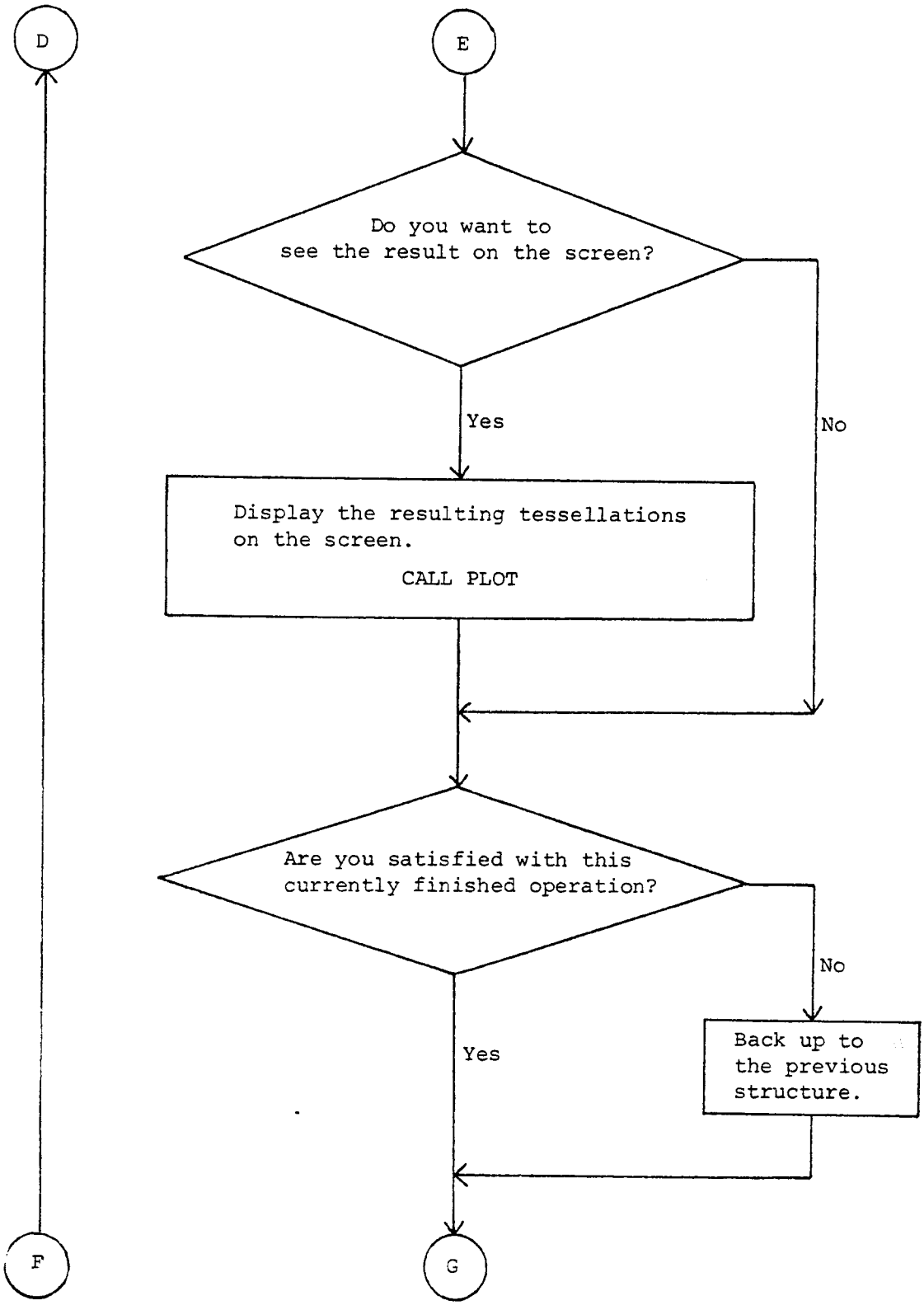


Figure 17. (Continued)

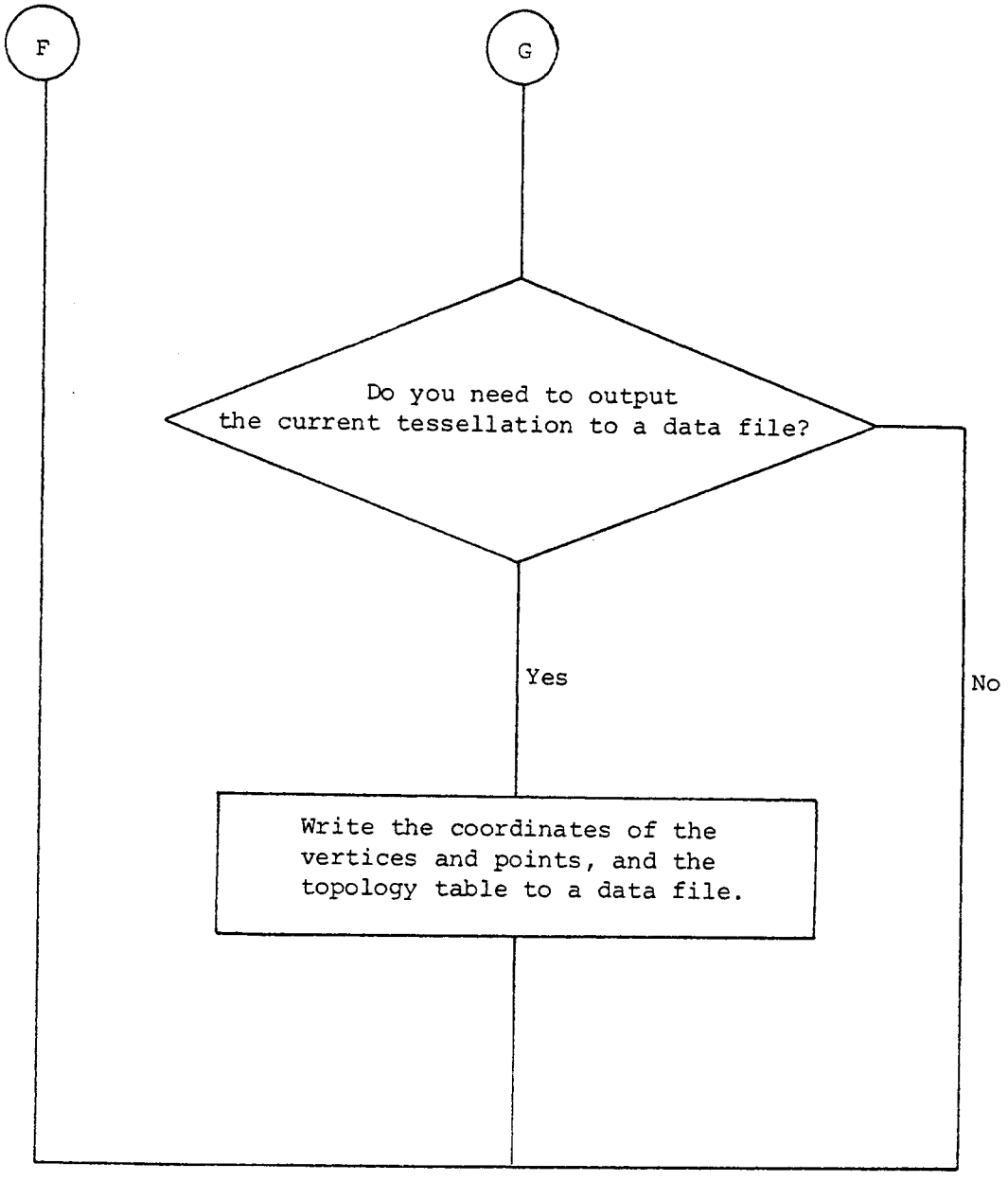


Figure 17. (Continued)

CHAPTER III

RELATED WORK AND APPLICATIONS

As with many other mathematical ideas, the Voronoi tessellation and Delaunay triangulation have been applied in various forms in different fields of study. This chapter highlights some practical applications of these tessellations in geography, statistics, crystallography, and mathematics.

3.1. Geography

To a geographer solving location-allocation problems, "Thiessen polygon" is a familiar term. Most solutions employ an essentially geographical procedure of searching for spatial efficiency. Another application is for the well-known Weber problem in which the route of minimum aggregate travel among a set of locations is determined.

One problem in which Voronoi tessellation finds good application is apportioning of the area of a map into regions about the same center points so that every location in a region is nearer to that region's center than to any other center. A simple geometric method to solve the above problem manually was proposed at the beginning of the twentieth century by climatologist A. H. Thiessen (cited by Taylor [7]). In the problem

Thiessen considered, the aim was to define regions around rainfall stations so that the total rainfall for several stations in one area could be weighted by the area they represented in computing an average rainfall for the whole area. The origin of the name "Thiessen polygon" for the region referred to as "Voronoi polygon" is due to the studies conducted by Thiessen. A reference to the Thiessen polygon will not be complete without mentioning Rhynsburger, who gave an analytic delineation of Thiessen polygons in 1973 [1].

3.2. Statistics and Crystallography

In statistics, especially in the study of point processes and spatial data analysis, the applications of planar Voronoi tessellation and Delaunay triangulation are well established. Miles [8] generated these random polygonal tessellations by a homogeneous planar Poisson point process and then conformed them to various distributions. Miles derives the theoretical and statistical properties of the planar Voronoi and Delaunay tessellations. Ripley [9] used the Voronoi polygons to describe the random pattern of biological cells.

An interesting application in three-dimensional space is in the field of crystallography, where the Voronoi tessellation is used to model crystal growth (referred to

as the cell model). Instead of Thiessen polygons or Voronoi polyhedra, the regions are called "crystals." Meijering [10] and Gilbert [11] have discussed the statistical values (for example, the mean values, the variances, etc.) of the distribution of crystal volumes.

3.3. Mathematics

Although the study of the properties of tessellations or simplices began early in the twentieth century, the mathematical background and properties of Voronoi and Delaunay tessellations were clearly defined only recently by Rogers [12]. Rogers describes the construction of the Voronoi polyhedra associated with a distribution of points throughout a space.

The determination of a region of feasible solutions of a linear objective function, given n linear inequalities, is achieved by linear programming techniques. The extremum solution occurs at the vertex of the region. From a pure mathematical point of view, the region is the intersection of n half-spaces. According to the definition of the Voronoi polyhedron given by Rogers, the feasible region in a linear programming problem is similar to the Voronoi Polyhedron. In 1979, Preparata and Muller [13] were responsible for developing an algorithm to find the intersection of n half-space, which requires only time $O(n \log n)$.

Among the various applications in the fields of applied mathematics, engineering science, computer graphics, and the production of contour maps, surface fitting and smooth interpolation among random points are important applications of the Delaunay triangulation.

McLain [14] described a method for smooth interpolation from random data points in two or more dimensions which gave a smooth surface passing exactly through those given data points. The first step in the method is to partition the plane into triangles by connecting the neighboring data points. A Delaunay triangulation based on the data points is well suited for this.

Powell and Sabin [15] introduced two methods of constructing piecewise quadratic approximations on triangles.

Lawson [16] presented an interesting property of triangulation, which is known as the max-min angle criterion. The criterion can be stated as follows: If two triangles in the triangulation share a common edge, they define a quadrilateral with that common edge as the diagonal. If that quadrilateral is strictly convex, then replacement of the chosen diagonal by the alternative one must not increase the minimum of the six angles in the two triangles making up the quadrilateral. This property is termed as locally equiangular.

Sibson [17] proved that there is a unique locally equiangular triangulation; namely, the Delaunay triangulation. As a triangular grid on which the interpolation is based, the Delaunay triangulation is probably the best triangulation because it has been shown to be "nearly equiangular."

For more general n-dimensional cases, a vector identity for the Voronoi tessellation was discussed by Sibson [18]. Applications in interpolation and smoothing problems in data analysis are also given.

The present work was a direct result of the need for good two-dimensional mesh generation capabilities for studying the potential flow around aerofoils using finite element techniques. The importance of the functional relations between the data points is well known in the finite element analysis of aerodynamic problems. The details of this application are not dealt with in this study. However, it may be said that the finite element method is a general approximation process applicable to a wide variety of engineering problems. In the finite element method the domain is divided into a discrete number of elements over each of which the trial functions apply.

The need for the development of any mathematical theory or algorithm is judged mainly by the degree of its

applicability in solving practical problems. The applications described in this chapter are just a few among the seemingly endless and exciting applications of this relatively new science of Voronoi tessellation and Delaunay triangulation.

CHAPTER IV

CONCLUSIONS AND RECOMMENDATIONS

Often a computer program for solving problems is good enough as long as it "works." However, when computational resources are limited, the simplicity, efficiency and precision of any algorithm or program become more important.

4.1. Efficiency of the Algorithm

In the algorithm discussed in the present study, the amount of work is directly proportional to the number of vertices to be generated, assuming that the work involved in searching for vertices to be deleted is negligible. Similarly, the time expended by the search routine to locate the vertices to be deleted increases as the number of data points increases.

The original searching technique used in the algorithm simply calculates the distance between the new point and each vertex then compares each distance with the associated radius one by one. This is the simplest implementation and the most efficient one for a small-scale configuration. Since some of the applications of the tessellations involve complex data analysis, the searching process in Step 1 should be properly modified. One

possible method is to treat it as a nearest neighbor search problem. Proper sorting of the vertices may be needed as a part of the preprocessing for nearest neighbor search. The vertices neighboring those already in the list of vertices to be deleted are compared until no more need to be deleted. The vertices included in the list are now closer to the new point than to any of its forming points.

The method described briefly above uses the local nature of the searching process instead of checking all the vertices in the current structure, which should decrease the operation time when the size of the data base is large.

Because of the limited storage space available in implementing the algorithm described previously, it is restricted due to the memory allocation capability provided by the computer language FORTRAN. The two double-linked lists (one for the construct and one for the unused space) fully use the space which is allocated at the beginning of the program. The operation of memory space arrangement could be optimized if a language which allows dynamic memory allocation (for example, PASCAL) were used.

4.2. Degeneracy

Another factor to be considered is degeneracy, which occurs when more than n+1 regions meet at a vertex in an n-dimensional space. In two dimensions, degeneracy will occur when four data points are cyclic. In other words, they all lie on one circle and the center is located at the vertex where the four associated regions meet. Such data most likely are intentionally placed on a regular grid, in which case it is usually easy to generate the structure without using the algorithm introduced in this work.

A second type of degeneracy occurs when all the data points are not distinct. Recalling the definitions in Chapter I, the data points used to construct the tessellations are all distinct, and the regions are contiguous only if they meet along a boundary segment instead of just at a vertex. Therefore, the degenerate cases are considered invalid. However, in reality, the degenerate case cannot always be avoided. Therefore, proper modifications in the program should be made. When the newly added point lies on the same surface of a hypersphere on which the forming points of any simplex in Delaunay triangulation lie (i.e., when the distance between the vertex and the new point is equal to the radius associated to that

vertex), the decision of whether to include the vertex in the list of vertices to be deleted or not is to be made.

The program listed in the Appendix does not include the degenerate vertex in the list of vertices to be deleted. An example of degeneracy in two dimensions is shown in Fig. 18.

4.3. Epilogue

Many scientific problems are essentially solving equations and dealing with various functions. Irregular data points are used in many different problems. Usually, the information concerning the distribution of the data points (such as a grid or equilateral triangulation) is required before comparisons of the functional values at these data points are meaningful; it makes computation of the Voronoi tessellation and Delaunay triangulation, which determine an irregular lattice, more important and essential.

In conclusion, it is interesting to note that a consistent similarity has been observed among such things as bubbles in froth, living tissues, cells of dragonfly wings, spots on a giraffe, and cracks in dried mud. The Voronoi structure seems to appear in various systems in the natural world, too!

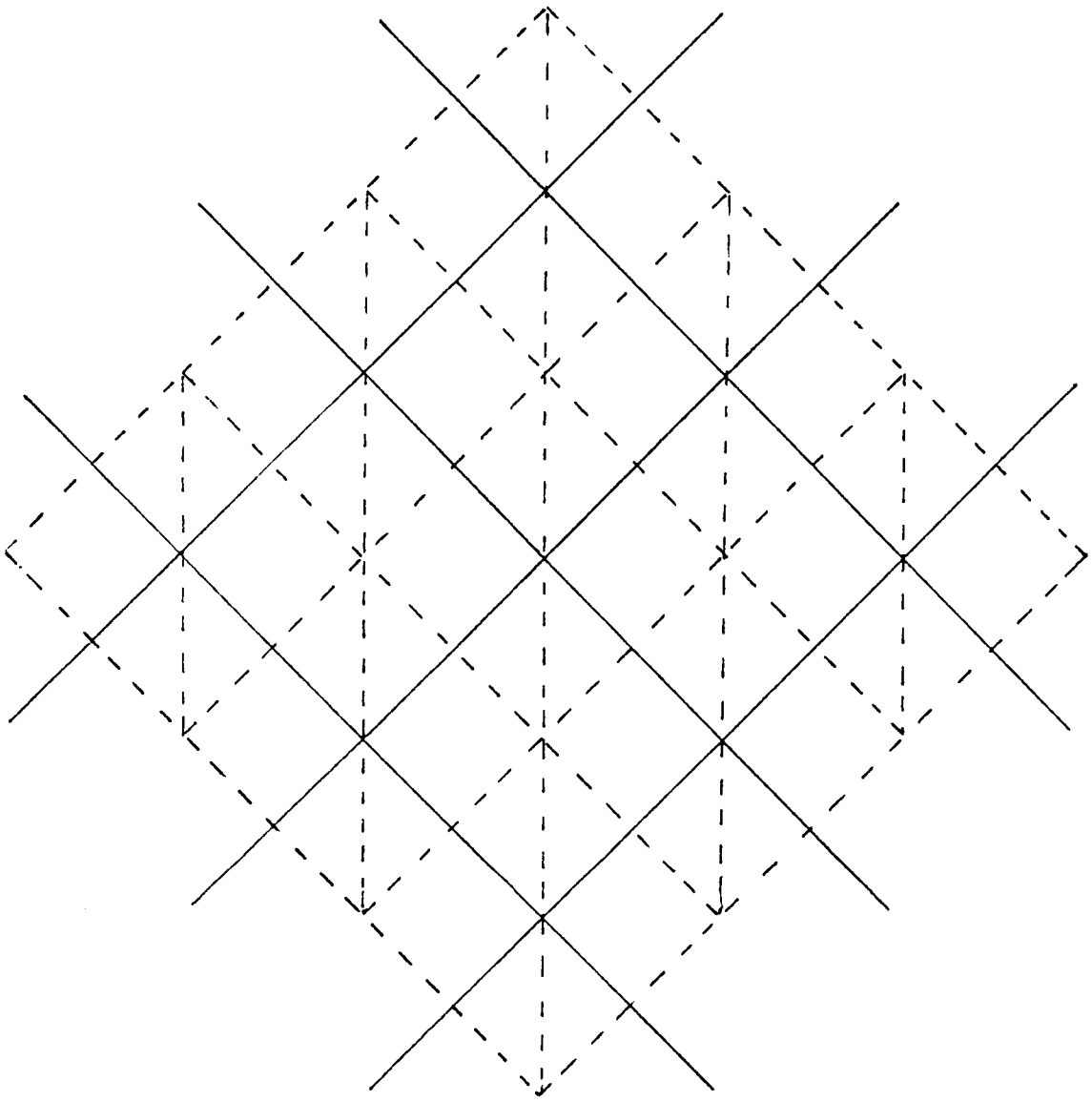


Figure 18. Example of a Two-Dimensional Degenerate Case.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Rhynsburger, Dierk. "Analytic Delineation of Thiessen Polygons," Geographical Analysis, 5:133-144, 1973.

2. Shamos, M. I. "Geometric Complexity," Proceedings of the Seventh Special Interest Group Automata and Computability Theory (SIGACT) Conference. New York: Association for Computing Machinery, Inc., 1975. Pp. 224-233.

3. Green, P. J., and R. Sibson. "Computing Dirichlet Tessellations in the Plane," The Computer Journal, 21(No. 2):168-173, 1978.

4. Brown, Kevin G. "Voronoi Diagrams from Convex Hulls," Information Processing Letters, 9(No. 5):223-228, December 1979.

5. Watson, D. F. "Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes," The Computer Journal, 24(No. 2): 167-172, 1981.

6. Bowyer, A. "Computing Dirichlet Tessellations," The Computer Journal, 24(No. 2):162-166, 1981.

7. Taylor, Peter J. Quantitative Methods in Geography: An Introduction to Spatial Analysis. Boston: Houghton-Mifflin Company, 1977.

8. Miles, R. E. "On the Homogeneous Planar Poisson Point Process," Mathematical Biosciences, 6:85-127, 1970.

9. Ripley, B. D. "Modelling Spatial Patterns," Journal of the Royal Statistical Society, 39(Series B): 172-192, 1977.

10. Meijering, J. L. "Interface Area, Edge Length, and Number of Vertices in Crystal Aggregates with Random Nucleation," Philips Research Lab Report 8, Eindhoven, Netherlands, 1952.

11. Gilbert, E. N. "Random Subdivisions of Space Into Crystals," Annals of Mathematical Statistics, 33:958-972, 1962.

12. Rogers, C. A. "Packing and Covering," Cambridge Mathematical Tract No. 54. Cambridge, Massachusetts: Cambridge University Press, 1964. Pp. 74-103.

13. Preparata, F. P., and D. E. Muller. "Finding the Intersection of n Half-Spaces in Time 0 (n log n)," Theoretical Computer Science, 8:45-55, 1979.

14. McLain, D. H. "Two-Dimensional Interpolation from Random Data," The Computer Journal, 19(No. 2): 178-181, 1976.

15. Powell, M. J. D., and M. A. Sabin. "Piecewise Quadratic Approximations on Triangles," Association for Computing Machinery (ACM) Transactions on Mathematical Software, 3(No. 4): 316-325, December 1977.

16. Lawson, C. L. "Generation of a Triangular Grid with Application to Contour Plotting," California Institute of Technology Technical Memorandum 299, Jet Propulsion Laboratory, Pasadena, California, 1972.

17. Sibson, Robin. "Locally Equiangular Triangulations," The Computer Journal, 21(No. 3):243-245, 1978.

18. Sibson, Robin. "A Vector Identity for the Dirichlet Tessellation," Mathematical Proceedings of the Cambridge Philosophical Society, 87:151-155, 1980.

APPENDIX

```
      PROGRAM TESSEL
C*************************************************************************
C  PURPOSE     TESSEL computes the Voronoi tessellation and Delaunay
C              triangulation in 2-D or 3-D space, and dispaly the
C              configuration whenever a new data point is added.
C
C
C  ARGUMENTS
C   ON INPUT   DIM
C                     dimension of the space
C
C              XMIN, XMAX, YMIN, YMAX
C                     the virtual window used in plotting routine UWINDO
C
C              VX, VY, VZ
C                     the coordinates of the view point used in plotting
C                     routine UVIEW
C
C              SX, SY, SZ
C                     the coordinates of the site point used in plotting
C                     routine UVIEW
C
C
C   ON OUTPUT  all arguments are unchanged
C
C
C    LANGUAGE  FORTRAN
C
C
C    PLOTTING
C    ROUTINES
```

```
C               U3MOVE, U3PEN, UEND, UMOVE, UPEN, USET, USTART, UVIEW,
C               UWINDO
C
C
C     INTERNAL
C   PARAMETERS
C                 NAME              FUNCTION
C-------------------------------------------------------------------
C                 MAXNV             maximum number of the vertices
C                 MAXNP             maximum number of the points
C                 NOV               number of the vertices
C                 NOP               number of the points
C                 IDP               identification number of the point to be
C                                   deleted
C                 NODP              number of the deleted points
C                 IFIRSTV           the first element in the linked list of
C                                   vertices
C                 LASTV             the last element in the linked list of
C                                   vertices
C                 IFIRSTA           the first available space in the linked
C                                   list
C                 LENLV             the length of the list of deleted vertices
C                 LENLP             the length of the list of related points
C                 NEWV              the number of the newly generated vertices
C                 VERTEX(DIM,IV)    coordinates of the vertex IV
C                 POINT(DIM,IP)     coordinates of the point IP
C                 RADIUS(IV)        radius of the hypersphere associated with
C                                   the vertex IV
C                 ICHILD(IV)        next vertex linked to vertex IV
C                 IPARENT(IV)       the vertex which vertex IV is linked to
C                 IFP(DIM+1,IV)     forming point of vertex IV
```

```
C               NV(DIM+1,IV)    neighbouring vertices of vertex IV
C               LISTV(MAXNV)    the list of all the vertices deleted by
C                               the new point
C               LISTP(MAXNP)    the list of the points related to the
C                               deleted point
C               IOLDFV          previous IFIRSTV
C               IOLDLV          previous LASTV
C               IOLDFA          previous IFIRSTA
C               IOLDCH(IV)      previous ICHILD(IV)
C               IOLDPA(IV)      previous IPARENT(IV)
C               IOLDFP(DIM+1,IV)        previous IFP(DIM+1,IV)
C               IOLDNV(DIM+1,IV)        previous NV(DIM+1,IV)
C               A(4),B(4),C(4),D(4)
C                               the coeeficients of the four plane
C                               equations
C               E1(3),E2(3),E3(3)
C                               the coeeficients of the three line
C                               equations
C               IPOSITION(4)
C                               the flags for checking whether the new
C                               data point is in the limited area defined
C                               by the initial boundary points
C
C
C
C  ALGORITHM    Please see details in the ALGORITHM section in this
C               report.
C
C
C
C  NOTE:        If MAXNV or MAXNP is to be increased the dimensions of
```

```
C               all the variables in COMMON area must be increased by the
C               same amount.
C
C
C
C       WRITTEN BY      Judy Chien-Hui Chang            September 1982
C
C
C
C*************************************************************************
C  Storage allocation
C*************************************************************************
        PARAMETER MAXNV=1000, MAXNP=3000
        DIMENSION CENTER(3), DELPT(3)
        INTEGER  COUNT, NO(4)
        LOGICAL UNDERLIMT
        INCLUDE 'TESSEL.CMN'
C*************************************************************************
C  Initialization
C*************************************************************************
        DO I=1,MAXNV
        ICHILD(I)=I+1
        IPARENT(I)=I-1
        END DO
        ICHILD(MAXNV)=0
C*************************************************************************
C  Input initial data and calculate the boundaries of the initial
C  simplex
C*************************************************************************
        TYPE*, 'PLEASE ENTER THE DIMENSION OF THE SPACE (2  OR  3):'
        ACCEPT*,DIM
```

```
        TYPE*,  'PLEASE ENTER THE VIRTUAL WINDOW - XMIN,XMAX,YMIN,YMAX: '
        ACCEPT*,XMIN,XMAX,YMIN,YMAX
        IF(DIM.EQ.3) THEN
        TYPE*,'PLEASE ENTER THE VIEW POINT AND THE SITE POINT: '
        TYPE*,'    VX,VY,VZ,    SX,SY,SZ ='
        ACCEPT*,VX,VY,VZ,SX,SY,SZ
        END IF
33      TYPE*,'PLEASE GIVE THE COORDINATES OF THE INITIAL POINTS: '
        IF (DIM.EQ.2) THEN
           TYPE*,'POINT1(X,Y),POINT2(X,Y),POINT3(X,Y): '
           ACCEPT*,((POINT(K,J),K=1,DIM),J=1,DIM+1)
           CALL LIMIT2D(IERROR)
           IF(IERROR.NE.0) THEN
           TYPE*,'Three points are on the same line.'
           GO TO 33
           END IF
        END IF
        IF (DIM.EQ.3) THEN
           TYPE*,'POINT1(X,Y,Z),POINT2(X,Y,Z),POINT3(X,Y,Z),
     &           POINT4(X,Y,Z): '
           ACCEPT*,((POINT(K,J),K=1,DIM),J=1,DIM+1)
           CALL LIMIT3D(IERROR)
           IF(IERROR.NE.0) GO TO 33
        END IF
C*************************************************************************
C  Calculate the geometric center of the initial simplex
C*************************************************************************
        DO I=1,DIM+1
        NO(I)=I
        END DO
        CALL CENTERV(NO,CENTER,IERROR)
```

```
C*************************************************************************
C  Store the initial construct of the tessellations
C*************************************************************************
        DO I=1,DIM
        VERTEX(I,1)=CENTER(I)
        END DO
        RADIUS(1)=DISTANCE(1,1)
        NOV= 1
        NOP= DIM+1
        NODP= 0
        IFIRSTV= 1
        ICHILD(IFIRSTV)=0
        LASTV= 1
        IFIRSTA= 2
        IPARENT(IFIRSTA)= 0
        DO I=1,DIM+1
           IFP(I,1)=I
           NV(I,1)=0
        END DO
        LENLV=0
        NEWV=0
        CALL SAVE
C
44      TYPE*, 'PLEASE ENTER 1 TO ADD A POINT'
        TYPE*, '             2 TO DELETE A POINT'
        TYPE*, '             0 TO EXIT'
        ACCEPT*, ICASE
        IF (ICASE.EQ.0)  GO TO 12345
        IF ((ICASE.NE.1).AND.(ICASE.NE.2)) GO TO 44
C*************************************************************************
C  Entering the new point then check if it is in the limit
```

```
C*************************************************************************
      IF (ICASE.EQ.1) THEN
11111    TYPE*,'PLEASE ENTER THE COORDINATES OF THE NEW POINT:'
         ACCEPT*, (POINT(I,NOP+1),I=1,DIM)
         UNDERLIMT=.FALSE.
         DO I=1,DIM+1
         IF(DIM.EQ.2)
     &      CHECK= E2(I)*POINT(1,NOP+1)-E1(I)*POINT(2,NOP+1)+E3(I)
         IF(DIM.EQ.3)
     &      CHECK=A(I)*POINT(1,NOP+1)+B(I)*POINT(2,NOP+1)
     &      +C(I)*POINT(3,NOP+1)-D(I)
         IF(CHECK.EQ.0) GO TO 11
         IF((CHECK.LT.0).AND.(IPOSITION(I).EQ.1)) GO TO 11
         IF((CHECK.GT.0).AND.(IPOSITION(I).EQ.-1)) GO TO 11
         END DO
         GO TO 22
11       TYPE*,'THE NEW POINT IS NOT IN THE LIMITED AREA!'
         GO TO 11111
C*************************************************************************
C  SUBROUTINE LISTOFV ---  Step 1
C  SUBROUTINE NEWDATA ---  Step 2, 3, 4 and 5
C*************************************************************************
22       CALL LISTOFV(IERROR)
         IF (IERROR.NE.0) THEN
            TYPE*,'INVALID DATA, LISTV=0'
            GO TO 44
         END IF
         CALL NEWDATA(IERROR)
         IF (IERROE.NE.0) THEN
            TYPE*,'ERROR IN PROCESSING NEW DATA'
            GO TO 11111
```

```
          END IF
        END IF
        IF (ICASE.EQ.2) THEN
55         TYPE*, 'PLEASE ENTER THE COORDINATES OF THE POINT TO BE
DELETED'
           ACCEPT*, (DELPT(I),I=1,DIM)
C*************************************************************************
C  Find the identification number of this deleted point
C*************************************************************************
           IF (NOP.LE.DIM+1) THEN
77              TYPE*,'THERE IS NO POINT TO BE DELETED'
                GO TO 44
           END IF
C
           IFOUND=0
           ICOUNT=DIM+2
           DO WHILE ((IFOUND.EQ.0).AND.(ICOUNT.LE.NOP))
                IF (IPFLAG(ICOUNT).GE.0) THEN
                   ISAME=0
                   DO I=1,DIM
                      IF (DELPT(I).NE.POINT(I,ICOUNT)) ISAME=1
                   END DO
                   IF (ISAME.EQ.0) THEN
                      IFOUND=1
                      IDP=ICOUNT
                      NODP=NODP+1
                      IPFLAG(IDP)=-1
                   END IF
                END IF
                ICOUNT=ICOUNT+1
           END DO
```

```
            IF (IFOUND.EQ.0) GO TO 77
C***************************************************************************
C  SUBROUTINE LISTVP --- Step 1
C  SUBROUTINE NEWPART --- Step 2, 3, 4 and 5
C***************************************************************************
            CALL LISTVP(IERROR)
            IF (IERROR.NE.0) THEN
               TYPE*,'INVALID DATA, LISTV=0'
               GO TO 44
            END IF
C
            IF (LENLP.EQ.DIM+1) THEN
               NEWV=1
               DO I=1,LENLP
                 NO(I)=LISTP(I)
               END DO
               CALL CENTERV(NO,CENTER,IERROR)
               DO I=1,DIM
                 VERTEX(I,IFIRSTA)=CENTER(I)
               END DO
               DO I=1,DIM+1
                 IFP(I,IFIRSTA)=LISTP(I)
                 NV(I,IFIRSTA)=0
               END DO
               RADIUS(IFIRSTA)=DISTANCE(LISTP(1),IFIRSTA)
               GO TO 66
            END IF
C
            CALL NEWPART(IERROR)
            IF (IERROR.NE.0) THEN
               TYPE*,'ERROR IN PROCESSING NEW DATA'
```

```
               NODP=NODP-1
               IPFLAG(IDP)=0
               GO TO 55
            END IF
         END IF
C*************************************************************************
C  SUBROUTINE FINDNV --- Step 6
C  SUBROUTINE UPDATE --- Step 7
C*************************************************************************
         CALL FINDNV
66       CALL UPDATE(ICASE)
C
C
         TYPE*,'DO YOU WANT TO SEE THE RESULT?  ENTER 1  IF YES,'
         TYPE*,'                                      0  OTHERWISE.'
         ACCEPT*,N
         IF(N.NE.1) GO TO 22222
         CALL PLOT
22222    TYPE*,'IF YOU WANT TO CANCEL LAST OPERATION PLEASE ENTER 1,'
         TYPE*,'                                                  0
OTHERWISE.'
         ACCEPT*,NN
         IF(NN.EQ.1) THEN
            CALL BACKUP
            CALL PLOT
         ELSE
            CALL SAVE
         END IF
         TYPE*,'DO YOU NEED THE OUTPUT OF THE CURRENT TESSELLATIONS?'
         TYPE*,'PLEASE ENTER 1,  IF YES (0  OTHERWISE).'
         ACCEPT*,NNN
```

```
      IF (NNN.NE.1) GO TO 44
C*************************************************************************
C  OUTPUT THE TESSELLATIONS
C
      WRITE(13,10)
10    FORMAT('1','*************************************************')
      WRITE(13,1) IFIRSTV
1     FORMAT(21X,'IFIRSTV=',I4)
      WRITE(13,2) LASTV
2     FORMAT(21X,'  LASTV=',I4)
      WRITE(13,3) IFIRSTA
3     FORMAT(21X,'IFIRSTA=',I4)
      WRITE(13,*) ' '
      WRITE(13,*) '   N            FP                  NV          ICHILD
IPARENT'
      DO I=1,NOV+7
      WRITE(13,5) I,(IFP(KK,I),KK=1,4),
     &              (NV(KK,I),KK=1,4),ICHILD(I),IPARENT(I)
      END DO
5     FORMAT(2X,I3,2X,4I4,2X,4I4,1X,I6,1X,I6)
      WRITE(13,*) ' '
      WRITE(13,*)'***************** POINTS ****************************'
      WRITE(13,6) NOP-NODP
6     FORMAT(21X,'NOP=',I4)
      DO I=1,NOP
         IF (IPFLAG(I).GE.0) THEN
            WRITE(13,7) I,(POINT(N,I),N=1,3)
         END IF
      END DO
7     FORMAT(2X,'(',I4,')',3F12.3)
      WRITE(13,*) ' '
```

```
      WRITE(13,*) '********  RADIUS ************ VERTICES ************'
      WRITE(13,8) NOV
8     FORMAT(38X,'NOV=',I4)
      J=IFIRSTV
      DO I=1,NOV
      RAD=SQRT(RADIUS(J))
      WRITE(13,*) '   (',J,')',RAD,(VERTEX(N,J),N=1,3)
      J=ICHILD(J)
      END DO
9     FORMAT(2X,'(',I4,')',E40.20,2X,3F12.3)
C
C*************************************************************************
      GO TO 44
12345 STOP
      END
```

```
      INTEGER DIM
      COMMON  /CMN/DIM, NOV, NOP, IFIRSTV, LASTV, IFIRSTA, LENLV,
     &         NODP, IDP, IPFLAG(3000), LENLP, LISTP(3000),
     &         NEWV, VERTEX(3,1000), POINT(3,3000), RADIUS(1000),
     &         ICHILD(1000), IPARENT(1000), IFP(4,1000), NV(4,1000),
     &         LISTV(1000),
     &          IOLDFV, IOLDLV, IOLDFA, IOLDCH(1000), IOLDPA(1000),
     &          IOLDFP(4,1000), IOLDNV(4,1000),
     &          A(4),B(4),C(4),D(4),E1(3),E2(3),E3(3),IPOSITION(4),
     &          XMIN,XMAX,YMIN,YMAX,VX,VY,VZ,SX,SY,SZ
```

```
      SUBROUTINE BACKUP
C*************************************************************************
C  This routine retrieves the previous structure which did not
C  include the recently added data point
C*************************************************************************
      INCLUDE 'TESSEL.CMN'
      NOV=NOV+LENLV-NEWV
      NOP=NOP-1
      IFIRSTV=IOLDFV
      LASTV=IOLDLV
      IFIRSTA=IOLDFA
      DO I=1,1000
      ICHILD(I)=IOLDCH(I)
      IPARENT(I)=IOLDPA(I)
      DO J=1,4
      IFP(J,I)=IOLDFP(J,I)
      NV(J,I)=IOLDNV(J,I)
      END DO
      END DO
      RETURN
      END
```

```
      SUBROUTINE CENTERV(NO,CENT,IERROR)
C*****************************************************************
C   This routine calculates the coordinates of the geometric
C   center of the triangles(tetrahedra) defined by three(four)
C   given points in two-dimensional(three-dimensional) space
C*****************************************************************
      INCLUDE 'TESSEL.CMN'
      DIMENSION NO(DIM+1),CENT(DIM),RMATRIX(3,4)
      IERROR=0
      DO I=1,DIM
      DO J=1,DIM
      RMATRIX(J,I)=POINT(I,NO(J+1))-POINT(I,NO(1))
      END DO
      END DO
      SUM1=0.
      DO I=1,DIM
      SUM1=SUM1+POINT(I,NO(1))**2.
      END DO
      DO I=1,DIM
      SUM2=0.
      DO J=1,DIM
      SUM2=SUM2+POINT(J,NO(I+1))**2.
      END DO
      RMATRIX(I,DIM+1)=(SUM2-SUM1)/2.
      END DO
      CALL SLS(RMATRIX,DIM,CENT,IERROR)
      RETURN
      END
```

```
      FUNCTION DISTANCE(NO1,NO2)
C*****************************************************************
C   This function calculates the distance between point NO1 and
C   vertex NO2
C Note:
C   Since this distance is used only for comparison, it is the
C   square of the real Euclidean distance between the two given
C   points.
C*****************************************************************
      INCLUDE 'TESSEL.CMN'
      DISTANCE= 0.
      DO 10 I=1,DIM
      DISTANCE= DISTANCE+(POINT(I,NO1)-VERTEX(I,NO2))**2
10    CONTINUE
      RETURN
      END
```

```
SUBROUTINE FINDNV
LOGICAL NOTNV, FLAGIN
INCLUDE 'TESSEL.CMN'
ICURRENT=IFIRSTA
IC=1
DO WHILE (IC.LE.NEWV)
   DO I=1,DIM+1
      IF (NV(I,ICURRENT).LT.0) THEN
        IDONE=0
        ICOUNT=1
        ICOMP=IFIRSTA
        DO WHILE ((IDONE.EQ.0).AND.(ICOUNT.LT.NEWV))
           IF (ICOMP.NE.ICURRENT) THEN
              NOTNV=.FALSE.
              DO K=1,DIM+1
               FLAGIN=.FALSE.
               DO L=1,DIM+1
               IF(IFP(K,ICURRENT).EQ.IFP(L,ICOMP))FLAGIN=.TRUE.
               END DO
               IF (K.EQ.I) THEN
                  IF (FLAGIN) NOTNV=.TRUE.
               ELSE
                  IF (.NOT.FLAGIN) NOTNV=.TRUE.
               END IF
              END DO
              IF (.NOT.NOTNV) THEN
               NV(I,ICURRENT)=ICOMP
               IDONE=1
              ELSE
               ICOUNT=ICOUNT+1
              END IF
```

```
            END IF
            ICOMP=ICHILD(ICOMP)
          END DO
        END IF
      END DO
      ICURRENT=ICHILD(ICURRENT)
      IC=IC+1
END DO
RETURN
END
```

```
      FUNCTION INLIST(ITEM)
C*************************************************************************
C   This function checks whether ITEM is in the list of the
C   deleted vertices
C*************************************************************************
      LOGICAL INLIST
      INCLUDE 'TESSEL.CMN'
      INLIST= .FALSE.
      IF (LENLV.EQ.0) GO TO 22
      DO  I=1, LENLV
      IF (ITEM.EQ.LISTV(I)) THEN
         INLIST= .TRUE.
         GO TO 22
      END IF
      END DO
22    RETURN
      END
```

```
      FUNCTION INLISTP(ITEM)
C*************************************************************************
C  This function checks whether ITEM is in LISTP
C*************************************************************************
      LOGICAL INLISTP
      INCLUDE 'TESSEL.CMN'
      INLISTP=.FALSE.
      IF (LENLP.EQ.0) GO TO 10
      DO I=1,LENLP
         IF (ITEM.EQ.LISTP(I)) THEN
              INLISTP=.TRUE.
              GO TO 10
         END IF
      END DO
10    RETURN
      END
```

```
      SUBROUTINE LIMIT2D(IERROR)
C*****************************************************************
C   This routine calculates the three line equations defined by
C   the initial points
C Note:
C   The form  aX + bY + c = 0  has the advantage over the
C   traditional form  Y = mX + e  having no singularities.
C*****************************************************************
C   mX - Y + e = 0
C   m = (Y2 - Y1)/(X2 - X1)
C   e = Y1 - mX1
C   e1= X2 - X1
C   e2= Y2 - Y1
C   e2*X - e1*Y + e1*Y1 - e2*X1 = 0
C   e2*x - e1*y + e3 = 0
C*****************************************************************
      INCLUDE 'tessel.CMN'
      INTEGER N(3)
      IERROR=0
      DO I=1,3
      IC=1
      DO 1 II=1,3
      IF(II.EQ.I) GO TO 1
      N(IC)=II
      IC=IC+1
1     CONTINUE
      N(3)=I
      E1(I) = POINT(1,N(2)) - POINT(1,N(1))
      E2(I) = POINT(2,N(2)) - POINT(2,N(1))
      E3(I) = E1(I)*POINT(2,N(1)) - E2(I)*POINT(1,N(1))
      CHECK= E2(I)*POINT(1,N(3)) - E1(I)*POINT(2,N(3)) + E3(I)
```

```
      IF(CHECK.EQ.0) THEN
         IERROR=1
         GO TO 2
      ELSE IF(CHECK.LT.0) THEN
         IPOSITION(I)=-1
      ELSE
         IPOSITION(I)=1
      END IF
      END DO
2     RETURN
      END
```

```
      SUBROUTINE LIMIT3D(IERROR)
C*****************************************************************
C   This routine calculates the four plane equations defined by
C   the initial points
C*****************************************************************
      INCLUDE 'TESSEL.CMN'
      INTEGER N(4)
      IERROR=0
      DO I=1,4
      IC=1
      DO 1 II=1,4
      IF(II.EQ.I) GO TO 1
      N(IC)=II
      IC=IC+1
1     CONTINUE
      N(4)=I
      CALL PLANE(N(1),N(2),N(3),A(I),B(I),C(I),D(I),IERROR)
      IF(IERROR.EQ.1) GO TO 2
      CHECK= A(I)*POINT(1,N(4))+B(I)*POINT(2,N(4))+
     &          C(I)*POINT(3,N(4))-D(I)
      IF(CHECK.EQ.0) THEN
         IERROR=1
         GO TO 2
      ELSE IF(CHECK.LT.0) THEN
         IPOSITION(I)= -1
      ELSE
         IPOSITION(I)= 1
      END IF
      END DO
2     RETURN
      END
```

```
      SUBROUTINE LISTOFV(IERROR)
C*********************************************************************
C    This routine gets the list of all the vertices deleted by
C    the new point
C*********************************************************************
      INCLUDE 'TESSEL.CMN'
      IERROR=0
      LENLV = 0
      ICURRENT= IFIRSTV
      DO WHILE (ICURRENT.NE.0)
      IF (DISTANCE(NOP+1,ICURRENT).LT.RADIUS(ICURRENT)) THEN
         LENLV= LENLV+1
         LISTV(LENLV)= ICURRENT
      END IF
      ICURRENT= ICHILD(ICURRENT)
      END DO
      IF (LENLV.EQ.0) IERROR=1
      RETURN
      END
```

```
      SUBROUTINE LISTVP(IERROR)
      INCLUDE 'TESSEL.CMN'
      IERROR=0
      LENLV=0
      LENLP=0
      ICURRENT=IFIRSTV
      DO WHILE (ICURRENT.NE.0)
         IFLAG=0
         DO I=1,DIM+1
            IF (IFP(I,ICURRENT).EQ.IDP) THEN
              IFLAG=I
              LENLV=LENLV+1
              LISTV(LENLV)=ICURRENT
              GO TO 10
            END IF
         END DO
10       IF (IFLAG.NE.0) THEN
            DO J=1,DIM+1
              IF((J.NE.IFLAG).AND.(.NOT.INLISTP(IFP(J,ICURRENT)))) THEN
                 LENLP=LENLP+1
                 LISTP(LENLP)=IFP(J,ICURRENT)
              END IF
            END DO
         END IF
         ICURRENT=ICHILD(ICURRENT)
      END DO
      IF (LENLV.EQ.0) IERROR=1
      RETURN
      END
```

```
      SUBROUTINE NEWDATA(IERROR)
C*************************************************************************
C   This routine calculates the newly generated vertices and then finds
C   their forming points and neighbouring vertices
C*************************************************************************
      DIMENSION NPT(4),RMT(3,4),TPT(2),TPT3(3)
      LOGICAL FLAGIN
      INCLUDE 'TESSEL.CMN'
      IERROR=0
      NEWV= 0
      ICURRENT= IFIRSTA
      DO 50 I=1,LENLV
      DO 60 J=1,DIM+1
      IF (.NOT.INLIST(NV(J,LISTV(I)))) THEN
         NEWV= NEWV+1
         IF (ICURRENT.EQ.0) THEN
         TYPE*,'(SUBROUTINE NEWDATAS)INSUFFICIENT STORAGE SPACE!'
         STOP
         END IF
C*************************************************************************
C   Find the forming points of the new vertex
C*************************************************************************
         NPT(1)=NOP+1
         IFP(1,ICURRENT)=NPT(1)
         IC=2
         DO 11 II=1,DIM+1
         IF(II.EQ.J) GO TO 11
         NPT(IC)=IFP(II,LISTV(I))
         IFP(IC,ICURRENT)=NPT(IC)
         IC=IC+1
11       CONTINUE
```

```
C*************************************************************************
C    Calculate the coordinates of the new vertex
C*************************************************************************
            CALL CENTERV(NPT,TPT3,IERROR)
            IF(IERROR.NE.0) GO TO 140
            DO II=1,DIM
            VERTEX(II,ICURRENT)=TPT3(II)
            END DO
            RADIUS(ICURRENT)=DISTANCE(IFP(1,ICURRENT),ICURRENT)
C*************************************************************************
C    Update the relative neighbouring vertices
C*************************************************************************
            DO II=1,DIM+1
               NV(II,ICURRENT)=-1
            END DO
            NV(1,ICURRENT)=NV(J,LISTV(I))
            IF(NV(J,LISTV(I)).EQ.0) GO TO 130
            DO  K=1,DIM+1
         IF(NV(K,NV(J,LISTV(I))).EQ.LISTV(I))NV(K,NV(J,LISTV(I)))=ICURRENT
            END DO
130         ICURRENT= ICHILD(ICURRENT)
         END IF
60       CONTINUE
50       CONTINUE
140      RETURN
         END
```

```
      SUBROUTINE NEWPART(IERROR)
C*****************************************************************************
C    This routine calculates the newly generated vertices and then finds
C    their forming points and neighbouring vertices
C*****************************************************************************
      DIMENSION NPT(4),RMT(3,4),TPT(2),TPT3(3),NO(4)
      LOGICAL FLAGIN
      INCLUDE 'TESSEL.CMN'
C
      IERROR=0
      ICURRENT= IFIRSTA
      NEWV= 0
      DO 100 I=1,LENLV
         IF (LISTV(I).LT.0) GO TO 100
         NEWV= NEWV+1
C-----------------------------------------------------------------------
--
C  Clear the marks of points in LISTP
C-----------------------------------------------------------------------
--
         DO IJK=1,LENLP
            IF (LISTP(IJK).LT.0) LISTP(IJK)=LISTP(IJK) * -1
         END DO
C-----------------------------------------------------------------------
C  Find the no. and position of the neighbouring vertex not in LISTV
C-----------------------------------------------------------------------
         DO J=1,DIM+1
            IF (IFP(J,LISTV(I)).EQ.IDP) THEN
              IPOSI=J
              ISAVE1=NV(J,LISTV(I))
              GO TO 99
```

```
            END IF
          END DO
          TYPE*,' INVALID DATA IN LIST OF VERTICES'
          IERROR=1
          RETURN
C*************************************************************************
C   Find the forming points of the new vertex
C*************************************************************************
99        DO K=1,DIM+1
            IFP(K,ICURRENT)=0
          END DO
          IC=1
          DO IJ=1,DIM+1
            IF (IJ.NE.IPOSI) THEN
              IFP(IC,ICURRENT)=IFP(IJ,LISTV(I))
              IC=IC+1
            END IF
          END DO
C
          DO 88 IJ=1,DIM
            DO IJK=1,LENLP
              IF (IFP(IJ,ICURRENT).EQ.LISTP(IJK)) THEN
                LISTP(IJK)=LISTP(IJK) * -1
                GO TO 88
              END IF
            END DO
88        CONTINUE
C-------------------------------------------------------------------------
C  Set the last N.V. of this new vertex
C-------------------------------------------------------------------------
          DO IJ=1,DIM
```

```
            NV(IJ,ICURRENT)=-1
         END DO
         NV(DIM+1,ICURRENT)=ISAVE1
C-----------------------------------------------------------------------
C  (Step 3) Find the last forming point
C-----------------------------------------------------------------------
         NCOUNT=0
         DO 90 J=1,DIM+1
            IF (J.EQ.IPOSI) GO TO 90
            DO 80 IJ=1,DIM+1
              DO IJK=1,DIM
                 IF (IFP(IJ,NV(J,LISTV(I))).EQ.IDP) GO TO 80
                 IF (IFP(IJ,NV(J,LISTV(I))).EQ.IFP(IJK,ICURRENT)) GO TO
80
              END DO
C
              DO IJK=1,LENLP
                 IF (IFP(IJ,NV(J,LISTV(I))).EQ.LISTP(IJK)) THEN
                 LISTP(IJK)=LISTP(IJK) * -1
                 GO TO 77
                 END IF
              END DO
77            CONTINUE
C
              IF (NCOUNT.NE.0) THEN
                 DIS=DISTANCE(IFP(IJ,NV(J,LISTV(I))),ICURRENT)
                 IF (DIS.GE.RADIUS(ICURRENT)) GO TO 80
              END IF
              IFP(DIM+1,ICURRENT)=IFP(IJ,NV(J,LISTV(I)))
              IRELAT=NV(J,LISTV(I))
              GO TO 70
```

```
80            CONTINUE
70         CONTINUE
C*************************************************************************
C   Calculate the coordinates of the center vertex and radius
C*************************************************************************
           DO IJ=1,DIM+1
              NPT(IJ)=IFP(IJ,ICURRENT)
           END DO
           CALL CENTERV(NPT,TPT3,IERROR)
           IF(IERROR.NE.0) GO TO 22
           DO IJ=1,DIM
           VERTEX(IJ,ICURRENT)=TPT3(IJ)
           END DO
           RADIUS(ICURRENT)=DISTANCE(IFP(1,ICURRENT),ICURRENT)
           NCOUNT=NCOUNT+1
90      CONTINUE
C-------------------------------------------------------------------------
C  Mark this vertex to be a used vertex
C-------------------------------------------------------------------------
       LISTV(I)= LISTV(I) * -1
C-------------------------------------------------------------------------
C  (Step 4) Check if the last forming point is correct
C-------------------------------------------------------------------------
       IFLAG=0
       DO J=1,LENLP
          IF (LISTP(J).GT.0) THEN
             DIS=DISTANCE(LISTP(J),ICURRENT)
             IF (DIS.LT.RADIUS(ICURRENT)) THEN
               IFP(DIM+1,ICURRENT)=LISTP(J)
               DO IJ=1,DIM+1
                  NPT(IJ)=IFP(IJ,ICURRENT)
```

```
                END DO
                CALL CENTERV(NPT,TPT3,IERROR)
                IF (IERROR.NE.0) GO TO 22
                DO IJ=1,DIM
                   VERTEX(IJ,ICURRENT)=TPT3(IJ)
                END DO
                RADIUS(ICURRENT)=DISTANCE(IFP(1,ICURRENT),ICURRENT)
                IFLAG=J
             END IF
          END IF
       END DO
C-------------------------------------------------------------------------
C  Mark the vertex in LISTV if IFLAG=0
C-------------------------------------------------------------------------
       IF (IFLAG.EQ.0) THEN
          DO J=1,LENLV
             IF (LISTV(J).EQ.IRELAT) THEN
               LISTV(J)=LISTV(J) * -1
               GO TO 66
             END IF
          END DO
66        CONTINUE
C-------------------------------------------------------------------------
C  Get the neighbouring vertex of the new vertex
C-------------------------------------------------------------------------
          DO J=1,DIM+1
             IF (IFP(J,IRELAT).EQ.IDP) THEN
               ISAVE2=NV(J,IRELAT)
               GO TO 55
             END IF
          END DO
```

```
55          CONTINUE
            DO 44 J=1,DIM+1
               DO K=1,DIM+1
                 IF (IFP(J,ICURRENT).EQ.IFP(K,IRELAT)) GO TO 44
               END DO
               NV(J,ICURRENT)=ISAVE2
               GO TO 33
44          CONTINUE
C-------------------------------------------------------------------------
C   (Step 5) Update the related neighbouring vertices
C-------------------------------------------------------------------------
33          IF (ISAVE2.NE.0) THEN
               J=1
               IDONE=0
               DO WHILE ((IDONE.EQ.0).AND.(J.LE.DIM+1))
                 IF (NV(J,ISAVE2).EQ.IRELAT) THEN
                    NV(J,ISAVE2)=ICURRENT
                    IDONE=1
                 END IF
                 J=J+1
               END DO
            END IF
         END IF
C
         IF (ISAVE1.NE.0) THEN
            J=1
            IDONE=0
            DO WHILE ((IDONE.EQ.0).AND.(J.LE.DIM+1))
                 IF (NV(J,ISAVE1).EQ.LISTV(I)) THEN
                    NV(J,ISAVE1)=ICURRENT
                    IDONE=1
```

```
                END IF
                J=J+1
            END DO
        END IF
        ICURRENT=ICHILD(ICURRENT)
100     CONTINUE
22      RETURN
        END
```

```
      SUBROUTINE PLANE(N1,N2,N3,R1,R2,R3,R4,IERROR)
C*************************************************************
C    This routine finds the equation of the plane defined by
C    three given points N1, N2 and N3
C*************************************************************
      DIMENSION RMT(3,4),COE(3)
      INCLUDE 'TESSEL.CMN'
      IERROR=0
      DO I=1,3
      RMT(1,I)= POINT(I,N1)
      RMT(2,I)= POINT(I,N2)
      RMT(3,I)= POINT(I,N3)
      RMT(I,4)= 1.
      END DO
      CALL SLS(RMT,3,COE,IERROR)
      IF(IERROR.EQ.1) THEN
      TYPE*,'Three points are on the same line (SUBROUTINE PLANE)'
      GO TO 1
      END IF
      R1=COE(1)
      R2=COE(2)
      R3=COE(3)
      R4=1.
1     RETURN
      END
```

```
      SUBROUTINE PLOT
C*************************************************************************
C    This routine draws the tessellations and triangulations
C    using GCS(Graphics Compatibility System) plotting routines
C*************************************************************************
      DIMENSION PT1(3), PT2(3), NPT(3), RMT(3,4)
      INCLUDE 'TESSEL.CMN'
      CALL DISKON(17)
      CALL USTART
      CALL UWINDO(XMIN,XMAX,YMIN,YMAX)
      IF(DIM.EQ.3) CALL UVIEW(VX,XY,VZ,SX,SY,SZ)
C*************************************************************************
C    The Voronoi tessellation part
C    Connecting the vertices by solid lines
C*************************************************************************
      CALL USET('LINE')
      CALL USET('RED')
      I=IFIRSTV
      DO 6 KK=1,NOV
         NBO=0
         DO IJK=1,DIM+1
            IF ((IFP(IJK,I).GE.1).AND.(IFP(IJK,I).LE.DIM+1)) NBO=1
         END DO
         IF (NBO.EQ.0) THEN
            DO  J=1,DIM
              PT1(J)= VERTEX(J,I)
            END DO
            DO 3 K=1,DIM+1
              IF(DIM.EQ.2) CALL UMOVE(PT1(1),PT1(2))
              IF(DIM.EQ.3) CALL U3MOVE(PT1(1),PT1(2),PT1(3))
              IF(NV(K,I).EQ.0) GO TO 3
```

```
                IF(NV(K,I).NE.0) THEN
                  DO  L=1,DIM
                     PT2(L)=VERTEX(L,NV(K,I))
                  END DO
                  IF(DIM.EQ.2) CALL UPEN(PT2(1),PT2(2))
                  IF(DIM.EQ.3) CALL U3PEN(PT2(1),PT2(2),PT2(3))
                END IF
3             CONTINUE
            END IF
            I=ICHILD(I)
6       CONTINUE
C*************************************************************************
C    The Delaunay triangulation part
C    Connecting the points by dash lines
C*************************************************************************
        CALL USET('DASH')
        CALL USET('BLUE')
        I=IFIRSTV
        DO 11 II=1,NOV
        DO 22 J=1,DIM
        IF((1.LE.IFP(J,I)).AND.(IFP(J,I).LE.DIM+1)) GO TO 22
        DO 33 K=1,DIM
           PT1(K)= POINT(K,IFP(J,I))
33      CONTINUE
        DO 44 L=J+1, DIM+1
           IF(DIM.EQ.2) CALL UMOVE(PT1(1),PT1(2))
           IF(DIM.EQ.3) CALL U3MOVE(PT1(1),PT1(2),PT1(3))
        IF((1.LE.IFP(L,I)).AND.(IFP(L,I).LE.DIM+1)) GO TO 44
        DO 55 N=1,DIM
           PT2(N)= POINT(N,IFP(L,I))
55      CONTINUE
```

```
            IF(DIM.EQ.2) CALL UPEN(PT2(1),PT2(2))
            IF(DIM.EQ.3) CALL U3PEN(PT2(1),PT2(2),PT2(3))
44      CONTINUE
22      CONTINUE
        I=ICHILD(I)
11      CONTINUE
        CALL UEND
        RETURN
        END
```

```
      SUBROUTINE SAVE
C*****************************************************************
C    This routine saves the currently generated structure
C*****************************************************************
      INCLUDE 'TESSEL.CMN'
      IOLDFV=IFIRSTV
      IOLDLV=LASTV
      IOLDFA=IFIRSTA
      DO I=1,1000
      IOLDCH(I)=ICHILD(I)
      IOLDPA(I)=IPARENT(I)
      DO J=1,4
      IOLDFP(J,I)=IFP(J,I)
      IOLDNV(J,I)=NV(J,I)
      END DO
      END DO
      RETURN
      END
```

```
      SUBROUTINE SLS(RMATRIX, ISIZE, SOLUTION, IERROR)
C*****************************************************************
C   This routine solves an (ISIZE x ISIZE) linear system
C   Note:
C        ISIZE is less than or equal to 3 due to the dimension
C        statement in this routine
C   Reference:  "Numerical Analysis"
C               by R. L. Burden, J. D. Faires, & A. C. Reynolds
C               p.268  Gaussian Elimination and Backward Substitution
C*****************************************************************
      DIMENSION RMATRIX(3,4), SOLUTION(3)
      INCLUDE 'TESSEL.CMN'
      IERROR=0
      DO 80 I=1,ISIZE-1
      DO 20 IP=I,ISIZE
      IF(RMATRIX(IP,I).NE.0) GO TO 30
20    CONTINUE
      TYPE*,'MATRIX TERM',IP,I,'=0'
      TYPE*,'NO UNIQUE SOLUTION EXISTS.   (SUBROUTINE SLS)1'
      CALL BACKUP
      IERROR=1
      GO TO 100
30    IF(IP.NE.I) THEN
      DO 40 K=1,ISIZE+1
      SAVE=RMATRIX(IP,K)
      RMATRIX(IP,K)=RMATRIX(I,K)
      RMATRIX(I,K)=SAVE
40    CONTINUE
      END IF
      DO 70 J=I+1,ISIZE
      F=RMATRIX(J,I)/RMATRIX(I,I)
```

```
      DO 60 K=1,ISIZE+1
      RMATRIX(J,K)=RMATRIX(J,K)-F*RMATRIX(I,K)
60    CONTINUE
70    CONTINUE
80    CONTINUE
      IF(RMATRIX(ISIZE,ISIZE).EQ.0) THEN
      TYPE*,'NO UNIQUE SOLUTION EXISTS.  (SUBROUTINE SLS)2'
      CALL BACKUP
      IERROR=1
      GO TO 100
      END IF
      SOLUTION(ISIZE)=RMATRIX(ISIZE,ISIZE+1)/RMATRIX(ISIZE,ISIZE)
      DO 90 I=ISIZE-1,1,-1
      SUM=0
      DO 10 J=I+1,ISIZE
      SUM=SUM+RMATRIX(I,J)*SOLUTION(J)
10    CONTINUE
      SOLUTION(I)=(RMATRIX(I,ISIZE+1)-SUM)/RMATRIX(I,I)
90    CONTINUE
100   RETURN
      END
```

```
      SUBROUTINE UPDATE(ICASE)
C*************************************************************************
C    This routine updates the double linked lists of all the
C    vertices and available spaces
C*************************************************************************
      INCLUDE 'TESSEL.CMN'
C
      DO I=1,LENLV
        IF (LISTV(I).LT.0) LISTV(I)=LISTV(I)*-1
      END DO
C*************************************************************************
C    Link the new generated vertices to the list
C*************************************************************************
      ICHILD(LASTV)= IFIRSTA
      IPARENT(IFIRSTA)= LASTV
      DO I=1,NEWV
         LASTV= ICHILD(LASTV)
      END DO
      IFIRSTA=ICHILD(LASTV)
      IPARENT(IFIRSTA)=0
      ICHILD(LASTV)= 0
C*************************************************************************
C    Delete the vertices from the list
C*************************************************************************
      DO J=1,LENLV
         IF(LISTV(J).EQ.IFIRSTV) THEN
            IFIRSTV=ICHILD(LISTV(J))
            IPARENT(IFIRSTV)= 0
         ELSE IF(LISTV(J).EQ.LASTV) THEN
              LASTV=IPARENT(LASTV)
              ICHILD(LASTV)= 0
```

```
      ELSE
        ICHILD(IPARENT(LISTV(J)))= ICHILD(LISTV(J))
        IPARENT(ICHILD(LISTV(J)))= IPARENT(LISTV(J))
      END IF
        ISAVE=IFIRSTA
        IPARENT(IFIRSTA)=LISTV(J)
        IFIRSTA=LISTV(J)
        ICHILD(IFIRSTA)=ISAVE
        IPARENT(IFIRSTA)=0
END DO
IF (ICASE.EQ.1) NOP=NOP+1
NOV=NOV+NEWV-LENLV
RETURN
```

VITA

Judy Chien-Hui Chang was born on July 21, 1956 in Taichung, Taiwan, Republic of China. She attended Hsiao Ming Girls' Middle School in that city and was graduated from Provincial Taichung Girls' High School in 1974.

Ms. Chang received the Bachelor of Science degree in Mathematics from the Fu Jen University, Taipei, Taiwan, in 1978. That same year she entered the University of New Brunswick, Fredericton, N. B., Canada, and in 1980 was awarded the Master of Arts degree in Mathematics.

The author entered The University of Tennessee Space Institute in 1980 and began study toward a Master of Science degree in Computer Science. This degree was awarded in June 1983.