A fast algorithm for constructing Delaunay triangulations in the plane

S. W. SLOAN

Department of Civil Engineering and Surveying, The University of Newcastle, NSW 2308, Australia

This paper describes an algorithm for computing Delaunay triangulations of arbitrary collections of points in the plane. A FORTRAN 77 implementation of the scheme is given. For N points distributed randomly within a square domain, the expected run time for the algorithm is approximately $0(N^{5/4})$. Empirical tests, for N up to 10 000, indicate that the actual run time is substantially less than this prediction and is generally better than $0(N^{1.1})$. Excluding the memory required to store the co-ordinates, the algorithm requires slightly greater than $14N$ words of integer memory to complete a typical triangulation. The efficiency of the proposed algorithm is verified by comparing its performance with other Delaunay triangulation procedures. Uses of the algorithm include the generation of finite element meshes and the construction of contour plots.

Key Words: Delaunay, triangulation, algorithm.

INTRODUCTION

The problem of triangulating arbitrary collections of points in the plane occurs frequently in engineering and examples include mesh generation for finite element analysis and the construction of contour plots.

The theory of Delaunay triangulations has been described previously[1] but will be discussed briefly for completeness. To describe the construction of a Delaunay triangulation it is convenient to consider the corresponding Dirichlet tesselation. The Dirichlet tessellation for five points in the plane is shown in Fig. 1 and is denoted by the heavy lines. This tessellation divides the plane into a collection of polygonal regions whose boundaries are the perpendicular bisectors of the lines joining the neighbouring data points. Each polygon is associated with a single data point. Any location within a given polygon is closer to the polygon data point than any other data point. The Delaunay triangulation that corresponds to the Dirichlet tessellation is constructed by connecting all data points that share a polygon boundary. The Delaunay triangulation for the five points in Fig. 1 is indicated by the faint lines.

In general, the vertices of the Dirichlet tessellation occur where three adjacent polygons meet. The three data points associated with each of these polygons form a Delaunay triangle. By definition, each vertex of a Dirichlet tessellation a equidistant from each of the three data points forming the Delaunay triangle. Thus, each vertex of the Dirichlet tessellation is uniquely associated with a Delaunay triangle and is located at its circumcentre. When the Delaunay triangulation is complete, this means that no data point may lie inside the circumcircle of any triangle.

Generally speaking, the Delaunay triangulation associated with an arbitrary set of points in the plane is unique.[2,3] In some instances, however, the triangulation may not be unique and is said to be degenerate. A very simple example which illustrates a degenerate triangulation is shown in Fig. 2, where four data points are located at the vertices of a square. The single vertex of the Dirichlet tessellation is located at the centroid of the square where four polygons meet. Two different Delaunay triangulations are possible with this configuration and both are equally valid. In practical algorithms, the problem of degeneracy is easily dealt with by making an arbitrary choice between alternative triangulations and does not pose any serious difficulties.

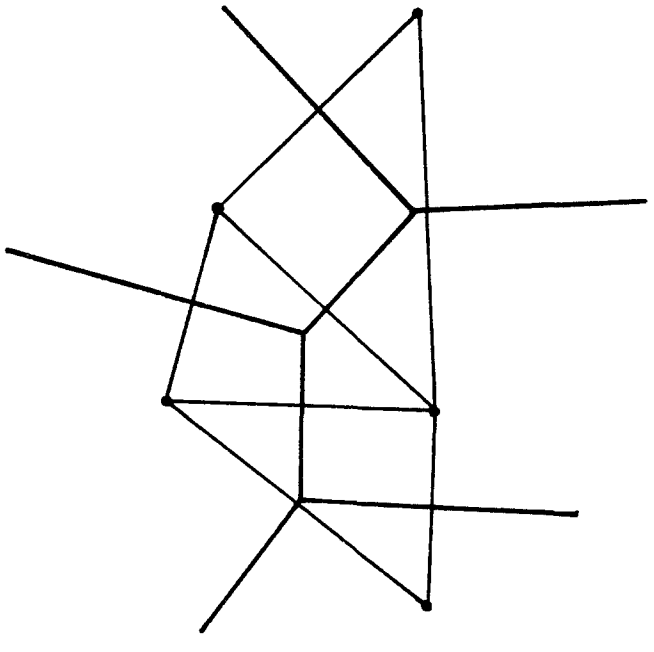


Figure 1. The Dirichlet tessellation and del-triangulation

Accepted April 1986. Discussion closes March 1987.

0141-1195/87/010034-22 $2.00

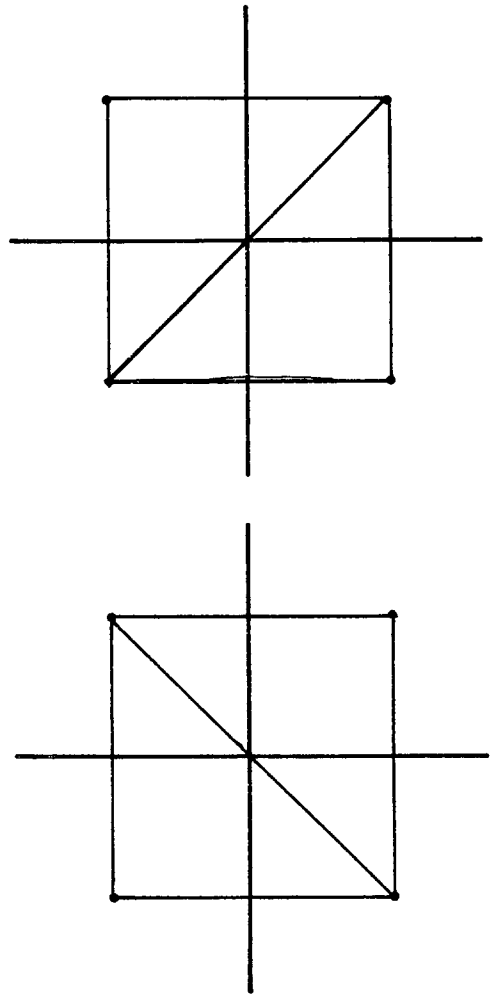


Figure 2. Degenerate Delaunay triangulations

One of the advantages of Delaunay triangulations, as opposed to triangulations constructed heuristically, is that they automatically avoid forming triangles with small included angles whenever this is possible. Indeed Lawson[2] and Sibson[3] have shown that Delaunay triangulations are, by definition, locally equiangular. This means that for every convex quadrilateral formed by two adjacent triangles, the minimum of the six angles in the two triangles is greater than it would have been if the alternative diagonal had been drawn and the other pair of triangles chosen. Because of this property, Delaunay triangulations are particularly suited to grid generation for finite element analysis and contouring algorithms.

A number of algorithms for constructing planar Delaunay triangulations have been proposed.[1,2,4–7] For a collection of N points, the average and worst case run times for the various algorithms are shown in Table 1. In engineering applications the average performance of a triangulation algorithm is generally more important than its worst case performance, since the latter tends to occur rarely in practice.[4] Average run times for triangulation algorithms are usually deduced by considering collections of points located randomly within square or circular domains.

For large sets of points, the results in Table 1 indicate that the first algorithm of Lee and Schachter[4] (these authors propose two algorithms) is the most efficient. This procedure, however, is complicated and difficult to implement. FORTRAN code for the Cline and Renka[7] algorithm has been made publicly available by Renka[8] and a simple FORTRAN 77 implementation of the Watson[6] scheme is described in Sloan and Houlsby.[9] Watson's algorithm, which is quite efficient for triangulating up to about 2000 points, has the advantage of being particularly simple. This paper describes a simple scheme which may be used to compute Delaunay triangulations for both small and large sets of points. Analysis of the algorithm indicates that its run time is $0(N^{5/4})$ for points that are distributed randomly within a square domain. Empirical comparisons with other procedures suggest that it is efficient.

OUTLINE OF ALGORITHM

The algorithm combines features of both the Watson[6] and Lawson[2] procedures. The Delaunay triangulation is assembled by introducing each point, one at a time, into an existing Delaunay triangulation which is then updated.

Following the idea of Watson, the process is started by selecting three points to form a 'supertriangle' which completely encompasses of all of the data points to be triangulated. Initially the Delaunay triangulation is thus comprised of a single triangle defined by the supertriangle vertices. When a new point P is introduced into the triangulation, we first find an existing triangle which encloses P and form three new triangles by connecting P to each of its vertices. Note that during this step the original enclosing triangle is deleted and the net gain in the total number of triangles is two. After the new point P has been inserted, the existing triangulation is updated to a Delaunay triangulation using the swapping algorithm of Lawson.[2] In this procedure all the triangles which are adjacent to the edges opposite P are placed on a last-in, first-out stack (ie. a maximum of three triangles are placed on the stack initially). Each triangle is then unstacked, one at a time, and a check is made to determine if P lies inside its circumcircle. If this is the case then the triangle containing P as a vertex and the adjacent triangle form a convex quadrilateral with the diagonal drawn in the wrong direction, and it must be replaced by the alternative diagonal to preserve the structure of the Delaunay triangulation. The swapping procedure replaces two old triangles with two new triangles with no net gain in the total number of triangles. Once the swap is completed, any triangles which are now opposite P are added to the stack (there are a maximum of two). The next triangle is then unstacked and the whole process is repeated until the stack is empty and this results in a new Delaunay triangulation containing the point P. An illustration of the swapping procedure is shown in Fig. 3. Note that if P lies outside (or on) the circumcircle for a stacked triangle, then no action is taken and we

Table 1. Average and worst case running times for various Delaunay triangulation algorithms

Algorithm	Average case	Worst case
Green and Sibson	$0(N^{3/2})$	$0(N^2)$
Lawson	$0(N^{4/3})$	$0(N^2)$
Lee and Schachter (1)	$0(N \log_2 N)$	$0(N \log_2 N)$
Lee and Schachter (2)	$0(N^{3/2})$	$0(N^2)$
Bowyer	$0(N^{3/2})$	$0(N^2)$
Watson	$0(N^{3/2})$	$0(N^2)$
Cline and Renka	$0(N^{4/3})$	$0(N^2)$

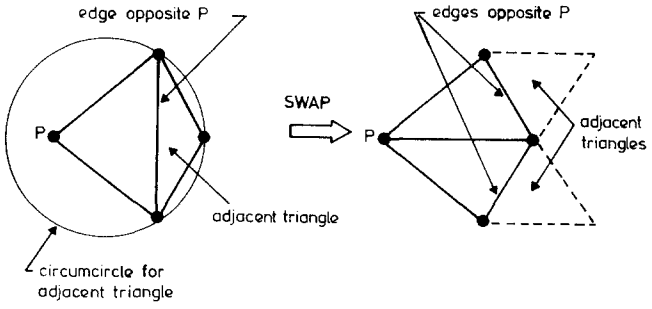


Figure 3. Lawson's swapping algorithm

simply skip to the next triangle on the stack. It has been shown by Lawson[2] that this iterative algorithm must result in a Delaunay triangulation and will always terminate after a finite number of swaps. Typically only a few levels of swaps are necessary for each edge which is initially opposite P and the process is thus efficient.

After all the points have been added to the triangulation, the final Delaunay triangulation is obtained by removing all of the triangles that contain one or more of the supertriangle vertices. Any vertex which appears in these deleted triangles, but is not a supertriangle vertex, must lie on the boundary of the triangulation. Since the insertion of each new point into the triangulation creates two new triangles the final number of triangles, including those formed with the vertices of the supertriangle, is $2N+1$.

IMPLEMENTATION OF ALGORITHM

An implementation of the algorithm for computing Delaunay triangulations is given in Appendix 1. To the best of the author's knowledge the code strictly obeys the syntax of FORTRAN 77 and thus should be portable. The program uses single precision arithmetic which is considered to be satisfactory for computation on 32 bit machines. To convert the implementation to double arithmetic, all REAL declarations should be replaced by DOUBLE PRECISION declarations and all real constants in PARAMETER statements should be replaced by double precision constants. The program is comprised of five subroutines (DELTRI, BSORT, QSORTI, DELAUN, PUSH) and four short function subprograms (TRILOC, POP, EDG, SWAP). Each of these will be discussed in turn to illustrate the detail of the overall algorithm.

Subroutine DELTRI

This is the only subroutine that needs to be called by the user to construct the Delaunay triangulation and controls the overall flow of the program. When calling DELTRI, NUMPTS is the total number of points in the data set and N is the number of points to be triangulated. The set of points to be triangulated is stored in the integer vector LIST prior to calling the subroutine. LIST is of length N where $N \leqslant$ NUMPTS. This allows the user to triangulate any subset of the total number of points and is particularly useful in practical applications. The co-ordinates of the points in the data set are stored in the real vectors X and Y. Each of these is of length NUMPTS + 3. The integer vector BIN is of length NUMPTS, and is required as auxiliary storage in subroutines BSORT and DELAUN. Throughout the program the structure of the Delaunay triangulation is stored in the integer arrays V and E. Both of these arrays are two-dimensional, with V containing the vertices for each triangle and E containing the adjacent triangles. The conventions for this data structure are shown for a simple example in Fig. 4. The dimensions of these arrays are $V(3, 2*N+1)$ and $E(3, 2*N+1)$.

At the beginning of subroutine DELTRI, the co-ordinates of the points to be triangulated are normalised to the values $(\hat{x}, \hat{y})$ according to

$$\hat{x}_P = (x_P - \text{XMIN})/\text{DMAX}$$

$$\hat{y}_P = (y_P - \text{YMIN})/\text{DMAX}$$

where

$$\text{DMAX} = \text{MAX}(\text{XMAX} - \text{XMIN}, \text{YMAX} - \text{YMIN})$$

and

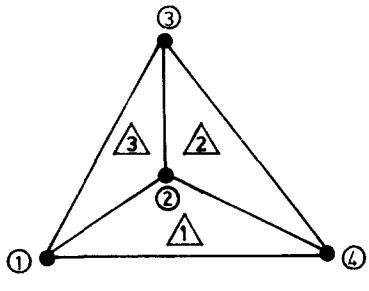
$$\begin{aligned} \text{XMIN} &= \text{MIN}\ \{x_P\} \\ \text{XMAX} &= \text{MAX}\ \{x_P\} \\ \text{YMIN} &= \text{MIN}\ \{y_P\} \\ \text{YMAX} &= \text{MAX}\ \{y_P\} \end{aligned} \qquad P \in \text{LIST}$$

This ensures that the values of $\hat{x}$ and $\hat{y}$ lie between 0 and 1 and proves convenient in the triangulation process.

After the triangulation has been computed, by calling the subroutines BSORT and DELAUN, subroutine DELTRI resets the co-ordinates of the points to their original values. Upon exiting from DELTRI, the Delaunay triangles are numbered from 1 to NUMTRI. Their vertex and adjacent triangle lists are stored in $V(I, J)$ and $E(I, J)$ where $I = 1$, 3 and $J = 1$, NUMTRI.

Subroutine BSORT

As described previously, the Delaunay triangulation is constructed by inserting each point, one at a time, into an existing triangulation. Before updating the triangulation, we first need to find an existing triangle which encloses the point to be inserted. The searching procedure used in the current algorithm is due to Lawson[2] and is implemented in the function subprogram TRILOC (to be described later). This process is initiated at the triangle most recently



triangle	1	2	3
vertices	1	2	3
	4	4	1
	2	3	2

V

triangle	1	2	3
adjacent triangles	0	1	0
	2	0	1
	3	3	2

E

(1) vertices listed anticlockwise
(2) first vertex at point of contact of first and third adjacent triangles

(1) adjacent triangles listed anticlockwise
(2) zero denotes no adjacent triangle

Figure 4. Data structure for triangulation algorithm

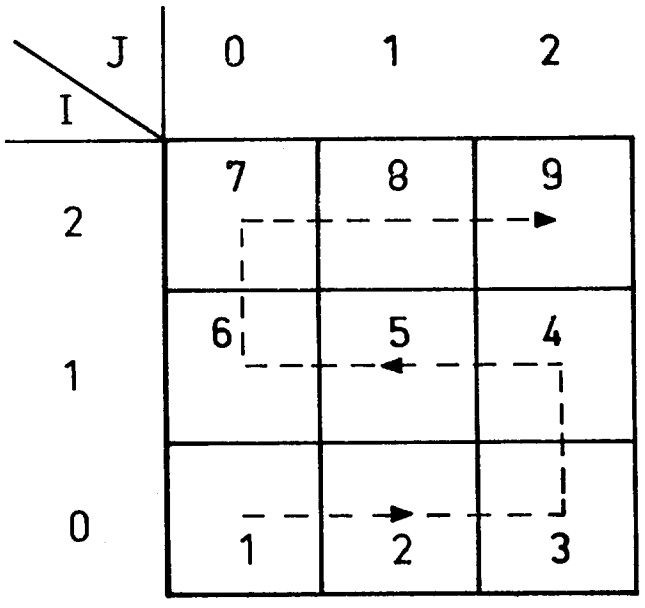


Figure 5. Bin sorting procedure for points in the plane

created and marches from one triangle to the next in the direction of the point to be inserted. The searching algorithm may be made efficient by presorting the points so that the distance between each point and its predecessor is small, thus reducing the number of triangles that need to be checked.

Following Lee and Schachter,[4] we ensure that consecutive points are in close proximity by using a bin sort. In this procedure, which is implemented by subroutine BSORT, the region to be triangulated is covered by a grid of rectangles called bins. Each point is placed in a bin according to its co-ordinates and these bins are accessed as shown in Fig. 5. Assuming that the points are distributed uniformly in the x-y plane, we expect the number of points in each bin to be approximately equal. Empiricial testing has indicated that it is sufficient to partition the domain to be triangulated into approximately $N^{1/2}$ bins. Thus the number of bins in the x- and y-directions, NDIV, is chosen as $N^{1/4}$ (to the nearest integer). Since the normalised co-ordinates for the points lie in the range from 0 to 1, the x- and y-dimensions of each bin are given by $\hat{x}_{max}$/NDIV and $\hat{y}_{max}$/NDIV. (In practice, these are multiplied by a number slightly greater than one to ensure that points with the maximum x- or y-co-ordinates fall within a bin.) To find the bin to which each point belongs, we first compute the row and column indices (I, J) according to

$$I = \text{INT}(\hat{y} * \text{NDIV} * 0.99/\hat{y}_{max})$$

$$J = \text{INT}(\hat{x} * \text{NDIV} * 0.99/\hat{x}_{max})$$

where $(\hat{x}, \hat{y})$ are the normalised co-ordinates. The bin number for the point is then given by

$$\text{BIN} = I * \text{NDIV} + J + 1 \quad (\text{I even})$$

$$\text{BIN} = (I + 1) * \text{NDIV} - J \quad (I \text{ odd})$$

After each point has been assigned a bin number, subroutine BSORT calls a quicksort routine to sort the points in ascending sequence of bin number. This completes the purpose of subroutine BSORT, since the points in LIST are now ordered so that consecutive points are in close proximity to one another. In passing, we note that the bin sort procedure may be omitted from the overall algorithm if desired. This will not affect the function of the program, but will decrease its efficiency for cases where the points are distributed uniformly in the x-y plane. This aspect will be examined further in a later section of the paper.

Subroutine QSORTI

This subroutine sorts the list of points in ascending sequence of their bin numbers and is called from subroutine BSORT. It uses the quicksort algorithm and is discussed fully in a paper by Houlsby and Sloan.[10]

Subroutine DELAUN

This routine is the heart of the triangulation algorithm. To begin the process, we define the vertices and co-ordinates for the supertriangle. The vertices of the supertriangle are allocated the numbers NUMPTS + 1, NUMPTS + 2 and NUMPTS + 3, and the corresponding co-ordinates are set to $(-100, -100)$, $(100, -100)$ and $(0, 100)$. The supertriangle is initially stored in the first column of the vertex and adjacency arrays, and its vertex co-ordinates are stored in the last three locations of the X and Y vectors. Since the x- and y-co-ordinates have been normalised to lie within the range 0 to 1, all of the points to be triangulated are automatically enclosed by the supertriangle. The size and shape of the supertriangle may be chosen arbitrarily. It is sufficient merely that it contains all of the points to be triangulated. It is worth noting, however, that if the corners of the supertriangle are very close to the window enclosing the points, the boundary of the final triangulation may be locally concave. Strictly speaking, such a triangulation does not correspond exactly to a Delaunay triangulation, since some long thin triangles along the boundary have been omitted. In most practical applications this is not a disadvantage, but may be avoided by locating the vertices of the supertriangle further away from the enclosing window.

After the supertriangle has been defined, subroutine DELAUN inserts each of the points into the triangulation one at a time. To introduce a new point P, a search is made to find an existing triangle T which encloses P. Triangle T is then deleted and three new triangles are created by connecting P to each of its vertices (Fig. 6). Note that each of these triangles is created such that P is the first vertex in the vertex array. The net gain in the total number of triangles is two, and the additional triangles are numbered NUMTRI + 1 and NUMTRI + 2 where NUMTRI is the total number of triangles prior to the insertion of P. Next, each triangle containing P as a vertex is placed on a last-in, first-out stack (provided that the edge opposite P is adjacent to some other triangle). With reference to Fig. 6, the triangles T, NUMTRI + 1 and NUMTRI + 2 are placed on the stack and the triangles opposite P are respectively A, B and C. Note that it is unnecessary to maintain a separate stack of adjacent triangles which are opposite to P, since these can be extracted from the adjacency arrays for the stacked triangles. For example, with reference to Fig. 6, the adjacent triangle which is opposite to P in triangle T is given by $A = E(2, T)$. In general for element I in the stack, the opposite adjacent triangle is given by $E(2, I)$. This completes the initial insertion phase in subroutine DELAUN, and we are now ready to update the triangulation to a Delaunay triangulation using the swapping algorithm of Lawson.[2]

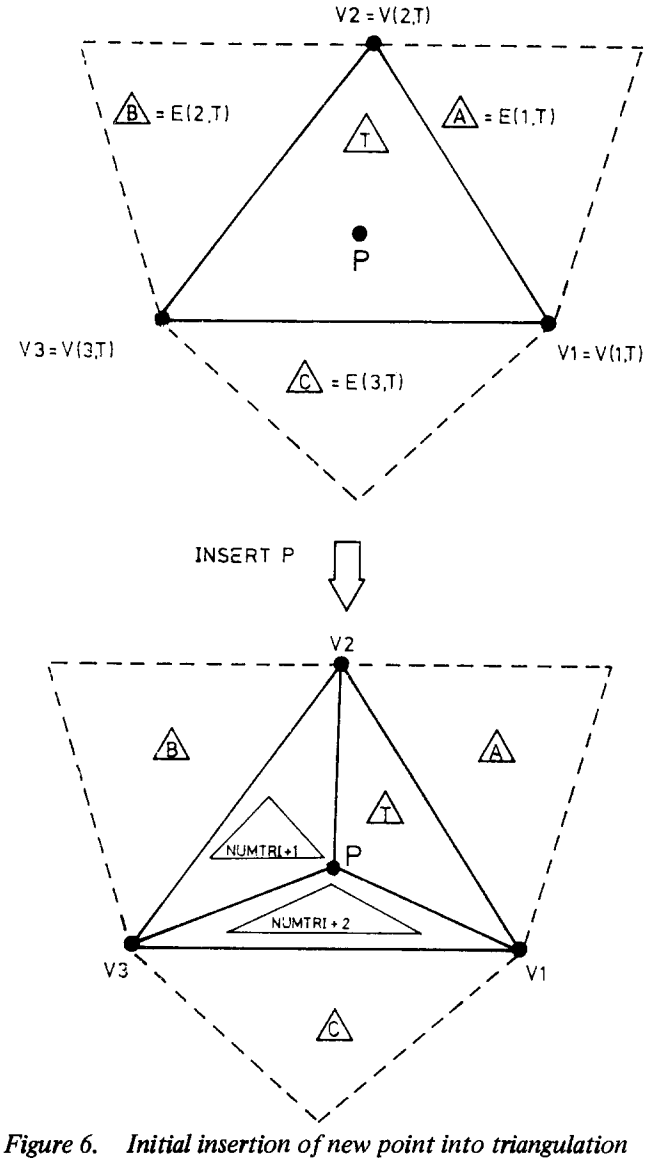


Figure 6. Initial insertion of new point into triangulation

In Lawson's procedure, we remove each triangle from the stack one at a time. The notation used in subroutine DELAUN is shown in Fig. 7. Triangle L is the triangle removed from the stack. Triangle R is the triangle opposite point P which is adjacent to L. Triangles L and R share the edge $V1 - V2$ and form a quadrilateral with vertices $P - V2 - V3 - V1$ (triangle L is to the left of $V2 - V1$ and triangle R is to the right of $V2 - V1$). If the point P is inside the circumcircle of triangle R, then the diagonal $V1 - V2$ needs to be replaced with the diagonal $P - V3$ to preserve the structure of the Delaunay triangulation. As pointed out by Lawson,[2] this circumcircle check maximises the minimum angle occurring in any pair of adjacent triangles forming a convex quadrilateral. If a swap is necessary, as shown in Fig. 7, the vertex and adjacent arrays for triangles L and R are updated (again with P as the first vertex in the vertex arrays), as are the adjacency arrays for triangles A and C. Provided that there is a triangle opposite P which is adjacent to L, triangle L is placed on the stack. Similarly for triangle R. The next triangle is then removed from the stack and the whole process is repeated until the stack is empty. This signifies that the insertion of P is complete and the new triangulation is a Delaunay triangulation.

In subroutine DELAUN, the stacked triangles are stored in the vector STACK which has a length of NUMPTS. This dimension was found to be sufficient for triangulating 10 000 points distributed randomly within a square domain, but may be increased if the need arises.

After all of the points have been triangulated, subroutine DELAUN deletes any triangle which contains one or more supertriangle vertices. During this phase the vertex and adjacency arrays are updated to fill any blanks created and the Delaunay triangles are numbered from 1 to NUMTRI. The vertex and adjacent triangle lists are stored in $V(I,J)$ and $E(I,J)$ respectively, where $I = 1, 3$ and $J = 1$, NUMTRI.

Subroutine PUSH

This subroutine places an item on a last-in, first out stack and is easily understood. The maximum permissible size of the stack is MAXSTK, and this is defined in subroutine DELAUN. Subroutine PUSH includes a check to determine if there is sufficient space in STACK to complete the triangulation. If this is not the case, then a diagnostic message is printed and the execution of the program is halted.

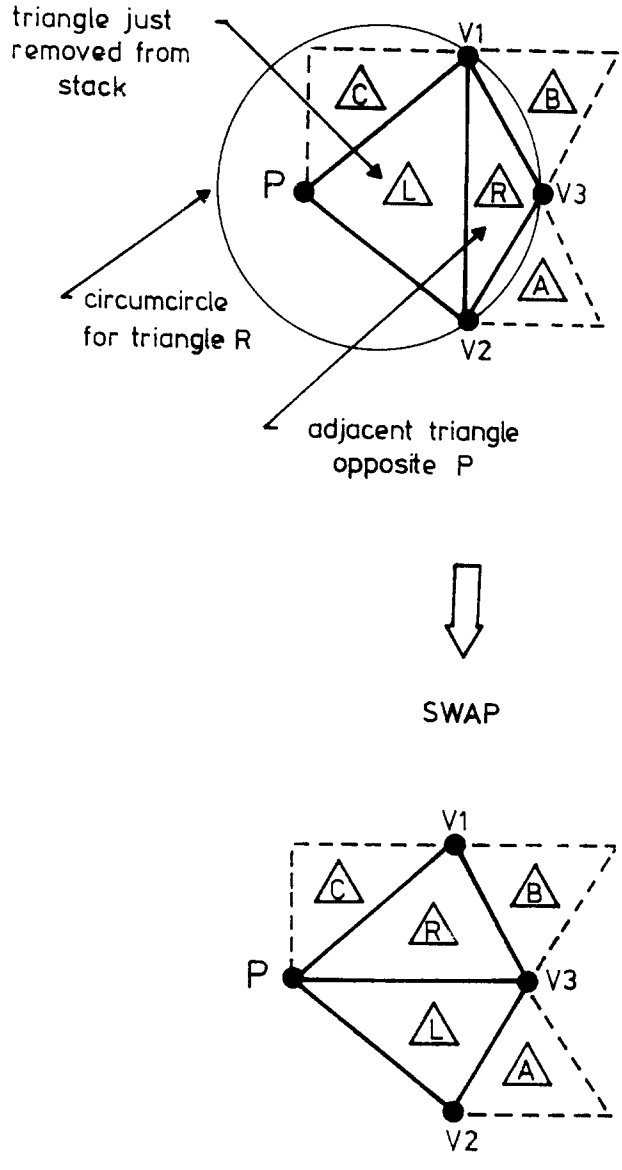


Figure 7. Implementation of Lawson's swapping algorithm

Function POP

This function removes an item from the top of a last-in, first-out stack and is complementary to subroutine PUSH. It includes a check to determine if the stack is already empty before attempting to remove an item. If this is so, a diagnostic message is printed and the execution of the program is halted.

Function EDG

This function finds the number of the edge (i.e. the row number in the triangle adjacency array) in element I which is adjacent to element J. Calls to this function should always results in a positive match. If elements I and J are found to be non-adjacent, function EDG prints a diagnostic message and terminates the execution of the program.

Function TRILOC

This function accepts the x-y co-ordinates of a point and finds an existing triangle which encloses it. The search is started at the triangle which has been most recently created, and checks if the point is to the right of any of its edges. Since the triangle vertices are always listed in an anticlockwise sequence, a point can only be enclosed by a triangle if it is to the left of each of its edges. (A point which lies on one of the edges of a triangle is also said to be enclosed by the triangle.) If the point lies to the right of any edge of the triangle, then the search shifts to the triangle which is adjacent to this edge and the process is repeated. In this way, the search marches from one triangle to the next in the general direction of the point as shown in Fig. 8. This ingenious searching algorithm is due to Lawson[2] and avoids the need to search all of the triangles in the grid.

Function SWAP

This function checks to determine if a pair of adjacent triangles form a convex quadrilateral with the maximum minimum angle. The adjacent triangles share an edge $V1-V2$ and form a quadrilateral with vertices $P-V2-V3-V1$ as shown in Fig. 9. The diagonal $V1-V2$ is replaced with the diagonal $P-V3$ if P lies inside the circumcircle for the triangle $V1-V2-V3$. With reference to Fig. 9, we see that P lies inside the circumcircle if $2\pi r - 2r\alpha < 2r\beta$, i.e. if $\alpha+\beta>\pi$. Similarly, P lies outside the circumcircle if $\alpha+\beta<\pi$. The neutral case

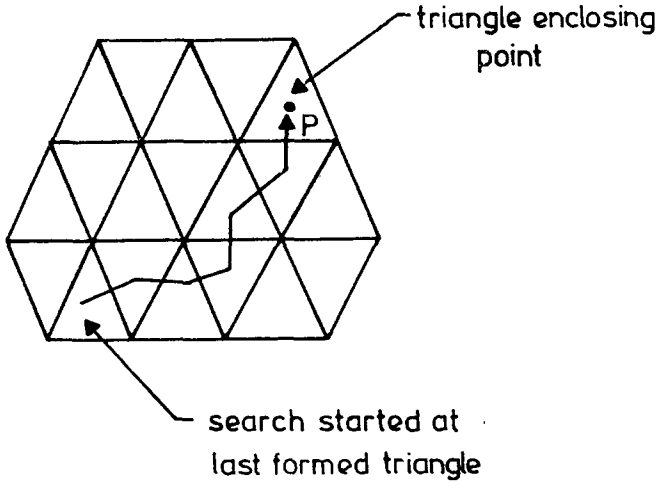


Figure 8. Triangle searching algorithm.

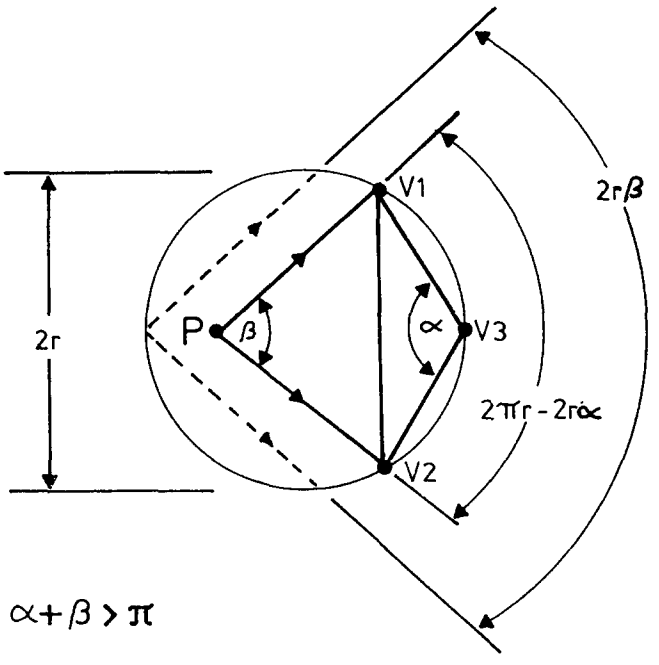


Figure 9. Geometry for circumcircle test

occurs when P lies on the circumcircle and $\pi=\alpha+\beta$. Since $\alpha+\beta<2\pi$, a swap needs to be performed if

$$\sin(\alpha+\beta)<0$$

Using the formula

$$\sin(\alpha+\beta)=\cos(\alpha)\sin(\beta)+\sin(\alpha)\cos(\beta)$$

this condition is equivalent to

$$\frac{x_{13}x_{23}+y_{13}y_{23}}{[\{x_{13}^2+y_{13}^2\}\{x_{23}^2+y_{23}^2\}]^{1/2}} \times \frac{x_{2P}y_{1P}-x_{1P}y_{2P}}{[\{x_{2P}^2+y_{2P}^2\}\{x_{1P}^2+y_{1P}^2\}]^{1/2}} + \frac{x_{13}y_{23}-x_{23}y_{13}}{[\{x_{13}^2+y_{13}^2\}\{x_{23}^2+y_{23}^2\}]^{1/2}} \times \frac{x_{2P}x_{1P}+y_{2P}y_{1P}}{[\{x_{2P}^2+y_{2P}^2\}\{x_{1P}^2+y_{1P}^2\}]^{1/2}}<0$$

where

$$x_{13}=x_1-x_3 \qquad y_{13}=y_1-y_3$$
$$x_{23}=x_2-x_3 \qquad y_{23}=y_2-y_3$$
$$x_{1P}=x_1-x_P \qquad y_{1P}=y_1-y_P$$
$$x_{2P}=x_2-x_P \qquad y_{2P}=y_2-y_P$$

Thus, P lies inside the circumcircle if

$$(x_{13}x_{23}+y_{13}y_{23})(x_{2P}y_{1P}-x_{1P}y_{2P}) < (y_{13}x_{23}-x_{13}y_{23})(x_{2P}x_{1P}+y_{1P}y_{2P})$$

This check, which is due to Cline and Renka,[7] is particularly efficient since it requires only ten multiplications, two additions and two subtractions. As pointed out by these authors, however, round off error may cause an incorrect decision to be made when $\sin(\alpha+\beta)$ approaches zero. This condition arises when:

(1) $\alpha+\beta$ is near π.

(2) α and β are both near 0.
(3) α and β are both near π.

The first case occurs when P is very close to the circumcircle of an adjacent triangle, whilst the second and third cases occur when the four vertices of the adjacent triangles are nearly collinear. A complete discussion of this problem has been given by Cline and Renka.[7] They established that the first condition has no ill-effects on the construction of a triangulation, except that the outcome of a swap test is not predictable. Allowing for the precision of floating point arithmetic, the triangulation produced will still be a Delaunay triangulation. The second and third cases, however, need to be accounted for as they may result in an incorrect triangulation. If α and β are both in the vicinity of π, a swap should be performed. The Cline and Renka[7] test, which is implemented in the logical function SWAP, is as follows:

STEP 1: Set $\text{COSA} = x_{13}x_{23} + y_{13}y_{23}$

$\text{COSB} = x_{2P}x_{1P} + y_{2P}y_{1P}$

STEP 2: If $\text{COSA} \geqslant 0$ and $\text{COSB} \geqslant 0$ then

Set SWAP to FALSE and EXIT

STEP 3: If $\text{COSA} < 0$ and $\text{COSB} < 0$ then

Set SWAP to TRUE and EXIT

STEP 4: Set $\text{SINA} = x_{13}y_{23} - x_{23}y_{13}$

$\text{SINB} = x_{2P}y_{1P} - x_{1P}y_{2P}$

and

$\text{SINAB} = \text{SINA}*\text{COSB} + \text{SINB}*\text{COSA}$

STEP 4: If $\text{SINAB} < 0$ then set SWAP to TRUE and EXIT. Else, set SWAP to FALSE and EXIT.

Although this algorithm requires a number of additional comparisons, it has been found to be relatively efficient as well as being numerically stable.

ANALYSIS OF ALGORITHM

In the bin sorting phase, the largest amount of work occurs in the quicksort procedure which requires an average of $0(N \log_2 N)$ operations. The actual triangulation algorithm is comprised of two distinct steps; the searching step and the swapping step. If the distribution of the points throughout the x-y plane is reasonably uniform, there will be roughly $0(N^{1/2})$ points in each bin. If the bins are approximately square, $0(N^{1/4})$ triangles will be searched to find the triangle enclosing each newly introduced point. The swapping algorithm requires roughly a constant number of operations for each point. (Empirical tests, for points distributed randomly within a square domain, indicate that an average of three swaps per point are necessary.) Once the triangulation has been completed, $0(N)$ operations are required to delete all of the triangles that contain one or more of the supertriangle vertices. Thus, overall, the algorithm requires an average of $0(N \log_2 N) + 0(N^{5/4}) + 0(N) + 0(N)$ operations. For large N, the average run time of the scheme is $0(N^{5/4})$.

As noted in a previous section, the bin sorting phase may be omitted from the algorithm if desired. The searching step will then examine roughly $0(N^{1/2})$ triangles as each point is inserted and, overall, an average of $0(N^{3/2}) + 0(N) + 0(N)$ operations are required. Thus the average run time of the algorithm without the bin sort is $0(N^{3/2})$.

The worst case run time for the algorithm is $0(N^2)$ and occurs, for example, when the set of data points lie on a parabola. This example is discussed in detail by Lee and Schachter,[4] and is the worst case for a variety of Delaunay triangulation schemes.

Excluding the memory required to store the co-ordinates of the points and supertriangle vertices, subroutine DELTRI requires a total of $14N + 6$ integer words of memory to compute and store the Delaunay triangulation. In addition to this requirement, subroutine QSORTI needs 64 words of locally-declared integer memory for the operation of two stacks (this is sufficient to sort a list of 2^{32} points).

APPLICATIONS

To assess the validity and efficiency of the proposed algorithm, it was applied to sets of points distributed randomly within a unit square. Figure 10 illustrates the Delaunay triangulation for ten such points. The performance of the scheme was measured by constructing Delaunay triangulations for sets of 100, 500, 1000, 3000, 4000, 5000 and 10000 points. The CPU times for the proposed algorithm are shown in Table 2, together with the CPU times for the implementations given by Sloan and Houlsby[9] and Renka.[8] These statistics are for the VAX 11/780 with full optimisation on the FORTRAN 77 compiler, and were measured using the internal clock. Two versions of the proposed algorithm were run; one with bin sorting and one without bin sorting. To assess the validity of the triangulation produced, a number of checks were conducted (the CPU times for these checks are not included in the timing statistics). Firstly, each triangle was tested to ensure that no data point lay within its circumcircle (allowing for the precision of the machine). Secondly, the area of each

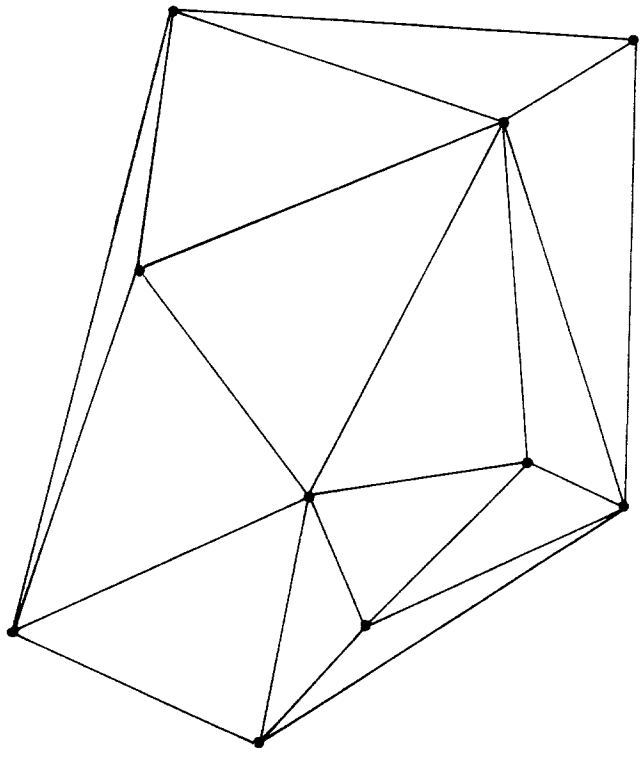


Figure 10. Delaunay triangulation for ten such points

Table 2. Timing statistics for triangulation algorithms (points distributed randomly over a unit square)

N	Proposed algorithm		Proposed algorithm (no bin sort)		Watson algorithm (Sloan and Houlsby[9])		Cline and Renka algorithm (Renka[8])		T_2/T_1	T_3/T_1	T_4/T_1
	T_1	a	T_2	a	T_3	a	T_4	a			
100	0.36		0.36		0.41		0.37		1.00	1.14	1.03
		1.07		1.16		1.17		1.28			
500	2.06		2.32		2.68		2.92		1.13	1.30	1.42
		1.01		1.21		1.31		1.26			
1000	4.15		5.35		6.66		7.01		1.29	1.60	1.69
		1.06		1.23		1.45		1.29			
2000	8.67		12.59		18.18		17.10		1.45	2.10	1.97
		1.10		1.32		1.52		1.24			
3000	13.55		21.53		33.67		28.87		1.59	2.48	2.13
		1.08		1.34		1.48		1.23			
4000	18.50		31.67		51.40		41.18		1.71	2.78	2.23
		1.03		1.29		1.57		1.35			
5000	23.30		42.19		73.01		55.65		1.81	3.13	2.39
		1.06		1.36		1.48		1.30			
10000	48.71		108.48		203.29		137.45		2.23	4.17	2.82

Notes:

1. All times in seconds for VAX 11/780.
2. 'a' values obtained by assuming times are $0(N^a)$.
3. Sloan and Houlsby[9] implementation converted to single precision arithmetic.
4. Renka[8] implementation run with points presorted in ascending sequence of their x-co-ordinates.

triangle was computed and a check made to ensure that it was positive. Finally, the adjacency arrays were used to check that the triangulation obeyed Euler's formula for planar graphs, i.e. $N - N_e + N_t = 1$ where N_e is the number of edges and N_t is the number of triangles.

The results shown in Table 2 indicate that the performance of the proposed algorithm compares favourably with that of the Watson[6,9] and Cline and Renka[7,8] algorithms. In all cases the CPU time required by the proposed scheme is less than that of the other schemes. The savings are most pronounced in comparison with the Watson algorithm which, for N values greater than about 2000, rapidly becomes uncompetitive. The Cline and Renka scheme, whilst more efficient than the Watson scheme, also requires more than twice the amount of CPU time for N greater than about 2000. The advantage of the Watson and Cline and Renka procedures is that they require less storage than the proposed algorithm (slightly greater than $9N$ and $7N$ words of integer memory respectively). The proposed scheme uses more memory because of the need to store the element adjacency arrays. In many applications, however, such as the construction of contour plots, this information is useful and often needs to be generated in any case. The $14N + 6$ words of memory required by the proposed algorithm is not considered to be excessive, particularly for modern machines with virtual memory, and appears to be justified by the increase in efficiency.

When the proposed algorithm is run without the bin sort, it is still relatively efficient. For large sets of points, however, the bin sort procedure reduces the total CPU time requirement significantly. Thus, provided the points are distributed throughout the x-y plane in a reasonably uniform manner, it would appear advisable to include this option.

The observed average case run times for the Watson and Cline and Renka schemes are in reasonable agreement with the theoretical predictions. As shown in Table 1, the average run times for these two algorithms are expected to be $0(N^{3/2})$ and $0(N^{4/3})$ respectively. Averaging the results for all values of N considered, the observed run times are approximately $0(N^{1.42})$ and $0(N^{1.28})$. The observed run times for the proposed algorithm, however, grow at a rate which is substantially less than the theoretical prediction of $0(N^{5/4})$. For values of N between 100 and 10000, the observed run time of the proposed scheme is approximately $0(N^{1.06})$. Empirical tests indicate that the theoretical operation counts are correct. The apparent discrepancy is due to the fact that the time required for one iteration of the search procedure is substantially less than the time required for one iteration of the swapping procedure. For the values of N considered, the average number of searches conducted is small and most of the time is spent in the swapping phase. Thus, overall, the average run time of the algorithm is slightly greater than $0(N)$ unless N is very large. Similar arguments hold when the proposed algorithm is used without the bin sort, except that the discrepancy between the predicted and observed run times is less due to the increased number of searches conducted.

CONCLUSIONS

An algorithm has been described for computing Delaunay triangulations in the plane. A FORTRAN 77 implementation of the scheme is given. Empricial tests indicate that the procedure is efficient and may be used for both small and large sets of points.

REFERENCES

1 Green, P. J. and Sibson, R. Computing Dirichlet tessellations in the plane, *The Computer Journal*, (1978) **21**, 168

2 Lawson, C. L. Software for C^1 interpolation, in Rice, J. (ed.) *Mathematical Software III*, Academic Press, New York, (1977) pp. 161-194.

3 Sibson, R. Locally equiangular triangulations, *The Computer Journal*, (1978) 21, 243
4 Lee, D. T. and Schachter, B. J. Two algorithms for constructing a Delaunay triangulation, *International Journal of Computer and Information Sciences*, (1980) 9, 219
5 Bowyer, A. Computing Dirichlet tesseliations, *The Computer Journal*, (1981) 24, 162
6 Watson, D. F. Computing the *n*-dimensional Delaunay triangulation with application to Voronoi polytopes, *The Computer Journal*, (1981) 24, 167
7 Cline, A. K. and Renka, R. L. A storage efficient method for construction of a Thiessen triangulation, *Rocky Mountain Journal of Mathematics*, (1984) 14, 119
8 Renka, R. L. Algorithm 624: Triangulation and interpolation at arbitrarily distributed points in the plane, *ACM Transactions on Mathematical Software*, (1984) 10, 440
9 Sloan, S. W. and Houlsby, G. T. An implementation of Watson's algorithm for computing two-dimensional Delaunay triangulations, *Advances in Engineering Software*, (1984) 6, 192
10 Houlsby, G. T. and Sloan, S. W. Efficient sorting routines in FORTRAN 77, *Advances in Engineering Software*, (1984) 6, 198

APPENDIX ONE: Delaunay Triangulation Program

```
C
C
      SUBROUTINE DELTRI(NUMPTS,N,X,Y,LIST,BIN,V,E,NUMTRI)
C*********************************************************************
C
C     SUBROUTINE DELTRI
C
C     PURPOSE:
C     --------
C
C     ASSEMBLE DELAUNAY TRIANGULATION FOR COLLECTION OF POINTS IN THE
C     PLANE
C
C     INPUT:
C     ------
C
C     'NUMPTS' - TOTAL NUMBER OF POINTS IN DATA SET
C
C     'N'      - TOTAL NUMBER OF POINTS TO BE TRIANGULATED
C              - N LE NUMPTS
C
C     'X'      - X-COORDS OF ALL POINTS IN DATA SET
C              - X-COORD OF POINT I GIVEN BY X(I)
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE X-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'Y'      - Y-COORDS OF ALL POINTS IN DATA SET
C              - Y-COORD OF POINT I GIVEN BY Y(I)
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE Y-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'LIST'   - LIST OF POINTS TO BE TRIANGULATED
C              - LIST OF LENGTH N
C              - IF N EQ NUMPTS, SET LIST(I)=I FOR I=1,2,...,NUMPTS
C                PRIOR TO CALLING THIS ROUTINE
C
C     'BIN'    - NOT DEFINED
C              - LIST OF LENGTH NUMPTS
C              - USED AS WORKSPACE IN SUBROUTINES BSORT AND DELAUN
C
C     'V'      - NOT DEFINED
C              - V HAS DIMENSIONS V(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'E'      - NOT DEFINED
C              - E HAS DIMENSIONS E(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'NUMTRI' - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'NUMPTS' - UNCHANGED
C
C     'N'      - UNCHANGED
C
C     'X'      - UNCHANGED, EXCEPT THAT LAST THREE LOCATIONS CONTAIN
C                NORMALISED X-COORDS OF SUPERTRIANGLE VERTICES
C
C     'Y'      - UNCHANGED, EXCEPT THAT LAST THREE LOCATIONS CONTAIN
C                NORMALISED Y-COORDS OF SUPERTRIANGLE VERTICES
C
C     'LIST'   - UNCHANGED
```

```
C
C     'BIN'    - NOT DEFINED
C
C     'V'      - VERTEX ARRAY FOR TRIANGULATION
C              - VERTICES LISTED IN ANTICLOCKWISE SEQUENCE
C              - VERTICES FOR TRIANGLE J ARE FOUND IN V(I,J) FOR I=1,2,3
C                AND J=1,2,...,NUMTRI
C              - FIRST VERTEX IS AT POINT OF CONTACT OF FIRST AND THIRD
C                ADJACENT TRIANGLES
C
C     'E'      - ADJACENCY ARRAY FOR TRIANGULATION
C              - TRIANGLES ADJACENT TO J ARE FOUND IN E(I,J) FOR I=1,2,3
C                AND J=1,2,...,NUMTRI
C              - ADJACENT TRIANGLES LISTED IN ANTICLOCKWISE SEQUENCE
C              - ZERO DENOTES NO ADJACENT TRIANGLE
C
C     'NUMTRI' - TOTAL NUMBER OF TRIANGLES IN FINAL TRIANGULATION
C              - NUMTRI LT 2*N+1
C
C     SUBROUTINES CALLED:
C     -------------------
C
C     BSORT,DELAUN
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*************************************************************************
C
      INTEGER N,I,LIST(*),V(3,*),E(3,*),NUMTRI,BIN(*),P,NUMPTS
C
      REAL XMIN,XMAX,YMIN,YMAX,DMAX,C00001,FACT,X(*),Y(*)
C
      PARAMETER(C00001=1.0)
C
C     COMPUTE MIN AND MAX COORDS FOR X AND Y
C     COMPUTE MAX OVERALL DIMENSION
C
      XMIN=X(LIST(1))
      XMAX=XMIN
      YMIN=Y(LIST(1))
      YMAX=YMIN
      DO 5 I=2,N
        P=LIST(I)
        XMIN=MIN(XMIN,X(P))
        XMAX=MAX(XMAX,X(P))
        YMIN=MIN(YMIN,Y(P))
        YMAX=MAX(YMAX,Y(P))
    5 CONTINUE
      DMAX=MAX(XMAX-XMIN,YMAX-YMIN)
C
C     NORMALISE X-Y COORDS OF POINTS
C
      FACT=C00001/DMAX
      DO 10 I=1,N
        P=LIST(I)
        X(P)=(X(P)-XMIN)*FACT
        Y(P)=(Y(P)-YMIN)*FACT
   10 CONTINUE
C
C     SORT POINTS INTO BINS
C     THIS CALL IS OPTIONAL
C
      CALL BSORT(N,X,Y,XMIN,XMAX,YMIN,YMAX,DMAX,BIN,LIST)
C
C     COMPUTE DELAUNAY TRIANGULATION
C
      CALL DELAUN(NUMPTS,N,X,Y,LIST,BIN,V,E,NUMTRI)
C
C     RESET X-Y COORDS TO ORIGINAL VALUES
C
      DO 30 I=1,N
       P=LIST(I)
       X(P)=X(P)*DMAX+XMIN
       Y(P)=Y(P)*DMAX+YMIN
   30 CONTINUE
C
      END
```

```
C
C
C
C
      SUBROUTINE BSORT(N,X,Y,XMIN,XMAX,YMIN,YMAX,DMAX,BIN,LIST)
C
C***********************************************************************
C
C     SUBROUTINE BSORT
C
C     PURPOSE:
C     --------
C
C     SORT POINTS SUCH CONSECUTIVE POINTS ARE CLOSE TO ONE ANOTHER IN THE
C     X-Y PLANE USING A BIN SORT
C
C     INPUT:
C     ------
C
C     'N'      - TOTAL NUMBER OF POINTS TO BE TRIANGULATED
C              - N LE NUMPTS, WHERE NUMPTS IS TOTAL NUMBER OF POINTS IN
C                DATA SET
C
C     'X'      - X-COORDS OF ALL POINTS IN DATA SET
C              - IF POINT IS IN LIST,THE COORDINATE MUST BE NORMALISED
C                ACCORDING TO X=(X-XMIN)/DMAX
C              - X-COORD OF POINT I GIVEN BY X(I)
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE X-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'Y'      - Y-COORDS OF ALL POINTS IN DATA SET
C              - IF POINT IS IN LIST,THE COORDINATE MUST BE NORMALISED
C                ACCORDING TO Y=(Y-YMIN)/DMAX
C              - Y-COORD OF POINT I GIVEN BY Y(I)
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE Y-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'XMIN'   - MIN X-COORD OF POINTS IN LIST
C
C     'XMAX'   - MAX X-COORD OF POINTS IN LIST
C
C     'YMIN'   - MIN Y-COORD OF POINTS IN LIST
C
C     'YMAX'   - MAX Y-COORD OF POINTS IN LIST
C
C     'DMAX'   - DMAX=MAX(XMAX-XMIN,YMAX-YMIN)
C
C     'BIN'    - NOT DEFINED
C              - LIST OF LENGTH NUMPTS
C
C     'LIST'   - LIST OF POINTS TO BE TRIANGULATED
C              - LIST OF LENGTH N
C
C     OUTPUT:
C     -------
C
C     'N'      - UNCHANGED
C
C     'X'      - UNCHANGED
C
C     'Y'      - UNCHANGED
C
C     'XMIN'   - UNCHANGED
C
C     'XMAX'   - UNCHANGED
C
C     'YMIN'   - UNCHANGED
C
C     'YMAX'   - UNCHANGED
C
C     'DMAX'   - UNCHANGED
C
C     'BIN'    - BIN NUMBERS FOR EACH POINT TO BE TRIANGULATED
C              - LIST OF LENGTH NUMPTS
C
C     'LIST'   - LIST OF POINTS TO BE TRIANGULATED
C              - POINTS ORDERED SUCH THAT CONSECUTIVE POINTS ARE CLOSE
C                TO ONE ANOTHER IN THE X-Y PLANE
C
C     SUBROUTINES CALLED:
C     -------------------
C
C     QSORTI
```

```
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C*************************************************************************
C
      INTEGER LIST(*),BIN(*),N,I,J,K,P,NDIV
C
      REAL X(*),Y(*),FACTX,FACTY,XMIN,XMAX,YMIN,YMAX,DMAX
C
C     COMPUTE NUMBER OF BINS IN X-Y DIRECTIONS
C     COMPUTE INVERSE OF BIN SIZE IN X-Y DIRECTIONS
C
      NDIV=NINT(REAL(N)**0.25)
      FACTX=REAL(NDIV)/((XMAX-XMIN)*1.01/DMAX)
      FACTY=REAL(NDIV)/((YMAX-YMIN)*1.01/DMAX)
C
C     ASSIGN BIN NUMBERS TO EACH POINT
C
      DO 10 K=1,N
        P=LIST(K)
        I=INT(Y(P)*FACTY)
        J=INT(X(P)*FACTX)
        IF(MOD(I,2).EQ.0)THEN
          BIN(P)=I*NDIV+J+1
                          ELSE
          BIN(P)=(I+1)*NDIV-J
        END IF
   10 CONTINUE
C
C     SORT POINTS IN ASCENDING SEQUENCE OF BIN NUMBER
C
      CALL QSORTI(N,LIST,BIN)
C
      END
C
C
C
C
      SUBROUTINE QSORTI(N,LIST,KEY)
C
C*************************************************************************
C     SUBROUTINE QSORTI
C
C     PURPOSE:
C     --------
C
C     ORDER LIST OF INTEGERS IN ASCENDING SEQUENCE OF THEIR INTEGER KEYS
C
C     INPUT:
C     ------
C
C     'N'     - POSITIVE INTEGER GIVING LENGTH OF LIST
C
C     'LIST'  - LIST OF INTEGERS TO BE SORTED
C               - LIST OF LENGTH N
C
C     'KEY'   - LIST OF INTEGER KEYS
C               - LIST OF LENGTH GE N
C
C     OUTPUT:
C     -------
C
C     'N'     - UNCHANGED
C
C     'LIST'  - LIST OF INTEGERS SORTED IN ASCENDING SEQUENCE OF THEIR
C               KEYS
C
C     'KEY'   - UNCHANGED
C
C     NOTES:
C     ------
C
C     - USES QUICKSORT ALGORITHM, EFFICIENT FOR 'N' VALUES GREATER THAN
C       ABOUT 12 (ALTHOUGH MAY BE SYSTEM DEPENDENT)
C     - ROUTINE SORTS LISTS UP TO LENGTH 2**MAXSTK
C
C     PROGRAMMER:
```

```
C     -----------
C
C     G T HOULSBY
C
C     LAST MODIFIED:
C     -------------
C
C     7 MAY 1985      S W SLOAN
C
C*********************************************************************
C
      INTEGER LIST(*),KEY(*),N,LL,LR,LM,NL,NR,LTEMP,STKTOP,MAXSTK,GUESS
C
      PARAMETER(MAXSTK=32)
C
      INTEGER LSTACK(MAXSTK), RSTACK(MAXSTK)
C
      LL=1
      LR=N
      STKTOP=0
   10 IF(LL.LT.LR)THEN
        NL=LL
        NR=LR
        LM=(LL+LR)/2
        GUESS=KEY(LIST(LM))
C
C       FIND KEYS FOR EXCHANGE
C
   20   IF(KEY(LIST(NL)).LT.GUESS)THEN
          NL=NL+1
          GOTO 20
        END IF
   30   IF(GUESS.LT.KEY(LIST(NR)))THEN
          NR=NR-1
          GOTO 30
        END IF
        IF(NL.LT.(NR-1))THEN
          LTEMP=LIST(NL)
          LIST(NL)=LIST(NR)
          LIST(NR)=LTEMP
          NL=NL+1
          NR=NR-1
          GOTO 20
        END IF
C
C       DEAL WITH CROSSING OF POINTERS
C
        IF(NL.LE.NR)THEN
          IF(NL.LT.NR)THEN
            LTEMP=LIST(NL)
            LIST(NL)=LIST(NR)
            LIST(NR)=LTEMP
          END IF
          NL=NL+1
          NR=NR-1
        END IF
C
C       SELECT SUB-LIST TO BE PROCESSED NEXT
C
        STKTOP=STKTOP+1
        IF(NR.LT.LM)THEN
          LSTACK(STKTOP)=NL
          RSTACK(STKTOP)=LR
          LR=NR
        ELSE
          LSTACK(STKTOP)=LL
          RSTACK(STKTOP)=NR
          LL=NL
        END IF
        GOTO 10
      END IF
C
C     PROCESS ANY STACKED SUB-LISTS
C
      IF(STKTOP.NE.0)THEN
        LL=LSTACK(STKTOP)
        LR=RSTACK(STKTOP)
        STKTOP=STKTOP-1
        GOTO 10
      END IF
C
      END
C
C
C
```

```
C
      SUBROUTINE DELAUN(NUMPTS,N,X,Y,LIST,STACK,V,E,NUMTRI)
C
C**********************************************************************
C
C     SUBROUTINE DELAUN
C
C     PURPOSE:
C     --------
C
C     ASSEMBLE DELAUNAY TRIANGULATION
C
C     INPUT:
C     ------
C
C     'NUMPTS' - TOTAL NUMBER OF POINTS IN DATA SET
C
C     'N'      - TOTAL NUMBER OF POINTS TO BE TRIANGULATED
C              - N LE NUMPTS
C
C     'X'      - X-COORDS OF ALL POINTS IN DATA SET
C              - X-COORD OF POINT I GIVEN BY X(I)
C              - IF POINT IS IN LIST, COORDINATE MUST BE NORMALISED
C                SUCH THAT X=(X-XMIN)/DMAX
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE X-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'Y'      - Y-COORDS OF ALL POINTS IN DATA SET
C              - Y-COORD OF POINT I GIVEN BY Y(I)
C              - IF POINT IS IN LIST, COORDINATE MUST BE NORMALISED
C                SUCH THAT Y=(Y-YMIN)/DMAX
C              - LIST OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS ARE USED TO STORE Y-COORDS OF
C                SUPERTRIANGLE VERTICES IN SUBROUTINE DELAUN
C
C     'LIST'   - LIST OF POINTS TO BE TRIANGULATED
C              - POINTS ARE ORDERED SUCH THAT CONSECUTIVE POINTS ARE
C                CLOSE TO ONE ANOTHER IN THE X-Y PLANE
C              - LIST OF LENGTH N
C
C     'STACK'  - NOT DEFINED
C              - LIST OF LENGTH NUMPTS
C              - USED AS WORKSPACE
C
C     'V'      - NOT DEFINED
C              - V HAS DIMENSIONS V(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'E'      - NOT DEFINED
C              - E HAS DIMENSIONS E(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'NUMTRI' - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'NUMPTS' - UNCHANGED
C
C     'N'      - UNCHANGED
C
C     'X'      - UNCHANGED
C
C     'Y'      - UNCHANGED
C
C     'LIST'   - UNCHANGED
C
C     'STACK'  - NOT DEFINED
C
C     'V'      - VERTEX ARRAY FOR TRIANGULATION
C              - VERTICES LISTED IN ANTICLOCKWISE SEQUENCE
C              - VERTICES FOR TRIANGLE J ARE FOUND IN V(I,J) FOR I=1,2,3
C                AND J=1,2,...,NUMTRI
C              - FIRST VERTEX IS AT POINT OF CONTACT OF FIRST AND THIRD
C                ADJACENT TRIANGLES
C
C     'E'      - ADJACENCY ARRAY FOR TRIANGULATION
C              - TRIANGLES ADJACENT TO J ARE FOUND IN E(I,J) FOR I=1,2,3
C                J=1,2,...,NUMTRI
C              - ADJACENT TRIANGLES LISTED IN ANTICLOCKWISE SEQUENCE
C              - ZERO DENOTES NO ADJACENT TRIANGLE
C
C     'NUMTRI' - NUMBER OF TRIANGLES IN FINAL TRIANGULATION
C              - NUMTRI LT 2*N+1
```

```
C
C     SUBROUTINES CALLED:
C     -------------------
C
C     PUSH
C
C     FUNCTIONS CALLED:
C     -----------------
C
C     TRILOC,EDG,SWAP,POP
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*****************************************************************************
C
      INTEGER V(3,*),N,I,T,LIST(*),NUMTRI,P,E(3,*),MAXSTK,TOPSTK,
     +        V1,V2,V3,L,R,POP,A,B,C,ERL,ERA,ERB,EDG,TRILOC,NUMPTS,
     +        TSTRT,TSTOP,STACK(*)
C
      REAL X(*),Y(*),XP,YP,C00000,C00100
C
      LOGICAL SWAP
C
      PARAMETER(C00000=0.0,
     +          C00100=100.0)
C
C     DEFINE VERTEX AND ADJACENCY LISTS FOR SUPERTRIANGLE
C
      V1=NUMPTS+1
      V2=NUMPTS+2
      V3=NUMPTS+3
      V(1,1)=V1
      V(2,1)=V2
      V(3,1)=V3
      E(1,1)=0
      E(2,1)=0
      E(3,1)=0
C
C     SET COORDS OF SUPERTRIANGLE
C
      X(V1)=-C00100
      X(V2)= C00100
      X(V3)= C00000
      Y(V1)=-C00100
      Y(V2)=-C00100
      Y(V3)= C00100
C
C     LOOP OVER EACH POINT
C
      NUMTRI=1
      TOPSTK=0
      MAXSTK=NUMPTS
      DO 100 I=1,N
        P=LIST(I)
        XP=X(P)
        YP=Y(P)
C
C       LOCATE TRIANGLE IN WHICH POINT LIES
C
        T=TRILOC(XP,YP,X,Y,V,E,NUMTRI)
C
C       CREATE NEW VERTEX AND ADJACENCY LISTS FOR TRIANGLE T
C
        A=E(1,T)
        B=E(2,T)
        C=E(3,T)
        V1=V(1,T)
        V2=V(2,T)
        V3=V(3,T)
        V(1,T)=P
        V(2,T)=V1
        V(3,T)=V2
        E(1,T)=NUMTRI+2
        E(2,T)=A
        E(3,T)=NUMTRI+1
C
C       CREATE NEW TRIANGLES
```

```
C
      NUMTRI=NUMTRI+1
      V(1,NUMTRI)=P
      V(2,NUMTRI)=V2
      V(3,NUMTRI)=V3
      E(1,NUMTRI)=T
      E(2,NUMTRI)=B
      E(3,NUMTRI)=NUMTRI+1
      NUMTRI=NUMTRI+1
      V(1,NUMTRI)=P
      V(2,NUMTRI)=V3
      V(3,NUMTRI)=V1
      E(1,NUMTRI)=NUMTRI-1
      E(2,NUMTRI)=C
      E(3,NUMTRI)=T
C
C     PUT EACH EDGE OF TRIANGLE T ON STACK
C     STORE TRIANGLES ON LEFT SIDE OF EACH EDGE
C     UPDATE ADJACENCY LISTS FOR ADJACENT TRIANGLES
C     ADJACENCY LIST FOR ELEMENT A DOES NOT NEED TO BE UPDATED
C
      IF(A.NE.0)THEN
        CALL PUSH(T,MAXSTK,TOPSTK,STACK)
      END IF
      IF(B.NE.0)THEN
        E(EDG(B,T,E),B)=NUMTRI-1
        CALL PUSH(NUMTRI-1,MAXSTK,TOPSTK,STACK)
      END IF
      IF(C.NE.0)THEN
        E(EDG(C,T,E),C)=NUMTRI
        CALL PUSH(NUMTRI,MAXSTK,TOPSTK,STACK)
      END IF
C
C     LOOP WHILE STACK IS NOT EMPTY
C
50    IF(TOPSTK.GT.0)THEN
        L=POP(TOPSTK,STACK)
        R=E(2,L)
C
C       CHECK IF NEW POINT IS IN CIRCUMCIRCLE FOR TRIANGLE R
C
        ERL=EDG(R,L,E)
        ERA=MOD(ERL,3)+1
        ERB=MOD(ERA,3)+1
        V1=V(ERL,R)
        V2=V(ERA,R)
        V3=V(ERB,R)
        IF(SWAP(X(V1),Y(V1),X(V2),Y(V2),X(V3),Y(V3),XP,YP))THEN
C
C         NEW POINT IS INSIDE CIRCUMCIRCLE FOR TRIANGLE R
C         SWAP DIAGONAL FOR CONVEX QUAD FORMED BY P-V2-V3-V1
C
          A=E(ERA,R)
          B=E(ERB,R)
          C=E(3,L)
C
C         UPDATE VERTEX AND ADJACENCY LIST FOR TRIANGLE L
C
          V(3,L)=V3
          E(2,L)=A
          E(3,L)=R
C
C         UPDATE VERTEX AND ADJACENCY LIST FOR TRIANGLE R
C
          V(1,R)=P
          V(2,R)=V3
          V(3,R)=V1
          E(1,R)=L
          E(2,R)=B
          E(3,R)=C
C
C         PUT EDGES L-A AND R-B ON STACK
C         UPDATE ADJACENCY LISTS FOR TRIANGLES A AND C
C
          IF(A.NE.0)THEN
            E(EDG(A,R,E),A)=L
            CALL PUSH(L,MAXSTK,TOPSTK,STACK)
          END IF
          IF(B.NE.0)THEN
            CALL PUSH(R,MAXSTK,TOPSTK,STACK)
          END IF
          IF(C.NE.0)THEN
            E(EDG(C,L,E),C)=R
          END IF
        END IF
```

```
          GOTO 50
        END IF
  100 CONTINUE
C
C     CHECK CONSISTENCY OF TRIANGULATION
C
      IF(NUMTRI.NE.2*N+1)THEN
        WRITE(6,'(''0***ERROR IN SUBROUTINE DELAUN***'')')
        WRITE(6,'('' ***INCORRECT NUMBER OF TRIANGLS FORMED***'')')
        STOP
      END IF
C
C     REMOVE ALL TRIANGLES CONTAINING SUPERTRIANGLE VERTICES
C     FIND FIRST TRIANGLE TO BE DELETED (TRIANGLE T)
C     UPDATE ADJACENCY LISTS FOR TRIANGLES ADJACENT TO T
C
      DO 120 T=1,NUMTRI
        IF((V(1,T).GT.NUMPTS).OR.
     +     (V(2,T).GT.NUMPTS).OR.
     +     (V(3,T).GT.NUMPTS))THEN
          DO 110 I=1,3
            A=E(I,T)
            IF(A.NE.0)THEN
              E(EDG(A,T,E),A)=0
            END IF
  110       CONTINUE
          GOTO 125
        END IF
  120 CONTINUE
  125 TSTRT=T+1
      TSTOP=NUMTRI
      NUMTRI=T-1
C
C     REMOVE TRIANGLES
C
      DO 200 T=TSTRT,TSTOP
        IF((V(1,T).GT.NUMPTS).OR.
     +     (V(2,T).GT.NUMPTS).OR.
     +     (V(3,T).GT.NUMPTS))THEN
C
C         TRIANGLE T IS TO BE DELETED
C         UPDATE ADJACENCY LISTS FOR TRIANGLES ADJACENT TO T
C
          DO 130 I=1,3
            A=E(I,T)
            IF(A.NE.0)THEN
              E(EDG(A,T,E),A)=0
            END IF
  130       CONTINUE
                              ELSE
C
C         TRIANGLE T IS NOT TO BE DELETED
C         PUT TRIANGLE T IN PLACE OF TRIANGLE NUMTRI
C         UPDATE ADJACENCY LISTS FOR TRIANGLES ADJACENT TO T
C
          NUMTRI=NUMTRI+1
          DO 140 I=1,3
            A=E(I,T)
            E(I,NUMTRI)=A
            V(I,NUMTRI)=V(I,T)
            IF(A.NE.0)THEN
              E(EDG(A,T,E),A)=NUMTRI
            END IF
  140       CONTINUE
        ENDIF
  200 CONTINUE
C
      END
C
C
C
C
      SUBROUTINE PUSH(ITEM,MAXSTK,TOPSTK,STACK)
C
C*********************************************************************
C
C     SUBROUTINE PUSH
C
C     PURPOSE:
C     --------
C
C     PLACE ITEM ON LIFO STACK AND INCREMENT STACK SIZE
C
C     INPUT:
C     ------
```

```
C
C     'ITEM'   - ITEM TO BE PLACED AT TOP OF LIFO STACK
C
C     'MAXSTK' - MAX SIZE OF STACK
C
C     'TOPSTK' - POINTER INDICATING CURRENT SIZE OF STACK
C              - MUST BE LT MAXSTK WHEN THIS ROUTINE IS CALLED
C
C     'STACK'  - LIFO STACK
C
C     OUTPUT:
C     -------
C
C     'ITEM'   - UNCHANGED
C
C     'MAXSTK' - UNCHANGED
C
C     'TOPSTK' - POINTER INDICATING CURRENT SIZE OF STACK
C              - NEW VALUE = OLD VALUE + 1
C
C     'STACK'  - LIFO STACK WITH ITEM ADDED
C              - STACK(TOPSTK)=ITEM
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C**********************************************************************
C
      INTEGER TOPSTK,MAXSTK,STACK(*),ITEM
C
      TOPSTK=TOPSTK+1
      IF(TOPSTK.GT.MAXSTK)THEN
        WRITE(6,'(''0***ERROR IN SUBROUTINE PUSH***'')')
        WRITE(6,'('' ***STACK OVERFLOW***'')')
        STOP
                        ELSE
        STACK(TOPSTK)=ITEM
      END IF
C
      END
C
C
C
C
      FUNCTION POP(TOPSTK,STACK)
C
C**********************************************************************
C
C     FUNCTION POP
C
C     PURPOSE:
C     --------
C
C     REMOVE ITEM FROM LIFO STACK AND DECREMENT STACK SIZE
C
C     INPUT:
C     ------
C
C     'TOPSTK' - POINTER INDICATING CURRENT SIZE OF STACK
C              - MUST BE GT ZERO WHEN THIS FUNCTION IS CALLED
C
C     'STACK'  - LIFO STACK
C
C     'POP'    - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'TOPSTK' - POINTER INDICATING SIZE OF STACK
C              - NEW VALUE = OLD VALUE - 1
C
C     'STACK'  - UNCHANGED
C
C     'POP'    - ITEM AT TOP OF STACK WHEN FUNCTION WAS CALLED
C
C     PROGRAMMER:
C     -----------
C
```

```
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*********************************************************************
C
      INTEGER POP,TOPSTK,STACK(*)
C
      IF(TOPSTK.GT.0)THEN
        POP=STACK(TOPSTK)
        TOPSTK=TOPSTK-1
                  ELSE
        WRITE(6,'(''0***ERROR IN FUNCTION POP***'')')
        WRITE(6,'('' ***STACK UNDERFLOW***'')')
        STOP
      END IF
C
      END
C
C
C
C
      FUNCTION EDG(L,K,E)
C
C*********************************************************************
C
C     FUNCTION EDG
C
C     PURPOSE:
C     --------
C
C     FIND EDGE IN TRIANGLE L WHICH IS ADJACENT TO TRIANGLE K
C
C     INPUT:
C     ------
C
C     'L'     - NUMBER OF TRIANGLE
C
C     'K'     - NUMBER OF ADJACENT TRIANGLE
C
C     'E'     - ADJACENCY ARRAY FOR TRIANGULATION
C             - TRIANGLES ADJACENT TO J ARE FOUND IN E(I,J) FOR I=1,2,3
C             - ADJACENT TRIANGLES LISTED IN ANTICLOCKWISE SEQUENCE
C             - ZERO DENOTES NO ADJACENT TRIANGLE
C             - E HAS DIMENSIONS E(3,2*N+1), WHERE N IS THE NUMBER OF
C               POINTS TO BE TRIANGULATED
C
C     'EDG'   - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'L'     - UNCHANGED
C
C     'K'     - UNCHANGED
C
C     'E'     - UNCHANGED
C
C     'EDG'   - NUMBER OF EDGE IN TRIANGLE L WHICH IS ADJACENT TO
C               TRIANGLE K
C             - E(EDG,L)=K
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*********************************************************************
C
      INTEGER L,K,I,E(3,*),EDG
C
      DO 10 I=1,3
        IF(E(I,L).EQ.K)THEN
          EDG=I
          RETURN
        END IF
   10 CONTINUE
```

```
      WRITE(6,'(''0***ERROR IN FUNCTION EDG***'')')
      WRITE(6,'('' ***ELEMENTS NOT ADJACENT***'')')
      STOP
C
      END
C
C
C
C
      FUNCTION TRILOC(XP,YP,X,Y,V,E,NUMTRI)
C*********************************************************************
C
C     FUNCTION TRILOC
C
C     PURPOSE:
C     --------
C
C     LOCATE TRIANGLE WHICH ENCLOSES POINT WITH COORDS (XP,YP) USING
C     LAWSON'S SEARCH
C
C     INPUT:
C     ------
C
C     'XP,YP'  - X-Y COORDINATES OF POINT
C
C     'X,Y'    - X-Y COORDINATES OF POINTS AND SUPERTRIANGLE VERTICES
C              - LISTS OF LENGTH NUMPTS+3
C              - LAST THREE LOCATIONS USED TO STORE COORDS OF
C                SUPERTRIANGLE
C
C     'V'      - VERTEX ARRAY FOR TRIANGULATION
C              - VERTICES LISTED IN ANTICLOCKWISE SEQUENCE
C              - VERTICES FOR TRIANGLE J ARE FOUND IN V(I,J) FOR I=1,2,3
C              - FIRST VERTEX IS AT POINT OF CONTACT OF FIRST AND THIRD
C                ADJACENT TRIANGLES
C              - V HAS DIMENSIONS V(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'E'      - ADJACENCY ARRAY FOR TRIANGULATION
C              - TRIANGLES ADJACENT TO J ARE FOUND IN E(I,J) FOR I=1,2,3
C              - ADJACENT TRIANGLES LISTED IN ANTICLOCKWISE SEQUENCE
C              - ZERO DENOTES NO ADJACENT TRIANGLE
C              - E HAS DIMENSIONS E(3,2*N+1), WHERE N IS THE NUMBER OF
C                POINTS TO BE TRIANGULATED
C
C     'NUMTRI' - NUMBER OF TRIANGLES IN TRIANGULATION
C
C     'TRILOC' - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'XP,YP'  - UNCHANGED
C
C     'X,Y'    - UNCHANGED
C
C     'V'      - UNCHANGED
C
C     'E'      - UNCHANGED
C
C     'NUMTRI' - UNCHANGED
C
C     'TRILOC' - NUMBER OF TRIANGLE CONTAINING POINT WITH COORDS (XP,YP)
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*********************************************************************
C
      INTEGER V(3,*),E(3,*),NUMTRI,V1,V2,I,T,TRILOC
C
      REAL X(*),Y(*),XP,YP
C
      T=NUMTRI
   10 CONTINUE
      DO 20 I=1,3
        V1=V(I,T)
        V2=V(MOD(I,3)+1,T)
```

```
        IF((Y(V1)-YP)*(X(V2)-XP).GT.(X(V1)-XP)*(Y(V2)-YP))THEN
          T=E(I,T)
          GOTO 10
        END IF
   20 CONTINUE
C
C     TRIANGLE HAS BEEN FOUND
C
      TRILOC=T
C
      END
C
C
C
C
      FUNCTION SWAP(X1,Y1,X2,Y2,X3,Y3,XP,YP)
C
C*************************************************************************
C
C     FUNCTION SWAP
C
C     PURPOSE:
C     --------
C
C     CHECK IF POINT WITH COORDS (XP,YP) LIES INSIDE THE CIRCUMCIRCLE
C     FOR THE TRIANGLE WITH COORDS (X1,Y1), (X2,Y2), (X3,Y3) USING
C     THE ALGORITHM OF CLINE AND RENKA WHICH ALLOWS FOR ROUNDOFF ERROR
C
C     INPUT:
C     ------
C
C     'X1,Y1'  - COORDS OF VERTICES DEFINING TRIANGLE
C     'X2,Y2'  - VERTICES LISTED IN ANTICLOCKWISE SEQUENCE AND ORDERED
C     'X3,Y3'    SUCH THAT P-V2-V3-V1 DEFINE A QUADRILATERAL
C
C     'XP,YP'  - COORDS OF POINT TO BE TESTED
C
C     'SWAP'   - NOT DEFINED
C
C     OUTPUT:
C     -------
C
C     'X1,Y1'  - UNCHANGED
C     'X2,Y2'
C     'X3,Y3'
C
C     'XP,YP'  - UNCHANGED
C
C     'SWAP'   - SET TO .TRUE. IF POINT LIES INSIDE CIRCUMCIRCLE
C              - SET TO .FALSE. IF POINT LIES ON OR OUTSIDE CIRCUMCIRCLE
C
C     PROGRAMMER:
C     -----------
C
C     S W SLOAN
C
C     LAST MODIFIED:
C     --------------
C
C     30 JAN 1986     S W SLOAN
C
C*************************************************************************
C
      REAL X1,Y1,X2,Y2,X3,Y3,XP,YP,X13,Y13,X23,Y23,X1P,Y1P,X2P,Y2P,COSA,
     +     COSB,SINA,SINB,C00000
C
      LOGICAL SWAP
C
      PARAMETER(C00000=0.0)
C
      X13=X1-X3
      Y13=Y1-Y3
      X23=X2-X3
      Y23=Y2-Y3
      X1P=X1-XP
      Y1P=Y1-YP
      X2P=X2-XP
      Y2P=Y2-YP
      COSA=X13*X23+Y13*Y23
      COSB=X2P*X1P+Y1P*Y2P
      IF((COSA.GE.C00000).AND.(COSB.GE.C00000))THEN
          SWAP=.FALSE.
        ELSEIF((COSA.LT.C00000).AND.(COSB.LT.C00000))THEN
            SWAP=.TRUE.
          ELSE
```

```
            SINA=X13*Y23-X23*Y13
            SINB=X2P*Y1P-X1P*Y2P
            IF((SINA*COSB+SINB*COSA).LT.C00000)THEN
              SWAP=.TRUE.
                                                ELSE
              SWAP=.FALSE.
            END IF
          ENDIF
C
          END
C
```