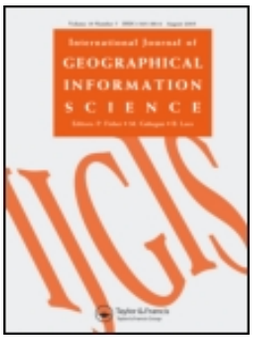


International Journal of Geographical Information Science

ISSN: 0269-3798 (Print) (Online) Journal homepage: www.tandfonline.com/journals/tgis19

Delaunay triangulations in TIN creation: an overview and a linear-time algorithm

Victor J. D. Tsai

INT. J. GEOGRAPHICAL INFORMATION SYSTEMS, 1993, VOL. 7, NO. 6, 501–524

Delaunay triangulations in TIN creation: an overview and a linear-time algorithm

VICTOR J. D. TSAI

Department of Civil and Environmental Engineering,
University of Wisconsin–Madison, Madison, Wisconsin 53706, U.S.A.

Abstract. The Delaunay triangulation is commonly used to generate *triangulated irregular network* (TIN) models for a best description of the surface morphology in a variety of applications in geographic information systems (GIS). This paper discusses the definitions and basic properties of the standard and constrained Delaunay triangulations. Several existing Delaunay algorithms are reviewed and classified into three categories according to their procedures: (1) *divide-and-conquer* methods, (2) *incremental insertion* methods, and (3) *triangulation growth* methods. Furthermore, a linear-time *Convex Hull Insertion* algorithm is presented to construct *TIN*s for a set of points as well as specific features such as constraint breaklines and exclusion boundaries. Empirical results over various sets of up to 50 000 points on personal computers show that the proposed algorithm efficiently expedites the construction of *TIN* models in approximately $O(N)$ for N randomly distributed points.

1. Introduction

In digital terrain modelling, the triangulated irregular network (*TIN*) model (Peucker *et al.* 1976) approximates a topographic surface by a set of contiguous triangular facets generated from a set of irregularly located points. One of the advantages of the *TIN* model is that it can describe a topographic surface at different levels of resolution as far as the information content is concerned. It has been recognized that the *TIN* model can accurately describe more complex surfaces and use less space and time than the grid cell data model of particular resolutions, e.g., the digital elevation model (*DEM*) (Mark 1975, McCullagh and Ross 1980, McCullagh 1988).

With the introduction of additional constraints, such as breaklines and exclusion boundaries, the *TIN* models can be extended to describe the surface morphology more adequately and reasonably than ever. Breaklines are linear features, such as ridges and valleys, used to define and control the behaviour of the surface in terms of continuity and smoothness. Similarly, exclusion boundaries explicitly define the regional boundaries of abrupt changes in surface behaviour. They may represent the internal boundaries like shoreline and buildings within the study area; they may be the external boundaries which define the extreme frame of the study area. Both breaklines and exclusion boundaries dominate the surface behaviour and are critical in describing the surface morphology using a triangulated model like *TIN*.

The *TIN* model has been widely used in a variety of applications in geographical information systems (GIS). Using *TIN* models, it is simpler to tackle the problems associated with spatial topology, automated contouring, two-and-half dimensional (2·5-D) visualization, surface drapes with other data, relief/hill shading, volumetric and cut-fill analysis, surface characterization and reconstructions, and site visibility analyses on terrain surfaces (Burrough 1986, Gold and Cormack 1987, Clarke 1990, Falcidieno and Spagnuolo 1991, Lee 1991). For example, attribute values at the very

0269–3798/93 $10·00

important points or the 'high information content' points on the landscape are measured and collected to generate a *TIN*-based terrain model. The intervening information and corresponding attributes are then interpolated on the single-valued functional surface, i.e., $z = f(x, y)$, which approximates data on these significant points. Terrain parameters such as slope, aspect, area and perimeter can be calculated for each triangular facet and stored as associated attributes of the facet. Moreover, thematic maps for certain discrete variables can be produced by overlaying and intersecting the *TIN* structure with the topological polygon structure (Burrough 1986).

The primary problem in *TIN* creation is how to allocate a unique set of triangles that can best describe the terrain and ensure bounded errors on the interpolation of bi-variate data at randomly sampled points (Clarke 1990). Among all possible triangulations, the Delaunay triangulation (Delaunay 1934) has all the salient properties for terrain fitting and is thus usually used in *TIN* creation. When non-crossing breaklines and exclusion boundaries are added into the problem as pre-specified constraints, however, a 'constrained' Delaunay triangulation has to be involved.

The remainder of this paper is organized as follows. Section 2 describes the definitions and basic properties of the standard and constrained planar Delaunay triangulations. Section 3 categorizes and briefly reviews several existing methods for standard Delaunay triangulations and the insertion of constraint segments. In §4, a linear-time *Convex Hull Insertion* algorithm is presented for creating the *TIN* models of a set of distinct points and pre-specified constraints. In §5, empirical results over various sets of up to 50000 points on personal computers indicate that the proposed *Convex Hull Insertion* algorithm expedites the construction of *TIN* models in expected linear time for randomly distributed points. Section 6 then concludes this paper with future extension of the Delaunay triangulation for three-dimensional (3-D) GIS data models. Completed in the appendix are the computations of the circumcircle of a triangle which is intensively involved in a Delaunay method.

2. Definitions and basic properties

2.1. *Standard (unconstrained) Delaunay triangulations*

The Delaunay triangulation is the straight-line dual of the Voronoi diagram (Voronoi 1908) and is constructed by connecting the points whose associated Voronoi polygons share a common edge. A Delaunay triangle is thus formed from three adjacent points whose associated Voronoi polygons meet at a common vertex, which is the centre of the circumscribed circle of the Delaunay triangle. For example, figure 1 depicts the duals of the Voronoi diagram and the Delaunay triangulation for a set of 16 points in the Euclidean plane. It is observed that the Delaunay triangulation follows the Euler–Poincaré theorem for connected planar graphs:

$$N_{regions} + N_{vertices} - N_{edges} = 2$$

The Delaunay triangulation of a set of points is an aggregate of connected but non-overlapping triangles such that the circumcircle of each triangle contains no other point in its interior. It is preferred as a straightforward definition of the Delaunay triangulation rather than the dual of the Voronoi diagram (Chew 1989). The property of 'the circumcircle of each triangle contains no other point in its interior' has been used as a criterion, called the *empty-circle criterion* and hereinafter the *Delaunay criterion*, to construct such triangulations for a set of distinct planar points. Figure 2 shows possible relationships between a new point to be added and an existing Delaunay triangle, and

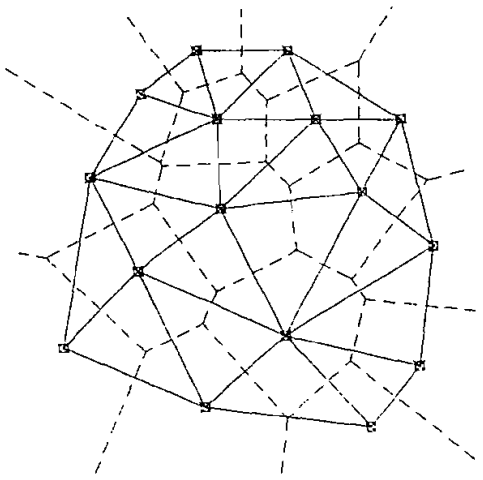


Figure 1. Delaunay triangulations (solid) and Voronoi diagrams (dashed) for a set of 16 planar points.

their outcomes after applying the *Delaunay criterion*. Note that in figure 2(*c*) the two ambiguous outcomes are both valid when the new point is on the circumcircle of a triangle; the shorter diagonal is desirable for a near optimal triangulation that minimizes the total edge length over all triangulations (Manacher and Zobrist 1979, Kirkpatrick 1980). Though the Delaunay triangulation does not approximate the optimal triangulation (Manacher and Zobrist 1979), it is close to optimal on the average (Lingas 1986). However, the Delaunay triangulation is unique whenever no more than three neighbouring points are co-circular in the Euclidean plane.

In plotting contours through triangular grids, Lawson (1972) suggested the *max–min angle criterion* to construct triangulations with local equiangularity: in every convex quadrilateral formed by two adjacent triangles, the swapping of diagonals does not increase the minimum of the six interior angles concerned. By the *max-min angle criterion*, the minimum measure of angles of all the triangles in the triangulation is maximized. Based on the *max–min angle criterion* Lawson (1977) also gave a local optimization procedure (*LOP*) to swap the diagonal of a convex quadrilateral, as already shown in figure 2(*b*), to achieve a most equiangular triangulation. Nevertheless, figure 3 illustrates the completion of the *LOP* swapping for a new point inserted into an existing triangulation until no diagonal swapping occurs. Both Lawson (1977) and Sibson (1978) have observed that the Delaunay triangulation ensures the *max–min angle criterion* and uniquely is locally equiangular.

2.2. *Constrained Delaunay triangulations*

The standard Delaunay triangulation can be naturally extended when a set of prescribed non-crossing breaklines and exclusion boundaries are forced in as part of the triangulation. Hence a 'constrained' Delaunay triangulation is introduced and is as close as possible to the standard Delaunay triangulation but integrates the constrained obstacles (Lee and Lin 1986, Bernal 1988, Chew 1989). Figure 4 gives a corresponding example to the standard Delaunay triangulation in figure 1 when two breaklines are forced to the original data set. Note that the triangles within the shaded shell are refined to include the breaklines.

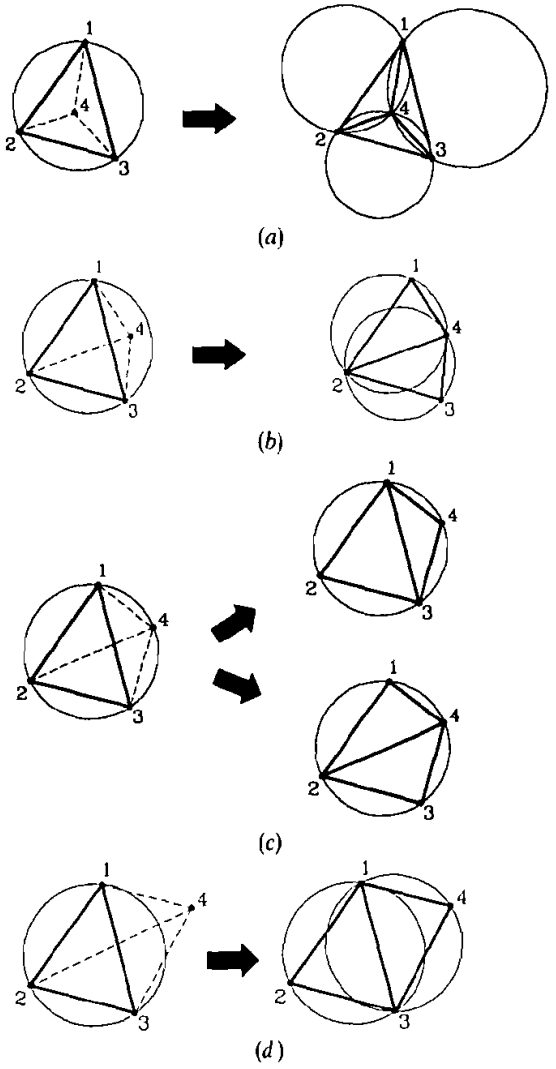


Figure 2. Possible relationships between a point and a triangle. (*a*) Inside the triangle; (*b*) inside the circumcircle; (*c*) on the circumcircle (diagonal $\overline{13}$ is desirable for minimum edge length); (*d*) outside the circumcircle.

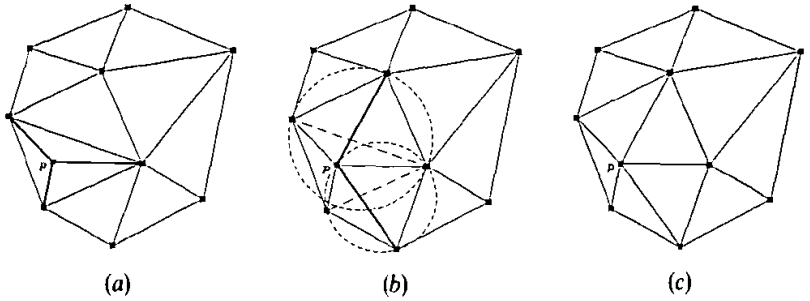


Figure 3. Completion of *Lawson's LOP* swapping. (*a*) Point insertion; (*b*) diagonal swapping; (*c*) the triangulation.

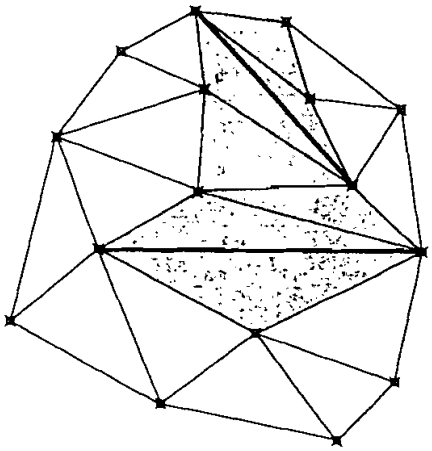


Figure 4. Delaunay triangulations (thin) for a set of 16 planar points and 2 breaklines (**bold**).

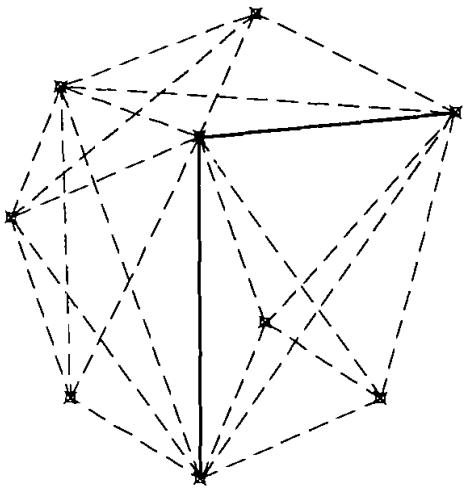
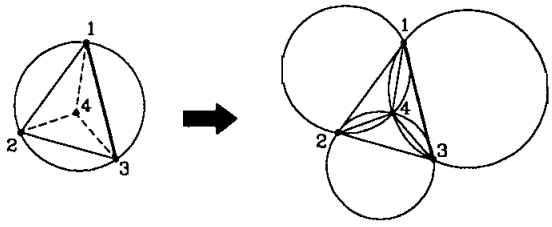


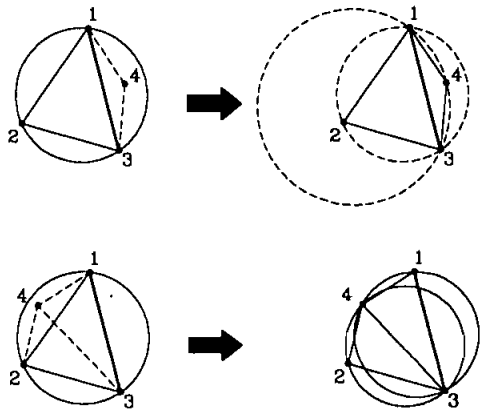
Figure 5. Visibility graph of a set of 9 points and 2 constraint segments.

Lee and Lin (1986) observed that the *Delaunay criterion* is still fulfilled and the local equiangularity property engaged by the constrained Delaunay triangulation with minor refinement. The visibility graph is helpful in redefining the *Delaunay criterion* and *Lawson's LOP* swapping for the constrained Delaunay triangulation when constraints are forced (Lee and Lin 1986, De Floriani and Puppo 1992). For a set of points and constraint breaklines, the visibility graph is constructed by connecting any two points which are mutually visible, i.e., the connecting line segment does not intersect any constraint breakline except at the end nodes as shown in figure 5. The *Delaunay criterion* and *Lawson's LOP* swapping can then be redefined as follows:

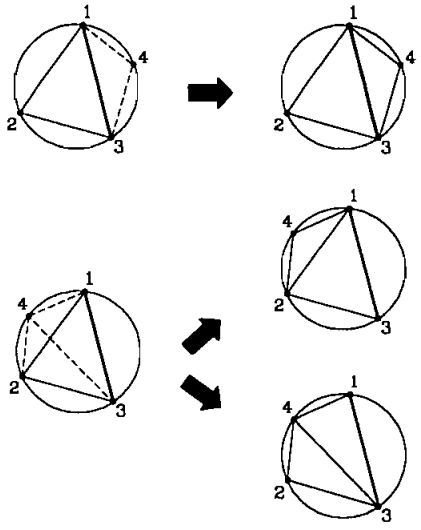
> The *constrained Delaunay criterion*: a triangle is a constrained Delaunay triangle if and only if there does not exist any other point inside its circumcircle and visible from all the three vertices of it.
>
> The *constrained Lawson's LOP*: a locally optimal diagonal of a convex quadrilateral formed by two adjacent triangles is chosen if and only if the *constrained Delaunay criterion* is satisfied.



(*a*)



(*b*)



(*c*)

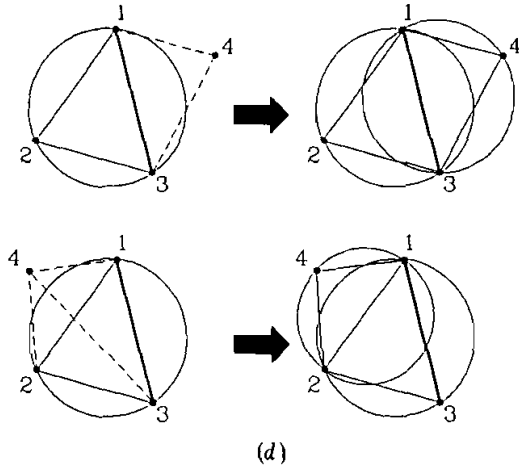


Figure 6. Possible relationships between a point and a triangle with constraint segment $\overline{13}$. (*a*) Inside the constrained triangle; (*b*) inside the circumcircle; (*c*) on the circumcircle (diagonal $\overline{12}$ is desirable for minimum edge length in the second example); (*d*) outside the circumcircle.

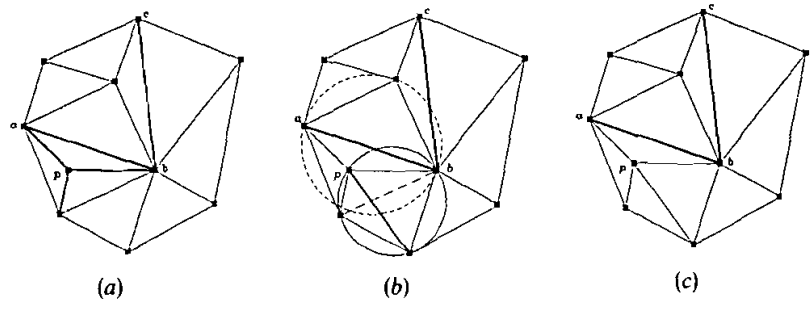


Figure 7. Completion of the *constrained Lawson's LOP* swapping in a triangulation with constraint segments $\overline{ab}$ and $\overline{bc}$. (*a*) Point insertion; (*b*) diagonal swapping; (*c*) triangulation.

In comparison with figure 2, figure 6 illustrates possible relationships between a new point and an existing constrained Delaunay triangle. Figure 7 gives an equivalent example to figure 3 for the *constrained Lawson's LOP* when a new point is added into existing constrained Delaunay triangulations. These criteria and *LOP*s, either standard or constrained, are intensively involved in the algorithms for Delaunay triangulations hereinafter.

3. Review of Delaunay methods

The computation of the Delaunay triangulations and Voronoi tessellations over a set of planar points has been intensively studied in the literature, and a variety of algorithms has been proposed for different applications. For a given set of $N \geqslant 3$ points, table 1 lists the expected run-time complexities for several of these algorithms. The run-time complexity of an algorithm depends on the number of points in the set, how the

Table 1. Run-time complexities for several Delaunay triangulation algorithms.

Algorithm	Average case	Worst case
Lawson (1977)[2]	$O(N^{4/3})$	$O(N^2)$
Green and Sibson (1978)[3]	$O(N^{3/2})$	$O(N^2)$
Lewis and Robinson (1978)[1]	$O(N \log N)$	$O(N^2)$
Brassel and Reif (1979)[3]	$O(N^{3/2})$	$O(N^2)$
McCullagh and Ross (1980)[3]	$O(N^{3/2})$	$O(N^2)$
Lee and Schachter (1) (1980)[1]	$O(N \log N)$	$O(N \log N)$
Lee and Schachter (2) (1980)[2]	$O(N^{3/2})$	$O(N^2)$
Bowyer (1981)[2]	$O(N^{3/2})$	$O(N^2)$
Watson (1981)[2]	$O(N^{3/2})$	$O(N^2)$
Mirante and Weigarten (1982)[3]	$O(N^{3/2})$	$O(N^2)$
Sloan (1987)[2]	$O(N^{5/4})$	$O(N^2)$
Dwyer (1987)[1]	$O(N \log \log N)$	$O(N \log N)$
Chew (1989)[1]	$O(N \log N)$	$O(N \log N)$
Macedonio and Pareschi (1991)[2]	$O(N^{3/2})$	$O(N^2)$

Notes: 1. *Divide-and-conquer* approach. 2. *Incremental insertion* approach. 3. *Triangulation growth* approach.

points are distributed, and the data structure and codes of implementation. It describes the upper bounds of the algorithm in the *is proportional to* notion which is often expressed in the *O-notation*, or *big-Oh notations*, defined as follows:

A function $g(N)$ is said to be $O(f(N))$ if there exist constants c_0 and N_0 such that $g(N)$ is less than $c_0 f(N)$ for all $N > N_0$. (Sedgewick 1988, p. 72)

According to the procedures in creating the triangulations, these algorithms and their derivatives can be classified into three categories: (1) the *divide-and-conquer* methods, (2) the *incremental insertion* methods, and (3) the *triangulation growth* methods. These methods are now briefly reviewed.

3.1. Divide-and-Conquer *algorithms*

Lewis and Robinson (1978) presented a triangulation method by applying a *problem reduction* technique to deal with bounded regions for contouring and finite element analysis applications. Lee and Schachter (1980) presented a *divide-and-conquer* algorithm for constructing Delaunay triangulations of a set of points in the plane; it is similar to Shamos and Hoey's (1975) primal Voronoi algorithm. The data set is sorted and divided into two disjoint subsets. After constructing a triangulation for each half, both triangulations are combined to form the final Delaunay triangulation. Dwyer (1987) improved Lee and Schachter's algorithm to the constrained case by dividing the data set into vertical strips, which are further subdivided into regions by crossing edges, and applying the *constrained Lawson's LOP* in diagonal swapping.

Given a set V of N distinct planar points, for example, a *divide-and-conquer* algorithm involves the following steps:

(1) Sort V in lexicographically ascending order such that $(x_i, y_i) < (x_{i+1}, y_{i+1})$ if and only if either $x_i < x_{i+1}$ or $x_i = x_{i+1}$ and $y_i < y_{i+1}$. This can be done by applying a recursive *conquer-and-divide* quick sort. Then recursively follow steps (2) to (5) until the final triangulation is constructed.

(2) Divide V into two equally sized halves V_L and V_R, such that V_L contains the first half of the points in the set and V_R contains the rest.
(3) Construct the Delaunay triangulations on each half. Apply *Lawson's LOP* to optimize the triangulation on each half.
(4) Compute the convex hull of each half. Find the lower and upper common tangents which are known to be in the final triangulations.
(5) Merge the two triangulations, starting with the lower common tangent of their convex hulls, zigzagging upward until their upper common tangent is reached. The merge procedure deletes non-Delaunay edges if one of its vertices is within the circumcircle of a triangle. Then it chooses correct Delaunay edges such that no other point is inside the circumcircle of a triangle by applying the *Delaunay criterion.*

Generally, in a *divide-and-conquer* approach the sorted data set is recursively divided down to subsets of equal size until the subset can be canonically treated. In the case of triangulations, a canonical subset contains at least 3 but less than 6 points so that the Delaunay triangulation can be easily constructed by using the first 3 points to form a triangle and adding the rest into it. As a result, the data set is divided like a balanced tree with all of its canonical subsets at the lowest leaf levels where local triangulations are formed. Then the triangulations are merged to form the final triangulation level by level through the tree to the root. Thus, the merge step is recursively proceeded in a bottom-up breadth-first way.

3.2. Incremental Insertion *algorithms*

The *incremental insertion* algorithms involve introducing non-processed points, one at a time, into existing Delaunay triangulations which are then refined. Most algorithms fall into this category with a little variation in data structures and the setting of the initial Delaunay triangulation, which encompasses all of the data points, as either a super triangle or a rectangle (Lawson 1977, Lee and Schachter 1980, Bowyer 1981, Watson 1981, Sloan 1987). For example, the framework of a *super triangle insertion* algorithm may involve the following steps:

(1) Define the vertices of a super triangle which contains all points and serves as the initial Delaunay triangle.
(2) Introduce a new point P from the set of points into existing triangulations.
(3) Find an existing triangle which encloses P, i.e., P is inside the triangle, and form three new triangles by connecting P to each of its vertices.
(4) Apply *Lawson's LOP* to update outwardly all triangles formed so far.
(5) Repeat steps (2) to (4) until all points are introduced.
(6) Remove all of the triangles that contain one or more vertices of the super triangle.

Instead of defining a super triangle, the iterative algorithm in Lee and Schachter (1980) triangulates a set of points within a rectangle whose vertices may be implicitly added. This algorithm was improved by Macedonio and Pareschi (1991) for triangulating points within a convex region. Given N points within a rectangle, the following procedures are involved:

(1) Remove all points which duplicate the vertices of the rectangle.
(2) Partition the rectangle into approximately $\sqrt{N}$ bins, or smaller rectangular regions.

(3) Sort the points by bins.
(4) Introduce the first point into the rectangle and connect the point to the four vertices of the rectangle to yield an initial triangulation.
(5) Introduce the next point into the existing triangulation and connect it to all the vertices of its enclosing triangle.
(6) Apply *Lawson's LOP* to outwardly update neighbouring triangles.
(7) Repeat steps (5) and (6) until all points are introduced.

3.3. Triangulation Growth *algorithms*

Green and Sibson (1978) implemented a recursive method for computing Dirichlet tessellations in the plane by scanning the points in turn, recursively modifying the contiguities of the Dirichlet polygons as each point is added where the Dirichlet polygons are growing around it. Brassel and Reif (1979) presented an algorithm, related to the work of Green and Sibson, for the subdivision of an area into Thiessen polygons (Thiessen 1911) by starting with an arbitrary point followed by finding its Thiessen neighbours in a *rotational search* over a one-dimensional sorted data list. The Delaunay triangulation is then created by connecting this point to all of its Thiessen neighbours and grows around until all points are involved in the triangulation. The fundamental procedures of a *triangulation growth* algorithm follow:

(1) Find the closest point to an arbitrary point in the set and connect the closest pair as a directed base line.
(2) Search for a third point lying in the right to the directed base line by applying the *Delaunay criterion.*
(3) Construct the Delaunay triangle and use the other two new directed edges (which are assigned as from the *from_node* of the base line to the third point, and then to the *to_node* of the base line,) as new base lines.
(4) Repeat steps (2) and (3) until all base lines are served.

Much of the algorithm is related to checking a considerable number of points to find the correct next neighbour for a given base line. For example, the *rotational search* is to calculate the centre and radius of the circumcircle. To minimize searching time for any given point, McCullagh and Ross (1980) gave a modified and improved algorithm by initially partitioning and sorting the data set into sorted box structures. An *expanding circle search* was proposed to determine the candidate neighbour of the base line, thus reducing calculation in finding the Delaunay triangulation. Maus (1984) gave a very similar algorithm to that proposed by McCullagh and Ross.

Alternatively, Mirante and Weingarten (1982) presented a *radial sweep* algorithm to construct the triangulations inward from the outer boundary of a set of randomly located points. The procedures used in the *radial sweep* algorithm include the following:

(1) Find the point which lies nearest to the centroid of the set as the starting point.
(2) Compute the distances and bearings from the central point to all other points in the set.
(3) Sort all other points in ascending order by bearing, distance and pitch.
(4) Radially sweep and form the radiating triangles by connecting the central point to all other points and connecting any two consecutive points. Points with the same bearing are used to form a pair of triangles on each side of the common line.

(5) Fill in the concavity by connecting the points that are on the boundary of the radiating triangles, thus forming new triangles outward until the outer boundary of the set is reached.
(6) Successively optimize the triangulations inward from the triangles on the outer boundary by checking the lengths of the two diagonals of a quadrilateral formed by a pair of triangles (similar to *Lawson's LOP*).

3.4. *Incremental insertion of constraint segments*

Once the standard Delaunay triangulation of the set of points is constructed using any one of the aforementioned algorithms, pre-specified constraint segments can then be inserted to complete a constrained Delaunay triangulation. As shown in figure 8, the following steps are further involved to handle constraint segments (Bernal 1988, De Floriani and Puppo 1992):

(1) Insert a new constraint segment into the triangulation.
(2) Identify those triangles whose interiors intersect the constraint segment and remove all internal edges shared by two such triangles, forming the *influence polygon* of the constraint segment. (Note: The *influence polygon* described here is different from the *area-of-influence polygon* (Hayes and Koch 1984), i.e., the *Voronoi* or *Thiessen polygon* of a point.)
(3) Connect all other vertices of the *influence polygon* to the *from_node* of the constraint segment.
(4) Optimize the triangulation within the *influence polygon* by applying the *constrained Lawson's LOP*, forcing the constraint segment as a known edge in the triangulation.
(5) Repeat steps (1) to (4) until all constraint segments are inserted into the triangulation.

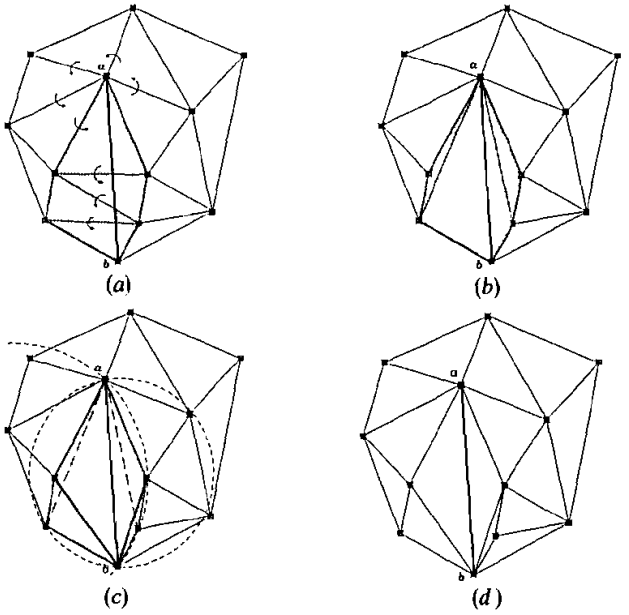


Figure 8. Insertion of the constraint segment $\overline{ab}$ into existing Delaunay triangulations. (*a*) Insert segment $\overline{ab}$, find its *influence polygon* (shaded); (*b*) connect all vertices of the *influence polygon* to node '*a*'; (*c*) optimize triangulations by *constrained Lawson's LOP* swapping; (*d*) refined triangulations with constraint segment $\overline{ab}$.

4. A complete *Convex Hull Insertion* algorithm

Several of the existing algorithms for Delaunay triangulation and their derivatives have been reviewed above in three categories as well as the insertion of constraint segments. However, none of these algorithms provides boundary clipping when constraint exclusion boundaries are involved in the application. Hence, these algorithms are incomplete for *TIN* creation problems with constraint boundaries. Boundary clipping, or polygon clipping, is necessary and critical especially to contouring applications in which interpolations should be excluded from the exclusion regions. Presented next is a complete algorithm which provides capabilities of both triangulation construction and boundary clipping for a set of planar points and constraints.

Tsai and Vonderohe (1991) presented a generalized algorithm, the *Convex Hull Insertion* algorithm, for the construction of Delaunay triangulations in the n-dimensional Euclidean space. Here the *Convex Hull Insertion* algorithm is further improved and extended to construct the Delaunay triangulation of a set of points with pre-specified constraints. The improvement speeds the convex hull computation and the incremental insertion of points and constraint segments by partitioning the data set into cells and maintaining spatial primitives in topological data structures. Extended sub-algorithms for inserting constraint segments and boundary clipping are also presented, thus completing the solution for *TIN* creation problems.

For a set of $N \geqslant 3$ distinct planar points and M constraint segments in random distribution, the proposed algorithm completes the *TIN* construction in expected linear time, i.e. it runs in approximately $O(N)$. However, it runs in $O(N^2)$ for the worst possible case. In this algorithm, both breaklines and exclusion boundaries are considered as constraint segments and their vertices are included in the set of points. The proposed *Convex Hull Insertion* algorithm involves the following phases:

(1) Partition the point set into N/k cells, i.e., $\sqrt{N/k}$ equidistant rows and columns, where k is the selected average number of points per cell (default $k=4$).
(2) Find the *Convex Hull* of the partitioned set.
(3) Construct the Delaunay triangulation of the *Convex Hull* by applying *Delaunay criterion.*
(4) Iteratively insert other points, which are not on the *Convex Hull*, and refine the existing Delaunay triangulations.
(5) Iteratively insert new constraint segments and refine existing Delaunay triangulations.
(6) Remove all the triangles that are within the internal exclusion boundaries or outside the external exclusion boundaries.

4.1. *Initial data partitioning*

Partitioning the set of points into cells has been increasingly used in geometric algorithms for Voronoi tessellation and Delaunay triangulation. For examples, see Shamos and Hoey (1975), Lee and Schachter (1980), McCullagh and Ross (1980), Maus (1984), Dwyer (1987), Chew (1989) and Macedonio and Pareschi (1991). Generally, the geographical partitioning process is a two-dimensional sort that enables fast access to points lying in proximity to others, thus improving the expected run time of the algorithm. However, sorting the points is lavish and unnecessary when data points should exist in the same order in which they are to be accessed. For example, in *TIN* models the vertices of constraint breaklines, exclusion boundaries, lines and triangles

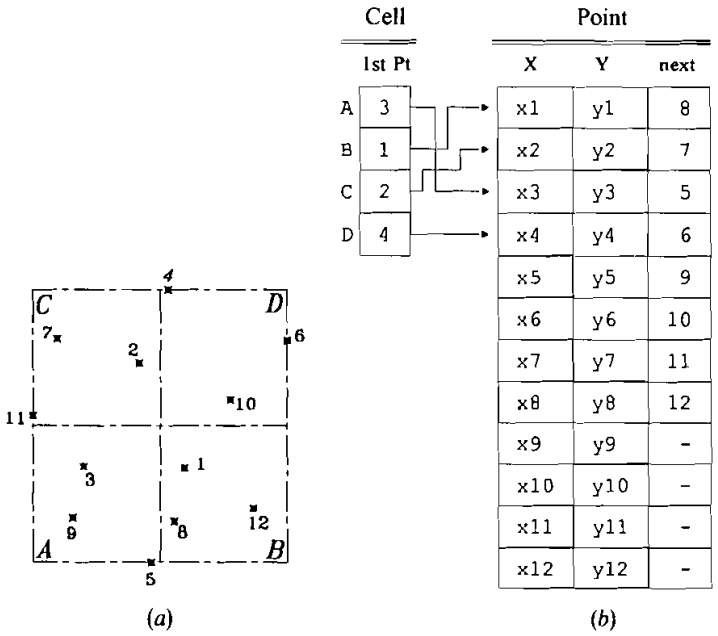


Figure 9. Cell partitioning on 12 randomly located points. (*a*) Partitioned cells; (*b*) cell and point structures.

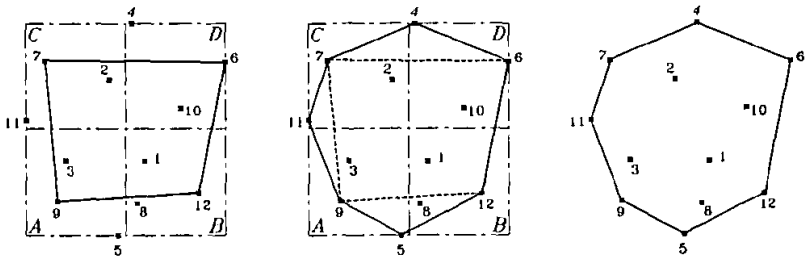
are often indexed to the points which retain the coordinates. Thus it is preferred in run-time programmes to leave the points unsorted such that these indices point to correct points while partitioning them into cells.

A faster alternative is to create a one-dimensional array to store the index of the first point in each cell, then store along with each point the index of next point lying in the same cell (Larkin 1991). Figure 9 shows the partitioning cell structures of a simplified data set containing 12 points. The partitioning cell structures not only accelerate the computation of the *Convex Hull*, but also expedite the insertions of points and constraint segments. The larger the data set, the greater the saving of time. Partitioning points in this manner runs in linear time, or $O(N)$, for a uniform distribution of points, but in $O(N^2)$ for the worst possible case (Larkin 1991). Similar data structures are also used to store the centroids of triangles for quick search of triangles whose circumcircles enclose the inserted point or which intersect a constraint segment.

4.2. Convex Hull *computation*

The *Convex Hull* of a set of planar points is defined to be the smallest convex polygon containing them all and has the property that any line connecting two points inside the polygon must lie entirely inside the polygon (Sedgewick 1988). It is the natural extreme boundary of the point set and is equivalent to the shortest path surrounding the points. Obviously, the *Convex Hull* is part of the standard Delaunay triangulation of the set.

By partitioning the set of points into cell structures, the *Convex Hull* of the set can be found in expected linear time approximately proportional to N, i.e., $O(N)$, for a uniform set and $O(N^2)$ for the worst possible case (Maus 1984, Larkin 1991). Based on



(a) (b) (c)

Figure 10. Completion of the *Convex Hull* computation. (*a*) Initial boundaries 7, 9, 12 and 6; (*b*) find hull vertices 11, 5 and 4; (*c*) the convex hull.

the algorithm improved by Larkin (1991), the following procedures are involved in computing the *Convex Hull* of the partitioned set as depicted in figure 10.

(A) Find the points with the minimum $x-y$, $x+y$, and maximum $x-y$, and $x+y$ values respectively. These points are all on the *Convex Hull* perimeter and always lie near the four corners of the set, e.g., points 7, 9, 12, and 6 in figure 10(*a*).

(B) Store these points in counterclockwise order in a linked circular list and remove any redundancy.

(C) For each point **I** and its subsequent point **J** in the list call the recursive sub-algorithm CONVEX(**I**, **J**) to find all the points on the hull to the right of the line segment $\vec{IJ}$.

The recursive sub-algorithm CONVEX(**I**, **J**)

(D) Examines all points lying in the cells that are intersected by or to the right of the line segment $\vec{IJ}$, and find the point **K** with the largest offset from $\vec{IJ}$, where points to the right of $\vec{IJ}$ are assigned positive offsets and those to the left negative ones.

(E) Tests the sign of the largest offset:

(i) if it is positive, insert **K** into the list between points **I** and **J**, and call CONVEX(**I**, **K**) and CONVEX(**K**, **J**).

(ii) if it is zero and **K** lies between **I** and **J**, insert point **K** into the list between points **I** and **J**, and call CONVEX(**I**, **K**) and CONVEX(**K**, **J**).

(iii) else terminate this call to CONVEX.

4.3. Convex Hull *triangulations*

The Delaunay triangulation of the *Convex Hull* itself can be found in a similar approach to the *triangulation growth* alogorithm described in §3.3. Given in counterclockwise order the vertices of the *Convex Hull*, e.g., m for them which is usually far less than N for a random point set, the following procedures compute the Delaunay triangulation of the *Convex Hull* in $O(m^2)$ as shown in figure 11:

(A) Store the first edge of the *convex hull* in a list of base edges which is initialized empty.

(B) For the next base edge, check the vertices lying to its left to find the third point by applying the *Delaunay criterion.*

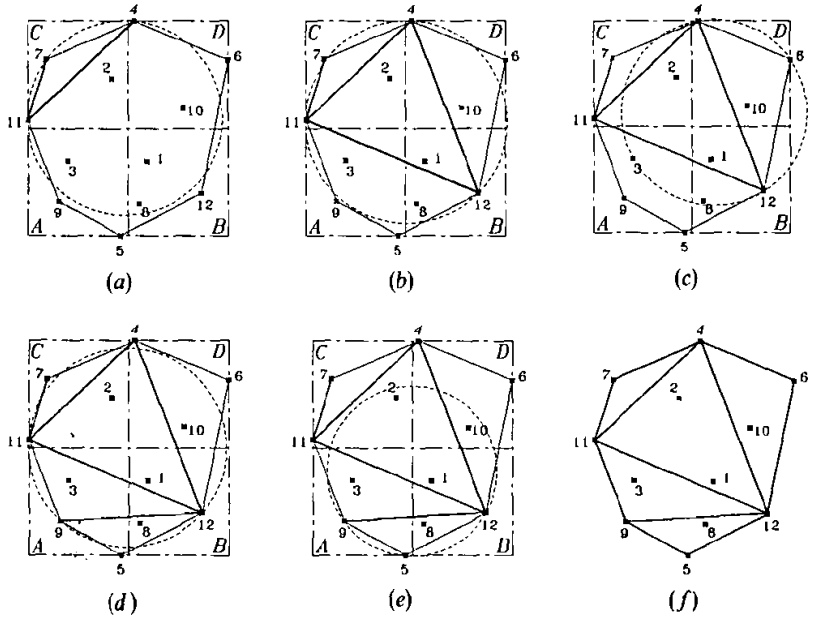


Figure 11. Completion of the *Convex Hull* triangulation. (*a*) Base edge $\overline{7,11}$; (*b*) base edge $\overline{4,11}$; (*c*) base edge $\overline{4,12}$; (*d*) base edge $\overline{12,11}$; (*e*) base edge $\overline{12,9}$; (*f*) the triangulation.

(C) Form the Delaunay triangle and append all internal edges to the list of base edges where new edges of the triangle are assigned as from the *from_node* of the base edge to the third point, and then to the *to_node* of the base edge.
(D) Update the adjacent topologies of edges and triangles in the topological data structures, which are described later in §5 for practical implementations.
(E) Repeat steps (B) to (D) to create all Delaunay triangles until all base edges are served.

4.4. *Insertion of other points*

All other points within the *Convex Hull* can be added iteratively into the initial Delaunay triangulation of the *Convex Hull*. By finding the *influence triangulation* of the added point, existing triangulations are then locally refined. For a randomly distributed point set, the following procedures update the triangulation in expected linear time as illustrated in figure 12:

(A) Find all of the triangles whose circumcircles enclose the new point by applying *Delaunay criterion*, forming the *influence triangulation* of the new point. Finding such triangles is expedited through the use of a partitioning cell structure for the centroids of triangles, and the topological data structures that keep the adjacent relationships of edges and triangles. That is, find the first candidate triangle through the partitioning cell structure, and then find the rest outwardly by investigating its neighbouring triangles through the topological data structures.
(B) Delete all internal edges shared by two adjacent triangles in the *influence triangulation* of the point.

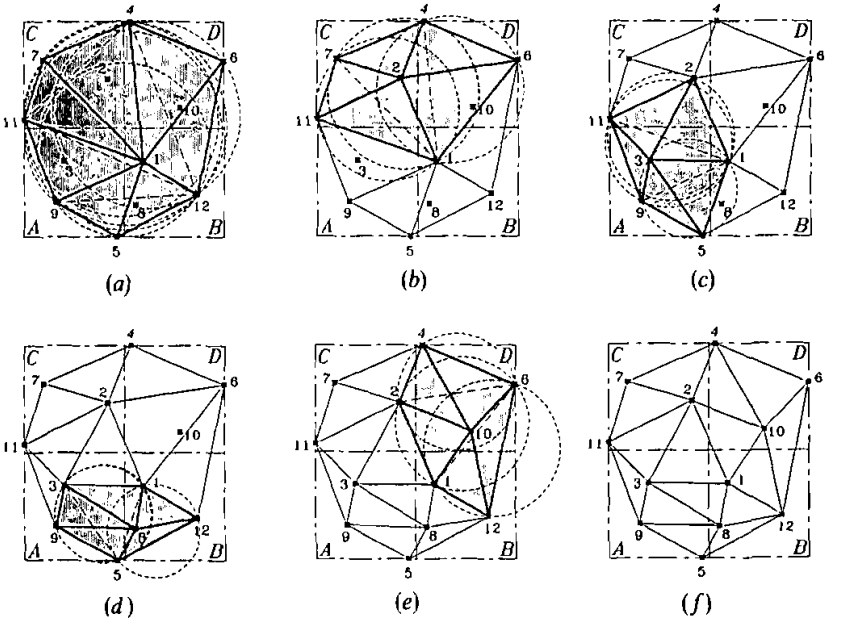


Figure 12. Insertion of other points into the existing triangulations (shaded region shows the *influence triangulation* of the new point). (*a*) Insert point 1; (*b*) insert point 2; (*c*) insert point 3; (*d*) insert point 8; (*e*) insert point 10; (*f*) the triangulation.

(C) Construct new edges and triangles by connecting the new point to the vertices of its *influence triangulation.* Note that *Lawson's LOP* is implicitly and automatically involved in this way.
(D) Update the adjacent topologies of edges and triangles which are within or adjacent to the refined *influence triangulation* of the new point.
(E) Repeat steps (A) to (D) until all other points within the *Convex Hull* are added.

4.5. *Insertion of constraint segments*

Once the Delaunay triangulation of the set of points has been constructed, constraint segments can be inserted into the triangulation to enforce breaklines and boundaries within the *TIN* model. As with the insertion of points, existing Delaunay triangulations are updated locally and densified by finding the *intersecting triangulation* and intersecting points of the constraint segment. Intersecting points are added into corresponding *intersecting triangulation* of a constraint segment as *virtual points* since their coordinates and attribute values are interpolated from the constraint segment itself. Figure 13 shows the following procedures for the insertion of constraint segments:

(A) Identify all of the triangles whose interiors intersect the constraint segment, computing the intersecting points and forming the *intersecting triangulation* of the segment.
(B) Add the intersecting points into the list of *virtual points.*
(C) Refine the *intersecting triangulation* of the segment with exactly the same procedures in the point insertion as described above by adding all *virtual points* into it. Note that only those triangles in the *intersecting triangulation* are refined.

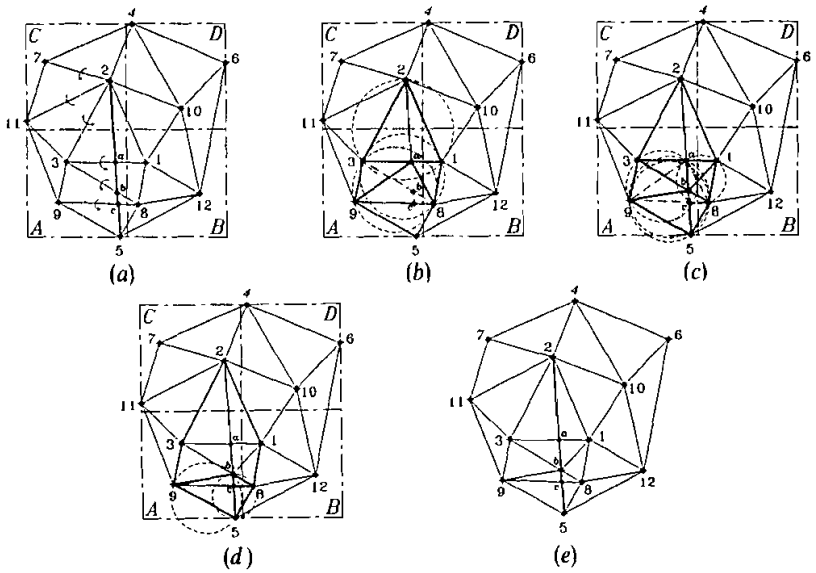


Figure 13. Insertion of the constraint segment $\overline{25}$ in the *Convex Hull Insertion* algorithm (shaded region shows the *influence triangulation* of the intersecting point). (*a*) Insert segment $\overline{25}$, find its *intersecting triangulation* and intersecting points *a*, *b*, and *c*; (*b*) insert point *a* into the *intersecting triangulation*; (*c*) insert point *b* into the *intersecting triangulation*; (*d*) insert point *c* into the *intersecting triangulation*; (*e*) refine triangulations with the constraint segment $\overline{25}$ as $\overline{2a}$, $\overline{ab}$, $\overline{bc}$, and $\overline{c5}$.

(D) Update the constraint segment by a series of short segments from the *from_node*, passing all of its intersecting *virtual points*, to the *to_node* of the constraint segment.

(E) Repeat steps (A) to (D) until all constraint segments are added.

4.6. *Exclusion boundary clippings*

Exclusion boundaries include both internal and external areas. Internal exclusion boundaries may define regions within which interpolations should be omitted in a contouring application. Similarly, external exclusion boundaries define the extreme frame outside which interpolations and extrapolations are incorrect, uninteresting, or invalid. Thus, all triangles within the internal exclusion regions and outside the external frame should be removed from the constrained Delaunay triangulation.

4.6.1. *Internal boundary clipping*

Since the Delaunay triangulation is a connected planar graph, it is triggered to remove all triangles within an internal boundary by finding one of them and travelling through its neighbours. For all internal exclusion boundaries in the set, the following procedures tackle internal boundary clipping as illustrated in figure 14:

(A) Store the first edge of an internal exclusion boundary in a list of deleting edges which is initialized empty.

(B) For the next deleting edge in the list, delete the triangle lying on its left and append all other non-boundary edges of the deleted triangle to the list of deleting edges.

(C) Update the adjacent topologies of the boundary edges and the triangles which are adjacent to the internal exclusion boundary.

(D) Repeat steps (B) and (C) until all deleting edges are served.

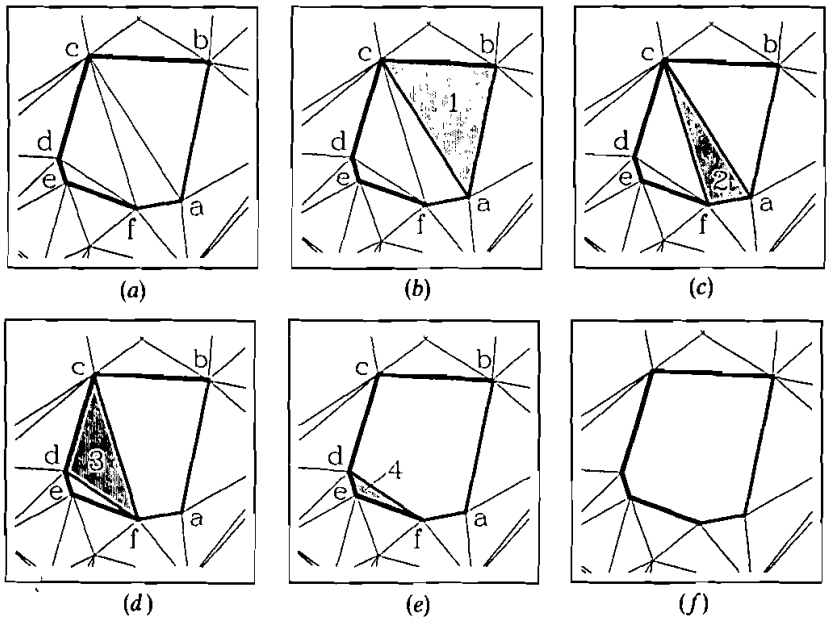


Figure 14. Completion of the internal boundary clipping. (*a*) Internal boundary; (*b*) *deleting edge* $\overline{ab}$, delete triangle 1; (*c*) *deleting edge* $\overline{ac}$, delete triangle 2; (*d*) *deleting edge* $\overline{fc}$, delete triangle 3; (*e*) *deleting edge* $\overline{fd}$, delete triangle 4; (*f*) clipped boundary.

4.6.2. *External boundary clipping*

As shown in figure 15, external boundary clipping similarly involves the following procedures:

(A) Store all edges of the Convex Hull that are not edges on external boundaries in a list of deleting edges which is initialized empty.
(B) For the next deleting edge in the list, delete the triangle lying on its left and append all other non-boundary edges of the deleted triangle to the list of deleting edges.
(C) Update the adjacent topologies of the boundary edges and the triangles which are adjacent to the external exclusion boundary.
(D) Repeat steps (B) and (C) until all deleting edges are served.

5. Implementation

The *Convex Hull Insertion* algorithm was implemented on personal computers using the *C++* computer programming language. Known features of the *C++* programming language, such as dynamic memory allocation and class objects, are fully utilized in the implementation to keep the capability and flexibility in handling arbitrary collections of points and constraint boundaries. For example, as shown in figure 16, a *C*-like description of the topological data structures implemented in the program encodes the *TIN* model as a set of topological relationships between points, edges and triangles. These data are all maintained in the extended memory blocks (above 1 megabyte) regardless of their sizes and dimensions, leaving the conventional memory area (below 640 kilobyte) for run-time dynamic buffering and general computations. Thus, the number of points and constraints that can be handled in the program is limited by the amount of memory available in a personal computer.

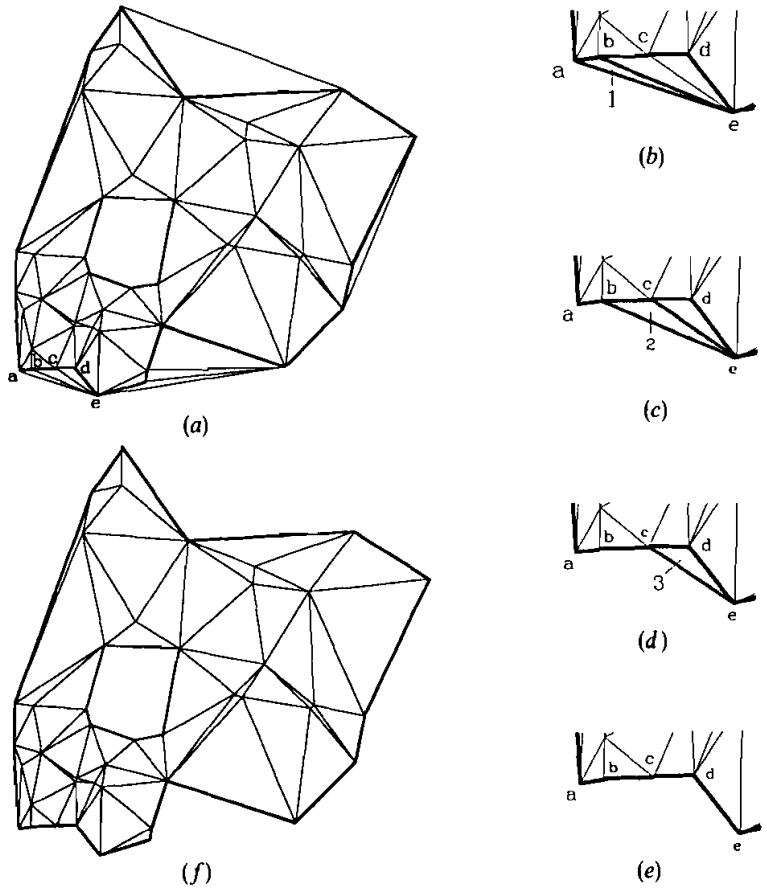


Figure 15. Completion of the external boundary clipping. (*a*) External boundaries without clipping; (*b*) *deleting edge* $\overline{ae}$, delete triangle 1; (*c*) *deleting edge* $\overline{be}$, delete triangle 2; (*d*) *deleting edge* $\overline{ce}$, delete triangle 3; (*e*) clipped external boundary $\overline{abcde}$; (*f*) triangulations after *internal* and *external boundary clippings*.

```
/*********************************************
 * Topological data structures for TIN models *
 *********************************************/

struct POINT {              /* Mass data points    */
     float  xy[2];          /* X & Y coordinates   */
     long   next;           /* Next POINT in cell  */
};

struct EDGE {               /* Edges in TINs           */
     long vt[2];            /* From_node and To_Node   */
     long lr[2];            /* Left and Right TINs     */
};

struct TIN {                /* Delaunay triangles      */
     long  ed[3];           /* Bounded EDGEs' indices  */
     long  at[3];           /* Adjacent TINs' indices  */
     float xc[2];           /* X & Y of circumcenter   */
     float r2;              /* Square of circumradius  */
     long  next;            /* Next TIN in this cell   */
};
```

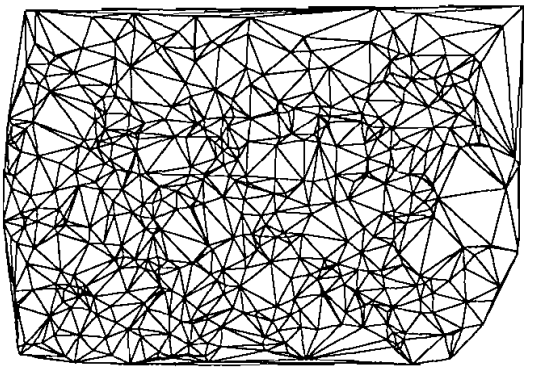
Figure 16. Topological data structures for *TIN* models.

For various distributions and sizes of data sets up to 50000 points, table 2 shows the execution times in seconds used by each phase in the *Convex Hull Insertion* program, running on a Hewlett Packard Vectra 486/66U personal computer. The execution time required for the program depends on the distribution and size of the point set, and the operating environment (including the central processing unit (CPU), the basic input/output system (BIOS), the disk operating system (DOS), memory and memory manager, etc.) under which the program runs. Although it may take $O(N^2)$ time in the worst case, the statistics timing in table 2 indicates that the *Convex Hull Insertion* algorithm constructs the triangulations and associated topologies for *TIN* models in approximately $O(N^{1.035})$ for randomly distributed points and $O(N^{1.277})$ for regular *DEM* grids. Meanwhile, run-time complexities of individual phases are also computed in table 2 to show the efficiency of the *Convex Hull Insertion* algorithm. To verify the completeness and authenticity of the algorithm, figure 17 shows the resulting triangulations of 500 randomly spaced points and a 20 by 25 regular *DEM* grids, respectively.

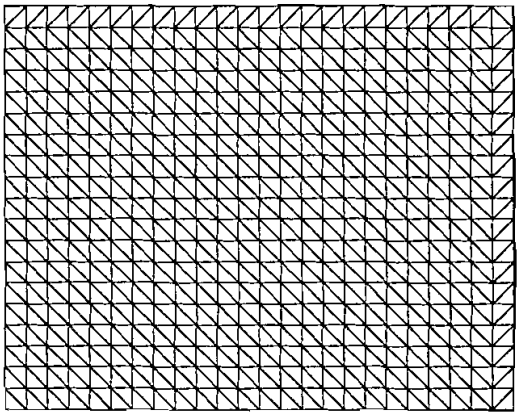
Table 2. Execution times[1] of the *Convex Hull Insertion* algorithm for constructing Delaunay triangulations of various sets of points.

	Number of Points (N)	Initial Data Partitioning (1)	*Convex Hull* Computation (2)	*Convex Hull* Triangulation (3)	Other Points Insertion (4)	Total (1)+(2)+(3)+(4)
REGULAR GRIDS	25 × 40	0·077	0·303	0·225 (126)[2]	11·639	12·244
	40 × 50	0·145	0·647	0·375 (176)	28·649	29·817
	50 × 60	0·218	0·984	0·523 (216)	47·983	49·708
	50 × 80	0·299	1·179	0·721 (256)	67·483	69·682
	50 × 100	0·372	1·417	0·952 (296)	87·263	90·003
	100 × 100	0·693	2·923	1·448 (396)	223·635	228·698
	100 × 200	1·485	5·582	3·398 (596)	509·951	520·416
	150 × 200	2·198	9·283	4·282 (696)	902·571	918·333
	200 × 200	2·776	11·652	5·241 (796)	1333·232	1352·901
	200 × 250	3·641	15·912	6·795 (896)	1760·529	1786·878
	$O(N^a)^3$	0·978	1·022	1·756	1·246	1·277
IRREGULAR SETS	1000	0·085	0·037	0·014 (16)[2]	8·203	8·339
	2000	0·171	0·062	0·017 (20)	16·497	16·747
	3000	0·266	0·083	0·014 (17)	25·485	25·848
	4000	0·369	0·110	0·014 (16)	34·562	35·055
	5000	0·448	0·129	0·016 (19)	42·880	43·473
	10000	1·059	0·251	0·022 (26)	88·429	89·761
	20000	2·742	0·492	0·016 (20)	195·180	198·430
	30000	3·473	0·781	0·027 (26)	278·162	282·443
	40000	3·590	0·978	0·029 (31)	348·396	352·993
	50000	4·519	1·195	0·033 (32)	438·788	444·535
	$O(N^a)^3$	1·088	0·893	1·153	1·031	1·035

Notes: 1. CPU time in seconds, excluding data input/output, for a Hewlett Packard Vectra 486/66U personal computer (Intel 80486 DX2/66Mhz with 8 Mb RAM, running under MS-DOS 5·0 and QEMM 386 6·0). 2. Number of points found on the convex hull of the set in convex hull computations. 3. Run-time complexity, assuming $t = c_0 f(N) = c_0 N^a$, where t is the CPU time used, N is the number of points in the computation, c_0 is a constant, and 'a' values obtained by least-squares fitting.



(*a*)



(*b*)

Figure 17. Triangulated irregular networks (*TIN*s) of irregular and regular point sets. (*a*) *TIN*s of 500 randomly located points; (*b*) *TIN*s of a 20×25 regular DEM grids.

6. Conclusions

Owing to its advantages in describing complex surfaces, the triangulated irregular network (*TIN*) model has been widely used in diverse applications in automated mapping, terrain surface modelling and analysis, and GIS. Since the Delaunay triangulation yields non-overlapping triangles as equiangular as possible, it is usually used to solve the problem of allocating points into *TIN* models. Meanwhile, the Delaunay triangulation has all the prominent properties for an optimal *TIN* model which provides multi-resolutional description of the terrain truth and ensures bounded errors on interpolation of attributes within the model.

In this paper, definitions and basic properties of both standard and constrained Delaunay triangulations on the plane are discussed. In addition to providing brief review and categorization of existing Delaunay methods, this paper presents a complete *Convex Hull Insertion* algorithm to construct *TIN* models for a set of planar points and constraints in expected linear time. Practical implementation and results from various sets of data up to 50000 points show that the proposed *Convex Hull Insertion* algorithm efficiently expedites the creation of *TIN* models for regular *DEM* grids and irregularly spaced points in approximately $O(N^{5/4})$ and $O(N)$, respectively.

Another advantage of the *Convex Hull Insertion* algorithm is that it is extendable to higher dimensions in which more complicated constraints and topological relationships may exist. For example, 3-D Delaunay tetrahedral tessellations are useful to 3-D or volumetric GIS applications in which values of attributes at (x, y, z) locations need to be maintained and analysed in true 3-D geometry. The 3-D Delaunay tetrahedral tessellation would minimize the largest containment radius of the tetrahedron, resulting in the most compact tessellation of a set of points in the 3-D space. While *TIN* models allow interpolations on surfaces only and 2·5-D views based on single-value attributes at (x, y) locations, 3-D Delaunay tetrahedral tessellations permit additional true volumetric interpolations and operations on attribute values at (x, y, z) locations, i.e., $U = f(x, y, z)$. With the extended *Convex Hull Insertion* algorithm, the construction of 3-D Delaunay tetrahedral tessellations can be fostered and anticipated for a true 3-D GIS data model.

Acknowledgments

The author gratefully expresses his appreciation to Professors Alan Vonderohe, Lynn Usery, and John Uicker of the University of Wisconsin–Madison for their constructive remarks in discussing these algorithms. The author also thanks the reviewers for their useful comments on earlier drafts of this paper. These remarks and comments have been very helpful in improving this work.

Appendix: Circumcircle computations

For three non-collinear points, $1=(x_1, y_1)$, $2=(x_2, y_2)$, and $3=(x_3, y_3)$ in the plane, the derivation of the circumcenter $C=(x_c, y_c)$ and the circumradius r of their circumcircle follows (Tsai and Vonderohe 1991).

By definition of the circumcircle of three distinct points, the following equations are solved for x_c, y_c, and r:

$$(x_c-x_1)^2+(y_c-y_1)^2=r^2 \tag{1}$$

$$(x_c-x_2)^2+(y_c-y_2)^2=r^2 \tag{2}$$

$$(x_c-x_3)^2+(y_c-y_3)^2=r^2 \tag{3}$$

Let $\mathbf{S}=C-1=(x_s, y_s)=(x_c-x_1, y_c-y_1)$, then

$$\begin{bmatrix}(x_2-x_1) & (y_2-y_1)\\ (x_3-x_1) & (y_3-y_1)\end{bmatrix}\begin{bmatrix}x_s\\ y_s\end{bmatrix}=0{\cdot}5\begin{bmatrix}(x_2-x_1)^2+(y_2-y_1)^2\\ (x_3-x_1)^2+(y_3-y_1)^2\end{bmatrix} \tag{4}$$

The unknown vector $\mathbf{S}=(x_s, y_s)$ in (4) is obtained by

$$\mathbf{S}=\begin{bmatrix}x_s\\ y_s\end{bmatrix}=0{\cdot}5\begin{bmatrix}(x_2-x_1) & (y_2-y_1)\\ (x_3-x_1) & (y_3-y_1)\end{bmatrix}^{-1}\begin{bmatrix}(x_2-x_1)^2+(y_2-y_1)^2\\ (x_3-x_1)^2+(y_3-y_1)^2\end{bmatrix} \tag{5}$$

Hence the circumradius r and circumcenter $C=(x_c, y_c)$ are obtained by:

$$r=\sqrt{x_s^2+y_s^2},\ \text{or}\ r^2=x_s^2+y_s^2 \tag{6}$$

$$C=\begin{bmatrix}x_c\\ y_c\end{bmatrix}=\mathbf{S}+1=\begin{bmatrix}x_s\\ y_s\end{bmatrix}+\begin{bmatrix}x_1\\ y_1\end{bmatrix}=\begin{bmatrix}x_s+x_1\\ y_s+y_1\end{bmatrix} \tag{7}$$

References

BERNAL, J., 1988, On constructing Delaunay triangulations for a set of constrained line segments, Technical Note 1252, National Institute of Standards and Technology, United States Department of Commerce.

BOWYER, A., 1981, Computing Dirichlet tessellations. *Computer Journal*, **24**, 162–166.

BRASSEL, K. E., and REIF, D., 1979, Procedure to generate Thiessen polygons. *Geographical Analysis*, **11**, 289–303.

BURROUGH, P. A., 1986, *Principles of Geographical Information Systems for Land Resources Assessment* (New York: Oxford).

CLARKE, K. C., 1990, *Analytical and Computer Cartography* (New Jersey: Prentice Hall).

CHEW, L. P., 1989, Constrained Delaunay triangulations. *Algorithmica*, **4**, 97–108.

DE FLORIANI, L., and PUPPO, E., 1992, An on-line algorithm for constrained Delaunay triangulation. *CVGIP: Graphical Models and Image Processing*, **54**, 290–300.

DELAUNAY, B., 1934, Sur la sphère vide. *Bulletin of the Academy of Sciences of the USSR, Classe des Sciences Mathématiques et Naturelles*, **8**, 793–800.

DWYER, R. A., 1987, A fast divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, **2**, 137–151.

FALCIDIENO, F., and SPAGNUOLO, M., 1991, A new method for the characterization of topographic surfaces. *International Journal of Geographical Informations Systems*, **5**, 397–412.

HAYES, W. B., and KOCH, G. S., 1984, Constructing and analyzing area-of-influence polygons by computer. *Computers & Geosciences*, **10**, 411–430.

GOLD, C., and CORMACK, S., 1987, Spatially ordered networks and topographic reconstructions. *International Journal of Geographical Information Systems*, **1**, 137–148.

GREEN, P. J., and SIBSON, R., 1978, Computing Dirichlet tessellations in the plane. *Computer Journal*, **21**, 168–173.

KIRKPATRICK, D. G., 1980, A note on Delaunay and optimal triangulations. *Information Processing Letters*, **10**, 127–128.

LARKIN, B. J., 1991, An ANSI C program to determine in expected linear time the vertices of the convex hull of a set of planar points. *Computers & Geosciences*, **17**, 431–443.

LAWSON, C. L., 1972, Generation of a triangular grid with application to contour plotting, California Institute of Technology, Jet Pollution Laboratory, Technical Memorandum No. 299.

LAWSON, C. L., 1977, Software for C^1 surface interpolation. In *Mathematical Software III*, edited by J. Rice (New York: Academic Press), pp. 161–194.

LEE, D. T., and LIN, A. K., 1986, Generalized Delaunay triangulation for planar graph. *Discrete & Computational Geometry*, **1**, 201–217.

LEE, D. T., and SCHACHTER, B. J., 1980, Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer and Information Sciences*, **9**, 219–242.

LEE, J., 1991, Analyses of visibility sites on topographic surfaces. *International Journal of Geographical Information Systems*, **5**, 413–429.

LEWIS, B. A., and ROBINSON, J. S., 1978, Triangulation of planar regions with applications. *Computer Journal*, **21**, 324–332.

LINGAS, A., 1986, The greedy and Delaunay triangulations are not bad in the average case. *Information Processing Letters*, **22**, 25–31.

MACEDONIO, G., and PARESCHI, M. T., 1991, An algorithm for the triangulation of arbitrarily distributed points: applications to volume estimate and terrain fitting. *Computers & Geosciences*, **17**, 859–874.

MANACHER, G. K., and ZOBRIST, A. L., 1979, Neither the greedy nor the Delaunay triangulation of a planar point set approximates the optimal triangulation. *Information Processing Letters*, **9**, 31–34.

MARK, D. M., 1975, Computer analysis of topography: a comparison of terrain storage methods, *Geografiska Annaler*, **57**, 179–188.

MAUS, A., 1984, Delaunay triangulation and the convex Convex Hull of n points in expected linear time. *BIT*, **24**, 151–163.

MCCULLAGH, M. J., 1988, Terrain and surface modeling systems: theory and practice, *Photogrammetric Record*, **12**, 747–779.

MCCULLAGH, M. J., and ROSS, C. G., 1980, Delaunay tringulation of a random data set for isarithmic mapping. *The Cartographic Journal*, **17**, 93–99.

MIRANTE, A., and WEINGARTEN, N.,1982, The radial sweep algorithm for constructing triangulated irregular networks. *I.E.E.E. Computer Graphics and Applications*, **2**, 11–21.

PEUCKER, T. K., FLOWER, R. L., LITTLE, J. J., and MARK, D. M., 1976, Digital Representation of Three-dimensional Surfaces by Triangulated Irregular Networks (TIN), Technical Report Number 10, United State Office of Naval Research, Geography Programs.

SEDGEWICK, R., 1988, *Algorithms* (New York: Addison-Wesley).

SHAMOS, M. I., and HOEY, D., 1975, Closest-point problems. *Proceedings of the 16th Annual Symposium on the Foundations of Computer Science* (Washington: I.E.E.E.), pp. 151–162.

SIBSON, R., 1978, Locally equiangular triangulations. *Computer Journal*, **21**, 243–245.

SLOAN, S. W., 1987, A fast algorithm for constructing Delaunay triangulations in the plane. *Advanced Engineering Software*, **9**, 34–55.

THIESSEN, A. H., 1911, Precipitation averages for large areas. *Monthly Weather Review*, **39**, 1082–1084.

TSAI, V. J. D., and VONDEROHE, A. P., 1991, A generalized algorithm for the construction of Delaunay triangulations in Euclidean n-space. *Proceedings of GIS/LIS '91, Atlanta, GA*, **2**, 562–571.

VORONOI, G., 1908, Nouvelles applications des parameters continus à la théorie des formes quadratiques. *Deuxième Mémoire: Recherches sur les parallelloedres primitifs. Journal fur die Reïne und Angewandte Mathematik*, **134**, 198–287.

WATSON, D. F., 1981, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *Computer Journal*, **24**, 167–172.