The efficiency of a triangular mesh generator rests on its triangulation algorithm and data structure. Digital terrain models (DTMs) require that the lines and points used to describe the ground be stored. The software that analyzes this data have to connect the data points with lines forming triangles. Providing the terrain as a DTM offers the end user numerous benefits.

The Delaunay criterion is not an algorithm for generating a mesh. It merely provides the condition to connect a set of points in space. Hence, it is necessary to provide a method for generating the triangulation. Constructive algorithms essentially proposed by Bowyer (1981), Watson (1981) and George and Hermeline (1992) are recognized as the basic foundations for effective algorithms. There are a wide variety of algorithms available to build a Delaunay triangulation for a set of points (Kanaganathan 1991).
5.1 IMPORTANT PROPERTIES OF TRIANGLES USED IN CONSTRUCTING TRIANGULATIONS

An important issue when constructing triangulations is to calculate the angles of a triangle. In general, the law of sines and cosines is applied. Let $p_1$, $p_2$ and $p_3$ be the three vertices; $a$, $b$ and $c$ be the length of the three edges and $\alpha$, $\beta$ and $\gamma$ be the opposite angles respectively (Figure 5.1).

![Figure 5.1 Vertices, Edges and Angles of a Triangle](image)

The following formulae are used:

$$\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c}$$

$$c^2 = a^2 + b^2 - 2ab \cos \gamma$$

The center of the inscribed circle is found by intersecting the straightlines which bisect each of the angles of a triangle.

The area of the triangle with vertices $p_1$, $p_2$ and $p_3$ is given by:

$$A(p_1,p_2,p_3) = \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = \frac{1}{2} \left( x_1 y_2 + x_2 y_3 + x_3 y_1 - x_1 y_3 - x_2 y_1 - x_3 y_2 \right)$$

where $\text{det}$ denotes the determinant.
All algorithms for computing Delaunay triangulations rely on fast operations for detecting whether a point is within a triangle’s circumcircle and an efficient data structure for storing triangles and edges. One way to detect if point $D$ lies in the circumcircle of $A$, $B$, $C$ is to evaluate the determinant.

\[
\begin{vmatrix}
A_x & A_y & A_x^2 + A_y^2 & 1 \\
B_x & B_y & B_x^2 + B_y^2 & 1 \\
C_x & C_y & C_x^2 + C_y^2 & 1 \\
D_x & D_y & D_x^2 + D_y^2 & 1
\end{vmatrix} = \begin{vmatrix}
A_x - D_x & A_y - D_y & (A_x - D_x)^2 + (A_y - D_y)^2 \\
B_x - D_x & B_y - D_y & (B_x - D_x)^2 + (B_y - D_y)^2 \\
C_x - D_x & C_y - D_y & (C_x - D_x)^2 + (C_y - D_y)^2
\end{vmatrix} > 0
\]

Assuming $A$, $B$ and $C$ to lie counterclockwise, this is positive if and only if $D$ lies in the circumcircle (Fortune and Van Wyk 1993).

Two kinds of algorithms have been developed – Incremental and Divide & Conquer.

### 5.2 Incremental Strategy

Incremental Algorithm is one of the earliest, the most straightforward and the most extensively used algorithm for Delaunay triangulation. It is very popular due to its simplicity and robustness. The triangulation is obtained right on one pass itself. They build triangulation gradually, by inserting new vertices or edges. Every step preserves and ensures the rule of empty circumcircle (Gold 1999, Lawson 1977, Guibas, Knuth and Sharir 1992, Anglada 1995, Žalik and Kolingerova 2003, Vigo 1997).

The incremental construction algorithm adds new Delaunay triangles to an existing triangulation. The basic idea of incremental strategy is as follows: First, take a subset of the input small enough so that we can solve problem easily. Then, one by one add remaining elements (of input) while maintaining
the Delaunay criterion at each step. The algorithm discussed here was originally presented by Green and Sibson (1978), but the implementation is based entirely on the excellent paper by Guibas and Stolfi (1985).

The process is initiated by generating a supertriangle, an artificial triangle which encompasses all the points. At any stage of the triangulation process, therefore, one has an existing triangular mesh and a sample point to add to that mesh.

The incremental algorithm consists of three main parts:

1. Locate a triangle (or an edge), containing the inserted point.
2. Insert the point into the current triangulation; this might create a non-Delaunay triangulation.
3. Check each side of the triangle whether it fulfills the Delaunay criterion; else make necessary swaps of the sides.

First locate the triangle with vertices $p_i$, $p_j$ and $p_k$ containing the new point $P$. The most effective way to locate a triangle containing the point $P$ is to start from any triangle in the triangulation and move to adjacent triangles, using the determinant $D$ to check on which side of the directed edge $AB$ is the point $P$.

$$D(A, B, P) = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{vmatrix}$$

If the point $P$ is to the left of directed edge $AB$, the next clockwise edge from $B$ is used, if not the next clockwise edge from $A$ is used. This test is repeated until three successive edges have $P$ on the left, thereby locating the
triangle containing the point P (method proposed by Mücke, Saias and Zhu 1996).

This triangle with vertices $p_i$, $p_j$ and $p_k$ containing the new point $p$ is subdivided into three new triangles if the point lies inside the triangle (Figure 5.2a) or into two new triangles if the point lies on an edge. In the latter case the neighbouring triangle is also divided as is shown in Figure 5.2b.

Check that the rule of empty circumcircle is preserved at every step i.e. a search is done for circumcircles of all triangles containing the vertex $p_r$, based on the results of the INCIRCLE test.

The triangulation is updated by performing diagonal swaps on edges.

Then the algorithm proceeds by incrementally inserting new points in the existing triangulation till all points to be inserted are exhausted.

At the end of the triangulation process any triangles which share edges with the supertriangle are deleted from the triangle list.

In case of degeneracy i.e. if four or more cocircular points are present in the input, theoretically, resulting Delaunay Triangulation is not unique. In such a case this algorithm will produce one of the possible triangulations as output.
FIGURE 5.3 Point Insertion Example

subroutine incremental_insertion
input : vertex list
output : triangle list
    initialize the data structures
determine the supertriangle
    add supertriangle vertices to the vertex list of the mesh
    add the supertriangle to the triangle list
    for each sample point in the vertex list
        locate the triangle in which the point lies
        if the point lies in the triangle
            add the two triangles to the triangle list
            add the three triangle edges
        endif
        if the point lies on the triangle
            add the two triangles to the triangle list
            add the two triangle edges
        endif
        for each far edge of each new triangle
            if the new point lies in the circumcircle
                swap the edge
            endif
        endfor
    endfor
    remove any triangles from the triangle list that use the supertriangle vertices
    remove the supertriangle vertices from the vertex list
end
After each point is added there is a net gain of two triangles. Thus the total number of triangles is twice the number of sample points. (This includes the supertriangle, when the triangles sharing edges with the supertriangle are deleted at the end the exact number of triangles will be less than twice the number of vertices, the exact number depends on the sample point distribution). Similarly, after the insertion of a point, there is a net gain of three sides. Thus, the total number of sides is thrice the number of sample points.

FIGURE 5.4 Example of Point Insertion
FIGURE 5.4 Example of Point Insertion (continued)
To define the complexity of the algorithm used we need to identify the components which are dependent on the number of points and therefore play an important part in the definition procedure. When a new point is inserted into an existing triangulation we need to search the list of triangles to find all elements whose circumcircle contains the new point. The time needed to construct the edges can be considered constant. Locating the containing triangle requires $O(n)$ time; however, since the search is resumed from the triangle that was found last,
the time taken is much lesser. In the worst case the insertion of a point can require $O(n)$ edges to be swapped. However, Guibas, Knuth and Sharir (1992) have shown that in practice the average number of edges tested per insertion is small. The incremental algorithm takes $O(n^2)$ time in the worst case.

Despite their $O(n^3)$ worst-case performance, incremental methods have gained popularity over the years, due to the discovery by Clarkson and Shor (1989), Guibas et al (1992), Mulmuley (1994) and Berg et al (1997) that an optimal $O(n \log n)$ expected running time can be obtained by randomizing the order of insertion and maintaining a suitable pointer structure as points are inserted.

The points can arrive on-line and there is no need to know all input points at the beginning because the points are inserted one at a time. However, the range of coordinates has to be known in advance. It also allows modification for constrained triangulation.

5.3 DIVIDE AND CONQUER ALGORITHM

A divide and conquer algorithm for triangulations in two dimensions is due to Lee and Schachter (1980) which was improved by Guibas and Stolfi (1985) and later by Dwyer (1987). In this algorithm, one recursively draws a line to split the vertices into two sets. The Delaunay triangulation is computed for each set, and then the two sets are merged along the splitting line. Using some clever tricks, the merge operation can be done in time $O(n)$, so the total running time is $O(n \log n)$. For certain types of point sets, such as a uniform random distribution, by intelligently picking the splitting lines the expected time can be reduced to $O(n \log \log n)$ while still maintaining worst-case performance.
The first worst-case time complexity $O(n \log n)$ algorithm for two-dimensional Delaunay triangulation was not an incremental algorithm, but a divide-and-conquer algorithm. Shamos and Hoey (1975) developed an algorithm for the construction of a Voronoi diagram, which may be easily dualized to form a Delaunay triangulation. In programming practice, Shamos and Hoey’s algorithm is unnecessarily complicated because forming a Delaunay triangulation directly is much easier, and is in fact the easiest way to construct a Voronoi diagram. Lee and Schacter (1980) were the first to publish a divide-and-conquer algorithm that directly constructs a Delaunay triangulation. The algorithm is nonetheless intricate and quite demanding to implement. Guibas and Stolfi (1985) provide an important aid to programmers by filling out many tricky implementation details.

The divide-and-conquer relies on a recursive approach and has two major steps.

- **Top down step**
  In this step, one recursively draws a line to split the vertices into two sets. When the sets are small enough, they are trivial to triangulate. The Delaunay triangulation is computed for each set.

- **Bottom-up step**
  This step consists of recursively merging the solutions. The two sets are merged along the splitting line to form larger ones. This is the most difficult step.

  The algorithm here follows closely the one proposed by Guibas and Stolfi. Like theirs, it runs in time $O(n \log n)$ and uses linear storage. First the points are sorted lexicographically (by $x$ coordinate, with ties resolved by $y$ coordinate). The sorted sites are then partitioned successively into two halves, a
left half (L) and a right half (R). The recursion terminates at either two or three
sites, in which case either an edge or a triangle is created. The triangulation is
completed by knitting together the triangulation of L and the triangulation of R.

The merge step is the most complicated part of the algorithm. It is
done bottom-to-top, in which some of the existing edges of the L triangulation
(LL-edges) and R triangulation (RR-edges) are removed and new LR-edges i.e.
cross edges with one endpoint from the L triangulation and the other endpoint
from the R triangulation, are added. All these LR-edges must cross a line parallel
to the y-axis and placed at the splitting x value. The cross edges are determined
in vertical order, starting with the lower triangulation. (Cross edges are regarded
as oriented from R to L). Successive cross edges are found by a three-step
process:

1. Find the best point in L for a cross edge connected to the origin of the
topmost cross edge.
2. Find the best point in R for a cross edge connected to the destination of
the topmost cross edge.
3. Choose the best point between the two chosen in (1) and (2) and add the
next cross edge.

The best candidate point from L is found by evaluating in
counterclockwise order the suitability of the sites in L adjacent to the L vertex of
the topmost cross edge. Initially the first counterclockwise site is assumed to be
the best site. The next counterclockwise site is a better site if it lies inside the
circumcircle of the first counterclockwise site and the endpoints of the topmost
cross edge.
In Figure 5.5, D is a better candidate than C. In this case, delete edge BC; update C to refer to D and D to the next counterclockwise adjacent point. The iteration stops as soon as a point D lies outside the circumcircle A, B, C or when the point D becomes invalid i.e. lies to the left or below the edge AB.

The choice between the best candidate point from L and the best candidate point from R is made as follows: if either site is invalid then the other site is chosen, otherwise the site with the smallest circumcircle is chosen.

The merge step terminates when there are no valid sites in either triangulation L or triangulation R.

Consider the following set of 10 points, which has been ordered.
FIGURE 5.6 Illustration of Divide and Conquer Strategy
FIGURE 5.6 Illustration of Divide and Conquer Strategy (continued)
In the algorithms developed, to resolve degeneracies, when two points are coincident, one of them is rejected. In the case of four points being cocircular, the Delaunay triangulation is not unique and the algorithms output any one of them.

5.4 ALGORITHM FOR INSERTION OF CONSTRAINED EDGES

A good terrain modeling algorithm should have the ability to control exactly how the triangulation works and thus to precisely control the resulting terrain surface. Predefined edges in triangulations are used to represent rivers and roads in terrain models and geological faults in geological horizon modeling. This is done by adding boundaries and breaklines. Constrained Delaunay triangulations ensure that specified edges are present in the mesh.

Given the set of input vertices and the set of input edges, the mesh in Figure 5.7 results. Since all the given edges are present in the resulting mesh, it is a constrained triangulation.

![Figure 5.7 Example - Introducing constrained edges into a Delaunay triangulation](image)
Boundaries may appear in the interior of a region as well as on its exterior surfaces. *Exterior boundaries* separate meshed and unmeshed portions of space and are found on the outer surface and in internal holes of a mesh. *Interior boundaries* appear within meshed portions of space and enforce the constraint that elements may not pierce them. These boundaries are typically used to separate regions that have different physical properties; for example, in a heat propagation problem at the contact plane between two materials of different conductivities. An interior boundary is represented by a collection of edges of the mesh.

Breaklines can be employed to create superior terrain meshes that accurately resemble the actual landscape to be recreated and correctly interpret the existing conditions. Breaklines define and control surface behavior in terms of smoothness and continuity. They have a significant effect in terms of describing surface behavior when incorporated in a terrain model. Two types of breaklines could be included: hard and soft. Hard breaklines define interruptions in surface smoothness i.e. a physical discontinuity in slope and are typically used to define streams, ridges, shorelines, building footprints, dams and other locations of abrupt surface change. Soft breaklines are used to ensure that known elevation values along a linear feature (such as a roadway) are maintained in a terrain model i.e. soft breaklines do not alter slope. Soft breaklines can also be used to ensure that linear features and polygon edges are maintained in the terrain model. Soft breaklines, however, do not define interruptions in surface smoothness. A study area boundary is an example of a soft breakline. Soft breaklines are the ones that can be overlooked during further processing / analysis such as volume calculations. Distinctions between "hard" and "soft" breaklines facilitate the generation of contours.
In the algorithms developed, boundaries and breaklines (called as constrained edges) force an edge of a triangle and are not interpolated across. A constrained edge creates a barrier in the Delaunay Triangulation and hence may not be swapped during incremental insertion of a vertex. Information on a constrained edge is augmented into the data structure as non-zero integral values, called constrained edge index. A non-constrained edge is given the value zero.

The algorithm for inserting a constrained edge $ab$ into a constrained Delaunay triangulation follows the following steps:

i. Remove the triangles $t_1; \ldots; t_k$ cut by $ab$ from the constrained Delaunay triangulation so that a region without triangulation is left.

ii. Add the edge $ab$ to the result.

iii. Re-triangulate the upper and lower regions of the edge $ab$ that were not triangulated in the first step.

The first step implies finding the triangle of the constrained Delaunay triangulation that contains the vertex ‘$a$’, which is cut by the edge $ab$. The most effective way to locate the triangle in the triangulation that contains a point is to start from any triangle in the triangulation and moving to adjacent triangles, until the triangle $t$ that contains the point as one of its vertices is reached. From triangle $t$, we can use the adjacency relationship between the triangles to move through the triangles that converge on $a$, moving in a clockwise direction until we find the triangle that is cut by the segment $ab$. This triangle is called $t_1$.

Once the triangle $t_1$ has been located, the remaining triangles $t_2; \ldots; t_k$ cut by $ab$ are easily found by using the same adjacency relationship.

FIGURE 5.8 shows the process for insertion of a constrained edge within a Constrained Delaunay Triangulation.
FIGURE 5.8 Insertion of a constrained edge into a Constrained Delaunay Triangulation

subroutine Constrained edges _insertion
input : vertex list, constrained edge list
output : triangle list
    insert the two end points into the triangulation, using incremental algorithm
    if an edge exists between the two points, make it constrained.
    else
        if an edge can be created without intersecting an existing edge
            create the edge
            make it constrained
        else
            if the intersecting edge is not constrained, delete it
        else
            find the point of intersection.
            split the constrained edge into two, at the point of intersection.
            make both parts as constrained
            add the point of intersection to the vertices list of the mesh
            make new edges from the point of intersection to the two endpoints.
            make these edges as constrained.
    endif
    endif
    enforce Delaunay condition on each side of the constrained edge
end subroutine
FIGURE 5.9 Illustration of insertion of intersecting constrained edges

The difficulty of introducing a constrained edge into a constrained Delaunay triangulation depends on the number of triangles that are cut by the edge. In certain cases, very small features (like a vertex lying next to a constrained edge) can cause a single constrained edge to be split an arbitrary number of times.

Since the algorithm is incremental, the points and the constrained edges of the mesh can be introduced in any order. Thus, to refine a terrain model, simply add the additional breaklines and the algorithm will instantly create a revised surface honoring the new breaklines.

Let \( n \) be the number of triangles of the initial constrained Delaunay triangulation and let \( e \) be the number of triangles of the constrained Delaunay
triangulation cut by edge ab. Then the worst case time complexity of the step that finds the triangle from the constrained Delaunay triangulation that contains point ‘a’ and is cut by edge ab is \( O(n) \), because locating the point inside the constrained Delaunay triangulation is \( O(n) \) and finding the triangle that has ‘a’ as vertex and whose opposite edge is cut by ab is also \( O(n) \). Constructing the upper and lower triangulations has a worst case time complexity of \( O(e) \). The number of vertices of these triangles will be \( O(e) \) in the worst case. This gives a worst case time complexity of \( O(e^2) \). As the number of edges from a triangulation is linear to the number of vertices, the algorithm to add a constrained edge has a worst case time complexity of \( O(n^2) \).

**5.5 SAVING THE TRIANGULATION**

A terrain model is represented by a set of vertices \( V \), a set of edges \( E \) and a set of triangular faces \( F \). The three-dimensional coordinates of the original data points are assigned to the vertices. Each edge connects two vertices and is the intersection of exactly two faces. The terrain is approximated by the mesh consisting of the triangles. Inside the triangles, the surface is assumed to be planar.

The triangulation is saved as files of vertices, edges and triangles. The vertices as coordinates of the points, are saved in a file with extension .pts.ds; the edges are saved in a file with extension .sid.ds and the triangles in a file with extension .tri.ds where ds stands for the data structure used.

**5.6 LOADING THE TRIANGULATION**

This algorithm reconstructs the original mesh / triangulation from the .pts.ds and .sid.ds or .tri.ds files. It also reconstructs the constrained edges.
The algorithm makes a stack of all the edges / triangles adjacent to each vertex. Then the respective adjacencies are found and bound together.

This algorithm is very important in order to be able to edit the survey data and to save the changes introduced by the user into the triangulation, giving discretionary control of the triangulation to the user.

5.7 APPLICATION: VOLUME CALCULATION

The creation of an accurate terrain model of existing conditions is the foundation for many applications - for example, knowing where the existing highs and lows are on your site and helps you evaluate current storm water runoff characteristics. When the time comes to make design changes, calculate volumes or cut cross sections, you will save enormous amounts of time and effort with a DTM available at hand. Values of volume in terrains that do not have regular geometric structure can be obtained more accurately by using their DTMs.

The DTM created has been used to calculate the quantity of earthwork in cut and fill applications. Volume calculations are done by the summation of the volume of each prism constituted by a triangle. Inside the triangle, the surface is assumed to be planar. Cut-and-fill factors are automatically computed, reducing tedious manual calculations.

Cut-and-fill volumes are calculated considering a base elevation. Values above base elevation is the cut volume and below is fill volume. If the
triangle under consideration crosses the elevation line, then we have a situation where part of the volume adds to the cut volume and the rest to the fill volume. In this case, the points of intersection are found. Then, their respective heights are interpolated and the volumes found.

![Illustration of Volume Calculation](image)

**FIGURE 5.10 Illustration of Volume Calculation**

If there are data collected over the same area at different times, the surfaces can be compared and change analysis can be performed. Both cut and fill earthwork volume calculations can be performed for single or multiple areas. In this case, a combined DTM is created, by interpolating the points in the triangulation where data is unavailable and the volume between the two layers is calculated.

Cut and fill volumes between two DTM have also been generated.