Chapter 1

Digital Elevation Models and TIN Algorithms

Marc van Kreveld

Abstract. This survey paper introduces various concepts and algorithms on terrains. The three basic models—the grid, the contour lines, and the triangulated irregular network—are described briefly, but the emphasis is on the triangulated irregular network (TIN) model. Algorithms are given for grid to TIN conversion, TIN traversal, contour line selection and computation, and drainage network computation. The efficiency of the algorithms is discussed as well.

1.1 Introduction

In the GIS literature, papers abound on terrain issues. Obviously, survey papers on terrains have been written (most notably, by Weibel and Heller [114]), and textbooks on GIS and automated cartography also deal with terrains [3, 10, 63, 67, 77, 103, 117]. But no survey has been written with the emphasis on algorithms on terrains. This is an attempt to do so. It is almost impossible to produce a complete survey, or even a nearly complete bibliography. Instead, this survey highlights the most important concepts and problems on terrain data, and discusses a few algorithms more thoroughly. The emphasis is on the efficiency of the algorithms, but it appears that 'model' and 'algorithm' cannot be seen as separate issues. Two algorithms that compute the drainage network on a terrain generally use a different model for the drainage network, so the algorithms that perform the computation cannot really be compared on efficiency: the algorithms don't compute the same thing. Even worse, the algorithms may be based on different terrain models.

Since the choice of a model and an algorithm go hand in hand, this survey will deal with both. We concentrate on the triangulated irregular network model for representing terrains, but sometimes we also deal with the other common model, the grid. The techniques underlying the algorithms have been developed both by computational geometers and by GIS researchers.

Another issue of importance is how the efficiency of algorithms should be analyzed. Computational geometers usually consider the worst possible inputs and make sure that the algorithm works well even in these cases. GIS researchers often don't analyze their algorithms, or give timings of an implementation. We'll adopt an intermediate view: if the worst case efficiency is of the same order as the typical efficiency for real-world inputs, then we'll use worst case analysis. If there seems to be an important difference, we'll try to track down why the worst case analysis is too pessimistic in practice, and try to motivate a more realistic efficiency statement.

This survey certainly doesn't include all aspects of terrain modelling and algorithms. Not included are data compression for terrains [40, 42], surface networks [88, 111, 116], and dealing with errors and uncertainty [4, 5, 70]. Other aspects are treated only briefly, like hierarchical terrain modelling, viewshed analysis, path planning, and statistics. Also, standard algorithms that have been described in all textbooks on computational geometry [14, 83, 90] are omitted. These include the computation of subdivision intersections (or: map overlay), Voronoi diagrams, and Delaunay triangulations.

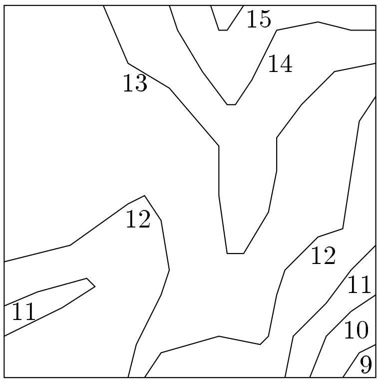
1.2 Terrain models and representation

1.2.1 The regular square grid

The regular square grid—or simply the grid—is a structure that specifies values at a regular square tesselation of the domain, see Figure 1.1 (a). In the computer it is stored as a two-dimensional array. For every square in the tesselation, or entry in the array, exactly one elevation value is specified. There are different interpretations of the grid. Firstly, the elevation stored can be thought of as the elevation for every point inside the square. In this case the digital elevation model is a non-continuous function. Secondly, the elevation stored can represent only the elevation at the center point of the square, or the average elevation inside the square. In these cases, an interpolation method is necessary to get a digital elevation model that specifies the elevation of every point. A possible interpolation method for any point p different from a square center is using the weighted average of the elevations of the four centers surrounding the point p, where the weight depends on the distances to the centers. But there are other possibilities too, like interpolation by splines.

12	12	13	13	14	15	14	14	14
12	12	12	13	14	15	14	14	13
12	12	12	13	13	14	14	13	12
12	12	12	13	13	13	13	12	12
12	12	12	12	13	13	13	12	12
12	12	12	12	12	13	12	12	12
12	12	11	12	12	13	12	12	11
11	11	12	12	12	12	12	11	10
12	12	12	12	12	12	12	10	9

(a) grid model



(b) contour line model

Figure 1.1: Two models for elevation.

1.2.2 The contour line model

In the contour line model to represent elevation, some set of elevation values is given, and every contour line with one of these elevations is represented in the model. So a collection of contour lines is stored along with its elevation, see Figure 1.1 (b). Sometimes the term isoline or isocontour is used when the elevation model represents something else than height above sea level. Throughout this survey we'll continue to use contour line in all situations.

A contour line is usually stored as a sequence of points with its x- and y-coordinates. It then represents a simple polygon or polygonal chain of which the elevation is specified. Since the contour line model specifies the elevation only at a subset of the domain, an interpolation method is needed to determine the elevation at other points. Since any point lies in a region bounded by contour lines of only two elevations, one usually only uses the bounding contour lines for the interpolation.

The contour lines can be stored in a doubly connected edge list [14, 89, 90] or quad edge structure [54]. An alternative to this representation is by means of the *contour tree* [**?**] (or *topographic change tree* [65]). An contour line subdivision is a special type of planar subdivision with no vertices of degree three or greater. In theory they could exist if a contour line contains a pass, but such a situation would be coincidental and undesirable in mapping. So one generally assumes that each contour line is either a cycle of edges (polygon) or a chain between two points on the boundary of the elevation model. Suppose that every contour line corresponds to a node in a graph, and two nodes are connected by an arc if they bound the same region. This graph is easily seen to be a tree, the *contour tree*. A contour line model can therefore be stored by storing the contour tree, and with every node the cycle or chain of edges that together form the contour line.

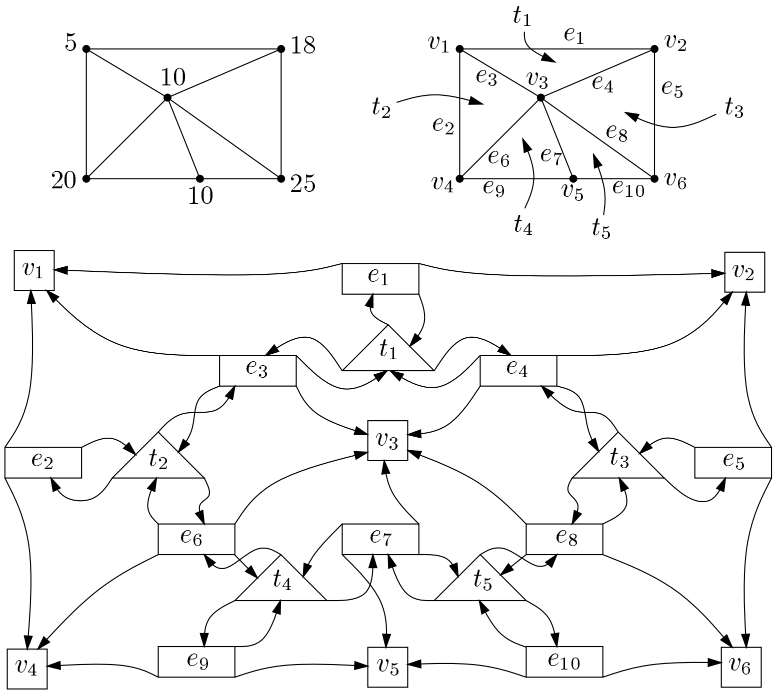


Figure 1.2: A TIN and the network structure for it. The three values and the list of each vertex are not shown.

1.2.3 The triangulated irregular network model

In the triangulated irregular network model, usually abbreviated to TIN, a finite set of points is stored together with their elevation. The points need not lie in any particular pattern, and the density may vary. On these points a planar triangulation is given. Any point in the domain will lie on a vertex, an edge or in a triangle of the triangulation. If the point doesn't lie on a vertex, then its elevation is obtained by linear interpolation (of 2 points if it lies on an edge, and of 3 points if it lies in a triangle). So the model is a piecewise linear model that—in 3-dimensional space—can be visualized as a simply connected set of triangles. A TIN is continuous but not differentiable in the whole domain. The TIN model for terrains has been used since the seventies [46, 85, 87].

A possible storage scheme of a TIN is the doubly connected edge list [89, 90] or the quad edge structure [54]. Both are ways of representing the topology of any planar subdivision. These representations allow for all necessary traversal operations in an efficient manner.

An alternative representation can be used for a triangulation, since it is a special type of planar subdivision, see also Figure 1.2. This alternative saves storage space and makes traversal and other operaties a bit simpler. For every triangle t, edge e, and vertex v, there is a record (or object) for that feature. The record of a triangle t has three fields with pointers. These pointers are directed to the records of each of the three edges incident to t. The record of an edge e has four fields with pointers. Two of the pointers are directed to the records of the two incident triangles, and the other two pointers are directed to the records of the incident vertices. The record of a vertex v has three fields with values. These are the x- and y-coordinates and the elevation of the vertex.

The topological network structure just described allows for finding—for every triangle—the elevations of its vertices in constant time, finding the adjacent triangles for a given triangle in constant time, and more. This allows us, for instance, to walk through the triangulation along a straight line efficiently, which is necessary to determine a profile of a terrain. Variations on the structure are possible, for instance by storing a list of pointers at the vertex records to the incident edges.

1.2.4 Hierarchical models

A hierarchical terrain model is a terrain model that represents a terrain in various levels of error, or, to make it sound less badly, various levels of imprecision. Most approaches of this type are based on TINs. Generally, TINs with more vertices have less error than TINs with fewer vertices. On the other hand, TINs with many vertices are more expensive to compute on. So if it is okay to have some error in an application, it may be better to work with a TIN with fewer vertices. A hierarchical terrain model allows the user to choose a terrain with the appropriate precision for each task.

Issues of importance of hierarchical terrain modelling are:

- The storage required by the model. Explicitly storing a terrain at many levels of detail leads to redundancy and excessive use of storage.
- The model may incorporate an efficient search structure automatically. For instance, locating a point on a terrain may be possible by first locating it on the coarsest level, and then locating it at repeatedly finer levels of detail.
- The triangulations should preferably be well-shaped, for instance, using the Delaunay triangulation.
- The model may allow a mixture of different detail levels in different parts of the terrain. This is useful in flight simulation, where the terrain close by must be shown with more detail than parts far away.
- It may be important that the model is consistent with respect to morphologic features. For instance, if there is a significant peak in a terrain at some detail level, one would wish that the same peak also exists on every finer detail level.

The list of issues already indicates that there probably isn't one best solution to all applications. Many hierarchical terrain models have been suggested [16, 18, 20, 35, 59, 95, 110]; the list of references is far from complete. A survey of the topic also exists [33].

1.3 Access to TINs

1.3.1 Traversal of a TIN

There is an old but good method to traverse a TIN and visit all its vertices, edges, and triangles in a simple way [47, 49, 51]. The nice thing about the method is that hardly any additional storage is needed: no mark bits, no stack, just one access point to the TIN.

Let T be a TIN stored in the structure just described, and let v be the bottom left vertex of T. For any triangle, we will give names to the incident edges. This implies that one edge receives two names, one for each incident triangle. Let t be

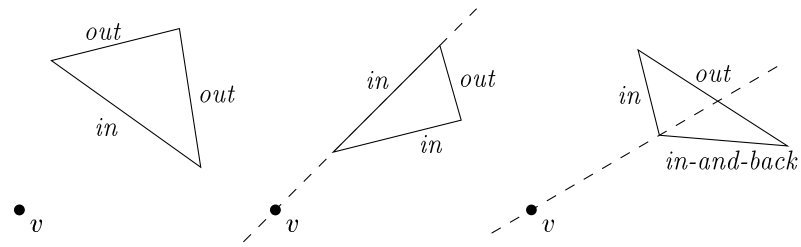


Figure 1.3: Types of edges for the traversal algorithm.

any triangle of the TIN, we name an incident edge to be *in* if the line containing that edge separates the vertex v and the interior of the triangle t. If the vertex and the interior of t lie to the same side, the edge is *out*. If the line supporting the edge contains the vertex v we have a special case. The edge is *out* if the interior of t lies left of the supporting line, otherwise it is *in*, including the case of horizontal supporting lines. This is well-defined for TINs with a rectangle as the boundary, since we chose v to be the bottom left vertex. Notice that any interior edge of the TIN is *in* for the one incident triangle and *out* for the other.

All triangles have either one *in* edge and two *out* edges, or two *in* edges and one *out* edge. The *in* edges will represent the edge through which the triangle will be entered during a traversal. To make sure that we enter a triangle only once, consider any triangle with two *in* edges. These two *in* edges share a vertex, and the line through this vertex and through v separates the two *in* edges. The *in* edge that lies above, or left of this line, or is supported by the line, will be the real *in* edge. The other *in* edge will be *in-and-back*: the traversal enters the triangle through it, but will go back through that same edge immediately after. Similarly, for triangles that have two *out* edges, one will be the first *out* edge and the other the last *out* edge. Consider the line through v and the vertex incident to both *out* edges, the first *out* edge is the one left of this line. Note that for any triangle, we can determine easily, in constant time, which of its edges are *in* and *out* distinguish the real *in* and the *in-and-back* edges, and distinguish the first and last *out* edges.

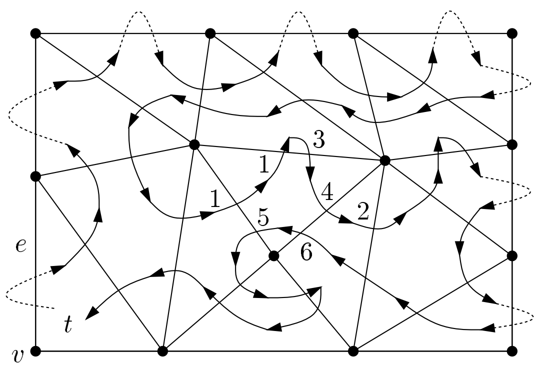


Figure 1.4: Traversing a TIN; numbers at arrows correspond to the algorithm.

The algorithm starts at the triangle that has the bottom left vertex v as a vertex, and has one edge on the left side of the bounding rectangle. Let that triangle be t, and let e be the edge on the left side. The algorithm proceeds from triangle to triangle by crossing edges, and make the decision on which edge to cross next based solely on the type of edge through which it was just entered, and the type of triangle it is currently in. Since there are two types of triangles, and each has three edges, the algorithm distinguishes six cases to decide how the traversal should proceed, see also Figure 1.4. Edge e will be the first edge crossed in the traversal. When the algorithm reaches e a second time, the traversal is completed.

1. If e is the only *in* edge of t, then let e' be the first *out* edge of t, and let t' be the triangle on the other side of e. Repeat the algorithm with $t := t'$ and $e := e'$.

2. If e is the real *in* edge of t, then let e' be the *out* edge of t, and let t' be the triangle on the other side of e'. Repeat the algorithm with $t := t'$ and $e := e'$.

3. If e is an *in-and-back* edge of t, then let t' be the triangle on the other side of e. Repeat the algorithm with $t := t'$ and e.

4. If t has two *out* edges and e is the first one, then let e' be the last *out* edge of t, and let t' be the triangle on the other side of it. Repeat the algorithm with $t := t'$ and $e := e'$.

5. If t has two *out* edges and e is the last one, then let e' be the *in* edge of t, and let t' be the triangle on the other side of it. Repeat the algorithm with $t := t'$ and $e := e'$.

6. If e is the only *out* edge of t, then let e' be the real *in* edge and t' the triangle on the other side of it. Repeat the algorithm with $t := t'$ and $e := e'$.

If there appears to be no triangle on the other side of the edge we have just crossed, then we return to the previous triangle through the same edge we just left immediately. Then we proceed as usual: the current triangle has just been entered through an edge that happens to be part of the bounding rectangle.

The algorithm visits every triangle exactly three times, once through each of its edges. At all times, we need only know the current triangle, the edge through which it was accessed, and the vertex v. We can report a triangle when we visit it for the first time. If all edges or all vertices of the TIN should be reported, some simple adaptations are needed.

The idea described above has been extended to the traversal of other subdivisions than just triangulations [15, 23].

1.3.2 Efficient access to a TIN

In a regular square grid structure there is direct access to every part of the terrain. If one wants to know the elevation at a point with coordinates x and y, these coordinates can simply be rewritten to index values in the two-dimensional array. In a TIN this is not so easy, because a TIN is a pointer structure. If one wants to know the elevation at a point given its coordinates, one could test each triangle to see if it contains the point. This is rather inefficient, obviously. We briefly review three methods to gain access to the TIN at a specific point.

Access using quadtrees

Quadtrees and other spatial indexing structures like R-trees can be used to get access to a TIN at a specific query point efficiently. A quadtree is a rooted search tree of degree four, meant to store 2-dimensional data. Its nodes represent a recursive decomposition of a big square (associated with the root of the quadtree) into four subsquares (the children of the root). Each of the four subsquares is decomposed into four yet smaller squares. Further decomposition stops as soon as the part of the TIN that falls in the subsquare is simple enough. For example, when only a few vertices of the TIN lie inside the square. A leaf node is created that represents this square, and at the leaf node we store a pointer to the triangle record of the triangle that contains the center of the square. From that triangle record we can walk in the topological structure to locate the triangle record of the triangle that contains the query point.

Another possibility is to take the smallest enclosing axis-parallel rectangle of each triangle of the TIN. The resulting set of rectangles may overlap, but one can expect that at each point there won't be many rectangles containing it. A quadtree, R-tree, or other 2-dimensional search tree can be used to store the rectangles, see for instance the book by Samet [92]. For a query point, we determine all rectangles that contain it, and then find the triangle enclosed by one of the located rectangles that really contains the query point.

Access using planar point location

In the computational geometry field several efficient methods have been developed to determine which region of a subdivision contains a given query point. This problem is known as point location. For a TIN with n triangles, it is possible to construct an $O(n)$ size data structure that allows for point location in $O(\log n)$ time. Among the many results, the most simple and efficient ones are by Sarnak and Tarjan [93], Clarkson and Shor [11], Kirkpatrick [60] and Seidel [96]. Descriptions can also be found in textbooks on computational geometry [14, 83, 90].

Jump-and-walk strategy

The simplest method to locate a point in a TIN among these three is the jump-and-walk strategy. It also requires very little additional storage, so it may well be the best choice in practice. Suppose that access to the TIN structure is provided by one pointer to some feature. Rather than traversing all triangles until we find one that contains the query point, we can also traverse the TIN in a straight line from the access point to the query point. We typically encounter much fewer triangles on the way than if we would traverse the whole structure. The query time will be even better to take more than one starting point. Among those we'll choose as the real starting point the one closest to the query point, in the Euclidean sense.

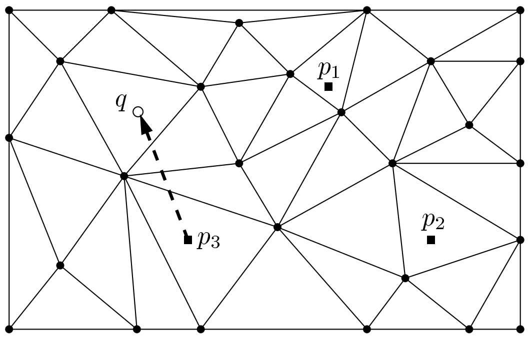


Figure 1.5: Jump-and-walk with three starting points.

In particular, choose m points $p_1, \ldots, p_m$ from the set of vertices of the TIN at random, and store a copy of those points in an unsorted list. With the copy we store a pointer to the vertex record in the topological TIN structure. When we query with a point q, we first determine the access point p_i closest to q in $O(m)$ time, and then we start tracing the line segment $\overline{p_i q}$, starting at the triangle containing p_i. If we choose m to be roughly $n^{1/3}$, the expected query time is $O(n^{1/3})$ under some distribution assumptions [?].

1.3.3 Windowing

When a user is interested in a part of a terrain, then this part must be extracted from the whole terrain. We assume that the selection of the interesting part is defined by some rectangle, the window, and all triangles intersecting that rectangle should be located. Again we take the TIN as the elevation model to which windowing is applied.

The algorithm consists of two steps. First, the position of the window on the terrain should be located, and second, the TIN triangles inside the window can be traversed. We can locate for instance the upper left corner of the window, and start traversing from there; we already discussed ways to locate a point in a TIN. Traversal can be done in time linear in the number of triangles intersecting the window. The ideas of traversal without mark bits can also be applied here [15].

1.4 Conversion between terrain models

Terrain data can be entered into a GIS in various formats. Often, contour line data is entered when paper maps with contour lines are digitized by hand. Also quite often, gridded data is the input format, for instance when the data is acquired by remote sensing or automatic photo-interpretation. There are various reasons why data in one format may need to be transformed into another. Gridded data usually is huge in size, resulting in high memory requirements and slow algorithms when the data is processed further. Contour line data often needs to be interpreted anyway before anything useful can be done with it. In several cases it may help to store and compute with TINs instead. Conversion from TINs into grids or contour lines may also be useful. The former problem is algorithmically quite straightforward and we won't discuss it here. The latter problem shows up when a TIN is visualized as a contour map; we deal with that issue later in Section 1.6.

This section gives a few algorithms and references for converting to TINs. First we discuss how point sample data can be converted to a TIN, then grid-to-TIN conversion is handled, and then we go from contour lines to a TIN.

1.4.1 From point sample to TIN

Suppose a set P of n points in the plane is given, each with an elevation value. To convert this information into a TIN, one could simply triangulate the point set. In fact, the triangulation is an interpolation of a region based on the points of P. It is common to use the Delaunay triangulation because it attempts to create well-shaped triangles. Efficient algorithms for the Delaunay triangulation have been known for a while. See for instance Lee and Schachter [68], Guibas and Stolfi [54], or any textbook on computational geometry [14, 83, 90].

When the interpolation provided by the Delaunay triangulation is not appropriate, one can use different, more advanced interpolation methods like natural neighbor interpolation [50, 91, 98], weighted moving averages [3], splines [37], or Kriging [3, 112]. It is possible to combine the advantages of the TIN with the quality of these more advanced interpolation methods as follows. Suppose we are given the point set P together with an interpolation function and a maximum error. The idea is: construct a TIN based on P and the interpolation function, such that at any point, the interpolated elevation and the elevation given by the TIN differ by at most the maximum error. Now it may not suffice to use only the points of P as TIN vertices, and we need to select more points. Ideally, we select as few points as possible, the optimization problem that shows up is very difficult. Heuristics can be used to choose additional points as vertices in the TIN, and hopefully not too many additional ones. It may also be the case that only a subset of the points of P is needed to represent the interpolation function with the desired accuracy. Or perhaps a completely different set of points is best. In any case, finding a TIn with minimum number of vertices will be difficult. It is known that the problem of computing a TIN with the minimum number of vertices, given a TIN and a maximum allowed error, is an NP-hard problem, implying that efficient algorithms are unlikely to exist.

1.4.2 From grid to TIN

Grid-to-TIN conversion can be seen as a special case of the conversion of sample points to a TIN. Also, grid-to-TIN conversion can be seen as a special case of TIN generalization: reducing the number of vertices of a TIN to represent a terrain. A grid can simply be triangulated to a fine regular triangulation. Various algorithms have been proposed in the literature; see for instance the survey by Garland and Heckbert [45], see also Lee [72]. Most of the methods have the following distinguishing features: (1) selecting which grid points to keep or discard, and (2) deciding when to stop selecting or discarding.

One method decides which grid points to keep or discard by initially assigning an importance to each grid point [7]. The importance is determined by comparing the elevation of a grid point with the interpolated elevation at the grid point based on the elevations of the eight neighbors. Only the grid points where the difference is greatest are kept. These points can be triangulated, for instance using the Delaunay triangulation, and become the TIN vertices.

A second method differs from the previous one by discarding grid points incrementally instead of using a precomputed importance [71, 72]. In a sense, the importance computation is postponed until the point is really discarded. A more detailed description follows later.

Thirdly, there are methods that start out with a coarse triangulation of only the four corner grid prints, and keep on refining the triangulation by adding more points in the triangles [31, 45, 55, 100]. Refining a triangle further stops when the triangle approximates the grid points that lie in it sufficiently well.

Another method start out by detecting surface specific features on the grid like peaks, pits, saddle points, ridges, and valley lines [38]. Then they complete

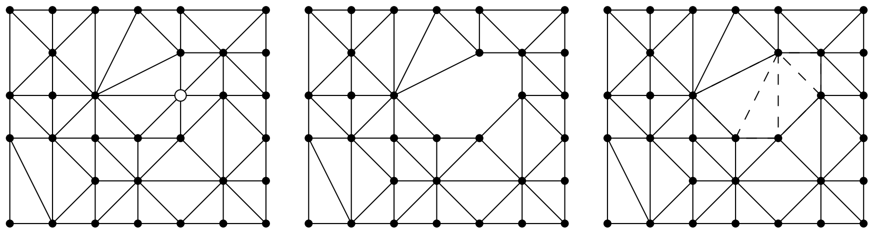


Figure 1.6: Left, a TIN with one vertex indicated by a circle. Middle, the polygon that appears when the indicated vertex is removed. Right, a Delaunay triangulation of the polygon.

these retained points and line segments to a TIN. Extraction of surface specific features is discussed later in this survey.

We describe two methods in more detail, namely, the drop heuristic method by Lee [71, 72] of the second type, and a method by Heller [55] (see also Fjällström [31] and Garland and Heckbert [45]) of the third type.

The drop heuristic

Lee's drop heuristic method takes a TIN as its input, and iteratively discards one vertex at a time to obtain a TIN with fewer vertices. Obviously, it also applies to grids as input if we consider it to be a triangulated regular grid. If a vertex is discarded, the incident edges are also removed and a polygon appears in the subdivision. To get back to a triangulation, the polygon is triangulated using the Delaunay triangulation. This will ascertain that if the algorithm starts with a Delaunay triangulation, then after every iteration we'll still have a Delaunay triangulation.

To decide which vertex should be discarded, each vertex is temporarily removed and the appearing polygon is triangulated—see Figure 1.6. Then we determine the vertical distance between the removed point and the new, simplified TIN. The removed vertex lies in one of the new triangles in the polygon, so this is easy to do. This vertical distance can be viewed as the error introduced by the deletion. Once we know the error that would be introduced, we add the removed vertex back to the TIN and temporarily remove another vertex. After we have done so for all vertices, we select the one for which the computed error is smallest and really discard it. The process continues until the created error is more than the prespecified allowed error.

It should be noted that the error at a vertex after it is discarded can become bigger when more vertices are discarded. So there is no guarantee that the error at all of the discarded vertices really is within the prespecified error. The drop heuristic method completely forgets about vertices that are discarded, although, at the expense of more computation, a variant of the method could still consider them. Below we'll analyze the typical running time of the standard algorithm.

A straightforward implementation of the algorithm requires $O(n \log n)$ time per iteration on a TIN with n vertices. Discarding the vertices temporarily to determine their error and retriangulation can be done in $O(n \log n)$ time in total. This is true because the total complexity of all polygons to be triangulated is linear in n. Unfortunately, the vertical distances may have to be recomputed after an iteration, because the deletion of some vertex may result in a change of introduced error of other vertices. It is possible to construct an example of a TIN with n vertices where the algorithm has to recompute the errors many times for many vertices, but this is not a typical case. Observe that if a vertex v is removed in some iteration, then only the vertices of the TIN adjacent to v can have a change in introduced error. For all other vertices, the error resulting from their removal stays the same. So the question is how many neighbors a vertex in the Delaunay triangulation has. In the worst case this number may be $n-1$, all other vertices, but on the average, a vertex has degree at most six. We'll analyze the typical case where any vertex that is removed has constant degree. Under this assumption we can design a variation of the given algorithm that will run in $O(n \log n)$ time. We describe this variation below.

1. For each vertex v in the TIN:

   - Temporarily remove it v.
   - Compute the Delaunay triangulation of the appearing polygon.
   - Determine the vertical distance $error(v)$ of v to the new TIN.
   - Add the removed vertex back to the TIN.

   Store $error(v)$ for each vertex v sorted in a balanced binary tree $\mathcal{T}$. At each node of $\mathcal{T}$ storing some $error(v)$, store a pointer to the vertex v in the TIN. At v, we store a pointer back to the corresponding node in $\mathcal{T}$.

2. Consider the node with smallest $error(v)$ in $\mathcal{T}$. If it is greater than the prespecified maximum error, the algorithm stops. Otherwise it proceeds with the next step.

3. Remove the node storing the smallest $error(v)$ from $\mathcal{T}$. Remove the corresponding vertex v from the TIN structure. Let $w_1, \ldots, w_j$ be the vertices adjacent to v. Retriangulate the polygon defined by $w_1, \ldots, w_j$ using the Delaunay triangulation.

4. For every vertex $w_i \in \{w_1, \ldots, w_j\}$:

   - Remove the node that stores $error(w_i)$ from $\mathcal{T}$.
   - Recompute the vertical distance to the terrain if w_i were removed as we did in the first step.
   - Insert the new $error(w_i)$ in $\mathcal{T}$.

   Continue at step 2.

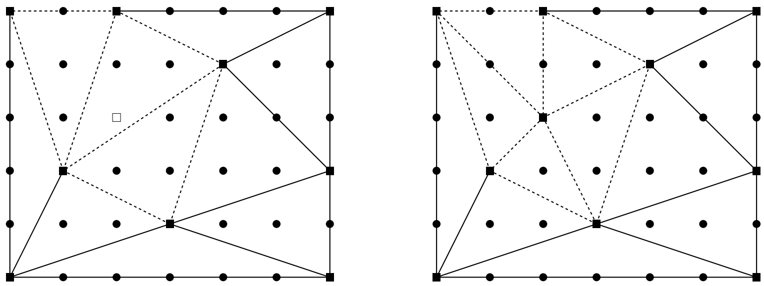


Figure 1.7: The right TIN shows the situation if the square grid point on the left is the one with maximum error.

If all vertices in the TIN have constant degree, then each iteration requires only $O(\log n)$ time. The polygon to be retriangulated only has a constant number of vertices, and only for these a new vertical distance need be computed. The insert and delete operations on the tree $\mathcal{T}$ take $O(\log n)$ time each. So the algorithm takes only $O(n \log n)$ time in typical cases.

Incremental refinement

The algorithm to be descibed next takes a grid and a maximum allowed error ϵ as the input. Unlike the drop heuristic, the algorithm to be described really does guarantee that the final TIN has error at most ϵ. The approach is to start with a coarse TIN with only a few vertices, and keep adding more points from the grid to the TIN to obtain less error.

1. Let P be the set of midpoints of grid cells, with their elevation value. Take the four corner points and remove them from P, and put them in a set S under construction.
2. Compute the Delaunay triangulation $\mathrm{DT}(S)$ of S.
3. Determine for all points in P in which triangle of $\mathrm{DT}(S)$ they fall. For points on edges we can choose either one. Store with each triangle of $\mathrm{DT}(S)$ a list of the points of P that lie in it.
4. If all points of P are approximated with error at most ϵ by the current TIN then the TIN is accepted and the algorithm stops. Otherwise, take the point with maximum approximation error, remove it from P and add it to S. Continue at step 2.

If we assume a simple and slow implementation of the algorithm, we observe that at most n times a Delaunay triangulation is computed. For each one, the points in P are distibuted among the triangles of $\mathrm{DT}(S)$. This requires $\Theta(n^3)$ tests of the type point in triangle, if a linear number of points are added to S.

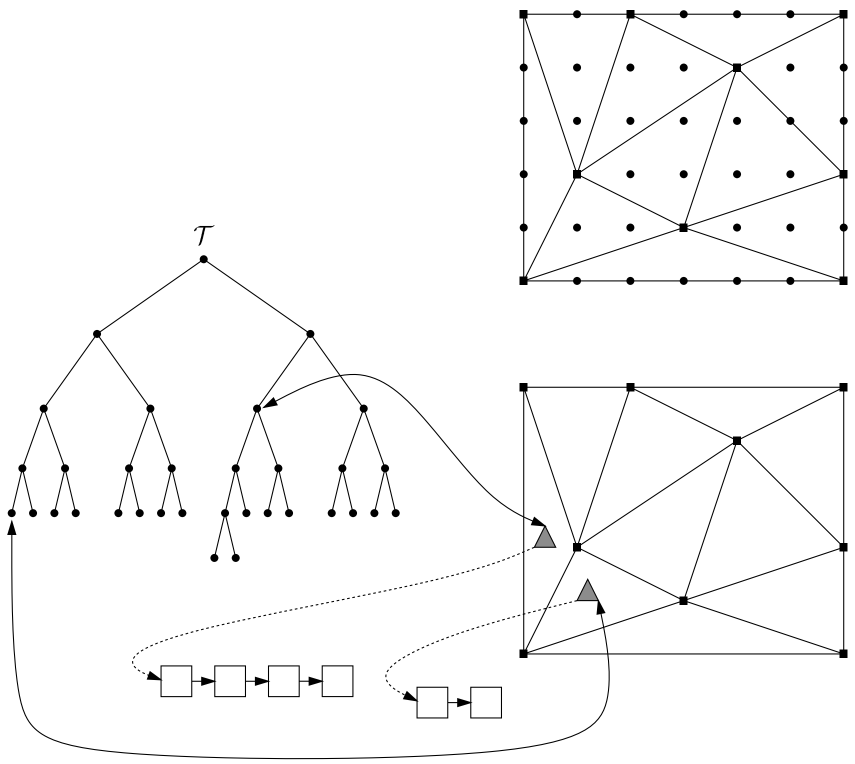


Figure 1.8: The situation for a TIN with vertices shown as small squares (top right), and the corresponding structure with a few of the pointers between triangle records, list elements, and tree nodes.

A much faster implementation has a worst case performance of $O(n^2 \log n)$ time, and in typical situations even better: typically $O(n \log n)$ time. The algorithm resembles incremental construction of the Delaunay triangulation to some extent [14, 52, 11]. Our algorithm, however, must also distribute the points of P and find the one with maximum approximation error. We'll show that these steps can be done efficiently.

Assume that $p \in P$ has been determined as the point with maximum error, and p must be removed from P and added to S. Then we locate the triangle t of $\mathrm{DT}(S)$ that contains p, and we find the vertices that will become neighbors of p in $\mathrm{DT}(S \cup \{p\})$. This update step of the Delaunay triangulation is the same as in the incremental construction algorithm. To distribute the points of $P \backslash \{p\}$ over the triangles of $\mathrm{DT}(S \cup \{p\})$, observe that only the triangles of which p is a vertex in $\mathrm{DT}(S \cup \{p\})$ have changed. So for all triangles of $\mathrm{DT}(S)$ that don't exist in $\mathrm{DT}(S \cup \{p\})$, we collect the associated lists of points. These points are distibuted among the new triangles and stored in new lists.

The problem that remains is locating the point with maximum error. It is solved as follows. For each triangle of the TIN we determine the point of P inside it with maximum error. These points are stored in the nodes of a balanced binary tree $\mathcal{T}$ sorted on error. This allows us to locate the point p with maximum error efficiently; it is in the rightmost leaf of $\mathcal{T}$. Before p is moved from P to S, the Delaunay triangulation must be changed accordingly. To find the triangle in $\mathrm{DT}(S)$ that contains p we'll use a pointer from the node in $\mathcal{T}$ to the triangle record in the TIN structure; such pointers are shown as dashed lines with arrows in Figure 1.8. The triangle records are shown as grey triangles. After updating the TIN to be $\mathrm{DT}(S \cup \{p\})$ we move p from P to S.

Then we reorganize the lists that were stored with the triangles. When p was added to the Delaunay triangulation, some triangles were destroyed. The point of P inside each one that had maximum error is deleted from $\mathcal{T}$. The lists of points of the destroyed triangles contain p and the points that must be distributed among the new triangles, and stored in new lists. For each of the new lists we must find the point that realizes the maximum error in the cooresponding triangle, and store it in $\mathcal{T}$. For efficiency reasons it is a good idea to use a cross-pointer from any list element that stores a point of P to the corresponding node in $\mathcal{T}$. Otherwise we may not be able to locate the points of which the error has changed efficiently in $\mathcal{T}$. These pointers are shown as solid lines with arrows in Figure 1.8. The pointers from the triangle records to the lists are shown dotted.

If k is the number of neighbors of p in $\mathrm{DT}(S \cup \{p\})$, then $k-2$ triangles were destroyed and k new ones were made. Let m be the number of points in the triangles incident to p in $\mathrm{DT}(S \cup \{p\})$. Then the iteration that added p as a vertex of the TIN requires $O(k + \log n)$ time for updating the Delaunay triangulation, $O(km)$ time to redistribute the m points over the k triangles, and $O(k \log n)$ time to update the balanced binary tree $\mathcal{T}$. In the worst case, m and k are both linear in n, giving an worst case performance of $O(n^3)$. But redistribution of the points can also be done in $O(k + m \log m)$ time by sorting the m points by angle around p. Since all new triangles in the TIN are incident to p, we can distribute the m points over the k triangles by using the sorted order. The modification improves the worst case running time to $O(n^2 \log n)$.

One can expect that k is usually constant, and after a couple of iterations of the algorithm, m will probably be much smaller than n. The more iterations, the smaller m tends to be. One can expect that the algorithm behaves more like the best case than like the worst case, for typical inputs. In the best case, k will be constant, and every list of points stored with a triangle reduces in length considerably each time it is involved in a redistribution. This means that later iterations in the algorithm go faster and faster, since m decreases from linear in n to a constant. If k is assumed to be a constant, we needn't use the modification to distribute the points, but simply spend $O(km) = O(m)$ time. Using an amortized analysis technique, one can show that the whole algorithm will take $O(n \log n)$ time under the assumptions given.

1.4.3 From contour line to TIN

Contour line to TIN conversion algorithms are useful because elevation data is often obtained by digitizing contour line maps. A contour line map is already a vector data structure, in fact, a planar subdivision where the vertices and lines are assigned the elevation of the contour line they are on. To convert the contour lines to a TIN, the obvious thing to do is triangulate all regions, that is, triangulate between the contour lines. Each region can be seen as a polygon with holes, and there are standard triangulation algorithms known for this problem in computational geometry [14, 83, 90].

Instead of using any triangulation it is a good idea to use one that gives nicely shaped triangles, like the Delaunay triangulation. However, the input to the triangulation algorithm is a polygon, not a point set. There exists a triangulation that follows the Delaunay triangulation as closely as possible, given some given set of edges must be present. It is called the *constrained Delaunay triangulation* [8, 19].

In the GIS literature, a couple of approaches to triangulate between contour lines have been described [9, 25, 43, 94]. One of the problems with the constrained Delaunay triangulation and some of the other methods is that they may create horizontal triangles. This side effect of the triangulation is known as the wedding cake effect. It is especially undesirable when visualizing the terrain with the use of hill shading. Several of the known methods avoid such horizontal triangles. Of course the choice of a suitable triangulation comes down to choosing a particular type of interpolation function between the contour lines.

1.5 Mathematical computations on terrains

In many applications it is useful to do things like adding or subtracting the elevation data in two terrains, or squaring the elevation data of a terrain. For example, suppose the data of two terrains represent the height above sea level, and the depth from the surface to the groundwater. Then the subtraction of the latter data set from the former one yields the height of the groundwater above sea level. Similarly, if the depths from the surface to two types of soil data is stored in terrains, then the thickness of the soil in between can be obtained by subtraction.

As an example where it is useful to square and cube terrain data, consider wind erosion [78]. Particles of a certain size can be lifted from the earth's surface by the wind, transported, and deposited again. It has been shown that the detachment capacity of wind varies with the square of the wind velocity, and the transporting capacity with its cube. To model erosion by wind, we need data on wind velocity at the surface, which can be seen as elevation data and modelled by a grid or TIN. Squaring this elevation data gives a model for the detachment capacity of the wind that can be used in further computations and simulations.

1.5.1 Adding and subtracting terrains

Assume that two TINs T_1 and T_2 are given, and we wish to add the elevation data. Subtracting would be the same after placing a minus sign before the elevation data of the terrain to be subtracted. The addition of two TINs can be determined exactly and stored into a new TIN, because the addition of piecewise linear functions (which TINs represent) again yields a piecewise linear function. The addition is done by performing an overlay of T_1 and T_2. There are several algorithms known for computing the overlay [27, 53, 74, 81]. After computing the overlay—the refinement of each of the TINs—we obtain a subdivision where all faces have three, four, five, or six edges. It is trivial to triangulate and obtain a proper TIN again. We now must fill in the height information for the vertices of

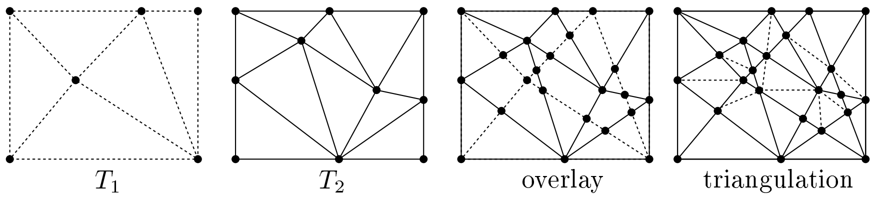


Figure 1.9: The overlay of two TINs and its triangulation.

the overlay. Every vertex originally in T_1 receives its height plus the interpolated height in T_2, and the analogous thing holds for the vertices of T_2. The vertices in the overlay that come from the intersection of two edges are assigned the height that is the sum of the interpolated heights on those two edges.

Note that the overlay of two TINs is a special case of the general subdivision overlay problem. Therefore, simpler algorithms are possible than in the general case. We describe such a simple algorithm.

1. Initialize an empty stack.

2. Take any vertex v of the TIN T_1. Locate its position on T_2 in any way. Unless v coincides with a vertex of v_2, create a vertex record for v in the structure for T_2.

3. Mark v. For each edge e incident to v that is unmarked, do the following.

   - Mark e.
   - Traverse T_2 by following that edge to the other endpoint w in T_1. The edge e is added to T_2 by creating new vertex records for every intersection point with edges of T_2, new edge records for all pieces into which e is partitioned by these intersection points, and new face records for all faces that are split.
   - If the other endpoint w of e is unmarked, create a vertex record for w in T_2, and push w on the stack. Store with the stack element a pointer to the vertex record of w to have fast access into T_2.

4. If the stack is not empty, pop a vertex and call it v. Then proceed at Step 3.

5. Triangulate all faces of T_2.

In the algorithm above, it is assumed that the topological structure of the TINs includes in the vertex records a pointer to the edge record of some incident edge. As we noted, the algorithm creates an overlay with faces that can have up to six incident edges. The topological structure needs an adaptation to incorporate this. The overlay of the two TINs is computed in $O(n+k)$ time, where n is the number of vertices of T_1 and T_2, and k is the number of vertices in the overlay. In theory, k can be as large as $\Omega(n^2)$, but in practice k can be expected to be close to linear in n.

1.5.2 Squaring a terrain

Suppose we want to compute and represent a function in two variables x and y that is the square of another function, represented by a terrain T. The square of T will obviously be a piecewise quadratic function, so a TIN can never represent the square of T without introducing error. The problem we'll discuss is representing the square of a TIN in another TIN but with a guaranteed maximum allowed error ϵ at any point. What would happen if the square of T were computed simply by squaring the elevation of each vertex, and represented by a TIN $\hat{T}$ with the same topological structure? The TIN $\hat{T}$ will always overestimate the true square T^2 of T. The error of $\hat{T}$ as a representation of T^2 is $\max(\hat{T}-T^2)$, maximized over all points (x,y) on the two terrains.

Let's consider one edge of T, where the lower vertex has elevation a and the higher vertex has elevation b. Then $\hat{T}$ will represent this edge as the linear interpolation from a^2 to b^2, whereas the true square of the edge will be a quadratic function from a^2 to b^2. The maximum error over the edge always occurs exactly in the middle of the edge, and the error itself has the value $\frac{1}{4}(b-a)^2$. So the error is not dependent on the position or length of the edge, only on the difference in elevation of the incident vertices. We conclude that the maximum error of $\hat{T}$ always occurs on an edge, and never interior to a triangle.

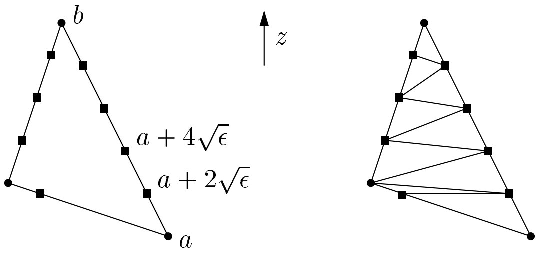


Figure 1.10: Refining a triangle.

Suppose that the maximum allowed error ϵ is given. To compute a TIN $\tilde{T}$ that represents the square of T with error at most ϵ, we'll refine the edges of T so that none of them has an elevation difference more than $2\sqrt{\epsilon}$. For any edge spanning the elevations from a to b, the number of points needed to refine that edge is $\lceil \frac{1}{2}(b-a)/\sqrt{\epsilon} \rceil$. We place these points at $a+2\sqrt{\epsilon}$, $a+4\sqrt{\epsilon}$, $a+6\sqrt{\epsilon}$, and so on, until the last one is at elevation $b-2\sqrt{\epsilon}$ or higher. We do so for every edge of the TIN, and then triangulate every triangle with the additional points as flat as possible. This can be done without introducing any edges that span an elevation more than $2\sqrt{\epsilon}$. Then we square the refined TIN to obtain $\tilde{T}$. From the discussion in the previous paragraph, the error of $\tilde{T}$ is at most ϵ.

1.6 Computation of contour lines

One of the most useful structures that can be obtained from a digital elevation model are the contour lines. Contour lines are probably the most common and natural way to visualize elevation data. Other applications lie in site planning. When a new construction site must be determined, one of the requirements may be that the site lie on an elevation below 1000 meters. Or a spatial query done by a user of a GIS may request all geographic objects of a certain type that have at least a certain elevation. For example, the parliament of a country may consider to partially fund an irrigation system for all crop fields that receive less than 250 mm percipitation annually. To estimate how much this will cost, the total area of these crop fields must be determined. This in turn requires the contour lines of 250 mm on an elevation model representing the annual percipitation.

In this section we use the term *contour line* for one connected set of line segments with a given elevation. We use the plural term *contour lines* for all connected sets of line segments with the given elevation. We next consider two methods for determining contour lines on a TIN. The first method simply scans the TIN to determine the contour lines, while the second method uses preprocessing to be able to find the contour lines more efficiently. This is particularly useful in interactive situations. The last issue treated in this section is the choice of elevations for which the contour lines are selected for display. It is a form of classification.

1.6.1 Direct computation of contour lines

When considering the contour lines on a TIN, observe that all vertices of the contour line lie on edges or vertices of the TIN, and all segments of the contour line lie on triangles or horizontal edges of the TIN. We assume that there are no horizontal triangles on the elevation of which we want the contour line. This can be enforced as follows. Suppose the contour lines of elevation Z are needed, and at some moment a horizontal triangle t with elevation Z is located. Then we only take the edges of t for which the other incident triangles have a vertex higher than Z. This basically comes down to tracing a contour line a very small amount higher than Z. With this enforcement we can from now on forget about whole triangles on the contour line. In a similar way we can forget about complications introduced by saddle points at elevation Z on the TIN (saddle points are vertices that have four or more incident segments of a contour line). If a contour line doesn't contain saddle points, it must be a simple polygon (closed) or a simple polygonal line between two points on the boundary of the TIN.

Given a TIN and an elevation value Z, there is a very simple way to find the contour lines of elevation Z: Traverse the whole TIN and for every triangle, determine if it contains a segment of the contour lines. If so, report it. This algorithm requires $O(n)$ time for a TIN with n triangles. One shortcoming of this algorithm is that it gives the segments on the contour lines in an arbitrary order. Sometimes it is necessary that each contour line be returned as a separate sequence of segments, for instance when smoothing should be performed.

There are two ways to obtain the contour lines in a structured form, as sequences of segments. The first way is by postprocessing the segments that were found by the trivial algorithm. Sort all endpoints of the segments lexicographically on the coordinates. Then all endpoints that are shared among two segments become adjacent in the order. This allows us to structure the separate segments to sequences, each of which is one contour line. If the contour lines together contain k segments, then the postprocessing step takes $O(k \log k)$ time. So in total, the method takes $O(n + k \log k)$ time. Since k is expected to be much smaller than n on real data, proportional to $\sqrt{n}$ is often argued, the overhead of $O(k \log k)$ time is no big deal.

The second way to obtain the contour lines in structured form is by tracing each contour line directly on the TIN. If the TIN is stored in a topological structure like the one described in Subsection 1.2.3, the traversal of one contour line from a starting point can easily be done in time linear in the number of segments of the contour line (there is a small catch if the contour line passes through a vertex of high degree; a possible solution [108] won't be discussed here since it won't be worthwhile in practice). It remains to find all starting points from which to start tracing. If the TIN structure has mark bits stored with the edge records or the triangle records, the following method can be used. Initially all mark bits are reset. For each triangle of the TIN, determine if its mark bit is reset and it contains a segment of the contour lines. If so, start tracing the contour line and set the mark of all triangles that are traversed. The tracing can stop if the boundary of the TIN is reached or a cycle has been completed. After the tracing has stopped, we continue with the next triangle. After all triangles have been tested, all mark bits must be reset again to allow a next request for contour lines. The whole algorithm clearly takes $O(n)$ time. A disadvantage is that mark bits are required in the structure.

1.6.2 Preprocessing for contour lines

The brute-force contour line extraction approach described above is unsatisfactory especially when the number of triangles that cross the elevation Z is much smaller than the total number of triangles in the TIN. A more efficient solution can be obtained in situations where preprocessing is allowed. Then we can build

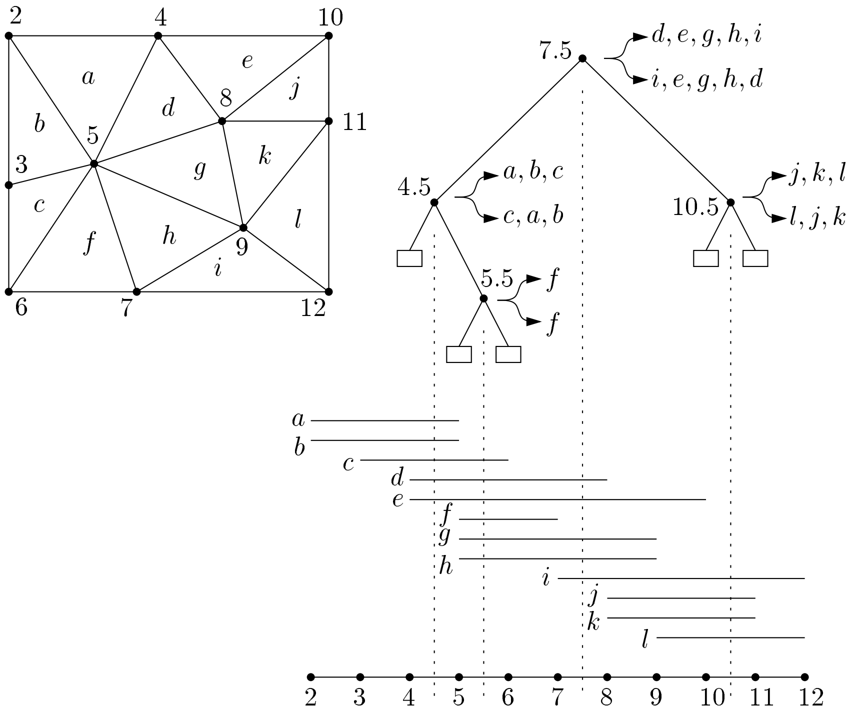


Figure 1.11: An example of a TIN and the corresponding interval tree. The split value and the two lists are shown with each node, where L is the upper list and R the lower list.

a data structure and query with the elevation of which the contour lines are requested. This idea has been the basis of two different approaches, described by De Floriani et al. [34] and by the author of this survey [108]. The method is also used in the visualization of isosurfaces, the higher-dimensional counterpart of contour lines [1, 73].

We describe the *interval tree*, a geometric data structure that stores a set of intervals of the real line. It was developed by Edelsbrunner [22] and McCreight [76] independently. Here we give a brief description—see Figure 1.11.

Let I be a set of open intervals of the form (a, b), where $a, b \in \mathbb{R}$ and $a < b$. The interval tree for I has a root node δ that stores a split value s. Let I_{left} be the subset of intervals (a, b) for which $b \leq s$, let I_{right} be the subset of intervals (a, b) for which $a \geq s$ and let I_δ be the subset of intervals for which $a < s < b$. The subsets $I_{left}, I_{right}, I_\delta$ form a partition of I. The subset I_δ is stored in two linear lists that are associated with node δ. One list L_δ stores I_δ on increasing value of the left endpoint, and the other list R_δ stores I_δ on decreasing value of the right endpoint. If I_{left} is not empty, then the left subtree of δ is defined recursively as an interval tree on the subset I_{left}. The right subtree of δ is defined in a similar way for I_{right}. It follows that any interval of I is stored exactly twice (namely, at one node in two lists). An interval tree for n intervals uses $O(n)$ storage, it can be constructed in $O(n \log n)$ time and if the split values s split roughly balanced, the interval tree has depth $O(\log n)$.

The query algorithm follows one path from the root to a leaf of the tree. Let q be the query value, thus, we want to report all intervals that contain q. At each node δ that is visited, it is determined by comparing q to the split value s stored at δ whether L_δ or R_δ is searched, and in which subtree the query continues. If $q < s$, then we search in the list L_δ and report all intervals that contain the query value. These intervals appear at the start of the list. Therefore, we can traverse L_δ and report intervals until one is reached that doesn't contain q. After searching in L_δ, the query proceeds in the left subtree. If $q > s$, then the list R_δ is searched and the query proceeds in the right subtree. All intervals that contain a query value are reported in $O(\log n + k)$ time, where k is the number of intervals that is reported.

To use an interval tree for our purposes of retrieving the contour lines, note that every triangle of the TIN has a z-span, given by the open interval bounded by the elevation of the vertices of the triangle with lowest and highest elevation. For any query elevation Z between this lowest and highest elevation, the triangle contributes to the contour lines with a line segment on that triangle. The set of z-spans defined by the triangles of the TIN are stored in an interval tree, and with each z-span a pointer to the corresponding triangle record in the TIN structure.

Not only triangles, but also horizontal edges of the TIN can contribute to the contour lines with a line segment. The z-span of a horizontal edge is the closed interval containing a single elevation, the elevation of that edge. The interval tree can easily be adapted to store these closed intervals. Given the query elevation Z, the search in the interval tree retrieves all triangles that lie partially below and partially above Z, and all edges with elevation Z. The line segments of elevation Z on these triangles and edges together form the contour lines for elevation Z. The query time is $O(\log n + k)$, where k is the number of segments in the contour lines.

We conclude that the contour lines of any elevation on a TIN can be found in only $O(\log n + k)$ time, if we are allowed to do preprocessing and use linear additional storage.

The method that was just described computes the contour lines in unstructured format. We continue by considering how the contour lines can be found as sequences of segments, and still use the interval tree to have fast query time. We can combine the methods of the previous subsection with the interval tree just described. By postprocessing the segments of the contour lines, we can get them in structured form in additional $O(k \log k)$ time. This makes the total query time $O(\log n + k \log k)$. This time, the additional term may be significant, because in practical cases the $k \log k$ term is likely to be significantly larger than $\log n + k$.

The method with mark bits can also be combined with the interval tree.

Since we stored with each interval a pointer to the corresponding triangle record in the TIN structure, we have immediate access to start tracing. One subtlety is the following. We cannot reset all mark bits after a query by traversing the whole TIN; that would blow up the query time to linear in n again. Instead, we repeat the query interval tree with the only objective to reset the mark bits. So we trace each contour line on the TIN structure again and reset all mark bits. This will double the query time but no more than that. So, to conclude we have seen that the contour lines of a query elevation Z can be obtained in structured form in $O(\log n + k)$ time, where k is the number of segments in the contour lines.

1.6.3 Classification

Classification is the operation of determining the elevation values of contours that are appropriate for mapping. These elevations that bound regions of the terrain in different classes can be chosen in several different ways. To mention a few, the elevations can be chosen at fixed intervals, such as 0, 500, 1000, 1500, and 2000. The classes induced are: up to 0, from 0 to 500, from 500 to 1000, and so on. One could also classify elevation data by choosing class boundaries using statistical measures, for instance at $\mu - 1.6\sigma$, $\mu - 0.8\sigma$, μ, $\mu + 0.8\sigma$, and $\mu + 1.6\sigma$, where μ is the mean elevation and σ is the standard deviation. A third way of classification is to compute class boundaries such that each class receives an equal amount of area on the map, given the number classes that can be used. Evans gives a good overview of types of classification [3, 24].

Several types of classification make use of the *density function*. It is well-known that a finite population of interval data can be described by a histogram. For continuous interval data, the density function—or frequency distribution—is the corresponding descriptive statistic. It shows how frequent each elevation occurs in the data. We study the computation of the density function of a TIN. Note that it is more appropriate to compute class intervals based on the density function than on the elevations of the vertices of the TIN. These vertices are generally not spread randomly, because large and nearly level regions are represented by only a few vertices. The elevations of these regions would be underrepresented by the set of elevations of all vertices, and an unfair classification would result (see e.g. [24, 56]). The following algorithm to compute the density function and the equal area clasification is by the author of this survey [109].

We begin with a useful observation and a straightforward algorithm. Consider just one triangle Δ in 3-space with vertices u, v, w. Assume for simplicity that $h(u) > h(v) > h(w)$, where $h(..)$ denotes the elevation of a vertex. Then the density on the triangle for a given elevation t is $l \cdot \cos(\alpha)$, where l is the length of the intersection of the triangle Δ with the plane $z = t$, and α is the angle between the normal of Δ and any horizontal plane. The density is zero for all elevations t with $t > h(u)$ or $t < h(w)$. It is given by a function f_{uv} depending linearly on t if $h(u) > t > h(v)$, and it is given by a different function f_{vw} depending linearly on t if $h(v) > t > h(w)$. So we have $f_{uv}(t) = a \cdot t + b$, where a and b depend only on the coordinates of u, v, w and thus are fixed. The same holds for f_{vw}, but with different a and b.

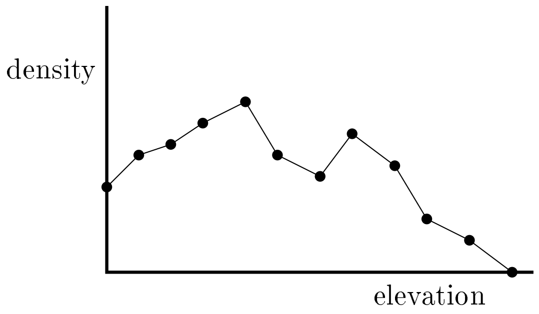


Figure 1.12: Density function of a TIN.

For simplicity of exposition we assume that all vertices have different elevations. This restriction can be overcome without problems, but some care must be taken. Let $v_1, \ldots, v_n$ be the vertices of the TIN, and assume that they are sorted on decreasing elevation. This holds without loss of generality because we can simply relabel the vertices to enforce $h(v_1) > h(v_2) > \cdots > h(v_n)$. Consider the density for an elevation t, where $t \in (h(v_j),\, h(v_{j+1}))$. In such an open interval, the density is the sum of a set of linear functions, which is again a function linear in t. We denote the linear function that gives the density over the whole TIN in the interval $(h(v_j),\, h(v_{j+1}))$ by $F_j(t)$. So the linear functions $F_0, F_1, \ldots, F_n$ form the density function, where each function is only valid in its interval. By default we set $F_0(t) = 0$ and $F_n(t) = 0$ for the intervals $(h(v_1), \infty)$ and $(-\infty, h(v_n))$, because for these elevations the density is zero. One can show that the density function based on a TIN with n vertices is a piecewise linear continuous function with at most $n + 1$ pieces. The density function need not be continuous when there are vertices with the same elevation.

The straighforward algorithm to construct the density function on the TIN is the following. Sort the vertices by elevation, and for each interval $(h(v_j),\, h(v_{j+1}))$, determine the set of linear functions contributing to it. Then add up these linear functions to get one linear piece F_j of the density function. Since we have $O(n)$ vertices, we have $O(n)$ intervals and for each we can easily determine in $O(n)$ time which linear functions contribute. The total time taken by this algorithm is $O(n^2)$.

The efficient computation of the density function is based on the sweeping approach. We will exploit the fact that the linear function F_j can be obtained easily from the linear function F_{j-1} since the contributing f are for the larger part the same ones. We compute the summed linear functions F from top to bottom, which comes down to a sweep with a horizontal plane through the TIN. Throughout the sweep we maintain the density function of the current elevation. Using sweeping terminology, every vertex of the TIN gives rise to one event. The event list is a priority queue storing all these $O(n)$ events in order of decreasing elevation. With each event we store a pointer to the vertex of the TIN that will cause the event. The status structure is trivial: it is simply the summed linear function F for the current position of the sweep plane, and is stored in two reals.

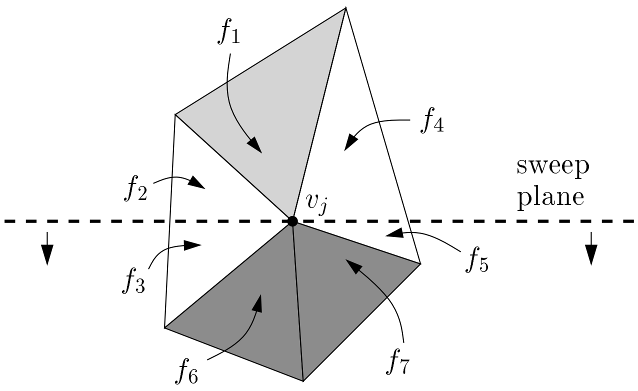


Figure 1.13: Passing a vertex with the sweep plane.

The final ingredient to the sweep algorithm is handling the events. When considering how F_{j-1} should be changed to get F_j when the sweep plane passes the vertex with elevation $h(v_j)$, we must examine how the density function changes. The vertex v_j is incident to some triangles, for which it can be the highest vertex, the lowest vertex or the vertex with middle elevation. We update F_{j-1} to get F_j according to the following rules:

- For all triangles for which v_j is the lowest vertex (lightly shaded in Figure 1.13), we subtract from F the appropriate linear function (f_1 in Figure 1.13).
- For all triangles for which v_j is the highest vertex (darkly shaded), we add to F the appropriate linear function (f_6 and f_7).
- For all triangles for which v_j is the middle vertex (white in the figure), we subtract the one linear function (f_2 and f_4) and add the other (f_3 and f_5).

We don't need to precompute or store the linear functions f on each triangle to update F; the f can be obtained from the coordinates of the vertices on the TIN when the event at vertex v_j is handled. We have fast access to vertex v_j in the TIN; recall that an extra pointer was stored in the event list.

We also evaluate the function F_j at the event. The sequence of evaluations gives the breakpoints of the (piecewise linear) density function. These breakpoints are computed from right to left in Figure 1.12 since the sweep goes from high to low elevations.

Considering the efficiency of the algorithm, the initial sorting of the events takes $O(n \log n)$ time for a TIN with n vertices. Extraction of an event takes

$O(\log n)$ time; for all events this adds up to $O(n \log n)$ time. Updating the status structure at an event v_j requires time linear in the number of triangles incident to v_j. Summed over all vertices this is linear in n by Euler's formula. The evaluation to determine the breakpoints requires constant time per event. So in total the sweep algorithm requires $O(n \log n)$ time.

Once the density function is computed, the class intervals may be determined. Suppose as an example that the objective is to determine seven classes such that each class occupies an equivalent amount of area on an contour line map. We assume that the contour line map and the TIN have the same domain, otherwise we can clip the TIN with the domain of the contour line map before doing the sweep. The total area of the contour line map is the same as the total area under the density function and is denoted A. The area under the density function in the elevation interval $[a, b]$ is denoted $A(a, b)$. If $F(t)$ denotes the (piecewise linear) density function, then

$$A(a,b) = \int_a^b F(t)dt$$

The value of $A(a, b)$ is exactly the area for the class $[a, b]$ on the contour line map. We know the total area A and compute $A/7$, the desired area for each class. We then determine the lowest elevation such that $A/7$ of the area is below that elevation. This operation is easy by scanning over the known density function $F(t)$ from left to right and maintaining the area under $F(t)$ (this is also a kind of sweep). This gives the lowest class boundary. Continuing the scan gives all six boundaries of the seven classes in $O(n)$ time. In a similar way one can compute a non-fixed number of classes with the property that the within-class variance is less than or equal to a certain threshold, for each class. Finally, the density function can be used class interval selection by natural breaks in the data: They are the local minima of $F(t)$. We refer to Burrough [3] and Evans [24] for other classification schemes.

The sweep algorithm that was described for the density function requires linear working storage to store all the events. For most realistic terrains, the working storage can be reduced considerably. We make the following simple observation. Every vertex except the local maxima—the peaks—have a higher neighbor in the TIN. So we can initialize the event list with the local maxima only. When the event at a vertex v is handled, we insert all lower neighbors of v in the event list. This guarantees that every event is present in the event list when the sweep plane reaches it. The storage required by the algorithm is linear in the sum of the number of local maxima and the number of edges in the largest complexity cross-section.

1.7 Topographic features

Geomorphologists study the shape of the land, and what processes influence it [?, ?]. The quantification of the shape of the land is necessary in order automatically recognize certain features of shape. This on its turn may lead to a partition of the land into regions where for instance erosional processes have the same behavior.

Terrain features can be zero-, one-, or two-dimensional. We discuss the most important ones in the next subsections. Then we treat slope and aspect defined on a terrain.

1.7.1 Points on terrains

Any point on a terrain has a certain elevation. When we also consider the neighborhood of a point, the slope and aspect at it can be defined. The slope (also called gradient) at a point is the maximum ratio of change in elevation and change of position in the xy-plane at that point. Mathematically, it is the maximum value of the directional derivative at that point (maximized over the direction). Note that the slope of an elevation model is an elevation model itself. Therefore it can be visualized, for example, as a contour line map by classification of the slopes.

The aspect or exposure of a point in an elevation model is the compass direction in which the directional derivative is maximum. With an aspect map it is easy to see which hill sides face to the south, for instance. The aspect of an elevation model is not an elevation model. Instead, it is a bivariate function that maps $\mathbb{R} \times \mathbb{R}$ to the circular scale $(-\pi, \pi]$. It is undefined at points that lie on a horizontal part of the terrain. The combination of slope and aspect is needed to produce hill shading on maps.

On a terrain there are certain special points that are more important or characteristic than others. These are the peaks, the pits, and the passes. The latter are also called saddles. A peak is a point such that in some neighborhood of it, there is no higher point. Similarly, in some neighborhood of a pit there is no lower point. A pass is a point where locally, four (or more) different parts of the contour lines meet. These definitions don't specify what neighborhood should be taken, and what should be considered a peak when there is a whole region of equal elevation points. Choices of this type have to be taken depending on the application.

Peaks, pits, and passes are elements that are used to describe terrain form. They are the basis of so-called surface networks and their relatives [88, 111, 116]. One such relative, the Warntz network, can be obtained by first identifying all passes, and from there, traverse the terrain in the directions of steepest ascent and steepest descent until peaks or pits are reached. The paths traversed together define a partition of the terrain into regions, of which one can hope that they have similar geomorphological features. What should be considered a pass for this idea to work well has considerable influence on the output [115].

1.7.2 Valleys and ridges

Valleys on a TIN are 1-dimensional features. They are usually defined as the edges for which the two incident triangles each have an outward normal vector whose vertical projection on the xy-plane is directed towards the valley edge, when it is also projected vertically [39, 105]. There are potential problems with

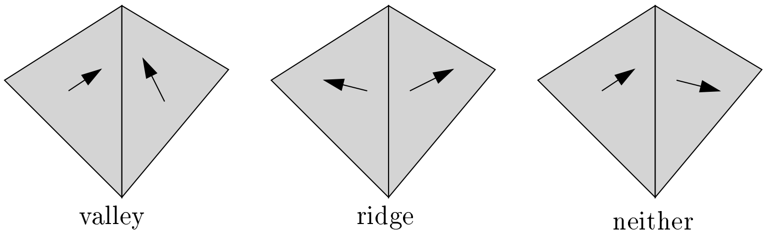


Figure 1.14: Three times two adjacent triangles and their outward normal vectors.

this definition if the outward normal vectors are parallel to the edge, in the projection. We could define an edge to be a valley edge if at least one incident triangle has its outward normal towards the edge, and the other one has its outward normal towards the edge or parallel to it. A similar definition can be made for ridge edges, where the outward normals lead away from the edge.

All of these definitions have the disadvantage that valley lines may be interrupted, even though the valley itself seems to be just one feature on the terrain. When we discuss drainage networks and basins, we'll see an alternative definition that can be used for valleys and ridges and avoids interruption as much as possible.

1.7.3 Curvature

The two-dimensional terrain specific features are obtained by considering *plan curvature* and *profile curvature*, being the curvature in a horizontal and vertical cross-section of the terrain. These curvatures specify whether the terrain is convex, flat, or concave in the cross-section. For smooth surfaces the convexity depends on the sign of the second derivative in the cross-section. One could partition the terrain into regions where the two curvatures are within certain boundaries, a type of classification [26, 44, 84, 102]. For instance, for profile convexity the bounds $-0.1\,^\circ/\mathrm{m}$ and $+0.1\,^\circ/\mathrm{m}$ are used. The terrain elements as in Figure 1.15 are the ones that can be obtained in such a classification. Profile curvature is related to the position on a hillside. Most hillsides are profile convex near the top and profile concave near the foot.

On gridded elevation models the curvature of a pixel can be determined by considering the 3×3 window of pixels, choosing a suitable interpolator for the nine pixels, and computing the plan and profile curvature of the interpolated surface at the center [44, 84].

On TINs a simple approach has been suggested: every edge of the TIN can be seen as flat, convex, or concave. Then triangles can be classified according to the type of incident edges. A convex region is one consisting only of triangles of which all incident edges are convex. A similar statement can be made for concave

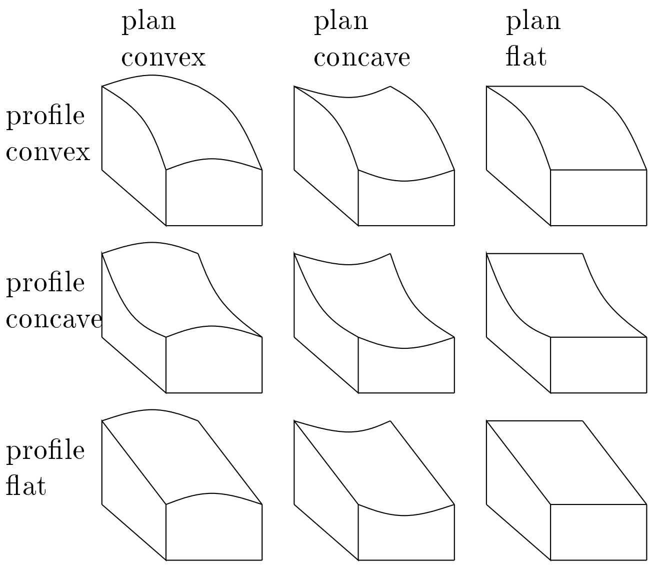


Figure 1.15: Nine landform elements classified by plan and profile curvature.

and flat regions. The triangles that are incident to triangles of different types are defined as saddle triangles. These definitions allow the regions of uniform curvature to be traced out on the TIN by straightforward graph traversal, in linear time [26].

A disadvantage of the TIN approach above is that it doesn't distinguish between plan and profile curvature. A region that is profile convex and plan concave is defined saddle, and so is a region that is profile concave and plan convex. It is possible to obtain a TIN curvature classification that includes plan and profile curvature. We define the plan curvature at a vertex v as follows. If v is a peak, the plan curvature is convex. If v is a pit, the plan curvature is concave. If v is a saddle, the plan curvature is undefined. In all other cases, the plan curvature of v is determined by the contour line through v. Vertex v is incident to two line segments s and s' on that contour line (usually across TIN triangles). When traversing the contour line with the higher terrain to the left and the lower terrain to the right, then v is plan convex if the contour line makes a left turn at v. If it makes a right turn, v is plan concave, and if it makes no turn (or a turn below some threshold), then it is plan flat. Note that saddle vertices have to be excluded because four line segments of the contour line meet at a saddle vertex.

We define the profile curvature at a vertex v as follows. If v is a peak, the profile curvature is convex. If v is a pit, the profile curvature is concave. If v is a saddle, the profile curvature is undefined. To define the profile curvature at another vertex v we must select a suitable vertical plane through v. Consider again the line segments s and s' on the contour line through v. We take the

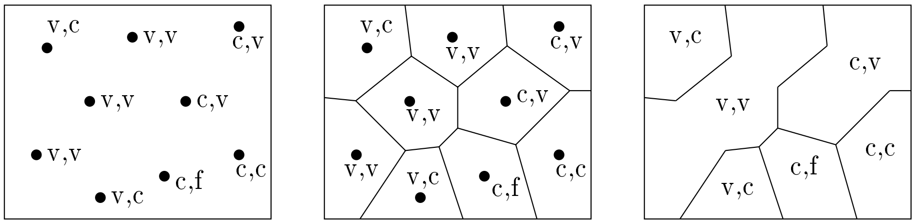


Figure 1.16: Vertices labelled with their plan and profile curvature, where v is used for con**v**ex, c is used for con**c**ave and f is used for **f**lat. The Voronoi diagram is shown in the middle, and boundaries of similar regions are erased to obtain a terrain partition.

vertical plane through v that separates s and s', and makes an equivalent angle with them. This plane is in a sense perpendicular to the tangent of the contour line at v, and therefore a reasonable choice. Next we consider the intersection of the vertical plane with the terrain at v, which is the profile. We can define concexity, concavity, and flatness in the profile in the obvious way.

Now we know for most of the TIN vertices their plan and profile curvature, see Figure 1.16. To obtain regions from this information we can use interpolation. For any point on a triangle or edge, we define its plan and profile curvature to be the same as the curvature of the nearest vertex (not a saddle). This nearest-neighbor interpolation approach induces a Voronoi diagram on the TIN vertices, excluding the saddle vertices. Adjacent regions that have the same curvature labels can be merged by erasing their common boundary. The result is a terrain partition into regions of uniform curvature both in plan and profile. Since the Voronoi diagram of n points can be computed in $O(n \log n)$ time [14, 83, 90], the terrain partition requires $O(n \log n)$ time to compute as well.

The approach can be supplemented with a scale-dependent parameter. For example, consider the plan curvature at a vertex v again. Instead of looking at the angle of the line segments on the contour line incident to v, we may locate two points p and p' on this contour line at a certain distance from v. This distance is the scale-dependent parameter. Then we determine the angle $\angle pvp'$, and decide upon the plan curvature. This refined approach may for instance cause small concavities in a convex region to be eliminated.

1.7.4 Drainage information

Any terrain induces a more or less natural flow of water on it. For instance, water always flows downward, following the direction of gravity, and water collects into streams. These streams join and form rivers. The more downstream, the bigger a river becomes. It is possible to predict from an elevation model where the streams will be. The collection of all streams and rivers is called the *drainage network*. In this section we only consider how the form of the terrain influences

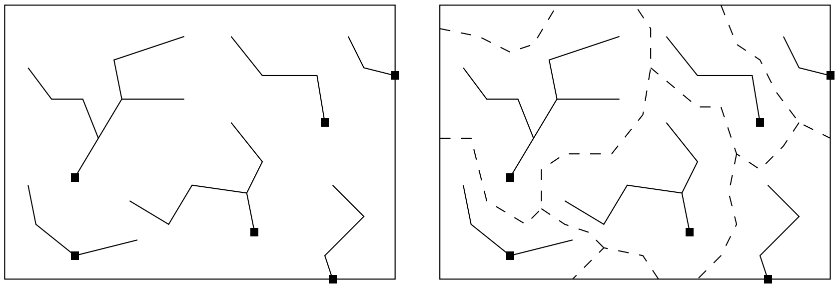


Figure 1.17: Left, the drainage network as a forest of trees rooted at the pits. Right, the basins of each river system.

the flow of water, and review a few possible definitions of the drainage network and related concepts. To find a definition that corresponds to the drainage network in reality is a problem that requires various types of data of the terrain. The area of hydrology also includes issues like surface permeability, subsurface flow, evaporation, and more [**?**, 78].

Generally, the drainage network can be seen as a group of connected acyclic networks (a forest of trees on the graph sense) of which the links are directed to the pits of the terrain, see Figure 1.17. Each connected network is also called a *river system*, and the part of the terrain that drains into some river system is called a *drainage basin* (or *basin*) of that system.

The drainage network on a grid

One of the first attempts to compute the drainage network on a terrain was by Peucker and Douglas [86]. Their algorithm works on a grid is extremely simple: slide a 2×2 window over the grid and flag the highest pixel in the window. After all subwindows have been treated, the unflagged pixels together form the drainage network. The maps produced by this method suffer from isolated dots and interrupted channels.

A more advanced approach was taken by O'Callaghan and Mark [82] and Mark [75], who also modelled the *accumulation* of water flowing in the terrain (they credit Speight [102] for this idea). Define for every pixel the drain neighbor to be one of the eight neighboring pixels to which the steepest descent is greatest. This drain neighbor is assumed to be unique. A pit doesn't have a descent direction and therefore no drain neighbor. Then assign every pixel one unit of water, and trace all units on the grid downward to the drain neighbors until they end in the pits. By maintaining counters to determine for every pixel how many units of water flow through it, the drainage network can be defined. It consists of all pixels for which the counter is higher than some well-chosen threshold. By treating the pixels in order of decreasing elevation, the method requires $O(n^2 \log n)$ time on an $n \times n$ grid (needed for the sorting). The method also requires quadratic additional storage.

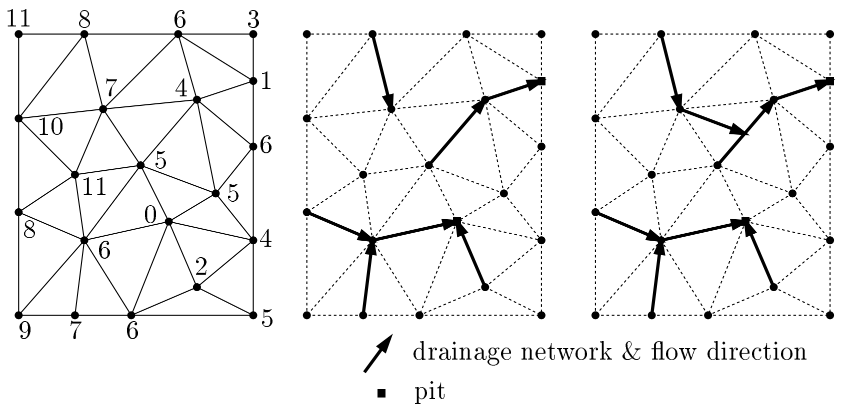


Figure 1.18: Left, a TIN with elevations of the vertices. Middle, the drainage network by the definition of Frank et al. Right, the drainage network by the definition of Yu et al.

The accumulation idea solves the problem of interrupted channels. If some pixel belongs to the drainage network because its counter exceeds the threshold, then the whole path along drain neighbors to a pit must also be part of the drainage network. The accumulation idea also helps to define drainage basins. Since the path from any pixel can be traced to a pit, it is possible to determine what pixels drain into any pit. So it is possible to outline the basins, the parts of the terrain drain into one single pit.

The drainage network on a TIN

On TINs, a definition of the drainage network has been suggested by Frank et al. [39]. They define the drainage network to consist of all valley edges of the TIN. This definition suffers from the possible interruption of streams, which can end in points other than pits, as observed by Theobald and Goodchild [105]. See for instance Figure 1.18. Furthermore, there is no concept of flow, so basins cannot be defined as an extension of the model for the drainage network.

Yu et al. [118] showed recently that the idea of accumulation can be applied to TINs as well. On a TIN, there are no pixels to assign units of water to, and it isn't a good idea either to assign water to complete triangles. But the direction of flow can still be defined conveniently of a point on a TIN as the direction of steepest descent. In the interior of the triangles this direction is unique, but on edges and vertices we may need to choose a direction if there is more than one direction of steepest descent. Once the direction of flow is defined, flow paths can be traced and one can discover where flow paths join. It is natural to define for each point on the terrain the area of the region from which the flow paths go through that point. For many points on the TIN, this area is zero because there is no 2-dimensional region. But there are also points that receive water from a region with positive area. Define the drainage network to be those points on the terrain that receive water from a region whose area exceeds a certain threshold area. We'll study the case where the threshold is set to 0. It is clear that the drainge network we obtain will include all drainage networks for larger threshold values.

One can show a number of properties of the drainage network defined this way. Most importantly, the drainage network consists of all valley edges, and furthermore, exactly of all flow paths from their lower vertices. This follows from the fact that water can only start accumulating at valley edges. The drainage network will have merge points where two or more streams join and continue together. These merge points are either vertices of the TIN, or points on valley edges. Since we assumed that at every point of the TIN the direction of flow is unique, streams cannot split.

When comparing the definitions of Frank et al. and Yu et al. we can easily observe that under the former definition, the drainage network has complexity at most linear in the number of edges of the TIN. It is considerably less obvious what the size of the drainage network is under the second definition. De Berg et al. [13] have shown that it is at most cubic in the number of edges of the TIN, and that the cubic bound is tight for some artificially constructed TINs. Whether the cubic worst case bound has any relevance in practice is doubtful. An emperical study on this issue has been done and the actual size on real terrain data appears to be roughly 20% more than under the definition by Frank et al. The tests were done on six different terrains represented by TINs with up to 12,000 vertices [106].

To compute the drainage network by the definition of Yu et al., we first identify all valley edges, and then follow the flow paths of their lower vertices to the pits. Since flow paths can merge, we can stop tracing any flow path from a point where another flow path has already gone through. So, whenever a flow path is traced, it is marked on the terrain itself to make sure that the same flow path isn't traced again and again. The resulting algorithm requires time $O(n + k)$, where k is the complexity of the drainage network.

Drainage basins, catchment areas, generalization, and spurious pits

The study of drainage on a terrain using the definition of Yu et al. [118] can be extended in various ways. Since it incorporates the notion of flow and accumulation, it becomes possible to determine the basins of the different river systems, and also the area of the terrain that drains into each river system. For any point on the terrain, we define the *catchment area* to be the part of the terrain that drains through that point, eventually. The definition above of the drainage network includes exactly the points that have a catchment area that is 2-dimensional, no matter how small the area. As a consequence, many small streams will be included in the drainage network.

We can also define a *generalized* drainage network by selecting the points of which the catchment area has at least a certain size. Since it is possible to determine the size of the catchment area for each point on the terrain—in particular, on the original drainage network—we can also compute the generalized drainage network. Other methods to compute the generalized drainage network include using stream orders to decide which streams can be omitted, or generalizing the terrain and then computing the drainage network [**?**, 113].

Terrains that don't cover a large area of land usually don't have many pits on them. When a drainage network algorithm detects a number of pits, some of them will actually be the result of imprecision in the data acquisition, or errors. Other pits that occur often are not the end of a river system, but lakes may start to form that overflow into another river system, making the two connected. Avoiding pits on a terrain is also called *drainage enforcement* or *spurious pit removal* [57, 58, 75]. For every pit, there is one pass where water will overflow first when the pit is filled. The overflowing water may start a new stream that joins some other river system. Which pits should be removed can be based on the pit perimeter, the elevation difference between pit and pass, or the lake capacity. The pit perimeter is the length of the polygon that lies on the contour line through the pass and containing the pit.

1.8 Miscellaneous applications

In this section we'll mention some other applications and algorithms on terrains with references. It is meant rather as an annotated bibliography than as a survey.

1.8.1 Paths in terrains

Planning routes or networks in mountainous regions is of interest to civil engineers. Problems like determining the best road to connect two places on a terrain, best location of a bridge over a valley, and the like are sometimes addressed in a geographic information system. There can be several optimization criteria, like minimizing the length of a route, finding a route that stays as low as possible, with minimum cost for construction, and optimizing the resulting travel time. Algorithms for abstract versions of these problems have been given mostly in the computational geometry literature [6, 17, 79, 80, 107, **?**].

1.8.2 Viewshed analysis

Viewshed analysis is the general name for visibility problems on terrains. The standard viewshed analysis problem is simply the question: "What parts of the terrain are visible from a specific point?" Optimization problems of visibility are minimization of visibility (horizon pollution of planned buildings, routes not visible from enemies) and maximization of visibility (observation posts for fire detection, scenic routes). Viewshed problems have been studied in a theoretical setting, but also in more practical situations [32, 69, **?**, 97, 101, 119].

As a measure for the area of visible regions on a terrain, the *visibility index* can be used on grid models. The visibility index of a given pixel is the number of other pixels that are visible from the given pixel [41, 104, 109].

The computation of horizons is related to viewshed analysis. One can make a distinction between visibility above the last horizon (against the sky), visibility agains a local horizon, and visibility against a hillside, and one can also compute values for non-visible points that represent the elevation increase needed to make the point visible [29, 30].

Some other issues of visibility on terrains include moving points of view, networks of interconnected sites, and the error present in visibility analyses [2, 12, 28, 36].

1.8.3 Temporal aspects of terrains

Modelling time in a geographic information system is a topic that has received a lot of attention recently [66]. The mapping of time can be one of the themes on a static map, but it is also possible to use dynamic or animated maps for the visualization. Here a sequence of maps of the same area and the same theme is used, where each map represents the situation at a fixed moment in time. The sequence as a whole can be used to animate the changes in the mapped themes over time [21, 61, 62, 64]. An example is moving pressure fronts in weather reports. This example shows that temporal aspects and animation can be issues for elevation data as well.

Suppose that a sequence of terrains is given, each representing the terrain at a fixed moment in time. To animate the change of the terrain one could show the terrain in perspective view or by contour lines. Since the data usually is available only at a discrete set of moments, interpolation between two terrains at consecutive moments becomes necessary. If the terrains are represented by TINs, the problem comes in different forms. Firstly, it can be the case that the sequence of TINs has the same underlying set of vertices and triangulation. The only differences are the elevations of the vertices. The second—and more difficult—version of the problem has different vertex sets or triangulations for the terrains in the sequence. It can be important to use a dynamic terrain model that allows for the addition and removal of vertices and edges on the TIN [48, 55].

One more use of TINs in geographic processing is the simulation of physical processes the influence certain terrain features. For example, one can study drainage on a terrain after storms and rain showers, and analyze how long it will take before the default drainage situation is restored [99].

1.8.4 Statistical analysis of elevation data

Statistical analysis of elevation data obtained at point samples is common in the earth sciences. The field is also called geostatistics [**?**, **?**, 56, 112]. Important concepts include spatial interpolation, summarizing the data by mean, standard deviation, skewness, auto-correlation, and so on. One of the most important purposes of the analysis is the prediction of data at places where no measurements were made.

1.9 Conclusions

This survey has explained a number of concepts on terrains, and some algorithms for various computations. The emphasis has been on TIN algorithms, because the TIN model for terrains is more elegant than the grid and contour line models. A common argument to use grids is the simplicity of the algorithms. However, the current trends in GIS research and in the field of computational geometry have shown that algorithms on TINs need not be difficult either. More programming effort is required, but this need not outweigh the advantages that TINs have to offer. We won't repeat arguments in the raster-vector debate; a summary of algorithmic methods and specific algorithms for TINs is useful in any case. The search for efficient algorithms on terrains is an interesting area of research where the GIS developers, GIS researchers, and computational geometers can work together to develop a variety of elegant and efficient solutions to practical problems on terrains. The analysis of efficiency of these solutions should be based on realistic assumptions on terrains.

Bibliography

[1] C.L. Bajaj, V. Pascucci, and D.R. Schikore. Fast isocontouring for improved interactivity. In *Proc. IEEE Visualization*, 1996.

[2] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman. Visibility with a moving point of view. *Algorithmica*, 11:360–378, 1994.

[3] P. A. Burrough. *Principles of Geographical Information Systems for Land Resourses Assessment.* Oxford University Press, New York, 1986.

[4] J.R. Carter. The effect of data precision on the calculation of slope and aspect using gridded DEMs. *Cartographica*, 29:22–34, 1992.

[5] K.-T. Chang and B.-W. Tsai. The effect of DEM resolution on slope and aspect mapping. *Cartography and Geographic Information Systems*, 18:69–77, 1991.

[6] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1990.

[7] Z.-T. Chen and J.A. Guevara. Systematic selection of very important points (vip) from digital terrain model for constructing triangular irregular networks. In *Proc. Auto-Carto 8*, pages 50–56, 1987.

[8] L. P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[9] A.H.J. Christensen. Fitting a triangulation to contours. In *Proc. Auto-Carto 8*, pages 57–67, 1987.

[10] K. C. Clarke. *Analytical and Computer Cartography.* Prentice Hall, Englewood Cliffs, 2nd edition, 1995.

[11] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[12] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *J. Symbolic Comput.*, 7:11–30, 1989.

[13] M. de Berg, P. Bose, K. Dobrint, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink, and S. Yu. The complexity of rivers in triangulated terrains. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 325–330. Carleton University Press, Ottawa, Canada, 1996.

[14] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry – Algorithms and Applications.* Springer-Verlag, Berlin, 1997. to appear.

[15] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. In *Proc. 3rd ACM Workshop on Advances in GIS*, pages 77–84, 1995.

[16] Mark de Berg and Katrin Dobrindt. On levels of detail in terrains. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C26–C27, 1995.

[17] Mark de Berg and Marc van Kreveld. Trekking in the alps without freezing or getting tired. In *1st Annual European Symposium on Algorithms (ESA '93)*, volume 726 of *Lecture Notes in Computer Science*, pages 121–132. Springer-Verlag, 1993.

[18] L. De Floriani. A pyramidal data structure for triangle-based surface representation. *IEEE Comput. Graph. Appl.*, 9:67–78, March 1989.

[19] L. De Floriani and E. Puppo. A survey of constrained Delaunay triangulation algorithms for surface representaion. In G. G. Pieroni, editor, *Issues on Machine Vision*, pages 95–104. Springer-Verlag, New York, NY, 1989.

[20] Leila De Floriani, Bianca Falcidieno, George Nagy, and Caterina Pienovi. Hierarchical structure for surface approximation. *Comput. Graph. (UK)*, 8(2):183–193, 1984.

[21] D. DiBiase, A.M. MacEachren, J.B. Krygier, and C. Reeves. Animation and the role of map design in scientific visualization. *Cartography and Geographic Information Systems*, 19:201–214, 1992.

[22] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Report F59, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1980.

[23] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[24] I. S. Evans. The selection of class intervals. *Trans. Inst. Br. Geogrs.*, 2:98–124, 1977.

[25] B. Falcidieno and C. Pienovi. A feature-based approach to terrain surface approximation. In *Proc. 4th Int. Symp. on Spatial Data Handling*, pages 190–199, 1990.

[26] B. Falcidieno and M. Spagnuolo. A new method for the characterization of topographic surfaces. *Int. J. of GIS*, 5:397–412, 1991.

[27] Ulrich Finke and Klaus Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 119–126, 1995.

[28] P. F. Fisher. Algorithm and implementation uncertainty in viewshed analysis. *Internat. J. Geogr. Inform. Syst.*, 7:331–347, 1993.

[29] P. F. Fisher. Stretching the viewshed. In *Proc. 6th Internat. Sympos. Spatial Data Handling*, pages 725–738, 1994.

[30] P. F. Fisher. Reconsideration of the viewshed function in terrain modelling. *Geogr. Syst.*, 3:33–58, 1996.

[31] P.-O. Fjällström. Polyhedral approximation of bivariate functions. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 187–190, 1991.

[32] L. De Floriani, B. Falcidieno, C. Pienovi, D. Allen, and G. Nagy. A visibility-based model for terrain features. In *Proc. 2nd Int. Symp. on Spatial Data Handling*, pages 235–250, 1986.

[33] L. De Floriani, P. Marzano, and E. Puppo. Hierarchical terrain models: survey and formalization. In *Proc. ACM Symp. on Applied Computing*, 1994.

[34] L. De Floriani, D. Mirra, and E. Puppo. Extracting contour lines from a hierarchical surface model. In *Eurographics'93*, volume 12, pages 249–260, 1993.

[35] L. De Floriani and E. Puppo. A hierachical triangle-based model for terrain description. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, proceedings*, volume 639 of *Lecture Notes in Computer Science*, pages 236–251, Berlin, 1992. Springer-Verlag.

[36] L. De Floriani, E. Puppo, and G. Nagy. Computing a line-of-sight network on a terrain model. In *Proc. 5th Int. Symp. on Spatial Data Handling*, pages 672–681, 1992.

[37] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.

[38] R. J. Fowler and J. J. Little. Automatic extraction of irregular network digital terrain models. *Comput. Graph.*, 13(2):199–207, August 1979.

[39] A.U. Frank, B. Palmer, and V.B. Robinson. Formal methods for the accurate definition of some fundamental terms in physical geography. In *Proc. 2nd Int. Symp. on Spatial Data Handling*, pages 583–599, 1986.

[40] Wm Randolph Franklin. Compressing elevation data. In *Advances in Spatial Databases (SSD'95)*, number 951 in Lecture Notes in Computer Science, pages 385–404, Berlin, 1995. Springer-Verlag.

[41] Wm. Randolph Franklin and C. K. Ray. Higher isn't necessarily better: Visibility algorithms and experiments. In *Proc. 6th Internat. Sympos. Spatial Data Handling*, pages 751–763, 1994.

[42] Wm Randolph Franklin and A. Said. Lossy compression of elevation data. In *Proc. 7th Int. Symp. on Spatial Data Handling*, pages 8B.29–8B.41, 1996.

[43] A.B. García, C.G. Nicieza, J.B.O. Meré, and A.M. Díaz. A contour line based triangulation algorithm. In *Proc. 5th Int. Symp. on Spatial Data Handling*, pages 411–421, 1992.

[44] P.K. Garg and A.R. Harrison. Quantitative representation of land-surface morphology from digital elevation models. In *Proc. 4th Int. Symp. on Spatial Data Handling*, pages 273–284, 1990.

[45] M. Garland and P.S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[46] C. Gold. The practical generation and use of geograhic triangular element data. In *Harvard Papers on Geographic Information Systems*, volume 5. 1978.

[47] C. Gold and S. Cormack. Spatially ordered networks and topographic reconstructions. In *Proc. 2nd Int. Sympos. Spatial Data Handling*, pages 74–85, 1986.

[48] C. Gold and T. Roos. Surface modelling with guaranteed consistency – an object-based approach. In *IGIS'94 Geographic Information Systems*, number 884 in Lecture Notes in Computer Science, pages 70–87, Berlin, 1994. Springer-Verlag.

[49] C. M. Gold, T. D. Charters, and J. Ramsden. Automated contour mapping using triangular element data structures and an interpolant over each irregular triangular domain. *Comput. Graph.*, 11(2):170–175, 1977.

[50] C.M. Gold. Neighbours, adjacency and theft – the Voronoi process for spatial analysis. In *Proc. 1st Eur. Conf. on Geographical Information Systems*, 1990.

[51] C.M. Gold and U. Maydell. Triangulation and spatial ordering in computer carthography. In *Proc. Canad. Cartographic Association Annual Meeting*, pages 69–81, 1978.

[52] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[53] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[54] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[55] M. Heller. Triangulation algorithms for adaptive terrain modeling. In *Proc. 4th Int. Symp. on Spatial Data Handling*, pages 163–174, 1990.

[56] E. H. Isaaks and R. M Srivastava. *An Introduction to Applied Geostatistics.* Oxford University Press, New York, 1989.

[57] S.K. Jenson. Automated derivation of hydrologic basin characteristics from digital elevation model data. In *Proc. ASP/ACSM*, pages 301–310, 1985.

[58] S.K. Jenson and C.M. Trautwein. Methods and applications in surface depression analysis. In *Proc. Auto-Carto 8*, pages 137–144, 1987.

[59] C.B. Jones, J.M. Ware, and G.L. Bundy. Multiscale spatial modelling with triangulated surfaces. In *Proc. 5th Int. Symp. on Spatial Data Handling*, pages 612–621, 1992.

[60] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[61] A. Kousoulakou and M.-J. Kraak. Spatio-temporal maps and cartographic communication. *Cartographic Journal*, 29:101–108, 1992.

[62] M.-J. Kraak and A.M. MacEachren. Visualization of the temporal component of spatial data. In *Proc. 6th Int. Symp. on Spatial Data Handling*, pages 391–409, 1994.

[63] M.-J. Kraak and F.J. Ormeling. *Cartography – visualization of spatial data.* Longman, Harlow, 1996.

[64] M.-J. Kraak and E. Verbree. Tetrahedrons and animated maps in 2d and 3d space. In *Proc. 5th Int. Symp. on Spatial Data Handling*, pages 63–71, 1992.

[65] I.S. Kweon and T. Kanade. Extracting topographic terrain features from elevation maps. *CVGIP: Image Understanding*, 59:171–182, 1994.

[66] G. Langran. *Time in Geographic Information Systems.* Taylor & Francis, London, 1992.

[67] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems.* Academic Press, Boston, MA, 1992.

[68] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *Internat. J. Comput. Inform. Sci.*, 9:219–242, 1980.

[69] J. Lee. Analyses of visibility sites on topographic surfaces. *Internat. J. Geogr. Inform. Syst.*, 5:413–430, 1991.

[70] J. Lee, P.K. Snyder, and P.F. Fisher. Modelling the effect of data errors on feature extraction from digital elevation models. *Photogrammatic Engineering and Remote Sensing*, 58:1461–1467, 1993.

[71] Jay Lee. A drop heuristic conversion method for extracting irregular network for digital elevation models. In *GIS/LIS '89 Proc.*, volume 1, pages 30–39. American Congress on Surveying and Mapping, November 1989.

[72] Jay Lee. Comparison of existing methods for building triangular irregular network models of terrain from grid digital elevation models. *Internat. J. Geogr. Inform. Syst.*, 5(3):267–285, July-September 1991.

[73] Y. Livnat, H.-W. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2:73–84, 1996.

[74] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume F40 of *NATO ASI*, pages 307–325. Springer-Verlag, Berlin, West Germany, 1988.

[75] D.M. Mark. Automated detection of drainage networks from digital elevation models. *Cartographica*, 21:168–178, 1984.

[76] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14:257–276, 1985.

[77] A. M. J. Meijerink, H. A. M. de Brouwer, C. M. Mannaerts, and C. R. Valenzuela. *Introduction to the Use of Geographic Information Systems for Practical Hydrology.* Number 23 in ITC Publications. ITC, Enschede, 1994.

[78] C. Mitchell. *Terain Evaluation.* Longman, Harlow, 2nd edition, 1991.

[79] J. S. B. Mitchell. An algorithmic approach to some problems in terrain navigation. *Artif. Intell.*, 37:171–201, 1988.

[80] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.

[81] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25:739–747, 1982.

[82] J.F. O'Callaghan and D.M. Mark. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing*, 28:323–344, 1984.

[83] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, New York, 1994.

[84] D.J. Pennock, B.J. Zebarth, and E. de Jong. Landform classification and soil distribution in hummocky terrain, Saskatchewan, Canada. *Geoderma*, 40:297–315, 1987.

[85] T.K. Peucker. Data structures for digital terrain modules: Discussion and comparison. In *Harvard Papers on Geographic Information Systems*, volume 5. 1978.

[86] T.K. Peucker and D.H. Douglas. Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Vision, Graphics, and Image Processing*, 4:375–387, 1975.

[87] T.K. Peucker, R.J. Fowler, J.J. Little, and D.M. Mark. The triangulated irregular network. In *Proc. DTM Symp. Am. Soc. of Photogrammetry—Am. Congress on Survey and Mapping*, pages 24–31, 1978.

[88] J.L. Pfaltz. Surface networks. *Geographical Analysis*, 8:77–93, 1976.

[89] F. P. Preparata and D. E. Muller. Finding the intersection of n half-spaces in time $O(n \log n)$. *Theoret. Comput. Sci.*, 8:45–55, 1979.

[90] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[91] M. Sambridge, J. Braun, and H. McQueen. Geophysical parameterization and interpolation of irregular data using natural neighbours. *Geophys. J. Int.*, 122:837–857, 1995.

[92] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[93] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986.

[94] L. Scarlatos. A compact terrain model based on critical topographic features. In *Proc. Auto-Carto 9*, pages 146–155, 1989.

[95] Lori Scarlatos and Theo Pavlidis. Adaptive hierarchical triangulation. In *Proc. 10th Internat. Sympos. Comput.-Assist. Cartog. (Auto-Carto)*, volume 6 of *Technical Papers 1991 ACSM-ASPRS Annual Convention*, pages 234–246, 1991.

[96] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[97] M. Sharir. The shortest watchtower and related problems for polyhedral terrains. *Inform. Process. Lett.*, 29:265–270, 1988.

[98] R. Sibson. A brief description of natural neighbour interpolation. In Vic Barnet, editor, *Interpreting Multivariate Data*, pages 21–36. Wiley, Chichester, 1981.

[99] A.T. Silfer, G.J. Kinn, and J.M. Hassett. A geographic information system utilizing the triangulated irregular network as a basis for hydrologic modeling. In *Auto-Carto 8*, pages 129–136, 1987.

[100] C. Silva, J. S. B. Mitchell, and A. E. Kaufman. Automatic generation of triangular irregular networks using greedy cuts. In *Visualization 95*, pages 201–208, San Jose CA, 1995. IEEE Computer Society Press.

[101] P. Sorensen and D. Lanter. Two algorithms for determining partial visibility and reducing data structure induced error in viewshed analysis. *Photogrammatic Engineering and Remote Sensing*, 28:1129–1132, 1993.

[102] J.G. Speight. Parametric description of landform. In G.A. Stewart, editor, *Land Evaluation papers of a CSIRO Symposium*, pages 239–250, 1968.

[103] J. Star and J. Estes. *Geographic Information Systems: an Introduction.* Prentice Hall, Englewood Cliffs, 1990.

[104] Y. A. Teng and L. S. Davies. Visibility analysis on digital terrain models and its parallel implementation. Technical Report CAR-TR-625, Center for Automation Research, University of Maryland, 1992.

[105] D.M. Theobald and M.F. Goodchild. Artifacts of TIN-based surface flow modelling. In *Proc. GIS/LIS*, pages 955–964, 1990.

[106] R. van Appelen. Drainage networks on TINs. Master's thesis, Department of Computer Science, Utrecht University, 1996.

[107] J. van Bemmelen, W. Quak, M. van Hekken, and P. van Oosterom. Vector vs. raster-based algorithms for cross country movement planning. In *Proc. Auto-Carto 11*, pages 304–317, 1993.

[108] M. van Kreveld. Efficient methods for isoline extraction from a TIN. *Int. J. of GIS*, 10:523–540, 1996.

[109] M. van Kreveld. Variations on sweep algorithms: efficient computation of extended viewsheds and classifications. In *Proc. 7th Int. Symp. on Spatial Data Handling*, pages 13A.15–13A.27, 1996.

[110] A. Voigtmann, L. Becker, and K. Hinrichs. Hierarchical surface representations using constrained Delaunay triangulations. In *Proc. 6th Int. Symp. on Spatial Data Handling*, pages 848–867, 1994.

[111] W. Warntz. The topology of a socio-economic terrain and spatial flow. *Papers of the Regional Science Association*, 17:47–61, 1966.

[112] R. Webster and M. A. Oliver. *Statistical Methods in Soil and Land Resource Survey*. Oxford University Press, New York, 1990.

[113] R. Weibel. An adaptive methodology for automated relief generalization. In *Proc Auto-Carto 8*, pages 42–49, 1987.

[114] R. Weibel and M. Heller. Digital terrain modelling. In D. J. Maguire, M. F. Goodchild, and D. W. Rhind, editors, *Geographical Information Systems – Principles and Applications*, pages 269–297. Longman, London, 1991.

[115] D. Wilcox and H. Moellering. Pass location to facilitate the direct extraction of Warntz networks from grid digital elevation models. In *Proc. Auto-Carto 12*, pages 22–31, 1995.

[116] G.W. Wolf. Metric surface networks. In *Proc. 4th Int. Symp. on Spatial Data Handling*, pages 844–856, 1990.

[117] M.F. Worboys. *GIS: A Computing Perspective*. Taylor & Francis, London, 1995.

[118] S. Yu, M. van Kreveld, and J. Snoeyink. Drainage queries in TINs: from local to global and back again. In *Proc. 7th Int. Symp. on Spatial Data Handling*, pages 13A.1–13A.14, 1996.

[119] B. Zhu. Improved algorithms for computing the shortest watchtower of polyhedral terrains. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 286–291, 1992.